

Sommaire

Chapitre 1

Chapitre 2

Chapitre 3

Chapitre 4

Chapitre 5

TD1

TD1_Correction

TD2

TD3

TD4

Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Algorithmique et Programmation C

Filières: L1-Math Appliqué et L1-TIC

Chapitre 1: Introduction au langage C

Réalisé par:
Dr. Sakka Rouis Taoufik

1

Chapitre 1: Introduction au langage C

I. Historique

- Le langage C a été conçu en 1972 par Dennis Richie et Ken Thompson chercheurs aux Bell Labs
- Il a été conçu à l'origine pour l'écriture du système d'exploitation UNIX (90-95% du noyau est écrit en C) et s'est vite imposé comme le langage de programmation sous UNIX.
- Il a été normalisé en 1989 par le comité X3J11 de l'American National Standards Institute (ANSI)

2

Chapitre 1: Introduction au langage C

II. Caractéristiques

- Le langage C est un langage **évolué, modulaire et structuré**, assez proche du langage machine destiné des applications de contrôle de processus (gestion d'entrées/sorties, application temps réel...).
- Le langage C possède un peu d'instructions. Il fait par contre appel à des bibliothèques.

Exemple :

math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

3

Chapitre 1: Introduction au langage C

II. Caractéristiques

Déclaratif : normalement, tout objet C doit être déclaré avant d'être utilisé. S'il ne l'est pas, il est considéré comme étant du type entier.

Format libre : la mise en page des divers composants d'un programme est totalement libre. Cette possibilité doit être exploitée pour rendre les programmes lisibles.

Transportable : les entrées/sorties sont réunies dans une librairie externe au langage

Souple et permissivité : peu de vérifications et d'interdits, hormis la syntaxe. Il est important de remarquer que la tentation est grande d'utiliser cette caractéristique pour écrire le plus souvent des atrocités.

4

Chapitre 1: Introduction au langage C

III. Compilation

- C est un langage compilé \neq langage interprété
- **Compiler un programme** c'est traduire un texte (code source) d'un langage de haut niveau (langage C) en code de bas niveau (code machine), de manière à ce que le système d'exploitation puisse, au besoin, déclencher l'exécution de ce programme. Dans un langage compilé, l'étape de traduction a lieu une fois pour toutes.
- **Interpréter un programme**, c'est faire en même temps la traduction et l'exécution du texte d'un langage de haut niveau (un script). → Dans un langage interprété, l'étape de traduction a lieu à chaque exécution.

5

Chapitre 1: Introduction au langage C

III. Compilation

- **Avantage**: compilation indépendante de chaque module

Si le code source d'un module est modifié,



Seuls ce module et ceux qui l'utilisent doivent être recompilés

- **Étapes de la compilation** :

1. Analyse lexicale
2. Analyse syntaxique
3. Analyse sémantique
4. Génération du code
5. Édition de liens

6

Chapitre 1: Introduction au langage C

III. Compilation

1) Analyse lexicale

Identifie les lexèmes (unités lexicales du langage). Les espaces sont inutiles ($3*x+1$ ou $3 * x + 1$), sauf comme séparateurs (`int x, intx`).

Exemple d'erreur lexicale :

code source `int x = @;`

compilation error: stray '@' in program

Erreur détectée uniquement au moment de l'analyse sémantique :

code source `intx = 0;`

compilation error: 'intx' undeclared (first use in this function)

7

Chapitre 1: Introduction au langage C

III. Compilation

2) Analyse syntaxique

Trouve la structure syntaxique, (arbre syntaxique), et teste l'appartenance au langage.

Exemple : Dans l'expression `x = 3 * x + 1`, est-ce que la sous-suite `x + 1` correspond à une structure syntaxique ?

Exemple d'erreur syntaxique :

code source Un `else` sans `if` le précédant immédiatement (point-virgule mal placé ?)

compilation error: expected expression before 'else'

8

Chapitre 1: Introduction au langage C

III. Compilation

3) Analyse sémantique

Trouver le sens des différentes actions voulues par le programmeur.

- Quels sont les objets manipulés par le programme,
- Quelles sont les propriétés de ces objets,
- Quelles sont les actions du programme sur ces objets.

Beaucoup d'erreurs peuvent apparaître durant cette phase : identificateur utilisé mais non déclaré (la réciproque génère un warning avec l'option `-Wall`), opération n'ayant aucun sens, etc.

code source variable `x` utilisée mais non déclarée

compilation error: 'x' undeclared (first use in this function)

9

Chapitre 1: Introduction au langage C

III. Compilation

4) Génération de code

- La génération du code se décompose en 3 phases:

➤ **Le traitement par le pré-processeur:** réalise des transformations d'ordre purement textuel (inclusion d'autres fichiers sources, etc.) (*.i)

➤ **la compilation:** traduit le fichier généré par le pré-processeur en langage assembleur (*.s)

➤ **l'assemblage:** transforme le code assembleur en fichier binaire appelé fichier *objet* (*.o)

10

Chapitre 1: Introduction au langage C

III. Compilation

5) Édition des liens

Le code objet des fonctions externes (bibliothèques) est ajouté à l'exécutable.

Le point d'entrée dans le programme est choisi (`main`). Insertion de données de débogage (option `-g`).

code source Oublie de `stdio.h` et utilisation de `printf`.

compilation warning: incompatible implicit declaration of built-in function 'printf'

code source Pas de fonction principale (`main`)

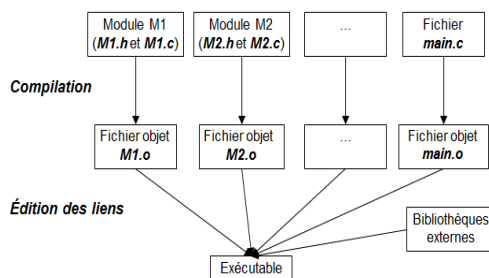
compilation Undefined symbols: "_main", ...

11

Chapitre 1: Introduction au langage C

III. Compilation

5) Édition des liens



12

Chapitre 1: Introduction au langage C

IV. Structure d'un programme C

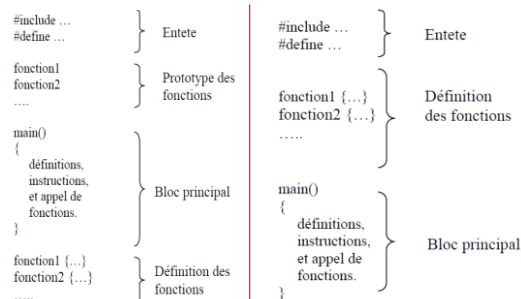
Pour simplifier les choses, dans ce cours, on considère qu'un programme C est écrit dans un seul fichier composé de :

- **Un entête (header)** constitué de méta-instructions ou **directives** destinées au préprocesseur. Ces directives permettent d'effectuer des manipulations sur le texte du programme source avant la compilation : Une directive du préprocesseur est une ligne de programme source commençant par le caractère dièse (#).
- **Un bloc principal appelé main ()**. Il contient des instructions élémentaires ainsi que les appels aux modules (fonctions) considérés par le compilateur comme des entités autonomes. Chaque appel à une fonction peut être considéré comme une instruction.
- **Le corps des fonctions** placées avant ou après la fonction main () dans un ordre quelconque, les unes après les autres.

13

Chapitre 1: Introduction au langage C

IV. Structure d'un programme C



14

Chapitre 1: Introduction au langage C

IV. Structure d'un programme C

Remarques :

Il est préférable de **commenter** les instructions difficiles.

- Ceci permet d'expliquer la signification de ces dernières au lecteur même s'il ne connaît pas le langage de programmation.
- Les commentaires en C peuvent être placés à n'importe quel endroit du programme.
- Ils commencent par /* et se terminent par */.
- Le langage C **distingue les minuscules, des majuscules**. Les mots réservés du langage C doivent être écrits en minuscules.

15

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

1) Identificateurs et mots-clés

- Un identificateur est le nom donné à une variable, une constante, une fonction, etc.
- Ce nom est composé de lettres non accentuées, de chiffres et du caractère dans un ordre quelconque sauf pour le premier caractère qui ne peut pas être un chiffre.
- Rq. Certains mots ne peuvent pas être utilisés comme **identificateurs** car ils sont réservés.

16

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

1) Identificateurs et mots-clés

- Les mots clés ne vous diront rien pour le moment, tout ce qu'il faut savoir c'est que vous ne devez pas utiliser ces mots pour nommer vos variables et fonctions :

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

17

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

2) Variables

- Une variable est un espace mémoire que le programme réserve lors de son exécution au moment de sa déclaration.
- C'est une donnée dont la valeur est modifiable.
- Dans tous les langages, une définition de variable a les rôles suivants :
 1. définir le domaine de valeur de cette variable (taille en mémoire et représentation machine) ;
 2. définir les opérations possibles sur cette variable ;
 3. définir le domaine de visibilité de cette variable ;
 4. permettre à l'environnement d'exécution du programme d'associer le nom de la variable à une adresse mémoire ;

18

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

2) Variables

a) Déclaration de variable

Lorsqu'on utilise un langage de programmation **déclaratif**, il faut toujours donner le type des variables avant de les utiliser. C'est ce qu'on appelle la déclaration.

Syntaxe :

```
nom_type nomVariable ;
nom_type nomVariable_1, nomVariable_2, nomVariable_N ;
```

Exemple :

```
int A ; /*déclaration d'une variable de type entier*/
float X, Y ; /*déclaration de deux variables de type réel*/
```

19

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

2) Variables

a) Affectation de variable

Le symbole \leftarrow que nous utilisons en algorithmique correspond au signe `=` du langage C.

Syntaxe :

```
nom_variable = valeur;
```

Exemple :

```
int x, y ;
x=8 ;
y=x ;
```

Une variable peut être initialisée lors de sa déclaration.

Exemple :

```
int a=3 ;
float c= 3.2;
```

20

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

3) Constantes

Une constante est un identificateur qui contient une valeur qui **ne sera jamais modifiée** au cours du programme.

Par exemple, la constante π peut être fixée à 3.14.

Les constantes peuvent être :

- des nombres entiers (12),
- des nombres réels (20.6),
- un caractère ('a')
- une chaîne de caractères ("bonjour").

