REPUBLIQUE TUNISIENNE
MINISTERE DE L'EDUCATION ET DE LA FORMATION

ALGORITHMIQUE ET PROGRAMMATION

4ème année de l'enseignement secondaire
Sciences de l'informatique

Les auteurs

Abdelhafidh ABIDI
Inspecteur Principal

Nadia AGREBI DEKHIL
Professeur Principal d'Enseignement Secondaire

Noureddine ZOUARI
Professeur Principal Hors Classe

Les évaluateurs

Habib SMEI
Maître Technologue

Rached DOUARI
Inspecteur Principal

Centre National Pédagogique

Préface

La matière « Algorithmique et programmation » repose essentiellement sur l'algorithmique et la résolution de problèmes. La maîtrise de l'algorithmique requiert deux qualités :

- avoir une certaine intuition, car il est impossible de savoir a priori quelles instructions permettront d'obtenir le résultat voulu. C'est là, si l'on y tient, qu'intervient la forme « d'intelligence » requise pour l'algorithmique. Cette qualité se développe avec l'expérience et la multiplication des problèmes à résoudre. Le raisonnement, au départ laborieux, finira par devenir « spontané ».
- être méthodique et rigoureux. En effet, chaque fois qu'on résout un problème, il faut systématiquement se mettre mentalement à la place de la machine qui va exécuter le programme de la solution.

Ce manuel est conçu pour aider à développer chez les élèves, entre autres, les qualités d'intuition et de rigueur dans leurs raisonnements.

Conforme aux nouveaux programmes, ce manuel est destiné aux élèves de la quatrième année secondaire de la section « Sciences de l'informatique ». L'intention de l'ouvrage est donc, d'aider l'élève à :

- apprendre à résoudre des problèmes et à écrire des algorithmes.
- acquérir les algorithmes les plus courants dans des domaines variés tel que la récurrence, l'arithmétique, l'approximation, les tris, …
- Pascal.

Chacun des sept chapitres de ce manuel est précédé par :

1- La liste des objectifs qui précisent les savoirs et les savoir-faire permettant ainsi de délimiter la portée de chaque notion étudiée.
2- Le plan du chapitre.

Comme pour le manuel d'algorithmique et de programmation de la troisième année de la section « Sciences de l'informatique », chaque chapitre de ce manuel comporte :

- des activités préliminaires
- l'étude de la notion (définition, syntaxe au niveau algorithmique et au niveau du langage de programmation Pascal)
- des applications sous forme d'exercices résolus
- une série d'exercices en fin de chapitre

- une partie lecture pour renforcer le volet culture informatique chez les apprenants.

Le premier chapitre intitulé « Les enregistrements et les fichiers » est conçu pour introduire deux nouvelles structures de données que les élèves n'ont pas vu en 3ème année.

Le deuxième chapitre présente la technique de raisonnement par récurrence qui sera utilisée dans beaucoup d'applications des chapitres suivants.

Comme dans le programme de 3ème année, les cinq derniers chapitres présentent les grandes familles d'algorithmes à savoir :

- les algorithmes de tri
- les algorithmes récurrents
- les algorithmes arithmétiques
- les algorithmes d'approximation
- les algorithmes avancés

Nous invitons les utilisateurs de ce manuel à nous faire part de leurs critiques et de leurs suggestions et nous les remercions d'avance.

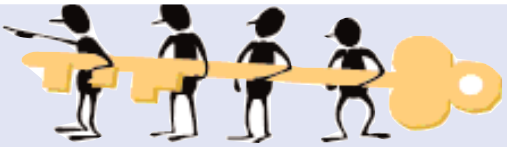
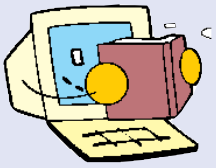
Les auteurs

Nadia.Dekhil@edunet.tn

Noureddine.Zouari@edunet.tn

Abdelhafidh.Abidi@edunet.tn

Les logos

	**Une remarque importante**
	**Une idée**
	**- Une solution d'une application ou d'une activité posée,** **- Une réponse à une ou à plusieurs questions**
	**Une série d'exercices non corrigés**
	**Lecture**
	**Des points importants à retenir du chapitre**
	**Question, activité, application ou exercice de degré de difficulté simple**
	**Question, activité, application ou exercice de degré de difficulté moyen**
	**Question, activité, application ou exercice de degré de difficulté assez élevé**
	**Question, activité, application ou exercice de degré de difficulté élevé**

Sommaire

Chapitre 1

Les enregistrements et les fichiers

Objectifs

- Définir la structure enregistrement
- Définir les fichiers et les modes d'accès
- Mettre à profit les structures enregistrement et fichiers pour résoudre des problèmes.

Plan du chapitre

Chapitre 1

Les enregistrements et les fichiers

A. Les enregistrements

I. Introduction

Activité 1

Un établissement scolaire organise les informations concernant ses classes dans une liste identique à la suivante :

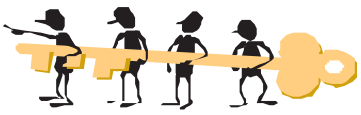
N°	Code	Nom & Prénom	Moyenne	Observations
1	C0120	Cherni Selim	14.25	Néant
2	K0235	Kefi Marwa	13.12	Redoublante
…	…	…	…	….
…	…	…	…	…
30	B3017	Bennour Raouf	11.75	Dispensé du sport

Question :

Le directeur de l'établissement veut créer un programme permettant la saisie et le traitement de ces listes sachant que chaque classe comporte au maximum 40 élèves.

a. Donnez la structure de données nécessaire pour les objets à utiliser.

b. Donnez une déclaration algorithmique de ces objets.



a. Nous remarquons que cette liste comporte une information **alphanumérique** (Code), des informations **numériques** (N°, Moyenne) et d'autres **alphabétiques** (Nom & Prénom, Observations).

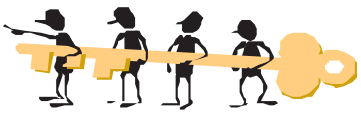
D'après les connaissances que nous avons acquises les deux dernières années, nous pouvons utiliser 5 variables déclarées comme suit :

b. Tableau de déclaration des objets :

Objet	Type / Nature	Rôle
Num	Tableau de 40 entiers	Tableau des numéros des élèves
Code	Tableau de 40 chaînes	Tableau des codes
Nom	Tableau de 40 chaînes	Tableau contenant les noms & prénoms
Moy	Tableau de 40 réels	Tableau des moyennes
Obser	Tableau de 40 chaînes	Tableau des observations

Question :

Est-il possible de regrouper ces variables au sein d'un même tableau ?



Bien sûr que **NON** car un tableau ne peut contenir que des éléments de même type. Mais nous pouvons utiliser 5 tableaux différents déclarés comme suit :

Tableau de déclaration des nouveaux types :

Type
Tab = Tableau de 40 Chaîne de caractères

Tableau de déclaration des objets :

Objet	Type / Nature	Rôle
Num	Tableau de 40 entiers	Tableau contenant les numéros des élèves
Code	Tab	Tableau contenant les codes
Nom	Tab	Tableau contenant les noms & prénoms
Moy	Tableau de 40 réels	Tableau regroupant les moyennes
Obser	Tab	Tableau rangeant les observations

Nous venons de voir que les variables simples ou les tableaux ne permettent pas de ranger des données de types différents.
Si nous voulons établir par exemple une structure comportant en même temps des informations alphanumériques, numériques et alphabétiques, nous devons créer un nouveau **TYPE** qui permet de les regrouper.
Nous allons voir une nouvelle structure appelée **ENREGISTREMENT** ou **ARTICLE** (**RECORD** en Pascal) qui permet de réaliser cette tâche.

II. Définition et déclaration

Définition

Un enregistrement est un type de données défini par l'utilisateur et qui permet de grouper un nombre fini d'éléments (ou champs) de types éventuellement différents.

Schématisons cette structure :

Champ 1 Type 1	Champ 2 Type 2	Champ 3 Type 1	Champ 4 Type 4	Champ 5 Type 5

Une seule entité d'une variable enregistrement

Déclaration d'une structure ENREGISTREMENT

En algorithmique

Tableau de déclaration des nouveaux types

Type
Nom_type = Enregistrement champ 1 : Type 1 - - - - champ n : Type n Fin Nom_Type

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
Identificateur_objet	Nom_type	Enregistrement pour …

En Pascal

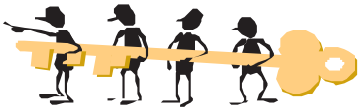
```
TYPE   Nom_type = Record
                    champ_1 : type_1 ;
                     -   -          - -
                    champ_n : type_n ;
       End;

VAR
      identificateur_objet : Nom_type ;
```

a. Les types (type 1, type 2, .. , type n) peuvent être soit prédéfinis, soit définis par l'utilisateur.

b. Dans ce qui suit, nous utiliserons les mots variable ou objet au lieu de identificateur_objet.

Activité 2

Déclarez (en algorithmique et en Pascal) une variable enregistrement représentant un nombre complexe composé d'une partie réelle et d'une partie imaginaire.



En algorithmique

Tableau de déclaration des nouveaux types

Type
complexe = Enregistrement p_réelle : Réel p_imaginaire : Réel Fin complexe

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
z	complexe	Variable de type complexe, représentant un nombre complexe

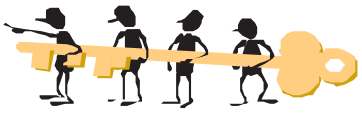
En Pascal

```
TYPE   complexe   = Record
                    p_reelle      : Real ;
                    p_imaginaire       : Real ;
       End ;

VAR
       z : complexe ;
```

Activité 3

Déclarez en algorithmique et en Pascal, une varaiable enregistrement qui comporte :
– le numéro du jour (jj) de 1 à 31,
– le mois (mm) en utilisant le type **mois** qui est un nouveau type défini par l'utilisateur et qui énumère les 12 mois de l'année,
– l'an (aa) qui est un entier.

Déclarez une variable nommée "calendrier" qui permettra l'utilisation de cet enregistrement.



En algorithmique

Tableau de déclaration des nouveaux types

Type
mois = (Janvier, Février, Mars, Avril, Mai, Juin, Juillet, Août, Septembre, Octobre, Novembre, Décembre) date = Enregistrement jj : 1..31 mm : mois aa : Entier Fin date

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
calendrier	date	Variable de type date représentant une date

En Pascal

```
Type mois = (Janvier, Fevrier, Mars, Avril, Mai, Juin, Juillet,
Aout, Septembre, Octobre, Novembre, Decembre );

date = Record
       jj : 1 .. 31 ;
       mm : mois ;
       aa    : Integer ;
End;

VAR
       calendrier : date ;
```

III. Utilisation des enregistrements

III.1 Affectation

L'affectation de valeurs aux différents champs d'une variable enregistrement se fait comme suit :

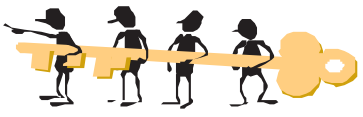
En algorithmique	En Pascal
variable.champ ← *valeur*	*variable.champ := valeur ;*

Remarquez le point entre variable et champ.

Activité 4

a. Déclarez une variable enregistrement pour représenter la fiche d'un étudiant sachant qu'elle contient les informations suivantes : Nom, Prénom, sexe (F ou G), date de naissance et la moyenne au baccalauréat.

b. Affectez respectivement les valeurs suivantes à cette variable : "Kéfi", "Nour", "F", "27/11/1983" et 13.25



En algorithmique :

a. Déclaration d'une variable enregistrement.

Tableau de déclaration des nouveaux types

Type
Fiche = enregistrement nom, prénom : Chaîne sexe : Caractère date_nais : Chaîne moy : Réel Fin Fiche

Tableau de déclaration des nouveaux types

Objet	Type / Nature	Rôle
étudiant	Fiche	Variable de type Fiche, représentant une fiche d'un étudiant

b. Affectation des valeurs à cette variable :

étudiant.nom ← "Kéfi"
étudiant.prénom ← "Nour"
étudiant.sexe ← "F"
étudiant.date_nais ← "27/11/1983"
étudiant.moy ← 13.25

En Pascal :

a. Déclaration d'une variable enregistrement.

```
TYPE Fiche  = Record
             nom, prenom : String ;
             sexe : Char ;
             Date_nais : String ;
             moy : Real ;
      End ;

VAR
     etudiant : Fiche ;
```

b. Affectation des valeurs à cette variable :

```
etudiant.nom   := 'Kéfi' ;
etudiant.prenom  := 'Nour' ;
etudiant.sexe := 'F' ;
etudiant.date_nais :=  '27/11/1983' ;
etudiant.moy  := 13.25 ;
```

a. Il est possible d'affecter une variable enregistrement dans une autre à condition qu'ils aient la même structure.

Exemple :

```
VAR   e1, e2 : Fiche ;
```

Il est possible d'écrire :

e1 := e2 ; (ou bien *e2 := e1;*)

Tous les champs de la variable enregistrement à affecter seront recopiés dans les champs de l'autre.

b. Un champ a exactement les mêmes propriétés qu'une variable du même type.

c. Le champ d'une variable enregistrement peut être lui-même un enregistrement.

Exemple :

Reprenez l'activité précédente et déclarez le champ date_nais comme étant un enregistrement composé par les champs (jour, mois, année).

Le tableau de déclaration de nouveaux types sera :

Tableau de déclaration des nouveaux types

Type
Date = Enregistrement jour : Entier mois : Chaîne année : Entier Fin Date Fiche = Enregistrement nom, prénom : Chaîne sexe : Caractère date_nais : Date moy : Réel Fin Fiche

Déclaration de la variable étudiant :

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
étudiant	Fiche	Variable de type Fiche, représentant une fiche d'un étudiant

III.2 Lecture

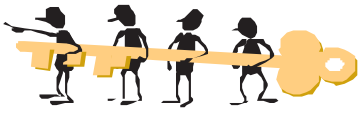
La lecture des valeurs des différents champs d'une variable enregistrement se fait comme suit :

Au niveau de l'analyse	Au niveau de l'algorithme	Au niveau du Pascal
variable.champ = Donnée	Lire (variable.champ)	*ReadLn (variable.champ);*

Remarquez toujours le point entre la variable et le champ.

Activité 5

Reprenez l'activité 4 et écrivez les instructions permettant de saisir à partir du clavier les champs de la variable enregistrement **Etudiant.**



Au niveau de l'analyse :

étudiant.nom = Donnée ("Entrer le nom de l'étudiant : ")
étudiant.prénom = Donnée ("Entrer le prénom de l'étudiant : ")
étudiant.sexe = Donnée ("Entrer le sexe de l'étudiant : ")
étudiant.date_nais = Donnée ("Entrer la date de naissance de l'étudiant : ")
étudiant.moy = Donnée ("Entrer la moyenne de l'étudiant : ")

Au niveau de l'algorithme :

Ecrire ("Entrer le nom de l'étudiant : ") ; Lire (étudiant.nom)
Ecrire ("Entrer le prénom de l'étudiant : ") ; Lire (étudiant.prénom)
Ecrire ("Entrer le sexe de l'étudiant : ") ; Lire (étudiant.sexe)
Ecrire ("Entrer la date de naissance de l'étudiant : ") ; Lire (étudiant.date_nais)
Ecrire ("Entrer la moyenne de l'étudiant : ") ; Lire (étudiant.moy)

Au niveau du Pascal :

```
Write ('Entrer le nom de l''étudiant : ') ;
ReadLn (etudiant.nom) ;
Write ('Entrer le prénom de l''étudiant : ') ;
ReadLn (etudiant.prenom) ;
Write ('Entrer le sexe de l''étudiant : ') ;
ReadLn (etudiant.sexe) ;
Write ('Entrer la date de naissance de l''étudiant : ') ;
ReadLn (etudiant.date_nais) ;
Write ('Entrer la moyenne de l''étudiant : ') ;
ReadLn (etudiant.moy) ;
```

III.3 Ecriture

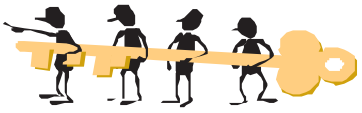
L'écriture des valeurs des différents champs d'une variable enregistrement se fait comme suit :

Au niveau de l'analyse et de l'algorithme	Au niveau du Pascal
Ecrire (variable.champ)	*Write (variable.champ) ;*

Remarquez toujours le point entre la variable et champ.

Activité 6

Reprenez l'activité 4 et écrivez les instructions permettant d'afficher les champs de la variable enregistrement **Etudiant.**



Au niveau de l'analyse et de l'algorithme :

Ecrire ("Nom : ", étudiant.nom)
Ecrire ("Prénom : ", étudiant.prénom)
Ecrire ("Sexe : ", étudiant.sexe)
Ecrire ("Date de naissance : ", étudiant.date_nais)
Ecrire ("Moyenne : ", étudiant.moy)

Au niveau du Pascal :

```
WriteLn ('Nom : ', etudiant.nom) ;
WriteLn ('Prénom : ', etudiant.prenom) ;
WriteLn ('Sexe : ', etudiant.sexe) ;
WriteLn ('Date de naissance : ', etudiant.date_nais) ;
WriteLn ('Moyenne : ', etudiant.moy) ;
```

III.4 Structure Avec .. Faire

Pour simplifier l'écriture et éviter l'utilisation répétée des champs et de la notation avec le point (variable.champ), nous pouvons utiliser l'instruction **Avec .. Faire (With .. Do).**

NB : Cette structure s'utilise avec une opération d'affectation, de lecture ou d'écriture.

Syntaxe

Au niveau de l'analyse et de l'algorithme	Au niveau du Pascal

Avec variable **Faire**
{ensemble d'actions}
Fin Avec
	`With variable Do`
	`Begin`
	`{ensemble d'actions}`
	`End;`

Activité 7

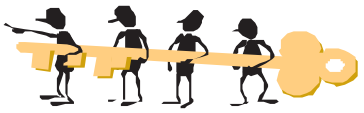
Rappelons que l'activité 4 utilise un enregistrement de nom **Fiche** et une variable **étudiant**

Question : Reprenez les actions suivantes extraites de cette activité et écrivez-les avec la structure **Avec..Faire**

Actions :

{Affectation}
étudiant.nom ← "Kéfi"
étudiant.prénom ← "Nour"
{Lecture}
Ecrire ("Entrer le sexe de l'étudiant : ") ; Lire (étudiant.sexe)
Ecrire ("Entrer la date de naissance de l'étudiant :") ; Lire (étudiant.date_nais)

{Ecriture}
Ecrire ("Moyenne : ", étudiant.moy)



Au Niveau de l'algorithme :

Avec étudiant **Faire**

{Affectation}
nom ← "Kéfi"
prénom ← "Nour"

{Lecture}
Ecrire ("Entrer le sexe de l'étudiant : ") ; Lire (sexe)
Ecrire ("Entrer la date de naissance de l'étudiant :") ; Lire (date_nais)

{Ecriture}
Ecrire ("Moyenne : ", moy)

Fin Avec

Au Niveau du Pascal :

```
With etudiant Do
Begin
      nom := 'Kéfi' ;
      prenom :=  'Nour' ;

      Write ('Entrer le sexe de l''étudiant : ') ; ReadLn (sexe) ;
      Write ('Entrer la date de naissance de l''étudiant :') ; ReadLn
(date_nais)  ;

      WriteLn ('Moyenne : ', moy) ;
End;
```

Activité 8

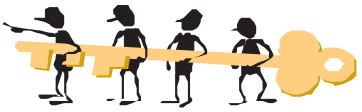
Soit la structure **Personne** constituée par :
- un nom (chaîne de 30 caractères maximum)
- un numéro fiscal (entier)
- un numéro de téléphone (10 caractères maximum)
- un numéro de carte bancaire (entier non signé).

1- Ecrivez les analyses, les algorithmes des différents modules d'un programme nommé Fiche, qui permet la saisie et l'affichage de l'enregistrement d'une personne.

2- Traduisez ce programme en Pascal et l'enregistrez sous le nom **Enreg_1**

Chapitre

1



1- Analyses et algorithmes :

Analyse du programme principal

Résultat : Affichage d'une fiche
Traitement : – Une fiche peut être représentée par une structure enregistrement comportant 4 champs (le nom, le numéro fiscal, le numéro de téléphone et le numéro de carte bancaire).
– L'affichage des différents champs sera la tâche de la procédure Afficher.
– La saisie des différents champs se fera par la procédure Saisir.

Algorithme du programme principal :

0) Début Fiche
1) Proc Saisir (individu)
2) Proc Afficher (individu)
3) Fin Fiche

Codification des nouveaux types

Type
Personne = Enregistrement nom : Chaîne de 30 caractères fisc : Entier tel : Chaîne de 10 caractères banc : Entier non signé Fin Personne

Codification des objets globaux

Nom	Type / Nature	Rôle
individu	Personne	Enregistrement pour une fiche personne
Saisir	Procédure	Saisie des champs
Afficher	Procédure	Affichage des champs

Analyse de la procédure Saisir

Résultat : Saisir les champs
Traitement : La saisie des différents champs de l'enregistrement se fera par des opérations de lecture avec l'utilisation de la structure Avec .. Faire, sur la variable individu.

Algorithme de la procédure Saisir

```
0) Début procédure Saisir (Var individu : Personne )
1) Avec individu Faire
        Ecrire ("Entrer le nom de la personne : ") ; Lire (nom)
        Ecrire ("Entrer son code fiscal : ") ; Lire (fisc)
        Ecrire ("Entrer son numéro de téléphone : ") ; Lire (tel)
        Ecrire ("Entrer le numéro de sa carte bancaire : ") ; Lire (banc)
   Fin Avec
2) Fin Saisir
```

Analyse de la procédure Afficher

Résultat : Afficher les champs
Traitement : L'affichage des différents champs de l'enregistrement se fera par des opérations d'écriture avec l'utilisation de la structure **Avec .. Faire**, sur la variable individu.

Algorithme de la procédure Afficher

```
0) Début procédure Afficher (individu : Personne )
1) Avec individu Faire
        Ecrire ("Nom : ", nom)
        Ecrire ("Code fiscal : ", fisc)
        Ecrire ("Numéro de téléphone : ", tel)
        Ecrire ("Numéro de sa carte bancaire : ", banc)
   Fin Avec
2) Fin Afficher
```

2- Traduction en Pascal

Selon la version du Pascal installée sur les machines, nous déclarons l'utilisation de l'unité de gestion de l'écran par la clause :

Uses Crt ; (Si la version est Free Pascal, Turbo Pascal 7 ou toute autre version sous DOS)

Uses WinCrt ; (Si la version est TPW 1.5 ou toute autre version sous Windows).

Dans la suite, nous allons utiliser la clause ***Uses*** Crt ; pour une version Free Pascal.

```
PROGRAM Fiche ;
USES Crt ;
TYPE Personne = Record
            nom : String [30] ;
            fisc : Integer ;
            tel : String [10] ;
            banc : Word ;
      End ;

VAR  individu : Personne ;

PROCEDURE Saisir (VAR individu : Personne ) ;
Begin
     With individu Do
     Begin
       Write ('Entrer le nom de la personne : ') ; ReadLn (nom) ;
       Write ('Entrer son code fiscal : ') ; ReadLn (fisc) ;
       Write ('Entrer son numéro de téléphone : ') ; ReadLn (tel) ;
       Write ('Entrer le numéro de sa carte bancaire : ') ; ReadLn
(banc) ;
     End ;
End ;

PROCEDURE Afficher (individu : Personne ) ;
Begin
     With individu Do
     Begin
       WriteLn ('Nom : ', nom) ;
       WriteLn ('Code fiscal : ', fisc) ;
       WriteLn ('Numéro du téléphone : ', tel) ;
       WriteLn  ('Numéro de la carte bancaire : ', banc) ;
     End;
End;

{Programme Principal }
BEGIN
      Saisir (individu) ;
      Afficher (individu);
END.
```

III.5 Vecteur d'enregistrements

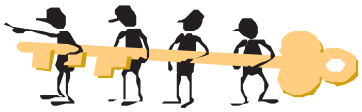
Activité 9

Reprenons l'enregistrement **Fiche** de l'activité 4

```
Fiche  = Enregistrement
          nom, prénom : Chaîne
          sexe : Caractère
          date_nais : Chaîne
          moy : Réel
Fin Fiche
```

Nous voulons utiliser cet enregistrement non pas pour un seul étudiant, mais pour tous les étudiants d'une classe.

Question : Pouvons-nous déclarer un tableau d'enregistrements ?



Un tableau ne peut contenir que des éléments de même type, y compris le type enregistrement. Nous pouvons donc utiliser un tableau ou un vecteur d'enregistrements.

Activité 10

Nous supposons que le nombre d'étudiants dans une classe est égal à N ($4 \leq N \leq 30$).

Question : Proposez une structure de données utilisant un vecteur d'enregistrements pour représenter ces N étudiants ?

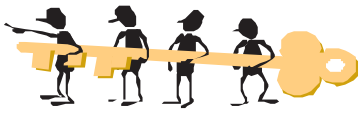


Tableau de déclaration des nouveaux types

Type
Date = Enregistrement jour : Entier mois : Chaîne année : Entier Fin Date

```
Fiches  = Enregistrement
                nom, prénom : Chaîne
                sexe : Caractère
                Date_nais : Date
                moy : Réel
Fin Fiches

Tab = Tableau de 30 Fiches
```

Déclaration de la variable étudiant :

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
T	Tab	Tableau de type Tab, représentant les fiches des étudiants

Activité 11

Une société veut informatiser la gestion de ses employés. Elle détient pour chacun les informations suivantes :

- Le nom et le prénom (chaîne de caractères)
- Le grade : uniquement G1, G2, G3 ou G4
- Le code fiscal (un entier non signé)
- L'assurance maladie (O pour oui et N pour non)

Le nombre d'employés est N avec $4 \leq N \leq 120$.

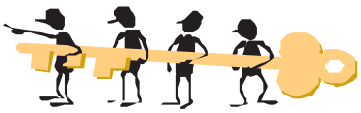
Questions :

– Ecrivez un programme modulaire nommé **GESTION**, qui permet la saisie de toutes les fiches de renseignements puis d'afficher :

1- Toutes les fiches (une par une et avec une attente jusqu'à l'appui sur la touche ENTREE).

2- Le nombre d'employés ayant un grade donné et leur pourcentage par rapport au nombre total des employés.

– Traduisez ce programme en Pascal et l'enregistrez sous le nom **Enreg_2**.



1- Analyses et algorithmes :

Analyse du programme principal : Gestion

Résultat : Affichage des fiches et du résultat de recherche des grades et leurs pourcentage.

Traitement :

– Une fiche d'un employé peut être représentée par une structure d'enregistrement.

– L'affichage du nombre d'occurrences d'un grade donné et son pourcentage est fait par la procédure Result.

– La recherche du nombre d'employés qui ont ce grade est faite par la fonction Cherche.

– La lecture et le test du grade sont les tâches de la procédure Lecture. Cette procédure sera appelée aussi par la procédure Saisie puisque dans les deux cas nous avons le même traitement (saisie du grade avec test de validité).

– L'affichage de toutes les fiches, une par une, se fera par la procédure Affichage.

– Les N fiches peuvent être rangées **dans un tableau d'enregistrements**. C'est le rôle de la procédure Saisie.

Algorithme du programme principal :

0) Début Gestion
1) Proc Saisie (N, T)
2) Proc Affichage (N, T)
3) Proc Lecture (G)
4) Nb_G ← FN Cherche (N, T, G)
5) Proc Result (N, Nb_G, P, G)
3) Fin Gestion

Codification des nouveaux types

Type
Fiche = Enregistrement nom : Chaîne de 40 caractères grade : Chaîne de 2 caractères code : Entier non signé am : Caractère Fin Fiche
Tab = Tableau de 120 fiches
Gr = Chaîne de 2 caractères

Codification des objets globaux

Nom	Type / Nature	Rôle
T	Tab	Tableau de type Tab
N	Entier	Nombre d'employés
G	Gr	Grade à rechercher
Nb_G	Entier	Nombre de grades trouvés
P	Réel	Pourcentage du grade trouvé
Saisie	Procédure	Saisir N et remplir le tableau d'enregistrements
Affichage	Procédure	Afficher les fiches
Lecture	Procédure	Lire le grade de l'employé
Cherche	Fonction	Chercher le nombre d'employés qui ont un grade donné
Result	Procédure	Afficher le nombre d'occurrences d'un grade et son pourcentage.

Analyse de la procédure Saisie

Résultat : Saisir le nombre d'employés et remplir le tableau T par les N enregistrements (fiches).

Traitement : La saisie du nombre d'employés N se fait avec un test de validité (**Répéter .. jusqu'à**).

- Pour chaque fiche, la saisie de ses différents champs et leurs affectations à un élément du tableau T nécessite pour une raison de simplification, l'utilisation de la structure **Avec .. Faire**. La saisie et le test de validité du champ grade se feront par l'appel de la procédure Lecture :

```
Pour i de 1 à N Faire
  Avec T[i] Faire
        Nom = Donnée ("Entrer le nom et le prénom de l'employé : ")
        Proc Lecture (grade)
        Code = Donnée ("Entrer son code fiscal : ")
        Répéter
            am = Donnée ("Assuré (Oui / Non ) ? : ")
        Jusqu'à Majuscule (am) Dans ["O", "N"]
  Fin Avec
```

Algorithme de la procédure Saisie

```
0) Début procédure Saisie (Var N : Entier ; Var T : Tab )
1) Répéter
        Ecrire ("Entrer le nombre d'employés : ") ; Lire (N)
    Jusqu'à  N Dans [4 .. 120]
2) Pour i de 1 à N Faire
        Avec T[i] Faire
                Ecrire ("Entrer le nom et le prénom de l'employé : ") ; Lire (nom)
                Proc Lecture (grade)
                Ecrire ("Entrer son code fiscal : ") ; Lire (code)

                Répéter
                        Ecrire ("Assuré (Oui / Non ) ? : ")
                        Lire (am)
                Jusqu'à Majuscule (am) Dans ["O", "N"]
        Fin Avec
    Fin Pour
3) Fin Saisie
```

Analyse de la procédure Affichage

Résultat : Afficher les fiches une à une
Traitement : - L'affichage des différents champs composants une fiche se fera par des opérations d'écriture avec l'utilisation de la structure **Avec .. Faire**, sur la variable T.
- Pour afficher toutes les fiches, une itération complète (Pour .. faire) sera utilisée :

```
    Pour  i de 1 à n Faire
        Avec T[i] Faire
                Ecrire ("Nom & prénom      : ", nom)
                Ecrire ("Grade             : ", grade)
                Ecrire ("Code fiscal       : ", code)
                Ecrire ("Assurance maladie : ", am)
        Fin Avec
```

Algorithme de la procédure Affichage

```
0) Début procédure Affichage (N : Entier ; T : Tab )
1) Pour i de 1 à n Faire
        Avec T[i] Faire
                Ecrire ("Nom & prénom     : ", nom)
                Ecrire ("Grade            : ", grade)
                Ecrire ("Code fiscal      : ", code)
                Ecrire ("Assurance maladie :", am)
        Fin Avec
   Fin Pour
2) Fin Affichage.
```

Analyse de la procédure Lecture

Résultat : Saisir le grade d'un employé

Traitement : La saisie du grade d'un employé se fait avec un test de validité (Répéter .. jusqu'à) car un grade doit être G1, G2, G3 ou G4.

Algorithme de la procédure Lecture

```
0) Début procédure Lecture (Var G : Gr )
1) Répéter
        Ecrire ("Entrer le grade : ") ; Lire (G)
   Jusqu'à  (G="G1") ou (G="G2") ou (G="G3") ou (G="G4")
2) Fin Lecture
```

Analyse de la fonction Cherche

Résultat : Renvoyer le nombre d'occurrences d'un grade G.

Traitement : - Le calcul du nombre d'occurrences (nb) d'un grade G se fera par un parcours total du tableau et la comparaison de chaque champ Grade d'une fiche avec la valeur G à chercher :

```
nb ← 0
Pour i de 1 à N Faire
    Si (T[i].Grade = G) Alors nb ← nb + 1
```

Algorithme de la fonction Cherche

```
0) Début fonction Cherche (N : entier ; T : Tab ; G : Gr) : Entier
1) nb ← 0
   Pour i de 1 à N Faire
        Si (T[i].grade = G) Alors nb ← nb + 1
        Fin Si
   Fin Pour
2) Cherche ← nb
3) Fin Cherche
```

Analyse de la procédure Result

Résultat : Afficher le nombre d'occurrences d'un grade G et son pourcentage P.
Traitement : - Le pourcentage P est égal à (Nb_G * 100) /N

Algorithme de la procédure Result

0) Début procédure Result (N, Nb_G : Entier ; P : Réel ; G : Gr)
1) Ecrire ("Le nombre d'occurrences de ', G, " Est égal à : ", Nb_G)
2) P ← (Nb_G * 100) / N
3) Ecrire ("Le pourcentage de ", G, " Est de : ", P, " %")
2) Fin Result

Traduction en Pascal

```
PROGRAM Gestion ;
USES Crt ;
TYPE
      Gr = String [2] ;

      Fiche = Record
            nom : String [40] ;
            grade : Gr ;
            code  : Word ;
            am : Char ;
      End ;

      Tab = Array [1..120] Of Fiche ;

VAR
      T : Tab ;
      N, Nb_G : Integer ;
      G : Gr ;
      P : Real ;
PROCEDURE Lecture (VAR G : Gr ) ;
Begin
      Repeat
            Write ('Entrer le grade : '); ReadLn (G) ;
      Until (G='G1') OR (G='G2') OR (G='G3') OR (G='G4') ;
End ;
```

```
PROCEDURE Saisie (VAR N : Integer ; VAR T : Tab ) ;
VAR i : Integer ;
Begin
      Repeat
            Write ('Entrer le nombre d''employés : ')  ; ReadLn (N) ;
      Until N IN [4 .. 120] ;

      For i:= 1 To N Do
      Begin
        With T[i] Do
        Begin
            Write ('Entrer le nom et le prénom de l''employé : ') ; ReadLn (nom) ;
            Lecture (grade) ;
            Write ('Entrer le son code fiscal : ') ; ReadLn (code) ;

            Repeat
                        Write ('Assuré (Oui / Non ) ? : ' );
                        ReadLn (am) ;
            Until  Upcase (am) IN ['O', 'N'] ;
            End ;
      End ;
End ;
PROCEDURE  Affichage (N : Integer ; T : Tab ) ;
VAR i : Integer ;
Begin
      For i:=1 To N Do
      Begin
            With T[i] Do
            Begin
                  WriteLn ('Nom & prénom      : ', nom) ;
                  WriteLn ('Grade                        : ', grade) ;
                  WriteLn ('Code fiscal               : ', code) ;
                  WriteLn ('Assurance maladie : ', am) ;
            End;
            ReadLn ;
            ClrScr ;
      End ;
End ;
FUNCTION Cherche (N : Integer ; T : Tab ; G : Gr) : Integer ;
VAR nb, i : Integer ;
Begin
      nb := 0 ;
      For i:= 1 To N Do
            If ( T[i].grade = G) Then nb := nb + 1 ;

      Cherche := nb ;
End ;
```

```
PROCEDURE Result (N, Nb_G : Integer ; P : Real ; G : Gr) ;
Begin
     WriteLn ('Le nombre d''occurrencse de ', G, ' Est égal à : ', Nb_G) ;
     P :=  (Nb_G * 100 ) / N ;
     WriteLn ('Le pourcentage de ', G, ' Est de : ', P:6:2, ' %') ;
End ;

{Programme Principal}
BEGIN
      Saisie (N, T) ;
      Affichage (N, T) ;
      Lecture (G) ;
      Nb_G :=  Cherche (N, T, G) ;
      Result (N, Nb_G, P, G) ;
END.
```

Chapitre

1

B. Les fichiers

I. Introduction

Les informations utilisées dans tous les programmes que nous avons déjà écris ne pouvaient provenir que de deux sources : soit elles étaient incluses dans le programme lui-même, soit elles étaient entrées ou saisies en cours de route par l'utilisateur.
Après la fin du programme, ces informations sont perdues. Si nous exécutons de nouveau le programme, il faut réintroduire les mêmes ou d'autres informations. D'autre part, toutes ces informations ont un point commun : elles résident toutes dans la mémoire principale de l'ordinateur. Ceci signifie que l'effacement (volontaire ou non!) de la mémoire provoque la destruction de ces informations, ainsi que celles utilisées dans le programme "Pascal".

Il est parfois nécessaire de conserver certaines données après la fin de l'exécution du programme, pour une sauvegarde d'archives ou en prévision d'une utilisation future.

Ces considérations nous amènent à introduire la notion de **fichier**.
Dans la vie courante, par exemple dans une bibliothèque, chez un médecin ou dans une administration, un fichier est une collection de fiches de même aspect et dont le nombre n'est pas limité.

En informatique un fichier est un ensemble cohérent d'informations (des données ou des enregistrements dont le nombre n'est pas connu à priori) enregistré sur un support de stockage.

Définition

Un fichier est un ensemble structuré de données de même type, nommé et enregistré sur un support lisible par l'ordinateur (disque dur, disquette, flash disque, CD Rom, ..etc). Un fichier peut contenir des caractères (fichier textes), des programmes, des valeurs (fichier de données).

II. Organisation des fichiers

Les blocs de données formant un fichier sont enregistrés les uns à la suite des autres, de façon linéaire, comme indiqué dans le schéma suivant :

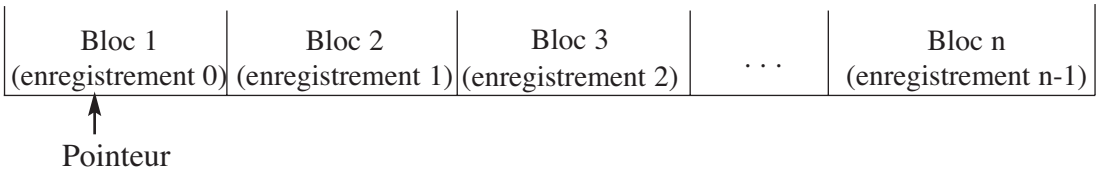
Bloc 1 (enregistrement 0)	Bloc 2 (enregistrement 1)	Bloc 3 (enregistrement 2)	. . .	Bloc n (enregistrement n-1)

Pour accéder (lire ou modifier) à un bloc (un enregistrement), nous devons tout d'abord le localiser, c'est-à-dire le pointer.

Le pointeur de fichier est un entier qui indique à partir de quelle position (ou adresse) du fichier la prochaine fonction d'entrée / sortie doit s'effectuer.

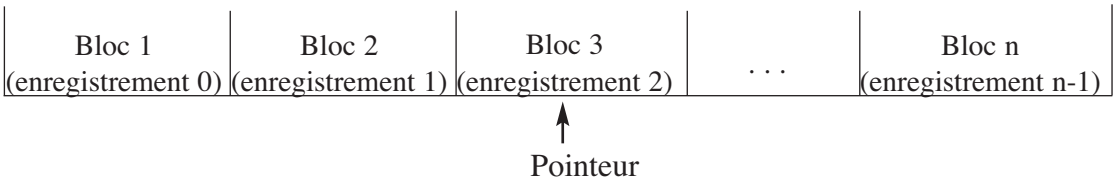
Exemple 1 :

Placez le pointeur du fichier sur le premier bloc c'est-à-dire l'enregistrement d'adresse 0. On dit aussi qu'on remet à zéro le pointeur du fichier (ReSet).



Exemple 2 :

Placez le pointeur du fichier sur le bloc 3 pour une modification par exemple.



Pointer signifie « faire référence à ».
La valeur d'un pointeur peut changer : cela ne signifie pas que l'objet ou le bloc pointé a été déplacé dans le fichier, mais plutôt que le pointeur pointe sur un autre bloc.

L'organisation caractérise les modes d'implémentation des informations ou des enregistrements dans le fichier et fournit les propriétés d'accès.
Il existe plusieurs types d'organisation des fichiers, nous allons voir essentiellement deux grands types d'organisations de fichiers caractérisées par une implémentation contiguë des enregistrements sur le support de stockage :

- **organisation séquentielle :** nous ne pouvons accéder aux enregistrements qu'en les parcourant les uns à la suite des autres.
- **organisation relative** (dite aussi directe) : les enregistrements sont contigus et identifiés par un numéro d'ordre.

Chaque organisation a ses particularités pour la création, les modifications ou les mises à jour d'un enregistrement. Par exemple, dans une organisation séquentielle, l'ajout ne peut se faire qu'en fin de fichier, les suppressions nécessitent en principe la recopie de tout le fichier, etc.).

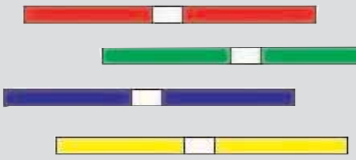
Toutes les organisations ne sont pas possibles sur tous les supports. Par exemple, sur une bande magnétique, seule une organisation séquentielle est possible.

III. Types d'accès

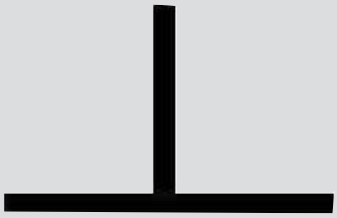
Activité 1

Nous disposons de 4 disques de couleurs différentes et un support où nous allons empiler un à un les disques.

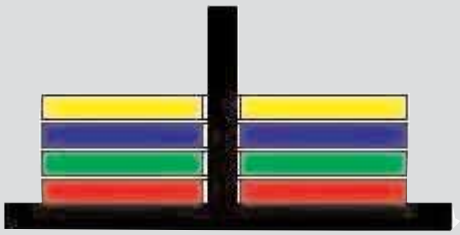
Nous obtenons :



Disques

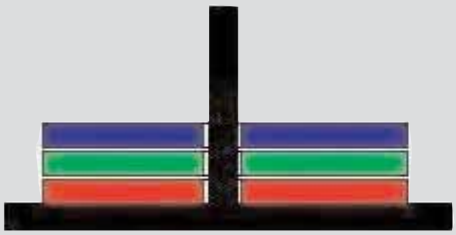


Support



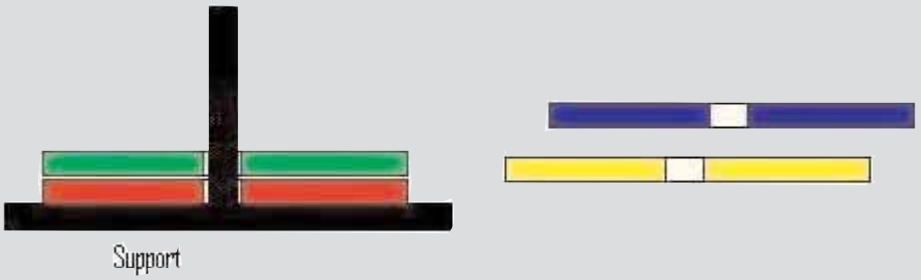
Support

Nous voulons accéder au disque vert. Nous ne pouvons pas le faire directement car il se trouve sous le disque bleu et ce dernier sous le disque jaune, donc il faut en premier lieu retirer le disque jaune. Nous aurons :



Support

Puis il faut retirer le disque bleu :



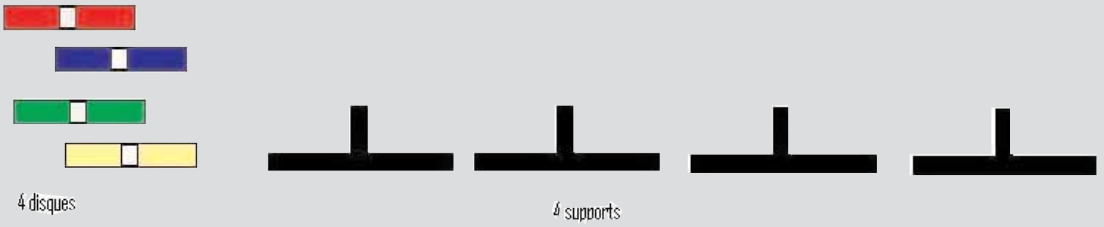
Maintenant nous pouvons accéder au disque vert.

Pour accéder à un disque particulier, il faut retirer tous les disques placés avant lui. Ce type d'accès est dit **accès séquentiel**.

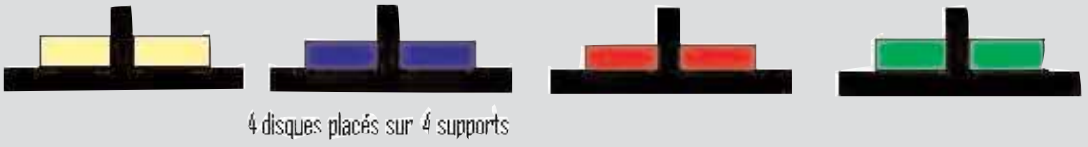
Dans ce cas le premier disque placé est le dernier retiré. Nous disons que "le premier entrant est le dernier sortant".

Activité 2

Nous disposons maintenant de 4 disques de couleurs différentes et 4 de supports distincts..



Nous allons placer chaque disque sur un support.



Nous voulons accéder au disque rouge. Pour cela il n'est pas nécessaire de retirer le disque jaune puis le disque bleu. Nous accèdons directement au disque rouge.

Accès au disque rouge

Pour accéder à un disque particulier, il suffit d'aller directement à ce disque et de le retirer. Ce type d'accès est dit **accès direct** (dit aussi accès aléatoire).

Activité 3

Pour mieux comprendre la différence entre les deux types d'accès, faisons l'analogie avec un magnétophone et un CD Audio. Pour pouvoir écouter la quatrième chanson de la cassette à bande magnétique, il faut écouter les trois chansons qui sont placées avant. Le magnétophone fournit un accès séquentiel aux chansons situées sur la bande magnétique. En revanche, pour écouter la quatrième chanson du CD Audio, il suffit de la sélectionner. Le CD Audio fournit un accès direct aux chansons situées sur le CD.

En informatique, nous distinguons deux types d'accès aux données d'un fichier :
- **Accès séquentiel :** Pour lire une information particulière, il faut lire toutes les informations situées avant.
- **Accès direct :** Nous pouvons accéder directement à l'information désirée, en précisant le numéro d'emplacement (le numéro d'ordre) de cette information.

Tout fichier peut être utilisé avec l'un ou l'autre des types d'accès. Le choix de ce type n'est pas un choix qui concerne le fichier lui-même, mais uniquement la manière dont il va être traité par la machine. C'est donc dans le programme, et seulement dans le programme, que l'on choisissons le type d'accès souhaité.

IV. Les fichiers à accès séquentiel

IV.1 Présentation

Un fichier à accès séquentiel (ou fichier séquentiel) est une suite de valeurs (appelées aussi enregistrements) disposées les unes à la suite des autres, de façon que la lecture d'une valeur donnée exemple de la quatrième valeur ne puisse se faire qu'après la lecture des valeurs précédentes.

Déclaration

La structure fichier se déclare soit comme un type prédéfini soit comme un nouveau type utilisateur.Son utilisation se fait à travers une variable (un identificateur du fichier) de type : Le type prédéfini ou le nouveau type déclaré.

En algorithmique

Tableau de déclaration des nouveaux types

Type
nom_fichier = **Fichier** de type_composants

type_composants représente le type des composants du fichier, ces composants doivent être de même type, exemple : entiers, réels, enregistrements,etc

Codification des objets

Objet	Type / Nature	Rôle
nom_logique	nom_fichier	Fichier pour …

Nous désignons par le mot "logique" utilisé dans l'appellation de l'objet, une variable utilisée seulement dans le programme, il est aussi appelé nom_interne. Nous allons voir par la suite qu'il y a un autre objet que nous allons déclarer et que nous allons nommé nom_physique appelé aussi nom_externe, car cette variable concerne le nom du fichier sur le support externe de stockage (disque dur, disquette, etc).

En Pascal

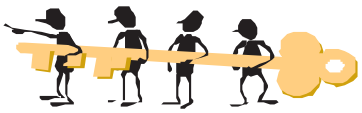
```
TYPE
      nom_fichier = File  Of  type_composants ;

VAR
      nom_logique : nom_fichier ;
```

Activité 4

Nous voulons saisir et enregistrer les notes des élèves d'une classe dans un fichier de réels.

