Cours	Algorithmique et structures de données
Auditoire	1ère année MPI
Etablissement	Institut National des Sciences Appliquées et de Technologies
Responsable du cours	Aymen SELLAOUTI
Années Universitaires	2014-2015 / 2015-2016

Objectifs :

L'objectif de ce module se décompose en trois parties :

- ✓ Apprendre à se familiariser avec les méthodes de résolution de problèmes tout en apprenant les principes de bases de l'algorithmique.
- ✓ Apprendre à analyser un problème et définir les structures de données et les opérateurs pour sa résolution.
- ✓ Appréhender le langage de programmation C.

Pré-requis : Aucun

Plan du Cours :

1. Introduction et concepts de base
- 1.1. Introduction
- 1.2. Définition d'un algorithme
- 1.3. Structure d'un algorithme
- 1.4. Les variables
- 1.5. Les types de données
- 1.6. Expressions et opérateurs
- 1.7. Lecture et écriture
- 1.8. Affectation
- 1.9. De l'algorithmique au C

2. Les structures conditionnelles
- 2.1. Introduction
- 2.2. Structure d'un test
- 2.3. Forme conditionnelle simple
- 2.4. Forme conditionnelle généralisée
- 2.5. Choix multiple
- 2.6. De l'algorithmique au C

3. Les structures itératives
- 3.1. Introduction
- 3.2. Itérations déterministes : la boucle for
- 3.3. Itérations indéterministes
  - 3.3.1. La boucle Tantque .. Jusqu'à
  - 3.3.2. La boucle Répéter
- 3.4. De l'algorithmique au C

4. Fonctions et Procédures
- 4.1. Introduction

Bibliographie :

[1] J.M. Rigaud et A. Sayah, Programmation en langage C, Ed. Cépadues, 1998.

[2] T. Cormen, Algorithmes Notions de base, Ed DUNOD, 2004

[3] J.P. Braquelaire, S'initier à la programmation - Avec des exemples en C, C++, C#, Java et PHP, Ed. Eyrolles, 2008.

[4] C. Delannoy, Programmer en langage C, 5ème édition, Ed. Eyrolles, 2009.

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structures de Données 1
Niveau MPI

Année universitaire
2015-2016

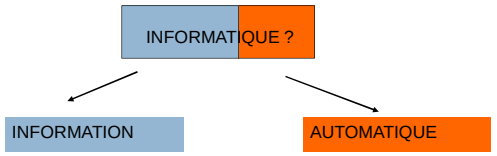
Chapitre 1
Introduction et concepts de bases

Wided Miled
Aymen Sellaouti

Chapitre 1
Concepts de base

Plan

Informatique



L'informatique (Information + Automatique) est la science du traitement automatique de l'information par une machine capable de traiter ou de manipuler les informations ou les données sous forme numérique ou binaire

Ordinateur

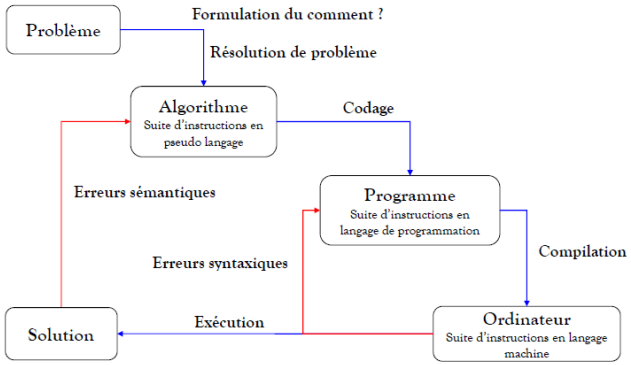
5

Machine qui saisit (périphériques d'entrée), stocke (mémoire), traite (programmes) et restitue (périphériques de sortie) des informations.

5

Algorithme et programme

6



Algorithme

7

Un algorithme est une suite d'instructions ordonnée, qui une fois exécutées correctement, conduit a un résultat donné.

Caractéristiques:

- Clarté : Pas d'ambiguïtés, compréhensible
- Déterminisme : Avec un ensemble de données donné, il faut qu'il fournisse le même résultat quelle que soit la machine.
- Fini : il doit se terminer quelle que soit la machine, le temps et la date d'exécution.
- Efficacité : l'algorithme doit effectuer le travail demandé avec le minimum de ressources.

Structure d'un algorithme

8

Entête	**Algorithme** Nom_algorithme
Déclarations	**Const** : Liste des constantes, **Var** : Liste des variables **Struct** : Liste des structures
Corps	**début** Action 1; Action 2; { Commentaires } … Action n; **fin**

Structure d'un algorithme

9

Exemple

- Algorithme de la somme de deux entiers

Nom : Somme_Deux_Entiers
Rôle : Additionner deux entiers
Entrée : a,b des entiers
Sortie : c un entier
Variables internes :aucun

Début
c ← a+b
Fin

Les éléments de base

10

1. Les données
   - Données numériques
   - Données alphanumériques
   - Données logiques
2. Les expressions
   - Expressions arithmétiques (operateurs arithmétiques)
   - Expressions logiques (operateurs relationnels et logiques)
3. Les actions
   - Déclaration
   - Affectation
   - Lecture/Ecriture

La notion de variable

11

- Dans un programme informatique, on a, en permanence, besoin de stocker provisoirement des valeurs :
  - données;
  - résultats obtenus par le programme
- Pour stocker une valeur au cours d'un programme, on utilise une variable.
- Une variable est une entité qui contient une information, elle est caractérisée par:
  - un nom, on parle d'identifiant (unique)
  - une valeur : information associée à une variable à un instant donné
  - un type, qui caractérise l'ensemble des valeurs que peut prendre la variable

La notion de variable

12

- Exemple : verre d'eau

Le contenu change
Mais le contenant
Reste le même

- Correspondance
  - Verre → Variable
    - Verre : objet contenant un liquide
    - Variable : zone mémoire contenant une valeur
  - Eau → Valeur

Déclaration d'une variable

13

- Le type d'une variable caractérise
  - L'espace des valeurs que peut prendre une variable donnée
  - L'ensemble des actions que l'on peut effectuer sur une variable
  - Apparaît dans l'entête de l'algorithme avec la déclaration des variables
- Déclaration

  Identifiant de la variable : son type
- Exemple

  Nom : chaîne de caractères

  Age : nombre naturel

  Distance : réel

Type de données

14

- Le type d'une variable est inchangeable. Il est déclaré une seule fois et reste le même.
- Le contenu de la variable doit être du même type. On affecte pas un réel à une variable de type entier

Deux grandes catégories de types:

- Simples: booléen, entier, réel, …
- Complexes: des structures complexes (que nous verrons dans la suite du cours)

Les types simples

15

Les types de variables les plus courants en algorithmique

- Type numérique
  - Entier : ensemble des **entiers relatifs Z**
  - Réel : ensemble des **nombres réels R**
- Type alphanumérique
  - Chaîne de caractère : toujours notée entre guillemets
- Type booléen
  - Booléen : stocke uniquement les valeurs logiques **VRAI** et **FAUX**

Expressions et opérateurs

16

- Une expression
  - est une combinaison d'opérateur(s) et d'opérande(s)
  - est évaluée durant l'exécution de l'algorithme
  - possède une valeur (son interprétation) et un type
- Un opérateur est un symbole d'opération qui permet d'agir sur des variable pour produire un résultat.
- Une opérande est une entité (variable, constante ou expression) utilisée par un opérateur.
- **Exemple :** a+b
  - a et b sont les opérandes
  - + est l'opérateur
  - a+b est appelée une expression

Expressions et opérateurs

17

- Un opérateur peut être Unaire ou Binaire :
  - **Unaire** s'il n'admet qu'une seule opérande, par exemple l'opérateur non
  - **Binaire** s'il admet deux opérandes, par exemple l'opérateur +
- Un opérateur est associé à un type de donnée et ne peut être utilisé qu'avec des variables, des constantes ou des expressions de ce type.
- Par exemple l'opérateur + ne peut être utilisé qu'avec les types arithmétiques (naturel, entier et réel) ou le type chaîne de caractères
  - **On ne peut pas additionner un entier et un caractère**

Expressions et opérateurs

18

- **Opérateurs Booléens :** Non, Et, Ou, OuExclusif

- Non

a	non a
Vrai	Faux
Faux	Vrai

- Ou

a	b	a ou b
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

- Et

a	b	a et b
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

- Ou exclusif

a	b	a ouExclusif b
Vrai	Vrai	Faux
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Expressions et opérateurs

19

- **Opérateurs sur les numériques**
  - On retrouve tout naturellement : **+, -, *, /, ^**
  - Pour les entiers : **div** et **mod**, permettent respectivement de calculer une division entière et le reste de cette division
  - L'opérateur d'**égalité** permet de savoir si les deux opérandes sont égales. Le résultat d'une expression contenant cet opérateur est un booléen.
  - On a aussi l'opérateur **d'inégalité : ≠**
  - Et pour les types possédant un ordre les opérateurs de comparaison <, ≤, >, ≥

Expressions et opérateurs

20

- **Opérateurs sur les numériques**
  - Pour les opérateurs arithmétique, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :

    ^ : (élévation à la puissance)
    * , / (multiplication, division)
    % (modulo)
    + , - : (addition, soustraction)

    exemple: **2 + 3 * 7 vaut 23**
  - En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

    exemple: **(2 + 3) * 7 vaut 35**

Expressions et opérateurs

21

- **Opérateurs alphanumérique**
  - & : la concaténation
  - Cet opérateur permet de **concaténer** deux **chaînes de caractères**

Exemple :

A ← " Bonjour"
B ← " tout le monde"
C ← A & B , C vaut " Bonjour tout le monde"

La lecture et l'écriture

22

- **L'instruction de lecture**
  - Une **instruction de lecture** permet à l'utilisateur de **rentrer des valeurs** au **clavier** pour qu'elles soient utilisées par le **programme**

Lire (A)

Dès que le programme rencontre une instruction Lire, l'exécution s'interrompt et attend la frappe d'une valeur au clavier.

- **L'instruction d'écriture**
  - Une **instruction d'écriture** permet au programme de **communiquer des valeurs** à l'utilisateur en les **affichant** à l'**écran.**

Ecrire ("La valeur de B est :" , B)

L'instruction d'affectation

23

Qu'est ce qu'on peut faire avec une variable ?

Réponse

La seule chose qu'on peut faire avec une variable, c'est l'affecter, c'est-à-dire lui attribuer une valeur

Convention

- En pseudo-code, l'instruction d'affectation se note avec le signe ←

L'instruction d'affectation

24

- On peut affecter à une variable la valeur d'une autre variable.

Exemple

Toto ← 24,5
Tutu ← Toto

- La valeur de Tutu est maintenant celle de Toto. Tutu contient donc la valeur 24,5

- On peut affecter à une variable le résultat d'une opération en fonction d'autres variables.

Exemple

Toto ← 24,5
Tutu ← Toto + 5,5

Important : Une instruction d'affectation ne modifie que ce qui est situé à gauche de l'affectation ←

L'instruction d'affectation

25

Une instruction d'affection doit respecter trois conditions :

- ✓ à gauche de l'affectation, on doit trouver **un nom de variable**, et uniquement cela. Dans le cas contraire, **il s'agit certainement d'une erreur !**
- ✓ à droite de l'affectation, on doit trouver une **expression** ;
- ✓ l'expression (située a droite de l'affectation) doit être **du même type** que la variable (située a gauche de l'affectation).

Définition d'une expression
Une expression est un **ensemble de valeurs**, reliées par des **opérateurs**, et équivalent à une seule valeur.

Langage C
De l'algorithmique au C

26

Eléments de base

27

Le langage C permet de découper un programme en modules.
La fonction principale est la fonction main qui va être exécutée la première.

```
#include <stdio.h>  /* instructions d'inclusion */

main( )
{

   /* partie déclarative */

   /* partie instructions (actions) */
}
```

Déclaration de variable

28

syntaxe : **<type> <liste de variables>**

- Le C propose les types simples suivants :
  - int, long, short, float, double, char
- On peut donc suivre les règles de traduction suivantes :

Algo. ⇒ C

Algorithmique	C
Entier,Naturel	int, long, short
Réel	float, double
Caractère	char
Booléen	int (1=VRAI, 0=FAUX)
Chaîne de caractères	*Voir les tableaux et pointeurs*

Traduire les opérateurs

29

Algo. ⇒ C

On traduit les opérateurs en respectant les règles suivantes :

Algorithmique	C
$=, \neq$	==, !=
$<, \leq, >, \geq$	<, <=, >, >=
et, ou, non	&&, ||, !
+, -, *, /	+, -, *, /
div, mod	/, %

Attention

En C l'affectation est une opération (opérateur =)

Traduire les opérateurs

30

Opérateurs ++ et −−

Les opérateurs unaires ++ et −− sont des opérateurs particuliers qui peuvent avoir jusqu'à deux effets de bord :

- En dehors de toute affectation, elle incrémente l'opérande associée, par exemple
  - `i++` et `++i` sont équivalents à `i=i+1`
- Lorsqu'ils sont utilisés dans une affectation, tout dépend de la position de l'opérateur par rapport à l'opérande, par exemple :
  - `j=i++` est équivalent à `j=i ;i=i+1 ;`
  - `j=++i` est équivalent à `i=i+1 ;j=i ;`

Traduire les opérateurs

31

Instructions simples

Elles finissent toujours par un ';'

Exemple

```
a=a+1;
```

Instructions composées

Les instructions composées qui permettent de considérer une succession d'instructions comme étant une seule instruction.
Elles commencent par "{" et finissent par "}"

Exemple

```
{a=a+1;b=b+2;}
```

Traduire lire

32

scanf

- L'instruction `scanf` (du module `stdio.h`) permet à l'utilisateur de saisir des informations au clavier
- Syntaxe :

  scanf("chaîne de formatage",pointeur var1, ...)

  - La chaîne de formatage spécifie le type des données attendues, par exemple :
    - %d pour les entiers (int, short, long)
    - %f pour les réels (float, double)
    - %s pour les chaînes de caractères
    - %c pour les caractères

Par exemple

```
int i;
float x;
scanf("%d%f",&i,&x);
```

Traduire Ecrire

33

printf

- L'instruction `printf` (du module `stdio.h`) permet d'afficher des informations à l'écran
- Syntaxe :

  printf ("chaîne de caractères" [, variables])

  - Si des variables suivent la chaîne de caractères, cette dernière doit spécifier comment présenter ces variables :
    - %d pour les entiers (int, short, long)
    - %f pour les réels (float, double)
    - %s pour les chaînes de caractères
    - %c pour les caractères
  - La chaîne de caractères peut contenir des caractères spéciaux :
    - \n pour le retour chariot
    - \t pour les tabulations

Traduire Ecrire

34

- Par exemple :

```
int i=1;
float x=2.0;
printf ("Bonjour\n");
printf ("i = %d\n",i);
printf ("i = %d, x = %f\n",i,x);
```

- . . . affiche :

  Bonjour
  i = 1
  i = 1, x=2.0

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 2
Les structures conditionnelles

Wided Miled
Aymen Sellaouti

Chapitre 2
Les structures conditionnelles

Plan

Introduction

- Les structures conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.
- Une condition est une expression écrite entre parenthèses à valeur booléenne.

