Atelier de Programmation I

Rania REBAÏ BOUKHRISS
Docteur en Informatique
Enseignante Permanente à l'IIT
Responsable du Cycle Licence en Informatique
rania.rebai@iit.ens.tn

2021-2022

Objectifs de cours

- Cours d'initiation au langage C qui vise à :
  - Découvrir le langage de programmation C
  - Savoir réagir devant un problème de programmation
  - Ecrire des programmes corrects partant des programmes simples à des programmes complexes.
- Public Cible :
  - Ce cours est destiné aux étudiants du premier année Licence en Informatique

Organisation du cours

- Volume horaire: ce cours est présenté, de manière hebdomadaire, comme suit :
  - ±10,5H de cours
  - ± 31,5H de travaux pratiques
- Evaluation :
  - Coefficient : 3
  - Devoir Surveillé (30%)
  - Examen Principal (70%)

Plan du cours

Introduction Générale

Algorithme Vs Programme

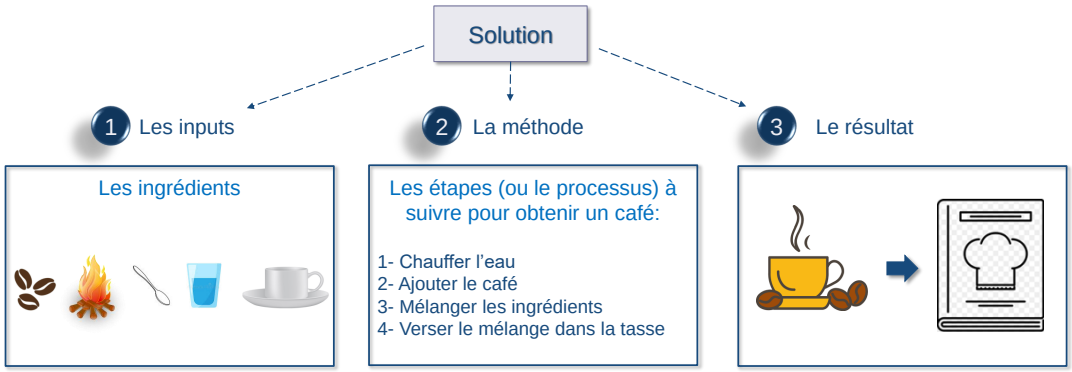
- L'algorithme ainsi que le programme informatique sont des méthodes ou encore des techniques utilisées pour résoudre un problème précis

Les deux permettent de trouver la solution à un problème

Quelle est donc la différence ?

Algorithme Vs Programme

- Concevoir une solution au problème suivant : Faire un tasse de café



Algorithme Vs Programme

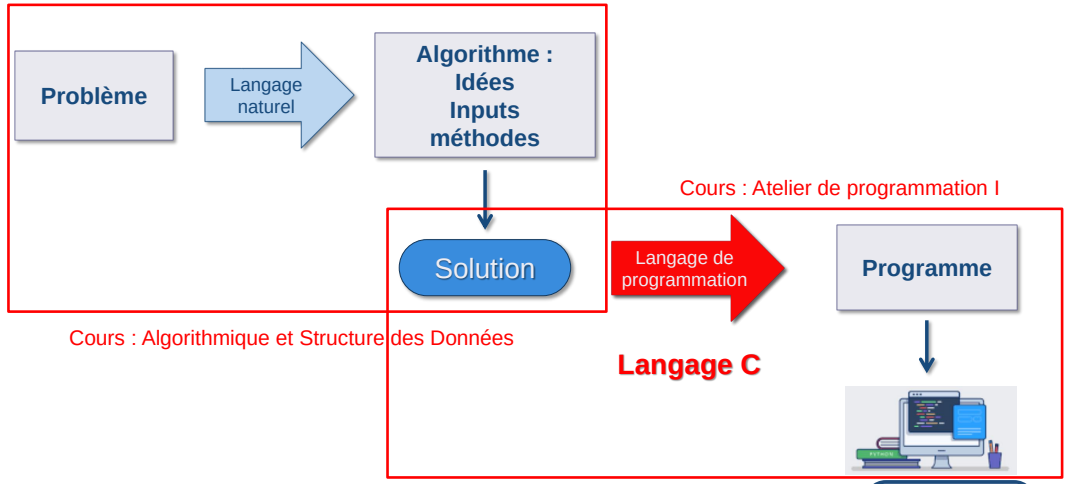
- Algorithme : Ensemble des étapes vers la solution

Algorithme Vs Programme

- **Algorithme** : est la solution à un problème écrite en langage naturel compréhensible par des humains
- **Programme** : est la solution à un problème écrite en langage de programmation compréhensible par un ordinateur

➡ Un programme c'est un algorithme traduis dans un langage de programmation

Algorithme Vs Programme



Historique et avantages du langage C

Historique

- Le langage C a été mis au point par D.Ritchie et B.W.Kernighan au début des années 70.
- Leur but était de permettre de développer un langage qui permettrait d'obtenir un système d'exploitation de type UNIX portable.
- D.Ritchie et B.W.Kernighan se sont inspirés des langages B et BCPL, pour créer un nouveau langage : le langage C.

Historique et avantages du langage C

Avantages

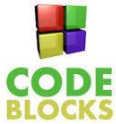
- **Universel:** C n'est pas destiné pour un domaine d'applications spécifique
- **Rapide:** C permet de développer des programmes concis et rapides
- **Portable:** C est un langage qui peut être utilisé sur n'importe quel système ayant un compilateur C.
- **Extensible:** C peut être étendu et enrichi par l'utilisation de bibliothèques.
- **Modulaire/procédure :** Un programme C peut être divisé en sous-programmes appelés fonctions.

Environnements de développement

Windows, Linux et Mac OS

Open source

Windows, Linux et Mac OS



Libre

Windows, Linux et Mac OS

Open source

Windows

Open Source

Windows, Linux et Mac OS

Open Source

Notions de base

Bibliothèques de fonctions

- Le langage C fait appel à des bibliothèques de fonctions qui sont disponibles sous forme précompilées (.lib).
- Pour les utiliser dans un programmes C, il faut inclure des fichiers en-tête (.h)
- Ces fichiers contiennent les prototypes des fonctions prédéfinies dans les bibliothèques et créent un lien entre les fonctions précompilées et les programmes.
- Pour inclure les fichiers en-tête: #include <fichier.h>

Bibliothèques de fonctions

- Pour le compilateur, différents types de fichiers seront identifiés par leurs extensions:
  - **.c** : fichier source
  - **.obj** : fichier compilé
  - **.exe** : fichier exécutable
  - **.lib** : bibliothèque de fonctions précompilées
  - **.h** : bibliothèque en-tête

Structure générale d'un programme écrit en C

```
#include<stdio.h>
#include<conio.h>
.
.
.
void main ( )
{
    .
    .
    .
    printf( "Bonjour" ) ;
    getch ( ) ;
}
```

Inclure les fichiers en-tête

La fonction principale

On place dans les accolades du main les instructions à exécuter

Notions de base : Les Fonctions

La fonction main:

- L'exécution d'un programme C mène à l'appel automatique de la fonction main.
- Elle est inévitablement présente dans tous les programmes.
- Elle constitue la fonction principale des programmes en C.

```
void main()
{
        <déclarations>
        <instructions>
}
```


Notions de base : Les variables

Les variables

- Avant d'être utilisée, chaque variable doit être déclarée.

  <Type_variable> Nom_variable

- Toute variable est caractérisée par :
  1. Un type qui indique généralement et implicitement la longueur physique de la zone en nombre d'octets
  2. Un identificateur respectant la règle d'écriture des identificateurs
- Les variables contiennent les valeurs utilisées par le programme lors de son exécution
- A chaque type de variables est associé ses opérateurs

Notions de base : Les instructions

Instructions

- En C, toute instruction simple est terminée par un point-virgule ; (même si elle se trouve en dernière position dans un bloc d'instructions).
- Exemple:

```
Printf("\n Hello \n") ;
```

Notions de base : Les identificateurs

Les identificateurs

- Les noms des fonctions et des variables en C sont composés d'une suite de lettres et de chiffres.
- L'ensemble des symboles autorisés est donc :

  { a...z, A...Z, 0...9, _ }
- Le premier caractère doit être obligatoirement une lettre (ou le symbole '_')
- C distingue les minuscules et les majuscules, ainsi:
  - 'Variable_name' n'est pas le même variable que 'variable_name'
- La longueur des identificateurs n'est pas limitée, mais C distingue seulement les 31 premiers caractères.

