- ***CHAPITRE 3*** **: Types de base, Opérateurs et Expressions**

- **1. Types simples**

  - *Un type* définit l'ensemble des valeurs que peut prendre une variable, le nombre d'octets à réserver en mémoire et les opérateurs que l'on peut appliquer dessus.

  - En *C*, il n'y a que deux types de base :

    - ***les entiers*** **et**
    - ***les réels*.**

Chap. 3 : 1. Types simples

- **1.1 Types entiers :**
  - 4 variantes d'entiers : *caractères* (char), *entiers courts* (short int), *entiers longs* (long int) et *entiers standards* (int).
  - Caractéristiques *:*

Définition	*Valeur minimale*	*Valeur maximale*	*Nombre d'octets*
Char	-128	127	1
short int	-32768	32767	2
Int	-32768	32767	2
long int	-2147483648	2147483647	4

  - Remarques :
    - Un caractère est un nombre entier (il s'identifie à son code *ASCII*). Par conséquent, une variable de type char peut contenir une valeur entre -128 et 127 et elle peut subir les mêmes opérations que les variables du type short, int ou long.
    - Un nombre entier de type int est souvent représenté sur *1 mot machine* (16 bits ou 32 bits). Dans notre cas, 1 mot machine = 16 bits.
    - Si l'on ajoute le préfixe unsigned à l'une de ces variantes, alors on manipule des entiers non signés :
      - **unsigned char** : indique des valeurs entières entre 0 et 255.
      - **unsigned int** (resp. short) : entre 0 et 65535.
      - **unsigned long** : entre 0 et 4294967295.

Chap. 3 : 1. Types simples

- **1.2 Types réels :**
  - 3 types de réels :
    - réels simple précision (**float**),
    - réels double précision (**double**) et
    - réels très grande précision (**long double**).
  - Caractéristiques :

Définition	*Précision*	*Répartition des bits S , M , E*	*Domaine (approximatif)*	*Nombre d'octets*
float	Simple	**1 , 23 , 8**	$\pm 1{,}1\ 10^{-38}$ à $\pm 3{,}4\ 10^{38}$	4
double	Double	**1 , 52 , 11**	$\pm 2{,}2\ 10^{-308}$ à $\pm 1{,}7\ 10^{308}$	8
long double	Très grande	**1 , 64 , 15**	$\pm 3{,}4\ 10^{-4932}$ à $\pm 1{,}1\ 10^{4932}$	10

Chap. 3 : 2. Déclaration des variables simples

Les *variables* et les *constantes* sont les données principales manipulées par un programme.

- Les variables :
  - Les *déclarations* introduisent les variables, fixent leur type et parfois aussi leur valeur de départ (initialisation).
  - Syntaxe de déclaration *:*

    <type> <NomVar1>, <NomVar2>, ..., NomVarN> ;
  - Exemple en *C* :
    - int x, y ;
    - short compteur ;
    - float hauteur, largeur ;
    - double r ;
    - char touche ;

Chap. 3 : 2. Déclaration des variables simples

- **Les constantes littérales :**

  Dans un programme C, on peut manipuler les constantes littérales en nombre de 4 :

  - constantes entières,
  - constantes réelles,
  - constantes caractères et
  - constantes chaînes de caractères.

Chap. 3 : 2. Déclaration des variables simples

- **2.1 Les constantes entières**
  - sous forme décimale : 100, 255.
  - sous forme octale, en faisant précéder le nombre par le caractère 0 (zéro) : 0144, 0377.
  - sous forme hexadécimale, en faisant précéder le nombre par 0x ou 0X : 0x64, 0Xff
  - **Remarques** :
    - Le type attribué à une constante est automatique (C choisit la solution la plus économique)
    - On peut forcer l'ordinateur à attribuer à la constante entière un type de notre choix, en employant l'un des suffixes suivants :

Suffixe	*Type*	*Exemple*
u ou **U**	**unsigned (int**	550u
l ou **L**	**long**	123456789L
ul ou **UL**	**unsigned long**	12092ul

Chap. 3 : 2. Déclaration des variables simples

- **2.2 Constantes réelles :**
  - en notation décimale : 123.4
  - en notation exponentielle : 1234e-1 ou bien 1234E-1
  - **Remarques :**
    - Par défaut, les constantes réelles sont du type double.
    - Le suffixe f ou F force l'utilisation du type float.
    - Le suffixe l ou L force l'utilisation du type long double.