Structure d'un test

- En Algorithmique, il y a deux formes possibles pour un test :
  - Forme conditionnelle simple
    - ✓ Forme simple réduite
    - ✓ Forme complète ou alternative
  - Forme conditionnelle généralisée

Forme conditionnelle simple

Forme Simple Réduite

[Instructions d'initialisation]

Si condition Alors

Instructions *à exécuter dans le cas où le résultat de l'évaluation de la condition est* ***VRAI.***

Fin Si

Remarque :

Dans le cas où l'évaluation de la condition est FAUX, les instructions ne seront pas exécutées.

Forme Alternative ou Complète

[Instructions d'initialisation]

Si condition Alors

instructions 1 *à exécuter si l'évaluation de la condition est* ***VRAI.***

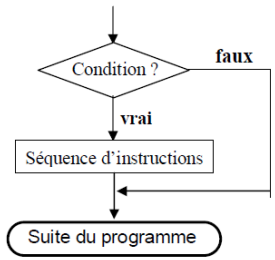
Sinon

instructions 2 *à exécuter si l'évaluation de la condition est* ***FAUX.***

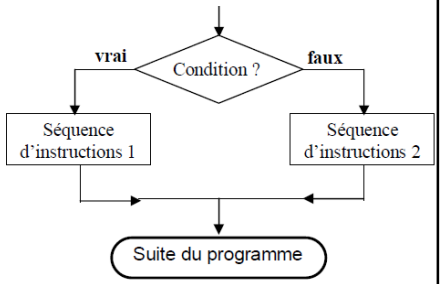
Fin Si

Forme conditionnelle simple

Forme Simple Réduite



Forme Alternative ou Complète



Qu'est ce qu'une condition?

- Une condition est une comparaison qui, à un moment donné, est vraie ou fausse.
- Les opérateurs de comparaison sont:
  - $=$: égal à ...
  - $\neq$: différent de ...
  - $<$: strictement plus petit que ..
  - $>$: strictement plus grand que .
  - $\leq$: plus petit ou égal à ...
  - $\geq$: plus grand ou égal à ...

Qu'est ce qu'une condition?

- la condition peut être une condition simple ou une condition composée de plusieurs conditions.
- Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques: ET, OU, OU Exclusif et NON

Exemples:

- x est compris entre 2 et 6: (x > 2) ET (x < 6)
- n est divisible par 3 ou par 2: (n%3) OU (n%2)

Exemple 1

- Ecrire un algorithme qui calcule et affiche la valeur absolue d'un entier quelconque lu au clavier.

```
Algorithme Valeur_Absolue1
Var : x, abs : entier
Début
  Ecrire('Entrer un entier')
  Lire(x)
   si (x > 0) alors
      abs ← x
   sinon
      abs← -x
   Finsi
   Ecrire('la valeur absolue de ',x, 'est',abs)
Fin
```

Exemple 1

- Ecrire un algorithme qui calcule et affiche la valeur absolue d'un entier quelconque lu au clavier.

```
Algorithme Valeur_Absolue2
Var : x, abs : entier
Début
  Ecrire('Entrer un entier')
  Lire(x)
   abs ← x
   si (x < 0) alors
      abs← -x
   Finsi
   Ecrire('la valeur absolue de ',x, 'est',abs)
Fin
```

Exemple 2

- Ecrire un algorithme qui demande un nombre entier à l'utilisateur puis teste et affiche s'il est divisible par 3

```
Algorithme Divisible_par3
Var : n : entier
Début
  Ecrire('Entrer un entier')
  Lire(n)
     si (n % 3) alors
       Ecrire(n, 'est divisible par 3')
     sinon
       Ecrire(n, 'n'est divisible par 3')
     Finsi
Fin
```

Exemple 3

- Ecrire un algorithme qui calcule le maximum de deux entiers x et y

```
Algorithme Maximum1
Var :  x,y,maxi : entier
Début
   Ecrire('Entrer deux entiers x et y')
   Lire(x,y)
     si (x  < y) alors
            maxi ← y
     sinon
            maxi ← x
     Finsi
    Ecrire('Le maximum de ',x, 'et' ,y, 'est', maxi)
  Fin
```

Exemple 3

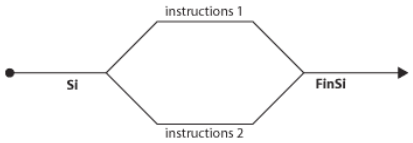
- Ecrire un algorithme qui calcule le maximum de deux entiers x et y

```
Algorithme Maximum2
Var :  x,y,maxi : entier
Début
   Ecrire('Entrer deux entiers x et y')
   Lire(x,y)
   maxi ←x
   si (x < y) alors
        maxi ← y
   Finsi
   Ecrire('Le maximum de',x, 'et' ,y, 'est', maxi)
Fin
```

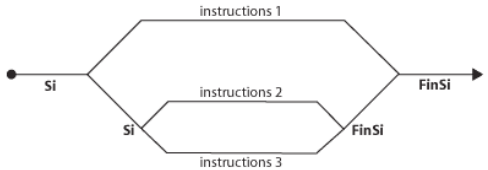
Forme conditionnelle généralisée

15

- Une condition permet de choisir entre deux actions



- Dans plusieurs cas, le choix doit se faire entre plus de deux actions



Exemple 4

- Un algorithme qui donne l'état de l'eau selon sa température doit pouvoir fournir trois réponses: (1) solide, (2) liquide, (3) gazeux.

```
Algorithme Température_Eau_1
Var :  Temp: réel
Début
   Ecrire('Entrer la température de l'eau')
   Lire(Temp)
      si (Temp  <= 0) alors   Ecrire('C'est de la glace')
      Finsi
      si  (Temp > 0) ET (Temp < 100)  alors  Ecrire('C'est du liquide')
      Finsi
      si (Temp  >= 100)  alors   Ecrire('C'est de la vapeur')
      Finsi
Fin
```

Exemple 4

Algorithme Température_Eau_2

Var : Temp: réel

Début

Ecrire('Entrer la température de l'eau')

Lire(Temp)

si (Temp <= 0) **alors** **Ecrire**('C'est de la glace')

Sinon

si (Temp < 100) **alors** **Ecrire**('C'est du liquide')

Sinon **Ecrire**('C'est de la vapeur')

Finsi

Finsi

Fin

- Les structures de tests imbriqués sont un outil indispensable à la simplification et à l'optimisation des algorithmes.

Forme conditionnelle généralisée

18

[Instructions d'initialisation]

Si condition 1 **Alors**

instructions 1

Sinon Si condition 2 **Alors**

instructions 2

Sinon Si condition 3 **Alors**

instructions 3

Sinon

instructions

Fin Si

Fin Si

Fin Si

[Instructions d'initialisation]

Si condition 1 **Alors**

instructions 1

SinonSi condition 2 **Alors**

instructions 2

SinonSi condition 3 **Alors**

instructions 3

Sinon

instructions

Fin Si

Exemple 5

- Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on inclut le cas où le produit peut être nul sans le calculer)

Algorithme Signe_Produit

Var : x,y : réels

Début

Ecrire('Entrer deux réels x et y')

Lire(x,y)

Si (x = 0) OU (y = 0) **Alors**

Ecrire ("Le produit est nul")

SinonSi (x < 0 ET y < 0) OU (x > 0 ET y > 0) **Alors**

Ecrire ("Le produit est positif")

Sinon

Ecrire ("Le produit est négatif")

Finsi

Fin

Choix multiple

20

Choix multiple

Selon < *expression* > **faire**

valeur1*: première séquence d'instructions;*

valeur2*: deuxième séquence d'instructions;*

...

valeurN*: Nème séquence d'instructions;*

Sinon (*N+1) ème séquence d'instructions;*

FinSelon

Exemple 6

moisA30Jours

```
Algorithme moisA30Jours
Var :  mois : Entier, résultat : Booléen
début
    selon mois faire
        4,6,9,11 : résultat ← Vrai
        sinon : résultat ← Faux
    finselon
fin
```


Langage C
De l'algorithmique au C

Traduire Si…Sinon…

Forme Simple Réduite

```
if (condition réalisée)
   {liste d'instructions;}
```

Forme Simple complète

```
if (condition réalisée)
   {liste d'instructions}
else
  {autre série d'instructions}
```

Forme généralisée

```
if ( condition1 )
        Instruction à exécuter
        si la condition1 est vraie.
else if ( condition2 )
        Instruction à exécuter
        si la condition2 est vraie.
else if ( condition3 )
        Instruction à exécuter
        si la condition3 est vraie.
...
else if (conditionN-1)
else
        Instruction à exécuter
        si les N-1 condtions sont toutes fausses
```

Traduire Selon


```
switch (Variable)
{
case Valeur 1:
        Liste d'instructions;
        break;
case Valeur 2:
        Liste d'instructions;
        break;
……
case Valeur n:
        Liste d'instructions;
        break;
default:
Liste d'instructions;
}
```

Pour exécuter les mêmes instructions pour différentes valeurs :

Exemple :

```
switch(variable)
{
case 1:
case 2:
     {instructions exécutées pour variable = 1
      ou pour variable = 2}
      break;
case 3:
    {instructions exécutées pour variable = 3}
    break;
default:
    {instructions exécutées pour toute
     autre valeur de variable} }
```

Exemple

Exemple

```
switch(choix) {
    case 't' : printf("vous voulez un triangle\n"); break;
    case 'c' : printf("vous voulez un carre\n"); break;
    case 'r' : printf("vous voulez un rectangle\n"); break;
    default : printf("erreur. recommencez !\n");
}
```

Exemple

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char reponse;

  printf("Que voulez-vous faire ?\n");
  printf("q pour quitter\n");
  printf("s pour sauvegarder\n");
  printf("o pour ouvrir un nouveau fichier\n");

  scanf("%c", &reponse);

  switch(reponse){
  case 'q': printf("au revoir\n"); break;
  case 's': printf("sauvegarde \n"); break;
  case 'o': printf("ouverture\n");break;
  default: printf("ce n'est pas un choix valable\n");
  }
}
```

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 3 Les structures itératives

Wided Miled
Aymen Sellaouti

Chapitre 3
Les structures itératives

Plan

Intoduction

- Lorsque l'on veut répéter plusieurs fois un même traitement, plutôt que de copier n fois la ou les instructions, on peut demander à l'ordinateur **d'exécuter n fois une suite d'instructions**
- Il existe deux grandes catégories d'itérations :
  - Les itérations **déterministes : le nombre de boucle est défini à l'entrée de** la boucle
  - Les itérations **indéterministes : l'exécution de la prochaine boucle est** conditionnée par une expression booléenne

Itérations déterministes

- Il existe une seule instruction permettant de faire des boucles déterministes, c'est l'instruction `pour`
- Sa syntaxe est :

pour *identifiant d'une variable de type scalaire* ←*valeur de début* **à** *valeur de fin* **faire**
    instructions à exécuter à chaque boucle
finpour

- dans ce cas la variable utilisée prend successivement les valeurs comprises entre *valeur de début* et *valeur de fin*

Exemple: la factorielle

Ecrire un algorithme qui calcule la factorielle d'un entier.

```
Algorithme Factorielle
Var
n, fact,i : entier
Début
        fact ← 1
        pour i ← 1 à n faire
                fact ← i * fact
        fin _pour
        Écrire ("La factorielle de n est ", fact)
Fin
```

Itérations indéterministes

- Il existe deux instructions permettant de faire des boucles indéterministes :
  - L'instruction **tant que** :

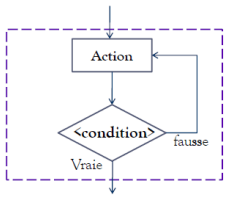
tant que *expression booléenne* **faire**
    instructions
fintantque

    - qui signifie que tant que l'expression booléenne est vraie on exécute les instructions
  - L'instruction **répéter jusqu'à ce que** :

repeter
    instructions
jusqu'a ce que *expression boolèenne*

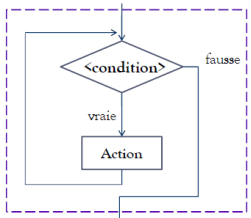
    - qui signifie que les instructions sont exécutées jusqu'à ce que l'expression booléenne soit vraie

Itérations indéterministes




```
répéter
        Action
jusqu'à (condition)
```

Les instructions sont répétées jusqu'à la condition soit vérifiée
Action exécutée **au moins une fois**




```
tant que (condition) faire
        Action
fin_tant_que
```

Tant que condition est vraie, les instructions sont répétées
Action exécutée 0 ou plus

Itérations indéterministes

Structure Répéter

Répéter
Instructions
Jusqu'à *Booléen*

- Arrivée à la première ligne **Répéter**, **quelque soit** la valeur du **booléen**, le programme rentre dans la **boucle**.
- La machine exécute **au moins une fois** la série d'instructions de la **boucle**.
- La machine effectue **au moins une itération** dans la boucle.

Structure Tant que

Tant que *Booléen* **Faire**
Instructions
Fin Tant que

- Arrivée à la première ligne **Tant que**, pour rentrer dans la boucle, la valeur du **booléen doit être VRAI**.
- La machine peut **ne jamais exécuter** la série d'instructions de la **boucle**.
- La machine peut **ne jamais effectuer** d'**itération** dans la **boucle**.

Boucle infinie !

Attention

- Si vous ne voulez pas que votre algorithme « tourne » indéfiniment, l'expression booléenne doit faire intervenir des variables dont le contenu doit être modifié par au moins une des instructions du corps de la boucle

Exemple de boucle infinie :

```
i ← 2
TantQue  (i > 0)
i ← i+1      (attention aux erreurs de frappe : + au lieu de -)
FinTantQue
```

Exemple: contrôle de saisie

Écrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

```
Algorithme Saisie_Nombre_1
  Var : N : entier
  Début
    Répéter
      Ecrire ("Entrez un nombre entre 1 et 3")
      Lire (N)
    Jusqu'à (N >= 1 ET N <= 3)
  Fin
```

Exemple: contrôle de saisie

Écrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : "**Entrez un nombre plus petit !**", et inversement, "**Entrez un nombre plus grand !**" si le nombre est inférieur à 10.

Exemple: contrôle de saisie

```
Algorithme Saisie_Nombre_2
 Var : N : entier
 Début
   Répéter
      Ecrire ("Entrez un nombre entre 10 et 20")
      Lire (N)
      Si N < 10 Alors
         Ecrire ("Entrez un nombre plus grand !")
      Sinon Si N > 20 Alors
         Ecrire ("Entrez un nombre plus petit !")
      Fin Si
   Jusqu'à (N >= 10 ET N <= 20)
 Fin
```

Exemple: somme

Somme des *n* premiers naturels

Nom: somme
Role: Calculer la somme des n premiers entiers positifs, s=0+1+2+...+n
Entrée: n : **Naturel**
Sortie: s : **Naturel**
Entrée/Sortie:
Déclaration: i : **Naturel**