Notions de base : Les identificateurs

Les identificateurs

Exemples d'identificateurs corrects	Exemples d'identificateurs incorrects
nom1	1nom
nom_2	Nom.2
_nom_3	-nom-3
Nom_de_variable	Nom␣de␣variable
deuxieme_choix	deuxième_choix
mot_francais	mot_français

Notions de base : Les commentaires

Les commentaires

- Un commentaire commence toujours par les deux symboles /* et se termine par les deux symboles */
- Il est interdit d'utiliser des commentaires imbriqués.
- Exemple:

```
/* ceci est un commentaire correct sur plusieurs lignes */

/* ceci est /* évidemment */ incorrect */

// ceci est un commentaire correct sur une seule ligne
```


Premier programme en C

Exercice: afficher "bonjour" à l'écran

- Pour le faire, il faut:
  - Inclure les bibliothèques
  - Inclure le main

```
main()
{
    déclaration des variables: aucune
    instruction: écrire "bonjour "
}
```

  - La fonction prédéfinie qui permet d'écrire à l'écran est **printf**, elle est contenue dans le fichier en-tête **stdio.h**
  - Sa syntaxe est:

```
printf("Afficher ce que l'on veut écrire");
```

Premier programme en C

Voici donc notre premier programme:

```
#include <stdio.h>
void main()
{
        printf("bonjour\n");  /*toute instruction se termine par un point virgule*/
 }
```


Notions de base : Les séquences d'échappement

Les séquences d'échappement

- Il existe en C plusieurs couples de symboles qui contrôlent l'affichage ou l'impression du texte.
- Les séquences d'échappement sont toujours précédées par le caractère d'échappement "\".

\t tabulation
\n nouvelle ligne
Si l'on veut écrire le symbole " ou \ : **\"** et ****

Types de base, opérateurs, expressions

Types de base, opérateurs, expressions

Types de base, opérateurs, expressions

- Dans un programme, on trouve des variables et des constantes
- Chaque variable ou constante possède un type
- Les constantes et les variables peuvent être combiner dans des expressions à l'aide d'opérateurs

Types de base

- Le type d'une donnée détermine :
  - le nombre d'octets à réserver en mémoire
  - l'ensemble des valeurs admissibles
  - l'ensemble des opérateurs qui peuvent y être appliqués

Types de base : entier

- Trois types de base servent à représenter les entiers signés (de signe quelconque)

Type	Description	Taille mémoire	Chaine de format
int	entier standard signé	4 octets	%d
short	entier court signé	2 octets	%hd
long	entier long	4 octets	%ld

- short : (2 octets) entiers compris entre -32768 et +32767 (-2^{15} et $2^{15}-1$)
- int ou long : (4 octets) entiers compris entre -2147483648 et +2147483647 (-2^{31} et $2^{31}-1$)

Types de base : entier

- Si on préfixe un type entier par unsigned, on le restreint a des valeurs uniquement positives.

Type	Description	Taille mémoire	Chaine de format
unsigned int	entier positif	4 octets	%u
unsigned short	entier court non signé	2 octets	%hu
unsigned long	entier long non signé	4 octets	%lu

- Unsigned short : (2 octets) entiers compris entre 0 et + 65 534 (0 et $2^{16}-1$)
- Unsigned int : (4 octets) entiers compris entre 0 et + 4 294 967 294 (0 et $2^{32}-1$)
- Unsigned long : (4 octets) entiers compris entre 0 et + 4 294 967 294 (0 et $2^{32}-1$)

Types de base : réel

Trois types de base servent a représenter les flottants :

Type	Description	Taille mémoire	Chaine de format
float	Réel standard	4 octets	%f
double	Réel double précision	8 octets	%f ou %e
long double	Réel quadruple précision	16 octets	%Le

•float : (4 octets) dans ce cas max vaut 255 : ensemble des nombres [-3.40282347E38 .. -1.40239846E-45], 0, [1.40239846E-45 .. 3.40282347E38]
•double : (8 octets) dans ce cas max vaut 2047 : ensemble des nombres [-1.79769313486231570E308..-4.94065645841246544E-324], 0, [4.94065645841246544E-324 .. 1.79769313486231570E308]

Types de base : caractère

Une variable du type char peut subir les mêmes opérations que les variables du type short, int ou long

Type	Description	Taille mémoire	Chaine de format
char	caractère signé	1 octet	%c
unsigned char	caractère non signé	1 octet	%c

•char: (1 octet) entiers compris entre -128 et 127 (-2^7 et 2^7-1)
•Unsigned char: (1 octet) entiers compris entre 0 et 255 (0 et 2^8-1)

- **Remarque** : L'opérateur sizeof(type) renvoie le nombre d'octets réservés en mémoire pour chaque type d'objet.
- Exemples :

```
n = sizeof(char); /* n = 1 */
m = sizeof(int); /* m = 4 */
```


Types de base : booléen

Il n'existe pas de type spécifique pour les variables booléennes

Tous les types de variables numériques peuvent être utilisés pour exprimer des opérations logiques:

- La variable logique FAUX correspond à la valeur numérique 0
- La variable logique VRAI correspond toute valeur différente de 0

Déclaration des variables simples

Syntaxe :

<Type> <NomVar1>,<NomVar2>,...,<NomVarN>;

Exemples :

```
int n1,n2,n3;
float m1,m2,m3;
char  c1,c2,c3;
```

Définition des constantes

1ère Méthode : utilisation du mot clé const lors de la déclaration

- **Exemple :**

```
void main()
{
        const float PI = 3.14159;
        ...
}
```

2ème Méthode : utilisation de la directive de compilation #define

- **Exemple :**

```
#define PI 3.14159
void main()
{
        ...
}
```


Affectation / Initialisation

Initialisation : Utilisation du symbole =

Exemples :

```
int i;
i = 50;
```
⟺
```
int i = 50;
```

```
char x;
x = 'A';
```
⟺
```
char x = 'A';
```

```
char  c1=65,c2=98,c3='a';
int n1=1,n2=2,n3=3;
```


Affectation / Initialisation

Affectation :

Syntaxe :

```
<NomVariable> = <Expression/Valeur>;
```

- L'affectation avec des **valeurs constantes**

```
n1=10;        n2=12.75;        c1='r';    c2='R';
```

- L'affectation avec des **valeurs de variables**

```
c1=c2;        n1=c1;
```

- L'affectation avec des **valeurs d'expressions**

```
AIRE = PI*pow(R,2);        c = a + b;
```


Les opérateurs standard

Les opérateurs arithmétiques

signe	opération
+	Addition
-	Soustraction
*	Multiplication
/	Division entière et relationnelle
%	modulo (reste d'une div. entière)

- Opérateurs arithmétiques pour les entiers : +, -, *, /, %
- Opérateurs arithmétiques pour les réels : +, -, *, /

Les opérateurs standard

Les opérateurs arithmétiques

- Il est possible d'appliquer ces opérateurs sur les variables de type char.
- Exemple :

```
char c, d;
c = 'G';
d = c + 'a' - 'A';   /* d='g' */
```


Les opérateurs standard

Les opérateurs de comparaison

signe	opération
==	égal à
!=	différent de
<, <=, >, >=	Inférieur, inférieur ou égal, supérieur, …

Les opérateurs standard

Les opérateurs logiques booléens

signe	opération
&&	ET logique
||	OU logique
!	Négation logique

Les opérateurs standard

- Les résultats des opérations de comparaison et des opérateurs logiques sont du type int :
  - La valeur 1 correspond à la valeur booléenne vrai
  - La valeur 0 correspond à la valeur booléenne faux
- Les opérateurs logiques considèrent toute valeur différente de zéro comme vrai et zéro comme faux:
  - 53 && 5.3 → 1
  - !78.22 → 0
  - 1 || !(11 > 56) → 0

Les opérateurs standard

Les opérateurs logiques sur les entiers bit à bit

signe	opération
&	ET
|	OU
^	OU exclusif
~	Complément à un
<<	Décalage à gauche
>>	Décalage à droite

Les opérateurs standard

Les opérateurs logiques sur les entiers bit à bit

- Exemple : Soit a = 77 (01001101) et b = 23 (00010111)

	Valeur	
Expression	Binaire	Décimale
a	01001101	77
b	00010111	23
a & b	00000101	5
a | b	01011111	95
a ^ b	01011010	90
~a	10110010	178
b << 2	01011100	92
b << 5	11100000	224
b >> 1	00001011	11

multiplication par 4
ce qui dépasse disparaît
division entière par 2

Les opérateurs standard

Opérateurs d'affectation

signe	opération		
+=	a = a+b;	⇔	a+= b;
-=	a = a-b;	⇔	a-= b;
*=	a = a*b;	⇔	a*= b;
/=	a = a/b;	⇔	a/= b;
%=	a = a%b;	⇔	a%= b;

Les opérateurs standard

Opérateurs d'incrémentation (++) et de décrémentation (--)

- `i = i+1;` ⇔ `i++;`
- `i = i-1;` ⇔ `i--;`

signe	opération
X=i++	passe d'abord la valeur de i à X puis incrémente i
X=i--	passe d'abord la valeur de i à X puis décrémente i
X=++i	incrémente d'abord i puis passe la valeur incrémentée à X
X=--i	décrémente d'abord i puis passe la valeur décrémentée à X

Les opérateurs standard

- Opérateurs d'incrémentation (++) et de décrémentation (--)
  - X=? N=10

Instruction	Résultat
X=N++;	N=11 et X=10
X=++N;	N=11 et X=11

Priorité des opérateurs

	Classes de priorités	Ordre de l'évaluation
Priorité 1 (la plus forte)	()	→
Priorité 2	! ++ --	←
Priorité 3	* / %	→
Priorité 4	+ -	→
Priorité 5	< <= > >=	→
Priorité 6	== !=	→
Priorité 7	&&	→
Priorité 8	||	→
Priorité 9 (la plus faible)	= += -= *= /= %=	←

Les expressions et les instructions

Les expressions

- Les **constantes** et les **variables** sont des expressions
- Les expressions peuvent être combinées entre elles par des opérateurs et forment ainsi des expressions plus complexes.
- Les expressions peuvent contenir des appels de fonctions
- Les expressions peuvent apparaître comme paramètres dans des appels de fonctions.