Question :

Déclarez (en algorithmique et en Pascal) la structure de fichier adéquate.



En algorithmique

Tableau de déclaration des nouveaux types

Type
notes_élève = Fichier de réels

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
notes	notes_élèves	Fichier pour les notes des élèves.

La variable *notes* qui est l'identificateur logique du fichier de type note_élèves servira à effectuer des opérations sur un fichier de ce type.

En Pascal

```
TYPE
        notes_eleves = File Of Real ;
VAR
        notes : notes_eleves
```

Activité 5

Au lieu de sauvegarder seulement les notes des élèves, nous proposons d'enregistrer dans un fichier appelé **CARNET**, la fiche de renseignements et les notes de chaque élève.
Une fiche de renseignements comporte les informations suivantes :
– Nom et prénom,
– Sexe (F ou G),
– Date de naissance,
Les notes sont : note1, note2 et note3.

Nous remarquons que cette fiche comporte des données de différents types (chaîne de caractères, caractères, réels, etc).
La structure fichier n'accepte que des données de même type. Dans ce cas, nous devons utiliser le type enregistrement.

Déclarations

Tableau de déclaration des nouveaux types

Type
{déclaration de l'enregistrement} Fiche = enregistrement nom_prénom : Chaîne [30] sexe : Caractère date_naissance : Chaîne note1, note2, note3 : Réel Fin Fiche {déclaration du fichier} Carnet = Fichier de Fiche

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
info	Fiche	Variable de type Fiche.
élève	Carnet	Variable pour l'utilisation du fichier Carnet.

IV.2 Traitement sur les fichiers

Etape 1 : Association (ou Assignation)

Les données du fichier seront enregistrées sur un support externe comme une disquette, un disque dur ou tout autre support d'entrée / sortie. Ce fichier doit avoir un nom et de préférence une extension. Ce nom de fichier est appelé le nom externe ou le nom physique.

Nous avons vu que dans le programme, la variable déclarée dans le tableau de déclaration des objets, pour l'utilisation du fichier, représente le nom interne ou le nom logique du fichier.

nom_externe ou nom_physique : c'est le nom du fichier pour le système d'exploitation considéré, par exemple : fichier_eleve.dat

N.B : Un fichier **.dat** est un fichier de données (data)

nom_interne ou nom_logique : c'est le nom du fichier déclaré et utilisé par le programme, par exemple : élève

Nous constatons que pour un fichier il existe deux noms (un nom interne et un nom externe). Il faut donc, avant d'utiliser un fichier, relier ou encore associer son nom logique au nom physique du fichier par la commande **<Associer>** (en Pascal **ASSIGN**).

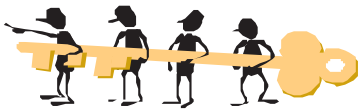
Syntaxe

Algorithmique	Pascal
Associer (nom logique, nom physique)	`ASSIGN (nom_logique, nom_physique) ;`

La syntaxe algorithmique :
Assigner (nom logique, nom physique)
est aussi très utilisée.

Activité 6

Associez, la variable logique élève du fichier de type carnet de l'activité précédente, à un nom physique sachant que le fichier sera enregistré sur le disque dur "C", dans le dossier "Archive" et portera le nom de "Fichier_eleve" et l'extension **dat**.



En algorithmique

Associer (élève, "C:\Archive\Fichier_eleve.dat")

En Pascal

```
Assign (eleve, 'C:\Archive\Fichier_eleve.dat') ;
```

Une fois cette assignation ou association faite, chaque fois que nous voulons communiquer avec le fichier " **C:\Archive\Fichier_eleve.dat** ", nous l'appellerons par le nom logique "élève" qui est l'identificateur que nous avons choisi comme variable fichier.

Sans avoir recours à la déclaration d'autres unités dans la clause Uses, Pascal ne permet pas la création d'un chemin complet. Si le nom physique comporte des dossiers, ces derniers doivent être crées, au niveau du système d'exploitation, par le programmeur ou l'utilisateur sinon, lors de l'exécution, le programme affiche un message d'erreur de type "Runtime error … at …. "

Etape 2 : Ouverture

Après l'étape d'association, nous pouvons maintenant ouvrir le fichier soit en lecture, soit en écriture ou en réécriture.

Pour les fichiers de données, l'ouverture peut se faire de deux façons différentes, chacune a ses caractéristiques :

1- Ouverture et création :

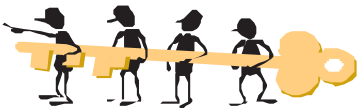
L'instruction "Recréer" permet d'ouvrir un fichier et d'effacer son contenu (recréer le fichier), si le fichier n'existe pas, il sera créé.

Syntaxe

Algorithmique	Pascal
Recréer (nom logique)	`ReWrite (nom_logique) ;`

Activité 7

Ouvrez, en vu d'une nouvelle création, le fichier "élève" de l'activité précédente.



Algorithmique	Pascal
Recréer (élève)	`ReWrite (eleve) ;`

2- Ouverture et remise à zéro :

Ouvrir un fichier existant et repositionner ou remettre à zéro (ReSet) son pointeur. Cette instruction permet l'ajout ou la modification des enregistrements.

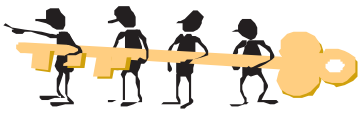
Il est à noter que le premier élément ou enregistrement d'un fichier occupe la position numéro zéro du fichier (nous commençons à compter à partir de zéro).

Syntaxe

Algorithmique	Pascal
Ouvrir (nom logique)	`ReSet (nom_logique) ;`

Activité 8

Ouvrez, et placez le pointeur sur le premier élément (remettre à zéro le pointeur) pour des modifications, du fichier "élève" déjà crée par "ouvrir".



Algorithmique	Pascal
Ouvrir (élève)	`ReSet (eleve) ;`

Etape 3 : Ecriture dans un fichier

L'écriture ou la modification d'une valeur ou d'un enregistrement dans un fichier se fait grâce à l'instruction **<Ecrire>** suivi du nom logique du fichier puis de la variable ou de la valeur à enregistrer dans le fichier :

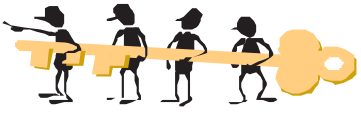
Syntaxe

Algorithmique	Pascal
Ecrire (nom logique, variable)	`Write (nom_logique, variable) ;`

A chaque écriture d'une valeur ou d'un enregistrement, le pointeur du fichier avance automatiquement d'une position.

Activité 9

Nous proposons de saisir un entier N et de l'écrire (l'enregistrer) dans un fichier appelé **Valeurs**.
Ecrivez en algorithmique et en Pascal les instructions nécessaires pour réaliser cette tâche.

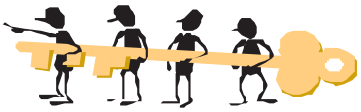


En algorithmique	En Pascal

{Lecture de l'entier N}
Lire (N)
 	
{Ecriture dans le fichier}
Ecrire (Valeurs, N)
	`{Lecture de l'entier N}`
	`ReadLn (N) ;`
	
	`{Ecriture dans le fichier}`
	`Write (Valeurs, N) ;`

Activité 10

Nous voulons saisir un enregistrement de type "fiche" et le sauvegarder dans le fichier "élève".
Donnez des solutions différentes au problème.



Solution 1 **Sans la structure "Avec .. Faire"**	**Solution 2** **Avec la structure "Avec .. Faire"**
{Etape 1 : Lecture de tous les champs de l'enregistrement} Ecrire ("Donner le nom de l'élève : ") Lire (info.nom_prénom) Ecrire ("Donner son sexe ") Lire (info.sexe) Ecrire ("Donner sa date de naissance : ") Lire (info.date_naissance) Ecrire ("Donner sa première note : ") Lire (info.note1) Ecrire ("Donner sa deuxième note : ") Lire (info.note2) Ecrire ("Donner sa troisième note : ") Lire (info.note3) {Etape 2 : Ecriture de l'enregistrement dans le fichier} Ecrire (élève, info)	{Etape 1 : Lecture de tous les champs de l'enregistrement} **Avec** info **Faire** Ecrire ("Donner le nom de l'élève : ") Lire (nom_prénom) Ecrire ("Donner son sexe ") Lire (sexe) Ecrire ("Donner sa date de naissance : ") Lire (date_naissance) Ecrire ("Donner sa première note : ") Lire (note1) Ecrire ("Donner sa deuxième note : ") Lire (note2) Ecrire ("Donner sa troisième note : ") Lire (note3) **Fin Avec** {Etape 2 : Ecriture de l'enregistrement dans le fichier} Ecrire (élève, info)

Etape 4 : Lecture à partir d'un fichier

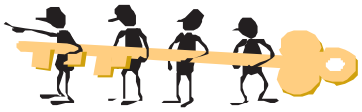
La lecture d'une valeur ou d'un enregistrement du fichier se fait par l'instruction **<Lire>** suivi du nom logique du fichier puis de la variable à lire.

Syntaxe

Algorithmique	Pascal
Lire (nom logique, variable)	`Read (nom_logique, variable) ;`

Activité 11

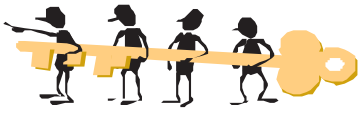
Nous proposons de lire l'entier V pointé dans le fichier de nom **Valeurs**. Ecrivez en algorithmique et en Pascal, les instructions nécessaires pour réaliser cette tâche.



Algorithmique	Pascal
Lire (Valeurs, V)	`Read (Valeurs, V) ;`

Activité 12

Nous voulons lire l'enregistrement "info" de type "fiche" qui est sauvegardé dans le fichier "élève" puis d'afficher tous les champs de cet enregistrement. Ecrivez en algorithmique et en Pascal, les instructions nécessaires pour réaliser cette tâche.



Solution 1 Sans la structure "Avec .. Faire"	Solution 2 Avec la structure "Avec .. Faire"
{Etape 1 : Lecture de l'enregistrement info à partir du fichier} Lire (élève, info) {Etape 2 : Ecriture des champs de l'enregistrement info de type fiche} Ecrire ("Nom : ", info.nom_prénom) Ecrire ("Sexe ", info.sexe) Ecrire ("Date de naissance : ", info.date_naissance) Ecrire ("Première note : ", info.note1) Ecrire ("Deuxième note : ", info.note2) Ecrire ("Troisième note : ", info.note3)	{Etape 1 : Lecture de l'enregistrement info à partir du fichier} Lire (élève, info) {Etape 2 : Ecriture des champs de l'enregistrement info de type fiche} **Avec** info **Faire** Ecrire ("Nom : ", nom_prénom) Ecrire ("Sexe ", sexe) Ecrire ("Date de naissance : ", date_naissance) Ecrire ("Première note : ", note1) Ecrire ("Deuxième note : ", note2) Ecrire ("Troisième note : ", note3) **Fin Avec**

Etape 5 : Fermeture d'un fichier

A la fin du traitement, nous devons fermer le (ou les) fichier(s) ouvert(s).
La fermeture d'un fichier spécifique se fait par l'instruction **<Fermer>** suivi du nom logique du fichier.

Syntaxe

Algorithmique	Pascal
Fermer (nom logique)	`Close (nom_logique) ;`

Activité 13

Fermez le fichier ouvert nommé "élève".

Algorithmique	Pascal
Fermer (élève)	`Close (eleve) ;`

Etape intermédiaire : Test de fin du fichier

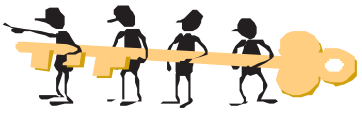
Au cours d'un programme, nous pouvons à tout moment tester si nous avons atteint la fin d'un fichier en lecture par la fonction booléenne **<Fin_fichier>** suivie du nom logique du fichier qui retourne **VRAI** si la fin du fichier est atteinte.

Syntaxe

Algorithmique	Pascal
Fin_fichier (nom logique)	`Eof (nom_logique);`

Activité 14

Affichez le nombre de garçons présents dans le fichier élève sachant que le nombre de fiches de ce fichier est inconnu.
Ecrivez en algorithmique et en Pascal, les instructions nécessaires pour réaliser cette tâche.



Analyse :

Résultat : Afficher **nbg** le nombre de garçons parmi les élèves.
Traitement : - Le nombre d'élèves du fichier est inconnu, et pouvant être nul, d'où la structure itérative à condition d'arrêt Tant que .. Faire sera utilisée :

[nbg ← 0]
Tant que Non (Fin_fichier (élève)) **Faire**
Lire (élève, info)
Si info.sexe ="G" **Alors** nbg ← nbg +1

En algorithmique	En Pascal
nbg← 0	`nbg := 0 ;`
Tant que Non (**Fin_fichier** (élève)) Faire	`While not(Eof(eleve)) DO`
Lire (élève, info)	`Begin`
Si info.sexe = "G" Alors nbg := nbg +1	`Read (eleve, info);`
Fin Si	`If info.sexe='G' Then nbg:=nbg+1 ;`
Fin Tant Que	`End ;`
Ecrire ("Nombre de garçons : ", nbg)	`WriteLn ('Nombre de garcons:', nbg);`

Vérification des Entrées/Sorties en Pascal

Si, dans un programme donné, nous voulons remettre à zéro un fichier par l'instruction ***ReSet*** *(nom_logique)* et que ce fichier n'existe pas, un message d'erreur est affiché et le programme est arrêté.
Pour éviter ceci, nous pouvons tester l'existence ou non d'un fichier et puis agir en conséquence.

Test d'existence du fichier
La directive de compilation **{$I ..}** utilisée avec la fonction **IOResult** qui renvoie le résultat des opérations d'entrée/sortie, permet au programme en cours d'utilisation de contrôler toute la communication avec les périphériques externes comme les unités de disque et d'éviter les arrêts sur l'erreur d'entrée/sortie **(IO Error)** quand un fichier n'existe pas.

Par défaut la directive est **{$I+}** et sous l'effet de cette option, le compilateur génère du code supplémentaire pour tester la validité de chaque opération d'entrées/sortie. Si une erreur est détectée, le programme s'arrête sur une erreur d'exécution.

Avec la directive **{$I-}**, l'utilisateur doit gérer lui-même ce type d'erreur grâce à la variable IOResult. (IOResult : Input Output Result).

Exemple : Saisie et ouverture du nom physique d'un fichier. Si le fichier n'existe pas il sera créé sans arrêt du programme sur l'erreur : *fichier inexistant (IO Error)*.

```
Write ('Nom du fichier : '); ReadLn (nom);
Assign (Classe , nom );
    {$I-}
ReSet ( Classe );
IF IOResult <> 0 THEN ReWrite( Classe) ;
   {$I+}
```

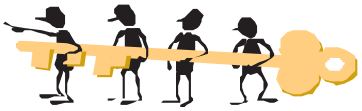
Activité 15

Nous nous proposons d'écrire un programme nommé **F_Positif**, qui :
- Enregistre sur le disque dur "C" et dans le dossier "4SI", un fichier ayant pour nom **Nombres.FCH** comportant des entiers positifs saisis au clavier. Le nombre de ces entiers est inconnu à l'avance. La condition d'arrêt est la saisie d'un nombre négatif. Ce dernier ne sera pas enregistré dans le fichier.
- Affiche tous les éléments du fichier.

Questions :
- Proposez une analyse de ce problème,
- Déduisez les algorithmes correspondants,
- Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Fichier1**

Chapitre

1



1- Analyses et algorithmes :

Analyse du programme principal

Résultat : - Afficher le contenu du fichier **Nombres.FCH**, associé au nom interne **Liste**. Nous appellerons la procédure Affichage (Liste)
Traitement : - La saisie des différents entiers positifs est confiée à la procédure Saisie(Liste).

Algorithme du programme principal

0) Début F_Positif
1) Proc Création (Liste)
2) Proc Saisie (Liste)
3) Proc Affichage (Liste)
4) Fermer (Liste)
5) Fin F_Positif

Codification des nouveaux types :

Type
F_Entiers = Fichier d'entiers

Codification des objets globaux :

Nom	Type / Nature	Rôle
Liste	F_Entiers	Nom logique du fichier
Création	Procédure	Création et ouverture du fichier
Saisie	Procédure	Saisie et enregistrement des entiers dans le fichier
Affichage	Procédure	Lecture et affichage du contenu du fichier

Analyse de la procédure Création

Résultat : Créer et ouvrir le fichier **Liste**
Traitement : La création et l'ouverture du fichier liste se feront avec le nom logique et par l'instruction Recréer (liste).
– Une association du nom logique avec le nom physique doit être effectué:
Associer (Liste, Chemin).

Algorithme de la procédure Création

0) Début procédure Création (Var Liste : F_Entier)
1) Chemin ← "C:\4SI\Nombres.FCH"
2) Associer (Liste, Chemin)
3) Recréer (Liste)
4) Fin Création

Codification des objets locaux

Nom	Type / Nature	Rôle
Chemin	Constante = "C:\4SI\Nombres.FCH"	Nom physique et chemin du fichier

Analyse de la procédure Saisie

Résultat : Saisir et enregistrer les entiers dans le fichier
Traitement : Il s'agit d'une action répétitive à condition d'arrêt :

V = Donnée ("Entrer un entier : ")
Tant que V ≥ 0 **Faire**
Ecrire (Liste, V)
V = Donnée ("Entrer un entier : ")

Algorithme de la procédure Saisie

0) Début procédure Saisie (**Var** Liste : F_Entiers)
1) Ecrire ("Entrer un entier : ") ; Lire (V)
2) **Tant que** V ≥ 0 **Faire**
Ecrire (Liste , V)
Ecrire ("Entrer un entier : ") , Lire (V)
Fin Tant que
2) Fin Saisie

Codification des objets locaux

Nom	Type / Nature	Rôle
V	Entier	Valeur au clavier

Analyse de la procédure Affichage

Résultat : Lire et afficher les éléments du fichier.
Traitement : - La lecture à partir du fichier se fera, élément par élément, par l'instruction <Lire (nom logique, variable)> et ce jusqu'à la fin du fichier. C'est une répétition à condition d'arrêt :

Tant que Non (Fin_fichier (Liste)) **Faire**
Lire (Liste, V)
Ecrire (V)

NB : Il faut prévoir ouvrir le fichier.

Algorithme de la procédure Affichage

0) Début procédure Affichage (**Var** Liste : F_Entiers)
1) Ouvrir (Liste)
2) **Tant que** Non (Fin_fichier (Liste)) **Faire**
Lire (Liste, V)
Ecrire (V)
Fin Tant que
3) Fin Affichage

Traduction en Pascal

```
PROGRAM F_Positif ;
Uses Crt ;
Type
      F_Entiers = File Of integer ;

Var
    Liste : F_Entiers ;
    V : Integer ;

PROCEDURE Creation (VAR Liste : F_Entiers) ;
VAR   chemin : String ;
Begin
      chemin := 'C:\4SI\Nombres.FCH' ;
      Assign (Liste , Chemin ) ;
      ReWrite (Liste) ;
End ;

PROCEDURE Saisie (VAR Liste : F_Entiers) ;
Begin
      Write ('Entrer un entier : ') ; ReadLn ( V) ;
      While (v>=0) Do
      Begin
            Write (Liste , V) ;
            Write ('Entrer un entier : ') ; ReadLn ( V) ;
      End ;
End ;
```

```
PROCEDURE Affichage (VAR Liste : F_Entiers) ;
Begin
      ReSet (Liste) ;
      While Not (Eof (Liste)) Do
      Begin
            Read (Liste ,  V) ;
            WriteLn (V) ;
      End ;
 End ;
{ Programme principal }

BEGIN
      Creation (Liste) ;
      Saisie (Liste) ;
      Affichage (Liste) ;
      Close (Liste) ;
END.
```

Activité 16

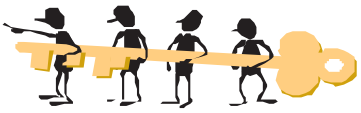
Une société veut informatiser la gestion de son personnel, en sauvegardant dans un fichier nommé **PERSONNEL** ses fiches de renseignements.
Chaque fiche comporte les renseignements suivants :
Nom & prénom (chaîne de caractères), N° CIN (entier non signé), matricule (entier non signé), grade (A, B, C ou D), adresse (chaîne de caractères), N° de téléphone (chaîne de caractère) et l'ancienneté (Réel).

* Ecrivez un programme nommé **GESTION** qui permet :

– La saisie et la sauvegarde des fiches dans le fichier PERSO.DAT sur le disque "C". La fin de la saisie est possible si nous répondons N (Non) à la question "Continuer (O/N) ? ".

– L'affichage des fiches (une fiche par page).

* Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Fichier2**.



1- Analyses et algorithmes :

Analyse du programme principal

Résultat : Enregistrement et affichage des fiches.

Traitement : - L'affichage des fiches est la tâche de la procédure Affichage

- La saisie des différentes fiches est confiée à la procédure Saisie.

NB : La structure enregistrement peut être utilisée dans ce cas de problème.

Algorithme du programme principal :

0) Début Gestion
1) Proc Création (classeur)
2) Proc Saisie (classeur, fiche)
3) Proc Affichage (classeur, fiche)
4) Fermer (classeur)
5) Fin Gestion

Codification des nouveaux types :

Type
Les_Fiches = Enregistrement nom_prenom, adresse, tel : chaîne cin, matricule : entier non signé grade : caractère ancien : Réel Fin Les_Fiches Personnels = Fichier de Les_Fiches

Codification des objets globaux :

Nom	Type / Nature	Rôle
Classeur	Personnels	Nom logique du fichier
Fiche	Les_Fiches	Une fiche de renseignements
Création	Procédure	Création et ouverture du fichier
Saisie	Procédure	Saisie et sauvegarde des enregistrements dans le fichier
Affichage	Procédure	Lecture et affichage du contenu du fichier

Analyse de la procédure Création

Résultat : Créer et ouvrir le fichier "Classeur"

Traitement : La création et l'ouverture du fichier **Classeur** se feront avec le nom logique et par l'instruction Ouvrir (Classeur).

- Une association du nom logique avec le nom physique est obligatoire : Associer (Classeur, Chemin)

Algorithme de la procédure Création

0) Début procédure Création (Var Classeur : Personnels)
1) Chemin ← " C:\Perso.dat "
2) Associer (Classeur, Chemin)
3) Ouvrir (Classeur)
4) Fin Création

Codification des objets locaux :

Nom	Type / Nature	Rôle
Chemin	Constante = "C:\Perso.dat"	Nom physique du fichier

Analyse de la procédure Saisie

Résultat : Saisir et enregistrer les enregistrements dans le fichier
Traitement : La saisie des enregistrements se fera par des opérations de lecture, leur écriture dans le fichier se fera au fur et à mesure. Le test de fin de la répétition de la saisie est la réponse par Non à la question "Continuer (O/N) ? ". D'où l'utilisation d'une structure itérative à condition d'arrêt :

```
Répéter
    Avec Fiche Faire
… {Lecture des champs}
    Fin Avec
        Ecrire (Classeur, Fiche)
      Répéter
        Rep = Donnée ("Continuer (O / N ) ? : ")
     Jusqu'à (REP dans ["O", "N"] )
    Jusqu'à (Majuscule (rep) = "N")
```

Algorithme de la procédure Saisie

```
0) Début procédure Saisie (Var Classeur  : Personnels  ; Fiche : Les_Fiches)
1) {Saisie des champs}
   Répéter Saisie des champs
        Avec Fiche Faire
            Ecrire ("Nom & prénom : ") , Lire (nom_prenom)
            Ecrire ("N° Cin           : ") , Lire (cin)
            Ecrire ("Matricule        : ") , Lire (matricule)
            Répéter
                Ecrire ("Grade              : ") , Lire (grade)
            Jusqu'à (grade dans ["A".."D"] )
            Ecrire ("Adresse          : ") , Lire (adresse)
            Ecrire ("N° de téléphone : ") , Lire (tel)
            Ecrire ("Ancienneté       : ") , Lire (ancien)
        Fin Avec
  {Ecriture dans le fichier}
   Ecrire (Classeur, Fiche)
   {Question de continuité}
        Répéter
            Ecrire ("Continuer (O / N ) ? : ")
            Lire (rep)
        Jusqu'à (REP dans ["O", "N"] )
    Jusqu'à (REP = "N")
2) Fin Saisie
```

Codification des objets locaux :

Objet	Type / Nature	Rôle
rep	Caractère	Réponse à la question Continuer (O / N) ?

Analyse de la procédure Affichage

Résultat : Lire et afficher, fiche par fiche, les éléments du fichier.
Traitement : - La lecture à partir du fichier se fera, enregistrement par enregistrement, à l'aide de l'instruction <Lire (nom logique, variable)> et ce jusqu'à la fin du fichier.

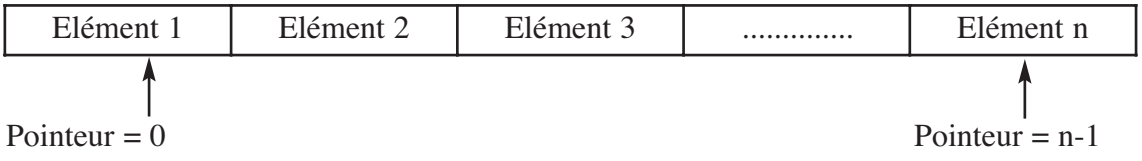
Algorithme de la procédure Affichage

```
0) Début procédure Affichage (Var Classeur : Personnels ; Fiche : Les_Fiches)
1) Ouvrir (Classeur)
2)   Tant que Non (Fin_fichier (Classeur)) Faire
       Lire (Classeur, Fiche)
       Effacer l'écran
       Avec Fiche Faire
            Ecrire ("Nom & prénom : ", nom_prenom)
            Ecrire ("N° Cin       : ", cin)
            Ecrire ("Matricule    : ", matricule)
            Ecrire ("Grade        : ", grade)
            Ecrire ("Adresse      : ", adresse)
            Ecrire ("N° de téléphone : ", tel)
            Ecrire ("Ancienneté   : ", ancien)
       Fin Avec
       Lire ( )  {Pause jusqu'à l'appui sur ENTRER}
   Fin Tant que
3) Fin Affichage
```

V. Les fichiers à accès direct

V.1 Définition

Un fichier est dit à accès direct si on peut accéder directement à chacun de ses éléments. Cette opération s'effectue grâce à la procédure **<POINTER>** (en Pascal **SEEK**), qui nécessite deux paramètres : Le **nom logique** du fichier et le **numéro d'ordre de l'élément** auquel on veut accéder. Ce numéro vaut 0 (zéro) pour le 1er élément. Ce qui fait que pour accéder au n^{ième} élément il faut pointer l'élément n-1.



Syntaxe

Algorithmique	Pascal
Pointer (nom logique, numéro)	`Seek (nom_logique, numero);`

Il faut faire attention de ne pas se positionner (ou demander un élément) après la fin physique du fichier. Ce contrôle de dépassement doit être fait par le programmeur grâce à la fonction **<Taille_fichier>** (en Pascal **FileSize**), qui permet de savoir combien d'éléments se trouvent dans un fichier.

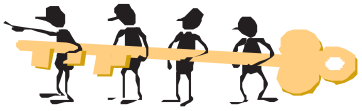
Syntaxe

Algorithmique	Pascal
Taille_fichier (nom logique)	`FileSize (nom_logique);`

A l'aide de la fonction **FileSize**, on peut se positionner à la fin du fichier de manière à rajouter des éléments.

Activité 17

Reprenez le fichier **Liste** de l'activité 15 et y ajouter un entier.



Ouvrir (liste)
Ecrire ("donner un entier : ") ; Lire (v)
p ← taille_fichier (liste)
Pointer (liste, p)
Ecrire (liste, v)

Nous pouvons aussi faire :

Ouvrir (liste)
Ecrire ("donner un entier : "); Lire (v)
Pointer (liste, taille_fichier (liste))
Ecrire (liste, v)

V.2 Autres fonctions et procédures prédéfinies

Ces fonctions et ces procédures sont valables aussi bien pour l'accès direct que pour l'accès séquentiel.

a. Fonction Position_fichier

Syntaxe

Algorithmique	Pascal
Position_fichier (nom logique)	`FilePos (nom_logique);`

Rôle : Fonction qui fournit le numéro d'ordre (la position) de l'élément sur lequel se trouve le pointeur du fichier. Ce numéro est un entier qui vaut 0 pour le premier élément

Remarque : Cette fonction ne peut pas s'appliquer à un fichier texte.

b. Procédure Effacer

Syntaxe

Algorithmique	Pascal
Effacer (nom logique)	`Erase (nom_logique);`

Rôle : Procédure qui efface le fichier ayant pour nom de travail "nom logique". Celui ci peut être indifféremment ouvert ou fermé avant effacement.

c. Procédure Renommer

Syntaxe

Algorithmique	Pascal
Renommer (nom logique, nouveau nom physique)	`Rename (nom_logique, nouveau_nom_physique);`

Rôle : Procédure qui change le nom physique d'un fichier.

Nous devons fermer le fichier avant de le renommer.

d. Procédure Tronquer

Syntaxe

Algorithmique	Pascal
Tronquer (nom logique)	`Truncate (nom_logique);`

Rôle : Tronque le fichier de travail "nom logique" à la position courante du pointeur de fichier. Tout ce qui se trouve au-delà du pointeur est alors perdu.

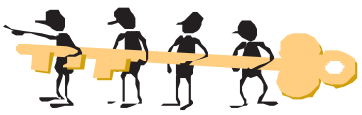
Le langage Pascal ne gère pas en standard les instructions du système d'exploitation tel que renommer, effacer, etc.
Pour les utiliser dans un programme il est préférable de déclarer l'utilisation de l'unité "Dos" :
USES Crt, Dos ; {si on travaille avec free Pascal ou Pascal 7.x}
USES WinCrt, WinDos ; {si on travaille avec Pascal sous Windows TPW 1.x}

V.3 Application

Nous voulons accéder de façon directe et lire les données enregistrées dans le fichier **Liste** de l'activité 15 de la façon suivante :

- Saisir au clavier le numéro d'ordre de la donnée à lire
- Si le numéro d'ordre saisi ne correspond pas à une position valide dans le fichier, le message "Vous êtes hors du fichier" sera affiché, sinon c'est la donnée lue qui sera affichée.
- Répéter les deux actions précédentes jusqu'à la réponse par Non à la question "Continuer (O/N) ? ".

Ecrivez une procédure, qui réalise cette tâche.



Analyse de la procédure Lecture

Résultat : Saisir la position d'un enregistrement, pointer le fichier et lire son contenu.
Traitement : La position saisie ne doit pas être inférieure à zéro et ne doit pas dépasser la taille du fichier, donc un test de validité est nécessaire (Répéter .. jusqu'à).
- La répétition de la lecture se fera jusqu'à la réponse par N (Non) à la question "Continuer (O/N) ? " : (Répéter .. jusqu'à)

Algorithme de la procédure Lecture

```
0) Début procédure Lecture (Var Liste : F_Entiers)
1) {Connaître la taille du fichier}
        T ← Taille_fichier (Liste)
2) {Lire les entiers à partir du fichier}
   Répéter
        Répéter
                Ecrire ("Numéro de l'entier à lire : ") , Lire (num)
                 Si ( (num ≥ T) OU (num < 0) )Alors Ecrire ("Vous êtes hors du fichier ")
                Fin Si
        Jusqu'à ( num < T) OU (num ≥ 0)
        Pointer (Liste, num)
        Lire (Liste, V)
        Ecrire ("L'entier est : ", V)
        Répéter
                Ecrire ("Continuer (O/N) ? ")
                Lire (rep)
                rep ← Majuscule (rep)
        Jusqu'à (rep dans ["O", "N"] )
   Jusqu'à (rep = "N")
3) Fin Lecture
```

VI. Les fichiers texte

VI.1 Définition

Un fichier texte (ou le plus souvent et abusivement fichier ASCII), est un fichier dont le contenu représente uniquement une suite de caractères.

Il faut noter qu'en informatique, l'espace et le retour à la ligne sont considérés comme des caractères au même titre qu'une lettre alphabétique, un chiffre ou un signe de ponctuation.

VI.2 Présentation

Les fichiers textes constituent une catégorie particulière. En effet, à la différence des autres fichiers, ils contiennent des caractères du type "Retour chariot" (CR) ou "Fin de ligne" (EoLn) et "Fin de texte" (code CTRL-Z).

Une variable de type fichier texte est déclarée par le mot réservé **<TEXT>**. Comme pour les autres fichiers, il faut associer (en Pascal **ASSIGN**) le nom externe (le nom physique) du fichier à un nom logique qui est la variable de type **TEXT**. Un fichier texte peut être **OUVERT** en lecture par **<Ouvrir>** (en Pascal **ReSet**) et en écriture par **<Recréer>** (en Pascal **ReWrite**).

Les fichiers texte sont utilisés par de nombreux logiciels pour conserver les données de configuration. Ils sont également utilisés pour contenir les textes écrits en langages de programmation. En outre, la plupart des langages de programmation offrent des fonctions prédéfinies pour manipuler du texte brut, ce qui rend la gestion des fichiers textes particulièrement accessible.
La structure fichier texte est déclarée comme une variable de type texte.

En algorithmique

Tableau de déclaration des objets

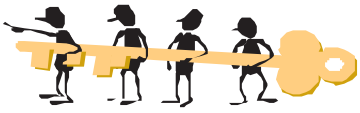
Objet	Type / Nature	Rôle
nom_logique	Texte	Fichier texte …

En Pascal

```
VAR    nom_logique   : Text ;
```

Activité 18

Déclarez un fichier texte nommé (**Ftexte**) et associez-le au nom physique "Essai.txt" à enregistrer sur disquette.



En algorithmique

Tableau de déclaration des objets :

Objet	Type / Nature	Rôle
Ftexte	Texte	Fichier texte

Associer (Ftexte , "A:ESSAI.TXT")

En Pascal

```
VAR    Ftext : text ;

BEGIN
      Assign ( Ftext , 'A:ESSAI.TXT' ) ;
      …
      Close (Ftext) ;
END.
```

VI.3 Procédures et fonctions prédéfinies

Nous retrouvons toutes les fonctions et les procédures déjà vues pour les fichiers de données. De plus, les fichiers textes permettent de rajouter les commandes liées aux chaînes de caractères (string).

a. Fonction Fin_ligne

Syntaxe

Algorithmique	Pascal
Fin_ligne (nom logique)	`EoLn (nom_logique);`

Rôle : Fonction qui retourne Vrai (TRUE) si l'on se trouve sur le caractère CR (retour chariot) et Faux (FALSE) dans le cas contraire.

b. Procédure Ajouter

Syntaxe

Algorithmique	Pascal
Ajouter (nom logique)	`Append (nom_logique);`

Rôle : Procédure qui ouvre le fichier et positionne son pointeur à la fin de ce dernier. Seul l'ajout d'éléments est alors possible.

c. Procédure Lire_nl

Syntaxe

Algorithmique	Pascal
Lire_nl (nom logique, variable)	`ReadLn (nom_logique, variable);`

Rôle : Procédure qui lit le contenu d'une ligne, puis pointe la prochaine ligne (nl = nouvelle ligne). Elle place le pointeur de fichier sur le début de cette ligne, c'est à dire juste après la première marque CR-LF (Retour Chariot et Fin de Ligne) rencontrée.

d. Procédure Ecrire_nl

Syntaxe

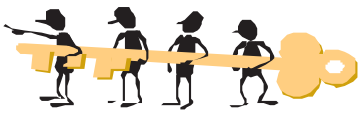
Algorithmique	Pascal
Ecrire_nl (nom logique, valeur)	`WriteLn (nom_logique, valeur);`

Rôle : Procédure qui introduit dans le fichier texte un contenu puis une séquence CR-LF pour marquer la fin de ligne.

VI.4 Applications

Application 1

Ecrivez une procédure nommée **Ecriture_texte**, qui :
- Saisit des lignes de texte et les sauvegardent dans un fichier texte.
- Après chaque saisie et enregistrement des lignes, nous testons la sortie par "Quitter (O/N)?".



Analyse de la procédure Ecriture_texte

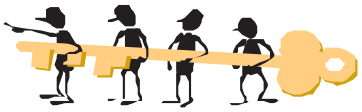
Résultat : Saisir et enregistrer des lignes de texte
Traitement : Répéter
La saisie d'une ligne
L'enregistrement de la ligne dans le fichier
Saisir le choix de continuation
jusqu'à choix = Non

Algorithme de la procédure Ecriture_texte

```
0) Début procédure Ecriture_Texte (Var Lignes : Text)
1) Répéter
        Ecrire ("Taper une ligne de texte : ")
        Lire (phrase)
        Ecrire_nl (Lignes , phrase)

        Répéter
            Ecrire ("Quitter (O/N) ? ")
            rep ← Lire_Touche
            rep ← Majuscule (rep)
        Jusqu'à (rep Dans ["O", "N"] )

 Jusqu'à (rep = "O")
2) Fin Ecriture_Texte
```

Application 2

Ecrivez une procédure nommée **Affichage_texte**, qui lit puis affiche le texte enregistré par la procédure "Ecriture texte" de l'activité précédente.



Analyse de la procédure Affichage_texte

Résultat : Affichage des lignes de texte

Traitement : - **Tant que** la fin du fichier n'est pas encore atteinte **Faire**
Lire une ligne du fichier
afficher cette ligne

- Pour commencer la lecture des lignes, il faut ouvrir le fichier en plaçant le pointeur sur le début.

Algorithme de la procédure Affichage_texte

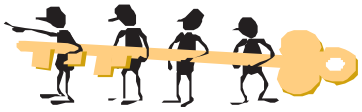
```
0) Début procédure Affichage_Texte (Var Lignes : Text)
1) Ouvrir (Lignes)
2) Tant que Non (Fin_fichier (Lignes)) Faire
     Lire_nl (Lignes, ligne)
     Ecrire (ligne)
   Fin tant que
3) Fin Affichage_texte
```

Application 3

* En utilisant les deux procédures précédentes, écrivez un programme nommé **Ecriture_Lecture**, qui :

- ouvre un fichier dont le chemin est "C:\Lignes.fch".
- saisit et enregistre les lignes de texte.
- lit le fichier et affiche son contenu.
- ferme le fichier.

* Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Fichier3**.



Analyse du programme principal

Résultat : Affichage des lignes.
Traitement : - L'affichage des lignes est réalisé par la procédure Affiche_Texte.
- Pour afficher les lignes de texte, le fichier doit être ouvert.
- La saisie des lignes et leur enregistrement dans le fichier sont la tâche de la procédure Ecriture_Texte.
- Pour saisir les lignes, le fichier doit être créé
NB : Il faut prévoir les actions d'association et de fermeture du fichier.

Algorithme du programme principal :

0) Début Ecriture_Lecture
1) Associer (Lignes, "C:\Lignes.fch")
2) Recréer (Lignes)
3) Proc Ecriture_Texte (Lignes)
4) Ouvrir (Lignes)
5) Proc Affichage_Texte (Lignes)
6) Fermer (Lignes)
7) Fin Ecriture_Lecture.

Tableau de déclaration des objets :

Objet	Type / Nature	Rôle
Lignes	Texte	Fichier texte
phrase	Chaîne de caractères	phrase à saisir
rep	Caractère	Répondre par O/N à la question "Quitter O/N ?"
Affichage_Texte	Procédure	Lit le fichier et affiche les lignes
Ecriture_Texte	Procédure	Saisit les lignes et les enregistre dans le fichier

Traduction en Pascal

```
PROGRAM Ecriture_Lecture ;
USES Crt ;
VAR   Lignes   : Text ;
      rep      : Char ;

PROCEDURE Ecriture_Texte (VAR Lignes : Text) ;
Var  phrase : String ;
Begin
      Repeat
            Write ('Taper une ligne de texte : '); ReadLn (ligne) ;
            WriteLn (Lignes,  phrase);
            Repeat
                  Write ('Quitter (O/N) ? ') ;
                  rep :=  ReadKey ;
                  rep :=  UpCase (rep) ;
            Until (rep IN  ['O', 'N'] ) ;
      Until (rep = 'O') ;
End ;
```

```
PROCEDURE Affichage_Texte (VAR Lignes : Text) ;
Begin
      Reset (Lignes) ;
      While Not (Eof (Lignes)) Do
      Begin
            ReadLn (Lignes, phrase) ;
            WriteLn (phrase) ;
      End ;
End;

{Programme principal }
BEGIN
      Assign (Lignes, 'C:\Lignes.fch') ;
      ReWrite (Lignes) ;
      Ecriture_Texte (Lignes) ;
      Reset (Lignes) ;
      ClrScr ;
      Affichage_Texte (Lignes) ;
     Close (Lignes) ;
END.
```

Retenons

ENREGISTREMENT :

Un enregistrement est un type de données défini par l'utilisateur qui permet de grouper des éléments de divers types. Contrairement aux tableaux qui ne permettent de grouper que des éléments de même type, les enregistrements nous permettent de combiner différents types de données.

FICHIER :

Un fichier est ensemble de données stockées sur un support d'enregistrement.
Un fichier texte est un fichier dont le contenu représente uniquement une suite de caractères informatiques.

Un fichier est enregistré sur un support tel que le disque dur, sous la forme "**nom_du_fichier.ext**".
".ext" représente l'extension du fichier.

L'accès à un fichier peut se faire de deux façons :

*** Accès séquentiel :**
L'accès au $i^{ème}$ élément du fichier ne peut se faire qu'après une lecture des (i-1) éléments précédents.

*** Accès direct :**
L'accès direct permet d'accéder directement à un élément du fichier par l'intermédiaire de son numéro d'ordre (sa position dans le fichier).

Retenons

Algorithmique	Pascal	Rôle
Associer (nom logique, nom physique)	`Assign (nom_logique, nom_physique );`	Relier ou associer le nom logique d'un fichier à son nom physique
Recréer (nom logique)	`ReWrite (nom_logique);`	Recréer le fichier, s'il n'existe pas, il sera créé
Ouvrir (nom logique)	`ReSet (nom_logique);`	Ouvrir un fichier existant et remettre à zéro son pointeur.
Ecrire (nom logique, variable)	`Write (nom_logique, variable) ;`	Ecrire ou modifier une valeur ou d'un enregistrement dans un fichier
Lire (nom logique, variable)	`Read (nom_logique, variable) ;`	Lire une valeur ou un enregistrement d'un fichier
Fermer (nom logique)	`Close (nom_logique);`	Fermer un fichier
Fin_fichier (nom logique)	`Eof (nom_logique);`	Fonction booléenne vérifiant si la fin du fichier a été atteinte
Pointer (nom logique, numéro)	`Seek (nom_logique,numero) ;`	Accéder à un élément d'un fichier à accès direct
Taille_fichier(nom logique)	`FileSize (nom_logique);`	Retourner la taille d'un fichier à accès direct
Effacer (nom logique)	`Erase (nom_logique);`	Supprimer un fichier
Renommer (nom logique, nouveau nom)	`Rename (nom_logique, nouveau_nom);`	Changer le nom d'un fichier
Fin_ligne (nom logique)	`EoLn (nom_logique);`	Fonction booléenne testant la fin d'une ligne dans un fichier texte
Ajouter (nom logique)	`Append (nom_logique);`	Ouvrir un fichier et positionner son pointeur à la fin pour ajouter des enregistrements.

Exercices

Exercice 1

Répondez par V (Vrai) ou (Faux) à chacune des questions suivantes :

a. Un tableau peut contenir en même temps :

- ☐ plusieurs variables de types différents
- ☐ uniquement 2 variables de types différents
- ☐ uniquement des variables de même type

b. Un tableau à deux dimensions (matrice) peut contenir en même temps :

- ☐ plusieurs variables de types différents
- ☐ uniquement 2 variables de types différents
- ☐ uniquement des variables de même type

c. Un type enregistrement peut contenir en même temps :

- ☐ plusieurs champs de types différents
- ☐ uniquement 2 champs de types différents
- ☐ uniquement des champs de même type

Exercice 2

Un compte en banque concerne une personne spécifiée par son nom, un numéro de compte (un entier), et un montant (un réel).
Question : Déclarez un enregistrement pour cette structure.

Exercice 3

Le programme ci dessous calcule puis affiche le translaté d'un point M1(x1, y1). Il est à noter que la translation d'un point M du plan de coordonnées x et y est représentée dans ce programme par ses coordonnées cartésiennes abscisse (absi) et ordonnée (ordo).

Des parties de ce programme ont étés effacées. Nous vous demandons de les réécrire.

```
PROGRAM Translation ;
Uses Crt ;
TYPE
      Point = Record
            …… : Real ;
            …… : Real ;
      End ;

VAR
      M :  ……    ;
      x , y :     Real ;

BEGIN
      Write ('Donner les coordonnées du point : ');
      ReadLn ( …… ) ;
      ReadLn ( …… ) ;
      Writeln ('Donner les valeurs du vecteur de translation : ') ;
      Readln (x);   Readln (y);
      { Translation du point }
      M.absi   : = …… + ……  ;
      M.ordo : = …… + …… ;

      Writeln (' Les nouvelles coordonnées sont : ');
      Writeln (M.absi , '          ' , M.ordi );
END.
```

Exercice 4

Soit la structure Info constituée par le nom (chaîne de 30 caractères maximum), le numéro de téléphone (10 caractères maximum), le numéro de carte bancaire (entier non signé).
Ecrivez un programme qui saisit puis affiche les enregistrements pour 3 personnes.

Exercice 5

Un nombre complexe **z** est représenté par sa partie réelle et sa partie imaginaire.
Ecrivez un programme qui saisit deux nombres complexes z1 et z2, puis calcule leur somme dans z3 et l'affiche sous la forme a + ib.

Exercice 6

Ecrivez un programme nommé BIBLIO permettant de représenter les informations d'une référence bibliographique : le titre du livre, le nom de l'auteur, le nom de l'éditeur, l'année de publication et le nombre de pages.
Exemple de livre :
La chartreuse de Parme de Stendhal édité par Gallimard en 1987 et qui compte 683 pages.
Ce programme permet :
– La saisie des références (au minimum 2 et au maximum 150) dans un tableau,
– La saisie d'une année
– La recherche et l'affichage de tous les livres qui ont été publiés cette année.

Exercice 7

D'après la partie lecture, pour quelle raison préférait-on utiliser des fichiers carton plutôt que des supports magnétiques ?

- ☐ Parce que les supports magnétiques étaient moins fiables
- ☐ Parce que les supports magnétiques étaient plus chers
- ☐ Parce que les supports magnétiques étaient moins pratiques.

Suite de la partie lecture - site : info.sio2.be/infobase/18/2.php

Exercice 8

Répondez par V (Vrai) ou F (Faux) à chacune des questions suivantes :

a. L'accès séquentiel à un fichier permet :

- ☐ la lecture d'un élément précis
- ☐ la lecture un à un des éléments
- ☐ la lecture à partir du dernier jusqu'au premier élément.

b. L'accès direct à un fichier permet :

- ☐ la lecture d'un seul élément précis
- ☐ la lecture de deux éléments en même temps
- ☐ la lecture de plusieurs éléments en même temps.

c. Un type fichier peut contenir en même temps :

- ☐ plusieurs champs de types différents
- ☐ uniquement 2 champs de types différents
- ☐ uniquement des champs de même type.

Exercice 9

Ecrivez un programme nommé **POSITIFS** qui :

– Saisit et enregistre au fur et à mesure des nombres entiers dans un fichier dont le nom physique est "C:\NOMBRE1.FCH". Le nombre d'entiers ne doit pas dépasser 100.
– Lit et transfère les nombres de ce fichier dans un tableau unidimensionnel T1.
– Transfère dans un deuxième fichier dont le nom physique est "C:\NOMBRE2.FCH", tous les nombres positifs de T1.

Exercice 10

Créez un fichier texte appelé **exercice.txt** et y stockez des valeurs de type chaîne de caractères (string) récupérées depuis le clavier jusqu'à la lecture de la chaîne **FIN**(peut importe la casse, 'fin' ou 'fIn'). La chaîne de caractères, 'FIN' ne sera pas stockée dans le fichier.
Effacez l'écran, relisez et affichez le contenu du fichier.