Par conséquent on les écrit souvent en **majuscules** pour les différencier des autres identificateurs.

21

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

3) Constantes

Déclaration de constante

Il existe deux façons pour déclarer une constante :

Syntaxe 1 :

```
#define nom_constant valeur_constant
```

- Cette déclaration est effectuée dans la partie pré-processeurs (après l'inclusion des bibliothèques)
- Elle permet la déclaration d'une constante qui sera **visible par tout le programme** et sa **durée de vie** est celle de toute l'application.

Exemples :

```
#define PI 3.14 /* Rq sans ; */
#define TVA 0.16
```

22

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

3) Constantes

Syntaxe 2 :

```
const type nom_constant = valeur_constant ;
```

- Cette déclaration est effectuée dans le bloc
- Elle permet la déclaration d'une constante qui **sera visible seulement dans le bloc** et elle sera détruite en sortant du bloc.

Exemple :

```
const float Pi = 3.14 ; /* Rq avec ; */
```

23

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

- C est un langage **typé** : toute variable, constante ou fonction est dotée d'un type précis.
- Les types **de bases** de C sont au nombre de six :
 - void
 - int
 - char
 - float
 - double
 - long double

24

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

void

- C'est le type vide. Il a été introduit par la norme ANSI.
- Il est surtout utilisé pour préciser les fonctions sans argument ou sans retour.

25

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

int

- C'est le type entier. Ce type se décline avec des qualificatifs pour préciser sa taille (**long** ou **short**), et le fait qu'il soit uniquement positif (**unsigned**) ou positif et négatif (**signed**). Le qualificatif **signed** est appliqué par défaut, ainsi il n'y a pas de différence entre une variable de type **int** et une variable de type **signed int**.

Exemples

```
long int a ; /* long a ; */
short int b ; /* short b ; */
unsigned int c ;
signed int d ; /* int d ; */
```

26

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

char

- Ce type est très proche de l'octet. Il représente un entier sur huit bits.
- Sa valeur peut évoluer entre -128 et +127.
- Il est le support des caractères au sens commun du terme.
- Ces caractères sont représentés par la table ASCII.
- Comme le type **int** le type **char** peut être qualifié de manière à être signé ou non.

Exemple

```
char c1; signed char c2; unsigned char c3;
```

Rq. La norme ANSI introduit un type permettant de supporter des alphabets comprenant plus de 255 signes, ce type est appelé **wchar_t**. Il est défini dans le fichier **<stddef.h>**.

27

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

float

- Ce type sert pour les calculs avec des parties décimales.

double

- C'est un type qui permet de représenter des valeurs ayant une partie décimale avec une plus grande précision que le type **float**. Comme nous le verrons dans l'expression des constantes et dans les calculs, ce type est le plus courant pour représenter des valeurs avec parties décimales.

long double

- Ce type est récent, il permet de représenter des nombres avec parties décimales qui nécessitent une très grande précision.

28

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

Type	PDP 11	Intel 486	Sparc	Pentium	Alpha
Année	1970	1989	1993	1993	1994
char	8 bits	8bits	8bits	8bits	8bits
short	16 bits	16 bits	16 bits	16 bits	16 bits
int	16 bits	16 bits	32 bits	32 bits	32 bits
long	32 bits	32 bits	32 bits	32 bits	64 bits
float	32 bits	32 bits	32 bits	32 bits	32 bits
double	64 bits	64 bits	64 bits	64 bits	64 bits
long double	64 bits	64 bits	64 bits	64 bits	128 bits

29

Chapitre 1: Introduction au langage C

V. Les composantes élémentaires d'un programme C

5) Déclaration de nouveaux types

Le langage C permet de définir de nouveaux types de variables en utilisant le mot clé **typedef** selon la syntaxe:

```
typedef type_de_base nom_nouveau_type ;
```

Exemple

```
#define PI 3.14
typedef float reel ;
void main() {
    reel perimetre, rayon =8,7;
    perimetre = 2* PI* rayon;
    ...
}
```

30

Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Algorithmique et Programmation C

Filières: L1-Math Appliqué et L1-TIC

Chapitre 2: Les opérateurs de base & Les fonctions d'E/S standard

Réalisé par:

Dr. Sakka Rouis Taoufik

1

Chapitre 2 : Les opérateurs de base & Fonctions E/S

I. Introduction

Le langage C apporte quelques notions innovantes en matière d'opérateurs et de fonctions d'E/S.

En particulier, le langage C considère l'affectation comme un opérateur normal alors que les langages qui l'ont précédé (par exemple FORTRAN, ADA) la considèrent comme une opération privilégiée.

Dans ce chapitre on se limite aux principaux opérateurs du langage C et aux fonctions printf () et scanf () qui représentent les fonctions standard de la bibliothèque d'E/S «stdio.h».

2

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

- Les opérateurs sont les éléments du langage qui permettent de faire du calcul ou de définir des relations.
- Ils servent à combiner des variables et des constantes pour réaliser des expressions.

3

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

1) Les opérateurs arithmétiques

Le langage C connaît :

- deux opérateurs arithmétiques unaires : + et –
- cinq opérateurs binaires :
 - addition (+)
 - soustraction (-)
 - division (/)
 - multiplication (*)
 - modulo (%) (Reste de la division entière)

4

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

1) Les opérateurs arithmétiques

Exemple:

```
...
int a, b =125, c;
float x, y=2.75;
...
a= -b;
c=3*a+b%2;
...
```

5

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

2) Les opérateurs d'affectation

Le langage C permet de construire des opérateurs binaires d'affectation à partir des opérateurs binaires arithmétiques, des opérateurs de masquage et des opérateurs de décalage, en les faisant suivre d'un égal (=).

L'utilisation de ces opérateurs d'affectation composés permet la simplification des expressions.

opérateur	utilisation	équivalent
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

6

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Les opérateurs `--` et `++` permettent de décrémentation et d'incrémenter des variables de type entier.

Dans une première approche, nous pouvons considérer que :

`var--` ; est équivalent à `var = var - 1` ;

→ en partant d'une variable `var` ayant une valeur de 8 que cette variable contient la valeur 7 une fois l'expression calculée à l'exécution ;

`var++` ; est équivalent à `var = var + 1` ;

→ en partant de `var` ayant une valeur de 8 que cette variable contient la valeur 9 à la fin de l'exécution de l'expression.

7

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Il est possible d'utiliser les opérateurs unaires d'incrémentation et de décrémentation derrière la variable (postfixé) ou devant celle-ci (préfixé).

Ce qui permet de post-incrémenter, de pré-incrémenter, de post-décrémenter ou de pré-décrémenter.

- Lorsque l'opérateur est préfixé, l'opération est appliquée avant que la valeur correspondant à l'opération ne soit calculée.
- Dans le cas où l'opération est post-fixée, la valeur de la variable avant l'opération est utilisée pour les autres calculs et ensuite l'opération est appliquée.

8

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Exemple 1 :

```
int i=0, j=0 ;
j = ++i ;
```

→ C'est une pré-incrémentation de la variable i. Cela signifie incrémenter i de 1 puis mettre la valeur de i dans j. À la fin de cette opération i vaut 1 et j vaut 1.

→ Donc on peut conclure que **j=++i** est équivalente aux deux instructions suivantes :

```
i = i+1 ;
j = i;
```

9

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Exemples 2 :

```
int i=0, j=0;
j = i++ ;
```

→ Cette instruction signifie mettre la valeur de i dans j puis incrémenter i de 1. En partant des mêmes valeurs, i valant 0 et j valant 0, à la fin de cette instruction i vaut 1 et j vaut 0.

→ Donc on peut conclure que **j=i++** est équivalente aux deux instructions suivantes :

```
j=i ;
i = i+1 ;
```

10

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

4) Les opérateurs logiques (booléens)

Ce type d'opérateurs permet de vérifier si plusieurs conditions sont vraies:

Les deux opérateurs de tests (**&&** et **||**) sont appelés respectivement le “**et logique**” et le “**ou logique**”.

- Le “et logique” permet de décrire qu’une condition constituée de deux parties est satisfaite si et seulement si **les deux parties sont satisfaites**.
- Le “ou logique” permet de décrire qu’une condition constituée de deux parties est satisfaite si et seulement si **l’une des deux parties est satisfaite**.

11

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

4) Les opérateurs logiques (booléens)

- La négation logique (**!**) sert à inverser une condition en la faisant passer de vrai à faux et réciproquement.
- En langage C, une expression est fausse si la valeur qu’elle retourne est égale à **0**, elle est vraie sinon. De plus, la norme spécifie que **!0 vaut 1**.

Opérateur	Dénomination	Effet	Syntaxe
	OU logique	Vérifie qu’une des conditions est réalisée	((condition1) condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&condition2))
!	NON logique	Inverse l’état d’une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

12

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

5) Les opérateurs de manipulation de bits

Les opérateurs de masquage et de décalages sont utilisés pour manipuler les bits des variables entières.

&	:	et logique	~	:	complément à 1
/	:	ou inclusif	<<	:	décalage à gauche
^	:	ou exclusif	>>	:	décalage à droite

Exemple

```
int a, b, c ;
a= 0x6db7 ; /* |-0-|-0-|0110 1101|1011 0111| */
b= 0xA726 ; /* |-0-|-0-|1010 0111|0010 0110| */
c= a&b;    /* |-0-|-0-|0010 0101|0010 0110| (0x2526) */
c=a|b;    /* |-0-|-0-|1110 1111|1011 0111| (0xefb7) */
c= a^b;    /* |-0-|-0-|1100 1010|1001 0001| (0xca91) */
```

13

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

6) Les opérateurs de comparaison

Les opérateurs de comparaison servent à réaliser des tests entre les valeurs de deux expressions.

Comme nous le verrons dans le chapitre 7 ces opérateurs sont surtout utilisés à l'intérieur des instructions de contrôle de flot (tests).

Les opérateurs de relation algébrique sont au nombre de six :
(**==** **<** **<=** **>** **>=** **!=**).