Les expressions et les instructions

Les expressions

- Exemples d'Expressions :

```
j=1
j++
printf(" Hello !\n")
N=pow(B,2)
(m+n)>=99
b=(i*2+j*5)+4
X!=Y
```

Les expressions et les instructions

Les instructions

- Si une expression comme j=0 ou j++ ou printf("Hello!\n") est suivie d'un point-virgule, elle devient une instruction
- Exemples:

```
j=1;
j++ ;
printf(" Hello !\n");
N=pow(B,2) ;
b=(i*2+j*5)+4;
```


Les expressions et les instructions

Evaluation des résultats

- En C toutes les expressions sont calculées (ou évaluées) pour retourner une valeur comme résultat
- Exemples :

```
(4-1!=3)        retourne la valeur 0 (faux)
X=2*3           retourne la valeur 6
```

- Les affectations sont aussi interprétées comme des expressions, donc on peut profiter de la valeur rendue par l'affectation

```
((B=5+1) == 7)
```


Les fonctions arithmétiques standard

Ces fonctions appartiennent à la bibliothèque **math.h**

- **sqrt :** racine carrée d'un entier ou d'un réel
  - *Exemple : sqrt(16) = 4*
- **floor et ceil** : arrondi d'un réel à l'entier le plus proche
  - *Exemple : floor(2.3) = 2 floor(4.7) = 4 inférieur*
    ceil (2.3) = 3 ceil (4.7) = 5 supérieur
- **pow** : puissance nième de x (x^n)
  - *Exemple : pow(2,3)=2^3=8*
- **exp(x)** : exposant de x (exp(x))

Les conversions de types

Les conversions de type automatiques : Calculs et affectations

- Si un opérateur a des opérandes de différents types, les valeurs des opérandes sont converties automatiquement dans un type commun.
- Ces **manipulations implicites** convertissent en général des types plus 'petits' en des types plus 'larges' pour ne perdre pas en précision.
- Lors d'une **affectation**, la donnée à droite du signe d'égalité est convertie dans le type à gauche du signe d'égalité.
  - Dans ce cas, il peut y avoir perte de précision si le type de la destination est plus faible que celui de la source.

Les conversions de types

Les conversions de type automatiques : Calculs et affectations

- Exemple : Considérons le calcul suivant:

```
int J = 7;
float B = 3,5;
double X;
X = J - B;
```

- Pour évaluer l'expression 'J-B', la valeur de J est convertie en float (le type le plus large des deux).
- Le résultat de la soustraction est du type float, mais avant d'être affecté à X, il est converti en double.
- Le résultat final est : X = 3,50 de type double

Les conversions de types

Les conversions de type forcées (casting) :

- Pour modifier explicitement le type d'un objet, on utilise l'opérateur de conversion de type, appelé cast ( )
- Syntaxe :

```
(<Type>) <Expression>
```

- Exemple :

```
int a = 5, b = 2, c;
float x;
c = a/b;   /* c = 2 */
x = a/b;   /* x = 2.0 */
x = (float) a/b;   /* x = 2.5 */
```


Exercice

Soit les déclarations suivantes :

```
int n = 10 , p = 4 ;
long q = 2 ;
float x = 1.75 ;
```

Donner le type et la valeur de chacune des expressions suivantes :

a) n + q
b) n + x
c) n % p +q
d) n < p
e) n >= p
f) n > q
g) q + 3 * (n > p)
h) q && n
i) (q-2) && (n-10)
j) x * (q==2)
k) x *(q=5)

Exercice

Solution:

```
int n = 10 , p = 4 ;
long q = 2 ;
float x = 1.75 ;
```

Expression	Valeur retournée	Type de retour
n + q	12	long
n+x	11,75	float
n%p+q	2+2=4	long
n<p	0	int
n>=p	1	int
n>p	1	int
q+3*(n>p)	2+3*1=2+3=5	long
q && n	1	int
(q-2) && (n-10)	0	int
x * (q==2)	1,75*1=1,75	float
x *(q=5)	8,75	float

Lecture et écriture de données

Lecture et écriture de données

1. Écriture formatée de donnée (printf)
2. Écriture non formatée (putchar/puts)
3. Lecture formatée de donnée (scanf)
4. Lecture non formatée (getchar/getch/gets)
5. Exercices

Lecture et écriture de données

La bibliothèque <stdio.h> contient un ensemble de fonctions qui assurent la communication de la machine avec l'utilisateur

Écriture de donnée :
- **printf()** : écriture formatée de donnée
- **putchar()** : écriture d'un caractère
- **puts()** : écriture d'une chaine de caractère

Lecture de donnée :
- **scanf()** : lecture formatée de données
- **getchar()** : lecture d'un caractère
- **gets()** : lecture d'une chaine de caractère

Écriture formatée de donnée (printf)

La fonction printf()

- **Description :** Permet l'affichage des messages et des variables de tous types
- **Syntaxe :**

```
printf("<Message>",<Expr1>,<Expr2>, ... )
```

Avec : <Message> est une chaîne de caractères qui peut contenir :

- du texte
- des séquences d'échappement (\n, \t, …)
- et/ou des spécificateurs de format

<Expr1> ,... : variables et expressions dont les valeurs sont à représenter

Écriture formatée de donnée (printf)

- **Les spécificateurs de format:** indiquent la manière dont les valeurs des expressions <Expr1>, ... sont imprimées

Symbole	Type	Afficher comme
%d ou %i	int	Entier relatif
%u	unsigned int	Entier non signé
%hd	short	Entier court
%ld	long	Entier long
%hu	unsigned short	Entier court
%lu	unsigned long	Entier long
%o	int	Entier exprimé en octal
%x	int	Entier exprimé en hexadécimal
%c	char	Caractère
%f	float ou double	Réel en notation décimale virgule fixe
%lf	long float ou long double	Réel long en notation décimale virgule fixe
%e	float ou double	Réel en notation exponentielle
%le	long float ou long double	Réel long en notation exponentielle
%s	char *	Chaîne de caractères

Écriture formatée de donnée (printf)

- **Les séquences d'échappement**

Code	Effet
\n	Fin de ligne
\t	Tabulation
\v	Tabulateur vertical
\f	Saut de page
\\	Barre oblique inverse
\?	Point d'interrogation
\'	Apostrophe
\"	Guillemet
\0	NULL

Écriture formatée de donnée (printf)

- Donner le résultat des instructions suivantes :

```
printf("%3d",n);
```

n = 20 ➡ ^20

n = 3 ➡ ^^3

n = 123 ➡ 123

```
printf("%10f",x);
```

x = 1.2345 ➡ ^^1.234500

x = 12.345 ➡ ^12.345000

Écriture formatée de donnée (printf)

- Exemple :

```
#include <stdio.h>
void main()
{
        char c;
        c = 'A';
        printf("%c", c); /* affichage du caractère A */
        printf("%d", c); /* affichage du code ASCII de A*/
}
```

- Affichage multiple :

```
printf("a=%d  r=%f  x=%c", a, r, x);
```

Écriture non formaté (puts/putchar)

L'affichage d'un seul caractère sur l'écran de l'ordinateur: putchar

- **Syntaxe** :

```
putchar(C);  /*afficher le contenu de l'objet C (de type char)*/
```

- Exemples :

```
putchar('x');   /*affiche la lettre x*/
putchar(65);   /*affiche le caractère de code Ascii 65*/
putchar('\n');  /*retour à la ligne*/
```


Écriture non formaté (puts/putchar)

L'affichage d'une chaine de caractères : puts

- **Syntaxe** :

```
puts(ch);  /*affichage d'une chaine de caractères*/
```

- Exemples :

```
puts("message");  /*affiche le mot « message » intégralement*/
```