Exercice 11

Ecrivez une fonction nommée Existe, qui permet de détecter la présence ou non d'un fichier de données sur une unité d'enregistrement.
– Utilisez les directives de compilation pour gérer le contrôle des entrées / sorties.

Exercice 12

Le fichier "pays.txt" a déjà été créé par un programme et existe sur la partition "C", dans le dossier "Pays". Nous voulons maintenant pouvoir lire une ligne spécifique de ce fichier, correspondant à un pays particulier.
Nous supposons que la longueur maximale d'une ligne du fichier est de 60 caractères.
Ecrivez un programme qui :
– Ouvre en lecture le fichier "pays.txt"
– Demande à entrer au clavier un nom P de pays à chercher.
– Cherche dans le fichier, la première ligne contenant le pays sélectionné.
– Dans le cas ou le pays n'est pas trouvé, le message "Le pays, P, n'existe pas dans le fichier", sinon le texte suivant sera affiché à l'écran :
"Le pays, P, existe dans la ligne : … (la ligne en question) …

Exercice 13

Créer sur la partition de votre choix, dans un dossier nommé "Classes", un fichier a accès direct dont le nom sera saisit au clavier et qui correspond au nom d'une classe. L'extension de ce fichier sera dans tous les cas "FCH".
Ce fichier recevra les noms, prénoms et moyennes de N élèves de la classe en question.

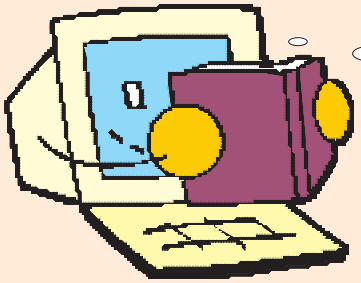
Ecrivez un programme qui permet :
– d'écrire ou de lire le fichier d'une classe suivant un premier menu comportant les choix suivants : Lecture, Ecriture et Quitter.
– d'accéder à un élève particulier ou à toute la classe par l'intermédiaire d'un deuxième menu comportant les choix suivants : Classe, Elève et Retour au premier menu.

NB : L'accès à un élève se fait à l'aide de son numéro d'ordre.

Exercice 14

Pour mieux gérer et organiser ses films, le responsable d'un vidéo club décide d'informatiser sa vidéothèque par la création d'un fichier faisant objet d'une base de donnée de ses films.
Ecrivez un programme permettant les actions offertes par les menus suivants :

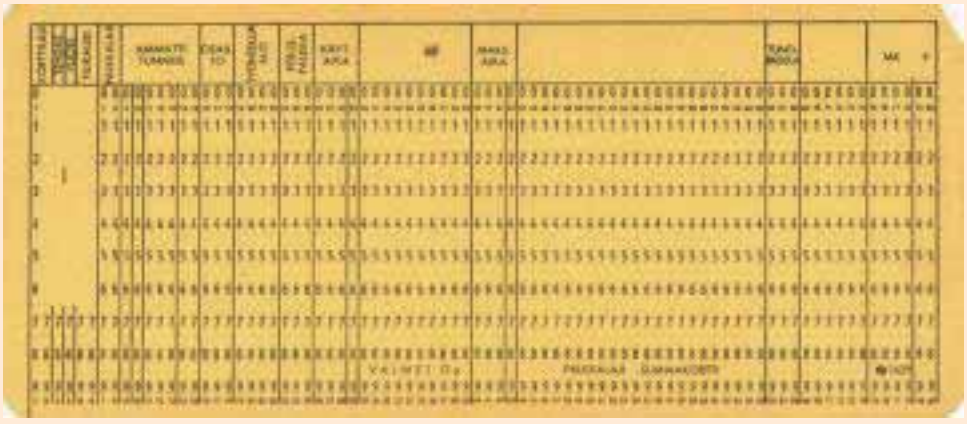
Menu Fichiers	**Menu Edition**	**Menu Quitter**
– Créer un nouveau fichier	– Ajouter un film	– Sortir
– Ouvrir un fichier existant	– Modifier un film	
– Supprimer un fichier	– Chercher un film	
– Fermer un fichier	– Supprimer un film	



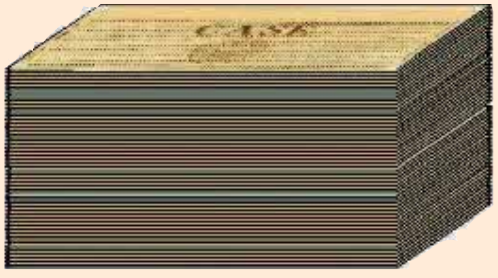
Lecture

Notion de fichier

Dans les années 1960, les supports magnétiques (disques durs, disquettes, ...) étaient encore très chers. D'autres méthodes ont été utilisées pour stocker les informations fournies aux ordinateurs. Nous allons maintenant étudier cette notion de " fichier ".
Dans les premiers temps de l'informatique, les informations étaient enregistrées sur des cartes perforées.



De telles cartes comportaient 80 colonnes dans lesquelles on pouvait coder les valeurs 0 et 1 en perforant la carte ou en la laissant intacte.
Sur une carte perforée, on pouvait donc enregistrer 80 octets.



Pour enregistrer des quantités plus grandes d'informations, il suffit alors d'utiliser un grand nombre de ces cartes. On obtient alors un paquet de fiches plus ou moins épais qui constitue un "fichier informatique".
Ces fichiers contenaient donc les données à faire traiter par l'ordinateur ou les programmes à faire exécuter.

Quand les supports magnétiques ont été utilisés plus fréquemment, on a continué d'appeler " fichier informatique " tous ces paquets d'informations enregistrées.

Un fichier informatique est donc n'importe quel ensemble d'informations (données à traiter, résultats du traitement, programme) enregistré sur un support quelconque.

D'après le site : info.sio2.be/infobase/18/2.php

Chapitre 2

La récursivité

Objectifs

- Définir la récursivité
- Reconnaître et identifier des algorithmes récursifs
- Résoudre des problèmes faisant appel à la récursivité

Plan du chapitre

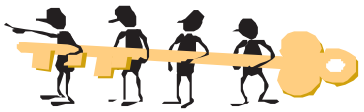
Chapitre 2

La récursivité

I. Introduction

Activité 1

Question 1 : Proposez une analyse, puis déduisez l'algorithme d'une fonction permettant de calculer la factorielle d'un entier n.
Comment appelle t-on le procédé utilisé dans ce traitement ?



Analyse de la fonction Factorielle

Résultat : Factorielle

Traitement : La factorielle est obtenue après :

- Une initialisation (F ← 1),
- Ensuite, une *structure itérative complète* (Pour i de 2 à n Faire) contenant l'instruction :
  F ← F * i

Algorithme de la fonction Factorielle

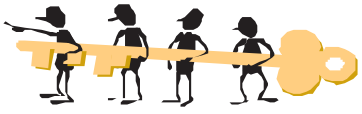
```
0) Fonction Factorielle (n : Entier non signé) : Entier Long
1)  F ← 1
2) Pour i de 2 à n Faire
        F← F * i
   Fin Pour
3) Factorielle ← F
4) Fin Factorielle
```

Tableau de déclaration des objets :

Objets	Type / Nature	Rôle
F	Entier Long	Contenant la factorielle de n
i	Entier non signé (Mot)	Compteur

Il s'agit d'un procédé itératif pour calculer n!

Question 2 : Pouvez vous proposer un procédé différent pour trouver la valeur de la factorielle de n ?



Et si pour résoudre un problème, on réduisait sa taille ?
Dans certains cas, la solution d'un problème peut se ramener à résoudre le même problème de taille plus réduite, qui lui-même pourra se ramener à résoudre le même problème de taille encore plus petite ... jusqu'à aboutir à la résolution d'un problème élémentaire.

Explications :

? Donnez la formule mathématique de la factorielle de n

▶ { n ! = n * (n-1) * (n-2) * (n-3) * … * 1
et 0 ! = 1

ou encore

{ factorielle (n) = n * (n-1) * (n-2) * (n-3) * … * 1
et factorielle (0) = 1 } *Formule 1*

? Remplacez la partie colorée de la Formule1 donnée ci-dessus par une expression équivalente utilisant la fonction factorielle

▶ { factorielle (n) = n * factorielle (n-1)
et factorielle (0) = 1 } *Formule 2*

Constatations :

❖ Pour définir factorielle (n) au niveau de la Formule 1, nous avons utilisé un traitement itératif.

❖ Pour définir factorielle (n) au niveau de la Formule 2, nous avons utilisé un autre procédé répétitif : nous avons appelé la même fonction sur d'autres données plus simples (factorielle (n-1)). Ce procédé s'appelle ***Traitement Récursif***.

Exemple :

Pour calculer factorielle (4) ou 4!, nous procédons ainsi :

1ère étape : 4 ≠ 0 alors
Factorielle (4) = 4 * Factorielle (4-1)
= 4 * Factorielle (3)

2ème étape : 3 ≠ 0 alors
Factorielle (3) = 3 * Factorielle (3-1)
= 3 * Factorielle (2)
d'où **Factorielle (4) = 4 * 3 * Factorielle (2)**

3ème étape : 2 ≠ 0 alors
Factorielle (2) = 2 * Factorielle (2-1)
= 2 * Factorielle (1)
d'où **Factorielle (4) = 4 * 3 * 2 * Factorielle (1)**

4ème étape : 1 ≠ 0 alors

Factorielle (1) = 1 * Factorielle (1-1)

= 1 * Factorielle (0)

d'où **Factorielle (4) = 4 * 3 * 2 * 1 * Factorielle (0)**

5ème étape : La valeur 0 est rejointe, c'est le **cas particulier** qui provoquera l'arrêt des étapes du calcul de la factorielle, alors

Factorielle (0) = 1

D'où **Factorielle (4) = 4 * 3 * 2 * 1 * 1**

Le résultat final = 24

Conclusion :

Voici la solution **récursive** de la fonction Factorielle :

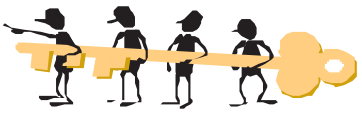
0) Fonction Factorielle (n : Entier non signé) : Entier Long
1) Si (n = 0) Alors Factorielle ← 1
   Sinon Factorielle ← n * Factorielle (n-1)
   Fin Si
2) Fin Factorielle

Constatations :

La solution récursive comporte :

- ❖ L'appel récursif en changeant la valeur du paramètre. Dans l'exemple de la fonction Factorielle, le paramètre est décrémenté de 1 à chaque appel, (**Factorielle ← n * Factorielle (n-1)**)
- ❖ La condition d'arrêt de l'appel récursif. En effet, nous arrêtons le traitement quand n devient égal à 0 (**Si n = 0 Alors Factorielle ← 1**)

Question 3 : Traduisez et testez la solution du problème permettant d'afficher la factorielle d'un entier n donné, en utilisant la version récursive de la fonction Factorielle. Enregistrez votre programme sous le nom **Fact_rec**.

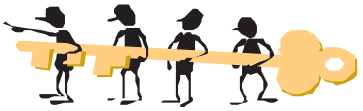


```
PROGRAM Fact_rec;
Uses Crt;
Var n : Word;

Function Factorielle (m : Word) : LongInt;
Begin
      If (m = 0) Then Factorielle := 1
      Else Factorielle := m * Factorielle (m - 1);
End;
```

```
{Programme Principal}
BEGIN
      Write ('n = '); ReadLn (n);
      Write ('n! = ', Factorielle (n));
END.
```

Activité 2

Question 1 : Proposez une analyse, puis déduisez l'algorithme itératif d'une fonction permettant de vérifier si une chaîne de caractères CH est un palindrome.
Nous appellons "palindrome" un mot ou une phrase qui se lit de la même façon dans les deux sens (de gauche à droite et de droite à gauche).
Exemples : radar, rotor, été.



Analyse de la fonction Palindrome

Résultat : Palindrome,
Traitement : Le résultat booléen de la fonction Palindrome est obtenu après :

- [i ← 1],
- Ensuite, une structure itérative à condition d'arrêt (Répéter … Jusqu'à) vérifiant les actions suivantes :
  - À chaque itération, nous comparons deux à deux les caractères symétriques de CH relatifs à une position i (Verif ← (ch[i] = ch[Long (ch) – i + 1])).
  - Nous arrêtons la répétition à la première différence (Verif = Faux) **ou** lorsque nous atteignons les caractères du milieu (i > Long (ch) Div 2).

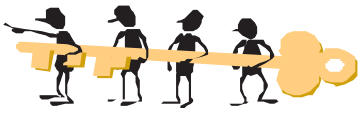
Algorithme itératif de la fonction Palindrome

```
0) Fonction Palindrome (ch : Chaîne) : Booléen
1) i ← 1
   Répéter
                Verif ← (ch[i] = ch[Long (ch) – i + 1])
                i ← i + 1
   Jusqu'à (i > Long (ch) Div 2) Ou (Verif = Faux)
2) Palindrome ← Verif
3) Fin Palindrome
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
i	Octet	Compteur
Verif	Booléen	Tester l'état de ch (Palindrome ou non)

Question 2 : Dégagez une relation récursive permettant de vérifier si une chaîne de caractères CH est un palindrome.



➢ Essayons de dégager cette relation à partir d'un premier exemple (ch = "radar", de longueur = 5) :

1ère étape : Long ("radar") = 5 ; comme **5 n'est pas inférieur à 2**,
et comme **les caractères extrêmes de la chaîne "radar" sont identiques = "r"**
alors, **Palindrome ("radar") = Palindrome ("ada")**

2ème étape : Long ("ada") = 3 ; comme **3 n'est pas inférieur à 2**,
et comme **les caractères extrêmes de la chaîne "ada" sont identiques = "a"**
alors, **Palindrome ("ada") = Palindrome ("d")**

3ème étape : Long ("d") = 1 ; comme **1 est inférieur à 2, c'est le cas *particulier* qui provoquera l'arrêt des étapes de la vérification si ch est palindrome ou non,** alors **Palindrome = Vrai**

Dans cet exemple, ch est un palindrome car on a obtenu une chaîne formée d'un seul caractère ("d") sans rencontrer de i vérifiant ch[i] ≠ ch[Long (ch)-i+1].

➢ Réessayons avec l'exemple suivant (ch = "exemple", de longueur = 7) :

1ère étape : Long ("exemple") = 7 ; comme **7 n'est pas inférieur à 2,**
et comme **les caractères extrêmes de la chaîne "exemple" sont identiques = "e"**
alors, **Palindrome ("exemple") = Palindrome ("xempl")**

2ème étape : Long ("xempl") = 5 ; comme **5 n'est pas inférieur à 2,**
et comme **les caractères extrêmes de la chaîne "xempl" sont différents ("x"≠ "l")**
alors, **Palindrome = Faux**

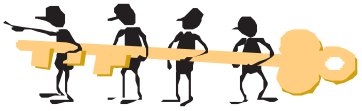
Dans cet exemple, ch n'est pas un palindrome car ch[2] ≠ ch[7-2+1] (puisque "x" ≠ "l")

Remarques : Un mot d'un seul caractère est un palindrome, sinon il l'est dès que ses caractères extrêmes sont les mêmes et le mot restant est aussi un palindrome.

➢ d'où, la relation **récursive** suivante :

Si Longueur (ch) < 2 alors ch est un palindrome
Sinon si ch[1] = ch[Long (ch)] alors Palindrome (ch) = Palindrome (ch sans ses premier et dernier caractères)
Sinon ch n'est pas un palindrome.

Question 3 : Déduisez l'algorithme récursif de la fonction Palindrome.



Algorithme :

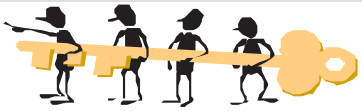
```
0) Fonction Palindrome (ch : Chaîne) : Booléen
1) Si Long (ch) < 2 Alors Palindrome ←Vrai
   Sinon Si ch[1] = ch[Long (ch)]
        Alors Palindrome ← Palindrome (Sous_chaîne (ch, 2, Long (ch) - 2))
        Sinon Palindrome ← Faux
   FinSi
2) Fin Palindrome
```

Conclusion :

La solution récursive comporte :

- Un appel récursif en changeant la valeur du paramètre. Dans l'exemple de la fonction Palindrome, le paramètre ch est démuni de son premier et de son dernier caractère s'ils sont identiques, (**Palindrome ← Palindrome (Sous_chaîne (ch, 2, Long (ch) - 2))**)
- Une 1ère condition d'arrêt de l'appel récursif. En effet, nous arrêtons le traitement lorsque les caractères extrêmes du paramètre sont différents (**Palindrome ← Faux**)
- Une 2ème condition d'arrêt de l'appel récursif, lorsque le paramètre est composé d'un seul caractère (**Si Long (ch) < 2 Alors Palindrome ← Vrai**)

Question 4 : Traduisez et testez la solution du problème permettant d'afficher un message indiquant si une chaîne de caractères donnée est un palindrome ou non, en utilisant la version récursive de la fonction Palindrome. Enregistrez votre programme sous le nom **ch_Pal**.



```
PROGRAM palindrome_rec;
Uses Crt;
Var ch : String;

Function Palindrome (ch : String) : Boolean;
Begin
      If Length(ch) < 2 Then Palindrome := True
      Else If (ch[1] = ch[length(ch)])
           Then Palindrome := Palindrome (Copy (ch, 2, Length (ch) - 2))
      Else Palindrome := False;
End;
```

```
{Programme Principal}
BEGIN
      Write ('Donner ch = '); ReadLn (ch);
      If Palindrome (ch) Then WriteLn (ch, ' est un palindrome.')
      Else WriteLn (ch, ' n''est pas un palindrome.');
END.
```

II. Définitions et principes

II.1 Définitions

La **récursivité** est une méthode algorithmique qui consiste à appeler un sous-programme dans son propre corps. Cette méthode permet de résoudre des problèmes où l'on doit itérer un nombre de fois dépendamment du nombre de données.
Un sous-programme récursif est un module qui s'"appelle" lui-même. A chaque appel, il y a mémorisation d'une valeur différente d'un même paramètre formel.

Le principal avantage d'un algorithme récursif est qu'il est généralement plus simple, plus lisible et naturel à comprendre. Par contre, le principal problème est la définition du ou des cas d'arrêt. Il est important de s'assurer que le programme se termine correctement dans tous les cas.

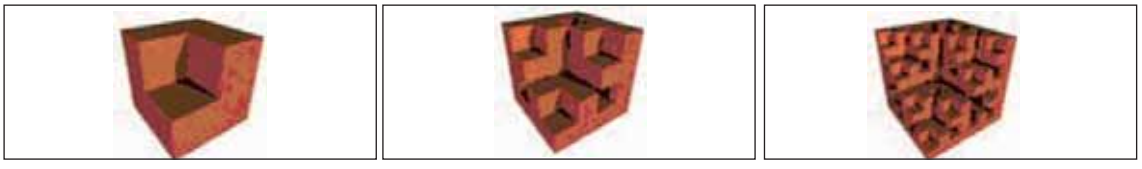
La limite technique de la récursivité est la mémoire stockant les données intermédiaires des sous-programmes, que nous appelons la "pile". Cette dernière déborde lorsque le nombre des appels devient important.

Remarques :

1) Les fonctions **récursives** sont appelées depuis leur propre corps, soit directement, soit indirectement à travers une ou plusieurs fonctions relais. Par exemple, si la fonction P appelle directement P, nous disons que la récursivité est directe. Si P appelle une fonction P1, qui appelle une fonction P2, ..., qui appelle une fonction Pn et qui enfin appelle P, nous disons qu'il s'agit d'une récursivité indirecte.

2) La récursivité est dite **croisée** lorsque plusieurs procédures ou fonctions s'appellent mutuellement.

II.2 Principes



Un programme récursif doit :

➢ traiter un ou plusieurs cas particuliers représentés par une ou plusieurs conditions d'arrêt, sinon il tournera indéfiniment.

➢ traiter un ou plusieurs cas généraux représentés par des appels récursifs. Chaque fois qu'un programme récursif est appelé (par lui-même), un ou plusieurs des arguments qui lui sont transmis doivent converger et arriver à la condition d'arrêt.

C'est à dire que la complexité du problème doit être réduite à chaque nouvel appel récursif.

Remarque :

L'itération et la récursivité, correspondent à deux modes de pensée différents. Le principe de passage du mode récursif au mode itératif s'appelle : la dérecursification.

III. Applications

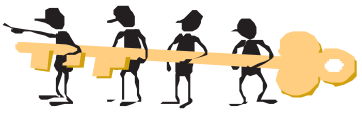
1. Application 1

Nous proposons d'afficher la somme des n premiers entiers.

a. Proposez une analyse au problème en utilisant un procédé **récursif**,

b. Déduisez les algorithmes correspondants,

c. Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Somme.**



a) Analyse du problème

Résultat : Ecrire ("La somme des ", n, " premiers entiers = ", FN Somme (n))

Traitement : *Somme* est une fonction récursive permettant de calculer la somme des n premiers entiers. Elle est appelée au niveau du programme principal dans un contexte d'affichage.

Données : un entier n obtenu grâce à la procédure Saisir

b) Algorithme du programme principal et codification des objets

0) Début Somme_N
1) Proc Saisir (n)
2) Ecrire (" La somme des ", n, " premiers entiers = ", FN Somme (n))
3) Fin Somme_N

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n	Entier non signé	Nombre à sommer
Saisir	Procédure	Permet de saisir n
S	Fonction	Permet de sommer les n premiers entiers

c) Analyse de la fonction Somme

Résultat : Somme
Traitement : La somme des m premiers entiers est obtenue comme suit :
Un cas particulier : **Si** m = 0 **alors** Somme ← 0
Un cas général : **Sinon** Somme ← m + Somme (m-1).

d) Algorithme récursif de la fonction Somme

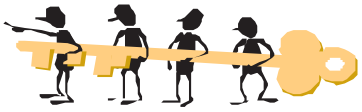
0) Fonction Somme (m : Entier non signé) : Entier Long
1) **Si** (m = 0) **Alors** Somme ← 0
Sinon Somme ← m + Somme (m - 1)
Finsi
2) Fin Somme

e) Traduction en Pascal de la fonction Somme

```
Function Somme (m : Word) : LongInt;
Begin
      If m = 0 Then Somme :=  0
      Else Somme := m + Somme (m - 1);
End;
```

2. Application 2

Nous proposons de calculer et d'afficher la valeur de a^n avec a un réel donné et n un entier donné.
a. Proposez une analyse au problème en utilisant un procédé récursif,
b. Déduisez les algorithmes correspondants,
c. Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Exposant**.



a) Analyse du problème

Résultat : Ecrire (a, " à la puissance ", n, " = ", FN **Puissance (a, n)**)

Traitement : *Puissance* est une fonction récursive permettant de calculer la valeur de a^n.
Données : Un réel a et un entier n.

b) Algorithme du programme principal et codification des objets

0) Début Puiss
1) Ecrire ("Donner un entier : ") , Lire (a)
2) Ecrire ("Donner la puissance : ") , Lire (n)
3) Ecrire (a, " à la puissance ", n, " = ", FN **Puissance (a, n)**)
4) Fin Puiss

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
a	Réel	Donnée
n	Entier	L'exposant
Puissance	Fonction	Permet de déterminer la valeur a^n

c) Analyse de la fonction Puissance

Résultat : Puissance
Traitement : Le résultat de la fonction Puissance est obtenu comme suit :
Un cas particulier : **Si** n = 0 **alors** Puissance ← 1
Un 1er cas général : **Si** n < 0 **alors** Puissance ← 1 / Puissance (a, -n)
Un 2ème cas général : **Sinon** Puissance ← a * Puissance (a, n-1)

d) Algorithme récursif de la fonction Puissance

0) Fonction Puissance (a : Réel ; n : Entier) : Réel
1) **Si** n = 0 **Alors** Puissance ← 1
Sinon Si n < 0 **Alors** Puissance ← 1 / Puissance (a, -n)
Sinon Puissance ← a * Puissance (a, n-1)
Finsi
2) Fin Puissance

e) Traduction en Pascal de la fonction Puissance

```
Function Puissance (a : Real; n : Integer) : Real;
Begin
      If (n = 0) Then Puissance := 1
      Else If (n < 0) Then Puissance := 1 / Puissance (a, -n)
      Else Puissance := a * Puissance (a, n - 1);
End;
```

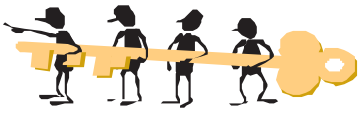
3. Application 3

Nous proposons de trier un tableau T de n entiers dans l'ordre décroissant et en utilisant la méthode par sélection.

a. Proposez une analyse au problème en utilisant un procédé **récursif**,
b. Déduisez les algorithmes correspondants,
c. Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Selection**.

Chapitre

2



a) Analyse du problème

Résultat : Affichage du tableau T trié ; une tâche réalisée par la procédure *Afficher_T*
Traitement : Le tri sera effectué par la procédure recursive Tri_Sel_Rec
Données :

- un tableau T à remplir par n entiers grâce à la procédure *Remplir*
- n : le nombre d'éléments du tableau, obtenu grâce à la procédure Saisir,

b) Algorithme du programme principal et codification des objets

0) Début Selection
1) Proc Saisir (n)
2) Proc Remplir (n, T)
3) Proc Tri_Sel_Rec (1, n, T)
4) Proc Afficher_T (n, T)
5) Fin Selection

Tableau de codification des nouveaux types

Type
Tab = Tableau de n entiers

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n	Entier non signé	Nombre d'éléments de T
T	Tab	Le tableau à trier
Saisir	Procédure	Permet de saisir le nombre des éléments de T
Remplir	Procédure	Permet de remplir le tableau T
Tri_Sel_Rec	Procédure	Permet de trier le tableau T
Afficher_T	Procédure	Permet d'afficher le tableau T

c) Analyse de la procédure Tri_Sel_Rec

Résultat : un tableau T trié dans l'ordre décroissant
Traitement :

- T trié est obtenu suite à un appel récursif conditionnel de la procédure *Tri_Sel*_Rec sur T omis de son 1er élément (**Tri_Sel_Rec (deb+1, fin, T)**). Cet appel est exécuté **si** l'indice du premier élément de la partie à trier est différent de celui du dernier **(deb+1 ≠ fin).**
- Chaque appel récursif prévoit :
  - d'abord, une recherche de l'indice de la valeur maximale de T en faisant appel à la fonction *Max* (**Maximum ← Max (deb, fin, T)**).
  - ensuite, **Si** l'indice de la valeur du maximum est différent de deb, nous procédons à une permutation réalisée par la procédure *Permuter* (**Permuter (deb, maximum, T)**).

NB : La fonction *Max* et la procédure *Permuter* ont été l'objet d'un apprentissage au niveau de la 3ème année.

d) Algorithme de la procédure Tri_Sel_Rec

0) Procedure Tri_Sel_Rec (deb, fin : Entier ; Var T : Tab)
1) Maximum ← FN Max (deb, fin, T)
   Si Maximum ≠ deb **Alors** Proc Permuter (T, deb, maximum)
   FinSi
2) **Si** deb+1 ≠ fin **Alors** Proc Tri_Sel_Rec (deb+1, fin, T)
   Finsi
3) Fin Tri_Sel_Rec

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
Maximum	Entier	Représente l'indice de la valeur maximale de la partie du tableau qui reste à trier
Max	Fonction	Retourne l'indice du maximum d'une partie d'un tableau
Permuter	Procédure	Permet de permuter deux entiers

e) Traduction en Pascal de la procédure Tri_Sel_Rec

```
Procedure Tri_Sel_Rec (deb, fin : Integer; Var T : Tab);
Var Maximum : Integer;

Function Max (deb1, fin1 : Integer; T : Tab) : Integer;
Var i, M : Integer;
Begin
      M := deb1;
      For i := deb1+1 To fin1 Do
            If (T[i] > T[M]) Then M := i;
      Max := M
End;

Procedure Permuter (Var x, y : Integer);
Var aux : Integer;
Begin
      aux := x;
      x := y;
      y := aux;
End;

Begin
      Maximum := Max (deb, fin, T);
      If (Maximum <> deb) Then Permuter (T[deb], T[Maximum]);
      If (deb+1 <> fin) Then Tri_Sel_Rec (deb+1, fin, T);
End;
```

4. Application 4

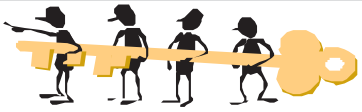
Nous proposons de vérifier l'existence d'un entier m dans un tableau T contenant n entiers, en utilisant la technique de la recherche dichotomique.

NB : Nous supposons que le tableau est déjà trié dans l'ordre croissant.

a. Proposez une analyse au problème en utilisant un procédé récursif,

b. Déduisez les algorithmes correspondants,

c. Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Dichotom**.



a) Analyse du problème

Résultat : Si FN **Dichotomique (1, n, m, T) Alors** Ecrire (m, " existe dans le tableau T.")
Sinon Ecrire (m, " n'existe pas dans le tableau T.")
Finsi

Traitement : *Dichotomique* est une fonction récursive booléenne, appelée au niveau du programme principal.

Données :

- un entier m : objet de la recherche
- un tableau T à remplir par n entiers grâce à la procédure *Remplir*,
- un entier n représentant le nombre d'éléments du tableau T et qui sera la tâche de la procédure *Saisir*

b) Algorithme du programme principal et codification des objets

0) Début Recherche
1) Proc Saisir (n)
2) Proc Remplir (n, T)
3) Ecrire ("Donner un entier : ") , Lire (m)
4) **Si** FN Dichotomique (1, n, m, T) **Alors** Ecrire (m, " existe dans le tableau T.")
Sinon Ecrire (m, " n'existe pas dans le tableau T.")
Finsi
5) Fin Recherche

Tableau de codification des nouveaux types

Type
Tab = Tableau de n entiers

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n	Entier non signé	Nombre d'entiers dans le tableau T
m	Entier	Objet de la recherche
T	Tab	Tableau d'entiers
Saisir	Procédure	Permet de saisir le nombre d'entiers dans le tableau
Remplir	Procédure	Permet de remplir le tableau trié
Dichotomique	Fonction	Permet de vérifier l'existence ou non de m dans T

c) Analyse de la fonction Dichotomique

Résultat : Dichotomique
Traitement :

- Le résultat booléen de la fonction Dichotomique est obtenu suite à l'instruction conditionnelle généralisée suivante :
  - Cas 1 : **Si** m = T[milieu] **alors** m existe dans T
  - Cas 2 : **Si** m < T[milieu] et deb < milieu, **alors** nous vérifions l'existence de m dans la première moitié du tableau (Dichotomique ← FN Dichotomique (deb, milieu-1, m, T))
  - Cas 3 : **Si** m > T[milieu] et fin > milieu, **alors** nous vérifions l'existence de m dans la deuxième moitié du tableau (Dichotomique ← FN Dichotomique (milieu+1, fin, m, T))
  - Cas 4 : **Sinon** m n'existe pas dans T
- milieu ← (deb + fin) Div 2

d) Algorithme récursif de la fonction Dichotomique

```
0) Fonction Dichotomique (deb, fin, m : Entier ; T : Tab ) : Booléen
1) milieu ← (deb + fin) Div 2
2) Si m = T[milieu] Alors Dichotomique ← Vrai
   Sinon Si (m < T[milieu]) Et (deb < milieu)
   Alors Dichotomique ← Dichotomique (deb, milieu-1, m, T)
   Sinon Si (m > T[milieu]) Et (fin > milieu)
   Alors Dichotomique ← Dichotomique (milieu+1, fin, m, T)
   Sinon Dichotomique ← Faux
   Finsi
3) Fin Dichotomique
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
milieu	Entier	Indice de l'élément du milieu de la partie dans laquelle s'effectue la recherche.

e) Traduction en Pascal de la fonction Dichotomique

```
Function Dichotomique (deb, fin, m : Integer; T : Tab) : Boolean;
Var milieu : Integer;
Begin
      milieu := (deb + fin) Div 2;
      If (m = T[milieu]) Then Dichotomique := True
      Else If (m < T[milieu] ) And (deb < milieu)
      Then Dichotomique := Dichotomique (deb, milieu-1, m, T)
      Else If (m > T[milieu] ) And (fin > milieu)
      Then Dichotomique := Dichotomique (milieu+1, fin, m, T)
      Else Dichotomique := False
End;
```

Retenons

– Tout objet est dit **récursif** s'il se définit à partir de lui-même. Ainsi, un sous-programme est dit **récursif** s'il comporte, dans son corps, au moins un appel à lui-même.

– La réalisation d'un module récursif revient toujours à une analyse par cas, dans laquelle nous distinguons les **cas généraux** des **cas particuliers**.

Exercices

Exercice 1

Mettez la lettre V (Vrai) dans la case qui correspond à chaque proposition si vous jugez qu'elle est vraie sinon mettez la lettre F (Faux).

a. Une procédure

est obligatoirement récursive ☐ n'est jamais récursive ☐ peut être récursive ☐

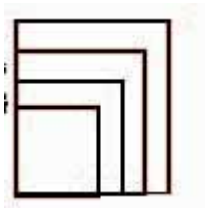
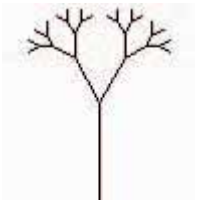
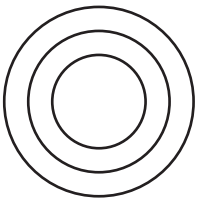
b. Dans un module récursif, nous utilisons obligatoirement une structure :

conditionnelle ☐ itérative à condition d'arrêt ☐ itérative complète ☐

c. Pour appliquer la récursivité, un ordinateur utilise :

une pile ☐ un fichier ☐ un tableau ☐

d. Parmi les dessins suivants, lesquels pouvant être réalisés grâce à un module récursif :

 ☐  ☐  ☐

Exercice 2

Que fait la fonction suivante :

```
0) Fonction inconnue (st : chaîne) : chaîne
1) Si st = "" Alors inconnue ← ""
   Sinon inconnue ← st[Long (st)] + FN inconnue (Sous_chaîne (st, 1, Long (st) - 1))
   Finsi
2) Fin inconnue
```

Exercice 3

Trouvez l'erreur dans les fonctions récursives suivantes écrites en Pascal :

a. La fonction Pair teste la parité de N et doit retourner True si N est pair

```
Function Pair (N : Integer) : Boolean ;
Begin
      If (N = 0) Then Pair := True
      Else If (N = 1) Then  Pair := False
      Else Pair := Pair (N - 1);
End;
```

b. La fonction Impair teste la parité de N et doit retourner True si N est impair

```
Function Impair (N : Integer) : Boolean;
      Function Pair (N : Integer) : Boolean ;
      Begin
            If (N = 0) then Pair :=  True
            Else Pair :=  Impair (N - 1);
      End;
Begin
      If (N = 1) Then Impair :=  True
      Else Impair :=  Pair (N - 1);
End;
```

Exercice 4

Proposez une analyse et déduisez l'algorithme d'une fonction récursive appelée *Pair_C* qui teste si un mot contient un nombre pair ou non d'un caractère C donné.

Exercice 5

Proposez une analyse et déduisez l'algorithme d'une fonction récursive qui, étant donné un entier M, détermine la valeur la plus proche de M dans un tableau T de N entiers.

Exercice 6

Proposez une analyse et déduisez l'algorithme d'une fonction récursive appelée Contigus qui détermine si une chaîne comporte deux caractères *contigus* identiques :

Exemples :

Contigus("passer") a pour résultat vrai
Contigus("fonction") a pour résultat faux
Contigus("b") a pour résultat faux.

Pour chacun des exercices suivants :

a. Proposez une analyse modulaire au problème en utilisant un procédé récursif dans votre solution,

b. Déduisez les algorithmes correspondants.

Exercice 7

Nous proposons d'inverser une chaîne de caractères.
Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom Inverse.

Exercice 8

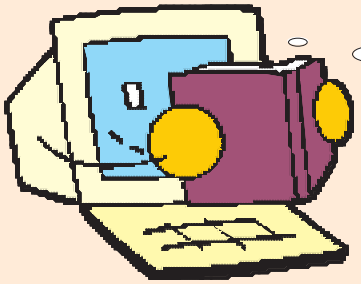
Nous proposons de vérifier l'existence d'un entier m dans un tableau T contenant n entiers, en utilisant la technique de la recherche séquentielle.
Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Rech_Seq**.

Exercice 9

Nous proposons d'afficher le PPCM de deux entiers donnés.
Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **PPCM**.

Exercice 10

Nous proposons de décomposer un entier N donné en produit de facteurs premiers.
Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Fact_Prem**.



Lecture

La récursivité est un concept naturel

On affirme souvent que la notion de fonction récursive est extrêmement difficile à appréhender par les étudiants, car dit-on, «définir une notion en s'appuyant sur la notion que l'on est en train de définir est complètement incompréhensible». On ajoute souvent que c'est l'une des difficultés majeures de l'apprentissage de la programmation fonctionnelle, car la récursivité y joue un rôle bien plus important que dans les autres paradigmes de programmation. Je conteste ce point de vue, car je suis persuadé que tout le monde a déjà rencontré cette vieille notion (elle apparaît déjà dans les écrits d'Archimède sur la quadrature de la parabole et de la lunule). C'est pourquoi je me suis mis à traquer la récursivité dans la vie courante, et ai appris à la reconnaître un peu partout! Voici ce que ça donne et ce n'est pas triste...

En philosophie, *chez les grecs au 5ème siècle avant JC :*

Connaîs-toi toi-même.

Médecin, soigne-toi toi-même.

Socrate: Je sais que je ne sais rien.

En logique, *toujours chez les grecs anciens :*

Le paradoxe du menteur qui dit : Je mens

En littérature,

Chez La Bruyère : Vouloir oublier quelqu'un, c'est y penser.

En blague,

D'après Guy Bedos : Je perds la mémoire, c'est dur, mais ça offre au moins trois avantages :

- *Premièrement: je n'ai plus de mauvais souvenirs.*
- *Deuxièmement: je vois sans arrêt des têtes nouvelles.*
- *Troisièmement: je n'ai plus de mauvais souvenirs.*

En langue naturelle,

Un dictionnaire est un livre qui explique les mots avec des définitions qui sont faites de phrases qui sont des suites de mots.

En proverbe,

On n'est jamais mieux servi que par soi-même!

En croyances absurdes,

Ne soyez pas superstitieux : ça porte malheur!

En titre, *Raymond Smullyan a écrit un livre sur la logique intitulé :*

Titre : ***Quel est le titre de ce livre ?***

En biologie,

Oeuf et poule sont définis mutuellement récursivement : la poule naît d'un oeuf (qui est produit par une poule (qui naît d'un oeuf (qui est produit par une poule (qui naît d'un oeuf (qui est produit ...)))))

http : // pauillac.inria.fr

La récursivité pas à pas

Une définition... récursive !
"Une procédure récursive est une procédure récursive."

Une définition plus "sérieuse"
Une procédure récursive est une procédure qui s'appelle elle-même.

La récursivité est un domaine très intéressant de l'informatique, un peu abstrait, mais très élégant; elle permet de résoudre certains problèmes d'une manière très rapide, alors que si on devait les résoudre de manière itérative, il nous faudrait beaucoup plus de temps et de structures de données intermédiaires.

La récursivité utilise toujours la pile du programme en cours.

On appelle "pile" une zone mémoire réservée à chaque programme; sa taille peut être fixée manuellement par l'utilisateur. Son rôle est de stocker les variables locales et les paramètres d'une procédure. Supposons que nous sommes dans une procédure "proc1" dans laquelle nous avons des variables locales. Ensuite nous faisons appel à une procédure "proc2"; comme le microprocesseur va commencer à exécuter "proc2" mais qu'ensuite il reviendra continuer l'exécution de "proc1", il faut bien stocker quelque part les variables de la procédure en cours "proc1"; c'est le rôle de la pile. Tout ceci est géré de façon transparente pour l'utilisateur. Dans une procédure récursive, toutes les variables locales sont stockées dans la pile, et empilées autant de fois qu'il y a d'appels récursifs. Donc la pile se remplit progressivement, et si on ne fait pas attention on arrive à un "débordement de pile". Ensuite, les variables sont désempilées.

Tout programme récursif comporte une instruction (ou un bloc d'instructions) nommée "point terminal" ou "point d'appui" ou "point d'arrêt", qui indique que le reste des instructions ne doit plus être exécuté.

http://www.chambily.com

Chapitre 3

Les algorithmes de tri

Objectifs

Acquérir des habilités de résolution de problèmes à travers l'apprentissage de quelques algorithmes de tri.

Plan du chapitre

Chapitre 3

Les algorithmes de tri

I. Introduction

Selon le dictionnaire «Le Petit Larousse», "**trier**" signifie «répartir des objets suivant certains critères». De manière plus restrictive, le terme "**tri**" en algorithmique est souvent attaché au processus de classement d'une suite d'éléments dans un ordre donné. Par exemple, trier M moyennes de type réel dans l'ordre décroissant ou trier N noms dans l'ordre alphabétique croissant. D'une façon générale, tout ensemble muni d'un ordre total peut fournir une suite d'éléments à trier.
Il existe deux catégories de tris :
Les tris internes : Méthodes destinées à des masses de données limitées, stockées dans une structure de données se trouvant dans la mémoire centrale (Exemple : tableaux).
Les tris externes : Méthodes destinées à de grandes masses de données, stockées dans des structures de données telle que les fichiers.

Vous avez utilisé dans le programme de 3ème année les méthodes de tri par sélection et à bulles.

Activité 1

1- Donnez le principe du **tri par sélection**.
2- Déduisez un algorithme de la procédure **SELECTION** (**n : entier; Var T : Tab)**
3- Avec l'aide de votre enseignant, écrivez un programme en Pascal qui permet de remplir aléatoirement un tableau de n entiers, de le trier en utilisant la méthode de tri par sélection puis de l'afficher.

La méthode de tri par sélection utilise l'algorithme formel suivant :
1- Placer dans l'élément d'indice 1 du tableau T la plus petite valeur présente dans le tableau et mettre à sa place l'ancienne valeur de T[1].
2- Placer dans l'élément d'indice 2 de T la plus petite valeur présente dans la tranche de tableau T[2..N]
3- Placer dans l'élément d'indice 3 de T la plus petite valeur présente dans la tranche de tableau T[3..N]
4- et ainsi de suite jusqu'à l'étape N-1

Activité 2

1- Donnez le principe du tri à bulles.
2- Déduisez un algorithme de la procédure **BULLES (n : Entier; Var T : Tab)**

3- Avec l'aide de votre enseignant, écrivez un programme en Pascal qui permet de remplir aléatoirement un tableau de n entiers, de le trier dans l'ordre croissant en utilisant l'algorithme de tri à bulles puis de l'afficher.

La méthode de tri à bulles utilise l'algorithme formel suivant :

L'algorithme du tri à bulles (**bubble sort** en anglais) consiste à comparer les différentes valeurs adjacentes du tableau T, et à les permuter s'ils ne sont pas dans le bon ordre. L'algorithme se déroule ainsi :

1- Les deux premiers éléments du tableau sont comparés, si le premier élément est supérieur au second, une permutation est effectuée.

2- Ensuite, sont comparées et éventuellement permutées les valeurs 2 et 3, 3 et 4 jusqu'à (n-1) et n.

3- Une fois cette étape achevée, il est certain que le dernier élément du tableau est le plus grand. L'algorithme reprend donc pour classer les (n-1) éléments qui précédent.

4- L'algorithme se termine quand il n'y a plus de permutations possibles.

Remarques :

1- Pour classer les n valeurs du tableau T, il faut, au pire des cas, effectuer l'algorithme n fois.

2- Cette méthode porte le nom de tri à bulles car, petit à petit, les plus grands éléments du tableau remontent, par le jeu des permutations, en fin du tableau.

Les différents algorithmes de tri présentent des performances différentes, en terme de temps d'exécution et en terme d'espace mémoire requis pour leur exécution.

Dans la suite de ce chapitre, vous allez apprendre à utiliser le tri par insertion, le tri Shell ainsi que des applications sur les méthodes de tri.

II. Tri par insertion

II.1 Principe

Le principe du tri par insertion est d'insérer à la n-ième itération, le n-ième élément à la bonne place dans la liste formée par les (n-1) éléments qui le précèdent.

Voici, en pseudo-code, l'algorithme du tri par insertion :

```
PROCEDURE Tri_Insertion (n : Entier; Var T : Tab)
        Pour i de 2 à n Faire
            INSERER T[i] à sa place dans T[1..(i-1)]
FIN
```

Le principe général est le suivant :

- Considérer que les (i-1) premiers éléments de la liste sont triés et placer le $i^{ème}$ élément à sa place parmi les (i) premières places.
- Répéter cette action jusqu'à atteindre la fin de la liste.

Le processus d'insertion consiste à :

- utiliser une variable intermédiaire Tmp pour conserver la valeur à insérer,
- déplacer les éléments T[i-1], T[i-2], ... vers la droite tant que leur valeur est supérieure à celle de tmp.
- affecter alors à l'emplacement laissé libre par ce décalage la valeur de Tmp.

Remarque :

Une variante du tri par insertion consiste à utiliser une recherche dichotomique. En effet, à la n-ième itération, les éléments de 1 à n-1 sont triés, et nous pouvons donc utiliser un algorithme de recherche dichotomique pour trouver la place où insérer l'élément n.

II.2 Exemple

Nous proposons d'utiliser la méthode de tri par insertion pour trier un tableau T d'entiers en ordre croissant.

Considérons le tableau T contenant les 7 éléments suivants :

T	-5	30	0	-2	42	22	9
	1	2	3	4	5	6	7

Commençons par T[2] puisque si le tableau contient un seul élément, il est déjà trié.

Etape 1

Cherchons la position d'insertion de **T[2]** dans la première partie du tableau T avec le souci de la garder triée.

Puisque 30 > -5, donc les deux premiers éléments sont triés.

T	-5	30	0	-2	42	22	9
	1	2	3	4	5	6	7

Etape 2

Nous cherchons la position d'insertion de **T[3]** dans la première partie du tableau T avec le souci de la garder triée.

- Nous affectons à Tmp la valeur de T[3] = 0
- Nous décalons T[2] à droite, car il est supérieur à 0
- Nous affectons à la dernière case décalée la valeur de Tmp

T	-5	0	30	-2	42	22	9
	1	2	3	4	5	6	7

Etape 3

Cherchons la position d'insertion de **T[4]** dans la première partie du tableau T avec le souci de la garder triée.

- Nous affectons à Tmp la valeur de T[4] = -2
- Nous décalons T[3] et T[2] à droite, car ils sont supérieurs à -2
- Nous affectons à la dernière case décalée la valeur de Tmp

T	-5	-2	0	30	42	22	9
	1	2	3	4	5	6	7

Activité

1- Sur votre cahier, continuez les autres étapes jusqu'à obtenir un tableau trié.
2- Combien faut-il d'étapes pour trier ce tableau?
3- Pouvez-vous donner une réponse dans le cas général en fonction de n (n étant le nombre d'éléments du tableau) ?