14

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

6) Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
<code>==</code> A ne pas confondre avec le signe d'affectation (<code>=</code>)!!	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	<code>x==3</code>	Retourne 1 si X est égal à 3, sinon 0
<code><</code>	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	<code>x<3</code>	Retourne 1 si X est inférieur à 3, sinon 0
<code><=</code>	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	<code>x<=3</code>	Retourne 1 si X est inférieur ou égal à 3, sinon 0
<code>></code>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	<code>x>3</code>	Retourne 1 si X est supérieur à 3, sinon 0
<code>>=</code>	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	<code>x>=3</code>	Retourne 1 si X est supérieur ou égal à 3, sinon 0
<code>!=</code>	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	<code>x!=3</code>	Retourne 1 si X est différent de 3, sinon 0

15

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

7) L'opérateur d'adressage

L'opérateur d'adressage ou de référencement noté (`&`) permet de retourner l'adresse en mémoire (référence) de la variable dont le nom le suit.

Exemple

`&var` donne l'adresse en mémoire de la variable var.

16

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

8) L'opérateur de taille

L'opérateur **sizeof** donne la taille en octets de l'opérande selon la syntaxe:

sizeof expression

ou

sizeof (expression)

Exemple:

```
int a, b, c;
double x;
a = sizeof (long) ;    /* a ← 4 */
b = 3 * sizeof x ;     /* b ← 24 */
c = sizeof "informatique"; /* c ← 13 */
```

17

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs

9) L'opérateur conditionnel (? :)

Cet opérateur est un opérateur **ternaire** qui met en jeu trois expressions de la façon suivante

expr1 ? expr2 : expr3

Il permet de construire une opération de test avec retour de valeur.

Si expr1 fournit une valeur non nulle,
alors la valeur de l'expr2 est fournie comme résultat
si non la valeur de l'expr3 est fournie comme résultat.

18

Chapitre 2 : Les opérateurs de base & Fonctions E/S

II. Les opérateurs


10) L'opérateur conditionnel (? :)

Exemple 1:

$X = (a == b) ? c : d$

Cette expression retourne la valeur de la variable c si la valeur de la variable a est égale à celle de la variable b. Dans le cas contraire, l'expression retourne la valeur de la variable d.

Exemple 2 :

<pre>if (A>B) MAX=A; else MAX=B;</pre>	}		$MAX = (A > B) ? A : B;$
---	---	---	--------------------------

19

Chapitre 2 : Les opérateurs de base & Fonctions E/S

III. Les Fonctions d'E/S standard

Les entrée-sorties en langage C ne sont pas prises en charge directement par le compilateur mais elles sont réalisées à travers des fonctions de bibliothèque.

Le compilateur ne peut pas faire de contrôle de cohérence dans les arguments passés à ces fonctions car ils sont de type variable.

Ceci explique l'attention toute particulière avec laquelle ces opérations doivent être programmées.

La bibliothèque standard d'E/S du langage C est la bibliothèque **<stdio.h>**

20

Chapitre 2 : Les opérateurs de base & Fonctions E/S

III. Les Fonctions d'E/S standard

1) La fonction d'affichage : printf ()

Syntaxe :

printf ("String Format" [, argument1... argumentN]);

String format : chaîne de caractères avec des caractères de formatage si nécessaire.

Un argument : peut être une valeur ou une variable ou une expression ou un appel de fonction.

Le nombre des arguments après le String Format dépend du nombre de variables pour lesquels vous voulez afficher les valeurs.

Exemple

21

Chapitre 2 : Les opérateurs de base & Fonctions E/S

III. Les Fonctions d'E/S standard

1) La fonction d'affichage : printf ()

format	signification
%c	caractère
%s	chaîne de caractères
%d	nombre entier en décimal
%f	nombre réel sous la forme [-]mmm.nnnnnn
%e	nombre réel sous la forme mantisse/exposant [-]m.nnnnnne[+ -]xx
%g	nombre réel sous la forme la plus courte entre les types %e et %f
%o	nombre entier en octal
%p	pointeur ou adresse de la valeur numérique
%u	nombre entier non signé en décimal
%x	nombre entier en hexadécimal
%X	nombre entier en hexadécimal ; lettres affichées en majuscules

22

Chapitre 2 : Les opérateurs de base & Fonctions E/S

III. Les Fonctions d'E/S standard

1) La fonction d'affichage : printf ()

Exemple:

```
int x=8 ;
float y=5.6 ;
char c= 'b' ;

printf(" la valeur de la variable c = %c", c) ;
printf(" l'adresse de la variable c = %x", &c) ;
printf("x= %d et y=%f", x, y) ;
```

23

Chapitre 2 : Les opérateurs de base & Fonctions E/S

III. Les Fonctions d'E/S standard

1) La fonction de saisie : scanf ()

C'est une fonction implémentée dans la bibliothèque standard **stdio.h**, elle permet la saisie des variables : elle modifie leurs contenus.

Syntaxe :

```
scanf ("Format1 [...Formatn]", &argument1[, ... &argumentN]);
```

Format doit être un format du tableau précédent (%d, %f...)

Argument ne peut être qu'une variable dont la valeur va être saisie par l'utilisateur.

L'utilisation de l'opérateur & est obligatoire sauf pour la saisie des variables de type pointeur.

24

Chapitre 2 : Les opérateurs de base & Fonctions E/S

III. Les Fonctions d'E/S standard

1) La fonction de saisie : scanf ()

Exemple 1:

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

Exemple 2:

```
#include <stdio.h>
void main()
{
    int i;
    printf("entrez un entier sous forme hexadécimale i =");
    scanf("%x", &i);
    printf("i = %d\n", i);
}
```

/*Si la valeur 1a est saisie, alors le programme affiche i = 26*/

25

Chapitre 2 : Les opérateurs de base & Fonctions E/S

IV. Exercices d'application

Exercice 1:

Ecrire un programme qui permute et affiche les valeurs de trois variables A, B, C de type entier qui sont entrées au clavier :

A ==> B , B ==> C , C ==> A

Exercice 2:

Ecrire un programme qui affiche la résistance équivalente à trois résistances R1, R2, R3 (type **double**),

- si les résistances sont branchées en série:

$$R_{\text{ser}} = R1 + R2 + R3$$

- si les résistances sont branchées en parallèle:

$$R_{\text{par}} = (R1.R2.R3)/(R1.R2 + R1.R3 + R3.R2)$$

26

Chapitre 2 : Les opérateurs de base & Fonctions E/S

IV. Exercices d'application

Exercice 3:

Définir deux entiers i et j initialisés avec les valeurs 10 et 3 respectivement.

Faire écrire les résultats de :

$i + j$ puis $i - j$ puis $i * j$ puis i / j puis $i \% j$

Affecter les valeurs 0x0FF et 0xF0F respectivement à i et j.

Faire écrire en hexadécimal les résultats de :

$i \& j$ puis $i | j$ puis $i \wedge j$ puis $i \ll 2$ puis $j \gg 2$

Exercice 4:

En utilisant 4 entiers i, j, k et l, avec k initialisé à 12 et l à 8, écrire le programme qui :

- lit les valeurs de i et j
- écrit la valeur de k si i est nul ;
- écrit la valeur de i + l si i est non nul et j est nul
- écrit la valeur de i + j dans les autres cas.

27

Chapitre 2 : Les opérateurs de base & Fonctions E/S

IV. Exercices d'application

Solution de l'exercice 1:

```
#include <stdio.h>
void main()
{
    int A, B, C, AIDE;
    printf("Introduisez trois nombres (A, B, C) : ");
    scanf("%i %i %i", &A, &B, &C);
    /* Affichage à l'aide de tabulations */
    printf("A = %i\tB = %i\tC = %i\n", A, B, C);
    AIDE=A; A=C; C=B; B=AIDE;
    printf("A = %i\tB = %i\tC = %i\n", A, B, C);
    return 0;
}
```

28

Chapitre 2 : Les opérateurs de base & Fonctions E/S

IV. Exercices d'application

Solution de l'exercice 2:

```
#include <stdio.h>
void main()
{
    double R1, R2, R3, RRES; printf("Introduisez les
    valeurs pour R1, R2 et R3 : ");
    scanf("%lf %lf %lf", &R1, &R2, &R3);
    RRES=R1+R2+R3;
    printf("Resistance resultante seriele : %f\n",
    RRES);
    RRES=(R1*R2*R3)/(R1*R2+R1*R3+R2*R3);
    printf("Resistance resultante parallele : %f\n",
    RRES); return 0;
}
```

29

Chapitre 2 : Les opérateurs de base & Fonctions E/S

IV. Exercices d'application

Solution de l'exercice 4:

```
#include <stdio.h>
void main ( )
{
    /* definition de 4 entiers */
    int i, j, k = 12, l = 8;
    /* lecture des valeurs de i et de j */
    printf("\n Entrer la valeur de i :");
    scanf("%d", &i);
    printf("\n Entrer la valeur de j :");
    scanf("%d", &j);

    /* ecriture du resultat selon les valeurs de i et de j */
    printf("\n resultat : %d\n", (!i ? k : (!j ? i + l : i + j)));
}
```

30

Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Algorithmique et Programmation C

Filières: L1-Math Appliqué et L1-TIC

Chapitre 3: Les instructions de contrôle

Réalisé par:

Dr. Sakka Rouis Taoufik

1

Chapitre 3 : Les instructions de contrôle

I. Introduction

Les instructions de contrôle servent à contrôler le déroulement de l'enchaînement des instructions à l'intérieur d'un programme.

Ces instructions peuvent être des instructions:

- ❖ **conditionnelles** : permettent de réaliser des tests, et suivant le résultat de ces tests, d'exécuter des parties de code différentes.
 - ➔ C distingue deux types de structures conditionnelles:
 - la structure simple **if**
 - la structure à choix multiples **switch**.
- ❖ **itératives**: sont commandées par trois types de boucles : le **for**, le **while** et le **do while**

2

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

1) L'instruction de teste : if

L'opérateur de test du langage C se présente généralement sous la forme suivante :

Syntaxe :

```
if ( expression ) {
    ..... ;      /*bloc d'instructions*/
}
else {
    ..... ;      /*bloc d'instructions*/
}
```

Dans le cas où aucun traitement n'est évoqué, si l'expression logique est fausse, la structure conditionnelle devient :

```
if ( expression ) {
    ..... ;      /*bloc d'instructions*
}
```

3

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

1) L'instruction de teste : if

Les { } ne sont pas nécessaire lorsque les blocs ne comporte qu'une seule instruction.