Lecture formatée des données (scanf)

La fonction scanf

- **Description :** Permet la saisie des variables de tous types
- **Syntaxe :**

```
scanf("<format1><format2>...",<AdrVar1>,<AdrVar2>, ...)
```

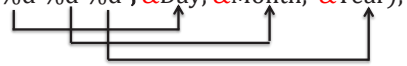
  - Le symbole & indique l'adresse de la variable qui lui est associée

Lecture formatée des données (scanf)

La fonction scanf

- Exemple:

```
int Day, Month, Year;
scanf("%d %d %d", &Day, &Month,  &Year);
```



  - Permet la saisie de trois entiers relatifs, séparés par des espaces, tabulations ou interlignes.
  - Les valeurs sont affectées respectivement aux trois variables Day, Month et Year

Lecture formatée des données (scanf)

La fonction scanf

```
#include<stdio.h>
void main()
{
    int A, B;
scanf("%4d%2d",&A,&B);
printf("%d et %d",A,B);
scanf("%d",&A);
printf("\n new =%d",A);
}
```

Valeur insérée:
12345678

Quel sera l'affichage à l'écran?

```
12345678
1234 et 56
 new =78
```


Lecture non formatée des données (getchar/getch/gets)

getchar() : la saisie d'un seul caractère, puis confirmation par 'Entrer'

- **Syntaxe** :

```
C=getchar(); /*Saisir le caractère C (de type char) puis confirmation par 'Entrer' */
```

getch() (de la bibliothèque **<conio.h>**): la saisie d'un seul caractère sans confirmation par 'Entrer'

- **Syntaxe :**

```
C=getch(); /* Saisir le caractère C (de type char) sans confirmation par 'Entrer'*/
```

gets(): la saisie d'une chaîne de caractères pouvant contenir des espaces et acceptant la chaîne vide, puis confirmation par 'Entrer'

- **Syntaxe :**

```
gets(chaine); /*Saisir la chaine de caractère « chaine » (de type char*)*/
```


Exercices

Écrire un programme permettant de saisir une date sous la forme :

- jj mm aa
- jj/mm/aaaa

Solution :

```
#include<stdio.h>
void main()
{
      int JOUR, MOIS, ANNEE;
      printf("Donner la date du jour : jj mm aaaa\n");
      scanf("%2d %2d %4d", &JOUR, &MOIS, &ANNEE);
      //scanf("%d/%d/%d", &JOUR, &MOIS, &ANNEE);
      printf("La date du jour est : %02d %02d %4d",JOUR,MOIS,ANNEE);
 }
```


Exercices

Écrire un programme permettant de saisir trois entiers le premier de 2 chiffres le 2ème de 3 chiffres et le 4ème de 4 chiffres

Solution :

```
#include<stdio.h>
void main()
{
      int A, B, C;
      printf("Donner trois entiers: ");
      scanf("%2d%3d%4d",&A,&B,&C);
      printf("A= %d, B=%d, C=%d",A,B,C);
 }
```

Exercices

- Écrire un programme permettant de saisir un caractère
  - En utilisant la touche entrée
  - Sans utiliser la touche entée
- **Solution :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char c1, c2;
        //En utilisant la touche entrée
        printf("Donner un caractère: ");
        c1=getchar(); /*scanf("%c",&c1);*/
        //Sans utiliser la touche entrée
        printf("Donner un autre caractère: ");
        c2=getch();
 }
```


Exercices

- Écrire un programme permettant d'afficher :
  - Le contenu d'une variable entière
  - Le contenu d'une variable réelle
  - Le contenu d'une variable de type caractère de deux manières
- **Solution :**

```
#include<stdio.h>
void main()
{
        int n;
        float x;
        char c;
        printf("Donner un entier: "); scanf("%d",&n);
        printf("Donner un réel: "); scanf("%f",&x); fflush(stdin);
        printf("Donner un caractère: "); c=getchar();
        printf("n=%d\n",n);
        printf("n=%f\n",x);
        printf("c=%c\n",c); /* printf("n=%d\nx=%f\nc=%c\n",n,x,c); */
        putchar(c);
}
```


Les structures de contrôle conditionnelles

Les structures de contrôle conditionnelles

Les structures de contrôle conditionnelles

Définition:

- Les structures de **contrôle conditionnelles** permettent d'effectuer des tests et d'exécuter un bloc d'instructions selon le résultat obtenu
  - L'instruction conditionnelle « if »
  - L'instruction de sélection multiple « switch »

L'instruction conditionnelle « if »

L'instruction conditionnelle « if » simple

- **Syntaxe:**

```
if(<condition>)
{
        <bloc d'instructions>
}
```

- La **<condition>** peut désigner :
  - une variable d'un type numérique,
  - une expression fournissant un résultat numérique.
- Cette **<condition>** est évaluée et le résultat est une valeur numérique:
  - Si ce résultat est égal à 1 ou différent de 0 (la condition est interprétée à vrai)
    → bloc d'instructions associé est exécuté
  - Si ce résultat est égal à 0 (la condition est interprétée à faux)
    → bloc d'instructions associé est non exécuté

L'instruction conditionnelle « if »

L'instruction conditionnelle « if » simple

- **Exemple 1:**

```
if(N)
        printf("%d est différent de zéro",N);
```

  - Si le bloc d'instructions contient une seule instruction, on peut éliminer les accolades **{ }**
    - *Si N=0 → l'instruction* `printf("%d est différent de zéro",N)` *n'est pas exécutée*
    - *Si N≠0 → l'instruction* `printf("%d est différent de zéro",N)` *est exécutée*

L'instruction conditionnelle « if »

L'instruction conditionnelle « if » simple

- **Exemple 2:**

```
if (p == q)
{
        printf ("p est égales à q\n");
}
```

  - Si **p** et q sont égales, on affiche la phrase "p est égales à q".

⚠ Faire la différence entre **p == q** avec **p = q**

  - **p == q** est une comparaison qui vérifie si p est égale à q.
  - **p = q** est une affectation, la valeur de la variable q est affecté à la variable p.

L'instruction conditionnelle « if » simple

- **Exemple 3:**

```
if (!(p == q))
{
      printf ("p est différent de q\n");
}
```

- L'ajout du NOT (!) à l'expression (p == q) inverse le résultat.
- Donc, la phrase "p est différent de q" est affichée si l'expression (p == q) n'est pas vérifiée, càd si **p** et **q** ne sont pas égaux.

L'instruction conditionnelle « if » simple

- **Exemple 4:**

```
if ((p == q) || (p > q))
{
      printf ("p n'est pas inférieur à q \n");
}
```

- Si **p** et **q** sont égales OU **p** est supérieur à **q**, on affiche la phrase "p n'est pas inférieur à q"

L'instruction conditionnelle « if » simple

- **Exercice 1:** Écrire un programme C qui saisie, calcule et affiche la valeur absolue d'un entier (sans utiliser la fonction ABS)

```
#include<stdio.h>
void main()
{
      int X;
      printf("Donner X ");
      scanf("%d", &X);
      if (X < 0)
            X = X * -1;
      printf("la valeur absolue = %d", X);
}
```


L'instruction conditionnelle « if » simple

- **Exercice 2:** Écrire un programme C qui permet de saisir les notes de deux matières, puis calculer et afficher la moyenne du module de ces deux matières si les deux notes saisies sont au dessus de 10.

```
#include<stdio.h>
void main()
{
      float N1, N2, Moy;
      printf("Donner N1 et N2 ");
      scanf("%f %f", &N1, &N2);
      if (N1>=10 && N2>=10)
      {
            Moy = (N1+N2)/2;
            printf("la moyenne de ce module = %f", Moy);
      }
}
```

L'instruction conditionnelle « if … else »

- **Syntaxe**

```
if(<condition>)
{
        <bloc d'instructions 1>
}
else
{
        <bloc d'instructions 2>
}
```

- La condition est évaluée dont le résultat est une valeur numérique:
  - Si ce résultat est égal à 1 ou différent de 0 (la condition est interprétée à vrai)
    → le bloc d'instructions 1 sera exécuté
  - Si ce résultat est égal à 0 (la condition est interprétée à faux)
    → le bloc d'instructions 2 sera exécuté

L'instruction conditionnelle « if … else »

- **Syntaxe**

```
if ( condition )
      <une seule instruction> ;
else
      < une seule instruction> ;
```

  - Les accolades peuvent être éliminées, si le bloc d'instructions associé à « if » ou à « else » contient une seule instruction

L'instruction conditionnelle « if … else »

- **Exemple**

```
if (p == q)
      printf (" p est égales à q\n");
else
      printf ("p est différent de q\n");
```

- **Si p et q sont égales :** la message "p est égales à q" sera affichée
- **Sinon:** on affiche "p est différent de q"

L'instruction conditionnelle « if … else »

- **Exercice 1 :** Ecrire un programme en C qui permet de lire à partir du clavier deux entiers et d'afficher l'entier le plus petit