II.3 Résolution du problème

Nous utiliserons l'approche de l'analyse modulaire qui consiste à décomposer le problème en modules.

a) Analyse du programme principal

Résultat : L'affichage du tableau trié est la tâche de la procédure Afficher
Traitement :
Nous devons donc trier le tableau, ce qui sera la tâche de la procédure Trier

- *Pour pouvoir trier le tableau, nous devons tout d'abord le remplir, ce qui se fera par la procédure Remplir*
- *Pour remplir le tableau, nous devons connaître le nombre d'éléments à saisir ce qui est la tâche de la procédure Saisir.*

b) Algorithme du programme principal et codification des objets

0) Début tri_INS
1) Proc Saisir (n)
2) Proc Remplir (T, n)
3) Proc Trier (T, n)
4) Proc Afficher (T, n)
5) Fin tri_INS

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n	Entier	Taille du tableau à trier
T	Tab	Tableau à trier
Saisir	Procédure	Permet de saisir la taille du tableau à trier
Remplir	Procédure	Permet de remplir le tableau à trier
Trier	Procédure	Permet de trier le tableau en utilisant la méthode du tri par insertion
Afficher	Procédure	Permet d'afficher le tableau trié

Dans ce qui suit, nous allons détailler l'analyse de la procédure Trier. Les autres modules ont été déjà traités lors de la résolution d'autres problèmes.

a) Analyse de la procédure Trier

Résultat : Trier le tableau T

Traitement :

A l'aide de l'algorithme formel de la méthode de tri par insertion et en exécutant manuellement l'exemple, nous pouvons déduire ce qui suit :

Il s'agit d'un traitement répétitif complet de l'élément n° 2 jusqu'au dernier élément du tableau : Pour c de 2 à n Faire

Pour chaque valeur du compteur et si l'élément correspondant n'est pas à sa place, nous réalisons les actions suivantes :

- Ranger la valeur de T[c] dans la variable Tmp
- Décaler vers la droite les valeurs de T[c-1], T[c-2], ... jusqu'à arriver à une valeur qui est inférieure à T[c]. La procédure DECALER permettra de réaliser cette action.
- Affecter au dernier élément décalé la valeur de Tmp

```
Si T[c-1] > T[c]
Alors     Tmp ← T[c]
          Proc DECALER (T, c-1, p)
          T[p+1] ← Tmp
FinSi
```

b) Algorithme de la procédure Trier

```
0) Début Procédure Trier (n : entier; VAR T : Tab)
1) Pour c de 2 à n Faire
     Si T[c-1] > T[c]
     Alors     Tmp ← T[c]
               Proc DECALER (T, c-1, p)
               T[p+1]← Tmp
     FinSi
   Fin Pour
2) Fin Trier
```

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
c	Entier	Compteur
Tmp	Entier	Variable intermédiaire
p	Entier	Position d'insertion
DECLARER	Procédure	Permettant de décaler des éléments d'un tableau d'un nombre de positions

a) Analyse de la procédure DECLARER

Résultat : Décaler à droite les éléments du tableau T d'indice deb à l'indice fin

Traitement :

Il s'agit d'un traitement répétitif à condition d'arrêt :

Tant que *(fin >= 1) Et (T[fin] > Tmp)* ***Faire***

- Cette boucle est initialisée par l'instruction suivante : fin← deb
- L'action de décalage est une simple affectation : T[fin+1]←T[fin]
- La décrémentation par 1 de la variable fin : fin ← fin - 1

b) Algorithme de la procédure DECLARER

0) Début Procédure DECALER (VAR T : Tab; deb : Entier; Var fin : Entier)
1) fin ← deb
Tant que (fin >= 1) ET (T[fin] > Tmp) **Faire**
T[fin+1] ← T[fin]
fin ← fin - 1
Fin Tant que
2) Fin DECALER

e) Traduction en Pascal de la procédure Trier

```
Procedure Trier (n : Integer; Var T : Tab);
 Var c, Tmp,  p : Integer;
 Procedure Decaler (Var T : Tab; deb : Integer; Var fin : Integer);
   Begin
          fin := deb;
          While (fin >= 1) And (T[fin] > tmp) Do
                Begin
                      T[fin + 1] := T[fin];
                      fin := fin - 1;
                End;
   End;
{Procédure Trier}
 Begin
   For c := 2 To n Do
     If T[c-1] > T[c] Then
      Begin
            Tmp := T[c];
            Decaler (T, c-1, p);
            T[p+1] := Tmp;
      End;
 End;
```

Application

Proposez une solution récursive de la méthode de tri par insertion.

III. Tri Shell

III.1 Insuffisances de la méthode de tri par insertion

En analysant l'algorithme de la méthode de tri par insertion, nous pouvons remarquer qu'en moyenne, il s'agit d'un algorithme d'ordre n^2 puisqu'il comporte deux boucles imbriquées d'ordre n chacune. Il est toutefois évident que si le vecteur est initialement presque trié dans le bon ordre, le nombre d'opérations sera beaucoup plus réduit.
Si la méthode de tri par insertion est efficace quand la liste est à peu près triée, elle est inefficace en moyenne car elle ne change les valeurs que d'une position par instruction. En effet, cette méthode traite un à un les éléments de la liste à trier et réalise une rotation des éléments précédents jusqu'à avoir la position d'insertion de l'élément en cours.

III.2 Principe

Donald L. Shell proposa, en 1959, une variante du tri par insertion. Ce tri consiste à trier séparément des sous-tableaux du tableau initial objet du tri, formés par les éléments répartis de p éléments.

N.B.

Le tri Shell est une amélioration du tri par insertion. Au lieu de faire une rotation de tous les éléments, nous ferons une rotation par pas de p ce qui permet d'affiner le tri du tableau et de faire moins de déplacements d'éléments.
Nous commençons donc par un pas assez élevé et nous le diminuons au fur et à mesure jusqu'à arriver à un pas de 1. Ceci permet d'éliminer les plus grands désordres pour abréger le travail aux étapes suivantes. Le pas est diminué à l'aide d'une suite, mais il faut construire cette suite de manière à ce que les valeurs ne soient pas multiples entre elles sinon nous allons traiter des éléments et pas d'autres.

Shell propose la suite d'incréments vérifiant $p_1 = 1$, $p_{n+1} = 3p_n+1$ en réalisant les tris du plus grand incrément possible vers le plus petit.
D'autre part, puisque au dernier stade, p_0 est égal à 1, nous déterminerons le pas maximal par récurrence en inversant la relation donnée ci-dessus :

$$p_{k+1} = 3 * p_k + 1$$

et en s'assurant qu'il y a encore un nombre suffisant de composantes dans les sous-vecteurs considérés.

Remarque :

Le tri Shell trie chaque liste d'éléments séparés de p positions chacun avec le tri par insertion. L'algorithme effectue plusieurs fois cette opération en diminuant p jusqu'à p égal à 1 ce qui équivaut à trier tous les éléments ensemble

III.3 Exemple

Nous proposons d'utiliser la méthode du tri Shell pour trier un tableau T d'entiers en ordre croissant.

Considérons le tableau T contenant les 15 éléments suivants :

T	-5	30	0	-2	42	22	9	-4	10	15	40	-9	12	28	14
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Etape 1

La première action à faire consiste à déterminer la valeur maximale du pas.

Etape 2

N.B. En utilisant l'expression p ← p Div 3, nous pouvons déduire que les valeurs que prendra le pas p sont 13, 4 et 1.

Etape 2-1

Pour p=13, nous appliquons le tri par insertion pour chacun des sous-vecteurs de pas 13.

- Trions par insertion linéaire le sous-vecteur qui regroupe les composantes 1 et 14 et qui ont pour valeurs respectives –5 et 28.
  Le tableau ne subira pas de modifications car –5 < 28, d'où les valeurs garderont leurs places.
- Trions par insertion linéaire le sous-vecteur qui regroupe les composantes 2 et 15 qui ont pour valeurs respectives 30 et 14.

Nous allons obtenir le tableau suivant :

T	-5	14	0	-2	42	22	9	-4	10	15	40	-9	12	28	30
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Etape 2-2

Pour p=4, nous appliquons le tri par insertion pour chacun des sous vecteurs de pas 4 en commençant par l'élément d'indice 1, puis l'élément d'indice 2, etc.

- Trions par insertion linéaire le sous vecteur qui regroupe les composantes 1, 5, 9 et 13 et qui ont pour valeurs respectives : -5, 42, 10, 12.

Nous allons obtenir le tableau suivant :

T	-5	14	0	-2	10	22	9	-4	12	15	40	-9	42	28	30
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Remarquez que les valeurs des cases d'indices 1, 5, 9 et 13 sont triées par ordre croissant.

- Trions par insertion linéaire le sous vecteur qui regroupe les composantes 2, 6, 10 et 14 et qui ont pour valeurs respectives : 14, 22, 15 et 28.

Nous allons obtenir le tableau suivant :

T	-5	14	0	-2	10	15	9	-4	12	22	40	-9	42	28	30
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Remarquez que les valeurs des cases d'indices 2, 6, 10 et 14 sont triées en ordre croissant

- Quels sont les indices des éléments à trier en 3ème lieu ? Donnez le contenu du tableau T après avoir réalisé cette étape.
- Même question pour la 4ème opération.

Etape 2-3

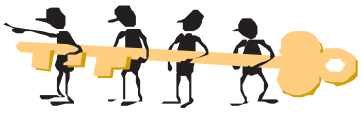
Pour p=1, nous appliquons le tri par insertion pour chacun des sous-vecteurs de pas 1. Ce qui revient à appliquer le tri par insertion simple à tout le tableau.

Constatations :

Il est bien évident que la dernière étape aurait pu suffire, puisqu'il s'agit d'un tri par insertion linéaire simple. Toutefois, dans le contexte où nous nous sommes progressivement placés, nous avons fait déplacer les grandes valeurs vers la fin du tableau et les petites vers le début. Le nombre de comparaisons dans le dernier passage sera très réduit et nous devrons déplacer moins souvent les éléments considérés.

III.4 Résolution du problème

Nous proposons de trier un tableau T en utilisant la méthode de tri Shell.
1- Analysez ce problème
2- Analysez chaque module proposé
3- Déduisez les algorithmes correspondants.



a) Analyse du programme principal :

Résultat : L'affichage du tableau trié est la tâche de la procédure **Afficher**.
Traitement :

- Nous devons donc trier le tableau ce qui sera la tâche de la procédure **Trier**.
- Pour pouvoir trier le tableau, nous devons tout d'abord le remplir ce qui se fera par la procédure **Remplir**.
- Pour remplir le tableau, nous devons connaître le nombre d'éléments à saisir ce qui est la tâche de la procédure **Saisir**.

b) Algorithme du programme principal

```
0) Début Tri_Shell
1) Proc Saisir (n)
2) Proc Remplir (T, n)
3) Proc Trier (T, n)
4) Proc Afficher (T, n)
5) Fin Tri_Shell
```

Dans ce qui suit, nous allons détailler l'analyse de la procédure **Trier**. Les autres modules ont été déjà traités lors de la résolution d'autres problèmes.

a) Analyse de la procédure Trier :

Résultat : Trier le tableau T.
Traitement :

- **L'action du tri :**

« Etant une variante du tri par insertion, ce tri consiste à trier séparément des sous tableaux du tableau initial objet du tri, formés par les éléments répartis de p éléments.»

Nous pouvons déduire de ce qui précède qu'il s'agit d'un traitement répétitif pour chacun des sous tableaux formés à partir d'une valeur du pas : **Tant que** (p ≠ 0) **Faire**

➢ **Pour** i de p à n **Faire** :
- valeur ← T[i]
- Décaler de la valeur du pas, vers la droite, les valeurs de T[c-p], T[c-2*p], etc.
- Affecter au dernier élément décalé le contenu de la variable Valeur

```
Tant que p ≠ 0 Faire
        p ← p div 3
        Pour i de p + 1 à n Faire
            valeur ← T[i]
            j ← i
            Tant que (j > p) ET (T[j-p] > valeur) Faire
                    T[j] ← T[j-p]
                        j ← j-p
            Fin Tant que

            T[j] ← valeur
        Fin pour
    Fin Tant que
```

- **Détermination de la valeur maximale du pas :**

Pour définir les valeurs successives du pas (que nous allons nommer p), Shell propose la relation suivante : $\mathbf{p_{k+1} = 3 * p_k + 1}$ avec 1 comme valeur de départ de p.

Pour avoir la valeur maximale du pas, nous allons utiliser une boucle à condition d'arrêt. Il est évident que la valeur du pas doit être plus petite que le nombre des éléments du tableau T. Nous aurons donc :

```
p ← 0
Tant que (p < n) Faire
   p ← 3*p + 1
Fin Tant que
```

b) Algorithme du programme principal

```
0) Début Procédure Trier (n : entier; VAR T : Tab)
1) p ← 0
Tant que (p < n) Faire
   p ← 3*p + 1
Fin Tant que
```

```
2) Tant que (p ≠ 0) Faire
        p ← p Div 3
        Pour i de p + 1 à n Faire
            valeur ← T[i]
            j ← i
            Tant que (j > p) ET (T[j-p] > valeur) Faire
                    T[j] ← T[j-p]
                    j ← j-p
            Fin Tant que
            T[j] ← valeur
        Fin pour
    Fin Tant que
3) fin Trier
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
p	Entier	Représente le pas
i, j	Entier	Compteurs
valeur	Entier	Variable intermédiaire

Activité

Traduisez la solution en un programme Pascal. Enregistrez votre travail sous le nom **Tri_Shel**.

IV. Applications

IV.1 Application 1 : Tri par fusion

IV.1.1 Principe

Il s'agit d'un tri suivant le paradigme "Diviser pour régner". Le principe est le suivant :

1- Nous divisons le tableau à trier en deux sous tableaux (en prenant par exemple, un élément sur deux pour chacun des sous tableaux).
2- Nous trions chacun d'entre eux.
3- Nous fusionnons les deux tableaux obtenus pour reconstituer le tableau trié.

Le principe est donc simple, en effet, pour trier une liste, nous la coupons d'abord en deux parties (de préférence de tailles égales ou proches). Ces deux parties sont triées puis fusionnées. La fusion construit une liste triée à partir des deux listes triées.

En effet, étant donnés deux tableaux d'éléments triés, de longueurs respectives L1 et L2, il est facile d'obtenir un troisième tableau d'éléments triés de longueur L1+L2, par

« interclassement » (ou fusion) des deux précédents tableaux.
Dans la suite de cette partie, nous allons nous intéresser à la présentation de l'action de fusion.

Remarque :

Nous constatons que la méthode de tri par fusion nécessite un tableau intermédiaire aussi grand que le tableau initial à trier et c'est là où réside le principal inconvénient car cela peut se révéler handicapant dans les situations où la place mémoire est restreinte.

IV.1.2 Exemple

Nous proposons d'utiliser la méthode de tri par fusion pour fusionner les deux tableaux d'entiers triés T1 et T2 de longueurs respectives 7 et 8.

T1	-5	-2	0	4	6	22	40
	1	2	3	4	5	6	7

T2	-7	0	8	8	12	28	35	39
	1	2	3	4	5	6	7	8

Le résultat sera rangé dans le tableau T.
Combien d'éléments devons-nous allouer pour le tableau T ? Justifiez votre réponse.

Bien sûr, nous allons déclarer un tableau T de 15 entiers (15 étant la somme des nombres d'éléments de T1 et de T2).

Etape 1

Nous allons commencer par :

- comparer le premier élément de chacun des deux tableaux T1 et T2
  Le plus petit est T2 [1] = -7
- placer -7 dans T [1] et se pointer à l'élément n°2 de T2
- se pointer à l'élément n°2 de T

T1	-5	-2	0	4	6	22	40
	1	2	3	4	5	6	7

T2	-7	0	8	8	12	28	35	39
	1	2	3	4	5	6	7	8

T	-7														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Remarquez que nous sommes toujours à la première position de T1.

Etape 2

Dans cette étape, nous allons :

- comparer le premier élément de T1 et le deuxième élément de T2
  Le plus petit est T1 [1] = -5
- placer -5 dans T [2] et se pointer à l'élément n°2 de T1
- se pointer à l'élément n°3 de T

T1	-5	-2	0	4	6	22	40
	1	2	3	4	5	6	7

T2	-7	0	8	8	12	28	35	39
	1	2	3	4	5	6	7	8

T	-7	-5													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Etape 3

Dans cette étape, nous allons :

- comparer T1 [2] et T2 [2]
  Le plus petit est T1 [2] = -2
- placer -2 dans T [3] et se pointer à l'élément n°3 de T1
- se pointer à l'élément n°4 de T

T1	-5	-2	0	4	6	22	40
	1	2	3	4	5	6	7

T2	-7	0	8	8	12	28	35	39
	1	2	3	4	5	6	7	8

T	-7	-5	-2												
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

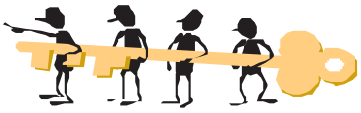
Activité

Réalisez manuellement la suite des étapes jusqu'à avoir placé tous les éléments de T1 et de T2 dans le tableau T.

IV.1.3 Résolution du problème

Nous proposons de trier un tableau T de b entiers ($5 \leq b \leq 100$) en utilisant la méthode de tri par fusion.

1- Analysez ce problème
2- Analysez chaque module proposé
3- Déduisez les algorithmes correspondants
4- Traduisez la solution en un programme Pascal.



a) Analyse du programme principal :

Résultat : L'affichage du tableau fusionné et trié T est la tâche de la procédure **Afficher.**

Traitement :

- Nous allons fusionner les deux tableaux triés T1 et T2 en appelant la procédure **Fusionner**
- Le remplissage des deux tableaux se fera suite à l'appel deux fois de la procédure **Remplir** qui réalisera :
  1- Le remplissage à tour de rôle des tableaux T1 et T2. Nous ferons de façon à ce que cette action soit effectuée en même temps que l'action de tri.
  2- La saisie du nombre d'éléments de chaque tableau.

b) Algorithme du programme principal

0) Début Tri_Fusion
1) Proc Remplir (T1, n1)
2) Proc Remplir (T2, n2)
3) Proc Fusionner (T1, n1, T2, n2, T)
4) Proc Afficher (T, n1+n2)
5) Fin Tri_Fusion

Dans ce qui suit, nous allons détailler l'analyse de la procédure **Remplir**

a) Analyse de la procédure Remplir :

Résultat : Remplir le tableau A de m entiers
Traitement :

- Le remplissage du tableau A par m entiers est une itération complète.
  Pour c de 2 à m Faire
- Pour chaque valeur du compteur, nous devons prévoir la saisie d'un entier qui doit être supérieur ou égal à son prédécesseur. Ceci nous amène à prendre en considération les contraintes suivantes :
  1- Commencer par saisir le premier élément A[1] avant d'entrer dans l'itération
  2- Pour chacun des éléments qui restent (de l'élément n°2 à l'élément n°m),
  Répéter
  Saisir A[c]
  Jusqu'à A[c] ≥ A[c-1]
- Saisir le nombre d'éléments du tableau est un traitement itératif à condition d'arrê
  Répéter
  Saisir m
  Jusqu'à m Dans [5,100]

Remarque :

Cette procédure sera appelée à deux reprises, une première fois pour remplir le tableau T1 par n1 entiers et une deuxième fois pour remplir le tableau T2 par n2 entiers.

b) Algorithme de la procédure Remplir :

0) Début Procédure Remplir (Var A : Tab ; Var m : entier)
1) **Répéter**
Lire (m)
Jusqu'à (m dans [5..100])
2) Lire (A[1])
Pour c de 2 à m **Faire**
Répéter
Lire (A[c])
Jusqu'à (A[c] ≥ A[c-1])
FinPour
3) Fin Remplir

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
c	Entier	Compteur

Détaillons maintenant l'analyse de la procédure **Fusionner.**

c) Analyse de la procédure Fusionner :

Résultat : Trier en fusionnant deux tableaux T1 et T2 de n1 et n2 entiers

Traitement :

- Le tri par fusion des deux tableaux T1 de n1 éléments et T2 de n2 éléments est une itération complète :

  Pour c de 1 à (n1+n2) Faire

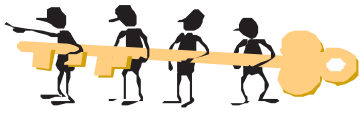
Pour chaque valeur du compteur, nous devons spécifier :
- Quel élément devons-nous ranger dans le tableau T ?
- A quel tableau appartient-il ?

- Il s'agit donc de l'action conditionnelle suivante :

Si T1[c1] < T2[c2] **alors**
ranger T1[c1] dans T[c]
incrémenter c1 de 1
Sinon
ranger T2[c2] dans T[c]
incrémenter c2 de 1

Activité

Que se passe-t-il si, par exemple, tous les éléments de T1 ont été rangés dans le tableau T alors que ceux de T2 ne le sont pas encore ?



En effet, la solution présentée précédemment ne prévoit pas ce cas, d'où la nécessité d'apporter la rectification suivante : Il s'agit toujours d'un traitement répétitif que nous devons décomposer en deux parties.

Partie1 : Ranger les éléments des tableaux T1 et T2 jusqu'à la fin de l'un des tableaux T1 ou T2.

```
[c ← 0, c1← 1, c2 ← 1]
        Répéter
                c ← c+1
                Si (T1 [c1] < T2 [c2]) Alors
                        T[c] ← T1 [c1]
                        c1 ← c1 + 1
                Sinon
                        T[c] ← T2 [c2]
                        c2 ← c2 + 1
        Jusqu'à (c1 > n1) OU (c2 > n2)
```

Partie 2 : Ranger le reste des éléments du tableau T1 ou T2 (il s'agit d'une action de copie)

```
Si (c1 > n1) Alors                {tous les éléments de T1 ont été rangés dans
                                   T, il reste à  copier les éléments de T2}

Pour i de c2 à n2 Faire
      T[c] ← T2[i]
      c ← c + 1
       Sinon
Pour i de c1 à n1 Faire           {copier le reste des éléments de T2}
      T[c] ← T1[i]
      c ← c + 1
```

Remarque :

Il est inutile de prévoir un paramètre pour définir le nombre d'éléments du tableau résultat T. En effet, la valeur de ce paramètre est la somme de n1 et n2.

d) Algorithme de la procédure Fusionner :

```
0) Début Procédure Fusionner (T1 : Tab; n1 : entier; T2 : Tab; n2 : entier; Var T : Tab)
1) c ← 0, c1 ← 1, c2 ← 1
2) Répéter
      c ← c + 1
      Si (T1[c1] < T2[c2]) Alors
         T[c] ← T1[c1]
         c1 ← c1 + 1
      Sinon
         T[c] ← T2[c2]
         c2 ← c2 + 1
      Fin si
   Jusqu'à (c1 > n1) OU (c2 > n2)

3) Si (c1 > n1) Alors
      Pour i de c2 à n2 Faire
         c ← c + 1
         T[c] ← T2[i]
      Fin Pour
   Sinon
            Pour i de c1 à n1 Faire
              c ← c + 1
              T[c]← T1[i]
            Fin Pour
   Fin Si
4) Fin Fusionner
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
c, c1, c2, i	Entier	Compteurs

Activité

Traduisez la solution en un programme Pascal. Enregistrez votre travail sous le nom **Tri_Fus1**.

Remarque :

Pour réduire le nombre de paramètres de la procédure Fusionner, nous allons donner la solution suivante qui interclasse deux suites d'éléments placés dans un tableau T, respectivement entre les indices **début** et **mil** et entre les indices **mil + 1** et **fin**. La solution fait appel à un tableau intermédiaire que nous appellerons Temp.

```
0) Début Procédure Fusionner (début, mil, fin : entier ; Var T : Tab)
1) i ← début, j ← mil + 1
2) Pour k de début à fin Faire
     Si ((j > fin) ou ((i ≤ mil) ET (T[i] < T[j]))) Alors
          Temp[k] ←T[i]
          i ← i + 1
     Sinon
          Temp[k] ← T[j]
          j ← j + 1
     Fin Si
   Fin Pour
3) Pour k de début à fin Faire   {copier le contenu de Temp dans T}
        T[k] ←Temp[k]
  Fin pour
4) Fin Fusionner
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
i, j, k	Entier	Compteurs
Temps	Tab	Tableau intérmédiaire

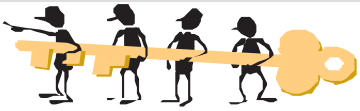
Activité

Traduisez la solution en un programme Pascal. Enregistrez votre travail sous le nom **Tri_Fus2**.

IV.2 Application 2 : Classement

Nous disposons des noms et des moyennes de 30 élèves d'une classe 4SI et nous proposons d'afficher sous forme de tableau le rang de chaque élève en plus des données déjà citées.

1. Définissez les structures de données à utiliser
2. Donnez l'analyse de ce problème.



1) Structure de données :

Comme structure de données, nous avons le choix d'utiliser :

1- un tableau d'enregistrements appelé ELEVE ayant la structure suivante :

ELEVE est un enregistrement de

- Nom de type chaîne de caractères
- Moyenne de type réel
- Rang de type entier

2- trois tableaux à une dimension pour ranger respectivement les noms, les moyennes et les rangs.

Dans la suite de la résolution, nous allons opter pour l'utilisation du tableau d'enregistrements appelé **TAB_EL**.

2) Analyse du programme principal :

Résultat : L'affichage du tableau TAB_EL sera effectué par la procédure **Afficher.**

Traitement :

- La détermination des rangs des élèves sera réalisée en appelant la procédure **Rechercher_Rang**
- La saisie des noms et des moyennes des 30 élèves se fera suite à l'appel de la procédure **Saisir**.

Algorithme du programme principal :

0) Début Traitement_EL
1) Proc Saisir (TAB_EL)
2) Proc Rechercher_Rang (TAB_EL)
3) Proc Afficher (TAB_EL)
4) Fin Traitement_EL

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
Tab_el	Tab	Tableau d'élèves
Saisir	Procédure	Permet de remplir un tableau de 30 élèves
Rechercher_Rang	Procédure	Permet de déterminer le rang de chaque élève
Afficher	Procédure	Permet d'afficher le tableau résultat

Dans ce qui suit, nous allons détailler l'analyse de chacune de ces procédures.

a) Analyse de la procédure Saisir :

Résultat : Remplir le tableau TAB_EL par les 30 enregistrements relatifs aux élèves de la classe 4SI.

Traitement :

- Le remplissage du tableau TAB_EL par 30 enregistrements est un traitement répétitif qui nécessite une itération complète :

Pour c de 1 à 30 Faire

Chaque valeur du compteur correspond au remplissage d'un enregistrement d'un élève donné qui comprend la saisie :

1- du nom
2- de la moyenne comme étant un réel compris entre 0 et 20.

N.B. : Puisqu'il s'agit d'un tableau d'enregistrements, la saisie se fait de la façon suivante :

1- remplir les champs nom et moyenne d'une variable temporaire de type enregistrement
2- copier la valeur de l'enregistrement dans le tableau TAB_EL.

b) Algorithme de la procédure Saisir :

```
0) Début Procédure Saisir (Var TAB_EL : Tab)
1) Pour c de 1 à 30 Faire
      Lire (Enreg.nom)
      Répéter
            Lire (Enreg.moy)
      Jusqu'à (Enreg.moy ≥ 0) et (Enreg.moy ≤ 20)
      TAB_EL[c] ← Enreg
   Fin Pour
2) Fin Saisir
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
c	Entier	Compteur
Enreg	Elève	Enregistrement intérmédiaire

Détaillons maintenant l'analyse de la procédure Afficher.

a) Analyse de la procédure Rechercher_Rang :

Résultat : Compléter le remplissage du tableau TAB_EL par les 30 rangs relatifs aux élèves de la classe 4SI.

Traitement :

➢ Le remplissage du tableau TAB_EL par les 30 rangs est une itération complète :

Pour i de 1 à 30 Faire

Pour chaque élément du tableau, nous allons compter le nombre d'enregistrement ayant une moyenne supérieure à la sienne. Cela nécessite un autre traitement répétitif sous la forme d'une itération complète :

Pour j de 1 à 30 Faire

Ce traitement répétitif comporte une action conditionnelle :

Si (Enreg2.Moy > Enreg1.Moy)
Alors Enreg1.Rang ← Enreg1.Rang + 1
FinSi

Une initialisation du rang à 1 doit être prévue avant cette itération complète :

Enreg1.Rang ← 1

b) Algorithme de la procédure Rechercher_Rang :

```
0) Début Procédure Rechercher_Rang (Var TAB_EL : Tab)
1)Pour i de 1 à 30 Faire
        Enreg1 ← Tab_el[i]
        Enreg1.Rang ← 1
        Pour j de 1 à 30 Faire
                Enreg2 ← Tab_el[j]
                Si (Enreg1.Moy > Enreg1.Moy)
                Alors Enreg1.Rang ← Enreg1.Rang + 1
                FinSi
        FinPour
   FinPour
2)Fin Rechercher_Rang
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
i, j	Entier	Compteurs
Enreg1, Enreg2	Elève	Enregistrements intérmédiaires

Détaillons maintenant l'analyse de la procédure Rechercher_Rang.

a) Analyse de la procédure Afficher :

Résultat : Afficher le tableau TAB_EL relatif aux élèves de la classe 4SI.

Traitement :

➢ L'affichage TAB_EL constitué de 30 enregistrements d'élèves est une itération complète : Pour i de 1 à 30 Faire

Chaque valeur du compteur correspond à l'affichage d'un enregistrement d'un élève qui comprend :

1- le nom
2- la moyenne
3- le rang

b) Algorithme de la procédure Afficher

0) Début Procédure Afficher (TAB_EL : Tab)
1) **Pour** i de 1 à 30 **Faire**
Enreg ← Tab_el[i]
Ecrire (“Nom de l'élève : ”, Enreg.Nom)
Ecrire (“Moyenne de l'élève : ”, Enreg.Moy)
Ecrire (“Rang de l'élève : ”, Enreg.Rang)
FinPour
2) Fin Afficher

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
i, j	Entier	Compteurs
Enreg	Elève	Enregistrement intérmédiaire

Retenons

- **Tri par insertion :** Cet algorithme consiste à traiter une à une les valeurs du tableau et à les insérer, au bon endroit, dans le tableau trié constitué des valeurs précédemment traitées et triées.
Les valeurs sont traitées dans l'ordre où elles apparaissent dans le tableau.

- **Tri Shell :** Ce tri, proposé en 1959 par Donald L. Shell, constitue une variante optimisée du tri par insertion.
Le tri par insertion provoquait le décalage de tous les éléments plus grands que l'élément à insérer.

Dans le tri Shell, les éléments ne sont pas décalés d'un élément à la fois, mais de plusieurs éléments, dont la différence d'indice est appelée "pas". Ainsi, à chaque étape, le tri est dégrossit puis le pas est réduit. Chaque réduction de pas provoque un affinage du tri.

- **Tri fusion :** La méthode "diviser pour régner" est tout à fait applicable au problème de tri par fusion. Plutôt que de trier le tableau complet, il est possible de trier deux sous tableaux de taille égale, puis de fusionner les résultats.

- **D'autres méthodes de tri** existent telles que le tri par comptage, tri par création, tri radix, tri rapide, tri par permutation, etc.

Citation du Chapitre

« L'ordre conduit à toutes les vertus mais qu'est-ce qui conduit à l'ordre »

Georg Christoph Lichtenberg

Exercices

Exercice 1 (Bac TP 2005)

L'algorithme suivant est celui d'une fonction permettant de retourner la position du plus petit élément dans un tableau A de k éléments à partir d'une position p.

```
0) Deffn pos_min (A : tab ; p, k : entier): entier
1)pm  ← p
  Pour i de p+1 à k Répéter
        Si (A[i] < A[pm])
          Alors pm  ← i
        Finsi
  Fin pour
2) pos_min  ← pm
3) Fin Pos_min
```

Utiliser la fonction **Pos_min** ci-dessus pour écrire un programme Pascal permettant de saisir un tableau **T** de **n** réels, le trier dans l'ordre croissant par la méthode de "tri par sélection" puis de l'afficher.

Exercice 2 (Bac TP 2005)

On propose ci-dessous l'algorithme d'une procédure de tri à Bulles :

```
0) DefProc TRI_Bulles (Var T : Tab; n : entier)
1) Pour i de 1 à n-1 Répéter
          Pour j de 1 à n-i Répéter
               Si  (T[j] < T[j+1])
              Alors Proc Permut (T[j], T[j+1])
              Fin si
          Fin Pour
   Fin Pour
2) Fin TRI_Bulles
```

Remarque :

Le module **Permut** (a, b) permute le contenu de deux entiers a et b.

Questions

1- Ecrire un programme Pascal intitulé **Tri** permettant de saisir **p** éléments entiers dans un tableau **V** et de faire appel au module **TRI_Bulles** ci-dessus.
2- Sous forme de commentaire, déterminer l'ordre du tri (croissant ou décroissant) accompli par le programme.
3- Dans le cas où le tableau **V** est déjà trié dès la saisie, les parcours effectués par le module **TRI_Bulles** s'avèrent inutiles. En effet, a**ucune permutation n'aura lieu au sein de ce module** dans ce cas.
Modifier la procédure TRI_Bulles pour tenir compte de cette contrainte.

Exercice 3 (Bac TP 2005)

Ecrire un programme Pascal intitulé Tri permettant de trier un tableau **T** de **N** entiers distincts (5 < N < 20) selon le principe suivant :
Pour chaque élément du tableau **T** :
- Déterminer le nombre d'éléments qui lui sont inférieurs.
- En déduire sa position au sein d'un autre tableau résultat appelé R.

Exemple : Pour un tableau T de 10 éléments :

6	2	0	5	12	25	13	8	14	3
1	2	3	4	5	6	7	8	9	10

Quatre valeurs sont inférieures au premier élément du tableau T. Cet élément sera donc placé à la position 5 du tableau **R.**

N.B : - Le candidat n'est pas appelé à vérifier que les éléments du tableau T sont distincts.

Exercice 4 (Bac TP 2005)

Ecrire un programme Pascal intitulé **Tri_permut** permettant de trier, dans l'ordre croissant, un tableau **T** de **N** éléments (5 < N < 20) selon le principe suivant :
- Effectuer toutes les permutations possibles sur les éléments du tableau T.
- Avant chaque permutation, faire appel à une fonction qui déterminera si la disposition des éléments de T est ordonnée ou non.
- Arrêter le traitement ci-dessus dès que le tableau T est trié.

Dans ce qui suit, l'algorithme de la procédure **Combinaisons** qui permet d'effectuer toutes les permutations possibles sur les N éléments du tableau **T**. Les procédures **Permut** et **Affiche** permettent respectivement de permuter deux éléments du tableau **T** et d'afficher le contenu de ce dernier.

0) Def Proc **Combinaisons** (N : entier; var T : Tab)
1) **Pour** i de 1 à N **Répéter**
    Pour j de 1 à N-1 **Répéter**
        Proc Permut(T[j], T[j+1])
        Proc Affiche(T, N)
    Fin pour
Fin pour
2) Fin Combinaisons

Exercice 5

Tri par rangement ou par création

Soit un vecteur de n entiers.

Nous cherchons à trier ce vecteur par ordre croissant par une méthode dite du rangement. Cette méthode est décrite par les étapes suivantes :

1- Trouver le minimum local de la partie restant à trier
2- Mettre le minimum dans un vecteur résultat
3- Recommencer jusqu'à avoir rangé tous les éléments

Questions

Proposez une analyse modulaire au problème dans chacun des cas suivants :

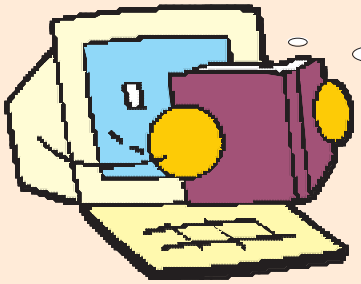
- Le vecteur initial peut être modifié
- Le vecteur initial reste inchangé

Exercice 6

a) Proposez une analyse modulaire en utilisant un procédé récursif, au problème permettant de trier un vecteur d'entiers par la méthode de tri **'Par fusion'**,

b) Déduisez les algorithmes correspondants,

c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Fus_Rec**.



Lecture

LES TRIS

TRIS INTERNES

Tri par insertion.

- *Tri par insertion linéaire*
- *Tri par insertion en double sens*
- *Tri Shell*
- *Tri par arbre binaire*

Tri par échange.

- *Tri à bulle*
- *Tri cocktail shaker*
- *Tri par la méthode de K.E.Batcher*
- *Quicksort*

Tri par sélection, par fusion, par distribution

TRIS EXTERNES

- *Tri polyphase*

www.chez.com/algo/tri/

Chapitre 4

Les algorithmes récurrents

Objectifs

- Acquérir des habilités de résolution de problèmes à travers l'apprentissage des algorithmes numériques.
- Proposer des solutions à quelques problèmes récurrents.

Plan du chapitre

Chapitre 4

Les algorithmes récurrents

I. Introduction

Un algorithme ou un traitement est dit **récurrent** s'il utilise un procédé itératif ou récursif pour engendrer un résultat qui peut dépendre de p résultats précédents, nous parlons alors d'un algorithme ou d'un traitement **récurrent d'ordre p.**

Ordre 1 : Un algorithme récurrent d'ordre 1 est un algorithme donnant un résultat dépendant du résultat précédent.

Ordre 2 : Un algorithme récurrent d'ordre 2 est un algorithme donnant un résultat dépendant des deux résultats précédents.

Ordre 3 : Un algorithme récurrent d'ordre 3 est un algorithme donnant un résultat dépendant des trois résultats précédents.

Ordre p : Un algorithme récurrent d'ordre p est un algorithme donnant un résultat dépendant des p résultats précédents.

II. Calcul de somme

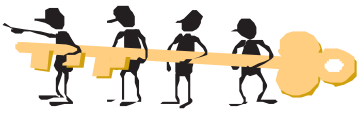
Activité 1

Nous voulons calculer la somme des éléments d'une matrice carrée d'entiers d'ordre n ($4 \leq n \leq 20$).

Questions :

1- Déterminez si ce traitement est récurrent. Dans l'affirmative donnez son ordre.

2- Proposez une analyse, puis déduisez l'algorithme de la fonction nommée SOMME_MAT, qui calcule la somme des éléments de la matrice carrée.



1- Le calcul de la somme des éléments d'une matrice nécessite une initialisation à zéro de la variable S contenant la somme; nous ajoutons a cette dernière le premier élément de la matrice, (S ← S + M[1, 1]), nous obtenons un deuxième résultat S auquel nous ajoutons le deuxième élément de la matrice (S ← S + M[1, 2]) et ainsi de suite. C'est le cumul de

tous les éléments de la matrice dans la variable S.
Puisque ce traitement fait toujours référence à l'élément précédent, donc c'est un **traitement récurrent d'ordre 1**.

2/ Analyse de la fonction SOMME_MAT

Résultat : SOMME_MAT
Traitement : Pour calculer la somme des éléments de la matrice M de type MAT, nous devons cumuler tous les entiers qu'elle contient :

[S ← 0]
Pour ligne de 1 à N **Faire**
Pour colonne de 1 à N **Faire**
S ← S + M [ligne, colonne]

Algorithme de la fonction SOMME_MAT

0) Début fonction SOMME_MAT (M : MAT ; N : Entier non signé) : Entier
1) S ← 0
Pour ligne de 1 à N **Faire**
Pour colonne de 1 à N **Faire**
S ← S + M [ligne, colonne]
Fin Pour
Fin Pour
2) SOMME_MAT ← S
3) Fin SOMME_MAT

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
S	Entier	Variable de cumul
Ligne	Entier	Compteur des lignes de la matrice
Colonne	Entier	Compteur des colonnes de la matrice

Traduction en Pascal :

```
FUNCTION SOMME_MAT (M : MAT; N : Word) : Integer ;
VAR      S, ligne, colonne : Integer ;
Begin
      S := 0;
      For ligne := 1 To N Do
            For colonne := 1 To N Do
                  S := S + M[ligne, colonne] ;
      SOMME_MAT := S ;
End;
```

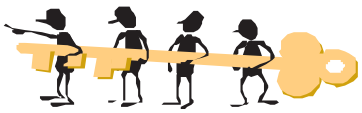
III. Algorithme récurrent sur les chaînes

Activité 2 : Suite de Thue-Morse (Suites de chaînes de caractères)

Avec toute transformation d'une chaîne de caractères en une autre, nous pouvons définir des suites récurrentes.
Par exemple, si nous considérons sur l'ensemble des chaînes constituées de 0 et de 1 (appelées aussi chaînes ou mots binaires) la transformation qui consiste à remplacer toute occurrence du caractère "0" par la chaîne "01" et toute occurrence du caractère "1" par la chaîne "10", nous pouvons définir la suite de Thue-Morse en partant de la chaîne "0" :

U0 → "0"
U1 → "01"
U2 → "0110"
U3 → "01101001"
U4 → "0110100110010110"

Questions :
1- Quel est l'ordre de récurrence de cette suite ?
2- Proposez une analyse, puis déduisez les algorithmes du problème permettant de calculer et d'afficher le $N^{ème}$ terme de la suite de Thue-Morse à partir d'un caractère A ("0" ou "1") donné.
3- Traduisez et testez la solution du problème. Enregistrez votre programme sous le nom **Thue_Morse**.



1) Le premier résultat dépend de la première valeur du caractère ("0" ou "1"), un deuxième résultat est obtenu à partir du précédent trouvé, et ainsi de suite. Nous concluons que c'est **une suite récurrente d'ordre 1**.

2) ***Analyse du problème***

a) Analyse du programme principal

Résultat : Ecrire ("La suite de Thue-morse à partir de ", A, " Est ", **FN** Thue_Morse (N, A))

Traitement : Thue_Morse est une fonction appelée au niveau du programme principal dans un contexte d'affichage. Cette fonction génère la suite Thue_Morse à partir d'un caractère A de départ.
Données : La procédure Saisie aura la tâche de lire l'entier N et le caractère A.

b) Algorithme du programme principal et codification des objets

0) Début Suite_Thue_Morse
1) Proc Saisie (N, A)
2) Ecrire ("La suite de Thue-morse à partir de ", A, " Est ", **FN** Thue_Morse (N, A))
3) Fin Suite_Thue_Morse

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
N	Entier non signé	Ordre du terme à calculer
A	Caractère	"0" ou "1"
Thue_Morse	Fonction	Génère la suite Thue_Morse

c) Analyse de la fonction Thue_Morse

Résultat : Thue_Morse
Traitement : [CH ← A]
Pour i de 1 à N **Faire**
j ← 1
Répéter
Si CH[j] = "0" **Alors** insère ("1", CH, j+1)
Sinon insère ("0", CH, j+1)
Fin Si
L ← Longueur (CH)
j ← j + 2
Jusqu'à j > L
Fin Pour

d) Algorithme de la fonction Thue_Morse

0) Début Fonction Thue_Morse (N : Entier non signé ; A: Caractère) : Chaîne
1) CH ← A

```
Pour  i de 1 à N Faire
        j ← 1
        Répéter
              Si CH[j] = "0" Alors insère  ("1", CH, j+1)
                             Sinon insère  ("0", CH, j+1)
              Fin Si
                 L ← Longueur (CH)
                 j ← j + 2
        Jusqu'à  j > L
  Fin Pour
2) Thue_Morse ← CH
3) Fin Thue_Morse
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
j, i	Entier non signé	Compteurs
CH	Chaîne	Chaîne représentant la suite Thue_morse
L	Entier non signé	Longueur de la chaîne

e) Traduction en Pascal

```
PROGRAM Suite_Thue_Morse ;
USES Crt ;

VAR   N : Word ;
      A : Char ;

PROCEDURE Saisie (VAR N : Word ; VAR A: Char );
Begin
      Repeat
            Write('Donner le nombre d''éléments de la suite : ');
            readLn (N);
      Until N In [2 .. 100] ;

      Repeat
            Write('Donner un caractère 0 ou 1 : ');
            readLn (A);
      Until A In ['0', '1'] ;
End;
```

```
FUNCTION Thue_Morse (N: Word; A : Char ) : String ;
VAR   Ch : String ;
      i, j, L : Word ;
Begin
      Ch := A ;
      For i := 1 To N Do
      Begin
            j := 1 ;
            Repeat
                  If Ch [j] = '0' Then Insert ('1', Ch, j+1)
                  Else Insert ('0', Ch, j+1);
                  L := Length (Ch) ;
                  j := j+ 2;
            Until j > L ;
      End;
      Thue_Morse := Ch ;
End;

{ Programme principal }
BEGIN
      Saisie (N, A) ;
      WriteLn('La suite de Thue-morse à partir de ', A, ' Est ',
Thue_Morse (N, A) ) ;
END .
```

IV. Triangle de Pascal

Activité 3

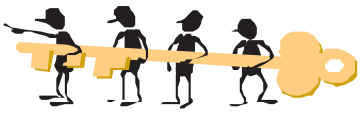
Nous voulons afficher les n premières lignes du Triangle de Pascal ($2 \leq n \leq 100$).
Nous rappelons que le principe de remplissage des n premières lignes de la matrice MAT représentant le Triangle de Pascal est le suivant :
Pour une ligne donnée :
Le premier et le dernier éléments sont égaux à 1,
Les autres éléments sont déterminés en appliquant la formule suivante :
MAT [ligne, colonne] = MAT [ligne-1, colonne] + MAT [ligne-1, colonne-1]

Exemple :

1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1

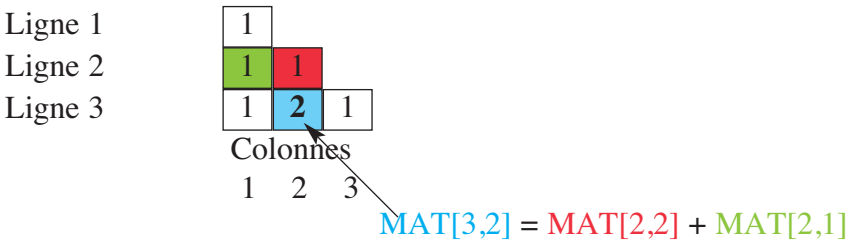
Questions :

1- Est-ce que ce traitement est récurrent ? Dans l'affirmative donnez son rang.
2- Proposez une analyse au problème en utilisant un procédé itératif.
3- Déduisez les algorithmes correspondants.
4- Proposez une analyse au problème en utilisant un procédé récursif.
5- Déduisez les algorithmes correspondants.

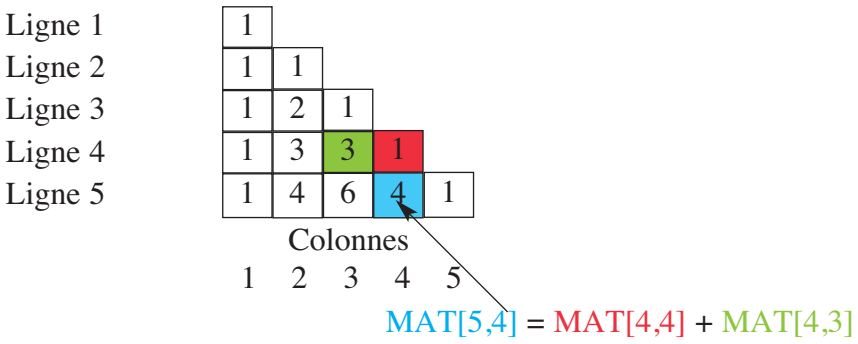


1) Exemple :

Pour n = 3, le Triangle de Pascal affiché est le suivant :



Pour n = 5, le Triangle de Pascal affiché est le suivant :



Nous constatons que le calcul du contenu de la case (5,4) fait référence au contenu de deux cases (4,4) et (4,3). C'est un traitement récurrent d'ordre 2.