```
debut
    s ← 0
    pour i ←0 à n faire
        s ← s+i
    finpour
fin
```

Exemple: le plus grand entier

invfact

Nom: invFact
Role: Détermine le plus grand entier e telque e ! ≤ n
Entrée: n : **Naturel**≥ 1
Sortie: e : **Naturel**
Entrée/Sortie:
Déclaration: fact : **Naturel**

```
debut
    fact ← 1
    e ← 1
    tant que fact ≤ n faire
        e ← e+1
        fact ← fact*e
    fintantque
    e ← e-1
fin
```

Exemple: le plus grand entier

Explication avec n=10

Nom: invFact
Role: Détermine le plus grand entier e telque e ! ≤ n
Entrée: n : **Naturel**≥ 1
Sortie: e : **Naturel**
Entrée/Sortie:
Déclaration: fact : **Naturel**

```
debut
    fact ← 1
    e ← 1
    tant que fact ≤ n faire
        e ← e+1
        fact ← fact*e
    fintantque
    e ← e-1
fin
```

	n	e	fact	fact ≤ n
fact ←1	10	?	1	vrai
e ←1	10	1	1	vrai
e ←e+1 fact ←fact*e	10	2	2	vrai
e ←e+1 fact ←fact*e	10	3	6	vrai
e ←e+1 fact ←fact*e	10	4	24	faux
e ←e-1	10	3	24	faux

17

Langage C
De l'algorithmique au C

Traduire les itérations

for

L'instruction `Pour` est traduite par l'instruction `for`, qui a la syntaxe suivante :

```
for( initialisation ;
     condition_d_arret ;
     operation_effectuée_à_chaque_itération )
  instruction ;
```

Exemple

```
for(i=0;i<10;i++)
   printf ("%d\n", i);
```

Traduire les itérations

while

L'instruction `Tant...que` est traduite par l'instruction `while`, qui a la syntaxe suivante :

```
while(condition)
      instruction ;
```

Exemple

```
i=0;
while(i<10){
   printf ("%d\n", i);
   i++;
}
```

Traduire les itérations

do..while

L'instruction `répéter` est traduite par l'instruction `do..while`, qui a la syntaxe suivante :

```
do
      instruction ;
while( condition );
```

Exemple

```
i=0;
do {
   printf ("%d\n", i);
   i++;
} while (i<=10);
```

Instructions while, do/while et for

```c
#define N 10

int i;

i = 1;
while( i <= N ) {
 printf("%d",i);
 i++;
}
```

```c
#define N 10

int i;

i = 1;
do {
 printf("%d",i);
 i++;
} while ( i<=N );
```

```c
#define N 10

int i;

for(i=1;i<=N;i++)
 printf("%d",i);
```

Exemple: racine carré

```
algorithme   RACINE_CARREE
  Var  N : réel
  Début
    répéter
        écrire("Entrer un nombre (>=0) : " )
        lire( N )
    jusqu'à  (N >= 0)
    écrire( "La racine carrée de ",N ,"vaut ",  √N )
  Fin algo
```

Exemple: racine carré

```c
#include <stdio.h>
#include <math.h>
main()
{
 float N;
 do
   {
    printf("Entrer un nombre (>= 0) : ");
    scanf("%f", &N);
   }
 while (N < 0);
 printf("La racine carrée de %.2f est %.2f\n", N, sqrt(N));
```

Attention !!
La condition d'arrêt de l'instruction *do/while* en C est la négation de la condition d'arrêt de l'instruction algorithmique *Répéter …jusqu'à*

Exercice 1

Exercice :
Ecrire un algorithme et sa traduction en C d'un programme qui calcule le PGCD de 2 entiers a et b.

PGCD(a,b) = PGCD(a-b,b) si a > b
= PGCD(a,a-b) si a< b

```c
main()
{
    int a,b ;
    scanf("%d %d",&a,&b) ;
            while(a !=b)
            {
            if (a > b)  a=a-b;
            else b=b-a;
            }
   Printf("%d",a);
}
```

Exercice 2

Exercice :

Ecrire un algorithme puis un programme C qui compte la fréquence des voyelles A, a, E, e, I, i, O, o, U et u dans une saisie d'un texte qui se termine par le caractère '#' (lecture caractère par caractère), puis affiche le résultat sous la forme suivante :

A, a : 5
E, e : 8
I, i : 0
O, o : 3
U, u : 10

Les voyelles minuscules et majuscules sont comptées ensemble.

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 4
Fonction et procédures

Wided Miled
Aymen Sellaouti

Plan

Chapitre 4 -1
Fonctions et procédures

Sous-programmes - Décomposer pour résoudre

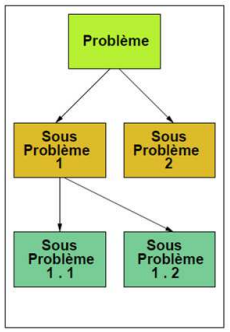
- Analyse descendante

  Approche incrémentale consistant à analyser la spécification d'un problème selon son arborescence

- Décomposition / combinaison

  à chaque niveau de l'arborescence :

  - Décomposer le problème en un certain nombre de sous-problèmes qu'on supposera résolus
  - Combiner les solutions des sous-problèmes pour résoudre le problème courant
  - Résoudre les sous-problèmes par la même approche



Sous-programmes - Décomposer pour résoudre

- Par exemple, pour résoudre le problème suivant :

  Ecrire un programme qui affiche les nombres parfaits compris entre 0 et une valeur n saisie par l'utilisateur.

  ex: 28=1+2+4+7+14.

Cela revient à résoudre :

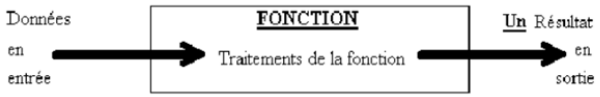
1. Demander a l'utilisateur de saisir un entier n
2. Afficher les nombres parfaits compris 0 et n
3. Savoir si un nombre donné est parfait
   - 3. 1 Calculer la somme des diviseurs d'un nombre
   - 3. 2 Savoir si un nombre est diviseur d'un autre nombre

Sous-programmes - Décomposer pour résoudre

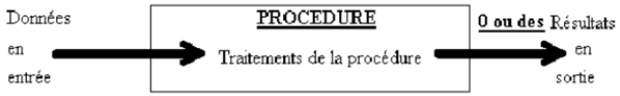
- Chacun de ces sous-problèmes devient un nouveau problème à résoudre.
- Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait "quasiment" résoudre le problème initial.
- **Donc écrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial.**
- En algorithmique il existe deux types de sous-programmes :
  - **Les fonctions**
  - **Les procédures**
- Un sous-programme est obligatoirement caractérisé par un nom (identifiant) unique
- Le programme qui utilise un sous-programme est appelé **le programme appelant.**

Sous-programmes - Décomposer pour résoudre

- **Une fonction rend un et un seul résultat.**



- **Une procédure rend de zéro à plusieurs résultats.**



Les fonctions – Déclaration d'une fonction

```
Fonction NomDeLaFonction (paramètres et leurs types) : type_fonction
Var variable locale 1 : type 1; . . .
  début
    instructions de la fonction
    Retourner expression
  FinFonction
```

Les fonctions – Déclaration d'une fonction

- **type_fonction** est le type du résultat renvoyé par la fonction
- **liste de paramètres** est la liste des paramètres formels donnés en entrée avec leurs types de la forme p_1: *type_1*, . . . , p_n: *type_n*,
- Le corps de la fonction doit comporter une instruction de la forme :

  retourner(*expression*);

  cette instruction fin à l'exécution de la fonction et retourne *expression* comme résultat
- Pour le choix d'un nom de fonction, il faut respecter les mêmes règles que celles pour les noms de variables.

Les fonctions – Exemple 1

10

Écrire une fonction qui renvoie la moyenne de deux entier.

```
Fonction Moyenne(A : entier, B : entier) : réel
Var : Moy : réel
  Moy ← (A + B) / 2
  Renvoyer  Moy
Fin
```

Les fonctions – Exemple 2

11

Écrire une fonction qui renvoie la factorielle d'un entier.

```
Fonction Factorielle(N : entier) : entier
Var : Fact, i : entier
  Fact ← 1
  Pour i ← 2 à N Faire
    Fact ← Fact * i
  Fin Pour
  Renvoyer  Fact
Fin
```

Les fonctions – Exemple 3

12

Écrire une fonction qui renvoie le maximum de deux réels.

```
Fonction Maximum(A : réel, B : réel) : réel
  Si (A ≥ B) Alors
    Renvoyer  A
  Sinon
    Renvoyer  B
  Fin Si
Fin
```

Les fonctions – Appel d'une fonction

13

- L'**appel** d'une fonction se fait par simple écriture de son nom dans le programme principale suivi des paramètres entre parenthèses (les parenthèses sont toujours présentes même lorsqu'il n'ya pas de paramètre).

  NomDeLaFonction(<liste de paramètres>);

- **liste de paramètres** est la liste des **paramètres effectifs** de la forme q_1,. ., q_n, avec q_*i* un paramètre de type *type p_i*
- L'appel de la fonction retourne une valeur calculée en fonction des paramètres effectifs
- La liste des paramètres effectifs doit être **compatible** avec la liste des paramètres formels de la déclaration de la fonction
- Lors de l'appel, chacun des paramètres formels **est remplacé** par le paramètre effectif

Les fonctions – Appel d'une fonction

- L'appel d'une fonction se fait par simple écriture de son nom dans le programme principale. Le résultat étant une valeur, devra être affecté ou être utilisé dans une expression

Exemple:

```
Fonction Pair (n : entier ) : booléen
    Debut
      retourne (n%2=0)
    FinFonction

Algorithme AppelFonction
  Var:  c : réel, b : booléen
  Début
    b ← Pair(3)
    …..
  Fin
```

- Lors de l'appel Pair(3) le paramètre formel n est remplacé par le paramètre effectif 3

14

Les fonctions – Appel d'une fonction

15

```
fonction minimum2 (a,b : Entier) : Entier
debut
   si a ≥ b alors
      retourner b
   sinon
      retourner a
   finsi
fin
fonction minimum3 (a,b,c : Entier) : Entier
debut
   retourner minimum2(a,minimum2(b,c))
fin
```

Les procédures – Déclaration

16

- Une procédure est un sous-programme semblable à une fonction mais qui ne retourne rien.
- Une procédure s'écrit en dehors du programme principal sous la forme :

  Procédure nom_procédure (paramètres et leurs types)

  Instructions constituant le corps de la procédure

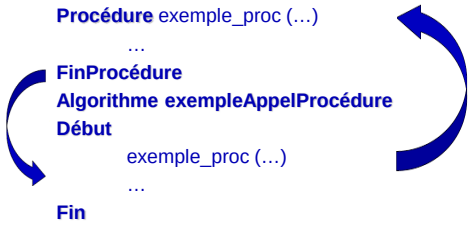
  FinProcédure

- Remarque : une procédure peut ne pas avoir de paramètres.

Les procédures – Appel

17

- L'appel d'une procédure, se fait dans le programme principale ou dans une autre procédure par une instruction indiquant le nom de la procédure :



- Remarque : contrairement à l'appel d'une fonction, **on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression**. L'appel d'une procédure est une instruction autonome.

Portée d'une variable

- La **portée d'une variable**: **l'ensemble des sous-programmes où cette variable** est connue (les instructions de ces sous-programmes peuvent utiliser cette variable)

- Une variable définie au niveau du programme principal (celui qui résout le problème initial) est appelée **variable globale**

- Sa portée est totale : **tout sous-programme du programme principal peut utiliser cette variable**

- Une variable définie au sein d'un sous programme est appelée **variable locale**

- La portée d'une variable locale est uniquement le sous-programme qui la déclare. Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée : Dans ce sous-programme la variable globale devient **inaccessible**

18

Paramètres formels/effectifs

19

- Les paramètres servent à échanger des données entre le programme principale (ou le sous-programme appelant) et le sous-programme appelé.

- Les paramètres placés dans la déclaration d'un sous-programme sont appelés paramètres formels. Ces paramètres peuvent prendre toutes les valeurs possibles mais ils sont abstraits (n'existent pas réellement)

- Les paramètres placés dans l'appel d'un sous-programme sont appelés paramètres effectifs. ils contiennent les valeurs pour effectuer le traitement

- Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels. L'ordre et le type des paramètres doivent correspondre.

Transmission de paramètres

- Il existe deux modes de transmission de paramètres dans les langages de programmation :

La transmission par valeur : les valeurs des paramètres effectifs sont affectées aux paramètres formels correspondants au moment de l'appel de la procédure. Dans ce mode le paramètre effectif ne subit aucune modification

La transmission par adresse (ou par référence) : les adresses des paramètres effectifs sont transmises à la procédure appelante. Dans ce mode, le paramètre effectif subit les mêmes modifications que le paramètre formel lors de l'exécution de la procédure

Remarque : le paramètre effectif doit être une variable (et non une valeur) lorsqu'il s'agit d'une transmission par adresse

20

Passage d'arguments dans une fonction

Passage des paramètres par valeur (en Entrée)

- C'est le cas standard
- Les paramètres formels sont initialisés par une copie des valeurs des paramètres effectifs
- Modifier la valeur des paramètres formels dans le corps de la fonction ne change pas la valeur des paramètres effectifs

Passage d'arguments dans une fonction

```
Programme exemple1
    Déclaration a : Entier, b : Naturel
    fonction abs (unEntier : Entier) : Naturel
        Déclaration valeurAbsolue : Naturel
    début
        si unEntier ≥ 0 alors
            valeurAbsolue ← unEntier
        sinon
            valeurAbsolue ← -unEntier
        fi nsi
        retourner valeurAbsolue
    fi n
début
    écrire("Entrez un entier :")
    lire(a)
    b ← abs(a)
    écrire("la valeur absolue de ",a," est ",b)
fi n
```

Lors de l'exécution de la fonction *abs*, la variable *a* et le paramètre *unEntier* sont associés par un passage de paramètre en entrée : La valeur de *a* est copiée dans *unEntier*

Passage d'arguments dans une procédure

- Les procédures sont des sous-programmes qui ne retournent **aucun** résultat.
- Par contre elle admettent des paramètres avec des passages:
  - En entrée, préfixés par **Entrée** (ou **E**)
  - En sortie préfixés par **Sortie** (ou **S**)
  - En entrée/sortie, préfixés par **Entrée/Sortie** (ou **E/S**)
- Les paramètres en Entrée sont considérés aussi comme paramètres passés par valeur.
- Les paramètre en sortie ou en Entrée/ sortie sont considéré aussi comme paramètres passés par adresse ou par référence.

Déclaration d'une procédure

- On déclare une procédure de la façon suivante :

Procédure *NomDeLaProcédure* **(** **E** *paramètre(s) en entrée;* **S** *paramètre(s) en sortie;* **E/S** *paramètre(s) en entrée/sortie* **)**
Var *variable(s) locale(s)*
début
instructions de la procédure
FinProcédure

- Et on appelle une procédure comme une fonction, en indiquant son nom suivi des paramètres entre parenthèses

Procédures – Exemple 1

25