Chap. 3 : 2. Déclaration des variables simples

- **2.3 Constantes caractères** :
  - Sont toujours indiqués entre apostrophes ' '
  - Exemples :
    - 'a' ; 'b' ; 'A' ; '+' ; ',' ; …
  - La valeur d'un caractère constant est son code ASCII. Les caractères constants peuvent donc apparaître dans des opérations arithmétiques ou logiques.
  - Ainsi, l'expression : 'a' + '?' vaut 160
    (le code ASCII de 'a' est égal à 97 alors que celui de '?' est 63)
  - Pour distinguer certains caractères spéciaux (les caractères de contrôle ou caractères non imprimables), on utilise le signe \ (antislash) → tableaux

Chap. 3 : 2. Déclaration des variables simples

- **Séquences d'échappement** :

Caractère de contrôle	*Signification*
'\a'	Bip sonore
'\b'	Retour arrière (*back space*)
'\t'	Tabulation horizontale
'\n'	Passage à la ligne suivante
'\r'	Retour chariot
'\0'	Caractère nul
'\\'	Trait oblique (*antislash*)
'\?'	Point d'interrogation
'\''	Apostrophe
'\"'	Guillemets
'\f'	Saut de page
'\v'	Tabulation verticale

Chap. 3 : 2. Déclaration des variables simples

- **2.4 Constantes chaînes de caractères :**
  - Sont représentées entre guillemets **" "**.
  - **Exemple :** "Ceci est une chaîne de caractère**"**
  - Le **compilateur C** rajoute à la fin de toute chaîne de caractères le caractère nul '**\0**' pour indiquer sa fin.
  - Dans une chaîne de caractère, on peut utiliser les séquences d'échappement. L'instruction suivante :

    printf("Bonjour, \n\tcomment vas-tu?\n")

    produit la sortie :

    Bonjour,

    comment vas-tu?

Chap. 3 : 2. Déclaration des variables simples

- **2.5 Initialisation des variables :**
  - En **C**, il est possible d'initialiser les variables à la déclaration.
  - **Exemples :**
    - **int** max = 1023 ;
    - **char** tabulation = '\t' ;
  - En utilisant l'attribut **const**, la valeur d'une variable ne change pas au cours du programme : C'est **une constante**.
  - **Exemples :**
    - **const int** MAX = 767 ;
    - **const char** NEWLINE = '\n' ;

Chap. 3 : 3. Opérateurs Standards

- Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ...

3.1 Opérateurs classiques

3.1.1 Opérateur d'affectation =

<variable> = <expression> ;

- L'expression est évaluée puis affectée à la variable.
- **Exemples :**
  - const int LONG = 141 ; /* affectation de valeurs constantes */
  - char NATION = 'M' ;
  - int val, resultat ;
  - char lettre ;

```
val = LONG ;                /* affectation de valeurs de variables */
lettre = NATION ;
resultat = 45 + 5 * val ;   /* affect. de valeurs d'expressions */
```

Chap. 3 : 3. Opérateurs Standards

- **3.1.2 Opérateurs arithmétiques**
  - +- * /
  - L'opérateur % permet d'obtenir le reste de la division entière.
  - L'opérateur / retourne un quotient entier si les deux opérandes sont entiers.

- **3.1.3 Opérateurs logiques**

  && : ET logique (and)
  || : OU logique (or)
  ! : négation logique (not)

  - S'appliquent à des expressions booléennes (**0** si **faux** et **valeur non nulle** si **vrai**)
  - ET retourne la valeur **1** si les deux opérandes sont non nuls, et **0** sinon.
  - OU retourne la valeur **1** si au moins un des opérandes est non nul, et **0** sinon.

  - **Exemples**
    - L'expression : **32 && 40** vaut **1**
    - L'expression : !65.34 vaut **0**
    - L'expression : !!0 vaut **0**

Chap. 3 : 3. Opérateurs Standards

3.1.3 Opérateurs de comparaison

- = = : l'opérateur égal à.
- **!=** : l'opérateur différent de.
- **<, <=, >, >=** : plus petit, plus petit ou égal, plus grand, plus grand ou égal.
- Ces opérateurs retournent la valeur **0** si la comparaison est fausse et **1** sinon
- Exemple

0 || !(32 > 12) retourne la valeur 0.

Chap. 3 : 3. Opérateurs Standards

3.1.4 Opérateurs de bits

- Ils travaillent sur les **bits**.
- Les **opérandes** doivent être de type **entier** (char, short, int, long, signés ou non).
- **Opérateurs de décalage de bits**

>> : décalage à droite. << : décalage à gauche.