L'expression n'est pas forcément un test qui retourne la valeur 0 ou +1.

L'expression peut être **un calcul ou une affectation**, car, comme nous l'avons déjà dit dans le chapitre 2, il n'y a pas de type booléen.

Une expression est **vraie** si elle ramène un résultat **non nul** ; elle est **fausse** si le résultat **est nul**.

4

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

1) L'instruction de teste : if

Exercice 1:

Réaliser en C un algorithme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe du produit de A et B sans faire la multiplication.

Exercice 2:

Réaliser en C un algorithme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe de la somme de A et B sans faire l'addition.

Exercice 3:

Réaliser en C un algorithme qui lit trois valeurs entières (A, B et C) au clavier. Trier les valeurs A, B et C par échanges successifs de manière à obtenir ($A \leq B \leq C$) puis afficher les trois valeurs.

5

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

2) L'instruction de sélection multiple : switch

Syntaxe :

```
switch (expression) {
    case constante1 :
        liste d'instructions1;
        break;
    case constante2 :
        liste d'instructions2;
        break;
    ...
    default :
        liste d'instructionsN;
        break;
}
```

6

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

2) L'instruction de sélection multiple : switch

L'instruction **switch** permet d'éviter les imbrications d'instructions **if**

L'instruction **switch** prend la valeur de la variable « expression » et compare à chacune des étiquettes **case**, dès qu'elle trouve celle qui correspond, les instructions qui suivent sont exécutées:

- soit jusqu'à la rencontre d'une instruction **break**,
- soit jusqu'à la fin du corps de l'instruction **switch**.

7

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

2) L'instruction de sélection multiple : switch

Exemple 1 :

On suppose que «choix» est une variable de type caractère, une instruction **switch** typique est la suivante :

```
switch (choix) {
    case 'R' : printf ("Rouge") ; break ;
    case 'B' : printf ("Bleu") ; break ;
    case 'J' : printf ("Jaune") ; break ;
}
```

8

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

2) L'instruction de sélection multiple : switch

Exemple 2 :

On suppose que «jour» est une variable de type entier, une instruction **switch** typique est la suivante :

```
switch (jour) {
    case 0 : case 1: case 2: case 3 : case 4:
        printf ("Au travail !") ; break ;
    case 5 : printf ("Aujourd'hui est samedi") ; break ;
    case 6 : printf ("Aujourd'hui est dimanche") ; break ;
    default: printf ("erreur") ;
}
```

9

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

2) L'instruction de sélection multiple : switch

Remarques

Les instructions qui suivent la condition **default** sont exécutées lorsqu'aucune constante des **case** n'est égale à la valeur retournée par l'expression.

Le dernier **break** est facultatif. Il vaut mieux le laisser pour la cohérence de l'écriture, et pour ne pas avoir de surprise lorsqu'un **case** est ajouté.

Plusieurs valeurs de case peuvent aboutir sur les mêmes instructions.

10

Chapitre 3 : Les instructions de contrôle

II. Instructions conditionnelles

2) L'instruction de sélection multiple : switch

Exercice 1:

Écrire un programme C qui lit une date sous la forme N° du jour, N° du mois et l'année. Il affiche ensuite la date avec le nom du mois.

Exercice 2:

Réaliser en C un algorithme qui permet de saisir un numéro de couleur de l'arc-en-ciel et d'afficher la couleur correspondante :

1: rouge, 2 : orangé, 3 : jaune, 4 : vert, 5 : bleu, 6 : indigo et 7 : violet.

11

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

Les instructions itératives sont commandées par trois types de boucles :

- le for
- le while
- le do while

12

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

1) La boucle for

Syntaxe :

```
for ( exp1 ; exp2 ; exp3 ) {
    /*bloc d'instructions*/
}
```

Remarques :

- Le for s'utilise avec trois expressions, séparées par des points virgules, **qui peuvent être vides.**
- Les { } ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

13

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

1) La boucle for

1. l'**expression expr1** est réalisée une seule fois lors de l'entrée dans la boucle, nous l'appellerons expression d'initialisation ;
2. l'**expression expr2** est la condition d'exécution de l'instruction. Elle est testée à chaque itération, y compris la première. Si l'expression expr2 prend la valeur vraie l'instruction contrôlée par le for est exécutée, sinon la boucle se termine ;
3. l'**expression expr3** contrôle l'avancement de la boucle. Elle permet de manière générale de calculer la prochaine valeur avec laquelle la condition de passage va être ré-testée, elle est exécutée après l'instruction à chaque itération avant le nouveau test de passage.

14

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

1) La boucle for

Exemples:

```
for (i = 0 ; i < 10 ; i++) {
    ...
}
```

```
i=0 ;
for ( ; i < 10 ; ) {
    ...
    ...
    i++;
}
```

```
for (i = 0 , j = 10 ; i < j ; i++ , j--) {
    ....
}
```

```
i=0 ;
for ( j=10; i < j ; i++ ) {
    ...
    j -- ;
}
```

15

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

1) La boucle for

Exercice 1:

Un nombre réel X et un nombre entier N étant donnés, proposer un programme C qui fait calculer X^N . Étudier tous les cas possibles (N positive ou négative).

Exercice 2:

Un entier naturel de trois chiffres est dit cubique s'il est égal à la somme des cubes de ses trois chiffres.

Exemple :

153 est cubique car $153 = 1^3 + 5^3 + 3^3$

Réaliser en C un algorithme qui cherche et affiche tous les entiers cubiques de trois chiffres.

16

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

2) La boucle while

Le **while** répète le bloc d'instructions tant que la valeur de l'expression s'interprète comme vraie (**différente de zéro**).

Si expression **est nulle** le bloc d'instructions **ne sera jamais exécuté**.

Syntaxe :

```
while (expression) {
    .....;
    .....; /*bloc d'instructions*/
    .....;
}
```

17

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

2) La boucle while

Exemple :

```
int i=1;
while (i < 10) {
    printf ("i = %d \n ", i) ;
    i++;
}
```

Remarques :

- Les { } ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.
- Le traitement s'exécute 0 ou n fois !

18

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

2) La boucle while

Exercice 1:

Réaliser en C un algorithme qui permet de déterminer la somme des chiffres d'un nombre entier positif donné.

Exemple :

pour N= 25418, on aura $2+5+4+1+8 = 20$

19

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

3) La boucle do while

- A l'inverse du **while**, le **do while** place son test en fin d'exécution, ceci assure que l'instruction est réalisée **au moins une fois**.
- Il ressemble au **REPEAT UNTIL** du langage **PASCAL**.

Syntaxe :

```
do {
    ..... ;
    ..... ; /*bloc d'instructions*/
    ..... ;
} while (expression) ; /* !!! N'oublier pas le ; */
```

20

Chapitre 3 : Les instructions de contrôle

III. Instructions itératives

3) La boucle do while

Exercice 1:

En utilisant la boucle do while, réaliser en C un algorithme qui cherche et affiche tous les entiers cubiques de trois chiffres.

Exercice 2:

Écrire un programme C qui permet de lire un entier positif et déterminer tous ses facteurs premiers.

Exemples:

$30 = 2 * 3 * 5$ | $36 = 2 * 2 * 3 * 3$ | $99 = 3 * 3 * 11$

21

Chapitre 3 : Les instructions de contrôle

IV. Les instructions de branchement non conditionnel

1) L'instruction continue

- Elle provoque le **passage à l'itération suivante** de **la boucle** en sautant à la fin du bloc.

➔ Elle provoque la non-exécution des instructions qui la suivent à **l'intérieur du bloc**.

Exemple :

```
void main() {
    int i;
    for (i = 0; i < 6; i++) {
        if (i==3)
            continue;
        printf ("i = %d \t", i);
    }
    printf ("\n La valeur de i a la sortie de la boucle = %d", i);
}
```

Cet exemple imprime

i= 0 i=1 i=2 i= 4 i= 5

La valeur de i a la sortie de la boucle = 6

22

Chapitre 3 : Les instructions de contrôle

IV. Les instructions de branchement non conditionnel

2) L'instruction break

- L'instruction break permet de sortir d'un bloc d'instruction associé à une instruction **répétitive** ou **alternative** contrôlée par les instructions **if**, **switch**, **for**, **while** ou **do while**.

Exemple :

```
void main () {
    int i;
    for (i = 0; i < 6; i++) {
        printf ("i = %d \t ", i);
        if (i==3)
            break ;
    }
    printf ("\n La valeur de i a la sortie de la boucle = %d", i);
}
```

Cet exemple imprime

i= 0 i=1 i=2 i=3

La valeur de i a la sortie de la boucle = 3

23

Chapitre 3 : Les instructions de contrôle

IV. Exercices d'application

Exercice 1 :

Un nombre est dit palindrome s'il est écrit de la même manière de gauche à droite ou de droite à gauche.

Exemples : 101 ; 22 ; 3663 ; 10801, etc.

Écrire un programme C permettant de déterminer et d'afficher tous les nombres palindromes compris dans l'intervalle [100..9999].

Exercice 2 :

Réaliser en C un algorithme qui affiche la suite de tous les nombres parfaits inférieurs ou égaux à un nombre naturel non nul donné noté n. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs autre que lui-même.

Exemple: $28 = 1 + 2 + 4 + 7 + 14$

Voici la liste des nombres parfaits inférieurs à 10000 : 6, 28, 496, 8128.

24

Chapitre 3 : Les instructions de contrôle

IV. Exercices d'application

Exercice 3 :

Les nombres de Fibonacci sont donnés par la récurrence :

$F_n = F_{n-2} + F_{n-1}$ avec $F_0 = 1$ et $F_1 = 1$.

Écrire un programme C qui affiche les 20 premiers nombres de Fibonacci.

Exercice 4 :

Deux entiers N1 et N2 sont dits frères si chaque chiffre de N1 apparaît au moins une fois dans N2 et inversement.

Exemples :

Si N1 = 1164 et N2 = 614 alors le programme affichera :

N1 et N2 sont frères

Si N1 = 405 et N2 = 554 alors le programme affichera :

N1 et N2 ne sont pas frères

Écrire un programme C qui saisit deux entiers N1 et N2, vérifie et affiche s'ils sont frères ou non.