```
Algorithme exemple
Var  entier1,entier2,entier3,min,max : Entier
Fonction minimum2 (a,b : Entier) : Entier
. . .
Fonction minimum3 (a,b,c : Entier) : Entier
. . .
Procédure calculerMinMax3 ( E a,b,c : Entier ; S min3,max3 : Entier )
Début
        min3← minimum3(a,b,c)
        max3 ←maximum3(a,b,c)
Fin
début
    écrire("Entrez trois entiers :")
    lire(entier1) ;
    lire(entier2) ;
    lire(entier3)
     calculerMinMax3(entier1,entier2,entier3,min,max)
     écrire("la valeur la plus petite est ",min," et la plus grande est ",max)
Fin
```

Procédures – Exemple 2

26

```
Programme exemple2
    Déclaration a : Entier, b : Naturel
    procédure echanger ( E/S val1 Entier; E/S val2 Entier;)
        Déclaration temp : Entier
    début
        temp ← val1
        val1 ← val2
        val2 ← temp
    fin
début
    écrire("Entrez deux entiers :")
    lire(a,b)
    echanger(a,b)
    écrire("a=",a," et b = ",b)
fin
```

Lors de l'exécution de la procédure *echanger*, la variable *a* et le paramètre *val1* sont associés par un passage de paramètre en entrée/sortie : Toute modification sur val1 est effectuée sur a (de même pour b et val2)

Conclusion

- Lorsqu'une séquence d'instructions **se répète plusieurs fois**, il est intéressant de faire un sous-programme correspondant à ce bloc d'instructions et de l'utiliser autant de fois que nécessaire
- Cette séquence d'instructions sera définie dans un sous-programme qui peut prendre la forme d'une procédure ou d'une fonction
- De plus, un programme est presque toujours décomposable en modules qui peuvent alors être définis de manière indépendante.
- Un programme est alors un ensemble de **procédures / fonctions.**
- Une fonction est un sous-programme qui prend **zéro ou plusieurs paramètres** et retourne un seul résultat calculé en fonction des valeurs passées en entrée
- Une fonction qui ne retourne pas de résultat est une procédure

27

28

Chapitre 4 -2
Fonctions et procédures en langage C

28

Structure d'un programme en C

29

✓ La décomposition d'un programme en sous programmes permet d'avoir des programmes plus lisibles, plus faciles à mettre à jour et plus simples à développer puisqu'on ne s'intéresse qu'à une seule tâche à la fois.

✓ Un programme en C est une collection de fonctions. L'une des fonctions doit s'appeler **main**. L'exécution d'un programme C correspond à l'exécution de la fonction **main**. Les autres fonctions sont exécutées si elles sont appelées dans la fonction **main**.

✓ Il n'y a pas de procédure en C. Pour traduire une procédure, on crée une fonction qui ne retourne pas de valeur, on utilise alors le type **void** comme type de retour.

Déclaration d'une fonction

❖ *typeRetour* **NomFonction (***type1* **arg1,** *type2* **arg2 …***typeN* **argN)**

```
// paramètres formels arg1, arg2, …,argN
{
   // variables locales
   instructions ;
   return val;
}
```

❖ . **L'entête de la fonction comprend :**

1. *typeRetour* : le type du résultat retourné, une fonction qui ne retourne rien est de type void.
2. Le nom de la fonction.
3. La liste de ses paramètres : arguments de la fonction. Cette liste peut être vide.

❖ **Le corps :** contient des déclarations de variables, des instructions C (conditions, boucles ...). L'instruction return permet de retourner le résultat de la fonction au programme qui l'a appelée

30

Déclaration d'une fonction

```
/* la fonction surface délivre un int,
   ses paramètres formels sont l'int largeur et l'int longueur */
int surface(int largeur, int longueur)
{
   int res;       /* déclaration des variables locales */
   res=largeur*longueur;
   return res;    /* retourne à l'appelant en délivrant res */
}
```

31

Appel des fonctions

Appel d'une fonction : Variable = Nom_fonction(arg1, arg2, …, argN);
Appel d'une procédure : Nom_procedure(arg1, arg2, …, argN);

```
#include <stdio.h>

int surface(int largeur, int longueur)
{
   int res;
   res=largeur*longueur;
   return res;
}

void main()
{
   int l,L,surf;
   printf("\n Entrer la largeur:"); scanf("%d", &l);
   printf("\n Entrer la longueur:"); scanf("%d", &L);
   surf=surface(l,L);   // appel de la fonction surface
   printf("\n Surface = %d\n",surf);
}
```

32

Appel des fonctions

Si la fonction est écrite **après** la fonction main, il faut donner son prototype (en-tête) avant cette dernière :

```
#include <stdio.h>
int surface(int largeur , int longueur);
void message();

void main()
{
   int l,L,surf;
   message(); scanf("%d", &l);
   message(); scanf("%d", &L);
   surf=surface(l,L);      /*appel avant déclaration de la fonction surface*/
   printf("\n Surface = %d\n",surf);
}

int surface(int largeur , int longueur)
{
   return largeur*longueur;
}
void message() { printf(" \n Saisir une valeur:");}
```

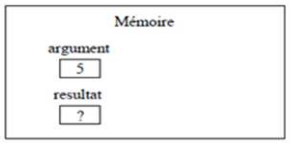
// Prototype

Appel des fonctions

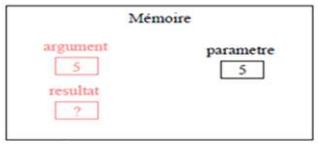
```
#include<stdio.h> /*inclusion d'un fichier permettant l'utilisation*/
                                  /* de fonctions déjà codées */


int AuCarre(int);          /* prototypage de notre fonction AuCarre */

int main()
{                                          /* l'exécution commence ici */
 int resultat;                  /* déclaration de variables locales */
 int valeur = 5;

 resultat = AuCarre(valeur);                        /*Appel de AuCarre*/
 printf("resultat = %d \n",resultat);    /*Appel fonction bibliothèque*/
}

int AuCarre(int parametre)              /* code de la fonction AuCarre*/
{
 return parametre*parametre;
}
```


Que se passe t-il en mémoire?

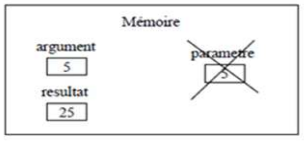
```
1    int AuCarre(int);
2
3    int main()
4    {
5      int resultat;
6      int argument = 5;
7
8      resultat = AuCarre(argument);
9    }
10
11   int AuCarre(int parametre)
12   {
13     return (parametre*parametre);
14   }
```



État de la mémoire à la ligne 7 de l'exécution



État de la mémoire à la ligne 12



État de la mémoire à la ligne 9

Fonctions – Factorielle d'un entier

Déclaration:

```
int fact(int n)
{
     int i, f ;
    for(i=1 , f=1 ; i<n ; i ++)
     f=f* i ;
    return (f );
}
```

Appel de cette fonction :

```
void main( )
{
  int x = Fact(4) ;
  int y = Fact(3) + Fact(2) ;
  printf("%d", Fact(5)) ;
}
```

Fonctions – Maximum de deux entiers

```
int max(int x, int y)
{
  int m ;
  if (x > y)
        m = x ;
  else
        m = y ;
  return (m) ;
}
```

```
int max(int x, int y)
{
  if (x > y)
      return (x) ;
  else
      return (y) ;
}
//sans aucune variable intermédiaire
```

```
void main( )
{
  int a ;
  a = Max(4, 5) ;
  printf("%d", a);                    // a = 5
  a = Max(a, 10) ;
  printf("%d", a);                    // a = 10
  a = Max(a, Max(a, 5)) ;
  printf("%d", a);                    // a = 10
}
```


Fonctions – Passage de paramètres

main()

int x = Fact(4) ;

x = 24

Appel →

← **Retour**

fact()

n = 4

Exécution ↓

f= 24

return(24)

Lors d'appel, il y a copie des valeurs des paramètres effectifs dans les paramètres formels. ➔ **Passage des paramètres par valeur.**

fact(4) ➔ Paramètre **effectif (appel de la fonction)**

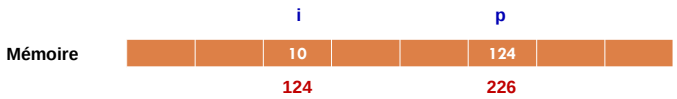
int fact(int n) ➔ Paramètre **formel (Déclaration de la fonction)**

Les pointeurs

Définition :

- ✓ Un pointeur est une variable qui permet de stocker une adresse
- ✓ Lorsque , l'on déclare une variable, par exemple un entier i, l'ordinateur réserve un espace mémoire pour y stocker la valeur de i
- ✓ L'emplacement de cet espace dans la mémoire est nommé adresse

Par exemple si on déclare une variable de type entier (initialisée a 10) et que l'on déclare un pointeur p dans lequel on range l'adresse de i (on dit que p pointe sur i) , on a le schéma suivant :



Les pointeurs

- On déclare une variable de type pointeur en préfixant le nom de la variable par le caractère * .

  int*p;

- Il existe deux opérateurs permettant d'utiliser les pointeurs:

  &: permet d'obtenir l'adresse d'une variable, permettant donc à un pointeur de pointer sur une variable

  *: permet d'accéder à la valeur de la variable pointée

Exemple 1:

On veut déclarer un pointeur p qui va contenir l'adresse d'un entier

int *p ; // cette déclaration signifie que p va contenir l'adresse d'un entier

int A = 20 ;

p = &A ; p contient l'adresse mémoire de la variable A.

***p** : le contenu de la case mémoire pointée par p (contenu de la variable A).

***p = A**

Les pointeurs

41

Exemple 2 :

```
int *p ;
int x = 5 ;
int y ;
p = &x ;
y = *p + 1 ;          // ➔ y = x + 1 ⇔ y = 6
*p = *p + 10 ;        // ➔ x = x + 10 ⇔ x = 15
*p +=2                // ➔ x = x + 2 ⇔ x = 17
(*p)++ ;        // ➔ x++ ⇔ x = 18
```

Si on a :

```
int *p ;
int A =&p ;
```

Alors p contient l'adresse mémoire de l'entier A ;
*p est un entier ➔ *p = A

Les pointeurs

42

Exemple 3 :

```
i n t i =3;
i n t *p ;
p = &i;
```

La taille d'un entier en octets :
- 2 (sur processeur 16 bits)
- 4 (sur processeur 32 bits)

Variable	Adresse	Valeur
i	4830000	3
p	4830004	4830000
*p	4830000	3

Les pointeurs

43

Exercice :

Considérons les variables suivantes :

```
int A = 1 ;
int B = 2 ;
int C = 3 ;
int *p1 ;
int *p2 ;
```

Le tableau suivant contient l'effet de chaque commande sur les variables.

Les pointeurs

44

	A	B	C	p1	p2
p1 = &A ;	1	2	3	&A	-
p2 = &C ;	1	2	3	&A	&C
*p1 **=** **(*****p2)++ ;	4	2	3	&A	&C
p1 = p2 ;	4	2	3	&C	&C
p2 = &B ;	4	2	3	&C	&B
*p1 -= *p2 ≅ (*p1=*p1-*p2);	4	2	1	&C	&B
*p2++ ;	4	3	1	&C	&B
*p1 *= *p2 ≅ (*p1=*p1 * *p2);	4	3	3	&C	&B
A = (*p2) ++ * (*p1)++ ;	16	3	3	&C	&B
p1 = &A ;	16	3	3	&A	&B

Arithmétique des pointeurs

45

- La valeur d'un pointeur est un entier.
- On peut appliquer à un pointeur quelques opérations arithmétiques :
  - Addition d'un entier a un pointeur.
  - Soustraction d'un entier a pointeur.
  - Différence entre deux pointeurs (de même type).

- Soit i un entier et p un pointeur sur un élément de type Type , L'expression p'=p+i (resp. p= p − i) désigne un pointeur sur un élément de type Type ,
la valeur de p est égale a la valeur de p incrémenté (resp. décrémenté) de i ∗ sizeof(Type).

Arithmétique des pointeurs

46

```
int
main(int arv,char * arg[]) {
    int i=5;
    int *p1,*p2;
    p1 = &i + 2;
    p2 = p1 − 2;
}
```

Si &i = 4830000 alors :

Variable	Adresse	Valeur
i	4830000	5
p1	4830004	4830008
p2	4830008	4830000

Initialisation des pointeurs

47

- Par défaut, lorsque l'on declare un pointeur, on ne sait pas sur quoi il pointe.

- Comme toute variable, il faut l'initialiser. On peut dire qu'un pointeur ne pointe sur rien en lui affectant la valeur NULL.

Exemple :

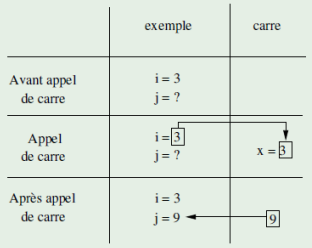
```
Int  i;
Int *p1, *p2;
p1 = &I;
p2 = NULL;
```

Passage de paramètre en entrée

. . . par un passage de paramètre par valeur

On copie la valeur du paramètre effectif dans le paramètre formel :

```
int carre (int x){
  return (x*x);
}

void exemple (){
  int i = 3;
  int j;

  j = carre (i);
  printf ("%d\n",j);
}
```

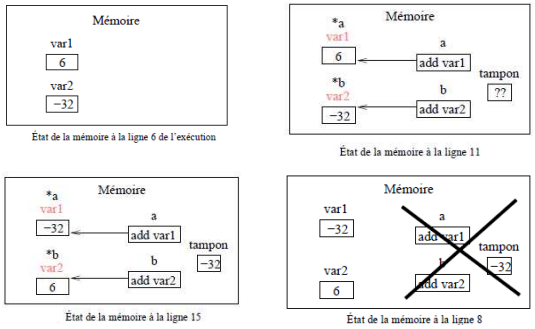


48

Passage de paramètres en entrée/sortie

```
1     void Echange(int* ,int *) ; /*prototypage de la fonction Echange*/
2
3     int main()
4     {
5       int var1=6,var2=-32;      /* déclaration de variables locales */
6
7       Echange(&var1,&var2) ;              /*Appel de Echange */
8     }
9     void Echange(int * a, int *b)
10    {
11      int tampon ;
12      tampon = *b ;
13      *b = *a ;
14      *a = tampon
15    }
```


Passage de paramètres en entrée/sortie



Passage de paramètres en entrée/sortie

51

void permut_v(int a, int b)
{
int x=a ;
a=b ;
b=x ;
}
	void main()
	{
	int i=5,j=3 ;
	permut_v(i,j) ;
	printf("i=%d, j=%d", i,j) ;
	}

main		permut_v
i=5, j=3	Appel →	a=5, b=3
i=5, j=3	← Retour	a=3, b=5 **Exécution**
les valeurs des variables i et j de la fct main ont été copiées dans les paramètres a et b de la fonction permut_v, la modification de a et b n'altère en aucun cas les valeurs de i et j		en mémoire a et b sont distinctes de i et j, les paramètres formels ont simplement reçu une copie des valeurs des variables i et j

Passage de paramètres en entrée/sortie

52

void permut_a(int *a, int *b)
{
int x =*a ;
*a = *b ;
*b =x ;
}
	void main()
	{
	int i=5,j=3 ;
	permut_a(&i,&j) ;
	printf("i=%d, j=%d", i,j) ;
	}

main		permut_v
i=5, j=3	Appel →	a=&i, b=&j,*a=5,*b=3
		Exécution ↓
	← Retour	*a=3, *b=5 → i=3,j=5
i=3, j=5		
La permutation a bien eu lieu car la fonction main a transmis à la fonction permut_a les adresses des variables i et j aux pointeurs a et b.		a contient l'adresse de i et b contient l'adresse de j → la modification de *a et *b modifie systématiquement les variables i et j de la fonction main.

Passage de paramètres en entrée/sortie