  - L'opérande gauche constitue l'objet à décaler et l'opérande droit le nombre de bits de décalage.
  - Si la quantité à décaler est signée alors le bit signe (la position binaire d'extrême gauche) est préservée lors d'un décalage à droite, c.-à-d. ce bit se propage de façon à garder la signe de la donnée.
  - Si la quantité est non signée, alors les bits laissés libres sont mis à 0.

Chap. 3 : 3. Opérateurs Standards

3.1.4 Opérateurs de bits - Opérateurs de décalage de bits-

Exemples

- **int** i, j ;
- **unsigned int** u, v ;

```
i = 5 ;          /* représentation de i : 0000 0000 0000 0101 */
j = i<<2 ;        /* représentation de j : 0000 0000 0001 0100
                    c.-à-d. la valeur 20 */
i = -3 ;          /* représ. de i : 1111 1111 1111 1101 */
j = i<<2 ;        /* représ. de j : 1111 1111 1111 0100
                    c.-à-d. la valeur -12 */
i = -32768 ;      /* représ. de i : 1000 0000 0000 0000 */
j = i>>2 ;        /* représ. de j : 1110 0000 0000 0000
                    c.-à-d. la valeur -8192 */
u = 32768 ;       /* représ. de u : 1000 0000 0000 0000 */
v = u>>2 ;        /* représ. de v : 0010 0000 0000 0000 */
```

Chap. 3 : 3. Opérateurs Standards

3.1.4 Opérateurs de bits

Opérateurs bit à bit *:*

& : ET logique.

| : OU inclusif.

^ : OU exclusif.

~ : complément à 1.

- Ici, les opérateurs portent sur les bits de même rang.
- Rappel :

1&1 == 1	1|1 == 1	1^1 == 0
1&0 == 0	1|0 == 1	1^0 == 1
0&1 == 0	0|1 == 1	0^1 == 1
0&0 == 0	0|0 == 0	0^0 == 0

Chap. 3 : 3. Opérateurs Standards

3.1.4 Opérateurs de bits -Opérateurs bit à bit

Exemples

- unsigned int n, c ;
  n = … ;
  c = n & 0177 ; /* 0177 est égal à (82 + 7 * 8 + 7) */
  c est constitué des 7 bits de poids faible de n, complétés à gauche par des 0

- unsigned int n, c ;
  n = … ;
  c = n | 0177 ;
  c est constitué des bits de n, les 7 bits de poids faible étant forcés à 1

- unsigned int n, c ;
  n = … ;
  c = n ^ 0177 ;
  c est constitué des bits de n, les 7 bits de poids faible étant inversés

Chap. 3 : 3. Opérateurs Standards

3.2 Opérateurs particuliers de C

3.2.1 Opérateurs d'affectation

- Pour la plupart des expressions de la forme :

  expr1 = (expr1) OP (expr2)

- Il existe une formulation équivalente utilisant un **opérateur d'affectation**:

  expr1 OP= expr2

- Opérateurs d'affectation utilisables :

+=	**-=**	***=**	**/=**	**%=**
<<=	**>>=**	**&=**	**^=**	**|=**

- **Exemples**

  a = a + b s'écrit a += b
  n = n << 2 s'écrit n <<= 2

Chap. 3 : 3. Opérateurs Standards

3.2 Opérateurs particuliers de C

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

- **Post-incrémentation**

  <var>++

  - La valeur de la variable <var> est d'abord utilisée telle quelle, puis incrémentée.

  - **Exemple**
    - int k, n ;
    - k = 0 ;

```
n = k++ ;      /* passe d'abord la valeur de k à n et incrémente après */
               /* ici, k vaut 1 et n vaut 0 */
```

Chap. 3 : 3. Opérateurs Standards

3.2 Opérateurs particuliers de C

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

- **Pré-incrémentation**

  ++<var>

  - **Exemple**

```
int k, n ;
k = 0 ;
n = ++k ;      /* incrémente d'abord et passe la valeur incrémentée à n */
               /* ici, k vaut 1 et n vaut 1 */
```

- **Post-décrémentation** **<var>--**
- **Pré-décrémentation** **--<var>**

Chap. 3 : 3. Opérateurs Standards

3.2 Opérateurs particuliers de C

3.2.3 Opérateur séquentiel (,)

<expr1> , <expr2>,…, <exprN>

- Exprime des calculs successifs dans une même expression
- Le type et la valeur de l'expression sont ceux du dernier opérande.
- **Exemple** :