2) Solution itérative :

a) Analyse du programme principal

Résultat : Affichage d'une matrice contenant les différentes valeurs du Triangle de Pascal, réalisé par la procédure Afficher_Triangle

Traitement : Remplissage du Triangle de Pascal, représenté par une matrice carrée d'ordre n. C'est la tâche de la procédure Remplir_MAT.
Données : un entier N qui représente le nombre de lignes du Triangle, nous utilisons la procédure Saisir.

b/ Algorithme du programme principal et codification des objets

0) Début Triangle_Pascal
1) Proc Saisir (N)
2) Proc Remplir_MAT (N, MAT)
3) Proc Afficher_Triangle (N, MAT)
4) Fin Triangle_Pascal

Tableau de codification des nouveaux types :

Type
Matrice = Tableau de max lignes et max colonnes d'entiers

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
Max	Constante = 20	Nombre maximum de lignes et de colonnes
N	Entier non signé	Nombre de lignes du Triangle de Pascal
MAT	Matrice	Représentant le Triangle de Pascal
Saisir	Procédure	Permet de saisir le nombre de lignes du Triangle
Remplir_MAT	Procédure	Permet de remplir la matrice représentant le Triangle de Pascal
Afficher_Triangle	Procédure	Permet d'afficher le Triangle de Pascal

c/ Analyse de la procédure Remplir_MAT

Résultat : Remplir la matrice MAT
Traitement : Nous constatons que :
Ligne 1 → MAT [1,1] contient la valeur 1
Ligne 2 → MAT [2,1] et MAT [2,2] contiennent la valeur 1
Ligne x →

```
Pour ligne de 3 à N Faire
        MAT [ligne, 1] ← 1   {première case de la ligne reçoit 1}
        MAT [ligne, ligne] ← 1 {dernière case de la ligne reçoit 1}
       Pour colonne de 2 à ligne-1 Faire
           MAT [ligne, colonne] ← MAT [ligne-1, colonne] + MAT [ligne-1, colonne-1]
        Fin Pour
Fin Pour
```

d/ Algorithme de la procédure Remplir_MAT

```
0) Début procédure Remplir_MAT (N : Entier; VAR MAT : Matrice);
1) MAT [1,1] ← 1
2) MAT [2,1] ← 1
3) MAT [2,2] ← 1
4) Pour ligne de 3 à N Faire
        MAT [ligne, 1] ← 1
        MAT [ligne, ligne] ← 1
         Pour colonne de 2 à ligne-1 Faire
          MAT [ligne, colonne] ← MAT [ligne-1, colonne] + MAT [ligne-1, colonne-1]
         Fin Pour
    Fin Pour
5) Fin Remplir_MAT
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
ligne	Entier non signé	Compteur de lignes de la matrice
colonne	Entier non signé	Compteur de colonnes de la matrice

e/ Analyse de la procédure Afficher_Triangle

Résultat : Afficher la matrice MAT
Traitement :

```
Pour ligne de 1 à N Faire
   Ecrire ( )   {avec retour à la ligne }
   Pour colonne de 1 à ligne Faire
      Ecrire (Mat [ligne, colonne],"    ")          { Sans retour à la ligne }
   FinPour
FinPour
```

f/ Algorithme de la procédure Afficher_Triangle

```
0) Procédure Afficher_Triangle (n : Entier non signé ; mat : matrice)
1) Pour ligne de 1 à N Faire
        Ecrire ( )
        Pour colonne de 1 à ligne Faire
                Ecrire (Mat [ligne, colonne],"    ")
        FinPour
      FinPour
2) Fin Afficher_Triangle
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
ligne	Entier non signé	Numéro de ligne
colonne	Entier non signé	Numéro de colonne

g/ Traduction en Pascal

```
PROGRAM Triangle_Pascal ;
USES Crt ;
const Max = 20;

TYPE Matrice = ARRAY[1.. max, 1.. max] Of Integer ;

VAR  N : Word ;
     MAT : Matrice ;

PROCEDURE Saisir (VAR N : Word );
Begin
     Repeat
           Write('Nombre de lignes du triangle : ');
           ReadLn(N) ;
     Until N In [3.. max] ;
End ;

PROCEDURE Remplir_MAT (N: Word; VAR MAT : Matrice);
VAR ligne, colonne : Word ;
Begin
  MAT [1,1] := 1 ;
     MAT [2,1] := 1 ;
     MAT [2,2] := 1 ;
     For ligne := 3 To N Do
     Begin
          MAT [ligne, 1] := 1;
          MAT [ligne, ligne] := 1;
          For colonne := 2 To ligne-1 Do
          Begin

              MAT [ligne, colonne] := MAT [ligne-1, colonne] + MAT
[ligne-1, colonne-1];
          End ;
     End ;
End ;
```

```
PROCEDURE Afficher_Triangle (N : Word; MAT : matrice);
VAR ligne, colonne : Word ;
Begin
      For ligne := 1 To N Do
      Begin
            WriteLn ;
            For colonne := 1 To ligne Do
               Write(Mat [ligne, colonne]:2,'     ') ;
      End;
End ;

{ Programme principal }
BEGIN
      Saisir (N) ;
      Remplir_MAT(N, MAT) ;
      Afficher_Triangle (N, MAT) ;
END .
```

3/ Solution récursive :

a) Analyse du programme principal

Résultat : Affichage d'une matrice contenant les différentes valeurs du Triangle de Pascal, réalisé par la procédure Afficher_Triangle
Traitement : Remplissage du Triangle de Pascal, représenté par une matrice carrée d'ordre N. C'est la tâche de la procédure Remplir_Triangle qui fait appel à une fonction Val_Triangle permettant de déterminer les différentes valeurs.
Données : un entier N qui représente le nombre de lignes du Triangle, nous utiliserons la procédure Saisir.

b) Algorithme du programme principal et codification des objets

0) Début Triangle_Pascal
1) Proc Saisir (N)
2) Proc Remplir_Triangle (N, MAT)
3) Proc Afficher_Triangle (N, MAT)
4) Fin Triangle_Pascal

Le tableau de codification des nouveaux types et le tableau de codification des objets globaux sont les mêmes que ceux de la solution itérative sauf que le nom de la procédure Remplir_MAT est devenu Remplir_Triangle.

c) Analyse de la procédure Remplir_Triangle

Résultat : Matrice remplie représentant le Triangle de Pascal relatif à N lignes
Traitement : Pour chacune des N lignes de la matrice, un parcours des colonnes (de la première à celle qui porte le numéro de la ligne), est nécessaire pour déterminer leurs valeurs en faisant appel à la fonction récursive Val_Triangle.

Algorithme de la procédure Remplir_Triangle

```
0) Procedure Remplir_Triangle (N : Entier non signé ; VAR  MAT : matrice)
1) Pour ligne de 1 à n Faire
       Pour colonne de 1 à ligne Faire
              Mat [ligne, colonne] ← FN Val_Triangle (colonne, ligne)
       FinPour
   FinPour
2) Fin Remplir_Triangle
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
ligne	Entier non signé	Numéro de ligne
colonne	Entier non signé	Numéro de colonne
Val_Triangle	Fonction	Permet de déterminer les valeurs du Triangle.

d) Analyse de la fonction Val_Triangle

Résultat : Val_Triangle
Traitement : Déterminer une valeur du Triangle selon la ligne (x) et la colonne (y) données comme paramètres.
Pour la première colonne, la valeur est égale à 1,
Si le numéro de ligne est égal au numéro de colonne, la valeur est aussi égale à 1,
sinon la valeur trouvée est égale à (la valeur trouvée à l'intersection de la colonne x et de la ligne y-1) + (la valeur trouvée à l'intersection de la colonne x-1 et de la ligne y-1)

```
Si (x = 1) OU (y = x) Alors Val_Triangle ← 1
   Sinon Val_Triangle ← FN Val_Triangle (x, y-1) + FN Val_Triangle (x - 1, y - 1)
```

Algorithme récursif de la fonction Val_Triangle

```
0) Fonction Val_Triangle (x, y : Entier) : Entier
1) Si (x = 1) OU (y = x) Alors Val_Triangle ← 1
   Sinon Val_Triangle ← FN Val_Triangle (x, y-1 ) + FN Val_Triangle (x - 1, y - 1)
   Finsi
2) Fin Val_Triangle
```

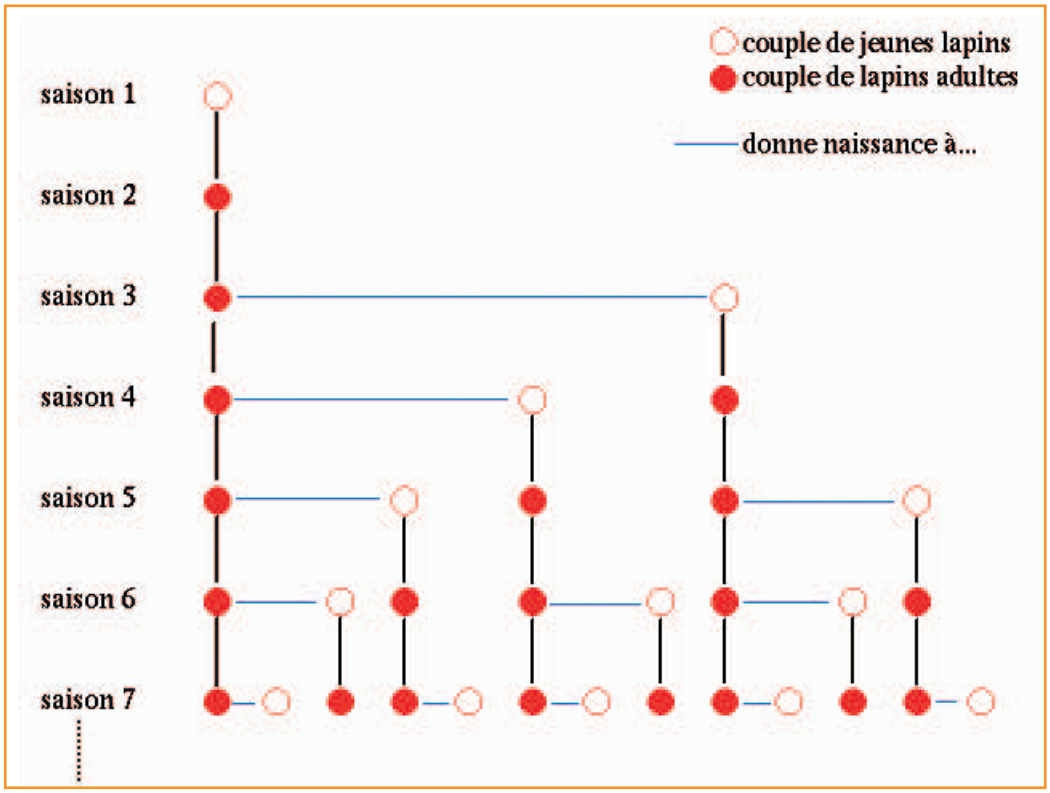
Les analyses et les algorithmes des procédures Saisie et Afficher_Triangle sont les mêmes que celles de la solution itérative.

g) Traduction en Pascal de la fonction Val_Triangle

```
FUNCTION Val_Triangle (x, y : Integer) : Integer ;
Begin
  If (x = 1) OR (y = x) Then Val_Triangle := 1
  Else Val_Triangle := Val_Triangle (x, y-1) + Val_Triangle (x-1, y-1) ;
End;
```

V. La suite de Fibonacci

Léonard de Pise, plus connu sous le nom de Fibonacci, étudia du point de vue numérique la reproduction des lapins. L'unité de base est un couple de lapins, il considère qu'un couple de jeunes lapins met une saison pour devenir adulte, attend une deuxième saison de gestation, puis met au monde un couple de jeunes lapins à chaque saison suivante. En supposant que les lapins ne meurent jamais, nous obtenons donc le schéma ci-dessous :



Lorsque nous mettons côte à côte le nombre de couples de lapins à chaque saison, nous obtiendrons... la suite de Fibonacci :

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, etc.

Curieusement, nous remarquons que le rapport entre un nombre de la suite et son précédent s'approche de plus en plus du nombre d'or, que nous allons découvrir dans le paragraphe suivant :

21/13 ≅ 1,615 ; 34/21 ≅ 1,619 ; 144/89 ≅ 1,617 ; 610/377 ≅ 1,618

Le nombre de couples de lapins U_n à la saison n est égal au nombre de couples de lapins adultes, c'est-à-dire le nombre total de lapins qu'il y avait à la saison précédente (n-1), auquel nous ajoutons le nombre de couples de jeunes lapins, qui est égal au nombre de couples de lapins adultes à la saison précédente, donc au nombre total de couples à la saison (n-2). C'est pourquoi nous avons : $\mathbf{U_n = U_{n-1} + U_{n-2}}$

La suite de Fibonacci est une suite récurrente dont chaque élément obéit à la relation de récurrence suivante : $\mathbf{U_n = U_{n-1} + U_{n-2}}$

avec $\mathbf{U_1 = 1}$ et $\mathbf{U_2 = 1}$

Donc c'est une suite récurrente d'ordre 2.

Activité 4

Pour un entier N donné, calculez le $n^{ème}$ terme de la suite de Fibonnaci

1) Ecrivez une fonction FIBO_IT1, solution itérative sans tableau.
2) Ecrivez une fonction FIBO_IT2, solution itérative utilisant un tableau.
3) Ecrivez une fonction FIBO_REC, solution récursive.

1) Solution itérative sans utiliser un tableau :

Analyse de la fonction Fibo_IT1

Résultat : le $n^{ème}$ terme de la suite de Fibonacci

Traitement :

```
FIBO_IT1 ← F
 [u1 ← 1, u2 ← 1]
   Si N ≤ 2 Alors F ← 1
   Sinon
      Pour i de 3 à N Faire
        F ← u1 + u2
        u1 ← u2
        u2 ← F
      Fin Pour
   Fin Si
```

Algorithme

```
0) Début fonction FIBO_IT1 (N : Entier) : Entier
1) u1 ← 1, u2 ← 1
   Si N ≤ 2 Alors F ← 1
   Sinon
      Pour i de 3 à N Faire
      F ← u1 + u2
      u1 ← u2
      u2 ← F
     Fin Pour
   Fin Si
2) FIBO_IT1 ← F
3) Fin FIBO_IT1
```

2) Solution itérative utilisant un tableau :

Analyse de la fonction Fibo_IT2

Résultat : le N^{ème} terme de la suite de Fibonacci

Traitement :

```
FIBO_IT2 ← F
[T[1] ← 1, T[2] ← 1]
   Si N ≤ 2 Alors F ← 1
      Pour i de 3 à N Faire
        T[i] ← T[i-1] + T[i-2]
      Fin Pour
            F ← T[n]
   Fin Si
```

Algorithme

```
0) Début fonction FIBO_IT2 (N : Entier ; T : Tab) : Entier
1) T[1] ← 1, T[2] ← 1
  Si N ≤ 2 Alors F ← 1
   Sinon
     Pour i de 3 à N Faire
        T[i] ← T[i-1] + T[i-2]
     Fin Pour
      F ← T[n]
    Fin Si
2) FIBO_IT2 ← F
3) Fin FIBO_IT2
```

3) Solution récursive :

Analyse de la fonction Fibo_Rec

Résultat : Fibo_Rec

Traitement : Le N^{ème} terme de la suite de Fibonacci est obtenu comme suit :

- Cas particulier : **Si** N ≤ 2 **alors** FIBO_REC ← 1
- Cas général :

  Sinon FIBO_REC ← FN FIBO_REC (N-1) + FN FIBO_REC (N-2).

Algorithme

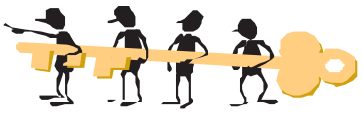
0) Début fonction FIBO_REC (N : Entier) : Entier
1) **Si** N ≤ 2 **Alors** FIBO_REC ← 1
   Sinon FIBO_REC ← FN FIBO_REC (N-1) + FN FIBO_REC (N-2)
   Fin Si
2) Fin FIBO_REC

VI. Le nombre d'or

Activité 5

Complétez le tableau suivant par Fib (n+1) / Fib (n) et par sa valeur approchée. Que déduisez-vous ?

n	Fib (n)	Fib (n+1) / Fib (n)	Valeur approchée de Fib (n+1) / Fib (n)
1	1	1	1
2	1	2	2
3	2	3/2	1,5
4	3	5/3	1,666
5	5	8/5	1,6
6	8	13/8	1,625
7	13	21/13	1,615
8	21		
9	34		
10	55		
11	89		
12	144		
13	233		



La suite (V_n) définie sur N* par $V_n = \frac{Fib\ (n+1)}{Fib\ (n)}$ semble converger vers $\lambda = \frac{1+\sqrt{5}}{2}$ appelé nombre d'or, dont une valeur approchée est 1,618.

Définition et valeur du nombre d'or

Le nombre d'or est la solution positive de l'équation : $x^2 - x - 1 = 0$
C'est à dire le nombre $\frac{1+\sqrt{5}}{2}$

Les 100 premières décimales du nombre d'or sont : 1,618 033 988 749 894 848 204 586 834 365 638 117 720 309 179 805 762 862 135 448 622 705 260 462 189 024 497 072 072 041.

Nous constatons que :
La suite de Fibonacci est une suite de nombres entiers. Voici le début de cette suite :
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, etc.
Un nombre de la suite est le résultat de la somme de ses deux précédents (N3 = N1 + N2). Voici maintenant pourquoi le nombre d'or et la suite de Fibonacci sont étroitement liés :

1/0 = n'existe pas.
1/1 = 1
2/1 = 2
3/2 = 1,5
5/3 = 1,6666...
8/5 = 1,6
13/8 = 1,625
21/13 = 1,61538...
34/21 = 1,61904...

$$\frac{Fib\ (n+1)}{Fib\ (n)} = 1,6....$$

Le nombre d'or est habituellement désigné par la lettre λ (phi) de l'alphabet grec en l'honneur de Phidias, sculpteur et architecte grec du Parthénon (**voir partie lecture**).

Activité 6

Soient deux suites U et V définies à partir de :

$U_1 = 1$ et $U_2 = 2$

$U_i = U_{i-1} + U_{i-2}$ pour $U \geq 3$

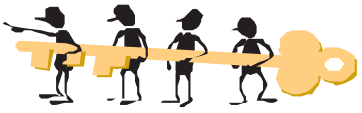
$V_i = U_i / U_{i-1}$ pour tout $i \geq 2$

La suite V_n tend vers une limite appelée nombre d'or.

On suppose que le $n^{ième}$ terme de la suite V soit V_n, donc un nombre approché du nombre d'or avec une précision e dès que $|V_n - V_{n-1}| < e$.

Ecrivez un programme Pascal nommé Nombre_Or, qui cherche V_n à 10^{-4} près et son rang.

Bac pratique 2004



```
PROGRAM  Nombre_Or ;
USES Crt ;

TYPE TAB_E = ARRAY[1..1000] OF Integer ;
     TAB_R = ARRAY[1..100] OF Real ;

VAR   U: TAB_E;
      V: TAB_R ;
      i: Integer ;

PROCEDURE  Init (VAR U : TAB_E) ;
BEGIN
   U[1] := 1;
   U[2] := 2;
   i := 2 ;
End;

PROCEDURE Calcul (VAR u : TAB_E ; VAR V : TAB_R; VAR i : Integer);
BEGIN
  Repeat
       i := i+1;
       U[i] := U[i-1] + U[i-2];
       V[i] := U[i] / U[i-1];
  Until  ABS(V[i] - V[i-1]) < 0.0001 ;
END;
```

```
PROCEDURE Affiche (V : TAB_R ; i : Integer) ;
BEGIN
  Writeln('Nbre d''or : ', V[i]);
  Writeln('Rang : ', i);
END;

{Programme principal }
BEGIN
   Init (U) ;
   Calcul (U, V, i) ;
   Affiche (V , i) ;
END.
```

Retenons

- Un algorithme ou un traitement est dit récurrent, s'il utilise un traitement itératif ou récursif pour engendrer un résultat qui peut dépendre de p résultats précédents.
- Un algorithme récurrent d'ordre 1 est un algorithme dont un résultat dépend du résultat précédent.
- Un algorithme récurrent d'ordre 2 est un algorithme dont un résultat dépend des deux résultats précédents.
- Un algorithme récurrent d'ordre 3 est un algorithme dont un résultat dépend des trois résultats précédents.
- Un algorithme récurrent d'ordre p est un algorithme dont un résultat dépend des p résultats précédents.

Exercice 1

Répondez par V (Vrai) ou F (Faux) à chacune des questions suivantes :

a. Un traitement récurrent d'ordre 1 fait référence :

- ☐ Au premier élément de la suite
- ☐ Au dernier élément de la suite
- ☐ A l'élément du milieu de la suite

b. Un traitement récurrent d'ordre 2 fait référence :

- ☐ Au deux premiers éléments de la suite
- ☐ Au deux derniers éléments de la suite
- ☐ A l'élément du milieu et à l'élément de fin de la suite

c. Que fait l'ensemble d'instructions suivantes :

```
A :='Fi' ;  B :='bonacci' ;  FIBO  :=1 ;
For  i :=1 To LENGTH ( Copy (A+B,6,3) ) * 4 Do
FIBO := FIBO * (i-1) + FIBO * (i-2) ;
```

- ☐ Calcule un terme de la suite de Fibonacci
- ☐ Calcule la somme des 12 premiers termes de la suite de Fibonacci
- ☐ Calcule une somme quelconque

Exercice 2

Écrivez un programme qui calcule et affiche la somme des carrés des n premiers entiers impairs.
Par exemple, pour n = 5
Le programme affiche :

n = 5 , la suite est : $1^2 + 3^2 + 5^2 + 7^2 + 9^2 = 165$.

Exercice 3

Ecrivez un programme qui affiche un triangle isocèle formé d'étoiles de N lignes (N est fourni au clavier) :

Exemple pour un nombre de lignes = 8

```
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
```

Exercice 4

Nous définissons la suite (x_n) avec $n \in N$ de la manière suivante :

$$x_0 = a \quad \text{et} \quad x_{n+1} = 4x_n (1 - x_n).$$

Écrivez un programme qui calcule x_n en fonction de a et de n donnés.

Exercice 5 : Calcul de la suite de Heron

Ecrivez un programme permettant de calculer les n premiers termes de la suite de Héron définie par :

$U_0 = x$

$U_{n+1} = (U_n /2) + (x / 2U_n)$

Où x est un réel positif.

Exercice 6

Nous rappelons que la fonction Random (i : Integer) renvoie un entier aléatoire compris entre 0 et i-1. L'appel de RANDOMIZE permet d'initialiser cette fonction.

Nous donnons le programme suivant :

```
PROGRAM mystere ;
Uses Crt ;
VAR   T:  array[1..20001]  Of Integer;
      U,S,i,n     :  Integer;
      coincide  :  Boolean;

PROCEDURE  X;
Begin
      Randomize  ;  {initialisation  de  la  fonction Random}
      For  i :=1  TO  20001  DO  T[i]:= 1+ Random(20000);
End;

{Programme principal}
BEGIN
      X;
      i:=1;     coincide:=false;
      Repeat
            i:= i+1;      S:=0 ;
            While  (S< i-1)  AND  NOT ( coincide) Do
            Begin
                  S:=S+1;
                  If  T[S]=T[i] Then  coincide:=TRUE
            End;
      Until  coincide=true;
      U:= i ;
      For  n:=1  to  i  Do
             Write (T[n] , '  ;  ');
      WriteLn ;
      Writeln ('U  =  ', U) ;
      Writeln ('S  =  ', S);
      Readln;
END.
```

1/ A la fin du programme, que contiennent les variables S et U ?
2/ Déterminez le rôle de ce programme.
3/ Vérifiez votre hypothèse en rajoutant, si nécessaire, des lignes de code au programme.

Exercice 7 : Deux suites

Nous définissons une suite double :

$$U_0 = 1, U_{n+1} = (U_n + V_n) / 2$$
$$V_0 = 2, V_{n+1} = \sqrt{U_{n+1} * V_n}$$

Nous admettons que les suites (U_n) et (V_n) sont adjacentes de limite $\sqrt{27}/\pi$

Écrivez un programme qui lit un entier n et affiche l'approximation du nombre π obtenue à partir de V_n .

Exercice 8

Calculez le $N^{ème}$ terme U_n de la suite de FIBONACCI qui est donnée par la relation de récurrence suivante :

$U_1=1$

$U_2=1$

$U_n=U_{n-1} + U_{n-2}$ (pour N>2)

Déterminez le rang N et la valeur U_n du terme maximal que l'on peut calculer si nous utilisons pour U_n :

- le type entier
- le type entier long .

Exercice 9 : Formule de Viète

Soient deux suites récurrentes u et v :

$V_0 = 0$ $\qquad V_{n+1} = \sqrt{\frac{1+V_n}{2}}$

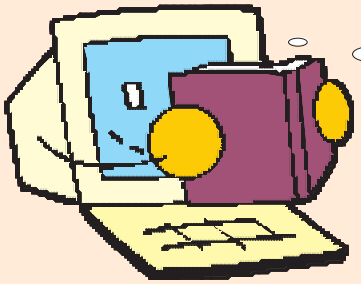
et $U_0 = 2$ $\qquad U_{n+1} = \frac{U_n}{V_n + 1}$

Vérifiez par un programme cette assertion.

Exercice 10 : Suite convergeant vers la racine de N

Soit la suite définie par Uo = N/2 et $U_{n+1} = (U_n + N / U_n) / 2$, où N est un entier naturel,

- En admettant sa convergence, vérifiez que sa limite est racine de N.
- Calculez ses termes successifs, tant que la valeur absolue de la différence de deux termes consécutifs dépasse 10^{-6}, c'est-à-dire $| U_{n+1} - U_n | > 10^{-6}$
- Affichez le nombre d'itérations effectuées.



Lecture

Leonardo Pisano

Fibonacci.

Leonardo Fibonacci (Pise, v. 1170 - v. 1250) est un mathématicien italien. Fibonacci (de son nom moderne), connu à l'époque sous le nom de Leonardo Pisano (**Léonard de Pise**), mais aussi de **Leonardo Bigollo** (bigollo signifiant voyageur), s'appelait en réalité **Leonardo Guilielmi.**

Biographie

Né à Pise, en Italie, son éducation s'est faite en grande partie en Afrique du Nord. Son père, Guilielmo Bonacci, gérait les marchés de la république de Pise en Algérie, en Tunisie et au Maroc. En 1202, il en rapporta les chiffres arabes et la notation algébrique (dont certains attribuent l'introduction à Gerbert d'Aurillac).

En 1202, il publie Liber Abaci (« Le livres des calculs »), un traité sur les calculs et la comptabilité fondée sur le calcul décimal à une époque où tout l'Occident utilisait encore les chiffres romains et calculait sur abaque. Ce livre est fortement influencé par sa vie dans les pays arabes ; il est d'ailleurs rédigé en partie de droite à gauche.

Par cette publication, Fibonacci introduit le système de notation indienne en Europe. Ce système est bien plus puissant et rapide que la notation romaine, et Fibonacci en est pleinement conscient. Il peina cependant à s'imposer avant plusieurs siècles. L'invention sera mal reçue car le public ne comprenait plus les calculs que faisaient les commerçants. En 1280, Florence interdit même l'usage des chiffres arabes par les banquiers. On jugea que le 0 apportait la confusion et des difficultés au point qu'ils appelèrent ce système cifra, qui signifie « code secret ».

Fibonacci est connu de nos jours pour un problème conduisant aux nombres et à la suite qui portent son nom, mais à son époque, ce sont surtout les applications de l'arithmétique au calcul commercial qui l'ont fait reconnaître : calcul du profit des transactions, conversion entre monnaies de différents pays. Son travail sur la théorie des nombres était ignoré de son vivant. Plus tard, des études peu sérieuses faites sur lui débouchèrent sur des usages ésotériques, que l'on retrouve même au niveau de certaines méthodes boursières (analyse technique). Le nom de Fibonacci, correspondant au « fils de Bonacci », lui a été attribué de manière posthume.

D'après le site : www. techno-science.net

Un petit historique du nombre d'or

Son nom

On le désigne par la lettre grecque λ (phi) en hommage au sculpteur grec Phidias (né vers 490 et mort vers 430 avant J.C) qui décora le Parthénon à Athènes. C'est Théodore Cook qui introduisit cette notation en 1914.

L'histoire

Il y a 10 000 ans : Première manifestation humaine de la connaissance du nombre d'or (temple d'Andros découvert sous la mer des Bahamas).

2800 avant J.C : La pyramide de Khéops a des dimensions qui mettent en évidence l'importance que son architecte attachait au nombre d'or.

V^{ème} siècle avant J.C (447-432 avant J.C) : Le sculpteur grec Phidias utilise le nombre d'or pour décorer le Parthénon à Athènes, en particulier pour sculpter la statue d'Athéna Parthénos . Il utilise également la racine carrée de 5 comme rapport.

III^{ème} siècle avant J.C : Euclide évoque le partage d'un segment en "extrême et moyenne raison" dans le livre VI des Eléments.

1498 : Fra Luca Pacioli, un moine professeur de mathématiques, écrit De divina proportione ("La divine proportion").

Au XIX^{ème} siècle : Adolf Zeising (1810-1876), docteur en philosophie et professeur à Leipzig puis Munich, parle de "section d'or" (der goldene Schnitt) et s'y intéresse non plus à propos de géométrie mais en ce qui concerne l'esthétique et l'architecture. Il cherche ce rapport, et le trouve (on trouve facilement ce qu'on cherche ...) dans beaucoup de monuments classiques. C'est lui qui introduit le côté mythique et mystique du nombre d'or.

Au début du XX^{ème} siècle : Matila Ghyka, diplomate roumain, s'appuie sur les travaux du philosophe allemand Zeising et du physicien allemand Gustav Theodor Fechner ; ses ouvrages L'esthétique des proportions dans la nature et dans les arts (1927) et Le Nombre d'or. Rites et rythmes pythagoriciens dans le développement de la civilisation occidentale (1931) insistent sur la prééminence du nombre d'or et établissent définitivement le mythe .

Au cours du XX^{ème} siècle : des peintres tels Dali et Picasso, ainsi que des architectes comme Le Corbusier, eurent recours au nombre d'or.

1945 : Le Corbusier fait bréveter son Modulor qui donne un système de proportions entre les différentes parties du corps humain.

La pyramide de Khéops

Le Parthénon d'Athènes

L'amour vache, Géricault

Où rencontre-t-on le nombre d'or

Il paraît que ...

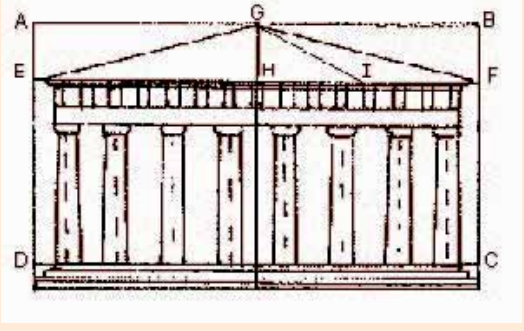
- Le rapport de la hauteur de la pyramide de Khéops par sa demi-base est le nombre d'or.
Il semble que ceci soit vrai, en dehors de toute considération ésotérique.
D'après Hérodote, des prêtres égyptiens disaient que les dimensions de la grande pyramide avaient été choisies telles que : "Le carré construit sur la hauteur verticale égalait exactement la surface de chacune des faces triangulaires"

- Le Parthénon d'Athènes fait apparaître un peu partout le nombre d'or.
Certains se sont employés à le chercher et l'ont bien sûr trouvé ! Et s'il avait cherché 2, l'auraient-ils trouvé ??

Le Parthénon s'inscrit dans un rectangle doré, c'est-à-dire tel que le rapport de la longueur à la hauteur était égal au nombre d'or.
Sur la figure : DC/DE = λ.

Sur la toiture du temple, GF/GI = λ.

Le rectangle GBFH est appelé rectangle Parthénon.



D'après le site : http://trucsmaths.free.fr/nombre_d_or.htm#historique

Chapitre 5

Les algorithmes d'arithmétique

Objectifs

- Acquérir des habilités de résolution de problèmes à travers l'apprentissage des algorithmes de calcul numérique,
- Proposer des solutions récursives à quelques problèmes arithmétiques.

Plan du chapitre

Chapitre 5

Les algorithmes d'arithmétique

I- Préambule

L'arithmétique est une branche des mathématiques qui étudie les relations entre les nombres. C'est aussi l'étude des nombres et des opérations élémentaires entre eux.
L'origine du mot arithmétique est :

- du grec : arithmétiké
  - arithmos : nombre
  - techné : art
- du latin : arithmetica

L'arithmétique faisait partie de la géométrie des Grecs anciens : études des nombres figurés. L'adoption de la numération de position à base dix fait réaliser d'immenses progrès, même si :

- La base 2 est universelle pour les ordinateurs
- La base 12 subsiste pour les heures
- La base 60 pour les angles et les durées en minutes et secondes

Exemples d'études arithmétiques :

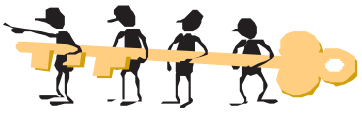
- Test de primalité
- Nombres parfaits
- Test de parité
- Somme de puissances
- Calcul de la factorielle
- Calcul de PGCD et PPCM, etc.

II- Calcul du PGCD (solution récursive)

II.1 Activité 1

Question 1 : Proposez une analyse, puis déduisez l'algorithme d'une fonction permettant de calculer le PGCD (le Plus Grand Commun Diviseur) de deux entiers positifs non nuls a et b, en utilisant la méthode de la différence.
Essayons cette méthode sur l'exemple suivant : a = 22 et b = 6

PGCD (22, 6) = PGCD (22-6, 6) = PGCD (16, 6)	Car 22 > 6
= PGCD (16-6, 6) = PGCD (10, 6)	Car 16 > 6
= PGCD (10-6, 6) = PGCD (4, 6)	Car 10 > 6
= PGCD (4, 6-4) = PGCD (4, 2)	Car 4 < 6
= PGCD (4-2, 2) = PGCD (2, 2)	Car 4 > 2
= 2	Car 2 = 2



Analyse de la fonction PGCD

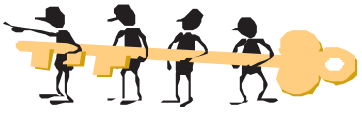
Résultat : PGCD de a et b
Traitement : Tant que a ≠ b Faire
Si a > b **Alors** a ← a - b
Sinon b ← b - a

❖Nous continuons à remplacer la plus grande valeur parmi a et b par la différence jusqu'à a = b

Algorithme de la fonction PGCD (solution itérative)

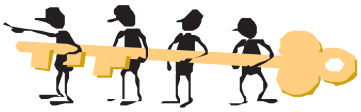
0) Fonction PGCD (a, b : Entier) : Entier
1) **Tant que** (a ≠ b) **Faire**
Si (a > b) **Alors** a ← a - b
Sinon b ← b - a
Fin Si
Fin Tant que
2) PGCD ← a
3) Fin PGCD

Question 2 : A partir de l'algorithme donné ci-dessus, dégagez une relation récursive de la fonction PGCD.



Essayons de dégager cette relation à partir de l'exemple présenté précédemment : a = 22 et b = 6
La relation récursive dégagée est la suivante :
Si a = b alors PGCD (a, b) = a
Sinon si a > b alors PGCD (a, b) = PGCD (a-b, b)
Sinon PGCD (a, b) = PGCD (a, b-a).

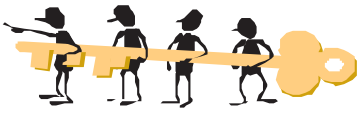
Question 3 : Déduisez l'algorithme récursif de la fonction PGCD.



0) Fonction PGCD (a, b : Entier) : Entier
1) **Si** (a = b) Alors PGCD ← a
Sinon Si (a > b) Alors PGCD ← FN PGCD (a-b,b)
Sinon PGCD ← FN PGCD (a,b-a)
Fin Si
2) Fin PGCD

II.2 Application

Traduisez et testez la solution récursive du problème permettant d'afficher le PGCD de deux entiers positifs et non nuls a et b donnés, en utilisant la méthode de la différence. Enregistrez votre programme sous le nom **PGCD_rec**.



```
Program Pgcd_diff;
Uses Crt;
Var
  a,b : Integer;
Procedure saisie (Var z : Integer);
 Begin
  Repeat
  Write('donner un entier positif et non nul : ');
  Readln(z);
  Until (z>0) ;
 End;
Function pgcd(n, m : Integer) : Integer;
 Begin
  If (n = m) Then pgcd := m
  Else If (n > m) Then pgcd := pgcd(n - m, m)
  Else pgcd := pgcd(n, m - n);
 End;

{Programme Principal}
 Begin
  Saisie(a);
  Saisie(b);
  Write('pgcd(',a,',',b,')= ', pgcd(a,b));
 End.
```

III- Calcul de A_n^p et C_n^p

III.1 Présentation

COMBINATOIRE DE P OBJETS PARMI N

Exemple : p = 3 et n = 5

PERMUTATIONS	**Ordre**	**Per = p!**	**6**
ARRANGEMENTS	**Ordre**	**A = n! / (n-p)!**	**60**
COMBINAISONS	**Sans ordre**	**C = A/P = n!/(n-p)!p!**	**10**

D'où

Arrangement de p éléments parmi n

C'est le nombre de permutations ordonnées possibles de p éléments parmi n
Exemple avec {a,b,c}: A(2,3) = 6
{a,b}, {b,a}, {a,c}, {c,a}, {b,c}, {c,b}

Combinaison de p éléments parmi n

C'est le nombre de permutations sans ordre possibles de p éléments parmi n
Exemple avec {a,b,c}: C(2,3) = 3
{a,b}, {a,c}, {b,c}

III.2 Calcul de A_n^p

Activité 1

D'après vos connaissances en Mathématiques,

1) Définissez un arrangement de p éléments d'un ensemble E de n éléments.

→ Un arrangement de p éléments d'un ensemble E à n éléments est un p-uplet d'éléments distincts de E.

2) Donnez la formule mathématique du nombre d'arrangements, noté $\mathbf{A_n^p}$

→ Le nombre d'arrangements de p éléments de l'ensemble E est représenté par la notation suivante : $\mathbf{A_n^p} = n(n-1)(n-2)\ldots(n-p+1)$

Ou encore $\mathbf{A_n^p} = \dfrac{n!}{(n-p)!}$

3) Quelles conditions doivent vérifier les entiers n et p pour valoir la définition et la formule données précédemment ?

→ n et p sont des entiers qui vérifient la condition suivante : $1 \leq p \leq n$

4) Proposez une analyse, puis déduisez les algorithmes correspondants au problème permettant de chercher puis d'afficher le $\mathbf{A_n^p}$ de deux entiers donnés n et p, avec $(1 \leq p \leq n)$.

a) Analyse du problème

Résultat : Affichage de $\mathbf{A_n^p}$. L'affichage sera représenté sous la forme suivante **A(n, p)** : Ecrire ("A(", n, ",", p, ") = ", **FN** Arrange (n,p))

Données : deux entiers n et p, introduits grâce à la procédure **Saisie**

b) Algorithme du programme principal et codification des objets

0) Début Arrangement
1) **Proc** Saisie (n, p)
2) Ecrire ("A (", n, ",", p, ") = ", **FN** Arrange (n,p))
3) Fin Arrangement

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n,p	Entier	Données
Saisie	Procédure	Permet de saisir n et p
Arrange	Fonction	Permet de calculer A_n^p

c) Analyse de la fonction Arrange

Résultat : Arrange
Traitement :
Arrange ← ar
[ar ← 1]
Pour i de a à a-b+1 (pas = -1) **Faire**
ar ← ar * i

} Structure itérative complète utilisée pour calculer le résultat de l'opération a(a-1)(a-2)…(a-b+1)

d) Algorithme de la fonction Arrange

0) Fonction Arrange (a, b : Entier) : Réel
1) ar ← 1
Pour i de a à a-b+1 (pas = -1) **Faire**
ar ← ar * i
FinPour
2) Arrange ← ar
3) Fin Arrange

e) Traduction en Pascal de la fonction Arrange

```
Function Arrange (a, b : Integer) : Real;
Var
  i, ar : Integer;
Begin
   ar := 1;
   For i := a Downto a-b+1 Do
      ar := ar*i;
   Arrange := ar;
End;
```

Questions :

1) Avec l'assistance de votre enseignant, traduisez et testez la solution du problème permettant de chercher puis d'afficher le A_n^p de deux entiers donnés n et p, avec $1 \leq p \leq n$. Enregistrez votre programme sous le nom **Arrang_1.**
2) Exécutez le programme obtenu pour trouver les valeurs suivantes :

A_5^3 = ... A_7^4 = ... A_{10}^2 = ... A_{13}^1 = ...

3) En essayant plusieurs jeux d'exécution :

◊Ecrivez plus simplement la valeur A_n^1 = ...

◊Comparez les valeurs A_n^n = ... et A_n^{n-1} = ...

III.3 Calcul de C_n^p

Activité 2

D'après vos connaissances en Mathématiques,
1) Définissez une combinaison de p éléments d'un ensemble E fini, non vide et de n éléments.

→ Une combinaison de p éléments d'un ensemble E de n éléments est une partie de E formée par p éléments.

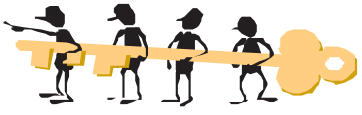
2) Comment représenter le nombre de combinaisons de p éléments de l'ensemble E ? Donnez sa formule mathématique.

→ Le nombre de combinaisons de p éléments de l'ensemble E est représenté par la notation suivante : $C_n^p = \frac{A_n^p}{p!} = \frac{n!}{p!(n-p)!}$

3) Quelles conditions doivent vérifier les entiers n et p pour valoir la définition et la formule données précédemment ?

→ n et p sont des entiers qui vérifient la condition suivante : $0 \leq p \leq n$

Proposez une analyse, puis déduisez les algorithmes correspondants au problème permettant de chercher puis d'afficher le C_n^p de deux entiers donnés n et p, avec $0 \leq p \leq n$.



a) Analyse du problème

Résultat : Affichage de C_n^p . L'affichage sera représenté sous la forme suivante C(n,p) :

Ecrire ("C (", n, ",", p, ") = ", FN Comb (n,p))

Données : deux entiers n et p introduits grâce à la procédure Saisie

b) Algorithme du programme principal et codification des objets

0) Début Combinaison_1
1) **Proc** Saisie (n,p)
2) Ecrire ("C(", n, ",", p, ") = ", **FN** Comb (n,p))
3) Fin Combinaison_1

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n,p	Entier	Données
Saisie	Procédure	Permet de saisir n et p
Comb	Fonction	Permet de calculer C_n^p

c) Analyse de la fonction Comb

Résultat : Comb

Traitement : Comb ← **FN** Factorielle(n) / (**FN** Factorielle(p) * **FN** Factorielle(n-p))

NB : La fonction Factorielle a été déjà l'objet d'un apprentissage précédent.

d) Algorithme de la fonction Comb

0) Fonction Comb (n, p : Entier) : Réel
1) Comb ← **FN** Factorielle (n) / (FN Factorielle (p) * FN Factorielle (n-p))
2) Fin Comb

Questions :

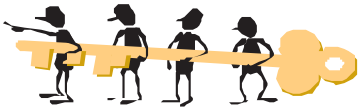
1) Avec l'assistance de votre enseignant, traduisez et testez la solution du problème permettant de chercher puis d'afficher le C_n^p de deux entiers donnés n et p, avec $0 \leq p \leq n$. Enregistrez votre programme sous le nom **Comb_1.**
2) Faîtes les modifications nécessaires pour proposer une solution utilisant la fonction Arrange traitée précédemment. Enregistrez votre nouveau programme sous le nom **Comb_2**.
3) Exécutez le programme obtenu pour trouver les valeurs suivantes :

C_{11}^1 = ... C_{32}^0 = ... C_{21}^{21} = ... C_9^8 = ...

III.4 Application

Nous voulons utiliser un procédé récursif pour chercher puis afficher le C_n^p de deux entiers donnés n et p, avec $0 \leq p \leq n$.

- Proposez une analyse à la fonction Comb,
- Déduisez l'algorithme correspondant,
- Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Comb_rec**.



a) Analyse de la fonction récursive Comb

Résultat : Comb

Traitement : D'après vos connaissances en Mathématiques, vous pouvez dégager la relation suivante :

Si p = 0 **ou** p = n **alors** $C_n^p = 1$

Sinon $C_n^p = C_{n-1}^{p-1} + C_{n-1}^{p}$

D'où **Si** (p = 0) **OU** (p = n) **alors** Comb ← 1
Sinon Comb ← FN Comb (n-1, p) + FN Comb (n-1, p-1)

b) Algorithme de la fonction récursive Comb

0) Fonction Comb (n, p : Entier) : Réel
1) **Si** (p = 0) **ou** (p = n) **alors** Comb ← 1
Sinon Comb ← FN Comb (n-1, p) + FN Comb (n-1, p-1)
FinSi
2) Fin Comb

c) Traduction en Pascal de la fonction récursive Comb

```
Function Comb(n, p : integer) : real;
Begin
      If (p = 0) Or (p = n) Then Comb := 1
      Else Comb := Comb(n-1,p) + Comb(n-1,p-1);
End;
```

IV- Quelques règles de divisibilité

Dans cette partie du chapitre, les problèmes ne seront pas résolus directement avec l'opérateur arithmétique **Mod**. Nous allons, par contre, analyser quelques problèmes répondant aux règles de divisibilité appropriées.

IV.1 Définitions

Un entier n est ***divisible*** par un entier m, si le reste de la division euclidienne de n par m est nul.

Une règle ***de divisibilité*** est une séquence d'opérations simples qui permet de reconnaître rapidement si un entier est divisible par un autre sans qu'il soit nécessaire d'effectuer la division. Ces règles sont généralement applicables pour les grands nombres.

IV.2 Divisibilité par 3

Activité 3

Soit les entiers suivants : 6, 42, 191, 39, 41, 30, 20, 13 et 8.
Dans la liste d'entiers donnée ci-dessus, encerclez ceux qui sont divisibles par 3.

Réponse :

(6), (42), 19, (39), 41, (30), 20, 13 et 8

Constatations :

Pouvez-vous trouver une propriété commune entre les entiers encerclés et expliquant la raison pour laquelle ils sont tous divisibles par 3.

Réponse :

Nous constatons que la somme des chiffres de chacun d'eux est divisible par 3 :

6 est divisible par 3,
$4 + 2 = 6 \Rightarrow$ 6 est divisible par 3,
$3 + 9 = 12 \Rightarrow 1 + 2 = 3$ est divisible par 3,
$3 + 0 = 3 \Rightarrow$ 3 est divisible par 3.
Par contre :
$1 + 9 + 1 = 1 + 1 = 2$ n'est pas divisible 3,
$4 + 1 = 5$ n'est pas divisible par 3,
et de même pour 20, 13 et 8.

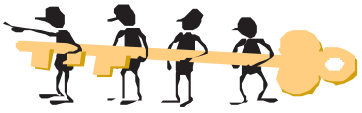
Conclusion :

Un entier est divisible par 3, si la somme des chiffres qui le composent est divisible par 3.

Application :

Nous proposons de vérifier si un entier n donné est divisible par 3, en utilisant la règle de divisibilité citée ci-dessus.

1- Proposez une analyse modulaire au problème,
2- Déduisez les algorithmes des modules dégagés,
3- Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Divi_3**.



a) Analyse du problème

Résultat : Ecrire (s, **FN** Div_3 (s))

Données : un nombre entier représenté sous forme d'une chaîne s, entrée grâce à la procédure Saisie.

b) Algorithme du programme principal et codification des objets

0) Début Divisible_3
1) **Proc** Saisie (s)
2) Ecrire (s, **FN** Div_3 (s))
3) Fin Divisible_3

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
s	Chaîne	Représente le nombre à tester
Saisie	Procédure	Permet de saisir s
Div_3	Fonction	Permet de retourner un message indiquant la divisibilité ou non du nombre donné par 3.

c) Analyse de la fonction Div_3

Résultat : Div_3

Traitement :

- Si la somme des chiffres composant le nombre à tester est divisible par 3 alors ce nombre l'est aussi sinon il n'est pas divisible par 3.

```
Si Somme Mod 3 = 0 alors
        Div_3 ← " est divisible par 3."
Sinon
        Div_3 ← " n'est pas divisible par 3."
Finsi
```

- [Somme ← 0]

Utiliser une structure itérative complète pour faire un parcours total de la chaîne numérique ch afin de cumuler la somme de ses chiffres.