```
#include<stdio.h> /* bibliothèque standard d'entrée sortie */
void main() /* fonction principale */
{
    int i, j; /* déclaration de deux entiers */
    printf("Saisissez deux entiers : ") ;
    scanf("%d %d",&i,&j); /* Saisie de 2 entiers */
    if(i < j)
   {
      printf("%d est plus petit que %d", i, j); /* Affichage de l'entier i s'il est le plus petit */
    }
    else
    {
      printf("%d est plus petit que %d", j, i); /* Affichage de l'entier j s'il est le plus petit */
    }
}
```

L'instruction conditionnelle « if … else »

- **Exercice 2 :** Ecrire un programme C permettant de :
  - Saisir un entier dans la zone identifiée par n1
  - Afficher si l'entier saisi est pair ou impair

```
#include<stdio.h>
void main()
{       int n1;
        printf("Donner un entier : ");
        scanf("%d",&n1); /* Saisie de l'entier n1 */

        if(n1%2==0)
                printf(" %d est un nombre pair \n",n1);
        else
                printf(" %d est un nombre impair \n",n1);
}
```


Les structures conditionnelles imbriquées

- **Syntaxe**

```
if(<condition 1>)
        if(<condition 2>)
        {
                <bloc d'instructions 1-1>
        }
        else
        {
                <bloc d'instructions 1-2>
        }
else
{
        <bloc d'instructions 2>
}
```


Les structures conditionnelles imbriquées

- **Syntaxe**

```
if(<condition 1>)
{
        <bloc d'instructions 1>
}
else
        if(<condition 2>)
        {
                <bloc d'instructions 2-1>
        }
        else
        {
                <bloc d'instructions 2-2>
        }
```


Les structures conditionnelles imbriquées

- **Exercice:** Ecrire un programme C qui permet de saisir un entier N et qui permet d'afficher s'il est positif, négatif ou nul.

Solution 1:

```
if(N>=0)
        if(N>0)
                printf("%d est positif\n",N);
        else
                 printf("%d est nul\n",N);
else
         printf("%d est négatif\n",N);
```

Solution 2:

```
if(N>0)
        printf("%d est positif\n",N);
else
        if(N<0)
                printf("%d est négatif\n",N);
        else
                printf("%d est nul\n",N);
```

Expression conditionnelle

Expression conditionnelle: le langage C possède une paire d'opérateurs qui peut être utilisée comme alternative à if – else

- **Syntaxe :**

```
<expr1> ? <expr2> : <expr3>
```

  - Si <expr1> fournit une valeur différente de zéro, alors la valeur de <expr2> est fournie comme résultat.
  - Si <expr1> fournit la valeur zéro, alors la valeur de <expr3> est fournie comme résultat

Expression conditionnelle

Exemple

- La suite d'instructions:

```
if (A>B)
    MAX=A;
else
    MAX=B;
```

→peut être remplacée par:

```
MAX = (A > B) ? A : B;
```


Structure conditionnelle switch

L'instruction **switch** : Permet d'examiner plusieurs cas en même temps afin de s'échapper des imbrications d'instructions « if...else »

Syntaxe :

```
switch(Variable)
{
     case Valeur_1 : Suite d'inst 1; break;
     case Valeur_2 : Suite d'inst 2; break;
     …
     case Valeur_n-1 : Suite d'inst n-1; break;
     case Valeur_n : Suite d'inst n; break;
     default : Suite d'inst 0;
}
```

- Si variable prend la valeur Valeur_1 alors on exécute la suite d'inst 1,
- Si elle prend la valeur Valeur_2 on exécute la suite d'inst 2, etc.

⋮

- Par défaut (càd, si aucune des valeurs ci-dessus ne correspond à la variable), alors on exécute la suite d'inst 0.

NB: n'oubliez surtout pas les **break** !

Structure conditionnelle switch

Exemple :

```
switch(N)
    {
        case 1: printf("one"); break;
        case 2: printf("two"); break;
        case 3: printf("three"); break;
        case 4: printf("four"); break;
        case 5: printf("five"); break;
        case 6: printf("six"); break;
        case 7: printf("seven"); break;
        case 8: printf("eight"); break;
        case 9: printf("nine"); break;
    }
```

→ Si on oublie tous les break ?

→ Dans ce cas, si N= 6 on obtient: sixseveneightnine. Ce qui n'est pas le résultat souhaité!

Structure conditionnelle switch

Exercice : Ecrire un programme qui permet de saisir le numéro de mois et d'afficher le nom d'un mois en fonction de son numéro.

```
switch ( numeroMois )
{
   case 1 : printf( " janvier " ) ; break ;
   case 2 : printf( " fevrier " ) ; break ;
   case 3 : printf( " mars " ) ; break ;
   case 4 : printf( " avril " ) ; break ;
   case 5 : printf( " mai " ) ; break ;
   case 6 : printf( " juin " ) ; break ;
   case 7 : printf( " juillet " ) ; break ;
   case 8 : printf( " aout " ) ; break ;
   case 9 : printf( " septembre " ) ; break ;
   case 10 : printf( " octobre " ) ; break ;
   case 11 : printf( " novembre " ) ; break ;
   case 12 : printf( " decembre " ) ; break ;
   default : printf( " Je connais pas ce mois . . . " ) ;
}
```


Exercices

Exercice 1 : Ecrire un programme C qui permet de vérifier si un caractère saisi par l'utilisateur est une voyelle ou non.

- En utilisant la structure conditionnelle (if...else)
- En utilisant la structure conditionnelle (switch)

Exercices

Solution 1: La structure conditionnelle if…else

```
#include<stdio.h>
void main()
{
       char c;
       printf("Saisissez un caractère : ");
       scanf("%c",&c);
       if(c== 'a' || c== 'e' || c== 'u'|| c== 'i'|| c== 'o' || c== 'y'||
          c== 'A' || c== 'E' || c== 'U'|| c== 'I'|| c== 'O' || c== 'Y')

             printf("%c est une voyelle", c);
       else
             printf("%c n'est pas une voyelle", c);
}
```


Exercices

Solution 2: La structure conditionnelle switch

```
#include<stdio.h>
main()
{
    char c;
    printf("Saisissez un caractère : ");
    scanf("%c",&c);
    switch(c)
    {
        case 'a' :
        case 'e' :
        case 'i' :
        case 'u' :
        case 'o' :
        case 'y' :
        case 'A' :
        case 'E' :
        case 'I' :
        case 'U' :
        case 'O' :
        case 'Y' : printf("%c est une voyelle", c);
        default : printf("%c n'est pas une voyelle", c);
    }
}
```

Exercices

Exercice 2 : Ecrire un programme C qui permet de saisir trois entiers et d'afficher l'entier le plus grand.

- En utilisant les structures conditionnelles imbriquées (if...else)
- En utilisant les expressions conditionnelles

Exercices

Solution 1: La structure conditionnelle if…else

```
#include<stdio.h>
void main()
{
        int a,b,c;
        printf("Entrez le 1er nombre :"); scanf("%d",&a);
        printf("Entrez le 2eme nombre :"); scanf("%d",&b);
        printf("Entrez le 3eme nombre :"); scanf("%d",&c);

        if(a >= b && a >= c)
        {
                printf("\nLe max est %d\n",a);
        }
        else if(b >= a && b >= c)
        {
                printf("\nLe max est %d\n",b);
        }
        else { printf("\nLe max est %d\n",c);}
}
```


Exercices

Solution 2: Les expressions conditionnelles

```
#include <stdio.h>
int main()
{
        int a,b,c,max;
        printf("donner trois entiers: ");
        scanf("%d %d %d",&a,&b,&c);
        max=(a>=b && a>=c)?a:(b>=a && b>=c)?b:c;
        printf(""\nLe max est %d\n",max);
        return 0;
}
```


Exercices

Exercice 3: Écrire un programme C permettant de calculer et d'afficher la somme, la soustraction, le produit **ou** la division de deux entiers a et b saisis au clavier. Le choix de l'opération est effectué par l'utilisateur:

'+' ou 'a' : addition

'-' ou 's' : soustraction

'*' ou 'm' : multiplication

'/' ou 'd' : division

- En utilisant la structure conditionnelle (if...else)
- En utilisant la structure conditionnelle (switch)

Solution 1: La structure conditionnelle if…else