  L'expression : **x = 5 , x + 6** a pour valeur 11

3.2.4 Opérateur conditionnel (? :)

<expression> ? <expr1> : <expr2>

- <expression> est évaluée. Si sa valeur est non nulle, alors la valeur de <expr1> est retournée. Sinon, c'est la valeur de <expr2> qui est renvoyée.
- **Exemple**

  max = a > b ? a : b

  si a est le plus grand, alors affectation à max du contenu de a sinon affectation du contenu de b

Chap. 3 : 3. Opérateurs Standards

3.2 Opérateurs particuliers de C

- **3.2.3 Opérateurs sizeof**

sizeof(<type>) ou **sizeof**(<variable>)

- Retourne le nombre d'octets occupés par le type de données ou la variable spécifiés.
- **Exemple**
  - **int** x ;
  - **sizeof**(**int**) /* retourne la valeur 2 (le type **int** occupe 2 octets). */
  - **sizeof**(x) /* retourne la valeur 2. */

Chap. 3 : 4. Expressions et Instructions

- Une **expression** est un **calcul** qui donne une **valeur comme résultat** (exemple : 8+5).
- Une **expression** peut comporter des **variables** et des **constantes** combinés entre eux par des **opérateurs** et former ainsi une **expression complexe**
- Les **expressions** peuvent contenir des **appels de fonctions** et elles peuvent **apparaître** comme **paramètres** dans des appels de **fonctions.**
- Toute **expression** suivie d'un **point virgule** devient une **instruction**.
- **Exemples**
  - i = 0;
  - i++;
  - a = (5 * x + 100 * y) * 2;
  - x = **pow**(a, 4); /* fonction puissance : **pow**(x, y) ↔ x^y */

Chap. 3 : 4. Expressions et Instructions

Remarque

- Les **affectations** sont aussi **interprétées** comme des **expressions**. **L'opérateur d'affectation** retourne la **valeur affectée**.

- On peut **enchaîner des affectations**. L'**évaluation** commence de la **droite vers la gauche**.

- **Exemples**
  - **b=(a = 5 + 3)+1** a = **8 et b = 9**
  - **a = b = c = d** équivalente à : **a = (b = (c = d))**

Chap. 3 : 5. Priorité et associativité des opérateurs

- Lors de l'évaluation des différentes parties d'une expression, les opérateurs respectent certaines lois de priorité et d'associativité.
- **Exemples**
  - La multiplication a la priorité sur l'addition
  - La multiplication et l'addition ont la priorité sur l'affectation.
- **Tableau des opérateurs et priorité**
  - **La priorité** est décroissante de haut en bas dans le tableau.
  - **La règle d'associativité** s'applique pour tous les opérateurs d'un même niveau de priorité. (→ pour une associativité de gauche à droite et ← pour une associativité de droite à gauche).
  - Dans une expression, les parenthèses forcent la priorité.

Priorité 1 (la plus forte):	()	→
Priorité 2:	! ++ --	←
Priorité 3:	* / %	→
Priorité 4:	+ -	→
Priorité 5:	< <= > >=	→
Priorité 6:	== !=	→
Priorité 7:	&&	→
Priorité 9 (la plus faible):	= += -= *= /= %=	←

Chap. 3 : 5. Priorité et associativité des opérateurs

- **Exemples**
  - **a = b = c** s'interprète comme **a = (b = c)**
  - a *= b += 5 s'évalue (a = 3 ; b = 4 ;) :
    b += 5 → 9
    a *= 9 → a = 27
  - !--a == ++!b s'évalue (a = 1 ; b = 4 ;) :
    !b → 0
    ++0 → 1
    --a → 0
    !0 → 1
    1 == 1 → 1

Chap. 3 : 6. Conversions de type (cast)