```
Pour i de 1 à Long (ch) Faire
    Valeur (ch[i], nb, e)
    Somme ← Somme + nb
FinPour
```

d) Algorithme de la fonction Div_3

0) Fonction Div_3 (ch : Chaîne) : Chaîne
1) Somme ← 0
   Pour i de 1 à Long (ch) **Faire**
      Valeur (ch[i], nb, e)
      Somme ← Somme + nb
   FinPour
2) **Si** Somme Mod 3 = 0 **alors** Div_3 ← " est divisible par 3."
   Sinon Div_3 ← " n'est pas divisible par 3."
   Finsi
3) Fin Div_3

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
Somme	Entier	Somme des chiffres de ch
nb	Entier	Conversion en numérique d'un caractère de ch
e	Entier	Indicateur d'erreurs lors de la conversion en numérique
i	Entier	Compteur

e) Traduction en Pascal du problème Divisible_3

```
Program Divisible_3;
Uses Crt;
Var s : String;

Procedure saisie (Var ch : String);
var num : Longint; e: Integer;
Begin
 Repeat
   Write('donner un entier : ');
   Readln(ch);
   Val(ch, Num, e);
 Until (e = 0)
End;

Function Div_3(ch : String) : String;
Var e, nb, somme, i : Integer;
Begin
  somme := 0;
  For i := 1 To Length(ch) Do
  Begin
    Val(ch[i], nb, e);
    somme := somme + nb
  End;
  If Somme Mod 3 = 0 Then div_3 := ' est divisible par 3.'
  Else div_3 := ' n''est pas divisible par 3.';
End;
```

```
{Programme Principal}
Begin
  Saisie(s);
  Write(s, Div_3(s));
End.
```

IV.3 Divisibilité par 4

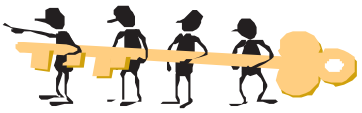
Un entier est divisible par 4, si le nombre composé des deux derniers chiffres est divisible par 4.

Exemples : 6287 n'est pas divisible par 4 (car 87 n'est pas divisible par 4), 4120 l'est (car 20 = 5 x 4, donc divisible par 4).

Application :

Nous proposons de vérifier si un entier n donné est divisible par 4, en utilisant la règle de divisibilité citée ci-dessus.

1- Proposez une analyse modulaire au problème,
2- Déduisez les algorithmes dégagés,
3-Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Divi_4.**



a) Analyse du problème

Résultat : Ecrire (n, **FN** Div_4 (n))

Données : un entier n, entré grâce à la procédure Saisie.

b) Algorithme du programme principal et codification des objets

0) Début Divisible_4
1) **Proc** Saisie (n)
2) Ecrire (n, **FN** Div_4 (n))
3) Fin Divisible_4

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n	Entier Long	Nombre à tester
Saisie	Procédure	Permet de saisir n
Div_4	Fonction	Permet de retourner un message indiquant la divisibilité ou non de n par 4.

c) Analyse de la fonction Div_4

Résultat : Div_4

Traitement :

➢ Si le nombre composé des deux derniers chiffres du nombre à tester est divisible par 4 alors le nombre initial l'est aussi sinon ce dernier n'est pas divisible par 4.

Si (m Mod 100) Mod 4 = 0 **Alors**
Div_4 ← " est divisible par 4."
Sinon
Div_4 ← " n'est pas divisible par 4."

d) Algorithme de la fonction Div_4

0) Fonction Div_4 (m : Entier Long) : Chaîne
1) **Si** (m Mod 100) Mod 4 = 0 **Alors** Div_4 ← " est divisible par 4."
Sinon Div_4 ← " n'est pas divisible par 4."
Finsi
2) Fin Div_4

e) Traduction en Pascal de la fonction Div_4

```
Function Div_4(m : Longint) : String;
Begin
  If ((m Mod 100) Mod 4) = 0 Then div_4 := ' est divisible par 4.'
  Else div_4 := ' n''est pas divisible par 4.';
End;
```

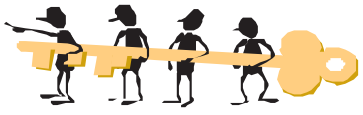
IV.4 Divisibilité par 5

Un entier est divisible par 5, si son chiffre des unités est égal à 0 ou à 5.

Exemples : 6287 n'est pas divisible par 5, car $7 \notin \{0, 5\}$
4120 est divisible par 5, car $0 \in \{0, 5\}$
3635 est divisible par 5, car $5 \in \{0, 5\}$

Application :
Nous voulons vérifier si un entier n donné est divisible par 5, en utilisant la règle de divisibilité citée ci-dessus.

1 - Proposez une analyse modulaire au problème,
2 - Déduisez les algorithmes dégagés,
3 - Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Divi_5**.



a) Analyse du problème

Résultat : Ecrire (n, **FN** Div_5 (n))
Données : un entier n, entré grâce à la procédure Saisie.

b) Algorithme du programme principal et codification des objets

0) Début Divisible_5
1) **Proc** Saisie (n)
2) Ecrire (n, **FN** Div_5 (n))
3) Fin Divisible_5

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
n	Entier Long	Nombre à tester
Saisie	Procédure	Permet de saisir n
Div_5	Fonction	Permet de retourner un message indiquant la divisibilité ou non de n par 5.

c) Analyse de la fonction Div_5

Résultat : Div_5
Traitement :
Si (2*m) Mod 10 = 0 **Alors**
Div_5 ← " est divisible par 5."
Sinon
Div_5 ← " n'est pas divisible par 5."

d) Algorithme de la fonction Div_5

0) Fonction Div_5 (m : Entier Long) : Chaîne
1) **Si** (2*m) Mod 10 = 0 **Alors** Div_5 ← " est divisible par 5."
Sinon Div_5 ← " n'est pas divisible par 5."
Finsi
2) Fin Div_5

e) Traduction en Pascal de la fonction Div_5

```
Function Div_5(m : Longint) : String;
Begin
  If (2*m) Mod 10 = 0
  Then div_5 := ' est divisible par 5.'
  Else div_5 := ' n''est pas divisible par 5.';
End;
```

IV.5 Autres règles de divisibilité

◈ Un entier est divisible par 2, si son chiffre des unités est divisible par 2.

Exemples : 5287 n'est pas divisible par 2
136 est divisible par 2
94618 est divisible par 2.

- Un entier est divisible par 9, si la somme de ses chiffres est divisible par 9.
- Un entier est divisible par 10, si son chiffre des unités est égal à 0.
- Un entier est divisible par 25, si le nombre composé des deux derniers chiffres est divisible par 25…

Questions :

1) Cherchez d'autres règles de divisibilité.
2) Soit n = 571u où u désigne le chiffre des unités dans l'écriture décimale de n. Ecrivez un programme Pascal permettant d'afficher les valeurs respectives de u et de n pour lesquelles l'entier n sera divisible par 4.
3) Soit n = 10c8u où c et u désignent respectivement le chiffre des centaines et le chiffre des unités dans l'écriture décimale de n. Ecrivez un programme Pascal permettant de déterminer puis d'afficher les couples (c, u) et l'entier n pour lesquels ce dernier sera divisible par 3.
4) Ecrivez un programme Pascal permettant de déterminer puis d'afficher tous les diviseurs d'un entier naturel non nul n donné.

V. Conversions entre bases de numération

V.1 Définition

Un **système de numération** est une méthode de comptage fondée sur une base de numération qui est un entier supérieur ou égal à 2. Soit N une base de numération, le système sera doté de N chiffres allant de 0 à N-1.

V.2 Exemples de bases de numération

- **Base 2** :

Le système de numération à base 2 (binaire) est un moyen de représenter les nombres avec 2 chiffres. Les chiffres utilisés sont 0 et 1.

- **Base 8** :

Dans le système octal, seuls les chiffres 0, 1, 2, 3, 4, 5, 6 et 7 sont utilisés pour coder un triplet de bits.

- **Base 10** :

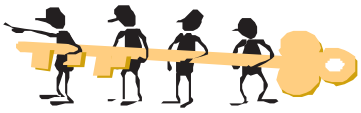
Le système de numération à base 10 est un moyen de représenter les nombres avec 10 symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. C'est le système couramment utilisé.

- **Base 16** :

Les nombres binaires étant de plus en plus longs, il a fallu introduire une nouvelle base : la base hexadécimale. Elle consiste à compter sur la base 16. Les chiffres utilisés sont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ainsi que les lettres A, B, C, D, E et F pour représenter respectivement les valeurs de 10, 11, 12, 13, 14 et 15.

Activité :

Nous avons vu en 3ème année comment convertir un nombre décimal en binaire. Proposez une analyse modulaire et déduisez les algorithmes permettant de réaliser ce traitement.



a) Analyse du problème

Résultat : La procédure Afficher (T, c) permet d'afficher le tableau contenant les restes des divisions successives par 2,

Traitement : Le remplissage du tableau des restes est effectué par la procédure Chercher (N, T, c)

Données : N est l'entier à convertir, il est entré grâce à la procédure Saisir (N)

b) Algorithme du programme principal

0) Début Conversion_10_2
1) **Proc** Saisir (N)
2) **Proc** Chercher (N, T, C)
3) **Proc** Afficher (T, C)
4) **Fin** Conversion_10_2

Tableau de codification des nouveaux types

Types
Binaire = 0,1 Tab = Tableau de 100 binaire

Tableau de codification des objets globaux

Objets	Types / Nature	Rôle
N	Entier	Nombre à convertir en binaire
C	Entier	Nombre de divisions par 2
T	Tab	Tableau contenant les restes des divisions successives par 2
Saisir	Procédure	Permet de saisir N
Chercher	Procédure	Permet de remplir T par les restes des divisions successives par 2
Afficher	Procédure	Permet d'afficher les éléments de T

c) Analyse de la procédure Chercher

Résultat : Pour remplir le tableau Reste, nous devons :

- [c ← 0]
- Procéder à des divisions successives par 2 qui s'arrêtent lorsque N = 0. Pour chaque division nous devons : Incrémenter le nombre de divisions c de 1, chercher le reste

de la division de N par 2 et le ranger dans un tableau puis changer la valeur de N par N Div 2.

```
Répéter
   C ← C + 1
   Reste [C] ← n mod 2
   n ← n div 2
Jusqu'à (n = 0)
```

d) Algorithme de la procédure Chercher

```
0) Procédure Chercher (n : entier ; Var Reste : Tab ; Var C : entier)
1) C ← 0
      Répéter
         C ← C + 1
         Reste [C] ← n mod 2
         n ← n div 2
      Jusqu'à (n = 0)
2) Fin Chercher
```

Nous allons voir dans ce qui va suivre d'autres algorithmes de conversion entre bases de numération.

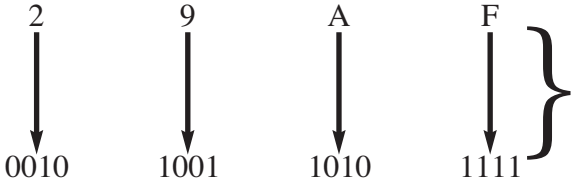
V.3 Conversion d'un nombre hexadécimal en binaire

Nous proposons de convertir un nombre hexadécimal en base 2.

Exemple :

Soit à convertir en binaire le nombre hexadécimal n = 29AF

Nous allons procéder de la manière suivante :



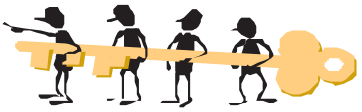
Chaque chiffre de n sera représenté sur 4 bits.

Le nombre binaire obtenu est : **10100110101111**

⇒ En effet, un chiffre hexadécimal correspond à 4 bits. Il suffit donc, de convertir un à un chaque chiffre hexadécimal en binaire et de les mettre les uns à la suite des autres.

Questions :

1) Proposez une analyse modulaire au problème,
2) Déduisez les algorithmes des modules dégagés,
3) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Hex_Bin**.



a) Analyse du problème

Résultat : Ecrire ("(", Hex, ")16 = (", **FN** Conv_Bin(Hex), ")2")

Données : une chaîne de caractères Hex, entrée grâce à la procédure Saisie. Hex ne peut contenir que des chiffres allant de 0 à 9 et de lettres allant de A à F.

b) Algorithme du programme principal

0) Début Hex_Bin
1) Proc Saisie (Hex)
2) Ecrire ("(", Hex, ")16 = (", **FN** Conv_Bin(Hex), ")2")
3) Fin Hex_Bin

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
Hex	Chaîne de caractères	Nombre hexadécimal à convertir en binaire
Saisie	Procédure	Permet de saisir Hex
Conv_Bin	Fonction	Permet de retourner une chaîne représentant la conversion en binaire du nombre hexadécimal Hex.

c) Analyse de la fonction Conv_Bin

Résultat : Conv_Bin

Traitement :

- Conv_Bin ← CH_Bin
- Pour supprimer les zéros superflus du début de CH_Bin :

  Tant que CH_Bin[1] = "0" **Faire**
  Efface (CH_Bin,1,1)

- [CH_Bin ← ""]

Ensuite, un parcours total sera effectué sur la chaîne CH_Hex, pour convertir chaque caractère représentant un chiffre hexadécimal en son correspondant binaire :

Pour i **de** 1 à Long (CH_Hex) **Faire**
CH_Bin ← CH_Bin + Bin_Car (CH_Hex[i])

d) Algorithme de la fonction Conv_Bin

0) Fonction Conv_Bin (CH_Hex : Chaîne) : Chaîne
1) CH_Bin ← ""
Pour i de 1 à Long (CH_Hex) **Faire**
CH_Bin ← CH_Bin + FN Bin_Car (CH_Hex[i])
FinPour

2) **Tant que** CH_Bin[1] = "0" **Faire**
Efface (CH_Bin,1,1)
Fin Tant Que
3) Conv_Bin ← CH_Bin
4) Fin Conv_Bin

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
CH_Bin	Chaîne	Conversion de CH_Hex en binaire
i	Entier	Compteur
Bin_car	Fonction	Déterminer une chaîne de quatre caractères, représentant la conversion en binaire d'un chiffre hexadécimal

e) Analyse de la fonction Bin_Car

Résultat : Bin_Car
Traitement :

- Bin_Car ← CH
- [CH ← "0000"]

Ensuite, des divisions successives par 2 du chiffre hexadécimal seront réalisées, les restes seront affectés directement dans leurs positions dans CH. Cette action est répétée jusqu'à ce que N = 0 :

```
i ← 4
Répéter
  R ← N Mod 2
  Convch (R, Ch_R)
  Ch[i] ← Ch_R[1]
  N ← N Div 2
  i ← i-1
Jusqu'à N = 0
```

- Le correspondant numérique du chiffre hexadécimal est obtenu de deux manières, selon les cas suivants :
  Si le chiffre est numérique, il suffit d'utiliser la procédure prédéfinie Valeur, **sinon** nous utiliserons la formule suivante : Code Ascii de la lettre - 55

Exemple : Le correspondant numérique du chiffre hexadécimal
A = Ord('A') - 55 = 65 - 55 = 10

f) Algorithme de la fonction Bin_Car

```
0) Fonction Bin_Car (Symb : Caractère) : Chaîne
1) Si Non (Symb Dans ["0".."9"])
   Alors N ← Ord (Symb) - 55
   Sinon Valeur (Symb, N, E)
   FinSi
```

2) CH ← "0000"
   i ← 4
   Répéter
      R ← N Mod 2
      Convch (R, Ch_R)
      Ch[i] ← Ch_R[1]
      N ← N Div 2
      i ← i-1
   Jusqu'à N = 0
3) Bin_Car ← CH
4) Fin Bin_Car

Tableau de codification des objets locaux

Objets	Types / Nature	Rôle
CH	Chaîne[4]	Représente la conversion en binaire d'un chiffre hexadécimal
i	Entier	Compteur
N	Entier	Correspondant numérique du chiffre hexadécimal
E	Entier	Indicateur d'erreurs de la conversion numérique
R	Entier	Reste de la division euclidienne par 2
Ch_R	Chaîne[1]	Représente le reste sous forme d'une chaîne

g) Traduction en Pascal de la fonction Bin_Car

```
Function Bin_Car (Symb : Char) : String ;
Var
  Ch : String[4];
  Ch_R : String[1];
  N, E, R, i : Integer;
Begin
   If Not (Symb In ['0'..'9'])
   Then N := Ord (Upcase(Symb)) - 55
   Else Val(Symb, N, E);
   Ch := '0000';
   i := 4;
   Repeat
      R := N Mod 2;
      Str (R, Ch_R);
      Ch[i] := Ch_R[1];
      N := N Div 2;
      i := i-1;
   Until N = 0;
   Bin_Car := Ch;
End;
```

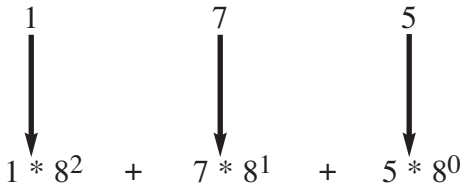
V.4 Conversion d'un nombre octal en décimal

Nous voulons convertir un nombre octal en base 10.

Exemple :

Soit à convertir en décimal le nombre octal n = 175

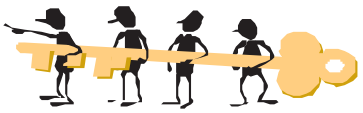
Nous allons procéder de la manière suivante :



Le nombre décimal obtenu est : 125

Questions :

a) Proposez une analyse modulaire au problème,
b) Déduisez les algorithmes des modules dégagés,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Oct_Dec**.



a) Analyse du problème

Résultat : Ecrire ("(", Oct, ")8 = (", **FN** Conv_Bin(Oct), ")10")

Données : un entier Oct, entré par le biais de la procédure Saisie. Oct ne peut contenir que des chiffres allant de 0 à 7.

b) Algorithme du programme principal

0) Début Oct_Dec
1) **Proc** Saisie (Oct)
2) Ecrire ("(", Oct, ")8 = (", **FN** Conv_Bin(Oct), ")10")
3) Fin Oct_Dec

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
Oct	Entier Long	Nombre octal à convertir en décimal
Saisie	Procédure	Permet de saisir Oct
Conv_Dec	Fonction	Permet de retourner un entier représentant la conversion en décimal du nombre octal Oct.

c) Analyse de la fonction Conv_Dec

Résultat : Conv_Dec

Traitement :

- Conv_Dec ← Dec
- [Dec ← 0]

Chaque chiffre du nombre octal sera multiplié par 8 élevé à une puissance relative à son rang, en appelant la fonction Puiss_8, puis, ajouté à Dec. Nous allons donc, utiliser la structure itérative complète :

Pour i de Long (CH_Oct) à 1 (pas = -1) **Faire**
Valeur (Ch_Oct[i], N, e)
Dec ← Dec + N * FN Puiss_8 (Long (Ch_Oct) - i)

- Pour pouvoir manipuler les chiffres du nombre octal, nous pouvons le convertir en une chaîne de caractères : ConvCh (N_Oct, CH_Oct)

d) Algorithme de la fonction Conv_Dec

0) Fonction Conv_Dec (N_Oct : Entier Long) : Entier Long
1) ConvCh (N_Oct, CH_Oct)
2) Dec ← 0
Pour i de Long (CH_Oct) à 1 (pas = -1) **Faire**
Valeur (Ch_Oct[i], N, e)
Dec ← Dec + N * FN Puiss_8 (Long (Ch_Oct) - i)
FinPour
3) Conv_Dec ← Dec
4) Fin Conv_Dec

Tableau de codification des objets locaux

Objets	Types / Nature	Rôle
Ch_Oct	Chaîne	Conversion du nombre octal en chaîne
Dec	Entier Long	Conversion du nombre octal en décimal
i	Entier non signé	Compteur
N	Entier	Conversion numérique d'un chiffre de Ch_Oct
e	Entier	Indicateur d'erreurs de la conversion numérique
Puiss_8	Fonction	Permet d'élever 8 à une puissance

e) Traduction en Pascal du problème Oct_Dec

```
Program Oct_Dec;
Uses Crt;
Var Oct : Longint;

Procedure Saisie(Var N_Oct : Longint);
Var
  trouve : Boolean;
  Ch_Oct : String;
  i : Word;
```

```
Begin
   trouve := True;
   While trouve Do
   Begin
      Write('Donner un nombre octal : ');
      Readln(N_Oct);
      Str(N_Oct, Ch_Oct);
      i := 1;
      While (Ch_Oct[i] In ['0'..'7']) And (i <=Length(Ch_Oct)) Do
        i := i + 1;
      If i > Length(Ch_Oct) Then Trouve := false ;
   End;
End;

Function Puiss_8 (M : Integer) : Longint;
Var
   P : Longint;
   j : Integer;
Begin
   P := 1;
   For j := 1 To M Do P := P * 8;
     Puiss_8 := P;
End;

Function Conv_Dec (N_Oct : Longint) : Longint;
Var
  i, e, N : Integer;
  Dec : Longint;
  Ch_Oct : String;
Begin
   Str(N_Oct, Ch_Oct);
   Dec := 0;
   For i := Length(CH_Oct) DownTo 1 Do
   Begin
      Val(Ch_Oct[i],N,e);
      Dec := Dec + N * Puiss_8(Length(Ch_Oct) - i);
   End;
   Conv_Dec := Dec;
End;

{Programme Principal}
Begin
   Saisie(Oct);
   Write('(', Oct, ')8 = (', Conv_Dec(Oct), ')10');
End.
```

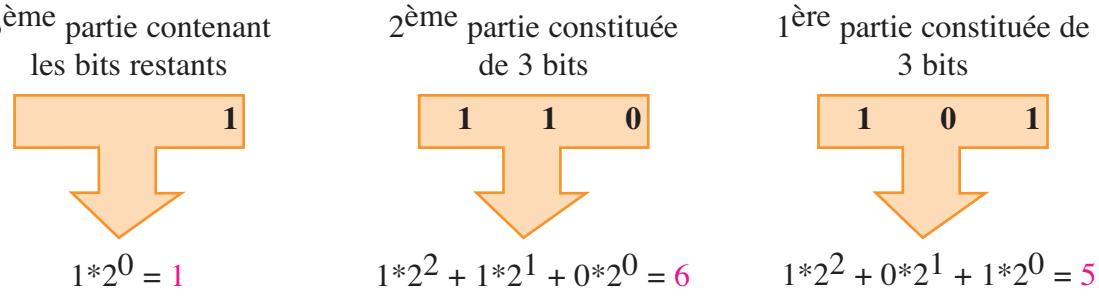
V.5 Conversion d'un nombre binaire en octal

Nous proposons de convertir un nombre binaire en base 8.

Exemple :

Soit à convertir en octal le nombre binaire x = 1110101

Nous allons procéder de la manière suivante :



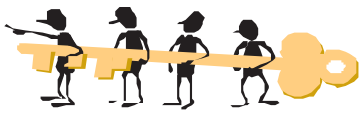
Le nombre octal obtenu est : 165

Questions :

a) Proposez une analyse modulaire au problème,

b) Déduisez les algorithmes des modules dégagés,

c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Bin_Oct.**



a) Analyse du problème

Résultat : Ecrire ("(", Bin, ")2 = (", **FN** Conv_Oct(Bin), ")8")

Données : une chaîne de caractères représentant le nombre binaire Bin , entrée par le biais de la procédure Saisie. Bin ne peut contenir que des "0" ou "1".

b) Algorithme du programme principal

0) Début Bin_Oct
1) **Proc** Saisie (Bin)
2) Ecrire ("(", Bin, ")2 = (", **FN** Conv_Oct (Bin), ")8")
3) Fin Bin_Oct

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
Bin	Chaîne	Nombre binaire à convertir en octal
Saisie	Procédure	Permet de saisir Bin
Conv_Oct	Fonction	Permet de retourner un entier représentant la conversion en octal du nombre binaire Bin.

c) Analyse de la fonction Conv_Oct

Résultat : Conv_Oct

Traitement :

- Conv_Oct ← Oct
- [N ← Long (Ch_Bin) Div 3, Oct ← "", L ← Long (Ch_Bin)]
- Extraire une partie de trois bits à partir du dernier bit, en commençant de la droite de CH_Bin; convertir chaque partie en octal en utilisant une fonction Puiss_2; convertir ce résultat en chaîne pour pouvoir le concaténer à l'objet destination de la conversion en octal Oct.

Nous allons utiliser une structure itérative complète dont le nombre d'itérations est égal à N :

```
Pour i de 1 à N Faire
   Ch1 ← Sous_chaîne (Ch_Bin, L - 2, 3)
   A_oct ← 0
   Pour j de 1 à 3 Faire
     Valeur(ch1[j], a, e)
     A_oct ← A_oct + a * FN Puiss_2 (3 - j)

   Convch (A_oct, ch2)
   Oct ← ch2 + Oct
    L ← L - 3
```

- Ajouter, si nécessaire, des "0" à gauche de CH_Bin, de façon à obtenir une longueur multiple de 3 :

```
Tant Que Long (Ch_Bin) Mod 3 ≠ 0 Faire
        Ch_Bin ← "0" + Ch_Bin
```

d) Algorithme de la fonction Conv_Oct

```
0) Fonction Conv_Oct (Ch_Bin : Chaîne) : chaîne
1) Tant Que Long (Ch_Bin) Mod 3 ≠ 0 Faire
        Ch_Bin ← "0" + Ch_Bin
   Fin Tant Que
2) N ← Long (Ch_Bin) Div 3
   Oct ← "", L ← Long (Ch_Bin)
   Pour i de 1 à N Faire
     Ch1 ← Sous_chaîne (Ch_Bin, L-2, 3)
     A_oct ← 0
     Pour j de 1 à 3 Faire
       Valeur (ch1[j], a, e)
       A_oct ← A_oct + a * FN Puiss_2 (3-j)
     FinPour
     Convch (A_oct, ch2)
     Oct ← ch2 + Oct
     L ← L - 3
    FinPour
3) Conv_Oct ← Oct
4) Fin Conv_Oct
```

Tableau de codification des objets locaux

Objets	Types / Nature	Rôle
Ch1, Ch2, oct	Chaîne	Chaînes intermédiaires
i, j, L	Entier	Compteurs
e	Entier	Indicateur d'erreurs de la conversion numérique
a, A_oct	Entier	Variables intermédiaires
Puiss_2	Fonction	Permet d'élever 2 à une puissance

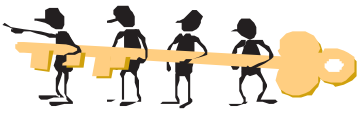
e) Traduction en Pascal de la fonction Conv_Oct

```
Function Conv_Oct(Ch_Bin : String) : String;
Var
  L, j, i, e, a, a_oct : Integer;
  ch1, ch2, oct : String;
Begin
   While Length(Ch_Bin) Mod 3 <> 0 do
      Ch_Bin := '0' + Ch_Bin;
   Oct := '';
   L := Length(Ch_Bin);
   For i := 1 To (length(Ch_Bin) Div 3) Do
   Begin
      ch1 := Copy (Ch_Bin, L - 2,3);
      A_oct := 0;
      For j := 1 To 3 Do
      Begin
        Val(ch1[j], a, e);
        A_oct := A_oct + a * Puiss_2 (3 - j);
      End;
      Str(A_oct, ch2);
      oct := ch2 + oct;
      L := L - 3;
   End;
   conv_oct := oct;
End;
```

V.6 Conversion d'un nombre d'une base B1 en une base B2

Nous voulons convertir un nombre d'une base B1 en base B2, avec $2 \leq b1 \leq 16$ et $2 \leq b2 \leq 16$.

a) Proposez une analyse modulaire au problème,
b) Déduisez les algorithmes des modules dégagés,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **B1_B2**.



a) Analyse du problème

Résultat : Ecrire ("(", N_B1, ")",B1," = (", **FN** Conversion_B2 (N_B1, B1, B2), ")", B2)

Données :

- Une chaîne de caractères N_B1 représentant le nombre de la base B1, entrée par le biais de la procédure Saisie_Nombre. N_B1 ne peut contenir que des chiffres de la base B1 (commençant obligatoirement par le chiffre 0).
- Les bases B1 et B2 sont données par la procédure Saisie_Base. B1 et B2 sont comprises entre 2 et 16.

b) Algorithme du programme principal

0) Début B1_B2
1) **Proc** Saisie_Base (B1)
2) **Proc** Saisie_Base (B2)
3) **Proc** Saisie_Nombre (N_B1, B1)
4) Ecrire ("(", N_B1,")",B1," = (", **FN** Conversion_B2 (N_B1, B1, B2), ")", B2)
3) Fin B1_B2

Tableau de codification des objets globaux

Objets	Types / Nature	Rôle
N_B1	Chaîne	Nombre à convertir
B1, B2	Entier	Les bases de conversion
Saisie_Base	Procédure	Permet de saisir une base de numération
Saisie_Nombre	Procédure	Permet de saisir le nombre à convertir
Conversion_B2	Fonction	Permet de retourner la conversion de N_B1 en base B2

c) Analyse de la fonction Conversion_B2

Résultat : Conversion_B2

Traitement : Ce traitement sera décomposé en deux modules : un passage d'une base B1 en base 10, puis une conversion en base B2 du nombre décimal obtenu.

- CH_B2 est le résultat de la conversion d'un nombre décimal N_10 en un nombre en base B2. Une fonction Conv_B2 traduira ce traitement,
- N_10 est le résultat de la conversion du nombre N_B1 en base 10. Cette tâche est effectuée par la fonction Conv_10.

d) Algorithme de la fonction Conversion_B2

0) Fonction Conversion_B2 (Ch_B1 : Chaîne; B1, B2 : Entier) : Chaîne
1) N_10 ← FN Conv_10 (CH_B1, B1)
2) CH_B2 ← FN Conv_B2 (N_10, B2)
3) Conversion_B2 ← CH_B2
4) Fin Conversion_B2

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
N_10	Entier Long	Conversion de N_B1 en base 10
Ch_B2	Chaîne	Conversion de N_10 en base B2
Conv_10	Fonction	Permet la conversion d'un nombre de base quelconque en base 10
Conv_B2	Fonction	Permet de convertir un nombre décimal en base quelconque

e) Analyse de la fonction Conv_10

Résultat : Conv_10

Traitement :

- Conv_10 ← Dec
- [Dec ← 0]

Chaque chiffre de N_B1 sera converti en valeur numérique, multiplié par la base élevée à une puissance relative à son rang (en utilisant une fonction Puiss_B1), puis, ajouté à Dec. Nous allons donc, utiliser une structure itérative complète :

Pour i de Long (Ch_B1) à 1 (pas = -1) **Faire**
Si Majuscule (Ch_B1[i]) Dans ["A".."F"]
Alors N ← Ord (Majuscule (Ch_B1[i])) - 55
Sinon Val (Ch_B1[i],N,e)

Dec ← Dec + N * FN Puiss_B1 (Long (Ch_B1) - i, B1)

f) Algorithme de la fonction Conv_10

```
0) Fonction Conv_10 (Ch_B1 : Chaîne; B1 : Entier) : Entier Long
1) Dec ← 0
   Pour i de Long (Ch_B1) à 1 (pas = -1) Faire
      Si Majuscule (Ch_B1[i]) Dans ["A".."F"]
      Alors N ← Ord (Majuscule (Ch_B1[i])) - 55
      Sinon Val (Ch_B1[i],N,e)
      FinSi
      Dec ← Dec + N * FN Puiss_B1 (Long (Ch_B1) - i, B1)
   FinPour
2) Conv_10 ← Dec
3) Fin Conv_10
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
i	Entier	Compteur
e	Entier	Indicateur d'erreurs de la conversion numérique
N	Entier	Conversion numérique de chaque chiffre de Ch_B1
Dec	Entier Long	Conversion de N_B1 en base 10
Puiss_B1	Fonction	Permet d'élever B1 à une puissance

g) Traduction en Pascal de la fonction Conv_10

```
Function Conv_10 (CH_B1 : String; B1 : Integer) : Longint;
Var
  i, e, N : Integer;
  Dec : Longint;
Begin
   Dec := 0;
   For i := Length(CH_B1) DownTo 1 Do
   Begin
      If Upcase(Ch_B1[i]) In ['A'..'F']
      Then N := Ord(Upcase(Ch_B1[i])) - 55
      Else Val(Ch_B1[i],N,e);
      Dec := Dec + N * Puiss_B1(Length(Ch_B1) - i,B1);
   End;
   Conv_10 := Dec;
End;
```

h) Analyse de la fonction Conv_B2

Résultat : Conv_B2

Traitement :

- Conv_B2 ← Ch
- [Ch ← " "]

Diviser successivement N_10 par B2 jusqu'à obtenir un quotient nul. La concaténation des restes forme le nombre converti. Nous utiliserons donc, une structure itérative à condition d'arrêt :

Répéter
R ← N_10 **Mod** B2
Si R ≥ 10
Alors Ch_R ← **Majuscule** (Chr (55 + R))
Sinon Convch (R, Ch_R)

Ch ← Ch_R + Ch
N_10 ← N_10 Div B2
Jusqu'à N_10 = 0

i) Algorithme de la fonction Conv_B2

0) Fonction Conv_B2 (N_10 : Entier Long; B2 : Entier) : Chaîne
1) Ch ← " "
Répéter
R ← N_10 Mod B2
Si R ≥ 10
Alors Ch_R ← Majuscule (Chr (55 + R))
Sinon Convch (R, Ch_R)
FinSi
Ch ← Ch_R + Ch
N_10 ← N_10 Div B2
Jusqu'à N_10 = 0
2) Conv_B2 ← Ch
3) Fin Conv_B2

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
Ch	Chaîne	Résultat de la conversion en base B2
R	Entier	Reste de la division euclidienne
Ch_R	Chaîne	Conversion en chaîne du reste R

Retenons

- Le PGCD de deux entiers positifs non nuls a et b est le Plus Grand Commun Diviseur. Parmi les méthodes utilisées pour rechercher le PGCD, nous citons la méthode de la différence.
- Le nombre **d'arrangements** dans un ensemble de n éléments noté $\mathbf{A_n^p}$ est égal à $\frac{n!}{(n-p)!}$
- Le nombre de **combinaisons** de p éléments parmi n noté $\mathbf{C_n^p}$ est égal à $\frac{n!}{p!\,(n-p)!}$

- Un entier est **divisible** par un autre, si le reste de sa division entière par ce dernier est nul.
- Une **règle de divisibilité** est une séquence d'opérations simples qui permet de reconnaître rapidement si un entier est divisible par un autre, sans qu'il soit nécessaire d'effectuer la division directe.
- La numération désigne les techniques de représentation des nombres. Les représentations écrites au moyen de signes constituent des systèmes de numération.
- De nombreuses **bases de numération** ont été employées tels que le système binaire (base 2) utilisé par les ordinateurs, le système octal (base 8), le système décimal (base 10), il est de loin le plus répandu de nos jours, le système duodécimal (base 12), le système hexadécimal (base 16) utilisé en électronique et en informatique car la conversion binaire-hexadécimale est très simple, etc.
- La représentation d'un nombre dans une base plus grande devient plus compacte.
- Il est possible de convertir un nombre appartenant à une base B1 donnée, dans une autre base B2.

Citation du Chapitre

« Compter en octal,
c'est comme compter en décimal,
si on n'utilise pas ses pouces »

Tom Lehrer

Exercices

Exercice 1

Mettez la lettre V (Vrai) dans la case qui correspond à chaque proposition si vous jugez qu'elle est vraie sinon mettez la lettre F (Faux).

a. 245025 + 9171 est divisible par

2 ☐ 3 ☐ 4 ☐

b. B43 peut être un nombre

Décimal ☐ Duodécimal (Base 12) ☐ Hexadécimal ☐

c. L'entier a+1 est représenté dans la base a (a >= 2) par le nombre

10 ☐ 11 ☐ Cela dépend de a ☐

d. La conversion du nombre décimal 3243 en hexadécimal est

ABC ☐ BAC ☐ CAB ☐

Exercice 2

Dans un contexte informatique, définir les expressions suivantes :

a) Base de numération
b) Diviseur d'un entier
c) Multiple d'un entier

Exercice 3

Une anagramme d'un mot donné est un mot de même longueur, ayant un sens ou non et formé par les mêmes lettres :

1. Ecrivez un programme Pascal permettant d'afficher le nombre d'anagrammes d'un mot donné composé uniquement de lettres,

2. Ecrivez un programme Pascal permettant d'afficher le nombre d'anagrammes commençant par une voyelle, d'un mot donné composé uniquement de lettres.

Exercice 4

Soient a et b deux réels et n un entier naturel non nul. Ecrivez un programme Pascal permettant de vérifier la formule du binôme exprimée par l'égalité suivante :

$$(a+b)^n = C_n^0\, a^n + C_n^1\, a^{n-1}\, b^1 + C_n^2\, a^{n-2}\, b^2 + \ldots + C_n^{n-1} a^1\, b^{n-1} + C_n^n\, b^n$$

Exercice 5

Soit n un entier naturel non nul. Ecrivez un programme Pascal permettant de calculer puis d'afficher la somme suivante :

$$C_n^0 - C_n^1 + C_n^2 - C_n^3 + \ldots\ldots + (-1)^n\, C_n^n$$

Exercice 6

Soit n = 61d1 où d désigne le chiffre des dizaines dans l'écriture décimale de n. Ecrivez un programme Pascal permettant d'afficher les valeurs respectives de d et de n pour lesquelles l'entier n sera divisible par 9.

Exercice 7

Soit n = m97u où m et u désignent respectivement le chiffre des milliers et le chiffre des unités dans l'écriture décimale de n. Ecrivez un programme Pascal permettant de déterminer puis d'afficher les couples (m, u) et l'entier n pour lesquels ce dernier sera divisible par 5 et par 9.

Exercice 8

Ecrivez un programme Pascal permettant de déterminer puis d'afficher l'ensemble des entiers naturels n compris entre 1 et 100 tels que n divise n + 8.

Exercice 9 (Bac 1995)

"Conversion entre bases "

On se propose de réaliser la conversion d'un nombre décimal N donné (N est un entier naturel strictement positif) en son équivalent dans une base B donnée.

Pour cela on effectue des divisions euclidiennes par B, les restes successifs seront rangés dans un vecteur (tableau) appelé RESTES à RMAX éléments (RMAX = 15).

1) Ecrire un algorithme de la procédure **SAISIE-DEC** qui saisit un entier naturel **N** strictement positif.

2) Ecrire un algorithme de la procédure **SAISIE-BASE** qui saisit un entier naturel **B** tel que $2 \leq B \leq 16$.

3) **a** - Ecrire un algorithme de la procédure **RANGER (N, B, RESTES, R)** , qui permet de :
- ranger dans le vecteur RESTES les restes successifs de la suite des divisions euclidiennes par B .
- calculer le nombre des restes R.

b - Traduire en langage Pascal l'algorithme de cette procédure.

4) Ecrire un algorithme de la procédure **RENVERSER (RESTES, R)** qui renverse les éléments rangés dans le vecteur RESTES.
(permuter **RESTES [1]** avec **RESTES [R],**
RESTES [2] avec **RESTES [R-1]**, etc.).

5) a - Ecrire un algorithme de la fonction **CONVERT (C)** qui permet de retourner le caractère qui correspond à l'entier C (C compris entre 0 et 15).

Exemples : **CONVERT** (0) = "0", **CONVERT** (9) = "9"
CONVERT (10) = "A" **CONVERT** (15) = "F"

b - Traduire en langage Pascal l'algorithme de cette fonction.

6) Ecrire un algorithme de la procédure **CONCATENATION (résultat, RESTES)** qui, en utilisant la fonction CONVERT permet de rassembler les restes en une chaîne de caractères intitulée résultat, représentant le nombre N dans la base B.

7) En utilisant les procédures et la fonction déjà définies, écrire un algorithme du programme principal intitulé **CONVERSION** permettant d'afficher le résultat de la conversion d'un décimal N dans une base **B.**

Pour chacun des exercices suivants,

a) Proposez une analyse modulaire au problème,

b) Déduisez les algorithmes des modules dégagés,

Exercice 10

Nous proposons de vérifier si un entier n donné est divisible par 7, en utilisant la règle de divisibilité suivante : Nous nous appuyons sur le fait que si le nombre mcdu est divisible par 7, alors : mcd-2*u est divisible par 7, et réciproquement.

Prenons un exemple : 7241. Nous conservons tous les chiffres sauf le dernier, et nous lui retranchons deux fois le dernier : 724-2*1=722. Nous procédons de même avec le résultat, soit 722 : 72-2*2=68. Or 68 n'est pas divisible par 7, donc 7241 non plus.

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Div_7.**

Exercice 11

Nous voulons vérifier si un entier n donné est divisible par 11, en utilisant la règle de divisibilité suivante : Il faut que la différence entre la somme des chiffres de rang pair et la somme des chiffres de rang impair soit divisible par 11.

Exemples : 651 n'est pas divisible par 11 (car (6+1)-5=2, et 2 n'est pas divisible par 11), mais 41679 l'est (car (4+6+9)-(1+7) = 11, et 11 est divisible par 11).

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Div_11.**

Exercice 12

Nous proposons de vérifier si un entier n donné est divisible par 6, en utilisant la règle de divisibilité suivante : Un nombre est divisible par 6 s'il est divisible à la fois par 2 et par 3.

NB : Ne pas utiliser directement l'opérateur arithmétique Mod.

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Div_6.**

Exercice 13

Nous voulons convertir un nombre octal en base 2.

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Oct_Bin.**

Exercice 14

Nous proposons de convertir un nombre binaire en base 16.

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Bin_Hex.**

Exercice 15

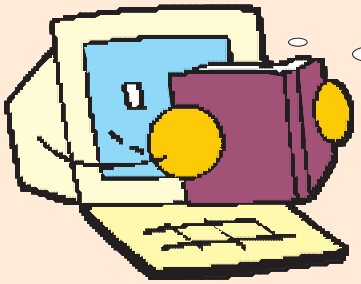
Nous proposons de convertir un nombre décimal en base binaire.

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **DecBin_R.**

Exercice 16

Nous voulons lire deux nombres binaires puis vérifier si l'un est divisible par l'autre.

Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Div_Bin.**



Lecture

Quelques règles de divisibilité

Par 2

Il suffit que le dernier chiffre soit divisible par 2 (c'est-à-dire que ce soit 0, 2, 4, 6, ou 8).
Exemples : 5287 n'est pas divisible par 2, mais 136 et 94618 le sont.

Par 3

Il suffit que la somme des chiffres qui composent le nombre soit divisible par 3.
Exemples : 734 n'est pas divisible par 3 (car 7+3+4=14, 14 n'est pas divisible par 3), mais 81234 l'est (car 8+1+2+3+4=18, 18 est divisible par 3).

Par 4

Il suffit que les deux derniers chiffres soient divisibles par 4.
Exemples : 61934 n'est pas divisible par 4 (34 n'est pas divisible par 4), mais 2028 l'est (28divisible par 4 car 28=4x7).

Par 5

De même que pour 2, il suffit que le dernier chiffre soit divisible par 5 (c'est-à-dire que ce soit 0 ou 5). Exemples : 2497 n'est pas divisible par 5, mais 625 et 4800 le sont.

Par 7

C'est un peu plus compliqué. On s'appuie sur le fait que si le nombre abcd est divisible par 7, alors : abc-2d est divisible par 7, et réciproquement.
Prenons un exemple : 7241. Nous conservons tous les chiffres sauf le dernier, et nous lui retranchons deux fois le dernier : 724-2x1=722. Nous procédons de même avec le résultat, soit 722 : 72-2x2=68. Or 68 n'est pas divisible par 7, donc 7241 non plus.
Autre exemple : 30086. 3008-2x6=2996 ; 299-2x6=287 ; 28-2x7=14 ; 14 étant divisible par 7, 30086 l'est aussi !

Par 9

De même que pour 3, il suffit que la somme des chiffres qui composent le nombre soit divisible par 9.
Exemples : 5019 n'est pas divisible par 9 (5+0+1+9=15, 15 n'est pas divisible par 9), mais 837 l'est (8+3+7=18, 18 est divisible par 9).

Par 11

Il faut que la différence entre la somme des chiffres de rang pair et la somme des chiffres de rang impair soit divisible par 11.
Exemples : 651 n'est pas divisible par 11 (car (6+1)-5=2, et 2 n'est pas divisible par 11), mais 41679 l'est (car (4+6+9)-(1+7)=11, et 11 est divisible par 11).

Par 25

De même que pour 4, il suffit que les deux derniers chiffres soient divisibles par 25.
Exemples : 6287 n'est pas divisible par 25 (87 n'est pas divisible par 25), mais 4150 l'est (50=25x2).

Chapitre 6

Les algorithmes d'approximation

Objectifs

Acquérir des habilités de résolution de problèmes à travers l'apprentissage d'algorithmes d'approximation

Plan du chapitre

Chapitre 6

Les algorithmes d'approximation

I. Introduction

La résolution exacte de la plupart des problèmes d'optimisation naturels impliquerait un temps de calcul prohibitif. Caractériser la difficulté de l'approximation de ces problèmes par des algorithmes de temps polynomial est donc un sujet d'étude inévitable en informatique et en mathématiques.

Depuis des décennies, les chercheurs essaient de concevoir des algorithmes d'approximation qui fournissent, en temps polynomial, des solutions «quasi-optimales» dont l'écart à l'optimum peut être borné supérieurement. Depuis quinze ans, ces chercheurs ont décrit de nouvelles techniques, basées sur l'arrondissement des valeurs de programmes linéaires, la programmation semi-définie et le plongement des espaces métriques.

Ils ont aussi montré qu'il était difficile de trouver des solutions approchées de certains problèmes.

Les problèmes d'optimisation forment donc un ensemble très riche de possibilités : de la possibilité d'approcher avec une précision arbitraire, à l'impossibilité de toute garantie sur la qualité de l'approximation.

II. Recherche du point fixe d'une fonction

Présentation :

En mathématiques, pour une application f d'un ensemble E, un élément x de E est un point fixe de f si $f(\mathbf{x}) = \mathbf{x}$.

Exemples :

- Dans le plan, la symétrie par rapport à un point A admet un unique point fixe : A
- L'application inverse (définie sur l'ensemble des réels non nuls) admet deux points fixes : -1 et 1

Toutes les fonctions n'ont pas nécessairement de point fixe; par exemple, la fonction f telle que $f(x) = x + 1$ n'en possède pas, car il n'existe aucun nombre réel x égal à x+1.

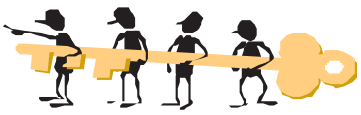
Dans la suite de ce paragraphe, nous allons établir des algorithmes permettant de calculer une valeur approchée du point fixe d'une fonction.

Soit la suite réelle (U_n) récurrente et définie par sa valeur initiale U_0 et par la relation de récurrence $U_{n+1} = f(U_n)$. Dans le cas où (U_n) converge, elle le fait nécessairement vers **un point fixe** de f.

Activité 1

Nous voulons résoudre l'équation sin(x) = 1-x

a) Proposez une analyse modulaire au problème,
b) Déduisez les algorithmes correspondants,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Pt_fixe**



a) Analyse du problème

Résultat : Ecrire ("Le point fixe est : ", xact, " trouvé après ", i, " itérations. ")

Traitement : xact représente le $i^{ème}$ terme de la suite. En effet, l'équation sin(x) = 1-x équivaut à 1-sin(x) = x, et donc l'application f est représentée comme suit : f(x) = 1-sin(x).

```
xact ← 1     {choix arbitraire}
i ← 0
Répéter
        xpre ← xact
        xact ← f(xact)
        i ← i + 1
Jusqu'à (Abs (xact-xpre) < Epsilon)
```

b) Algorithme du programme principal et codification des objets

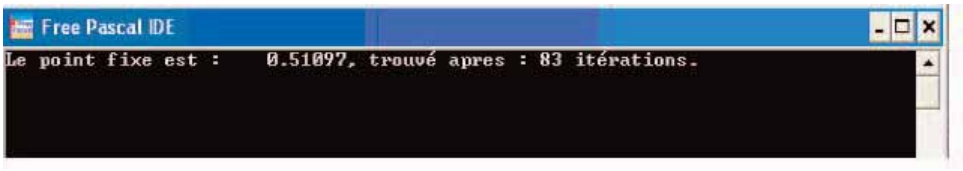
```
0) Début Pt_Fixe
1) i ← 0, xact ← 1
        Répéter
                xpre ← xact
                xact ← f(xact)
                i ← i + 1
        Jusqu'à (Abs (xact-xpre) < Epsilon)
2) Ecrire ("Le point fixe est : ", xact, " trouvé après ", i, " itérations. ")
3) Fin Pt_Fixe
```

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
Epsilon	Constante = 10^{-5}	Différence entre deux termes consécutifs
i	Entier non signé	Nombre d'itérations
xact, xpre	Réel	Deux termes consécutifs de f
f	Fonction	Retourne la valeur de 1-sin(x)

c) Traduction en Pascal

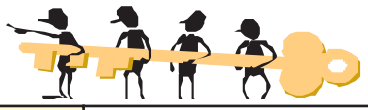
```
Program Pt_Fixe;
Uses Crt;
Const Epsilon = 1E-5;
Var
   i : word; xact, xpre : real;
   Function f (x : Real) : Real;
   Begin
         f := 1 - sin (x);
   End;
{Programme Principal}
Begin
  i := 0;
  xact := 1;
  Repeat
    xpre := xact;
    xact := f(xact);
    i := i + 1;
  Until (abs(xact-xpre) < Epsilon);
  Write ('Le point fixe est : ',xact:10:6,' trouvé après', i, ' itérations. ');
End.
```



III- Calcul de valeurs approchées de constantes connues

Activité 2

Citez quelques constantes numériques ? Donnez leurs valeurs respectives ?



e	2,718...	Le nombre de neper
π	3,1416...	Le nombre π
g	9, 80665	L'accélération normale de la pesanteur

Dans la suite, nous allons présenter des algorithmes permettant de calculer des valeurs approchées pour les constantes **π** et **e**.