```
#include <stdio.h>
void main()
{
	char operation;
	int r, a, b;
	printf("Entrez un signe d'operation:");
	scanf("%c" ,&operation);
	printf("Entrez  a: "); scanf("%d" ,&a);
	printf("Entrez  b: "); scanf("%d" ,&b);
	if(operation == '+' || operation == 'a') r=a+b;
	else if(operation=='*' || operation=='m') r=a*b;
	else if(operation=='/' || operation=='d') r=a/b;
	else if(operation=='-' || operation=='s') r=a-b;
	else printf("Non valide!");
	printf("%d %c %d = %d", a,operation,b,r);
}
```


Solution 2: La structure conditionnelle switch

```
#include <stdio.h>
void main()
{
	char operation;
	int r, a, b;
	printf("Entrez un signe d'operation:");
	scanf("%c" ,&operation);
	printf("Entrez  a: "); scanf("%d" ,&a);
	printf("Entrez  b: "); scanf("%d" ,&b);
	switch(operation)
	{
		case '+':
		case 'a': r=a+b; break;
		case '*':
		case 'm': r=a*b; break;
		case '/':
		case 'd': r=a/b; break;
		case '-':
		case 's': r=a-b; break;
		default: printf("Non valide!");
	}
	printf("%d %c %d = %d",a,operation,b,r);
}
```


Les structures de contrôle itératives

Les structures de contrôle itératives

Définition:

- Les structures itératives (ou structures répétitives) permettent d'exécuter des instructions en boucles soit pour un nombre d'itérations fixé à l'avance, soit jusqu'à ce qu'une condition soit remplie.
  - Structure itérative « for »
  - Structure itérative « while »
  - Structure itérative « do… while »

La structure itérative « for »

Cette forme de boucle est utilisée si le nombre de répétitions du bloc d'instructions est connu d'avance

Syntaxe:

```
for( <init.> ; <cond. répétition> ; <compteur> )
{
        <bloc d'instructions>
}
```

- **<init.>** est utilisée pour initialiser les variables utilisées dans le bloc d'instructions
- **<cond. répétition>** est évaluée pour décider si le bloc d'instructions formant le corps de la boucle doit être répétée ou non
- **<compteur>** est utilisé pour incrémenter ou décrémenter les compteurs utilisés
- Ces trois éléments sont séparés par des points-virgules ";"

La structure itérative « for »

Une instruction :

```
for( V=Vi ; V<=Vf ; V++ )
   Une seule instruction;
```

Une suite d'instruction :

```
for( V=Vi ; V<=Vf ; V++ )
{
   Une suite d'instructions;
   instruction 2;
   instruction 3;
}
```


La structure itérative « for »

Exemple 1:

1. Ecrire un programme qui affiche les lettres alphabétiques triées par ordre croissant

```
#include <stdio.h>
void main()
{
      int i;
      for (i=65;i<=90;i++)
            printf(" %c ",i);
}
```

2. Changer le programme pour donner l'affichage suivant

A b C d E f G h I j K l M n O p Q r S t U v W x Y z

La structure itérative « for »

Exemple 1:

2.

```
#include <stdio.h>
   void main()
   {
         int i;
         for (i=65;i<=90;i++)
              if(i%2==0)
                   printf(" %c ",i+32);
              else
                   printf(" %c ",i);
    }
```


La structure itérative « for »

Exemple 2: Écrire un programme C permettant d'afficher la somme des 10 entiers saisis par l'utilisateur.

```
#include<stdio.h>
 void main()
 {     int i, n, som=0;
       for (i=1;i<=10;i++)
        {
             printf("donner un entier");
             scanf("%d", &n);
             som=som+n;
        }
       printf("la somme = %d", som);
  }
```


La structure itérative « for »

Exemple 3:

1. Écrire un programme C permettant d'afficher si un nombre est premier. Un nombre est dit premier s'il n'est divisible que par un et lui-même.
2. Changer le programme pour qu'il affiche les 100 premiers nombres premier :

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

La structure itérative « for »

Exemple 3:

1.

```
#include <stdio.h>
void main()
{
      int i,N,nb;
      printf("Donner N: "); scanf("%d",&N);
      for(i=1,nb=0;i<=N;i++)
            if(N%i==0)
                  nb++;
      if(nb==2)
            printf(" %d est premier" ,N);
      else
            printf(" %d n'est pas premier" ,N);
      }
 }
```

La structure itérative « for »

Exemple 3:

```
2.  #include <stdio.h>
    void main()
    {
          int N,i,nb;
          for(N=1;N<100;N++)
          {
                nb=0;
                for(i=1;i<=N;i++)
                      if(N%i==0)
                            nb++;
                if(nb==2)
                      printf(" %d ",N);
          }
     }
```


La structure itérative « while »

Cette forme est très similaire à la précédente sauf qu'elle permet de construire une structure pour laquelle le bloc d'instructions à répéter peut éventuellement n'être jamais exécuté car la condition est vérifiée avant le bloc.

Syntaxe:

```
while(<expression logique>)
{
   <bloc d'instructions>;
}
```

- Les accolades peuvent être omis si le bloc d'instruction comprend une seule instruction

```
while( <expression logique> )
   <Une seule instruction>;
```


La structure itérative « while »

Syntaxe:

```
while( <expression logique>)
{
   <bloc d'instructions>;
}
```

- Tant que l'`<expression logique>` fournit la valeur vrai, le `<bloc d'instructions>` est exécuté
- Si l'`<expression logique>` fournit la valeur faux, l'exécution continue avec l'instruction qui suit l'instruction **while**
- Le `<bloc d'instructions>` est exécuté zéro ou plusieurs fois

La structure itérative « while »

Exemple: Ecrire un programme C permettant de vérifier Si X est un diviseur de Y (sans utiliser MOD). X et Y sont saisis par l'utilisateur.

```
#include <stdio.h>
void main()
{
     int X,Y;
     printf("Donner la valeur de X:"); scanf("%d",&X);
     printf("Donner la valeur de Y:"); scanf("%d",&Y);
    while(y>0)
           y-=X;
     if(Y<0)
           printf("non diviseur");
     else
           printf(" diviseur");
}
```

La structure itérative « while »

Exemple: Écrire un programme C qui convertit un nombre de minutes en un nombre d'heures et de minutes (sans utiliser DIV et MOD).

- *Exp : 320 mn = 05 h : 20 mn*

```
#include<stdio.h>
void main()
{
    int NB,M,H;
    printf("donner un nombre");
    scanf("%d",&NB);
    H=0;
    M=NB;
    while(M>=60)
    {
           M=M-60;
           H=H+1;
    }
      printf("%d mn = %02d h : %02d mn", NB, H, M);
 }
```


La structure itérative « do…while »

Cette forme permet de construire une structure répétitive dans laquelle la condition de rebouclage est vérifiée à la fin : on est donc certain d'exécuter au moins une fois le bloc d'instruction à répéter.

Syntaxe:

```
do
{
     <bloc d'instructions>;
}while(<expression logique>);
```

- Les accolades peuvent être omis si le bloc d'instruction comprend une seule instruction

```
do
     <bloc d'instructions>;
while(<expression logique>);
```


La structure itérative « do…while »

Exemple: Lecture d'une valeur strictement positive.

```
#include<stdio.h>
void main()
{
    int X;

    do
    {
          printf("donner un nombre X strictement positif");
          scanf("%d", &X);
    }
    while(!(X>0)); //while(X<=0);
}
```


La structure itérative « do…while »

Exemple: Écrire un programme C permettant de saisir un entier N avec 5≤N≤20.

```
int N;
do
{
      printf("Donner un entier entre 5 et 20 : ");
      scanf("%d",&N);

}while(N<5 || N>20);
```

La structure itérative « do…while »

Exemple: Écrire un programme C permettant de saisir une lettre majuscule.

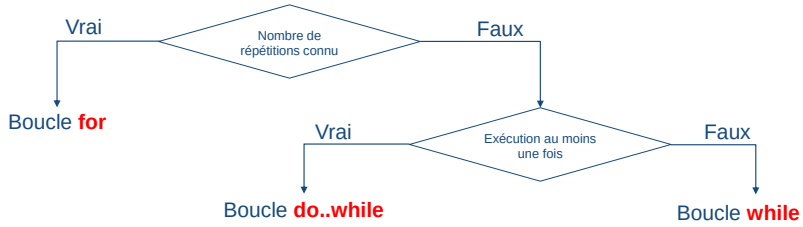
```
char c;
do
{
      printf("Donner une lettre en majuscule: ");
      scanf("%c",&c);

}while(c<65 || c>90); //while(c<'A' || c>'Z');
```


Choix de la structure répétitive adéquate

Le choix de la structure répétitive appropriée dépend du problème à résoudre:

- La boucle **FOR** est la plus adéquate si le nombre de répétitions est **connu**
- La boucle **DO..WHILE** est la plus recommandée si le nombre d'itérations varie de **1** à **n**
- La boucle **WHILE** est la plus appropriée si le nombre d'itérations varie de **0** à **n**



Choix de la structure répétitive adéquate

Bilan

Boucle	for	while	do..while
Nombre de répétition	Connu à l'avance	Inconnu à l'avance	Inconnu à l'avance
	0, 1 ou N fois	0, 1 ou N fois	1 ou N fois

Les ruptures de séquence

La rupture de séquence consiste à interrompre l'exécution d'une boucle alors que la condition de passage est encore valide.

- L'instruction break
- L'instruction continue
- L'instruction goto

Les ruptures de séquence

L'instruction break: interrompe le déroulement de la boucle et passe à la première instruction qui suit la boucle.