25

Chapitre 3 : Les instructions de contrôle

IV. Exercices d'application

Exercice 5 :

Dans une entreprise, le calcul des jours de congés payés s'effectue de la manière suivante :

- Si une personne est rentrée dans l'entreprise depuis moins d'un an, elle a le droit à deux jours de congés par mois de présence, sinon à 28 jours au moins.
- Si c'est un cadre, s'il est âgé d'au moins 35 ans et si son ancienneté est supérieure ou égale à 3 ans, il lui est accordé deux jours supplémentaires. Si ce cadre est âgé d'au moins 45 ans et si son ancienneté est supérieure ou égale à 5 ans, il lui est accordé 4 jours supplémentaires, en plus des 2 accordés pour plus de 35 ans.

Réaliser en C l'algorithme qui calcule le nombre de jours de congés à partir de l'âge, de l'ancienneté et de l'appartenance au collège cadre d'un employé. Afficher le résultat.

26

Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Algorithmique et Programmation C

Filières: L1-Math Appliqué & L1-TIC

Chapitre 4: Les sous-programmes

Réalisé par:

Dr. Sakka Rouis Taoufik

1

Chapitre 4 : Les sous-programmes

I. Introduction

En C, les sous-programmes s'appellent des fonctions.

Les fonctions sont des parties de code source qui permettent de réaliser le même type de traitement plusieurs fois et/ou sur des variables différentes.

Une fonction a toujours un type de retour, qui correspond au type du résultat qu'elle peut renvoyer et qui peut être n'importe lequel des types que nous avons précédemment étudié.

Le type de retour peut être **void** si on souhaite que la fonction ne renvoie rien.

Une fonction a aussi un nom et une liste de zéro, un, ou plusieurs, paramètres.

2

Chapitre 4 : Les sous-programmes

II. Déclaration

Syntaxe :

```
TypeDeRetour NomFN (Type1 Parametre1, Type2 Parametre2)
{
    /*variables locales*/

    instructions de la fonction

    return ValeurDeRetour ; /*optionnelle si la fonction ne
                             retourne rien (de type void)*/
}
```

3

Chapitre 4 : Les sous-programmes

III. Les prototypes

Le prototype d'une fonction (voir chapitre 1) est une description d'une fonction qui est définie plus loin dans le programme. On place donc le prototype en début du programme (avant la fonction principale main()).

Le prototype d'une fonction reprend exactement l'en-tête de la fonction, mais pas son corps, qui est remplacé par un point-virgule.

Cette description permet au compilateur de « vérifier » la validité de la fonction à chaque fois qu'il la rencontre dans le programme.

4

Chapitre 4 : Les sous-programmes

III. Les prototypes

Syntaxe :

TypeDeRetour NomFN (Type1 Parametre1, Type2 Parametre2) ;

Exemples :

```
# include <stdio.h>
int somme (int a, int b); /*déclaration du prototype de la fonction somme */

void main () {
    int x, y, z;
    ...
    z= somme (x, y) ;
    ...
}
/* implémentation du corps de la fonction somme */
int somme (int a, int b) {
    ...
}
```

5

Chapitre 4 : Les sous-programmes

IV. Passage des paramètres

Tout paramètre d'une fonction peut être de type ordinaire (int, float,...) ou de type pointeur.

Dans le cas d'un paramètre ordinaire : le passage de ce dernier est appelé **passage par valeur**. Dans ce cas la fonction fait une copie de la valeur du paramètre. Elle n'utilise que la copie, qui peut être modifiée. Mais à la fin du traitement de la fonction, la valeur de la variable passée en paramètre n'a pas été modifiée, puisque c'est seulement la copie qui a été modifiée.

Exemple :

```
int plus (int a, int b) {
    a = a + b
    return a ;
}
```

6

Chapitre 4 : Les sous-programmes

IV. Passage des paramètres

Dans le cas d'un paramètre de type pointeur : le passage de ce dernier est appelé **passage par adresse**. Dans ce cas la fonction ne fait pas de copie de la valeur du paramètre, comme c'est fait dans le cas du passage par valeur. Par conséquent, si le paramètre change de valeur, alors à la sortie de la fonction, la variable passée en paramètre contient la dernière modification, donc la dernière valeur.

Exemple :

```
void add (int a, int b, int *c) {
    *c = a + b;
}
```

7

Chapitre 4 : Les sous-programmes

V. Appels des fonctions

Une fonction est appelée comme étant une instruction du programme si elle ne retourne rien, sinon elle est appelée dans une expression d'une affectation, comparaison ou autre.

Exemple:

```
#include <stdio.h>
float carre (float X) {
    X= X*X;
    return (X) ;
}
void main(){
    float A = 2, B;
    B= carre (A) ;
    /*la valeur de A ne change pas après l'appel de la fonction carre */
    printf("le carre de %f est=%f", A , B);
}
```

8

Chapitre 4 : Les sous-programmes

V. Appels des fonctions

Exemple:

```
#include <stdio.h>
void permute (float *x, float *y) {
    float aux;
    aux= *x ;
    *x= *y ;
    *y =aux ;
}
void main(){
    float A = 2.3, B=3.2;
    permute (&A, &B) ;
    /*la valeur de A sera 3.2 et la valeur de B sera 2.3 */
    printf("A=%f et B=%f", A , B);
}
```

9

Chapitre 4 : Les sous-programmes

V. Récursivité

1) Définition

La récursivité est un outil très puissant en programmation. Lorsqu'elle est bien utilisée, elle rend la programmation plus facile. C'est avant tout une méthode de résolution de problème.

On distingue deux types de récursivité :

récursivité directe : lorsqu'une procédure fait appelle à elle même

récursivité indirecte ou croisée : lorsqu'un sous-programme A appel un autre sous-programme B qui appelle A.

10

Chapitre 4 : Les sous-programmes

V. Récursivité

2) Illustration algorithmique

Cas 1 : récursivité directe

Procédure ProcRecursive (paramètres)

Début

...

ProcRecursive (valeurs)

...

Fin proc

Cas 2 : récursivité indirecte

Procédure A (paramètres)

Début

...

B (valeurs)

{ appel de la procédure B dans A }

...

Fin proc

Procédure B (paramètres)

Début

...

A (valeurs)

{ appel de la procédure A dans B }

...

Fin proc

11

Chapitre 4 : Les sous-programmes

V. Récursivité

3) Étude d'un exemple : la fonction factorielle

Dénotée par $n!$ (se lit factorielle n), c'est le produit de nombres entiers positifs de 1 à n inclus.

Exemples :

$$4! = 1 \cdot 2 \cdot 3 \cdot 4,$$

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5,$$

Noter que $4!$ peut s'écrire $4 \cdot 3 \cdot 2 \cdot 1 = 4 \cdot 3!$ et que $5!$ peut s'écrire $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5 \cdot 4!$

→ On peut conclure que : $n! = 1$ si ($n=1$)
 $n! = n \cdot (n-1)!$ si non

12

Chapitre 4 : Les sous-programmes

V. Récursivité

3) Étude d'un exemple : la fonction factorielle

Algorithmiquement	Traduction en C
Fonction Facto (n : Entier) : Entier Début Si (n = 1) Alors Facto \leftarrow 1 Sinon Facto \leftarrow n * Facto (n-1) FinSi Fin Fn	<pre>int facto (int n) { if (n == 1) return 1 ; else return n*facto (n-1) ; }</pre>

13

Chapitre 4 : Les sous-programmes

V. Récursivité

4) Mécanisme de fonctionnement de la récursivité

Chaque cas est réduit à un cas plus simple. Le calcul de 4! se ramène à celui de 3!, le calcul de 3! se ramène au calcul de 2! ... jusqu'à arriver à 1! qui donne directement 1.

Après on fait un retour arrière. Le résultat d'une ligne i sert au calcul de la ligne i-1.

Illustration : Considérons le calcul de 4! par la fonction récursive définie ci-dessus :

```
Facto(4) renvoie 4 * Facto(3)
    Facto(3) renvoie 3 * Facto(2)
        Facto(2) renvoie 2 * Facto(1)
            Facto(1) renvoie 1 (arrêt de la récursivité)
        Facto(2) renvoie 2 * 1 = 2
    Facto(3) renvoie 3 * 2 = 6
Facto(4) renvoie 4 * 6 = 24
```

14

Chapitre 4 : Les sous-programmes

V. Récursivité

4) Conseils d'écriture d'une fonction récursive :

Ces conseils sont illustrés par l'exemple suivant :

Écrire une fonction récursive permettant de calculer la somme des chiffres d'un entier n positif

Exemple : n = 528, donc la somme des chiffres de n est 15

1. Observer le problème afin de :

a) Paramétrer le problème : on détermine les éléments dont dépend la solution et qui caractérisent la taille du problème.

b) Décrire la condition d'arrêt : quand peut-on trouver "facilement" la solution ? **(une solution triviale)** :

Si on a le choix entre n = 528 et n = 8, il est certain qu'on choisit n = 8. La somme des chiffres de 8 est 8.

→ Conclusion : Si n a un seul chiffre, on arrête. La somme des chiffres est n lui-même.

15

Chapitre 4 : Les sous-programmes

V. Récursivité

4) Conseils d'écriture d'une fonction récursive :

c) réduire le problème à un problème d'ordre inférieur pour que la condition d'arrêt soit atteinte un moment donné : $\text{somChif}(528) = 8 + \text{somChif}(52)$

$$= 8 + (2 + \text{somChif}(5)) = 8 + (2 + 5)$$

2. Écriture de la fonction :

Fonction somChif (n : entier) : entier

Début

Si (n < 10) **alors** { condition d'arrêt }

 somChif ← n;

Sinon { réduction du problème : }

 somChif ← n **mod** (10) + somChif (n **div** (10));

FinSi

Fin Fn

16

Chapitre 4 : Les sous-programmes

V. Récursivité

4) Conseils d'écriture d'une fonction récursive :

3. Traduction en C:

17

Chapitre 4 : Les sous-programmes

V. Récursivité

4) Conseils d'écriture d'une fonction récursive :

Exercice :

Illustrer les conseils précédents pour écrire une fonction récursive qui permet de calculer le produit de deux entiers positifs a et b sans utilisé l'opérateur de multiplication (*).