1) Valeur approchée de π :

a) Présentation :

Le nombre π n'est pas égal à 3,14 qui n'est qu'une approximation de π et cette approximation est suffisante pour les calculs les plus courants.

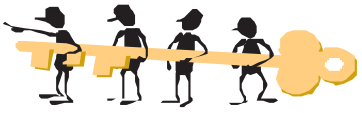
Beaucoup de techniques existent pour établir des valeurs approchées du nombre π. Dans l'Histoire, la première de ces méthodes est attribuée à Archimède. Elle consiste à encadrer le périmètre d'un cercle par ceux de deux polygones réguliers, le premier inscrit et le deuxième circonscrit au cercle. Ce sont des polygones réguliers. C'est à dire que leurs côtés ont, tous, la même longueur. Plus le nombre de côtés augmente, et plus le périmètre des polygones se rapproche de la circonférence du cercle jusqu'à presque se confondre avec lui. Le périmètre du polygone inscrit représente une approximation par défaut du périmètre du cercle. Tandis que le périmètre du polygone circonscrit représente une approximation par excès.

b) Valeur approchée par la formule d'Euler :

Nous proposons de calculer une valeur approchée de π en utilisant la formule d'Euler :

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots\ldots + \frac{1}{n^2} + \ldots\ldots$$

a) Proposez une analyse au problème,
b) Déduisez l'algorithme correspondant,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Pi_Euler**



i) Analyse du problème

Résultat : Ecrire ("La valeur approchée de Pi trouvée est : ", RacineCarrée (6*S2))

Traitement :

S2 ← 1, i ← 2

Répéter

S1 ← S2

S2 ← S1 + 1 / $(i)^2$

i ← i +1

Jusqu'à $\sqrt{(6 * S2)} - \sqrt{(6 * S1)} < Epsilon$

ii) Algorithme du programme principal et codification des objets

0) Début Pi_Euler
1) S2 ← 1, i ← 2
Répéter
S1 ← S2
S2 ← S1 + 1 / **Carré (i)**
i ← i +1
Jusqu'à RacineCarrée (6*S2)- RacineCarrée (6*S1) < Epsilon
2) Ecrire ("La valeur approchée de Pi trouvée est : ", **RacineCarrée** (6*S2))
3) Fin Pi_Euler

Tableau de codification des objets

Objets	Type / Nature	Rôle
i	Entier Long	Compteur
S1, S2	Réel	Deux termes consécutifs
Epsilon	Constante = 10^{-4}	Erreur de l'approximation

iii) Traduction en Pascal

```
Program Pi_Euler;
Uses Crt;
Const Epsilon = 10E-4;

Var
  i : longint;
  S1, S2 : Real;

Begin

   S2:= 1; i := 2;
   Repeat
    S1 := S2;
    S2 := S1 + 1/sqr(i);
    i := i+1;
   Until Sqrt(6*S2)- Sqrt(6*S1) < Epsilon;

   Write('La valeur approchée de Pi trouvée est : ',Sqrt(6*S2):10:6);

End.
```

c) Applications :

Application 1 :

Copiez le tableau suivant sur votre cahier, puis exécutez le programme Pi_Euler, et remplissez le tableau par les valeurs approchées de π trouvées :

Epsilon	Valeur approchée de π
10^{-6}	
10^{-7}	
10^{-8}	
10^{-9}	
10^{-10}	

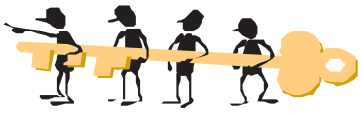
Application 2 :

Proposez une solution **récursive** pour le problème du calcul de la valeur approchée de π par la formule d'Euler.

d) Valeur approchée par la formule de Wallis :

Nous proposons de calculer une valeur approchée de π, en utilisant la formule de Wallis : π/2 = (2/1) * (2/3) * (4/3) * (4/5) * (6/5) * (6/7) * (...)

a) Proposez une analyse au problème,
b) Déduisez l'algorithme correspondant,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Pi_Wallis**



i) Analyse du problème

Résultat : Ecrire ("La valeur approchée de Pi trouvée est : ", **2*P2**)
Traitement : P2 ← 2, i ← 2, Num ← 2, Den ← 3
Répéter
P1 ← P2
P2 ← P1 * (Num/Den)
Si i Mod 2 = 0 **Alors** Num ← Num + 2
Sinon Den ← Den + 2

i ← i+1
Jusqu'à |2*P2-2*P1| < Epsilon

ii) Algorithme du programme principal et codification des objets

0) Début Pi_Wallis
1) P2 ← 2, i ← 2, Num ← 2, Den ← 3

Répéter
P1 ← P2
P2 ← P1 * (Num/Den)
Si i Mod 2 = 0 **Alors** Num ← Num + 2
Sinon Den ← Den + 2
FinSi
i ← i+1
Jusqu'à Abs(2*P2-2*P1) < Epsilon
2) Ecrire ("La valeur approchée de Pi trouvée est : ", **2*P2**)
3) Fin Pi_Wallis

Tableau de codification des objets

Objets	Type / Nature	Rôle
i	Entier	Compteur
P1, P2	Réel	Deux termes consécutifs
Epsilon	Constante = 10^{-4}	Erreur de l'approximation
Num	Entier	Représente le numérateur d'un terme du produit
Den	Entier	Représente le dénominateur d'un terme du produit

iii) Traduction en Pascal

```
Program Pi_Wallis;
Uses Crt;
Const
  Epsilon = 10E-4;
Var
 i, Num, Den : integer;
 P1, P2 : Real;

Begin
   P2:= 2;
   Num := 2;
   Den := 3;
   i := 2;
   Repeat
      P1 := P2;
      P2 := P1 * (Num /Den);
      If i mod 2 = 0
      Then  Num := Num+2
      Else  Den := Den+2 ;
      i := i+1;
   Until abs(2*P2 - 2*P1) < Epsilon;
   Write('La valeur approchée de Pi trouvée est : ',2*P2:10:6);
End.
```

e) Application :

Copiez le tableau suivant sur votre cahier, puis exécutez le programme Pi_ Wallis, et remplissez le tableau par les valeurs approchées de π trouvées :

Epsilon	Valeur approchée de π
10^{-4}	
10^{-5}	
10^{-6}	
10^{-7}	
10^{-8}	
10^{-9}	

2) Valeur approchée de e :

a) Présentation :

Le nombre e (nom donné par **Euler**), appelé aussi nombre exponentiel ou nombre de Neper (Napier's number) est une constante représentant la base du logarithme népérien. Elle est égale à 2,7182818284 5904523536 0287471352 6624977572 4709369995 9574966967 …

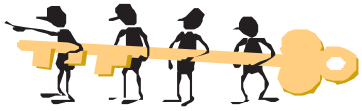
C'est un nombre **irrationnel**, sa valeur a été approchée par la formule suivante : 1 + (1/1!) + (1/2!) + ...

b) Valeur approchée avec les factorielles :

Nous voulons calculer une valeur approchée de e, à 10^{-4} en utilisant la formule suivante :

e = 1 + (1/1!) + (1/2!) + (1/3!) + (1/4!) + (1/5!) + (1/6!) + ...

a) Proposez une analyse au problème,
b) Déduisez l'algorithme correspondant,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **e_Fact**



i) Analyse du problème

Résultat : Ecrire ("La valeur approchée de e trouvée est : ", **S2**)

Traitement :

S2 ← 1, i ← 0

Répéter

S1 ← S2

i ← i + 1

S2 ← S2 + 1/ FN Fact(i)

Jusqu'à S2 - S1 < Epsilon

ii) Algorithme et codification des objets

0) Début e_Fact
1) S2 ← 1, i ← 0
   Répéter
      S1 ← S2
      i ← i + 1
      S2 ← S2 + 1/ FN Fact(i)
   Jusqu'à S2 - S1 < Epsilon
2) Ecrire ("La valeur approchée de e trouvée est : ", **S2**)
3) Fin e_Fact

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
i	Entier Long	Compteur
S1, S2	Réel	Deux termes consécutifs de la somme
Fact	Fonction	Retourne la factorielle d'un entier
Epsilon	Constante = 10^{-4}	Erreur de l'approximation

iii) Traduction en Pascal de la fonction e

```
Program e_Fact;
Uses Crt;
Const
  Epsilon = 10E-4;
var
 i : Longint;
 S1, S2 : Real;
 Function Fact (Nb : Longint):Longint;
 Var
   j, F : Longint;
 Begin
  F := 1;
  For j := 1 To Nb Do
    F := F * j;
  Fact := F;
 End;

{Programme Principal}
Begin
   S2:= 1;
   i := 0;
   Repeat
       S1 := S2;
       i := i + 1;
       S2 := S2 + 1/Fact(i);
   Until S2-S1 < Epsilon;
   Write('La valeur approchée de e trouvée est : ',S2:12:11);

End.
```

c) Applications :

Application 1 :

Copiez le tableau suivant sur votre cahier, puis exécutez le programme *e*_Fact, et remplissez le tableau par les valeurs approchées de e trouvées :

Epsilon	Valeur approchée de *e*
10^{-4}	
10^{-5}	
10^{-6}	
10^{-7}	
10^{-8}	
10^{-9}	

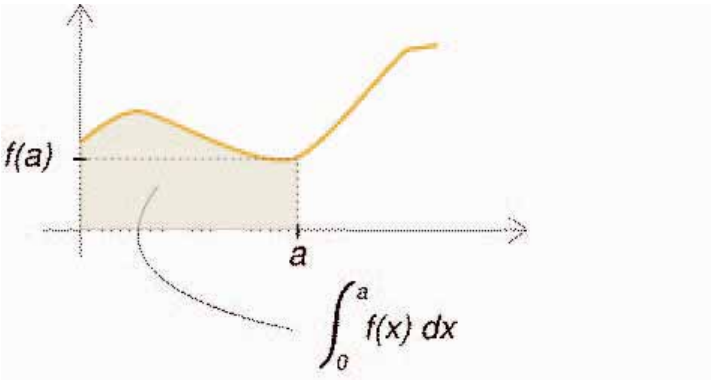
Application 2 :

Proposez une solution **récursive** pour le problème de calcul de la valeur approchée de ***e*** par la formule des factorielles.

IV. Calcul d'aires

1) Introduction :

Soit une fonction f continue et croissante dans l'intervalle [a,b].
Si nous ne connaissons pas de primitive de la fonction f, nous ne pouvons pas calculer $\int_a^b f(t)dt$. Nous chercherons alors, à en déterminer une valeur approchée.



Représentation graphique d'une intégrale

En mathématiques, l'intégrale d'une fonction réelle positive est la valeur de l'aire du domaine délimité par l'axe des abscisses et la courbe représentative de la fonction.
Pour les fonctions qui prennent des valeurs réelles négatives (gardant un signe constant par intervalles), une définition d'aire algébrique rend possible une aire négative.
Pour les calculs d'aires, des méthodes d'intégration numérique sont utilisées. L'objectif donc, est d'approcher la valeur de $\int_a^b f(t)dt$. La plupart des méthodes d'intégration numérique fonctionnent suivant le même principe :

Nous commençons par subdiviser l'intervalle [a, b] en n intervalles $[a_i, a_{i+1}]$ de diamètres égaux à $\frac{b-a}{n}$, avec $a_1 = a$ et $a_{n+1} = b$. La subdivision est régulière et plus n est grand, plus la longueur de chacun des intervalles devient petite. Le diamètre de chaque intervalle, c'est à dire sa longueur, est notée : $h = a_{i+1} - a_i$

Puis, pour chaque intervalle $[a_i, a_{i+1}]$, nous essayons d'approcher $\int_{ai}^{ai+1} f(t)dt$. C'est à dire que l'idée est de faire l'approximation suivante : sur chaque intervalle nous remplaçons l'aire sous la courbe qui représente la fonction f par une aire plus simple à calculer.

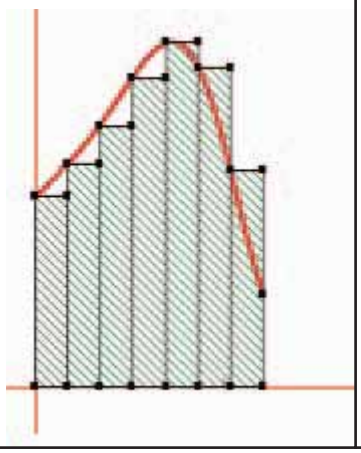
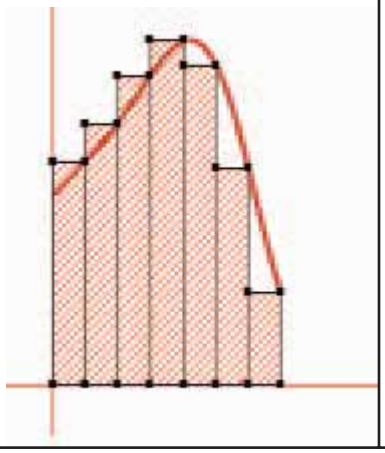
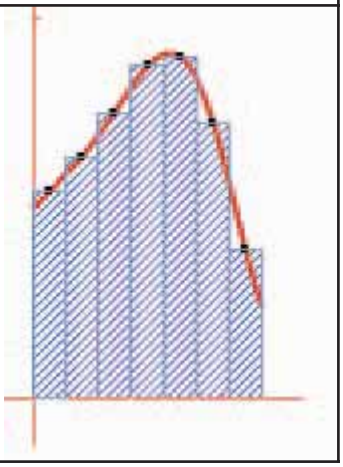
Plus nous augmentons le nombre de sous intervalles du segment [a,b], plus la valeur de l'intégrale approchée converge vers la vraie valeur de l'intégrale.

Les méthodes les plus utilisées sont : la méthode des rectangles, la méthode des trapèzes, la méthode de Simpson, la méthode de Romberg, etc.

2) Méthode des rectangles :

a) Principe :

Sur chaque intervalle $[a_i, a_{i+1}]$ est représenté un rectangle de largeur $a_{i+1} - a_i$ et d'hauteur :

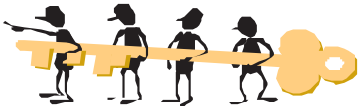
Soit $f(a_i)$: Rectangles à gauche,	Soit $f(a_{i+1})$: Rectangles à droite,	Soit $f((a_i + a_{i+1})/2)$: Rectangles du point milieu,
		

La méthode des rectangles consiste à remplacer " l'aire sous la courbe " par la somme des aires des rectangles obtenus.

b) Application :

Nous proposons de calculer, en utilisant la méthode des rectangles, l'aire résultante de la courbe de la fonction f : $x \rightarrow \frac{1}{1+x^2}$ sur un intervalle [a, b]

a) Proposez une analyse modulaire au problème,
b) Déduisez les algorithmes correspondants,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Meth_Rec**



Analyse du problème

Résultat : Ecrire ("Le calcul de l'aire par la méthode des rectangles = ", FN Rectangles (a, b, n))

Traitement : Rectangles est une fonction permettant de calculer l'aire par la méthode des rectangles.

Données : Deux réels a et b représentant les bornes du grand intervalle et un entier n, représentant le nombre d'intervalles après la subdivision. Leur lecture sera la tâche de la procédure Lecture

Algorithme du programme principal et codification des objets

0) Début Meth_Rec
1) Proc Lecture (a, b, n)
2) Ecrire ("Le calcul de l'aire par la méthode des rectangles = ", FN Rectangles (a, b, n))
3) Fin Meth_Rec

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
a, b	Réel	Bornes de l'intervalle initial
n	Entier	Nombre de sous intervalles
Lecture	Procédure	Permet de lire a, b et n
Rectangles	Fonction	Retourne une valeur approchée de l'aire à calculer.

Analyse de la fonction Rectangles

Résultat : Rectangles

Traitement :

- Rectangles ← Somme * h
- Somme ← 0, h ← $\frac{b1-a1}{n1}$, x ← a1 + h/2

Pour k de 1 à n1 **Faire**
Somme ← Somme + f(x)
x ← x + h {avancement}

Algorithme de la fonction Rectangles

0) Fonction Rectangles (a1, b1 : Réel ; n1 : Entier) : Réel
1) Somme ← 0

$h \leftarrow \frac{b1 - a1}{n1}$

$x \leftarrow a1 + \frac{h}{2}$

Pour k de 1 à n1 Faire
Somme ← Somme + f(x)
x ← x + h
FinPour
2) Rectangles ← Somme * h
3) Fin Rectangles

Tableau de codification des objets locaux

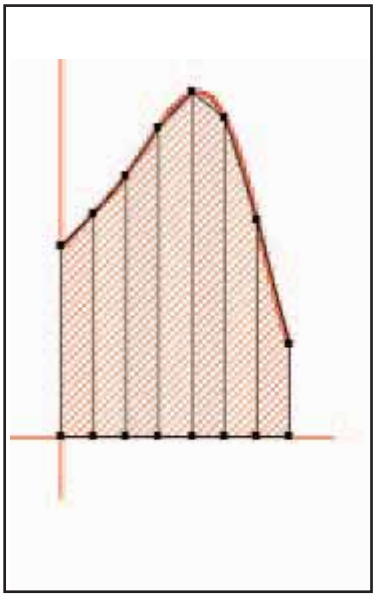
Objets	Type / Nature	Rôle
Somme, x	Réel	Variables intermédiaires
h	Réel	Largeur d'un sous intervalle
k	Entier	Compteur
f	Fonction	La fonction à intégrer

Traduction en Pascal

```
Program Meth_rectangles;
Uses crt;
Var  a, b : Real; n : Integer;

Procedure Lecture (Var a1, b1 : Real ; Var n1 : Integer) ;
Begin
 Write ('Donner la borne inférieure : '); Readln (a1);
 Repeat
 Write ('Donner la borne supérieure : '); Readln (b1);
 Until b1>a1;
 Repeat
 Write ('Donner le nombre d''intervalles : '); Readln (n1);
 Until n1>0;
End ;

Function Rectangles (a1, b1 : Real; n1 : Integer) : Real;
Var
 k : Integer;
 h, x, somme : Real;

 Function f (x1 : Real) : Real;
 Begin
     f := 1/(1+ sqr (x1));
 End;

Begin {Début de la fonction Rectangles}
   somme := 0;
   h := (b1 - a1 ) / n1;
   x := a1+ h / 2;
   For k := 1 To n1 Do
   Begin
     somme := somme + f (x);
     x := x + h
   End;
   rectangles := somme * h;
End;
```

```
{Programme Principal_ Méthode du point milieu}
Begin
  Lecture (a, b, n) ;
  Write ('Le calcul de l''aire par la méthode des rectangles = ',
  Rectangles (a, b, n));
End.
```

3) Méthode des trapèzes :



a) Principe :

Dans cette méthode, nous subdivisons l'intervalle [a, b] en n sous-intervalles, de même largeur h = (b-a)/n. Nous notons $a_i = a + i * h$.

Par linéarité de l'intégrale, nous savons que l'intégrale globale correspond à la somme des intégrales de f sur chaque sous-intervalle de [a,b]. Soit $[a_0, a_1]$ un de ces sous-intervalles, nous approchons l'intégrale de f sur cet intervalle par l'aire du trapèze reliant les points $(a_0, 0)$, $(a_0, f(a_0))$, $(a_1, f(a_1))$ et $(a_1,0)$. Cette aire vaut

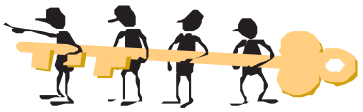
$h * ((f(a_0) + f(a_1)) / 2)$

L'intégrale globale s'obtient donc en additionnant ces n aires, c'est à dire la somme des $h * ((f(a_i) + f(a_{i+1})) / 2)$.

b) Application :

Utilisez la méthode des trapèzes, pour calculer l'aire résultante de la courbe de la fonction $f : x \rightarrow \frac{1}{1 + x^2}$ sur un intervalle [a, b]

a) Proposez une analyse modulaire au problème,
b) Déduisez les algorithmes correspondants,
c) Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Meth_Tra**



i) Analyse du problème

Résultat : Ecrire ("Le calcul de l'aire par la méthode des trapèzes = ", FN Trapezes (a, b, n))

Traitement : Trapezes est une fonction permettant de calculer l'aire par la méthode des trapèzes.

Données : Deux réels a et b représentant les bornes du grand intervalle et un entier n, représentant le nombre d'intervalles après la subdivision. Leur lecture sera la tâche de la procédure Lecture

ii) Algorithme du programme principal et codification des objets

0) Début Meth_Trap
1) Proc Lecture (a, b, n)
2) Ecrire ("Le calcul de l'aire par la méthode des trapèzes = ", FN Trapezes (a, b, n))
3) Fin Meth_Trap

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
a, b	Réel	Bornes de l'intervalle initial
n	Entier non signé	Nombre de sous intervalles
Lecture	Procédure	Permet de lire a, b et n
Trapezes	Fonction	Retourne une valeur approchée de l'aire à calculer.

iii) Analyse de la fonction Trapezes

Résultat : Trapezes

Traitement :

- Trapezes ← Somme * h
- h ← (a1 + b1/ n1), Somme ← (f(a1) + f(a1+h)) / 2, x ← a1

  Pour k de 1 à n1-1 **Faire**

  x ← x + h {avancement}

  Somme ← Somme + (f(x) + f(x+h)) / 2

iiii) Algorithme de la fonction Trapezes

0) Fonction Trapezes (a1, b1 : Réel ; n : Entier non signé) : Réel
1) h ← (a1+b1)/n1
   Somme ← (f(a1)+f(a1+h))/2
   x ← a1
   Pour k de 1 à n1-1 **Faire**
      x ← x + h
      Somme ← Somme + (f(x)+f(x+h))/2
   FinPour
2) Trapezes ← Somme*h
3) Fin Trapezes

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
Somme, x	Réel	Variables intermédiaires
h	Réel	Largeur d'un sous intervalle
k	Entier non signé	Compteur
f	Fonction	La fonction à intégrer

iiiii) Traduction en Pascal

```
Function Trapezes (a1, b1 : Real; n1 : Word) : Real;
Var
  k : Word;
  h, x, somme : Real;
  Function f (x1 : Real) : Real;
  Begin
     f := 1/(1 + Sqr (x1));
  End;

Begin
  h := (b1-a1)/n1;
  somme := (f(a1)+f(a1 + h))/2;
  x := a1;
  For k := 1 To n1-1 Do
  Begin
    x := x+h;
    somme := somme + (f(x)+ f(x+h))/2;
  End;
  trapezes := somme *h;
End;
```

Questions :

Copiez le tableau suivant sur votre cahier, puis exécutez les deux programmes Meth_rec et Meth_tra, et remplissez le tableau par les résultats trouvés, sachant que a = 1 et b = 5 :

n	Meth_rec	Meth_tra
2		
4		
5		
6		
7		
8		

Que remarquez-vous ?

Retenons

A) Un **algorithme d'approximation** calcule une solution approximative d'un problème en un temps raisonnable (en général polynomial).

B) Les racines d'une équation de la forme f(x) = x sont appelées des **points fixes**. En faisant varier x, nous pouvons déterminer une approximation du point fixe.

C) π est un nombre irrationnel. En effet, il est impossible de l'exprimer avec un nombre fini d'entiers, de fractions rationnelles et de leurs racines.
Il n'y a donc pas de développement décimal simple de π.

$$\frac{\pi}{2} = \frac{2}{1} + \frac{2}{3} + \frac{4}{3} + \frac{4}{5} + \ldots\ldots$$

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots. + \frac{1}{k^2} + \ldots. \quad \text{(formule d'Euler)}$$

Nous appelons constante de Neper, et nous notons e, l'unique réel tel que ln e = 1. C'est la base des logarithmes naturels. Il est appelé nombre exponentiel par Euler en 1761 et il vaut approximativement : e ≈ 2,718 281 828 459 045 235 360 287 4…
La formule la plus connue approchant la valeur de e est : 1 + (1/1!) + (1/2!) + (1/3!) + …

D) La mesure de **l'aire algébrique** se trouvant entre la courbe f(x), les droites x = a, x = b et y = 0, est représentée par **l'intégrale** "de a à b" de f(x).

Retenons

1- **La méthode des rectangles** consiste à approcher l'intégrale $\int_a^b f(t)dt$ par la somme des aires des n rectangles de base $[a_i, a_{i+1}]$ et de hauteur $f(a_i)$, i variant de 0 à n-1. Nous appelons aussi cette formule : formule du "point gauche".

2 - **La méthode des trapèzes** consiste à remplacer les arcs de courbe (M_iM_{i+1}) par les segments $[M_iM_{i+1}]$. Nous obtenons, sur $[a_i , a_{i+1}]$, une approximation de l'aire sous la courbe par l'aire du trapèze : de hauteur $(a_{i+1} - a_i)$ et de bases $f(a_i)$ et $f(a_{i+1})$.

Citation du Chapitre

« Bien que cela puisse paraître paradoxal, toute science exacte est dominée par la notion d'approximation »

Bertrand Russel (1872 - 1970)

Exercice 1

La valeur de π/4 peut être approchée comme la somme des n premiers termes de la suite suivante :

$U_n = (-1)^n / (2n + 1)$

$U_0 = 1$

1/ Proposez une analyse puis déduisez l'algorithme de la fonction Un qui renvoie le $n^{ème}$ terme de la suite.

2/ Proposez une analyse puis déduisez l'algorithme de la fonction app_pi(n) qui renvoie l'approximation de π calculée à partir des n premiers termes de la suite.

Exercice 2 Bac Tp 2001

Sachant que sin (x) = x + (x^3/3!) + (x^5/5!) + (x^7/7!) + ...

Pour x très proche de zéro, Ecrire un programme Pascal qui permet d'afficher sin(x) en utilisant la formule ci-dessus.

Le calcul s'arrête quand la différence entre deux termes consécutifs devient inférieure ou égale à 10^{-4} . La dernière somme calculée est une valeur approchée de sin(x).

Le candidat pourra utiliser la fonction FACT(a) suivante qui permet de calculer la factorielle de a (a !).

```
1. DEFFN FACT (a : entier) : entier
2. F ← 1
3.  Pour i de 1 à a répéter
          F ← F * i
    Fin pour
4. FACT ← F
5. Fin FACT
```

Exercice 3 : Bac Tp 2001

Soit l'expression mathématique suivante : $\pi/4 = 1-1/3+1/5-1/7+1/9-\ldots\ldots$
Ecrire un programme Pascal qui utilise l'expression ci-dessus pour déterminer et afficher une valeur approchée de π à 10^{-4} prés.
- Le calcul s'arrête quand la différence entre deux valeurs consécutives de cette expression devient strictement inférieure à 10^{-4}.

Pour chacun des exercices suivants :

a) Proposez une analyse modulaire au problème,

b) Déduisez les algorithmes correspondants.

Exercice 4

Soit U_n une suite de nombres avec $U_0 = 1$ et $U_1 = 1$, puis les suivants définis par $U_{n+2} = U_{n+1} + U_n$.

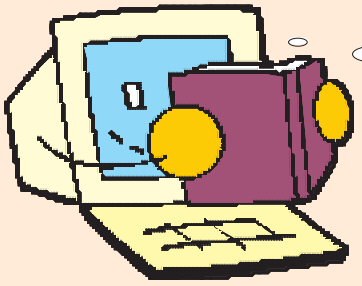
1) Calculez les n premiers termes de U_n et donnez une valeur approchée des quotients U_{n+1}/ U_n.
2) Que remarquez-vous en liaison avec le nombre d'or $\varphi = \dfrac{1+\sqrt{5}}{2}$?

Exercice 5

Cherchez le point fixe de la fonction $f(x) = \sqrt{1+x}$
Traduisez et testez la solution obtenue.
Enregistrez votre programme sous le nom **Pt_fixe2**

Exercice 6

Calculez une valeur approchée de : $\int_1^5 (x + 3 \sin x)dx$ en utilisant la méthode des trapèzes.
Traduisez et testez la solution obtenue. Enregistrez votre programme sous le nom **Trapeze.**



Lecture

L'historique de π

Le nombre pi est connu depuis l'antiquité, évidemment, pas au sens où nous l'entendons maintenant (notion abstraite de constante mathématique) mais en tant que rapport entre la longueur du cercle et son diamètre et d'ailleurs surtout en tant que méthode de calcul du périmètre du cercle (ou de l'aire du disque).

- **En 2000 avant J.C**, les Babyloniens connaissaient Pi (comme le rapport constant entre la circonférence d'un cercle et son diamètre, mais pas comme objet mathématique). Ils avaient comme valeur 3 + 7/60 + 30/3600 (ils comptaient en base 60) soit 3 + 1/8 = 3,125.
- **Vers 1650 avant J.C**, les Egyptiens avaient comme valeur $(16/9)^2$ qui vaut environ 3,16. Cette valeur a été retrouvée sur le fameux papyrus de Rhind, écrit par le scribe Ahmès, acheté par un Ecossais qui s'appelle ... Henry Rhind. Il est conservé au British museum.

Ensuite, pi apparaît :

- **En Chine vers 1200 avant J.C**, avec pour valeur 3.
- **Dans la Bible vers 550 avant J.C**, avec pour valeur 3.
- **En Grèce,** avec en particulier Archimède en 250 av. JC qui donne l'encadrement 223/71 < pi < 22/7 et Ptolémée en 150 qui utilise 3 + 8/60 + 30/3600 = 3,1416666.
- **En Chine** au $V^{ème}$ siècle, avec pour valeur 355/113.
- **En Inde** : 3 + 177/1250 = 3,1416 en 380 puis 3,16227 (racine carrée de 10) avec Brahmagupta en 640
- **Au Moyen-Orient** avec Al Khwarizmi en 800 (Ouzbekistan) et Al Kashi en 1429 (Turkestan) qui calcule 14 décimales de pi.
- **En Europe** : l'italien Fibonacci, en 1220, trouve la valeur 3,141818; au Pays-Bas avec Van Ceulen (20 décimales en 1596 puis 34 décimales en 1609 !), en France avec Viète (9 décimales en 1593).

Ensuite vint le développement des techniques de calculs avec l'analyse (dérivée, intégrales, sommes de séries, produits infinis, ...), Wallis en 1655, Newton (16 décimales en 1665), Gregory, Leibniz, Machin (100 décimales en 1706), puis Euler (20 décimales calculées en une heure !) vers 1760 et beaucoup d'autres.

Les champions contemporains sont les **frères Chudnovsky** avec 4 milliards de décimales en 1994 et **Kanada** et **Tamura** dont le dernier record datait de 1999 avec 206 milliards de décimales (en environ 33 heures de calculs).

Kanada a battu son propre record le 6 décembre 2002 avec une équipe de neuf autres chercheurs : 1 241 100 000 000 décimales ont été calculées à l'aide d'un super calculateur (400 heures de calculs !) en utilisant un algorithme que l'équipe a mis cinq ans à mettre au point.

La notation π

π est la première lettre du mot "périmètre" en grec. Il y a plusieurs versions sur l'apparition du symbole, mais l'époque est toujours la même : **vers 1600.**

Euler, utilise la lettre π, dans un ouvrage sur les séries, publié en latin en 1737 puis, en 1748, dans son "Introduction à l'analyse infinitésimale", ce qui imposa définitivement cette notation

Nature algébrique de π

- π **n'est pas un nombre décimal**, c'est-à-dire que les chiffres après la virgule ne sont pas en nombre fini (π a une infinité de décimales).

- π **est un nombre irrationnel**, c'est-à-dire qu'il ne peut pas s'écrire comme une fraction de deux nombres entiers. Autrement dit, cela signifie aussi que les chiffres de π ne sont pas prévisibles. On ne peut pas deviner une décimale sans la calculer explicitement, comme qu'on peut le faire avec le nombre 1/7 = 0,14285714285714... par exemple.
L'irrationalité de π fut démontrée en 1761 par l'Allemand Lambert.

- π **est un nombre transcendant** c'est-à-dire qu'il n'est solution d'aucune équation à coefficients rationnels. Cela fut démontré en 1881 par Lindemann.

Des décimales du nombre π

Depuis le 6 décembre 2002, il est connu 1 241 100 000 000 (plus de 1200 milliards !) de décimales du nombre pi ...
A quoi cela sert-il de connaître tant de décimales de pi ?

- Le calcul de décimales de pi est un très bon test pour vérifier la précision des calculs des ordinateurs (deux erreurs graves furent ainsi détectées sur les super-ordinateurs IBM 590 et R8000)
- Mais la motivation la plus importante n'est pas de connaître de plus en plus de décimales de pi mais bel et bien de les calculer. En effet, le calcul d'un si grand nombre de chiffres demande des algorithmes de calculs très perfectionnés et a permis de très grand progrès dans ce domaine

Les 100 premières décimales de pi :

3,141 592 653 589 793 238 462 643 383 279 502 884 197 169 399 375 105 820 974 944 592 307 816 406 286 208 998 628 034 825 342 117 067

Valeurs approchées de π

Voici des éclaircissements sur quelques valeurs approchées célèbres de π :
Tout d'abord, AUCUNE des valeurs données ci-dessous n'est LA valeur du nombre π.

- **"Trois quatorze"** (3,14) : la plus connue. Valeur approchée de π avec deux chiffres exacts après la virgule. C'est une très bonne valeur pour la plupart des calculs.

- **"Vingt-deux septièmes"** (22/7) : il y en a régulièrement qui soutiennent que c'est LE nombre π.

  En réalité, c'est la fraction la plus simple qui approche le mieux π. C'est la première fraction réduite du développement de pi en fraction continue : 3 + 1/7 = 21/7 + 1/7 = 22/7. A noter que c'est une valeur qu'Archimède avait calculée aux environs de 250 avant JC pour encadrer π. 22/7 vaut 3,142 857 142 857 142 ... et donne une valeur approchée de π avec deux chiffres exacts après la virgule.
- **355/113 :** moins connue mais c'est une très bonne approximation rationnelle de π. C'est une réduite du développement de pi en fraction continue : 3 + 1 / (7 + 1 / (15 + 1/ 1).

  355 / 113 vaut environ 3,141 592 92 qui est une valeur de π avec 6 chiffres exacts après la virgule (3,141592)…

http://trucsmaths.free.fr

Constantes

Liste des principales constantes mathématiques

Nom	Valeur à 4 décimales	Description
i	$\sqrt{-1}$	Base des nombres imaginaires
0	0	Elément neutre de l'addition
Nombre de Champernowne ou Nombre de Mahler	0,1234	Suite des nombres entiers accolés
Constante de Copeland - Erdös	0,2357	Succession des nombres premiers qui sont consécutifs
Constante de Artin	0,3739	Proportion de nombres premiers longs parmi les nombres premiers
Constante de Thue-Morse	0,4124	Nombre formé d'une suite binaire
Constante oméga	0,5671	Arithmétique
Constante d'Euler	0,5772	1/1 + 1/2 +1/3 + 1/4 + ... - log n
Constante de Gompertz	0,5963	Fractions continues
Constante de Ramanujan	0,7642	Quantité de nombres décomposables en somme de deux carrés
Constante de Catalan	0,9159	$1/1^2 - 1/3^2 + 1/5^2 - 1/7^2 + ...$
1	1	Élément neutre de la multiplication
Constante de la lemniscate	1,3110	Même rôle que π pour cette figure
Nombre d'or φ	1,6180	Divine proportion
Constante de Pythagore $\sqrt{2}$	1,4142	Diagonale du carré unité
$\sqrt{3}$	1,7320	Diagonale du cube unité
e	2,7182	Base du calcul exponentiel
π	3,1416	Aire du cercle de rayon unité
Nombre d'argent	3,246	Racine polynômes chromatiques
Valeurs de Feigenbaum	2,5029 et 4,6692	Valeur d'étirement des figures fractales.

Liste d'autres constantes

1- Accélération normale de la pesanteur	g = 9,8066	m/s^2	
2 - Constante gravitationnelle	G = 6,6725 10^{-11}	$m^3/(s^2.kg)$	
3- Pression normale	H = 1,0132 10^5	Pa	
4- Zéro de l'échelle Celsius des températures	T_0 = 2,7315 10^2	K	
5- Vitesse de la lumière dans le vide	c = 2,9979 10^8	m/s	par définition
6 - Nombre d'Avogadro	N = 6,0221 10^{23}	atomes/atome-gramme ou $gmol^{-1}$	± 0,0004
7 - Rayon classique de l'électron	r_e = 2,8179 10^{-15}	m	
8- Charge de l'électron	e = 1,6021 10^{-19}	coulombs	± 0,000007
9- Masse de l'électron au repos	m_e = 9,1095 10^{-31}	kg	± 0,00005
10- Masse du neutron au repos	m_n =1,6749 10^{-27}	kg	± 0,00001
11- Masse du proton au repos	m_p = 1,6726 10^{-27}	kg	± 0,00001
12- Facteur de conversion de la masse en énergie	1 g = 5,6100 10^{26}	MeV	± 0,00011
13- Constante de Planck	h = 6,62606876 10^{-34}		± 0,00000052
14- Constante de Dirac	hbar = 1,0545 10^{-34}	joules x seconde	
15- Constante de structure fine	α = 0,0072	sans dimension	≅ 1/137
16- Constante de Boltzmann	k =1,380662 10^{-23}	joules /degré absolu	
17- Volume molaire	V_m = 22,4141	l/gmol	
18- Constante universelle des gaz	R = 8,3145	J/(gmol.k)	

Perso.orange.fr

Chapitre 7

Les algorithmes avancés

Objectifs

Résoudre des problèmes présentant des degrés de difficultés croissants.

Plan du chapitre

Chapitre 7

Les algorithmes avancés

I. Introduction

Dans les chapitres précédents, vous avez résolu des problèmes de divers domaines et de difficultés variées. Dans ce chapitre, vous allez apprendre à chercher des solutions à des algorithmes avancés. Les problèmes à résoudre dans ce chapitre appartiennent à des domaines déjà vus, à savoir le tri des données, l'optimisation des solutions et la technique du backtracking. Vous allez découvrir des solutions à ses problèmes, en plus de quelques stratégies que vous alliez développer vous-mêmes.

II. Algorithme de tri : Tri rapide (QuickSort)

II.1 Définitions

Appelé aussi "Tri de Hoare" (du nom de son inventeur), c'est un algorithme de tri considéré comme l'un des plus rapides au niveau de l'exécution.
C'est une méthode de tri **récursive** dont l'efficacité est une fonction croissante du désordre dans le tableau à trier, c'est à dire que plus le tableau est désordonné, plus cette méthode de tri est efficace.

II.2 Principes

Le tri rapide consiste à choisir un élément (appelé **pivot**) et à le mettre à sa place définitive en procédant par permutations d'éléments, de telle sorte que tous ceux qui lui sont inférieurs (ou égaux) soient à sa gauche et que tous ceux qui lui sont supérieurs (ou égaux) soient à sa droite. Cette opération s'appelle **partition**.
Ensuite, nous recommençons récursivement le traitement du tri pour chacun des sous tableaux obtenus tant qu'ils ne sont pas réduits à un seul élément.

Comment choisir le pivot ?

Dans le tri rapide, le choix du pivot est essentiel. Le choix idéal est que le pivot coupe le tableau à trier en deux parties contenant le même nombre d'éléments (c'est à dire la moitié). Comme cela n'est pas possible, plusieurs stratégies sont utilisées pour se rapprocher de la meilleure partition.

Nous pouvons choisir comme pivot l'élément situé au milieu de la partie à trier. Nous pouvons aussi choisir le premier élément. Seulement, cela représente le pire des cas du comportement de la partition et ce, lorsque la séquence est initialement triée.
Il existe bien d'autres stratégies pour choisir le pivot de la partition.

Comment réaliser une partition ?

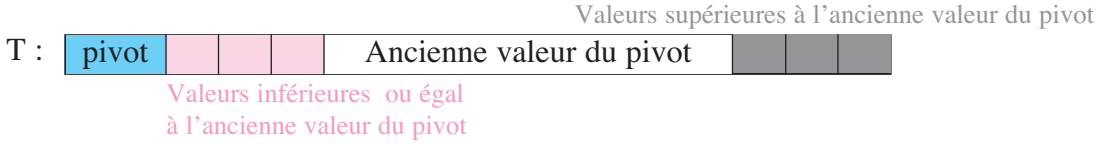
Si nous considérons que le pivot est le premier élément du tableau, alors nous rechercherons à partir du deuxième élément, un élément plus grand que le pivot (que nous appelons G). Une fois trouvé, nous rechercherons en partant du dernier élément en descendant, un élément plus petit que le pivot (que nous appelons P). Nous procédons alors à une permutation des deux éléments G et P.

Nous continuons la recherche d'un élément plus grand que le pivot en montant à partir de la position de G et d'un élément plus petit en descendant à partir de la position de P. Quand l'indice montant devient égal à l'indice descendant, nous arrêtons la recherche et nous permutons le pivot avec l'élément ayant cet indice.

Remarques

1- Le pivot est toujours le premier élément

2- Le tableau est divisé en deux parties : la première partie contient des valeurs inférieures à l'ancienne valeur du pivot et la deuxième des valeurs qui lui sont supérieures.



Activité 1

Utilisez la méthode de tri rapide pour trier en ordre croissant le tableau T suivant :

Soit N = 10

	1	i=2	3	4	5	6	7	8	9	j=10
T initial :	10	6	13	8	19	2	1	31	3	11

- Nous avons choisi comme pivot le premier élément du tableau à trier. Commençons maintenant l'exécution de l'algorithme de la partition :
- Le 2ème élément du tableau est égal à 6, c'est une valeur inférieure au pivot, donc il reste à sa place et nous passons au 3ème élément,
- Le 3ème élément du tableau est égal à 13, c'est une valeur supérieure au pivot (c'est la 1ère valeur supérieure au pivot rencontrée à partir du 2ème élément du tableau et allant vers la droite),
- Fixons cette valeur et son indice, puis commençons une recherche à partir du dernier élément et allant vers la gauche du 1er élément inférieur au pivot,

	1	2	i=3	4	5	6	7	8	j=9	10
T :	10	6	13	8	19	2	1	31	3	11

- La 1ère valeur inférieure au pivot rencontrée à partir du dernier élément du tableau et allant vers la gauche est 3 d'indice 9,

- Permutons maintenant la 1ère valeur supérieure au pivot rencontrée à partir du 2ème élément du tableau et allant vers la droite et la 1ère valeur inférieure au pivot rencontrée à partir du dernier élément du tableau et allant vers la gauche (Permutation de T[3] = 13 et T[9] = 3)

	1	2	i=3	4	5	6	7	8	j=9	10
T :	10	6	**3**	8	19	2	1	31	**13**	11

- Nous continuons maintenant la recherche du premier élément supérieur au pivot, en nous déplaçant vers la droite, mais à partir de la position 4 (= 3+1). L'élément trouvé est égal à 19 d'indice 5,
- De même, cherchons à partir de la position 8 (= 9-1) et en nous déplaçant vers la gauche, le premier élément inférieur au pivot. Il s'agit de la valeur 1 d'indice 7,

	1	2	3	4	i=5	6	j=7	8	9	10
T :	10	6	3	8	19	2	1	31	13	11

- Permutons les valeurs trouvées,

	1	2	3	4	i=5	6	j=7	8	9	10
T :	10	6	3	8	**1**	2	**19**	31	13	11

- Nous continuons la recherche du premier élément supérieur au pivot, en nous déplaçant vers la droite, mais à partir de la position 6 (= 5+1). Cet élément n'existe plus dans la partie non partitionnée du tableau (c'est à dire avant la jème position),

	1	2	3	4	5	i=6	j =7	8	9	10
T :	10	6	3	8	1	2	19	31	13	11

- Nous échangeons alors le ième élément avec l'élément pivot

	1	2	3	4	5	6	j =7	8	9	10
T :	2	6	3	8	1	10	19	31	13	11

- L'élément en position 6 est maintenant à sa place définitive. Il reste à trier récursivement (de la même manière) les morceaux du tableau dont les indices vont de 1 à 5 et de 7 à 10. Commençons par la partie de gauche,
- Nous choisissons comme pivot le premier élément de cette partie du tableau à trier (= 2),

	1	i=2	3	4	j=5	6	7	8	9	10
T :	2	6	3	8	1	10	19	31	13	11

- Recherchons, à partir de la position 2 et en nous déplaçant vers la droite, le premier élément supérieur au pivot. Il s'agit de l'élément 6 d'indice 2,
- De même, cherchons à partir de la position 5 et en nous déplaçant vers la gauche, le premier élément inférieur au pivot. Il s'agit de l'élément 1d'indice 5,

	1	i=2	3	4	j=5	6	7	8	9	10
T :	2	6	3	8	1	10	19	31	13	11

- Permutons les deux valeurs trouvées,

	1	i=2	3	4	j=5	6	7	8	9	10
T :	2	**1**	3	8	**6**	10	19	31	13	11

- Nous continuons maintenant la recherche du premier élément supérieur au pivot, en nous déplaçant vers la droite, mais à partir de la position 3 (= 2+1). L'élément trouvé est égal à 3 d'indice 3,

	1	2	i=3	4	j=5	6	7	8	9	10
T :	2	1	3	8	**6**	10	19	31	13	11

- De même, cherchons à partir de la position 4 (= 5-1) et en nous déplaçant vers la gauche, le premier élément inférieur au pivot. Cet élément n'existe plus dans la partie non partitionnée (c'est à dire entre la $i^{ème}$ et $j^{ème}$ position),
- Le compteur j à décrémenté jusqu'à égalité avec le compteur i sans trouver une valeur inférieure au pivot. Donc, aucun échange ne sera effectué.
- L'élément en position 3 ($i^{ème}$ et $j^{ème}$ position confondues) est maintenant à sa place définitive. Il reste à trier récursivement (de la même manière) les morceaux de la $1^{ère}$ partie du tableau dont les indices vont de 1 à 2 et de 4 à 5. Commençons par la partie de gauche,

	1	2	3	4	j=5	6	7	8	9	10
T :	2	1	3	8	6	10	19	31	13	11

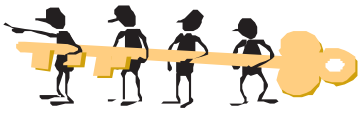
- Nous choisissons comme pivot le premier élément de cette partie du tableau à trier (= 2),

	1	i=j=2	3	4	5	6	7	8	9	10
T :	1	2	3	8	6	10	19	31	13	11

- Comme i = j, il y aura un échange entre le pivot et le $i^{ème}$ élément car il lui est inférieur,
- Les sous tableaux obtenus sont réduits à une seule case, donc ils sont forcément triés. D'où, la partie du tableau dont les indices vont de 1 à 3 et maintenant triée. C'est pour cela, nous allons s'intéresser maintenant à la partie droite dont les indices allant de 3 à 5,

Activité 2

Donnez la suite des étapes pour trier tout le tableau T.