```
#include<stdio.h>
Void main()
{
   int i;
   for (i=1;i<=5;i++)
   {
      if (i == 4)
           break;
      printf("i = %d\n", i);
   }
   printf("i à la sortie de la boucle = %d\n", i);
}
```

```
i = 1
i = 2
i = 3
i à la sortie de la boucle = 4
```


Les ruptures de séquence

L'instruction break

- **Remarque :** En cas de boucles imbriquées, break fait sortir de la boucle la plus interne

```
#include<stdio.h>
 void main()
 {
    int i, j;
    for (i=1;i<=3;i++)
       for (j=1;j<=5;j++)
       {
          if (j == 3)
                break;
          printf("i = %d et j = %d\n", i, j);
       }
    printf("i et j à la sortie des boucles=%d et %d\n", i, j);
 }
```

```
i = 1 et j = 1
i = 1 et j = 2
i = 2 et j = 1
i = 2 et j = 2
i = 3 et j = 1
i = 3 et j = 2
i et j à la sortie des boucles = 4 et 3
```


Les ruptures de séquence

L'instruction continue: permet de passer directement au tour de boucle suivant, sans exécuter les autres instructions de la boucle.

```
#include<stdio.h>
 void main()
 {
    int i;
    for (i=1; i<=5 ; i++)
    {
         if (i == 4)
              continue;
         printf("i = %d\n", i);
    }
    printf("i à la sortie de la boucle = %d\n", i);
 }
```

```
i = 1
i = 2
i = 3
i = 5
i à la sortie de la boucle = 6
```


Les ruptures de séquence

L'instruction goto: permet d'effectuer un saut jusqu'à l'instruction étiquette appelée label.

- **Un label** est une chaîne de caractères suivie du double point « : ».

```
#include<stdio.h>
 void main()
 {
    int i;
    for (i=1; i<=5 ; i++)
    {
       if (i == 4)
            goto fin;
       printf("i = %d\n", i);
    }
    fin : printf("i à la sortie de la boucle = %d\n", i);
 }
```

```
i = 1
i = 2
i = 3
i à la sortie de la boucle = 4
```

Exercice 1

Ecrire un programme C qui permet de saisir un nombre entier strictement positif N, lire N nombre réels quelconques puis calculer et afficher le plus grand (pg) et le plus petit (pp) de ces réels.

Solution

```
#include <stdio.h>
void main()
{
        int N, i;
        float nb,min,max;
        do
        {
                printf("Donner N:");
                scanf("%d",&N);
        }while(N<=0);
        for(i=1;i<=N;i++)
        {
                printf("Entrer le nombre %d:",i);
                scanf("%f",&nb);
                if(i==1)
                {       min=nb;
                        max=nb;
                }
                else
                {       if(nb<min)
                                min=nb;
                        if(nb>max)
                                max=nb;
                }
        }
        printf("Le plus grand reel est:%.2f",max);
        printf("Le plus petit reel est:%.2f",min);
}
```


Exercice 2

Ecrire un programme C qui permet de saisir un entier N strictement positif et Calculer la somme des N premiers termes de la série harmonique :

$$1 + 1/2 + 1/3 + ... + 1/N$$

Solution

```
#include<stdio.h>
main( )
{
   int i, N;
   float S = 0, Q; // Initialisation de la série
   do
   {
        printf("Donner N:");
        scanf("%d",&N); // Saisie de N
   }while(N<=0);

   for( i=1 ; i <= N ; i++)
   {
        Q=(float)1/i; // Conversion de type
        S = S + Q;
   }
   printf ("Somme= %f ", S) ;
}
```

Les Tableaux

Les tableaux

Les tableaux

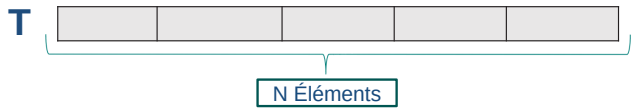
Définition:

- Un tableau est un ensemble ordonné d'éléments de même nature ; ils sont désignés par un nombre [0, N − 1].
- Un tableau peut être :
  - unidimensionnel appelé vecteur
  - bidimensionnel appelé matrice

Les tableaux unidimensionnels

Définitions

- Un tableau unidimensionnel **T** est une zone composée de **N** variables de même type regroupées sous le même nom.



- Le nombre d'éléments N (entier) est appelé la dimension du tableau.
- L'ensemble des éléments de T peuvent être :
  - Simples (entier, réel, caractère)
  - Composée (Structure, chaines de caractères)

Les tableaux unidimensionnels

Déclaration: avant d'être utilisé tout tableau doit être déclarer

- **Syntaxe**

```
<TypeSimple> <NomTableau>[<Dimension>];
```

Le type peut être :

- **Entier :** short, int, long, unsigned short, unsigned int, unsigned long,
- **Réel :** float, double, long double
- **Caractère :** char

- Les noms des tableaux sont des identificateurs qui doivent correspondre aux restrictions définies précédemment
- Exemples de déclarations :

int Tab[20]; float TabMoyenne[30];

Les tableaux unidimensionnels

Mémorisation:

- Le nom d'un tableau est l'adresse du premier élément du tableau.
- Les adresses des autres composantes sont calculées relativement à cette adresse d'une manière automatique
- Si un tableau est formé de N composantes et si une composante a besoin de M octets en mémoire, alors le tableau occupera de N*M octets.
- Exemple : **`int TAB[20];`**
  - Une variable du type int occupe 4 octets (c.-à-d: sizeof(int)=4),
  - pour le tableau TAB déclaré par: int TAB[20]; C réservera N*M = 20*4 = 80 octets en mémoire.

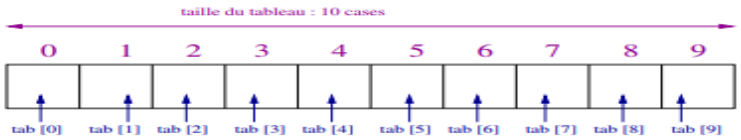
Les tableaux unidimensionnels

Exemple :

```
float tab [10];
```

Remarques :

- L'indice du premier élément est toujours **0**
- L'indice du dernier élément est alors **dim-1**



Les tableaux unidimensionnels

Initialisation

- On peut initialiser les composantes du tableau, lors de la déclaration d'un tableau, en indiquant la liste des valeurs respectives entre accolades.
  - Exemples:

```
int T[4] = {11, 33, 55, 77};   // T = 11    33    55    77
```

- Il faut faire attention au nombre de valeurs dans la liste qui doit correspondre à la dimension du tableau.
- Si la liste ne contient pas assez de valeurs pour toutes les composantes, les composantes restantes sont initialisées par zéro.

```
int TAB[5]={11,33};    // TAB=   11    33    0    0    0
```

Les tableaux unidimensionnels

Réservation automatique

- Si la dimension n'est pas indiquée explicitement lors de l'initialisation, alors l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

  - ***Exemples***

```
int T[] = {11, 33, 55, 77};
```

→ réservation de 4*sizeof(int) octets =16 octets

```
int TAB[4] = {11, 22, 33, 44, 55};
```

→ erreur

Les tableaux unidimensionnels

Accès aux éléments :

- **Syntaxe :**

```
Nom[indice]
```

- Où indice peut être :
  - une variable simple : tab[i]
  - une constante : tab[2]
  - une expression arithmétique : tab[2*i]
- L'indice doit être une valeur entière comprise entre 0 et N-1 (N est la taille du tableau).

Les tableaux unidimensionnels

Exemples :

```
//Affecter la valeur 17 au 2ème élément
Tab[1] = 17 ;

//Saisie de la valeur du 5ème élément
scanf("%f", &Tab[4]);

//Affichage de la valeur du 5ème élément
printf("%f", Tab[4]);

//ajout de 2 à la valeur contenue dans la case numéro 4   du tableau
Tab[3] = Tab[3] + 2 ;
```


Les tableaux unidimensionnels

Saisie des éléments du tableau de réels

```
void main()
{
   int  i,
   float Tab[10];
   printf("Donner les elements du Tableau Tab: \n");
   for(i=0;i<10;i++)
   {
         printf("T[%d]= ",i);
         scanf("%f",&Tab[i]);
   }
}
```

Affichage des éléments du tableau