```c
#include <stdio.h>

void surface(int largeur, int longueur, int *res)
{
   *res=largeur*longueur;
}
void main()
{
   int l,L,surf;
   printf("\n Entrer la largeur:"); scanf("%d", &l);
   printf("\n Entrer la longueur:"); scanf("%d", &L);
   surface(l,L, &surf);   // appel de la fonction surface
   printf("\n Surface = %d\n",surf);
}
```

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 5 : Les tableaux et les algorithmes de Tri

Wided Miled
Aymen Sellaouti

Plan

Chapitre 5 -1
Les tableaux

Introduction

Problème

Ecrire un algorithme qui permet de lire les notes de 100 étudiants et d'afficher les notes des 10 premiers d'entre eux.

Introduction

5

```
ALGORITHME étudiant
Var n1, n2, ....., n100 : réel;
Début
        Lire (n1);
        Lire (n2) ;
        .....
        Lire (n100);

        Ecrire ('Voici les notes des dix premiers étudiants') ;

        Ecrire (n1);
        Ecrire (n2);
        .....
        Ecrire (n10);

Fin étudiant
```

Introduction

6

Nécessité d'utiliser 100 variables de même type pour saisir les notes de tous les étudiants,

➡ ce qui augmente la taille de l'algorithme

⇩

Une nouvelle structure permettant de ranger les notes des étudiants.

Tableau monodimensionnel

7

Lorsque les données sont nombreuses et de même type, afin d'éviter de multiplier le nombre des variables, on les regroupe dans un tableau

Définition

Un tableau mono-dimensionnel ou vecteur est une manière de ranger des éléments ou des valeurs de même type, il regroupe ces éléments dans une structure fixe et permet d'accéder à chaque élément par l'intermédiaire de son rang ou indice.

Tableau monodimensionnel

8

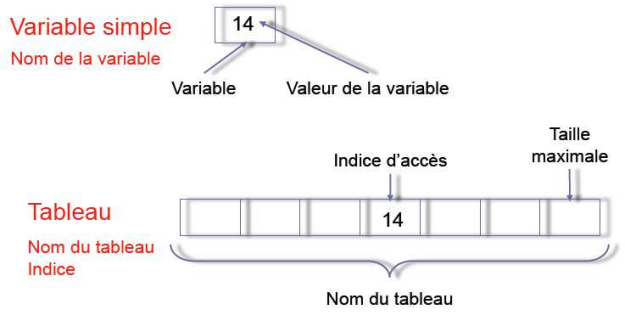
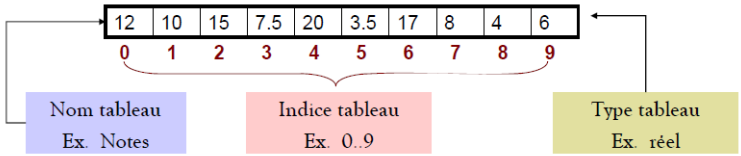


Tableau monodimensionnel

9

Le type d'un tableau précise le type (commun) de tous les éléments.



Syntaxe :

tableau [borne_inf .. borne_sup] de type_des_éléments

Tableau monodimensionnel

10

- Exemple :

45	54	1	-56	22	134	49	12	90	-26

- Chacun des dix nombres du tableau est repéré par son rang, appelé indice
- Déclaration : **Tab : tableau [0..9] de entier**
- Pour accéder à un élément du tableau, il suffit de préciser entre crochets l'indice de la case contenant cet élément.
- Exemple :

  Pour accéder au 5ème élément (22), on écrit : Tab[4]

Tableau monodimensionnel

11

Les instructions de lecture, écriture et affectation s'appliquent aux tableaux comme aux variables.

x ← Tab[0]

La variable x prend la valeur du premier élément du tableau, c'est à dire : 45

Tab[6] ← 43

Cette instruction a modifiée le contenu du tableau

Lecture et affichage

12

```
Procédure Lecture(E/S: tab: tableau [0,N-1]: Entier)
Var  i : Entier
Début
    Pour i de 0 à N-1 Faire
    écrire("Entrez la valeur" , i, "du tableau")
    lire(tab[i]) ;
    FinPour
Finprocédure
```

```
Procédure  Affichage (E: tab: tableau [0,N-1]: Entier)
Var  i : Entier
Début
    Pour i de 0 à N-1 Faire
    écrire("tab[",i,"]=",  tab[i]) ;
    FinPour
Finprocédure
```

Recherche d'un élément dans un tableau

13

On veut déterminer s'il existe un indice i allant de 0 à N-1 tel que val =T[i]

■ **Tableau non Ordonné**

```
Algorithme Recherche1
Var  i, val : Entier, T: tableau [0,N-1]: Entier, Trouve: Booléen
Début

    Trouve ← Faux
    Pour i de 0 à N-1 Faire
        Si  ( T[i]=val ) alors
            Trouve ← Vrai
        FinSi
    FinPour
Fin
```

Recherche d'un élément dans un tableau

14

■**Tableau ordonné**

```
ALGORITHME RechercheSeqTrie
Var
T : tableau [0..n-1] de entier
i, elem : entiers
Trouve : booléen
Début
        si elem > T[n-1] alors
           Trouve ← faux
        sinon
           i ← 0
           tantque T[i] < elem faire
                  i ← i + 1
           fintantque
           Trouvé ← (T[i] = elem)
        finsi
Fin
```

Recherche dichotomique dans un tableau trié

15

```
ALGORITHME RechercheDicho
Var
        T : tableau [0..N-1] de entier
        elem ,Binf, bsup : entier
        Trouve : booléen
Début
        binf←0; bsup ← nb-1; Trouve ←faux
        Répeter
           mil ←(binf+bsup) div 2

           si elem =T[mil] alors
                trouve ← vrai
           sinon Si elem<T[mil] alors
                bsup ← mil-1
                     sinon binf ← mil+1
                     finsi
           finsi

        Jusqu'à (Trouve ou binf>bsup)
Fin
```

Insertion d'un élément dans un tableau trié

16

Un tableau T de dimension N+1 contient N valeurs entières triées par ordre croissant; La (N+1)ième valeur est indéfinie.

Insérer une valeur VAL donnée au clavier dans le tableau T de manière à obtenir un tableau de N+1 valeurs triées.

Insertion d'un élément dans un tableau trié

17

Insérer une valeur dans un tableau trié

1. Lecture des éléments du tableau dans un ordre croissant
2. Lecture de la valeur a insérer VAL
3. Déplacer les éléments plus grands que VAL d'une position vers l'arrière.
4. VAL est copiée a la position du dernier élément déplacé
5. Afficher le nouveau tableau

Insertion d'un élément dans un tableau trié

18

```
Algorithme Insertion
Var  i, val : Entier, T: tableau [0,N]: Entier,
Début
    Pour i de 0 à N-1 Faire
    écrire("Entrez la valeur" , i, "du tableau ")
    lire(tab[i]) ;
    FinPour

    écrire("Entrez la valeur  à insérer")
    lire(val) ;
    i ← N-1
    Tantque (i>=0) ET (T[i]>val) Faire
        T[i+1]  ← T[i]
        i ← i-1
    FinTantQue
    T[i+1] ← val
Fin
```

Tableaux à deux dimensions

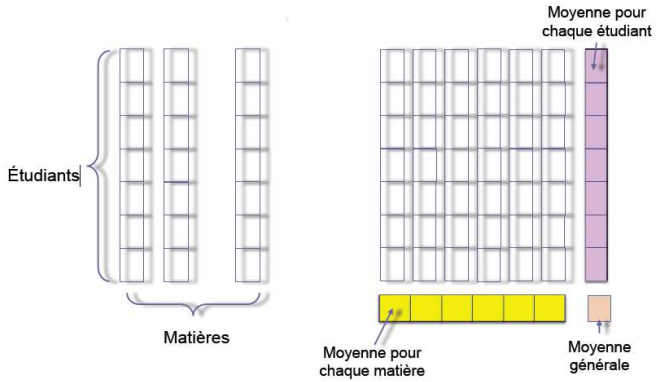
19

Problème

Comment sauvegarder et manipuler les notes des étudiants d'une classe, dans 6 matières?

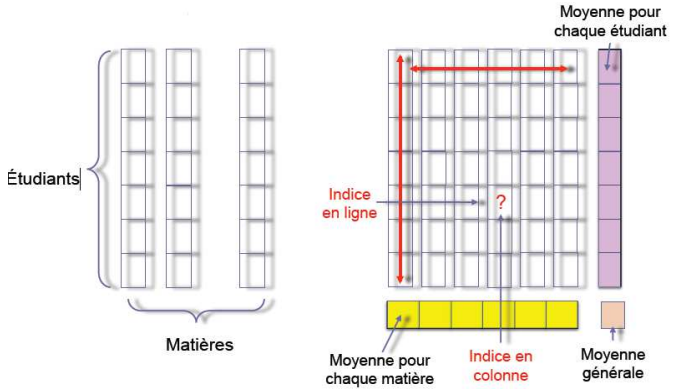
Tableaux à deux dimensions

20



Tableaux à deux dimensions

21



Tableaux à deux dimensions

22

Déclaration d'un tableau à deux dimensions

```
Var
    Tab : tableau [0 .. NMAX, 0 .. MMAX] de type_des_éléments
```

19

Appel :

```
X ← Tab[2,3]
```

`Tab[2,3]` permet d'accéder à l'élément de la matrice qui se trouve à l'intersection de la ligne 2 et de la colonne 3

Tableaux à deux dimensions

23

```
Algorithme InitTableau
Const
(N ,M← 100): entier
Var
    i,j : entier,
    Tab : tableau [0..N-1, 0..M-1] d'entier
Début
        Pour i ← 0 à N-1 faire
            Pour j ← 0 à M-1 faire
                Tab[i,j] ← 0
            FinPour
        FinPour
Fin
```

```
Algorithme AfficheTableau
Const
(N ,M← 100): entier
Var
    i,j, nbc, nbl : entier,
    Tab : tableau [0..N-1, 0..M-1] d'entier
Début
        Pour i ← 0 à nbc faire
            Pour j ← 0 à nbl faire
                Ecrire(Tab[i,j])
            FinPour
        FinPour
Fin
```

24

Chapitre 5 -2
Les algorithmes de tri

Tri d'un tableau

- Les tableaux permettent de stocker plusieurs éléments de même type au sein d'une seule entité. Lorsque le type de ces éléments possède un ordre total, on peut les ranger en ordre croissant ou décroissant.
- Trier un tableau c'est donc ranger ses éléments selon un **ordre déterminé.**
- Un **algorithme de tri** est un algorithme qui prend en entrée un tableau et qui donne en sortie ce même tableau avec les éléments ordonnés suivant une relation R.

Tri d'un tableau

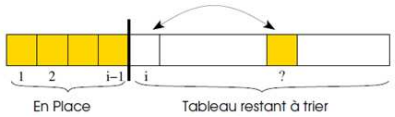
- Le tri s'effectue par des permutations successives des éléments du tableau. C'est un processus itératif qui consiste à parcourir tous les éléments du tableau, en appliquant à chaque itération une stratégie de tri.
- Il existe plusieurs méthodes de tri qui se différencient par leur **complexité** d'exécution et leur complexité de compréhension :
  - Algorithmes lents
    - Tri par sélection
    - Tri par insertion
    - Tri à bulles
  - Algorithmes rapides
    - Tri fusion
    - Tri rapide

Tri par sélection

L'idée du tri du consiste à chaque étape à rechercher le plus petit élément non encore trié et à le placer à la suite des éléments déjà triés. A une étape i, les $i-1$ plus petits éléments sont en place, et il nous faut sélectionner le ième élément à mettre en position i.

A chaque étape i

1. Rechercher le $i$ème élément → 1. Chercher le plus petit élément du tableau restant
2. Le placer en position i → 2. Echanger cet élément avec la carte en position i



Tri par sélection

Par exemple, pour trier <101, 115, 30, 63, 47, 20>, on va avoir les boucles suivantes :

- i=1 <101, 115, 30, 63, 47, 20>
- i=2 <20, 115, 30, 63, 47, 101>
- i=3 <20, 30, 115, 63, 47, 101>
- i=4 <20, 30, 47, 63, 115, 101>
- i=5 <20,30, 47, 63, 115, 101>
- Donc en sortie : <20, 30, 47, 63, 101, 155>

Tri par sélection

```
Procédure TriSélection (E/S: T: tableau[0,N-1]: entier)
Var: i, j, indice_min : entier
Début
        Pour i = 0 à N-2 Faire
        indice_min ← i
                Pour  j=i+1  à N-1  Faire
                        Si T[j] < T[indice_min]  alors
                  indice-min ← j
                Fin Si
                Fin Pour
        Echanger (T[i], T[indice-min])
        Fin Pour
Fin Procédure
```

Tri par insertion

À la i^{ème} étape :

- Cette méthode de tri insère le i^{ème} élément T[i] à la bonne place parmi T[0], T[2]…T[i-1].
- Après l'étape i, tous les éléments entre les positions 0 à i sont triés.
- Les éléments à partir de la position i ne sont pas triés.

Pour insérer l'élément T[i] :

- Si T[i] ≥T[i-1] : insérer T[i] à la i^{ème} position !
- Si T[i] <T[i-1] : déplacer T[i] vers le début du tableau jusqu'à la position j ≤ i telle que T[i] ≥T[j] et l'insérer en position j.

Tri par insertion

- **Étape 1:** on commence à partir du 2 ième élément du tableau (élément 4). On cherche à l'insérer à la bonne position par rapport au sous tableau déjà trié (formé de l'élément 9) :

9	4	1	7	3	⇒	4	9	1	7	3

- **Étape 2:** on considère l'élément suivant (1) et on cherche à l'insérer dans une bonne position par rapport au sous tableau trié jusqu'à ici (formé de 4 et 9):

4	9	1	7	3	⇒	1	4	9	7	3

- **Étape 3:**

1	4	7	9	3

- **Étape 4:**

1	3	4	7	9

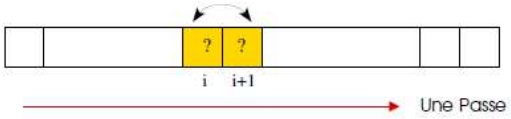
Tri par insertion