Solution :

a) la solution de ce problème dépend des deux opérandes n1 et n2

b) Si vous avez le choix entre : 12×456 , 12×0 , 12×1

Lesquels des trois calculs sont le plus simple ?

$$\begin{aligned} \text{c) } 12 \times 9 &= 12 + 12 + 12 + \dots + 12 && (9 \text{ fois}) \\ &= 12 + (12 + 12 + \dots + 12) && (8 \text{ fois}) \\ &= 12 + 12 \times 8 \end{aligned}$$

etc ...

18

Chapitre 4 : Les sous-programmes

VI. Exercices d'application

Exercice 1 :

Ecrire un programme en C qui lit deux nombre naturel non nul m et n et qui détermine s'ils sont amis. Deux nombres entiers n et m sont qualifiés d'amis, si la somme des diviseurs de n est égale à m et la somme des diviseurs de m est égale à n (on ne compte pas comme diviseur le nombre lui-même et 1). Proposer une solution modulaire.

Exercice 2 :

Réaliser en C un algorithme d'une fonction qui recherche le premier nombre entier naturel dont le carré se termine par n fois le même chiffre.

Exemple : pour $n = 2$, le résultat est 10 car 100 se termine par 2 fois le même chiffre.

19

Chapitre 4 : Les sous-programmes

VI. Exercices d'application

Exercice 3: Récursivité simple

Soit la suite numérique U_n suivante : Si $n = 0$ alors $U_0 = 4$
sinon si $n > 0$ alors $U_n = 2 * U_{n-1} + 9$

Écrire une fonction C qui calcul le terme U_n .

Exercice 4: Récursivité croisée

Écrire deux fonctions C qui permettent de calculer les $n^{\text{èmes}}$ (n passé en argument) termes des suites entières U_n et V_n définies ci-dessous.

$$\begin{cases} U_0 = 1 \\ U_n = V_{n-1} + 1 \end{cases}$$

$$\begin{cases} V_0 = 0 \\ V_n = 2 * U_{n-1} \end{cases}$$

20

Chapitre 4 : Les sous-programmes

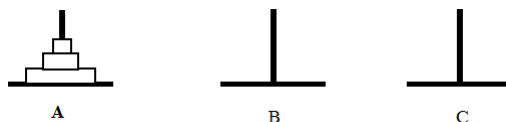
VI. Exercices d'application

Exercice 5: Tours de Hanoi

Le problème des tours de Hanoi est un grand classique de la récursivité car la solution itérative est relativement complexe. On dispose de 3 tours appelées A, B et C. La tour A contient n disques empilés par ordre de taille décroissante qu'on veut déplacer sur la tour B dans le même ordre en respectant les contraintes suivantes :

- On ne peut déplacer qu'un disque à la fois
- On ne peut empiler un disque que sur un disque plus grand ou sur une tour vide.

Illustration



21

Chapitre 4 : Les sous-programmes

VI. Exercices d'application

1) Observation

a) Ainsi, le paramétrage de la procédure déplacer sera le suivant :

Procédure déplacer(n : Entier ; A, B, C : Caractère)

b) Lorsque la tour A ne contient qu'un seul disque ($n=1$), la solution est évidente :

il s'agit de réaliser un transfert de la tour A vers B. Ce cas constitue donc la condition de sortie

c) Ainsi, pour déplacer n ($n>1$) disques de A vers B en utilisant éventuellement C, il faut :

- 1- déplacer ($n-1$) disques de A vers C en utilisant éventuellement B
- 2- réaliser un transfert du disque de A sur B
- 3- déplacer ($n-1$) disques de C vers B en utilisant éventuellement A.

2) Écriture de la procédure :

22

Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Algorithmique et Programmation C

Filières: L1_Math Appliqué & L1-TIC

Chapitre 5: Les tableaux

Réalisé par:

Dr. Sakka Rouis Taoufik

1

Chapitre 5 : Les tableaux

I. Déclaration

La déclaration d'un tableau à une dimension réserve un espace de mémoire contiguë dans lequel les éléments du tableau peuvent être rangés.

Syntaxe :

<type simple> <Nom tableau> [<dimension>] ;

Exemples :

```
int T [20] ;  
float T1 [100] ;  
char T2 [30] ;
```

2

Chapitre 5 : Les tableaux

I. Déclaration

La déclaration d'un tableau à deux dimensions (appelé matrice) réserve un espace de mémoire contiguë dans lequel les éléments du tableau peuvent être rangés.

Syntaxe:

<type simple><Nom tableau>[<dimlig>][<dimcol>] ;

Exemples :

```
int   A[10][10] ;
float B[2][10] ;
char  C[10][20] ;
```

3

Chapitre 5 : Les tableaux

I. Déclaration

En C, le nom d'un tableau est le représentant de **l'adresse du premier élément** du tableau. Les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse.

Exemple : int T[5] = {100,200,300,400,500} ;

.....	100	200	300	400	500
Adresse :	1E06	1E08	1E0A	1E0C	1E0E	1E10

→ On peut conclure que $T == \&T[0] == 1E06$

→ On peut conclure que **dans ce cas** $\&T[i] == \&T[0] + i * \text{sizeof}(\text{int})$

4

Chapitre 5 : Les tableaux

I. Déclaration

Dans le cas d'un tableau unidimensionnel, si la dimension n'est pas indiquée explicitement lors de l'initialisation, alors l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

Exemples:

```
int A [ ] = {10, 20, 30, 40, 50};
```

→ réservation de **5*sizeof(int)** octets

```
float B [ ] = {-1.05, 3.33, 87e-5, -12.3E4};
```

→ réservation de **4*sizeof (float)** octets

```
char C [ ] = {'a', 'b', 'c', 'd', 'e'};
```

→ réservation de **5*sizeof(char)** octets

5

Chapitre 5 : Les tableaux

I. Déclaration

Dans le cas d'une matrice, si les dimensions (nb des lignes ou nb des colonnes) ne sont pas indiquées explicitement lors de l'initialisation, l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

Exemples:

```
int B[ ][ ] = {{1,2,3,4},{10,20,30,40},{100,200,300,400}};
```

→ **Réservation** de **3*4* sizeof (int) = 24** octets

```
int B[][10] = {{0,10,20,30,40,50,60,70,80,90}, {10,11,12,13,14,15,16,17,18,19},{1,12,23,34,45,56,67,78,89,90} };
```

→ **Réservation** de **3*10*2 = 60** octets

6

Chapitre 5 : Les tableaux

II. Accès aux composantes d'un tableau

Considérons un tableau T de dimension N:

En algorithmique,

- l'accès au premier élément du tableau se fait par **T[1]**
- l'accès au dernier élément du tableau se fait par **T[N]**

En C,

- l'accès au premier élément du tableau se fait par **T[0]**
- l'accès au dernier élément du tableau se fait par **T[N-1]**

Exemple : int T[5] = {100, 200, 300, 400, 500} ;

Nom : Tnote	100	200	300	400	500
Indice :	0	1	2	3	4
Contenu	T[0]	T[1]	T[2]	T[3]	T[4]

7

Chapitre 5 : Les tableaux

II. Chargement et affichage d'un tableau

Solution 1:

```
#include<stdio.h>
void main() {
    int l, N, T[20];
    do {
        printf("donner le nombre d'éléments");
        scanf("%d", &N);
    } while (N<=0 || N>20) ;
    /*chargement */
    for (i=0; i<N; i++) {
        printf("T[%d]:",i);  scanf("%d", &T[i]);
    }
    /*affichage*/
    for (i=0; i<N; i++)
        printf("T[%d] = %d\t", i, T[i]);
}
```

8

Chapitre 5 : Les tableaux

II. Chargement et affichage d'un tableau

Solution 2:

/*Cette fonction permet le chargement d'un tableau de n entiers*/

```
void remplirTab (int X [ ], int n) {
    int i ;
    for (i = 0 ; i < n ; i++) {
        printf (" donner l'élément numéro %d ", i);
        scanf("%d", &X[i]);
    }
}
```

Attention : noter bien que le passage des tableaux est toujours par **adresse**

9

Chapitre 5 : Les tableaux

II. Chargement et affichage d'un tableau

Solution 2:

/*Cette fonction permet l'affichage des éléments d'un tableau de n entiers*/

```
void affichageTab (int X[ ], int n) {
    int i ;
    for (i = 0 ; i < n ; i++)
        printf (" Contenu de la case %d = %d", i, X[i]) ;
}
```

10

Chapitre 5 : Les tableaux

II. Chargement et affichage d'un tableau

Solution 2:

```
void main ( )
{
    int T [20], N ;
    do {
        printf("donner le nombre d'éléments");
        scanf("%d", &N);
    } while (N<=0 || N>20) ;
    remplirTab (T, N) ;
    afficheTab (T, N) ;
}
```

11

Chapitre 5 : Les tableaux

III. Chargement et affichage d'une matrice

/*Cette fonction permet le chargement d'une tableau de n lignes et m colonnes*/

```
void remplirMat (int X [ ] [ ], int n, int m) {
    int i, j ;
    for (i = 0 ; i < n ; i++)
        for (j = 0 ; j < m ; j++) {
            printf (" donner l'élément d'indice %d, %d", i, j) ;
            scanf("%d", &X[i][j]);
        }
}
```

12

Chapitre 5 : Les tableaux

III. Chargement et affichage d'une matrice

```
/*Cette fonction permet l'affichage des éléments d'une
matrice de n lignes et m colonnes*/
void affichageMat (int X[ ][ ], int n, int m) {
    int i, j ;
    for (i = 0 ; i < n ; i++) {
        for (j = 0 ; j < m ; j++)
            printf ("%7d", X[i][j]) ;
        /* Retour à la ligne */
        printf("\n");
    }
}
```

Attention : noter bien que le passage des matrices est toujours par **adresse**

13

Chapitre 5 : Les tableaux

IV. Exercices d'application

Exercice 1 : Somme, produit et moyenne des éléments

Écrire un programme C qui lit la taille N ($5 < N < 20$) d'un tableau T de type int. Remplit ce tableau par des valeurs entrées au clavier et affiche le tableau.

Calculer et afficher ensuite la somme, le produit et la moyenne des éléments du tableau.