```
printf("Tab: \t");
for(i=0;i<=9;i++)
      printf("%.2f\t",Tab[i]);
```


Exercice:

Écrire un programme qui permet de :

- Lire la taille du tableau **N** avec **0<N≤10**
- Lire et afficher un tableau **T** de **N** éléments.
- Calculer at afficher la somme des éléments de **T**.
- Chercher et afficher le maximum et le minimum de **T**.

Solution:

```
void main()
{
    int i, N, T[10], som, min, max;
    do{
          printf("Donner le nombre d'elements: ");
          scanf("%d",&N);
    }while(N<=0 || N>10);

    printf("Donner les elements du tableau T: \n");
    for(i=0;i<N;i++)
    {
          printf("T[%d]= ",i);
          scanf("%d", &T[i]);
    }
```


Solution :

```
    printf("T= \t");
    for(i=0,som=0;i<N;i++)
    {
            printf("%d\t",T[i]);
            som+=T[i];
    }
    printf("La somme des éléments du tableau=%d\n",som);
    min=T[0];
    max=T[0];
    for(i=1;i<N;i++)
    {
            if(T[i]< min)
                    min= T[i];
            if(T[i]>max)
                    max= T[i];
    }
    printf("La valeur minimale du tableau=%d\n",min);
    printf("La valeur maximale du tableau=%d\n",max);
}
```

Les tableaux à deux dimensions

Définitions :

- En C, **un tableau à deux dimensions** est à interpréter comme un tableau (uni-dimensionnel) de dimension L dont chaque composante est un tableau (uni-dimensionnel) de dimension C.
- On appelle L le nombre de lignes du tableau et C le nombre de colonnes du tableau.
- L et C sont alors les deux dimensions du tableau.
- Un tableau à deux dimensions contient donc L*C composantes.

Les tableaux à deux dimensions

Déclaration :

- **Syntaxe :**

```
<Type> <Nom>[<Dim1>][<Dim2>];
```

Avec

Type : Type des éléments du tableau.
Nom : Nom du tableau
Dim1 : Nombre de lignes
Dim2 : Nombre de colonnes

- **Représentation d'une matrice**

```
float M[3][4];
```

M

	0	1	2	3
0	M[0][0]	M[0][1]	M[0][2]	M[0][3]
1	M[1][0]	M[1][1]	M[1][2]	M[1][3]
2	M[2][0]	M[2][1]	M[2][2]	M[2][3]

Les tableaux à deux dimensions

Déclaration:

- **Exemples**
  - Tableau d'entiers :

```
int A[10][10];      ou bien   long A[10][10];
```

  - Tableau de réel :

```
float B[2][20];     ou bien   double B[2][20];
```

  - Tableau de caractères :

```
char C[15][40];
```


Les tableaux à deux dimensions

Mémorisation

- Comme pour les tableaux à une dimension, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau (c.-à-d. l'adresse de la première ligne du tableau).
- Les composantes d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire.
- Exemple :

```
short TAB[2][4];
```

  - Une variable du type short occupe 2 octets (c.-à-d: sizeof(short)=2),
  - pour le tableau TAB déclaré par: short TAB[2][4]; C réservera 2*4*taille = 2*4*2 = 16 octets en mémoire.

Les tableaux à deux dimensions

Initialisation

- On peut initialiser les composantes du tableau, lors de la déclaration d'un tableau, en indiquant la liste des valeurs respectives entre accolades.
- A l'intérieur de la liste, les composantes de chaque ligne du tableau sont encore une fois comprises entre accolades.
  - Exemples:

```
int MAT[3][2] ={{11,22},{55,66},{88,99}};
```

- Lors de l'initialisation, les valeurs sont affectées ligne par ligne en passant de gauche à droite.
- S'il y a des valeurs manquantes, ils seront initialisées par zéro.

Les tableaux à deux dimensions

Réservation automatique

- Si les dimensions ne sont pas indiquées explicitement lors de l'initialisation, alors l'ordinateur réserve automatiquement le nombre d'octets nécessaires.
- Exemples:

```
int M[][] ={{10,20},{30,40},{50,60}};
```

→ réservation de 3*2*sizeof(int) octets = 24 octets

```
int TAB[3][2] = {{11, 22, 33, 44},{55, 66, 77, 88}};
```

→ erreur

Les tableaux à deux dimensions

Accès aux éléments :

- **Syntaxe :**

```
Nom[ind_Ligne][ind_Colonne]
```

- Considérons un tableau T de dimensions L et C.
  - les indices du tableau varient de 0 à L-1 et de 0 à C-1.
  - la composante de la N^{ième} ligne et M^{ième} colonne est notée: A[N-1][M-1]

Les tableaux à deux dimensions

Accès aux éléments :

- **Exemples :**

```
//Affecter 99 à l'élément de la ligne 3 et de la colonne 4
M[2][3] = 99;

//Saisie de l'élément de la ligne 2 et de la colonne 1
scanf("%f", &M[1][0]);

// Affichage de l'élément de la ligne 1 et de la colonne 4
printf("%f", M[0][3]);
```

Saisie des éléments de la matrice

```
void main()
{
    int i, j, M[10][10];

    printf("Donner les elements de la matrice M: \n");
    for(i=0;i<10;i++)
    {
           for(j=0;j<10;j++)
           {
                  printf("M[%d][%d]= ",i,j);
                  scanf("%d",&M[i][j]);
           }
    }
}
```


Affichage des éléments de la matrice

```
printf("M:\n");
for(i=0;i<10;i++)
{
      for(j=0;j<10;j++)
      {
            printf("%d\t",M[i][j]);
      }
      printf("\n");
}
```


Exercice:

Écrire un programme qui permet de :

- Saisir la taille **L*C** d'une matrice avec 0 < **L** ≤ 10 et 0 < **C** ≤ 10
- Lire et afficher une Matrice **M** de **L** lignes et **C** colonnes.
- Calculer et afficher la somme des éléments de **M**.
- Chercher et afficher le maximum et le minimum de **M**.

Solution:

```
void main()
{
    int i, j, L, C, M[10][10], som, min, max;
    do{
            printf("Donner le nombre de ligne: ");
            scanf("%d",&L);
    }while(L<=0 || L>10);
    do{
            printf("Donner le nombre de colonne: ");
            scanf("%d",&C);
    }while(C<=0 || C>10);
    printf("Donner les elements de la matrice M: \n");
    for(i=0;i<L;i++)
            for(j=0;j<C;j++)
            {
                    printf("M[%d][%d]= ",i,j);
                    scanf("%d",&M[i][j]);
            }
```

Solution:

```
printf("M= \n");
for(i=0,som=0;i<L;i++)
{
        for(j=0;j<C;j++)
        {
                printf("%d\t",M[i][j]);
                som+=M[i][j];
        }
        printf("\n");
}
printf("La somme des éléments de la matrice=%d\n",som);
```


Solution:

```
        min=M[0][0];
        max=M[0][0];
        for(i=0;i<L;i++)
        {
                for(j=0;j<C;j++)
                {
                        if(M[i][j]<min)
                                min= M[i][j];
                        if(M[i][j]>max)
                                max= M[i][j];
                }
        }
        printf("La valeur minimale de la matrice=%d\n",min);
        printf("La valeur maximale de la matrice=%d\n",max);
}
```

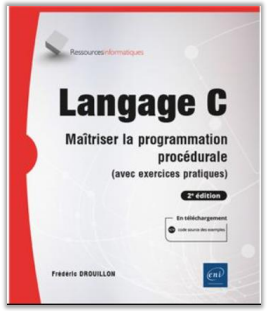

Bibliographie

Ce cours a été réalisé en se basant sur les références suivantes :

- Livre « **Le Livre Du C Premier Langage** » par Claude Delannoy.

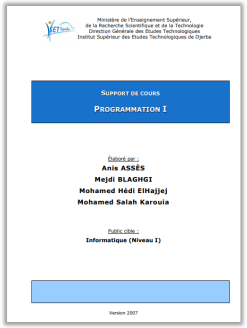
Bibliographie

- **Ce cours a été réalisé en se basant sur la référence suivante :**
  - Livre « **Langage C: Maîtriser la programmation procédurale (avec exercices pratiques** » par Frédéric Drouillon.



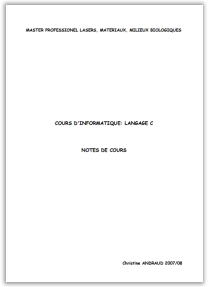
Bibliographie

- **Ce cours a été réalisé en se basant sur les références suivantes :**
  - Support de cours «**Programmation I**» par Anis Assès, Mejdi Blaghgi, Mohamed Hédi ElHajjej et Mohamed Salah Karouia.



Bibliographie

- **Ce cours a été réalisé en se basant sur les références suivantes :**
  - Support de cours «**COURS D'INFORMATIQUE: LANGAGE C**» par Christine ANDRAUD.



TO BE CONTINUED ...