```
Procédure TriInsertion (E/S: T: tableau[0,N-1]: entier)
Var: i,j, temp:entier
Début
Pour i allant de 1 à N-1 Faire
        temp ← T[i]
        j ← i-1
        Tant que (j>=0) et (T[j] > temp)  Faire
                T[j+1] ← T[j]
            j ← j-1
        Fin Tant que
        T[j+1]← temp
Fin Pour
Fin Procédure
```

Tri à bulles

Principe de la méthode : Sélectionner le minimum du tableau en parcourant le tableau de la fin au début et en échangeant tout couple d'éléments consécutifs non ordonnés.

Tant que le tableau n'est pas trié
1. Effectuer une passe →
   a. Parcourir le tableau
   b. Si 2 éléments successifs ne sont pas dans le bon ordre, les échanger



Tri à bulles

Prenons la liste de chiffres « 5 1 4 2 8 »

Première étape:
(**5 1** 4 2 8) ⟶ (**1 5** 4 2 8) Les éléments 5 et 1 sont comparés, et comme 5 > 1, l'algorithme les intervertit.
(1 **5 4** 2 8) ⟶ (1 **4 5** 2 8) Interversion car 5 > 4.
(1 4 **5 2** 8) ⟶ (1 4 **2 5** 8) Interversion car 5 > 2.
(1 4 2 **5 8**) ⟶ (1 4 2 **5 8**) Comme 5 < 8, les éléments ne sont pas échangés.

Deuxième étape:
(**1 4** 2 5 8) ⟶ (**1 4** 2 5 8) Même principe qu'à l'étape 1.
(1 **4 2** 5 8) ⟶ (1 **2 4** 5 8)
(1 2 **4 5** 8) ⟶ (1 2 **4 5** 8)
(1 2 4 **5 8**) ⟶ (1 2 4 **5 8**)
À ce stade, la liste est triée, mais pour le détecter, l'algorithme doit effectuer un dernier parcours.

Troisième étape:
(**1 2** 4 5 8) ⟶ (**1 2** 4 5 8)
(1 **2 4** 5 8) ⟶ (1 **2 4** 5 8)
(1 2 **4 5** 8) ⟶ (1 2 **4 5** 8)
(1 2 4 **5 8**) ⟶ (1 2 4 **5 8**)
Comme la liste est triée, aucune interversion n'a lieu à cette étape, ce qui provoque l'arrêt de l'algorithme.

Tri à bulles

```
procédure TriBulles (E/S t : Tableau[0,N-1] :entier)
var i,k : entier
début
pour i ←0 à N-1 faire
  pour k ← i+1 à N-1 faire
    si t[k]<t[k-1] alors
      echanger(t[k],t[k-1])
    FinSi
  FinPour
FinPour
Fin Procédure
```

Tri à bulles- optimisation

```
procédure TriBulOpt (E/S t : Tableau[0,N-1]:entier)
var i : entier, stop: booléen
Début
 Répéter
   Stop ← faux
    pour i ←0 à N-1 faire
      si t[i]>t[i+1] alors
        echanger(t[i],t[i-1])
          stop ← vrai
           FinSi
         FinPour
 Jusqu'à (stop ← faux)
Fin
```

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 6 Les chaînes de caractères

Wided Miled
Aymen Sellaouti

Chapitre 6
Les chaînes de caractères

Plan

Introduction

Définition :
Une chaîne de caractères C est un tableau unidimensionnel de caractères.
Par convention, la fin de la chaîne de caractères est indiquée par le caractère nul **'\0'**.
Par conséquent, une chaîne de caractères occupe en mémoire un espace correspondant au nombre de caractères significatifs de la chaîne plus un.

Déclaration
Déclaration en langage algorithmique : **chaîne <NomVariable>**
Déclaration en C : **char <NomVariable> [<Longueur+1>];**

Exemples :
char NOM [20];
char PRENOM [20];
char PHRASE [100];

- 'x' est un caractère constant, et
- "x" est un tableau de caractères qui contient deux caractères

Introduction

5

```
char TXT[5] = "Hello";
```

TXT:	'H'	'e'	'l'	'l'	'o'	☠

ERREUR pendant l'exécution

```
char TXT[8] = "Hello";
```

TXT:	'H'	'e'	'l'	'l'	'o'	'\0'	0	0

```
char TXT[6] = "Hello";
```

TXT:	'H'	'e'	'l'	'l'	'o'	'\0'

Introduction

6

```
char TXT[5] = "Hello";
```

TXT:	'H'	'e'	'l'	'l'	'o'	☠

ERREUR pendant l'exécution

```
char TXT[4] = "Hello";
```

ERREUR pendant la compilation

```
char A[6] = "Hello";
```

A:	'H'	'e'	'l'	'l'	'o'	'\0'
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

Introduction

7

La précédence des caractères dans l'alphabet est dépendante du code de caractères utilisé. Pour le code ASCII, nous pouvons constater l'ordre suivant:

. . . ,0,1,2, ... ,9, . . . ,A,B,C, ... ,Z, . . . ,a,b,c, ... ,z, . . .

Exemples :

"ABC" précède "BCD" car 'A'<'B'
"ABC" précède "B" car 'A'<'B'
"Abc" précède "abc" car 'A'<'a'
"ab" précède "abcd" car "" précède "cd"
" ab" précède "ab" car ' '<'a'

Exemple :

écrivez un programme qui prend une chaîne en minuscules et qui la convertit en majuscules

Introduction

8

```c
#include <stdio.h>

// a -> z code ascii 97 -> 122
// A -> Z code ascii 65 -> 90

int main(void)
{
  int indice;
  char mot[255];

  // lecture du mot
  printf("entrez un mot : ");
  scanf("%s", mot);

  // pour chaque caractere, s'il est en minuscule, le mettre en majuscule
  indice = 0;
  while(mot[indice] != '\0'){
    if( (mot[indice] >= 97) && (mot[indice] <= 122) )
      mot[indice] = mot[indice] - 32 ;
    indice++;
  }

  // afficher le resultat apres traitement
  printf("le mot en minuscules : %s\n", mot);

  return 0;
}
```

Fonctions sur les chaînes de caractères : Lecture et écriture

9

Les fonctions de <stdio.h>

- **scanf** : scanf("%s" , chaine)
- **gets** : Lecture de chaînes de caractères **gets(<Chaîne>)**
- **printf** : printf("%s" , chaine)
- **puts** : L'écriture de chaînes de caractères **puts(<Chaîne>)**
- **sscanf :** C'est la version de scanf dédiée aux chaines de caractères, càd que la source de lecture n'est plus l'entrée standard **stdin** mais une chaine de caractère et sa syntaxe est :
  sscanf(chaine_source,"%type",&var_dest);
- **sprintf**(chaine_source,"données à introduire",variables);

Fonctions sur les chaînes de caractères : la bibliothque <string.h>

10

Les fonctions de <string.h>

strlen(s) : fournit la longueur de la chaîne sans compter le '\0'
strcpy(s, t) : copie t vers s, Remarque : le '\0' est aussi copié
strncpy(s,t,n) : copie les n premiers caractères de t vers s
strcat(s, t) : ajoute t à la fin de s (à partir du '\0' de s qui sera décalé)
strcat(s,t,n) : concatène les n premiers caractères de t à la fin de s
strcmp(s, t) : compare s et t lexicographiquement :
négatif si s précède t
zéro si s est égal à t
positif si s suit t
strncmp(s, t, n) compare les n premiers caractères de s et t

Fonctions sur les chaînes de caractères : Lecture et écriture formatée

11

RAPPEL PRINTF ET SCANF (DE MÊME POUR SPRINTF ET SSCANF)

printf

- Concernant printf, un certain nombre de caractères optionnels peuvent être insérés entre le symbole % et le caractère spécifiant la conversion (d,x,e,…) :
  - Le signe – pour demander un cadrage à gauche, au lieu du cadrage à droite par défaut.
  - un nombre indiquant la taille minimale en caractères du champs à imprimer. Des espaces jouant le rôle de caractère de remplissage.
  - un point décimal, suivi d'un nombre donnant la précision de la partie fractionnaire, c'est à dire le nombre de chiffre significatifs après le point. Si la donnée n'est pas du type flottant, ce nombre représente la taille maximale du champs à imprimer.

Fonctions sur les chaînes de caractères : Lecture et écriture formatée

12

RAPPEL PRINTF ET SCANF (DE MÊME POUR SPRINTF ET SSCANF)

printf

- Exemple :
  - %8d : Imprime un nombre en décimal cadré à droite dont la longueur du champ imprimable est de huit caractéres. Des espaces de remplissage précédent le nombre.

  X=12345
  Printf (« %8d »,x);// affichera 3 espaces suivies de 12345
  Printf (« %-8d »,x);// affichera 12345 suivi de3 espaces
  - %-25s : Imprime une chaîne de caractères cadrée à gauche assurant une longueur minimum de 25 caractères.
  - %.6f : Imprime un nombre flottant avec un maximum de six chiffres significatifs.

Fonctions sur les chaînes de caractères : Lecture et écriture formatée

13

RAPPEL PRINTF ET SCANF (DE MÊME POUR SPRINTF ET SSCANF)

scanf

- Pour limiter le nombre de caractère lu par scanf, on mentionne le nombre de caractères à lire entre le et le caractère spécifiant la conversion
  - %3s : lira seulement les 3 premiers caractères.
- Le caractère * précédé de % spécifie que la valeur lue sera ignoré, donc non affecté à la variable suivante.
  - **scanf**("%d%*s%d%*s%d%*s", &heure, &minutes, &secondes) ; permettra de correctement extraire les données pour l'entré suivante : 17 H 35 min 30 secondes puisque les chaines "h","min" et "secondes" seront ignorées.

Fonctions sur les chaînes de caractères : Lecture et écriture formatée

14

DIFFÉRENCE ENTRE SCANF ET GETS

scanf

- Avec le code de %s de scanf, nous utilisons les délimiteurs habituels qui sont l'espace ou la fin de ligne. Cela induit l'incapacité de scanf à lire une chaine contenant des espaces.
- Exemple : Med Ali ne pourra pas être lu et stocké par scanf dans une chaine de caractère.
- Le caractère délimiteur n'est pas consommé (l'espace ou le \n ne sont pas lu dans la chaine)

gets

- Seul la fin de ligne (\n) sert de délimiteur
- Le \n est lu dans la chaine, il ne risque donc pas d'être pris en compte dans la prochaine lecture

Il est donc préférable d'utiliser le gets plutôt que le scanf

Exemple

15

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char t1[50], t2[50];

    strcpy(t1, "Hello, world!");
    strcpy(t2, "**************");
    strncpy(t1, t2, 3);
    printf("%s\n", t1);

    return 0;
}
```

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char t[50];

    strcpy(t, "Hello, world");
    strcat(t, " from");
    strcat(t, " strcpy");
    strcat(t, " and strcat");
    printf("%s\n", t);

    return 0;
}
```

Fonctions sur les chaînes de caractères Les fonctions de stdlib

16

FONCTIONS SUR LES CHAÎNES DE CARACTÈRES

Les fonctions de <stdlib>

▪Chaîne --> Nombre

atoi(s) : retourne la valeur numérique représentée par <s> comme int
atol(s) : retourne la valeur numérique représentée par <s> comme long
atof(s) : retourne la valeur numérique représentée par <s> comme double (!)

▪Nombre --> Chaîne

itoa (n_int, s, b)
ltoa (n_long, s, b)
ultoa (n_uns_long, s, b)

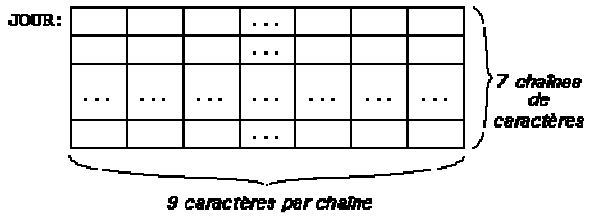
Avec : n_int: est un nombre du type int
n_long : est un nombre du type long
n_uns_long : est un nombre du type unsigned long
s : est une chaîne de caractères
b : est la base pour la conversion (2 ... 36)

Les tableaux de chaines de caractères

17

Un tableau de chaînes de caractères correspond à un tableau à deux dimensions du type char, où chaque ligne contient une chaîne de caractères.

Déclaration :
La déclaration char JOUR[7][9]; réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8 caractères significatifs).



Les tableaux de chaines de caractères

18

JOUR:									
	'l'	'u'	'n'	'd'	'i'	'\0'			
	'm'	'a'	'r'	'd'	'i'	'\0'			
	'm'	'e'	'r'	'c'	'r'	'e'	'd'	'i'	'\0'
	...	...	...	...	...	...	...	...	...
	'd'	'i'	'm'	'a'	'n'	'c'	'h'	'e'	'\0'

Char JOUR[7][9]={"lundi","mardi","mercredi","jeudi","vendredi","samedi","dimanche"};
Les tableaux de chaînes sont mémorisés ligne par ligne.
La variable JOUR aura donc besoin de 7*9*1 = 63 octets en mémoire.

Exemple : L'exécution des trois instructions suivantes:

```
char JOUR[7][9]= {"lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"};
int I = 2;
printf("Aujourd'hui, c'est %s !\n", JOUR[I]);
```

affichera la phrase: Aujourd'hui, c'est mercredi !

Les tableaux de chaines de caractères

19

Affectation
L'attribution d'une chaîne de caractères à une composante d'un tableau de chaînes se fait en général à l'aide de la fonction strcpy:
Exemple : La commande strcpy(JOUR[4], "Friday");
changera le contenu de la 5e composante du tableau JOUR de "vendredi" en "Friday".

Accès aux caractères
Evidemment, il existe toujours la possibilité d'accéder directement aux différents caractères qui composent les mots du tableau.

Exemple : L'instruction

```
for(i=0; i<7; i++)
    printf("%c  ", JOUR[i][0]);
```

va afficher les premières lettres des jours de la semaine: l m m j v s d

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 7 Les enregistrements

Wided Miled
Aymen Sellaouti

Chapitre 7
Les enregistrements

Plan

Définition

Contrairement aux tableaux qui sont des structures de données dont tous les éléments sont de même type, les enregistrements sont des structures de données dont les éléments peuvent être de types différents.

Les éléments qui composent un enregistrement sont appelés champs.

Les enregistrements sont aussi appelés structures, en analogie avec le langage C.

Syntaxe

5

Syntaxes :

(notation inspirée du Pascal)

Type
nom_type **= enregistrement**
nom_champ1: *type_champ1*
...
nom_champn: *type_champn*
finenreg

ou

(notation inspirée du C)

ou
Structure *nom_type*
nom_champ1: type_champ1
...
nom_champN: type_champN
FinStruct

Exemple:

Type
tpersonne = **enregistrement**
nom : chaîne
prénom : chaine
âge : entier
finenreg

ou

Type
Structure tpersonne
nom : chaîne
prénom : chaine
âge : entier
FinStruct

Déclaration

6

Syntaxe

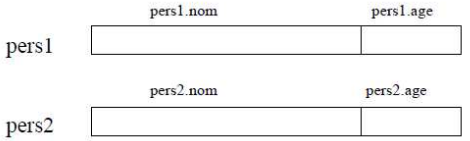
Var
nom_var : *nom_type*

Exemple:

Var
pers1, pers2, pers3 : tpersonne

Type
Structure tpersonne
nom : chaîne
age : entier
FinStruct

Représentation:
les enregistrements sont composés de plusieurs zones de données, correspondant aux champs



Exemple:
Soit l'entité suivante:

PRODUIT
code
lib
paht
pvht
txtva

légende:

code: code alphanumérique du produit
lib: libéllé
paht: prix d'achat hors taxes
pvht: prix de vente hors taxes
txtva: taux de TVA applicable

// Il faut d'abord définir le type structuré correspondant:

Type
produit = **enregistrement**
code: chaîne
lib: chaîne
paht: réel
pvht: réel
txtva: réel
finenreg

ou

Type
Structure produit
code: chaîne
lib: chaîne
paht: réel
pvht: réel
txtva: réel
FinStruct

// Ensuite il est possible de déclarer deux variables de ce type

Var
prod1, prod2 : produit

7

Accès aux champs

8

Accès aux champs d'un enregistrement

nom_enregistrement . nom_champ
représente la valeur mémorisée dans le champ de l'enregistrement

Par exemple, pour accéder à l'âge de la variable pers2, on utilise l'expression:
pers2.âge

Imbrication d'enregistrements

9

Un type structure peut être utilise comme type pour des champs d'un autre type de structure.