Exercice 2 : Occurrence de 0

Écrire un programme C qui lit la taille N d'un tableau T de type int (taille maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Effacer ensuite toutes les occurrences de la valeur 0 dans le tableau T et tasser les éléments restants. Afficher le tableau résultant.

14

Chapitre 5 : Les tableaux**IV. Exercices d'application****Exercice 3:**

Écrire un programme C qui lit les dimensions L et C d'un tableau T à deux dimensions (matrice) du type **int** (tailles maximales: 50 lignes et 50 colonnes). Remplir ce tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de tous ses éléments.

Exercice 4:

Écrire un programme C qui transfère un tableau M à deux dimensions L et C (tailles maximales: 10 lignes et 10 colonnes) dans un tableau V à une dimension L*C.

Exemple:

```
a b c d
e f g h ==> a b c d e f g h i j k l
i j k l
```

TD N°1

Tout premier programme C

Exercice 1 : Syntaxe

Indenter correctement le programme suivant :

```
#include <stdlib.h> #include  
<stdio.h> void main (  
) { int x ; printf ( "Entrer la valeur de x\n" ) ;  
scanf ( "%d", &x ) ; if ( x < 0) printf (  
"x est un entier negatif\n" )  
; else printf ( "x est un entier  
positif\n" ) ; }
```

Exercice 2 : Types de données

1. Expliquer la différence entre ces deux objets du langage C : + et '+'
2. Quels sont les types des données suivantes ? Donner également leurs valeurs.

1	'1'	"1"	1.0	.1e1
---	-----	-----	-----	------

3. Pour imprimer un caractère à l'aide de sa valeur, disons c, dans le code ASCII il suffit d'utiliser l'instruction printf ("%c",c). Écrire un programme dans lequel on déclare et initialise un caractère, et qui affiche le caractère suivant (par exemple, si on initialise le caractère à 'd', le programme affiche e).

Exercice 3 : Expressions

Éliminer les parenthèses superflues dans les expressions suivantes :

```
a = ( x + 5)          /*expression 1 */  
a = ( x = y ) + 2     /*expression 2 */  
a = ( x == y )        /*expression 3 */  
( i++) * ( n + p)     /*expression 4 */
```

Exercice 4 : E/S simple

Écrire un programme qui demande à l'utilisateur de saisir un entier et qui affiche à l'écran le caractère correspondant.

Exercice 5 :E/S conversationnelles

Quels résultats fournit le programme suivant :

```
#include <stdio.h>  
#include <stdlib h>  
void main ( ) {  
    int i , j , n ;  
    i = 0 ; n = i++;  
    printf ( "A : i=%d n=%d\n" , i , n ) ;  
    i = 10 ; n = ++i ;  
    printf ( "B : i=%d n=%d\n" , i , n ) ;  
    i = 20 ; j = 5 ; n = i++ * ++j ;  
    printf ( "C : i=%d j=%d n=%d\n" , i , j , n ) ;  
    i = 15 ; n = i += 3 ;  
    printf ( "D : i=%d n=%d\n" , i , n ) ;  
}
```

```

        i = 3 ; j = 5 ; n = i *--j ;
        printf ( "E : i=%d j=%d n=%d\n" , i , j , n ) ;
    }

```

Exercice 6 : E/S conversationnelles

Quels résultats fournit le programme suivant :

```

#include <stdio.h>
#include <stdlib.h>
void main ( ) {
    int n=543;
    int p=5;
    float x=34.5678;
    printf ( "A : %d %f \n" ,n , x ) ;
    printf ( "B : %4d %10f \n" ,n , x ) ;
    printf ( "C : %2d %3f \n" ,n , x ) ;
    printf ( "D : %10.3 f %10.3 e \n" ,x , x ) ;
    printf ( "E : %-5d %f \n" ,n , x ) ;
    printf ( "F : %*d\n" ,p , n ) ;
    printf ( "G : %*.*f \n" , 12 , 5 , x ) ;
    printf ( "H : %x : %8x : \n" ,n , n ) ;
    printf ( "I : %o : %8o : \n" ,n , n ) ;
}

```

Exercice 7 : E/S conversationnelles

1) Quelles seront les valeurs des deux variables n et p (de type int), par l'instruction suivante :

```
scanf ( "%d %d",&n, &p ) ;
```

Lorsqu'on lui fournit les données suivantes (le symbole ^ représente un espace et le symbole @ représente une fin de ligne, c'est-à-dire une "validation").

- (a) 253^45@
- (b) ^253^@
- ^^4^5@

2) Quelles seront les valeurs des deux variables n et p (de type int), par l'instruction suivante :

```
scanf ( "%4d %2d",&n,&p ) ;
```

Lorsqu'on lui fournit les données suivantes :

- (a) 12^45@
- (b) 123456@
- (c) 123456^7@
- (d) 1^458@
- (e) ^^4567^8912@

TD N°1

Tout premier programme C

Exercice 1 : Syntaxe

```
#include <stdlib.h>
#include <stdio.h>
void main () {
    int x ;
    printf ( "Entrer la valeur de x\n" ) ;
    scanf ( "%d", &x ) ;
    if ( x < 0)
        printf ("x est un entier negatif \n" ) ;
    else
        printf ( "x est un entier positif \n" ) ;
}
```

Exercice 2 : Types de données

Question 1)

+ est un opérateur du langage C (addition).

'+' est la constante littérale de type char +.

Question 2)

1 est un entier de type long, sa valeur est 1.

'1' est de type char, sa valeur est le caractère 1 (en ASCII sa valeur est 49).

"1" est une chaîne de caractères constituée du caractère 1.

1.0 est un float de valeur 1.

.1e1 est un float de valeur 1.

Question 3)

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

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

```
void main () {
    char x ;
    printf ( "Entrer la valeur de x \n" ) ;
    scanf ( "%c", &x ) ;
    printf("le succ de %c est %c \n", x, x++) ;
}
```

Exercice 3 : Expressions

$a = x + 5$ L'opérateur + est prioritaire sur l'opérateur d'affectation =.

$a = (x = y) + 2$ L'opérateur + étant prioritaire sur =, donc ici les parenthèses sont indispensables

$a = x == y$ L'opérateur de comparaison == est prioritaire sur l'opérateur d'affectation =.

$i++ * (n + p)$ L'opérateur ++ est prioritaire sur * ; par contre, l'opérateur * est prioritaire sur + ; de sorte qu'on ne peut éliminer les dernières parenthèses.

Exercice 4 : E/S simple

```
#include <stdlib.h>
#include <stdio.h>
void main () {
    int x ;
    printf ( "Entrer un entier \n" ) ;
    scanf ( "%d", &x ) ;
    printf("le caractère correspondant à %d est = %c \n", x, x) ;
}
```

Exercice 5 :E/S conversationnelles

Quels résultats fournit le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
void main () {
    int i , j , n ;
    i = 0 ; n = i++;
    printf ( "A : i=%d n=%d\n" , i , n ) ;
    i = 10 ; n = ++i ;
    printf ( "B : i=%d n=%d\n" , i , n ) ;
    i = 20 ; j = 5 ; n = i++ * ++j ;
    printf ( "C : i=%d j=%d n=%d\n" , i , j , n ) ;
    i = 15 ; n = i += 3 ;
    printf ( "D : i=%d n=%d\n" , i , n ) ;
    i = 3 ; j = 5 ; n = i *--j ;
    printf ( "E : i=%d j=%d n=%d\n" , i , j , n ) ;
}
```

```
A : i=1 n=0
B : i=11 n=11
C : i=21 j=6 n=120
D : i=18 n=18
E : i=12 j=4 n=12
```

Exercice 6 : E/S conversationnelles

Quels résultats fournit le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
void main () {
    int n=543;
    int p=5;
    float x=34.5678;
    printf ( "A : %d %f \n" ,n , x ) ;
    printf ( "B : %4d %10f \n" ,n , x ) ;
    printf ( "C : %2d %3f \n" ,n , x ) ;
    printf ( "D : %10.3 f %10.3 e \n" ,x , x ) ;
    printf ( "E : %-5d %f \n" ,n , x ) ;
    printf ( "F : %*d\n" ,p , n ) ;
}
```

```

printf ( "G : %*.*f \n" , 12 , 5 , x ) ;
printf ( "H : %x : %8x : \n" , n , n ) ;
printf ( " I : %o : %8o : \n" , n , n ) ;
}

```

Exercice 7 : E/S conversationnelles

1) Quelles seront les valeurs des deux variables n et p (de type int), par l'instruction suivante :

```
scanf ( "%d %d",&n, &p ) ;
```

Lorsqu'on lui fournit les données suivantes (le symbole ^ représente un espace et le symbole @ représente une fin de ligne, c'est-à-dire une "validation").

(a) 253^45@

(b) ^253^@

^^4^5@

2) Quelles seront les valeurs des deux variables n et p (de type int), par l'instruction suivante :

```
scanf ( "%4d %2d",&n,&p ) ;
```

Lorsqu'on lui fournit les données suivantes :

(a) 12^45@

(b) 123456@

(c) 123456^7@

(d) 1^458@

(e) ^^4567^8912@

TD N°2

Les structures conditionnelles

Exercice 1:

Réaliser en C un algorithme qui lit trois valeurs entières (A, B et C) au clavier et qui affiche la plus grande des trois valeurs, en utilisant:

- a) si-sinon et une variable d'aide MAX
- b) si-sinon-si sans variable d'aide
- c) les opérateurs conditionnels et une variable d'aide MAX
- d) les opérateurs conditionnels sans variable d'aide

Exercice 2:

Réaliser en C un algorithme qui lit trois valeurs entières (A, B et C) au clavier. Trier les valeurs A, B et C par échanges successifs de manière à obtenir ($A \leq B \leq C$) puis afficher les trois valeurs.

Exercice 3:

Réaliser en C un algorithme qui lit deux valeurs entières (A et B) au clavier et qui affiche :

- e) le signe du produit de A et B sans faire la multiplication.
- f) le signe de la somme de A et B sans faire l'addition.

Exercice 4:

Réaliser en C un algorithme qui permet de saisir un numéro de couleur de l'arc-en-ciel et d'afficher la couleur correspondante :

1 : rouge, 2 : orangé, 3 : jaune, 4 : vert, 5 : bleu, 6 : indigo et 7 : violet.