Nous proposons d'utiliser la méthode de tri rapide pour trier en ordre croissant un tableau T de n entiers ;

a) Proposez une analyse de ce problème.
b) Déduisez les algorithmes correspondants.
c) Traduisez la solution en Pascal.

a) Analyse du problème

Résultat : Affichage du tableau T trié, réalisé par la procédure Affiche
Traitement : Tri, en utilisant la méthode du tri rapide, du tableau T. C'est la tâche de la procédure récursive Tri qui fait appel à une procédure Partition permettant de diviser un tableau en deux parties selon le principe expliqué précédemment.
Données : un tableau T à remplir par n entiers grâce à la procédure Remplir

b) Algorithme du programme principal et codification des objets

0) Début Tri_Rapide
1) Proc Remplir (T, n)
2) Proc Tri (T, 1 ,n)
3) Proc Afficher (T, n)
4) Fin Tri_Rapide

Tableau de codification des nouveaux types

Type
Tab = Tableau de n entiers

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
N	Entier non signé	Nombre d'éléments du tableau T
T	Tab	Représentant le tableau à trier
Remplir	Procédure	Permet de remplir le tableau T par n entiers
Tri	Procédure	Permet de trier le tableau T en utilisant la méthode du Tri Rapide
Affiche	Procédure	Permet d'afficher le tableau trié

c) Analyse de la procédure Tri

Résultat : Tableau trié

Traitement : g représente le point de départ gauche et d le point de départ droit. Le traitement s'arrête lorsque les deux points g et d se rencontrent (d-g = 0)

Le pivot est fixé au 1er élément du tableau.

Nous utiliserons deux compteurs temporairement : L pour avancer à partir de g+1 (car g est le pivot) et M pour reculer à partir de d.

L ← g + 1 ; M← d

Nous chercherons à partir de L, le 1er élément supérieur au pivot s'il existe.

Tant que (L < M) ET (T[g] > T[L]) Faire

L ← L +1

Maintenant, nous allons chercher le 1er élément inférieur au pivot, mais à partir de la droite (M)

Tant que (L ≤ M) ET (T[g] ≤ T[M]) **Faire**

M ← M-1

Si les indices des deux éléments trouvés sont différents alors nous permutons leurs contenus.

Si (L < M) **Alors**

aux ← T[L]

T[L] ←T[M]

T[M] ← aux

Nous incrémentons L de 1 et nous décrémentons M de 1.

L ← L+1 , M ← M-1

Nous répétons ce traitement jusqu'à croisement de L et M (M < L). Ensuite, nous permutons le pivot avec l'élément du croisement : T[g] et T[M]

Nous allons appeler récursivement le traitement précédent deux fois : la 1ère pour trier la partie de g à M-1 et la 2ème pour trier la partie de M +1 à d

Tri (g, M-1)

Tri (M+1, d)

d) Algorithme de la procédure Tri

```
0) Procédure Tri (d, g : Entier)
1) Si d-g > 0 Alors
     L ← g + 1 , M ← d
     Répéter
        Tant que (L < M) ET (T[g] > T[L]) Faire
             L ← L+1
        Fin Tant que

        Tant que (L ≤ M) ET (T[g] ≤ T[M] ) Faire
                    M ← M-1
         Fin Tant que
        Si (L < M) Alors
          aux ← T[L]
          T[L] ← T[M]
          T[M] ← Aux
        Fin Si
        L ← L+1 , M ← M-1
     Jusqu'à (M < L)

     aux ← T[g]
     T[g] ← T[M]
     T[M] ← aux

     Proc Tri (g, M-1)
     Proc Tri (M+1, d)

   Fin Si
2) Fin Tri
```

Activité 3

Traduisez la procédure Tri et l'intégrez dans un programme Pascal permettant de trier le tableau T. Enregistrez votre programme sous le nom **tri_rap**.

III . Les tours de Hanoï

III.1 Définition

Le problème ou le jeu des tours de Hanoï consiste en une plaquette de bois sur laquelle sont plantés 3 piquets. Le premier porte n disques de diamètres différents deux à deux, empilés du plus grand au plus petit.

Le problème des tours de Hanoï consiste à faire passer tous ces disques au piquet B en s'aidant du troisième piquet et ceci en respectant les règles suivantes :

- Déplacer un seul disque à la fois
- Déplacer un disque uniquement sur un disque plus grand

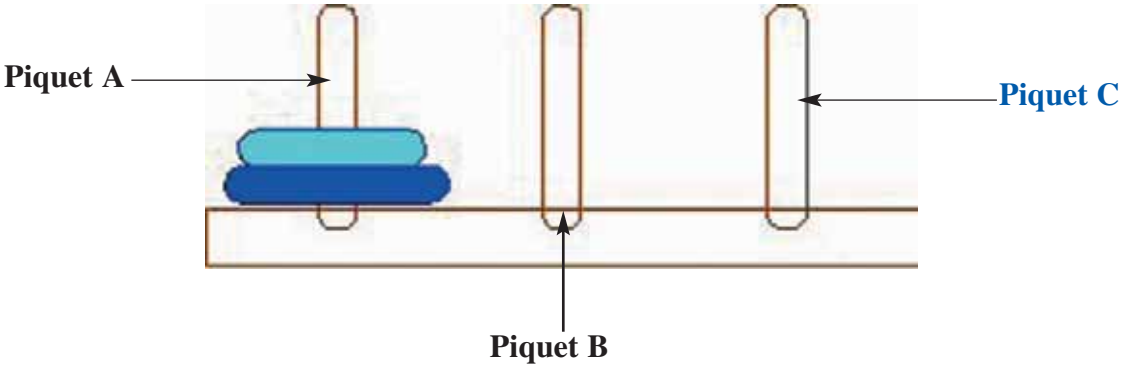
III.2 Principe

Le but de ce jeu est de bouger tous les disques vers un nouveau piquet en se servant d'un piquet intermédiaire et en respectant les règles suivantes :
- Nous ne pouvons déplacer qu'un disque à la fois.
- Nous ne pouvons déplacer un disque qui se trouve sous un autre.
- Un disque ne peut être déposé sur un disque plus petit.

Et tout ceci bien entendu, en un minimum de déplacements.

Constatation

Il existe un algorithme récursif très classique pour résoudre ce problème. Supposons que nous sachions déplacer (n-1) disques. Pour en déplacer n, il suffit de :
- déplacer (n-1) disques du piquet A au piquet C,
- puis de déplacer le grand disque du piquet A au piquet B,
- nous terminons par déplacer les (n-1) autres disques du piquet C vers le piquet B.



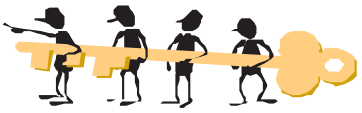
Chapitre

7

Activité 1

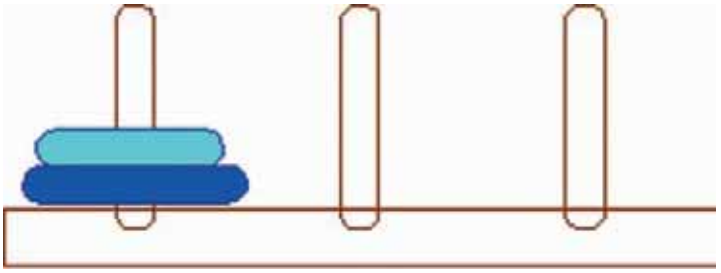
a- Représentez graphiquement toutes les étapes pour un jeu à 2 disques (n=2) sachant que le piquet initial est A, le piquet final est B et l'intermédiaire est C.

b- En déduire le nombre de déplacements pour arriver à l'état final

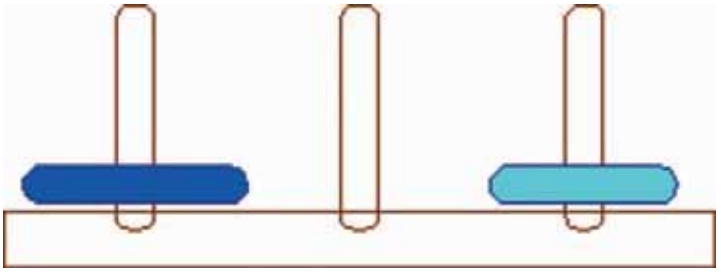


a- Représentation graphique :

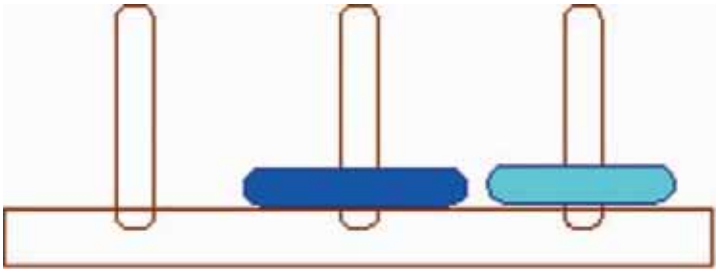
Etat initial



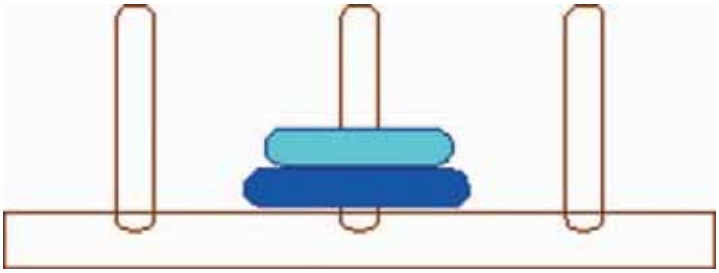
Premier déplacement



Deuxième déplacement



Troisième déplacement : état final



En résumé, pour deux disques, nous avons la solution suivante, qui comporte 3 déplacements : de A vers C, de A vers B et enfin de C vers B.

Activité 2

a- Donnez une solution pour un jeu à 3 disques (n=3).
b- Donnez le nombre de déplacements pour arriver à l'état final puis en déduire celui pour n disques.

a- Opérations de déplacement :

1/ disque 1 : de A vers B
2/ disque 2 : de A vers C
3/ disque 1 : de B vers C
4/ disque 3 : de A vers B
5/ disque 1 : de C vers A
6/ disque 2 : de C vers B
7/ disque 1 : de A vers B

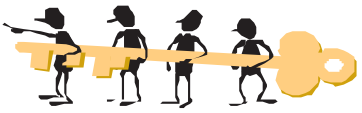
b- Le nombre de déplacements de disques nécessaire pour arriver à l'état final est 7.
Le tableau suivant donne le nombre de déplacements à faire pour ranger un nombre de disques allant de 1 à 9.

Nombre de disques	1	2	3	4	5	6	7	8	9
Nombre de déplacements	1	3	7	15	31	63	127	255	511
Constatations	2^1-1	2^2-1	2^3-1	2^4-1	2^5-1	2^6-1	2^7-1	2^8-1	2^9-1

D'une façon générale, le nombre de déplacements pour n disques est 2^n - 1.

Activité 3

a- Proposez une analyse d'une procédure récursive nommée **Hanoï** permettant de réaliser ce traitement pour n de l'intervalle [1,64].
b- Déduisez l'algorithme correspondant,
c- Traduisez et testez la solution obtenue.



a) Analyse

En regardant les déplacements effectués et en se basant sur la constatation du paragraphe III-1, nous pouvons déduire que :
Pour déplacer n disques du piquet A au piquet B, nous suivons la démarche suivante :
- nous déplaçons (n-1) disques vers le piquet C
- nous déplaçons le disque n du piquet A vers le piquet B
- nous déplaçons les (n-1) disques du piquet C vers le piquet B.

En utilisant une procédure récursive appelée **Hanoï**, ceci peut être traduit par :
Si (n > 0) **Alors**
  Hanoï (n-1, départ, intermédiaire, cible)
  Ecrire ("Disque du piquet ", départ, " vers le piquet ", intermédiaire)

Puis de déplacer le grand disque du piquet 1 au piquet 2,
Nous terminons en déplaçant les (n-1) autres disques du piquet 3 vers le piquet 2.
  Hanoï (n-1, cible, départ, intermédiaire)
Fin Si

b) Algorithme

0) Algorithme Procédure Hanoï (n : octet ; départ, cible, intermédiaire : octet)
1) **Si** (n > 0) **Alors**
    Proc Hanoï (n-1, départ, intermédiaire, cible)
    Ecrire ("Disque du piquet ", départ " vers le piquet ", intermédiaire)
    Proc Hanoï (n-1, cible, départ, intermédiaire)
  Fin Si
2) Fin Hanoï

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
N	Octet	Nombre de disques
départ	Octet	piquet 1
cible	Octet	piquet 2
intermédiaire	Octet	piquet 3

c- Traduction en Pascal

```
Program Tours_Hanoi;
Uses Crt ;
Var  n : Byte ;

Procedure Hanoi (n: Byte ; depart, but, inter : Byte);
Begin
  If (n > 0) Then
  Begin
    Hanoi (n-1, depart, inter, but) ;
    WriteLn ('déplacer un disque de ', depart, '  en  ', inter) ;
    Hanoi (n-1, but, depart, inter) ;
  End;
End;

{Programme Principal}
BEGIN
    Repeat
        WriteLn ('Combien de disques  ?' ); ReadLn (n) ;
    Until  n IN [1 .. 64] ;
    Hanoi (n, 1, 3, 2);
END.
```

Activité 4

Effectuez les modifications nécessaires dans le programme pour afficher le nombre de déplacements ?

Activité 5 : La légende

Selon une légende très ancienne, il existe un temple où les moines sont chargés de veiller sur 64 disques sacrés. Les disques, qui sont tous de tailles différentes, forment une tour. Comme ils sont précieux et très fragiles, un disque ne peut être placé sur un disque plus petit.

Hélas, le jour vient où quelques travaux dans le temple sont nécessaires et les disques doivent être déplacés. Ils sont très lourds et ne peuvent donc être transportés qu'un par un. De plus, il n'y a qu'un seul endroit assez sacré pour les stocker.

Les moines commencent donc à déplacer les disques de la tour d'origine vers la nouvelle place et l'endroit intermédiaire, gardant toujours chacune des trois piles en ordre (le disque le plus large en bas, le moins large en haut).

Alors qu'ils travaillent, les moines gardent en tête la terrible prophétie :

Chapitre

7

"Le temple s'écroulera avant que les disques soient mis en place."

Qu'en pensez-vous?

IV -Le problème du voyageur de commerce (PVC)

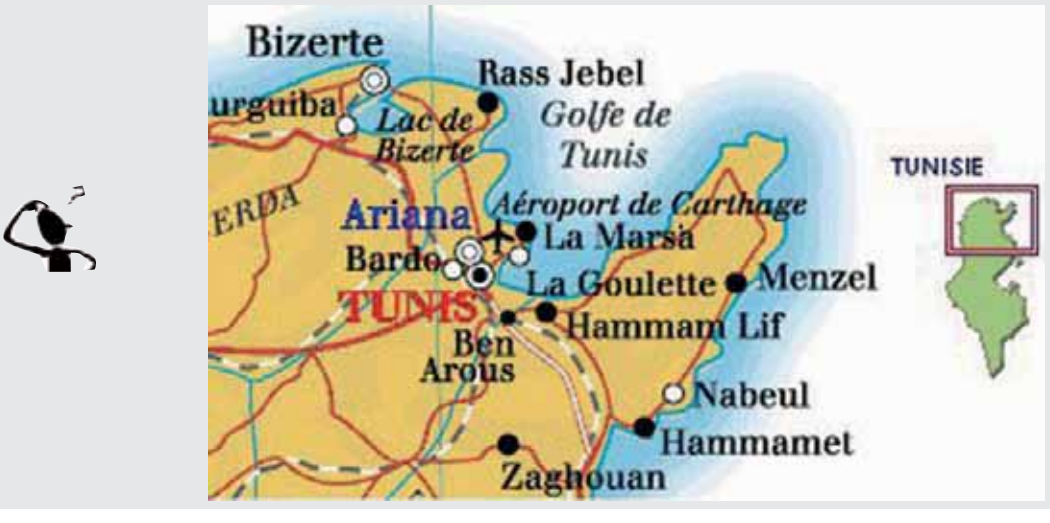
IV.1 Présentation

Un voyageur de commerce doit visiter n villes données en passant par chaque ville une et une seule fois. Il commence par une ville quelconque et termine en retournant à cette ville de départ. Les distances entre les villes sont connues.

Quel chemin faut-il choisir afin de minimiser la distance parcourue ?

La notion de distance peut-être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense.

Autrement dit, le problème consiste à trouver le trajet le plus court pour passer par un certain nombre de villes et revenir à la ville de départ.



Il existe diverses solutions donnant le chemin le plus court et qui nécessitent l'utilisation des structures de données avancées tel que les graphes, les arbres, etc. qui ne font pas partie de votre programme d'informatique de cette année.

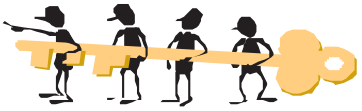
Nous allons voir une solution en se basant sur les structures de données déjà étudiées et dont la stratégie repose sur le fait qu'à une étape donnée, la prochaine ville à visiter sera la ville la plus proche que le commerçant n'a pas encore visité.

Autrement dit, pour déterminer la ville à visiter, il suffit de comparer les différentes distances entre la ville en cours Vc et les autres villes en vue de déterminer la ville non encore visitée et dont la distance qui la sépare de la ville Vc est minimale.

IV.2 Résolution

Activité 1

Proposez une analyse, puis déduisez les algorithmes d'un programme permettant d'afficher l'itinéraire que doit suivre un voyageur de commerce pour visiter n villes.
Nous supposons que le nombre de villes à visiter est inférieur ou égal à 20.



a) Analyse du problème

Résultat : Afficher l'itinéraire (les noms des villes visitées dans l'ordre).
Traitement : Affichage de la liste des n villes visitées : la procédure Affichage_Trajet.
- Former l'itinéraire des ville visitées : la procédure Former_Trajet.
- La saisie du nombre de villes à visiter n ainsi que de la liste des noms des n villes et des distances qui les séparent : la procédure Saisie_liste.
- Saisir le nom de la ville de départ : Saisie_ville_depart.

Les structures de données à utiliser :
- Nous fixons tout d'abord à 20 le nombre maximum des villes à visiter.
- Pour une ville, il nous faut son **nom** (une chaîne de caractères) et les **distances** qui les séparent des n villes à visiter. Nous utiliserons deux tableaux :
- un vecteur **Nom_Ville** de type Chaîne de caractères pour ranger les noms des villes
- une matrice carrée **Distance** pour ranger les distances entre les différentes villes.
- Un vecteur **Visite** de type booléen pour vérifier si une ville donnée a été déjà visitée. Ce vecteur sera initialisé à **Faux** au début du traitement et lors d'une visite d'une ville i, **Visite[i]** sera mis à **Vrai**.
- Un vecteur **Trajet** de n entiers représentant les numéros des villes visitées en ordre.
- Une variable **v_départ** pour saisir la ville de départ.
- Le nombre de villes **n**

Remarque
Pour alléger le traitement, les villes sont repérées par leurs indices variant de 1 à n.

b/ Algorithme du programme principal et codification des objets

0) Début voyageur
1) Proc Saisie_liste (n, Nom_Ville, Distance)
2) Proc Saisie_ville_départ (v_départ)
3) Proc Former_Trajet (n, Distance,v_départ,Visite,trajet)
4) Proc Affichage_Trajet (Trajet, n,Nom_Villes)
5) Fin voyageur

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
nbmaxville	Constante = 20	Nombre maximum de villes à visiter
v_départ	Entier	Numéro de la ville de départ
n	Entier	Le nombre de villes à visiter
Nom_Villes	NV	Tableau de n chaînes de caractères contenant les noms des villes à visiter
Distance	Dist	Matrice nxn contenant des réels représentant les distances entre les différentes villes à visiter
Visite	Visit	Vecteur de n booléens
Trajet	Traj	Vecteur de n entiers représentant les numéros des villes visitées dans l'ordre
Saisie_ville_départ	Procédure	Saisit le numéro de la ville de départ
Saisie_liste	Procédure	Saisit du nombre de villes à visiter ainsi que des noms des n villes et des distances qui les séparent deux à deux
Former_Trajet	Procédure	Former l'itinéraire ayant la distance minimale
Afficher_Trajet	Procédure	Afficher les noms des villes de l'itinéraire

c/ Analyse de la procédure Saisie_Liste

Résultat : Saisie du nombre de villes n et de la liste des noms et des distances qui séparent les n villes
Traitement : Saisir le nombre n de villes
Répéter
Ecrire (" Combien de villes voulez vous visiter ? ")
Lire (n)
Jusqu'à (n dans [1.. nbmaxville])

```
Saisir les noms et les distances qui séparent une ville donnée des autres villes
    Pour i de 1 à n Faire
        Lire (Nom_Ville[i])
        Pour de c de 1 à n Faire
            Lire (Distance[i,c])
```

Algorithme la procédure Saisie_liste

```
0) PROCEDURE Saisie_liste ( Var n : Entier ; Var Nom_Ville : NV ; Var Distance  : Dist)
1) Répéter
     Ecrire (" Combien de villes voulez vous visiter ?  ")
     Lire (n)
   Jusqu'à (n dans [1.. nbmaxville])
2) Pour  i de 1 à n Faire
     Lire (Nom_Ville[i])
     Pour de c de 1 à n Faire
        Lire (Distance[i,c])
     Fin Pour
   Fin Pour
3) Fin Saisie_liste
```

d) Analyse de la procédure Saisie_ville_départ

Résultat : Saisie du numéro de la ville de départ
Traitement : Saisir le numéro de la ville de départ.

```
Répéter
  Ecrire ("Veuillez saisir le numéro de la ville de départ ?  ")
  Lire (numv)
Jusqu'à (numv dans [1..n])
```

Algorithme de la procédure Saisie_ville_départ

```
0) PROCEDURE Saisie_ville_depart (Var numv : Entier)
1)Répéter
    Ecrire ("Saisir le numéro de la ville de départ ?  ")
    Lire (numv) ;
  Jusqu'à (numv dans [1..n])
2) Fin Saisie_ville_départ
```

e) Analyse de la procédure Former_Trajet

Résultat : Remplir le vecteur par les noms des villes formant l'itinéraire en fonction de leur priorité de visite
Traitement : Nous allons appliquer la stratégie déjà décrite, qui consiste à visiter la ville la plus proche qui n'a pas été encore visitée.

- Commençons par initialiser le vecteur Visite à **Faux**

```
Pour i de 1 à n Faire
    Visite[i] ← Faux
```

- Ranger le numéro de la ville de départ comme premier élément du vecteur Trajet

```
Trajet[1] ← v_départ
```

- Déterminer les villes à visiter pour que la distance parcourue soit minimale

```
Pour i de 2 à n Faire
    j ← FN Ville_proche_non_visit (i, Distance, n)
    Trajet[i] ← j
```

Algorithme de la procédure Former_trajet

```
0) PROCEDURE Former_Trajet (n : Entier ; Distance : Dist ; v_départ : Entier ;
   Visite : Visit ; Var Trajet : Traj)
1) Pour i de 1 à n Faire
     Visite[i] ← Faux
   Fin Pour
2) Trajet[1] ← v_départ
3) Pour i de 2 à n Faire
     j ← FN Ville_proche_non_visit (i, Distance, Visite, n)
     Trajet[i] ← j
     Visite[j] ← Vrai
   Fin Pour
4) Fin Former_Trajet
```

f) Analyse de la fonction Ville_proche_non_visit

Résultat : Chercher l'indice de la ville la plus proche de la ville n°i et qui n'a pas été encore visitée

Traitement :

- Etape d'initialisation :

Parcourir le vecteur Visite et initialiser la variable ind par le numéro de la première ville non encore visitée.

```
j ← 1
Tant que (Visite[j] = Vrai) Faire
     j ← j + 1
Fin Tantque
ind ← j
```

- Continuer le parcours du vecteur Visite pour chercher l'indice de la ville la plus proche non encore visitée

```
Pour j de j+1 à n Faire
     Si (Visite[j]=Faux) et (Distance[i,j] < Distance[i,ind]) Alors
          ind ← j
Ville_proche_non_visit ← ind
```

g) Analyse de la procédure Affichage_Trajet

Résultat : Afficher les noms des villes formant l'itinéraire du commerçant
Traitement :
- Parcourir le vecteur Trajet et afficher les noms des villes correspondant dans le vecteur Nom_Villes

Pour i de 1 à n Faire
Ecrire (Nom_Villes[Trajet[i]])

Algorithme de la procédure Affichage_Trajet

0) Procédure Affichage_Trajet (Trajet : Traj ; n :Entier ; Nom_Villes : NV)
1) **Pour** i de 1 à n **Faire**
Ecrire (Nom_Villes[Trajet[i]])
Fin Pour
Ecrire (Nom_Villes[Trajet[1]])
2) Fin Affichage_Trajet

Activité 2

Traduisez et testez la solution de ce problème.

```
PROGRAM voyageur ;
USES Crt ;
Const  nbmaxville = 20 ;

Type
 NV = Array [1..Nbmaxville] of String[20];
 Dist = Array [1..nbmaxville,1..nbmaxville] of real;
 Visit = Array [1..nbmaxville] of Boolean;
 Traj = Array [1..Nbmaxville] of integer;

VAR
 N, V_depart  : Integer ;
 Nom_villes : NV;
 Distance : Dist ;
 Visite : Visit ;
 Trajet : Traj ;

{*********************************************************}

PROCEDURE Saisie_liste (Var n : Integer; Var Nom_Ville : NV; Var
Distance : Dist);
```

```
Var
      i,c : Integer ;
Begin
Repeat
Write (' Combien de villes voulez vous visiter ?  ') ;
Readln (n)
Until (n in [1.. nbmaxville]);

For i := 1 To n Do
Begin
  Write('Nom de la ville ',i);
  Readln (Nom_Ville[i]);
  For c := 1 To n Do
  Begin
    Write('Donner la distance entre les villes ',i,' et ',c, ': ');
     Readln(Distance[i, c])
  End ;
End ;
End ;

{************************************************************}

PROCEDURE  Saisie_ville_depart  (n : Integer ; VAR numv : Integer) ;
Begin
Repeat
Write ('Saisir le numéro de la ville de départ ? ') ;
Readln (numv) ;
Until (numv in [1..n]);
End;

{************************************************************}

Function  Ville_proche_non_visit  (i  :  Integer;  Distance  :  Dist;
Visite : Visit; n : Integer) : Integer ;
Var
  j , ind : Integer ;
Begin
j := 1 ;
While (Visite[j] = True) Do
j := j + 1 ;

ind := j ;

For j := (j+1) to n Do
If (Visite[j]=False) and (Distance[i,j] < Distance[i,ind]) then ind
:= j ;
Ville_proche_non_visit := ind
End ;

{************************************************************}
```

```
PROCEDURE Former_Trajet (n : Integer; Distance : Dist; v_depart :
Integer; Visite : Visit; Var Trajet : Traj);
Var
  i , j : Integer ;
Begin
For i := 1 to n Do
Visite[i] := False ;

Trajet [1]:= v_depart ;
Visite[v_depart] := True ;

For i := 2 to n Do

Begin
j := Ville_proche_non_visit (i,Distance,Visite,n) ;
Trajet[i] := j;
Visite[j] := True
End ;
End ;

{************************************************************}

Procedure Affichage_Trajet (Trajet : Traj; n :Integer ; Nom_Villes : NV);
Var
      i : Integer ;
Begin
For i := 1 to n Do
  Write (Nom_Villes[Trajet[i]]);
Write (Nom_Villes[Trajet[1]]);
End ;

{ **************    programme principal    ************************}
BEGIN
  Saisie_liste (n, Nom_Villes, Distance);
  Saisie_ville_depart (n, v_depart);
  Former_Trajet (n, Distance, v_depart, Visite,trajet);
  Affichage_Trajet (Trajet, n, Nom_Villes);
END.
```

Activité 3 :

Modifiez le programme pour qu'il accepte un nombre de villes supérieur à 20 et comparez les temps d'exécution en fonction de ce nombre.

Activité 4

Faites des recherches sur Internet pour créer un dossier sur le voyageur de commerce.

Activité 5

Proposez une autre solution en appliquant la même stratégie de résolution et en changeant les structures de données utilisées. Vous pouvez penser à utiliser la structure enregistrement pour les données d'une ville (son nom, ses distances aux restes des villes).
Traduisez votre proposition en Pascal.

V -Le problème des huit dames :

V.1 Présentation

Le problème des huit dames (dit aussi problème des huit reines), consiste à placer huit reines d'un jeu d'échecs sur un échiquier de 8 x 8 cases sans que les dames ne puissent se menacer mutuellement. Par conséquent, deux dames ne devraient jamais partager la même rangée ou la même colonne ou la même diagonale.

Dans le cas général, le problème des dames consiste à placer n dames sur un échiquier de taille n x n, toujours de telle sorte qu'aucune dame ne soit prise : il ne faut donc pas plus d'une dame par ligne, par colonne et par diagonale.

Histoire

Durant des années, beaucoup de mathématiciens ont travaillé sur ce problème, qui est un cas particulier du problème généralisé des n-dames, posé en 1850 par Franz Nauck, et qui consiste à placer n dames sur un échiquier de n x n cases. En 1874, S. Gunther proposa une méthode pour trouver des solutions en employant des déterminants et J.W.L. Glaisher affina cette approche.

Solutions

Pour n dames nous avons les solutions suivantes :

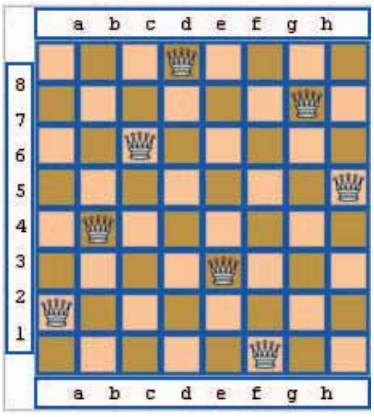
n	Solutions
1	1
4	2
7	40
10	724
13	73712

n	Solutions
2	0
5	10
8	92
11	2680
14	365596

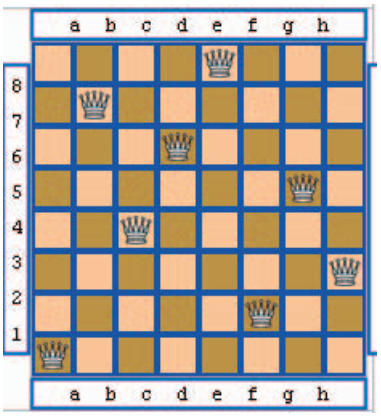
n	Solutions
3	0
6	4
9	352
12	14200
15	2279184

Le problème des huit dames a 92 solutions distinctes.

Voici une première solution :



Voici une deuxième solution :



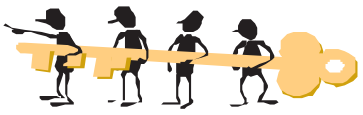
Activité 1

Représentez, sous forme d'échiquier, d'autres solutions du problème de 8 dames.

V.2 Résolution du problème

Activité 2

Proposez une analyse, puis déduisez les algorithmes d'un programme permettant d'afficher les positions et le nombre de solutions pour le problème des 8 dames.



a) Analyse du programme principal

Résultat : Ecrire ("nombre de solutions = ", solution)

Traitement : Nous appelons une procédure récursive Place_dame pour placer les huit dames sur l'échiquier, une à une.

Le tableau TD de 8 cases, est initialisé à zéro après appel de la procédure Init

Algorithme du programme principal

```
0) Algorithme Huit Dames
1) solution ← 0          { initialisation du nombre de solutions possibles }
2) Proc Init (TD, nbd)
3) Proc Place_dame (TD, 1)
4) Ecrire ("Nombre de solutions = ", solution )
5) Fin Huit Dames
```

Tableau de codification des nouveaux types

Type
Tab = Tableau de nbd entiers

Tableau de codification des objets globaux

Objets	Type / Nature	Rôle
nbd	Constante = 8	Nombre de dames
TD	Tab	tableau des positions des dames
solution	Entier	nombre de solutions trouvées
Init	Procédure	Initialise à zéro le tableau
Place_dame	Procédure	Position des dames dans l'échiquier

b) Analyse de la procédure Init

Résultat : Initialiser à zéro tous les éléments du tableau T
Traitement :

```
Pour i de 1 à ndames Faire
    T[i] ← 0
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
i	Entier	compteur

Algorithme de la procédure init

```
0) Procédure Init (Var T : Tab ; ndames : Entier)
1) Pour i de 1 à ndames Faire
       T[i] ← 0
   Fin Pour
2) Fin Init
```

c) Analyse de la procédure Place_dame

Résultat : Remplir le tableau T par les positions des dames et afficher les différentes solutions trouvées.
Traitement :
Pour i de 1 à nbd **Faire**
La procédure Place_dame place une dame dans chacune des positions libres de l'échiquier, la position trouvée sera notée dans la première case du tableau T.
Dans le tableau T, l'indice i indique la ligne de l'échiquier et T[i] indique la colonne de l'échiquier.

Le test suivant évite de mettre une dame sur la même ligne et sur la même colonne, i représente le numéro de la ligne et j représente le numéro de la colonne.
Pour j de 1 à ndames **Faire**
Si ((T[j] = i) ou (ABS (T[j] - i) = ABS (j - ndames)))
Alors arrêter le traitement, car deux dames se trouvent sur la même ligne, la même colonne ou la même diagonale, puis passer à la recherche d'autres positionnements. Pour ce faire, la procédure prédéfinie **Continue** est utilisée (disponible dans les versions 5, 6 et 7 de Turbo Pascal), provoquant le passage immédiat à l'itération suivante, par le biais d'un branchement inconditionnel **Aller à** (en Pascal **GoTo**), permettant le saut vers une étiquette appelé **suivant**.

S'il n'y a pas de menace, nous sauvegardons cette position dans le tableau T et nous appelons le même traitement pour la dame suivante. Donc, ce traitement est récursif.

```
T[nDames] ← i
Proc Place_dame (T, ndames+1)
```

Quand la huitième dame est placée à la bonne case, on appelle la procédure Affiche pour afficher les huit dames placées sur l'échiquier :

```
Si (ndames = nbd +1) Alors
    Proc Affiche (T, nbd)
    solution ← solution + 1
```

Tableau de codification des objets locaux

Objets	Type / Nature	Rôle
suivant	Label	Nom de l'étiquette
i, j	Entiers	Compteurs
Affiche	Procédure	Affiche les 8 dames sur un échiquier

Algorithme de la procédure place_dame

```
0) Procedure Place_dame (T : Tab ; ndames : Entier)
1) Si (ndames = nbd+1) Alors
        Proc Affiche (T, nbd)
        solution ← solution + 1
   Fin Si
2) Pour i de 1 à nbd Faire
        Pour j de 1 à ndames Faire
            Si ((T[j] = i) ou (Abs (T[j] - i) = Abs (j - ndames))) Alors
                Aller à Suivant
            Fin Si
        Fin Pour
        T[ndames] ← i
        Proc Place_dame (T, ndames+1)

     suivant : Continue
   Fin pour
3) Fin Place_dame
```

d) Analyse de la procédure Affiche

Résultat : Afficher les positions des dames sur l'échiquier

Traitement : Nous parcourons le tableau T contenant les positions de l'emplacement des différentes dames. Si la dame existe, nous affichons "1" sinon nous affichons "0"

```
Pour i de 1 à nbd Faire
    Pour j de 1 à nbd Faire
        Si (j = T[i]) Alors Ecrire ("1") Sinon Ecrire ("0")
```

Tableau de codification des objets locaux

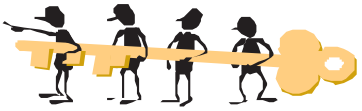
Objets	Type / Nature	Rôle
i, j	Entier	compteurs

Algorithme de la procédure Affiche

```
0) Procédure Affiche (T : Tab ; nbd : Entier)
1) Pour i de 1 à nbd Faire
        Pour j de 1 à nbd Faire
            Si (j = T[i]) Alors Ecrire ("1") Sinon Ecrire ("0")
            Fin Si
        Fin Pour
   Fin Pour
2) Fin Affiche
```

Activité 3

Traduisez et testez la solution du problème.



```
Program huitdames ;
Uses crt ;
Const  nbd = 8 ;
Type Tab=array [1..nbd] of Integer;
VAR
 TD: Tab;
 solution :Integer ;

{*****************************************************************}

PROCEDURE init (VAR T : Tab; ndames: Integer );
VAR     i : Integer ;
BEGIN
      for i:=1 to ndames do T[i]:=0;
End ;

{*****************************************************************}
PROCEDURE affiche (T: Tab ; nbd:Integer);
VAR     i, j : Integer ;
BEGIN
      for i:=1 to nbd do
      Begin
            for j:=1 to nbd do
                  if (j=T[i]) then Write ('1') else Write ('0');
            WriteLn;
      End;
      WriteLn ;
End;

{*********************************************************}
PROCEDURE place_dame (T : Tab; nDames : Integer);
Label    suivant ;
VAR   i, j : Integer ;
BEGIN
  If (nDames= nbd+1) Then
      Begin
                  affiche (T, nbd) ;
                   solution := solution + 1 ;
      End ;
```

```
   For i:=1 to nbd do
   Begin
     For j:=1 to ndames do
       If((T[j]= i) OR (abs(T[j]-i) = abs(j-nDames))) Then  GoTo suivant ;
     T[nDames]:=i ;
     place_dame (T, ndames+1);
     suivant : continue;
   End;
End;

{ **********    Programme Principal ***********  }
BEGIN
 solution:=0 ;
 init (TD, nbd);
 place_dame (TD,1);
 WriteLn ('Nombre de solutions = ' , solution );
END.
```

Activité 4

Exécutez le programme en variant le nombre de dames et comparez le temps d'exécution et le nombre de solutions constatés.

Activité 5

Faites des recherches sur Internet pour créer un dossier sur le problème des huit dames.

Retenons

• Les algorithmes cités dans ce chapitre sont d'un degré de difficultés élevé, à savoir :

- Une méthode de tri : Le tri rapide (en anglais QuickSort),

- Des algorithmes avancés d'optimisation tels que les tours de Hanoï et le voyageur du commerce,

- Un algorithme faisant appel à la technique du retour sur trace (backtracking) tel que le problème des huits dames,

• Le tri rapide consiste à trier un tableau en le découpant récursivement en 2 sous tableaux, où les éléments du premier tableau sont inférieurs aux éléments du second. Le tableau initial est donc découpé en sous tableaux de plus en plus petits, jusqu'à arriver à un sous tableau à un seul élément, qui est donc forcément trié. Ensuite, en remontant, les éléments sont triés petit à petit.

• Le problème des tours de Hanoï est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de «départ» à une tour d'« arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups.

• Le voyageur de commerce souhaite passer par toutes les villes d'un parcours, c'est à dire par le trajet le plus court, sans passer deux fois au même endroit. La solution de ce problème répond à la question suivante : Quel est le trajet le plus court qui passe par toutes les villes?

• Le retour sur trace, (en anglais backtracking), consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage.

• Le problème des huit dames consiste à placer huit dames sur un échiquier sans qu'elles puissent se prendre entre elles.

Exercice 1

Le tri comptage (appelé aussi tri casier) est un algorithme de tri particulier s'appliquant à des valeurs généralement entières. Cette méthode permet de classer les éléments d'un tableau en ordre croissant (ou décroissant) en réalisant les étapes suivantes :

- Créer un tableau intermédiaire contenant le nombre d'occurrences de chaque valeur du tableau initial. Les valeurs de l'indice du tableau intermédiaire correspondent aux différents éléments du tableau initial,
- Reconstruire le tableau initial à partir du tableau intermédiaire déjà créé.

Exemple

Tableau initial :

i	1	2	3	4	5	6	7	8	9	10
T[i]	10	2	13	8	9	2	13	3	13	11

Tableau intermédiaire :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T_intermediaire[i]	0	0	2	1	0	0	0	0	1	1	1	1	0	3

Tableau reconstitué :

i	1	2	3	4	5	6	7	8	9	10
T[i]	2	2	3	8	9	10	11	13	13	13

QUESTIONS :

1- Appliquez cette méthode de tri pour ordonner les éléments du tableau T suivant :

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
T[i]	2	12	3	18	9	10	1	13	3	13	1	9	9	22	11	6	12	3	4	5	6	5	5	3	5	6

2- Nous proposons d'écrire un programme en Pascal qui permet de trier n éléments de type entier d'un tableau T en utilisant la méthode du tri comptage.

a) Décomposez ce problème en modules et écrivez un algorithme pour chacun des modules proposés.

b) Traduisez la solution en un programme intitulé TRI_COMPTAGE

Exercice 2

Il existe d'autres méthodes de tris telles que le tri par création, tri radix, tri par permutation, etc.

1- Utilisez Internet pour chercher les algorithmes formels de ces méthodes.

2- Analysez puis déduisez un programme en Pascal pour chacune de ces méthodes.

Exercice 3

Le but de cet exercice est de trouver les différentes permutations de n objets.

Une telle permutation est une bijection T de I = {1, 2,....., n } sur
O = { objet 1 , objet 2, ..., objet n } .

Dans la suite, on suppose que les n objets sont les premiers naturels non nuls (1, 2, ... , n) disposés dans un tableau T à n éléments. Un état du tableau T correspond à une permutation qu'on notera (T(1) , T(2) , ... T(n))·
On propose l'algorithme suivant qui, à partir d'un état du tableau T (une permutation initiale), donne un autre état (une nouvelle permutation différente de l'initiale).
Etape 1 : chercher le plus grand i tel que T(i -1) < T(i) .
Etape 2 : chercher le plus grand j tel que T(i -1) < T(j). (i trouvé à l'étape 1)
Etape 3 : permuter les contenus de T(i -1) et T(j).
Etape 4 : renverser les contenus de la séquence T(i) , T(i+1), ... , T(n) en permutant les contenus de T(i) et T(n) , T(i+ 1) et T(n-1) , et ainsi de suite.
On admet qu'en partant de la permutation (1, 2, ... , n) correspondante à l'identité c'est-à-dire T(1) = 1 , T(2) = 2 , ... , T(n) = n et en répétant l'application de cet algorithme à chaque permutation obtenue, on trouvera les différentes permutations de ces n entiers. On notera que la dernière permutation obtenue par ce procédé est (n, n-1 , ... , 3, 2, 1).

QUESTIONS :

A) On suppose que n = 4
1) Prouver qu'en appliquant une fois cet algorithme à la permutation (1, 2, 3, 4), on obtient la permutation (1, 2, 4, 3).'
2) En appliquant une fois cet algorithme, quand c'est possible, en quoi seront transformées les permutations : (1, 4, 3, 2) et (4, 3, 2, 1) ?

B)
Dans la suite, n est supposé un entier naturel quelconque non nul .
1) Ecrire un algorithme de la fonction PGI qui donne le plus grand i tel que T(i -1) < T(i).

Cette fonction devra renvoyer 0 quand aucun i ne vérifie la condition"
T(i -1) < T(i) " .

2) Ecrire un algorithme de la fonction PGJ(r) qui donne le plus grand j tel que T(r-1) < T(j).
On suppose que r vérifie impérativement T(r-1) < T(r).

3) Ecrire un algorithme de la procédure PERMUT (k, L) qui permute les contenus de T(k) et T(L) .

4) Ecrire un algorithme de la procédure RENVERSE (k) qui renverse les contenus de la séquence T(k), T(k+1) , ... , T(n).

5) Utiliser les fonctions et les procédures définies précédemment pour élaborer un algorithme du programme principal dont le rôle est d'afficher sur l'écran toutes les permutations des entiers 1 , 2, ... , n .

C) Traduire cette solution en un programme Pascal.

(Sujet de baccalauréat 1992)

Exercice 4

Une deuxième méthode pour trouver une solution au problème du voyageur de commerce est la suivante : nous supposons que l'itinéraire comporte les villes 1 et 2. Nous cherchons à insérer la ville 3 dans ce chemin de manière à minimiser la longueur de l'itinéraire. Si plusieurs positions sont possibles, nous choisissons la dernière dans une exploration gauche-droite de la liste courante des villes. Nous procèdons ainsi successivement pour chaque ville P_i, i = 3, . . . , N.

1) Décrivez brièvement la mise en oeuvre de cette idée.
2) Précisez en particulier les structures de données utilisées.
3) Analysez ce problème et en déduire les algorithmes correspondants.
4) Traduisez la solution en Pascal.

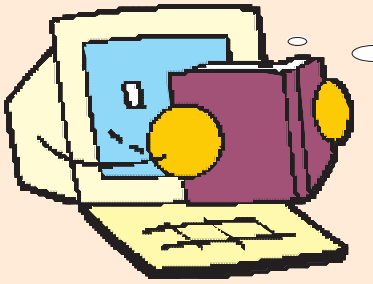
Exercice 5

Une troisième méthode pour trouver une solution au problème du voyageur de commerce est la suivante : sachant qu'un chemin entre les n villes $P_0, \ldots, P_{n-1}$ est un parcours $(P_{i0}, P_{i1}, \ldots, P_{in-1})$ du voyageur de commerce parmi ces villes qui passe par chaque ville une fois et une seule et qui revient à son point de départ. La longueur du chemin est la somme des distances (P_{ik}, P_{ik+1}) pour k = 0, . . . , n-1 avec $i_n = i_0$.

Un chemin entre les villes $P_0, \ldots, P_{n-1}$ est donc défini par une permutation de $(0, \ldots, n-1)$.

Autrement dit, pour trouver le chemin le plus court il suffit de calculer les distances de chaque permutation qui représentera un trajet et chercher la permutation qui donne le trajet le plus court. (voir exercice n°3).

1) Faites des recherches sur Internet pour trouver un algorithme de permutations de n objets
2) Analysez ce problème et en déduire les algorithmes correspondants.
3) Traduisez la solution en Pascal.



Lecture

Le problème du voyageur de commerce : Historique

➢ **19ème siècle**

Les premières approches mathématiques exposées pour le problème du voyageur de commerce ont été traitées au 19ème siècle par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman. Hamilton en a fait un jeu : Hamilton's Icosian game. Les joueurs devaient réaliser une tournée passant par 20 points en utilisant uniquement les connections prédéfinies.

➢ **1930**

Le PVC est traité plus en profondeur par Karl Menger à Harvard. Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merril Flood. Une attention particulière est portée sur les connections par Menger et Whitney ainsi que sur la croissance du PVC.

➢ **1954**

Solution du PVC pour 49 villes par Dantzig, Fulkerson et Johnson par la méthode du *cutting-plane*.

➢ **1975**

Solution pour 100 villes par Camerini, Fratta et Maffioli

➢ **1987**

Solution pour 532, puis 2392 villes par Padberg et Rinaldi

➢ **1998**

Solution pour les 13 509 villes des Etats-Unis.

➢ **2001**

Solution pour les 15 112 villes d'Allemagne par Applegate, Bixby, Chvàtal et Cook des universités de Rice et Princeton.

Bibliographie

Mathématiques
3ème année de l'enseignement secondaire Sciences de l'informatique
Centre National Pédagogique
Edition 2006

Mastering C
G-BOLON
BPB Publications 1988

Programmation Pascal avec TP
B-S- GOTTFRIED
Edition Série Schaum

Computer studies for you
S-DOYLE
Anchor Brendon 1987

A practical introduction to Pascal
I-R-WILSON and A-M-ADDYMAN
The Macmillan Computer science series 1982

A structured programming approach to data
D-Coleman
The Macmillan Computer science series 1982

Initiation à l'analyse et à la programmation
J-P-LAURENT
Dunod 1985

Aide mémoire de Turbo Pascal
J-M-De GROOTE et VIRGA
Marabout Informatique

Understanding computer science for advanced level
R-BRADLEY
Hutchinson Education 1987

L'informatique en Sup et en Spé
BRUNO PETAZZONI
Ellipses

Webographie

http://fr.wikipedia.org/

http://www.univ-paris12.fr/lacl/lacoste/Ada/Cours/Ada/Recursivite/html

http://publimath.irem.univ-mrs.fr/

http://lesfractales.nomades.ch/

http://www.supinfo-projects.com/

http://developpeur.journaldunet.com/tutoriel/theo/

http://www.cs.auckland.ac.nz/software/AlgAnim/

http://www.inf.fh-flensburg.de/

http://mathworld.wolfram.com/

http://physinfo.ulb.ac.be/

http://www.recreomath.qc.ca

http://www.mathgoodies.com

http://Petrequin.club.fr

http://J_Gourdin_www.ac-creteil.fr

http://f.nicolier.free.fr

http://www.pi314.net

http://www.techno-science.net