```
TYPE
Structure date
    jour: entier
    mois: chaîne
    année: entier
FinStruct

Structure personne
    nom: chaîne
    ddn: date
FinStruct
```

Pour accéder à l'année de naissance d'une personne, il faut utiliser deux fois l'opérateur '.'

pers1.ddn.année

Exemple

10

Un produit est livré par un seul fournisseur. Un fournisseur est caractérisé par son code, sa raison sociale, son adresse et son numéro de téléphone.

```
Type
Structure adresse
        num : entier
        rue: chaîne
        cp: chaîne
        ville: chaîne
FinStruct

Structure fournisseur
        code_frs : chaîne
        raison_sociale: chaîne
        ad_frs: adresse
        tel: chaîne
FinStruct

Structure Produit
        code: chaîne
        lib: chaîne
        paht: réel
        pvht: réel
        txtva: réel
        frs: fournisseur
FinStruct
```

Exemple

11

```
Var
    p: produit
```

Voilà l'instruction qui permet d'afficher le numéro de téléphone du fournisseur du produit p.frs.tel

```
Ecrire( "téléphone du fournisseur de ", p.lib, " : ", p.frs.tel )
```

Les tableaux d'enregistrements

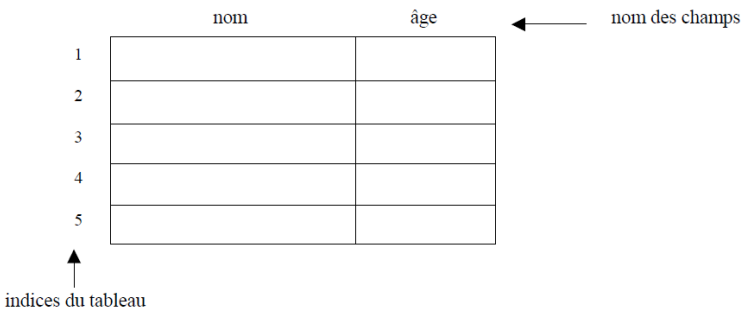
12

```
Const
NP = 20                    // nombre de personnes du groupe

Type
Structure personne
        nom: chaîne
        age: entier
FinStruct

Var
groupe:  tableau[1..NP] de personnes
```

Les tableaux d'enregistrements

13

groupe[2] représente la deuxième personne du groupe
groupe[2].nom représente le nom de la deuxième personne du groupe



Les Structures avec le Langage C

14

Syntaxe et déclaration

15

Il y a deux définitions possibles

```
struct Nom_structure
       {
          <type1> var1;
          <type2> var2;
          <typeN> varN;
       };

// Définition d'une variable :
struct Nom_structure Variable;
```

```
typedef struct
          {
            <type1> var1;
            <type2> var2;
            <typeN> varN;
          } Nom_structure;

// Définition d'une variable :
Nom_structure Variable;
```

Exemple

16

Un livre est caractérisé par les données suivantes :
un titre
un code
un auteur
un éditeur
un prix
On désire définir un type ouvrage tel que chaque élément de ce type soit caractérisé par ces données.

En C :

```
struct ouvrage
{
  char titre [20] ;
  int code ;
  char auteur [30] ;
  char editeur [20] ;
  float prix ;
}
```

Déclaration et accès

17

A partir de cette définition, on pourra définir des variables de type ouvrage, la déclaration se fait ainsi :

struct ouvrage l ;
la variable l est composée de :

Pour accéder à un champ d'une variable de type ouvrage :
nomvar.nomchamp . Ainsi,
l.titre est une variable de type chaîne de caractères.
l.code est une variable de type entier.

Exercices

18

Exercice 1 :
Ecrire une fonction qui permet d'afficher un livre.

Exercices

19

Exercice 1 :
Ecrire une fonction qui permet d'afficher un livre.

```
void affich_livre (struct ouvrage l)
{
   puts (l.titre) ;
   printf(«%d»,l.code) ;
   puts (l.auteur) ;
   puts (l.editeur) ;
   printf («%f»,l.prix) ;
}
```

Exercices

20

Exercice 2 :
Ecrire une fonction qui permet de saisir un livre au clavier. Il y 2 solutions :

Exercices

21

Exercice 2 :

Ecrire une fonction qui permet de saisir un livre au clavier. Il y 2 solutions :

On retourne l'ouvrage à la fin de la fonction

```
struct ouvrage lecture1 ()
{
   struct ouvrage l ;
   gets (l.tite) ;
   scanf («%d»,&l.code) ;
   gets (l.auteur) ;
   gets (l.editeur) ;
   scanf («%f»,&l.prix) ;
   return l ;
}
```

On passe l'ouvrage à la fonction par adresse

```
void lecture2 (struct ouvrage *l)
{
   gets (l->tite) ;
   scanf («%d»,&(l->code)) ;
   gets (l->auteur) ;
   gets (l->editeur) ;
   scanf («%f»,&(l->prix)) ;
}
```

➔ l'opérateur -> permet d'accéder aux champs d'une variable contenant l'adresse d'un enregistrement.

Exercices

22

```
void main()
{
   struct ouvrage l1, l2 ;
   l1 = lecture1 () ;
   lecture2 (&l2) ;
   afich_livre(l1) ;
   afich_livre(l2) ;
}
```

On peut déclarer un tableau de type ouvrage :

struct ouvrage livre [10] ;

livre est un tableau de 10 éléments de type struct ouvrage.
Pour accéder à un champ de l'élément d'indice i : livre [i].champ

Exercices

23

Exercice 3 :

Ecrire une fonction qui permet de saisir un tableau de livres au clavier.

Exercices

24

Exercice 3 :

Ecrire une fonction qui permet de saisir un tableau de livres au clavier.

```
void lecture_tab (int *n, struct ouvrage livre[ ])
{
   int i ;
   do
   {
         printf (« Donner le nombre d'ouvrages ») ;
         scanf (« %d »,n) ;
   }while ((*n <= 0) || (*n > 10)) ;
   for (i = 0 ; i < *n ; i ++)
         lecture2 (livre+i) ;
}
```

Exercices

25

```
typedef struct
{
   int jour ;
  char mois [12] ;
   int annee ;
}date ;
```

```
typedef struct
{
   char titre [20] ;
   int code ;
   char auteur [30] ;
   char editeur [20] ;
   float prix ;
   date date_edition ;
}ouvrage ;
```

Un ouvrage est :

Exercices

26

Exercice 4 :

Ecrire une fonction qui permet de saisir un ouvrage au clavier selon la structure précédente.

```
void lecture3 (ouvrage *l)
{
   gets (l->tite) ;
   scanf («%d»,&l->code) ;
   gets (l->auteur) ;
   gets (l->editeur) ;
   scanf («%f»,&l->prix) ;
   scanf(«%d»,&l->date_edition.jour) ;
   gets(l->date_edition.mois) ;
   scanf(«%d»,&l->date_edition.annee) ;
}
```

Exercices

27

Exercice 5 :

Ecrire une fonction qui permet d'afficher un livre selon la structure précédente.

```
void Affich_livre2 (ouvrage l)
{
   puts (l.titre) ;
   printf («%d»,l.code) ;
   puts (l.auteur) ;
   puts (l.editeur) ;
   printf («%f»,l.prix) ;
   printf («%d»,l.date_edition.jour) ;
   puts (l.date_edition.mois);
   printf («%d»,l.date_edition.annee) ;
}
```

Exercices

28

Si on veut ajouter un tableau indiquant le nombre de sorties du livre pour chaque jour du mois :

```
typedef struct
{
   char titre [20] ;
   int code ;
   char auteur [30] ;
   char editeur [20] ;
   float prix ;
   date date_edition ;
   int nb_sorties[31] ;
}ouvrage ;
```

Un ouvrage est :

Exercices

29

Exercice 6 :

Ecrire une fonction qui permet d'afficher un livre selon la structure précédente.

```
void Affich_livre3 (ouvrage l)
{
   puts (l.titre) ;
   printf («%d»,l.code) ;
   puts (l.auteur) ;
   puts (l.editeur) ;
   printf («%f»,l.prix) ;
   printf («%d»,l.date_edition.jour) ;
   puts (l.date_edition.mois);
   printf («%d»,l.date_edition.annee) ;
   for (i = 0; i <31; i ++)
         printf ("%d",l.nb_sorties[i]);
}
```

Exercices

30

Exercice7 :

On désire gérer les comptes bancaires de N personnes. Chaque compte est défini par les données suivantes : [le numéro de compte, le nom de la personne, le solde].

Définir la structure de données nécessaire pour représenter un compte.
Ecrire une fonction **saisir_compte**() qui permet de saisir les données d'un compte.
Ecrire une fonction **saisie**() qui saisie N comptes dans un tableau global C.
Ecrire une fonction **affiche_compte**() qui permet d'afficher un compte, prévoir une fonction exist() qui reçoit un numéro de compte et retourne vrai si le compte existe dans le tableau C et faux sinon.
Ecrire une fonction **solde**() qui renvoie le solde d'une personne dont le numéro de compte est passé en paramètre.
Ecrire une fonction **retrait**() qui reçoit un numéro de compte et un montant et met à jour le solde du compte. (le compte peut être débiteur d'une valeur de : -1000)
Ecrire une fonction **versement**() qui reçoit un numéro de compte et un montant et met à jour le solde du compte.

Ecrire le programme main qui permet de tester ces différentes opérations.

Exercices

31

Exercice8 :

On désire gérer la vie scolaire d'une classe de N élèves. Pour chaque élève, on saisit une fiche définie par les données suivantes : [le numéro de la CIN, le nom, la date de naissance, la moyenne]. La date est représentée par les trois valeurs : [jour, mois, année].

Après avoir défini les structures de données nécessaires, écrire les fonctions C suivantes :

Remplir_tab : permet de saisir les données de N élèves dans un tableau passé en paramètre.
Affiche_tab : qui permet d'afficher les données de N élèves du tableau passé en paramètre.
Rech_nom : qui recherche un élève d'un nom donné.
Rech_date : qui permet de rechercher un élève avec une date de naissance donnée.
Moyenne_classe : qui renvoie la moyenne générale de la classe.
Ajout_fiche : qui permet d'ajouter une fiche à la fin du tableau.
Supprime_fiche : qui permet de supprimer une fiche du tableau (choisir un champ de recherche pour la suppression).

Les fonctions de recherche retournent l'indice de la fiche correspondante dans le tableau (si elle existe) et -1 si la fiche n'existe pas.

Ecrire une fonction main proposant un menu de choix réalisant les différentes opérations implémentées.

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 8 Les fichiers

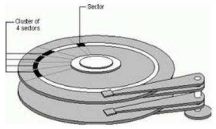
Wided Miled
Aymen Sellaouti

Chapitre 8
Les fichiers

Plan

Introduction

Disque Dur



Entrées/Sorties

Programme

- Ouvrir fichier X
- Lire D du fichier X
- Fermer fichier X
- R = Traitement (D)
- Ouvrir fichier Y
- Ecrire R dans fichier Y
- Fermer fichier Y

Introduction

5

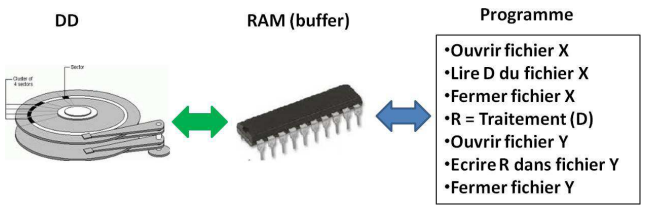
Il existe deux types de Fichiers (définitions Wikipédia) :

❑**Les Fichiers textes** : sont les fichiers dont le contenu représente uniquement une suite de caractères imprimables, d'espaces et de retours à la ligne (.txt,...). Ils peuvent être lus directement par un éditeur de texte.

❑**Les Fichiers binaires** : sont les fichiers qui ne sont pas assimilables à des fichiers textes (.exe, .mp3, .png,...). Ils ne peuvent pas être lus directement par un éditeur de texte .

Introduction

6



Dans mon programme , le système d'exploitation fait :

- Ouvrir un fichier.
  ⇒ créer un *buffer (b)* dans la RAM.
- Lire/écrire dans le fichier ouvert.
  ⇒ lire/écrire dans *b*.
- Fermer le fichier.
  ⇒ "flusher" le contenu de *b*, libérer *b*,...

Introduction

7

- En langage C, les informations nécessaires à maintenir l'association **programme** ⇔ **buffer** ⇔ **disque dur** sont décrites dans une structure `FILE` (`stdio.h`).
- Parmi les informations stockées dans la structure `FILE` , on trouve :
  - *le N° du fichier à ouvrir,*
  - *le type d'ouverture (lecture/écriture),*
  - *l'adresse du **buffer** associé,*
  - *la position du curseur de lecture,*
  - *la position du curseur d'écriture,*
  - ...
- Pour utiliser un fichier, il faut donc commencer par déclarer une variable de type `FILE`, ou plus exactement un pointeur sur `FILE` (`FILE *`), qu'on appel aussi *flux de données* :

```
FILE * nomPointeurFichier;
```

Ouverture et fermeture d'un fichier

8

Le language C offre deux fonctions pour l'ouverture et la fermeture d'un fichier :

- La fonction `fopen` : permet d'ouvrir un fichier, suivant un *mode*, et retourne un flux (pointeur sur `FILE`).

```
FILE * fopen(char* nomFichier, char* mode)
```

  La fonction retourne `NULL` si l'ouverture n'est pas possible

- La fonction `fclose` : permet de fermer un fichier (un flux) ouvert.

```
void fclose(FILE * pf)
```

Exemples

9

```
// Déclaration du flux
FILE * fp ;

// Ouvrir le fichier ./test.text en écriture
// et association au flux
if ((fp=fopen("./test.text","w"))==NULL){
    printf("Impossible d'ouvrir le fichier \n");
    return -1;
}

...