Exercice 5:

On veut déterminer si une année A est bissextile ou non.

Si A n'est pas divisible par 4 l'année n'est pas bissextile.

Si A est divisible par 4, l'année est bissextile sauf si A est divisible par 100 et pas par 400.

Exemples :

1980 et 1996 sont bissextiles car elles sont divisibles par 4

2000 est une année bissextile car elle est divisible par 400

2100 et 3000 ne sont pas bissextiles car elles ne sont pas divisibles par 400.

Réaliser en C un algorithme qui effectue la saisie de la donnée, détermine si l'année est bissextile ou non et affiche le résultat.

Exercice 6:

Un temps donné est représenté sous la forme: heure, minute et seconde de type naturel. On veut lui ajouter une seconde et afficher avec la même représentation le temps ainsi obtenu.

Réaliser en C un algorithme qui effectue la saisie des données, réalise le calcul et affiche les résultats.

TD N°3

Les structures itératives

Exercice 1:

Réaliser en C un algorithme qui permet de déterminer la somme des chiffres d'un nombre entier donné (exemple : pour $N = 25418$, on aura $2+5+4+1+8 = 20$).

Exercice 2:

Un nombre réel X et un nombre entier N étant donnés, proposer un programme C qui fait calculer X^N . Étudier tous les cas possibles (N positive ou négative).

Exercice 3:

Les nombres de Fibonacci sont donnés par la récurrence :

$$F_n = F_{n-2} + F_{n-1} \text{ avec } F_0 = 1 \text{ et } F_1 = 1.$$

Écrire un programme C qui affiche les 20 premiers nombres de Fibonacci.

Exercice 4:

Un entier naturel de trois chiffres est dit cubique s'il est égal à la somme des cubes de ses trois chiffres.

Exemple :

$$153 \text{ est cubique car } 153 = 1^3 + 5^3 + 3^3$$

Réaliser en C un algorithme qui cherche et affiche tous les entiers cubiques de trois chiffres.

Exercice 5:

Écrire un programme C qui permet de lire un entier positif et déterminer tous ses facteurs premiers.

Exemples:

$$30 = 2 * 3 * 5$$

$$36 = 2 * 2 * 3 * 3$$

$$99 = 3 * 3 * 11$$

Exercice 6:

Deux nombres entiers sont premiers entre eux s'ils n'ont pas d'autres diviseurs communs que 1.

- 7 et 13 n'ont que 1 comme diviseur commun donc 7 et 13 sont premiers entre eux.
- 12 et 32 ont plusieurs diviseurs communs : 1 ; 2 et 4 donc 12 et 32 ne sont pas premiers entre eux.

Écrire un programme C qui saisit deux entiers $N1$ et $N2$, vérifie et affiche s'ils sont premiers entre eux ou non.

Exercice 7:

Un nombre est dit palindrome s'il est écrit de la même manière de gauche à droite ou de droite à gauche.

Exemples : 101 ; 22 ; 3663 ; 10801, etc.

Écrire un programme C permettant de déterminer et d'afficher tous les nombres palindromes compris dans l'intervalle [100 .. 9999].

Exercice 8:

Réaliser en C un algorithme qui imprime pour un entier naturel n donné :

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
.....
.....
.....
1 2 3 4 5 6 ... n
```

Exercice 9 :

Deux entiers $N1$ et $N2$ sont dits frères si chaque chiffre de $N1$ apparaît au moins une fois dans $N2$ et inversement.

Exemples :

- Si $N1 = 1164$ et $N2 = 614$ alors le programme affichera : $N1$ et $N2$ sont frères
- Si $N1 = 405$ et $N2 = 554$ alors le programme affichera : $N1$ et $N2$ ne sont pas frères

Écrire un programme C qui saisit deux entiers $N1$ et $N2$, vérifie et affiche s'ils sont frères ou non.

Exercice 10:

On veut réaliser en même temps, à l'aide de soustractions successives, la division entière et le calcul du modulo de deux nombres naturels non nuls dividende et diviseur.

Réaliser en C l'algorithme qui effectue la saisie des données, réalise le calcul et affiche les résultats.

Exercice 11:

Réaliser en C un algorithme qui affiche la suite de tous les nombres parfaits inférieurs ou égaux à un nombre naturel non nul donné noté n . Un nombre est dit parfait s'il est égal à la somme de ses diviseurs autre que lui-même.

Exemple: $28 = 1 + 2 + 4 + 7 + 14$

Voici la liste des nombres parfaits inférieurs à 10000 : 6, 28, 496, 8128.

Exercice 12:

Dans une entreprise, le calcul des jours de congés payés s'effectue de la manière suivante :

- Si une personne est rentrée dans l'entreprise depuis moins d'un an, elle a le droit à deux jours de congés par mois de présence, sinon à 28 jours au moins.
- Si c'est un cadre, s'il est âgé d'au moins 35 ans et si son ancienneté est supérieure ou égale à 3 ans, il lui est accordé deux jours supplémentaires. Si ce cadre est âgé d'au moins 45 ans et si son ancienneté est supérieure ou égale à 5 ans, il lui est accordé 4 jours supplémentaires, en plus des 2 accordés pour plus de 35 ans.

Réaliser en C l'algorithme qui calcule le nombre de jours de congés à partir de l'âge, de l'ancienneté et de l'appartenance au collège cadre d'un employé. Afficher le résultat.

TD N°4

Les fonctions et les tableaux en C

Exercice 1:

Écrire en C un algorithme d'une fonction Triangle qui permet de vérifier si les 3 nombres a, b et c peuvent être les mesures des côtés d'un triangle rectangle.

Remarque: D'après le théorème de Pythagore, si a, b et c sont les mesures des côtés d'un rectangle, alors $a^2 = b^2 + c^2$ ou $b^2 = a^2 + c^2$ ou $c^2 = a^2 + b^2$

Exercice 2 :

Ecrire un programme en C qui lit deux nombre naturel non nul *m* et *n* et qui détermine s'ils sont amis. Deux nombres entiers n et m sont qualifiés d'amis, si la somme des diviseurs de n est égale à m et la somme des diviseurs de m est égale à n (on ne compte pas comme diviseur le nombre lui-même et 1). Proposer une solution modulaire.

Exercice 3 :

Réaliser en C un algorithme d'une fonction qui recherche le premier nombre entier naturel dont le carré se termine par n fois le même chiffre.

Exemple : pour n = 2, le résultat est 10 car 100 se termine par 2 fois le même chiffre.

Exercice 4: Récursivité simple

Soit la suite numérique U_n suivante : Si $n = 0$ alors $U_0 = 4$ sinon si $n > 0$ alors $U_n = 2 * U_{n-1} + 9$
Écrire une fonction C qui calcul le terme U_n .

Exercice 5: Récursivité croisée

Écrire deux fonctions C qui permettent de calculer les $n^{\text{èmes}}$ (n passé en argument) termes des suites entières U_n et V_n définies ci-dessous.

$$\begin{cases} U_0=1 \\ U_n=V_{n-1}+1 \end{cases} \quad \begin{cases} V_0=0 \\ V_n=2*U_{n-1} \end{cases}$$

Exercice 6: Fibonnaci

La suite de FIBONNACI est définie par :

Fib(1)= 1 ;

Fib(2)= 2 ;

Fib(n)= Fib(n-1)+ Fib(n-2) n >2;

Pour pouvoir programmer cette suite de FIBONNACI, on adopte la décomposition fonctionnelle suivante :

1. **init** : prépare les conditions nécessaires et suffisantes pour calculer **fib(3)**.

Programmer en C le sous-programme **init**.

2. **suivant** : calcule le terme suivant. Programmer en C le sous-programme **suivant**.

3. **terme** : rend le terme courant. Programmer en C le sous-programme **terme**.

4. **fib** : calcule le nombre de FIBONNACI d'un rang donné. Programmer en C le sous-programme **fib**.

5. **est_ce_fibo** : permet de voir si un nombre donné est un nombre de FIBONNACI.

Programmer en C le sous-programme **est_ce_fibo**.

Exercice 7 : Somme, produit et moyenne des éléments

Écrire un programme C qui lit la dimension N ($5 < N < 20$) d'un tableau T de type int. Remplit ce tableau par des valeurs entrées au clavier et affiche le tableau.

Calculer et afficher ensuite la somme, le produit et la moyenne des éléments du tableau.

Exercice 8 : Occurrence de 0

Écrire un programme C qui lit la dimension N d'un tableau T de type int (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Effacer ensuite toutes les occurrences de la valeur 0 dans le tableau T et tasser les éléments restants. Afficher le tableau résultant.

Exercice 9 : Inverse

Écrire un programme C qui lit la dimension N d'un tableau T de type int (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Ranger ensuite les éléments du tableau T dans l'ordre inverse sans utiliser de tableau d'aide. Afficher le tableau résultant.

Idée: Echanger les éléments du tableau à l'aide de deux indices qui parcourent le tableau en commençant respectivement au début et à la fin du tableau et qui se rencontrent en son milieu.

Exercice 10 : Fréquence

Soit T un tableau contenant n éléments de type entier et x un entier quelconque.

Écrire une fonction **int frequency (int [] T, int n, int x)** qui retourne le nombre d'apparitions de x dans le tableau T.

Exercice 11 : Éléments distincts

Écrire une fonction C qui permet de remplir un tableau T par n entiers différents (chaque entier est présent une seule fois).

Exercice 12 : Nombre d'éléments distincts

On se propose d'écrire un programme C qui saisit un entier n ($10 < n \leq 100$) puis un tableau T composé de n entiers. Le programme calcule le nombre d'éléments distincts de T.

Exercice 13 : Séquences strictement croissantes

Écrire une fonction C qui permet d'afficher les sous séquences strictement croissantes depuis un tableau de N entiers.

Exemple pour T = 5|7|9|2|3|1|20|25

La fonction affiche :

5|7|9

2|3

1|20|25

Exercice 14: Fréquence

Présenter d'une façon informelle et réaliser en C un algorithme permettant de compter la fréquence des éléments stockés dans un tableau. Ces éléments sont des entiers compris entre 0 et 99.

Exercice 15: Le plus fréquent

Proposer d'une façon informelle et réaliser en C un algorithme qui détermine l'élément le plus fréquent dans un tableau.