// Fermeture du flux (du fichier)
fclose(fp);
```

Modes d'ouverture d'un fichier

10

Les différents modes d'ouvertures d'un fichier sont :

Mode	Signification
"r"	ouverture d'un fichier texte en lecture
"w"	ouverture d'un fichier texte en écriture
"a"	ouverture d'un fichier texte en écriture à la fin
"rb"	ouverture d'un fichier binaire en lecture
"wb"	ouverture d'un fichier binaire en écriture
"ab"	ouverture d'un fichier binaire en écriture à la fin
"r+"	ouverture d'un fichier texte en lecture/écriture
"w+"	ouverture d'un fichier texte en lecture/écriture
"a+"	ouverture d'un fichier texte en lecture/écriture à la fin
"r+b"	ouverture d'un fichier binaire en lecture/écriture
"w+b"	ouverture d'un fichier binaire en lecture/écriture
"a+b"	ouverture d'un fichier binaire en lecture/écriture à la fin

Lecture/Ecriture dans un fichier

11

- Une fois le fichier ouvert, le langage C permet plusieurs types d'accès à un fichier :
  - Par caractère
  - Par ligne
  - Par enregistrement
  - Par données formatées

Accès caractère par caractère

12

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers textes existent :

- `int getc(FILE * pf)` : retourne le caractère suivant du flux `pf`. Elle retourne la constante `EOF` si elle rencontre la fin du fichier ou en cas d'erreur.
- `int putc(int c, FILE * pf)` : écrit le caractère c dans le fichier associé à`pf`. Retourne le caractère écrit ou `EOF` en cas d'erreur.

Remarques :

- `getchar()` ⇔ `getc(stdin)`
- `putchar(c)` ⇔ `putc(c,stdout)`

Accès par ligne

13

On peut accéder au contenu du fichier ligne par ligne ; en considérant les lignes comme chaîne de caractères dans les fichiers texte. Ainsi la fonction :

char * fgets(char *S, int max, FILE *f)

permet de lire une chaîne de caractères en s'arrêtant au caractère '\n' ou bien à max-1 caractères. Le résultat est stocké dans la chaîne de caractères S.

S : chaîne de caractères où sera stockée la ligne lue
max : nombre maximum de caractères à lire, en général cette variable le nombre maximum de caractères pouvant être lus, c'est à dire la taille de la zone de stockage: sizeof(s);
f : descripteur du fichier

Accès par ligne

14

La fonction ajoute à la chaîne S le caractère '\0' après le dernier caractère qu'elle a stocké dans le tableau.
La fonction retourne le pointeur S reçu en paramètre. Autrement dit elle retourne la ligne de texte lue à partir du fichier et stockée dans la chaîne S. Si la fin du fichier est atteinte, la fonction retourne le pointeur NULL.
Pour écrire dans un fichier une ligne de texte :

char fputs(char *S, FILE *f)

Cette fonction écrit la chaîne de caractères S dans le fichier de descripteur f, elle retourne le dernier caractère écrit.

Entrées/sorties formatés

15

On peut aussi lire et écrire des variables de types quelconques, en utilisant ***fprintf()*** et ***fscanf()*** qui permettent de réaliser le même travail que *printf()* et *scanf()* sur des fichiers ouverts en mode texte:

fprintf(FILE *f, char *format, argument) ;

fscanf(FILE *f, char *format, &argument) ;

Exemples

16

Que font les instructions suivantes ?

```
char chaîne[80]
FILE *f ;
f=fopen(" …. ", "r") ;
if( !f)
        printf(« impossible d'ouvrir le fichier ») ;
else
{               while( fgets(chaîne, sizeof(chaîne), f) !=NULL)
                        puts(chaîne) ;
                fclose(f) ;
}
```

Accès par enregistrement

17

Permet de lire ou écrire les objets de type structure. Le fichier doit être ouvert en mode **binaire**. Les données échangées ne sont pas traitées comme du texte.

int fread (void *bloc, int taille, int nb, FILE *f)
int fwrite (void *bloc, int taille, int nb, FILE *f)

Les paramètres sont décrits comme suit :

bloc : Adresse de l'espace mémoire à partir duquel on fait l'échange avec le fichier qui reçoit ou fournit l'enregistrement. Cet espace mémoire :

- reçoit les enregistrements lus (dans le cas de fread).
- fournit les données à écrire (dans le cas de fwrite).

Il faut que cet espace soit de taille suffisante pour supporter le transfert des données

taille : taille de l'enregistrement en nombre d'octets (sizeof(enregistrement))

nb : nombre d'enregistrements à lire ou à écrire

f : descripteur du fichier

Les 2 fonctions retournent le nombre d'enregistrement lus/écrits.

Exemple

18

```c
#include <stdio.h>
#include <stdlib.h>
#define NB 50
int main(int arv, char * arg[]) {
    FILE *in, *out;
    int tab1[NB], tab2[NB];
    int i;

    for (i = 0 ; i < NB; i++)
        tab1[i] = i;

    // écriture du tableau dans sortie.bin
    if ((out = fopen("./sortie.bin", "wb")) == NULL) {
        fprintf(stderr, "\nImpossible d'écrire");
        return(-1);
    }

    fwrite(tab1, NB * sizeof(int), 1, out);
    fclose(out);
```

Exemple

19

```c
    // lecture dans sortie.bin
    if ((in = fopen(./sortie.bin, "rb")) == NULL) {
        fprintf(stderr, "\nImpossible de lire");
        return(-1);
    }

    fread(tab2, NB * sizeof(int), 1, in);
    fclose(in);

    for (i = 0 ; i < NB; i++)
        printf("%d\t",tab2[i]);

    printf("\n");

    return(0);
}
```

Accès direct

20

Le langage C permet un accès directe au données d'un fichier.

- `int fseek (FILE * pf, long int offset, int origine)` : affecte l'indicateur de position associé à `pf`, par la position `offset + origine`.
  - `pf` : pointeur sur `FILE` identifiant le flux.
  - `offset` : nombre de bytes à partir de `origine`.
  - `origine` : position à partir de laquelle `offset` est ajouté. Peut-être spécifié par l'une des constantes suivantes,
    - SEEK_SET Début du fichier
    - SEEK_CUR Position courante du pointeur
    - SEEK_END Fin du fichier.
- `long int ftell ( FILE * pf )` : retourne la valeur actuelle de l'indicateur de position.

Exemple

21

```
#include <stdio.h>

int main (int arv, char * arg[]) {
  FILE * fp;
  long size;

  if ((fp = fopen ("monFichier.txt","rb"))==NULL) {
    fprintf(stderr,"\nImpossible d'ouvrir le fichier");
    return(-1);
  } else {
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fclose (fp);
    printf ("La taille de monFichier.txt: %ld bytes.\n",
            size);
  }
  return 0;
}
```

Exercices

22

Remarque

int feof(FILE *f) détecte la fin du fichier dans le flux f.

Exercice 1

Ecrire un programme C qui calcule et affiche le nombre d'occurrence d'un caractère saisie au clavier dans un fichier texte dont on saisie le nom.

Exercices

Solution

```
#include<stdio.h>
Void main()
{char c, nom_fich[20];
int nb;
File *f;
Printf("donnez le nom du fichier");
gets(nom_fich);
f=fopen(nom_fich,"r");
        If(f!=NULL)
                {
                printf("Taper le caractère");
                scanf("%",&c);
                Nb=0;
                while(!feof(f))
                if(fgetc(f)==c)
                Nb++;
        Printf("le caractère %c se  trouve %d fois dans le fichier %s", c,nb,nom_fich);
                }
fclose(f);
}
```

23

Exercices

Exercice 2

Ecrire un programme C qui crée un fichier binaire de nom reels.dat, puis enregistre N réels saisis au clavier dans ce fichier.

24

Exercices

Solution

```
#include<stdio.h>
void main()
{
float x; int N,I;
File *f;
f=fopen("reels.dat,"wb");
printf("Taper le nombre de réels a sauvgarder");
scanf("%",&N);
      for (i=1;i=N;i++)
      {printf("Taper un réel:");
      Scanf("%f",&x);
      fwrite(&x,sizeof(float),1,f);
      }
fclose(f);
}
```


Exercices

Exercice 3

Ecrire un programme C qui calcule et affiche la moyenne des nombres réels stockés dans le fichier reels.dat crée dans l'exercice précédent.

Exercices

Solution

```
#include<stdio.h>
void main()
{
float x,s=0;
File *f;
f=fopen("reels.dat,"rb");
While(!feof(f))
{fread(&x,sizeof(float),1,f);
s=s+x;
}
      printf("La somme des réels du fichier reels.dat est : %f",s);
fclose(f);
}
```


Exercices

Exercice 4

On définit des étudiants par un nom, un prénom et un code (deux étudiants différents ne peuvent pas avoir le même code). Ecrire en C les fonctions suivantes :

CreeFichier :
qui permet de saisir le nom d'un fichier, un entier N ainsi que les noms, prénoms et codes des N étudiants pour construire le fichier.

AfficheFichier :
qui liste le contenu d'un fichier dont le nom est donne en paramètre.

```
void CreeFichier  (char *nom_fichier)
{
ETUDIANT  e;
char nom-fichier[20];
int n;
FILE *fp;
gets(nom-fichier);
fp = fopen(nom-fichier, “wb”);
If (! fp) printf(“impossible de creer le fichier”);
else
{
  printf(“donner le nombre des etudiants”);
  scanf(“%d”,&n);
  for (i=0;i < n;i++)
        {
        scanf(“%d”,&e.code);
        scanf(“%s”,e.nom);
        scanf(“%s”,e.prenom);
        fprintf(fp,”%d\t%s\t%s”, e.code,e.nom,e.prenom);
        }
  fclose(fp);
 }
}
```

29

```
Typedef struct {
        int code;
        char nom[20];
        char prenom[20];
        }ETUDIANT;
```

```
void AfficheFichier  (char *nom_fichier)
{
ETUDIANT  e;
char nom-fichier[20];
int n;
FILE *fp;
gets(nom-fichier)
fp = fopen(nom-fichier, “rb”);
If (! fp) printf(“impossible d’ouvrir le fichier”);
else
{
while (!feof(fp))
        {
        if (fread(&e,sizeof(struct etudiant),1,fp)==1)
        {
        printf(“%d”, e.code);
        puts(e.nom);
        puts(e.prenom);
        }

fclose(fp);
 }
}
```

30

Institut National des sciences Appliquées et des Technologies

Algorithmique et Structure de Données 1
Niveau MPI

Année universitaire
2015-2016

Chapitre 9
Récursivité

Wided Miled
Aymen Sellaouti

Chapitre 9
Initiation à la récurssivité

Plan

Définition

- Une fonction est dite récursive si elle contient un appel à elle même.
- Il y a un appel à la fonction dans le corps de la fonction :

l'appel est dit récursif.

Définition

5

Exemple 1: Calcul du factoriel d'un nombre n.

Fact (n) = n * Fact (n – 1)
 	
Fact(0) = 1.
 	
Cette définition mathématique contient une récursion.
	int Fact (int n)
	{
	if (n == 0)
	return (1) ;
	else
	return (n * Fact(n-1));
	}
	
	cette écriture est naturelle et répond exactement à la définition mathématique

5

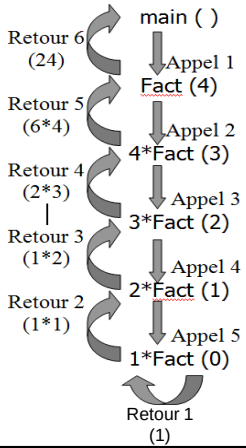
Exemple 1

6

- L'utilisation de la récursivité se rapproche du principe de la récurrence en mathématiques. Pour définir une fonction récursive, il faut :
- définir la valeur de la fonction pour un ensemble de cas simples sans utilisation d'appels récursifs;
- puis définir la valeur de la fonction dans le cas général en fonction des autres valeurs.
- Comment ça marche ?
- Calcul de Fact (4)

Exemple 1

7



Exemple 1

8

La fonction qui fait un appel récursif se met en attente du résultat de cet appel ; quand on arrive à un cas simple, il y a retour du résultat vers les fonctions en attente jusqu'à ce qu'on remonte à la fonction initiale qui a effectué le premier appel.

	Fonction	Valeur retournée
Pile d'exécution : LIFO Last In First Out	Fact (0)	1
	Fact (1)	1*1
	Fact (2)	1*2
	Fact (3)	2*3
	Fact (4)	4*6
	main ( )	On obtient : Fact(4)=24

Exemple 2

9

En mémoire, il y a une structure de pile d'exécution où les appels récursifs sont empilés dans l'ordre dans lequel ils arrivent pour se mettre en attente. Il sont dépilés par la suite dans l'ordre inverse quand il y a des retours de résultats successifs.

Exemple 2: Une suite U définie par :
U(n) = U(n-1) + U(n-2) avec U(0) = 1, U(1)=1 (appelée suite de Fibonacci)

```
int fibo ( int n )
{
  if (( n == 0 ) || ( n== 1))
    return (1);
  else
    return (fibo (n-1) + fibo (n-2));
}
```

Exercice 1

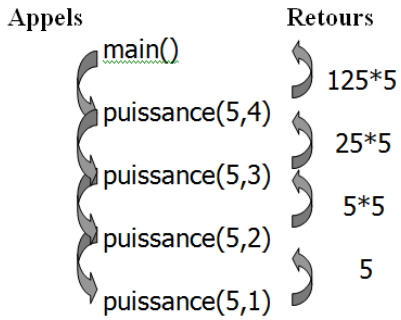
10

Exercice 1
Ecrire une fonction récursive *puissance* qui prend deux valeurs a et n et retourne a à la puissance n.

```
int puissance ( int a, int n )
{
        if ( n == 0 )
                return (1);
else if (n==1)
        return a;
  else
        return ( a* puissance(a,n-1));
}
```

Exercice 1

11



Exercice 2

12

Exercice 2
Ecrire une fonction récursive qui calcule le nombre de chiffres d'un entier n (longueur d'un nombre entier).

Exemple : n = 36543 ➔ 5 chiffres

L(36543) = L(3654) + 1
= L(365) + 1 +1
= L(36) + 1 +1 +1
= L(3) + 1 + 1 + 1 +1
= 1 + 1 + 1 +1+ 1

Exercice 2

13

```
int nbr_chiffres ( int n )

{
  if (n / 10 == 0 )
            return (1) ;
  else
            return( 1 +nbr_chiffres (n / 10 )) ;
}
```

Exercice 3

14

Exercice 3
Ecrire une fonction récursive qui calcule la somme des éléments d'un tableau d'entiersT.

```
int som_tab (int T[ ], int n)

{
  if ( n == 1 )
            return ( T[0] ) ;
  else
            return ( T[n-1] + som_tab( T, n-1 ));
}
```

Exercice 4

15

Exercice 4
Ecrire une fonction récursive qui calcule la somme des éléments d'un tableau d'entiers T à partir d'une position i.

```
int som_tab2 (int T[ ], int n, int i)
{
  if ( i == n-1 )
        return ( T[i] ) ;
  else
        return ( T[i] + som_tab2( T, n, i+1 ));
}
```

Exercice 5

16

Exercice 5
Ecrire une fonction récursive qui recherche si un entier ***x*** existe dans un tableau d'entiers T. La fonction retourne 1 si oui et 0 sinon.

```
int recherche ( int T[ ], int n, int x, int i )
{
  if (i==n)
        return 0;
  else if (T[i] == x )
        return (1) ;
  else
        return(recherche (T, n, x, i + 1) ;
}
```