6.1 Conversion automatique

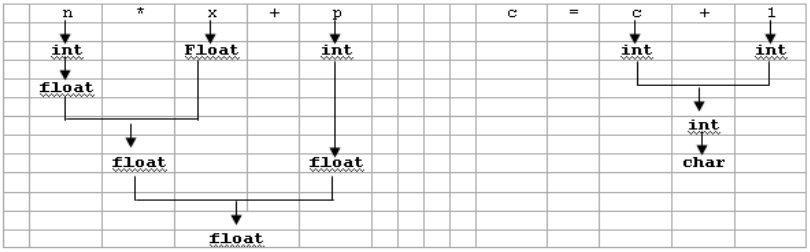
- Si un opérateur a des opérandes de différents types, les valeurs des opérandes sont converties automatiquement dans un type commun.
- **Règle de conversion automatique**
  Lors d'une opération avec :
  - deux entiers : les types **char** et **short** sont convertis en **int**. Ensuite, il est choisit le plus large des deux types dans l'échelle : **int, unsigned int, long, unsigned long.**
  - un entier et un réel : le type entier est converti dans le type du **réel**.
  - deux réels : il est choisit le **plus large** des deux types selon l'échelle : **float, double, long double**.
  - Dans une affectation : le **résultat** est toujours converti dans le type de la destination. **Si ce type est plus faible, il peut y avoir une perte de précision.**

Chap. 3 : 6. Conversions de type (cast)

6.1 Conversion automatique

Exemples :

1.
```
int n, p ;
float x ;
char c ;
```
Donner le type de chacune des expressions suivantes :
```
n * x + p
c = c + 1
```



2. *Exemple de perte de précision* :
```
unsigned int a = 70 000 ; /* la valeur de a sera 4464 */
/* ceci est accepté, aucun avertissement du compilateur C */
```


Chap. 3 : 6. Conversions de type

6.2 Conversion forcées(casting)

- Le type d'une expression peut être forcé, en utilisant **l'opérateur cast** :

(<type>) <expression>

- **Exemple**
  - **char** a = 49; // **a = '1'**
  - **int** b = 4 ;
  - **float** c ;

    c = (float) a / b;
  - Quelle est la valeur de c ?

    12.25

Chap. 4 : Lire & Ecrire des données

4.1 Ecriture formatée de données : printf()

- La fonction printf permet d'afficher du texte, des valeurs de variables ou des résultats d'expressions sur écran (sortie standard).

- **Forme générale** :

  printf("<format>", <expr1>, <expr2>, …, <exprN>) ;

- La partie **"<format>"** est une chaîne de caractères qui peut contenir :
  - du **texte**
  - des **caractères de contrôle** ('\n' , '\t' , …)
  - des **spécificateurs de format**.

Chap. 4 : Lire & Ecrire des données

4.1 Ecriture formatée de données : printf()

- La partie **"<format>"** contient exactement **un spécificateur** pour **chaque expression** <expr1>, <expr2>, … et <exprN>.

- Les **spécificateurs de format** commencent toujours par le symbole **%** et se terminent par **un ou deux caractères** qui indiquent le **format d'affichage**.

Spécificateurs de format pour printf :

Spécificateur	*Rôle (afficher :)*	*Type*
%c	un seul caractère	**char**
%d ou %i	un entier relatif	**int**
%u	un entier naturel (non signé)	**unsigned int**
%o	un entier sous forme octale	**int**
%x	un entier sous forme hexadécimale (a-f)	**int**
%X	un entier sous forme hexadécimale (A-F)	**int**
%f	un réel	**float** ou **double**
%e	un réel en notation exponentielle (e)	**float** ou **double**
%E	un réel en notation exponentielle (E)	**float** ou **double**
%s	une chaîne de caractères	**char** *

Chap. 4 : Lire & Ecrire des données

4.1 Ecriture formatée de données : printf()

- **Exemples :**
  - La suite d'instructions :
    - int a = 1234 ;
    - int b = 566 ;
      printf("**%i** plus **%i** est **%i**\n", a, b, a + b) ;
    - va afficher sur l'écran :
      1234 plus 566 est 1800
  - La suite d'instructions :
    - char b = 'A' ; /* le code ASCII de A est 65 */
      printf("Le caractère **%c** a le code **%i**\n", b, b) ;
    - va afficher sur l'écran :
      Le caractère A a le code 65

Chap. 4 : Lire & Ecrire des données

4.2 Lecture formatée de données : scanf()

- **scanf** lit depuis le clavier (entrée standard). Elle fait correspondre les caractères lus au format indiqué dans la chaîne de format.
- La **spécification de formats** pour scanf est identique à celle de printf, sauf qu'au lieu de fournir comme arguments des variables à scanf, ce sont **les adresses de ces variables que l'on transmet.**
- L'**adresse d'une variable** est indiquée par le **nom de la variable** précédé du signe **&.**
- **Forme générale** :

scanf("<format>", <AdrVar1>, <AdrVar2>, …, <AdrVarN>)

Chap. 4 : Lire & Ecrire des données

4.2 Lecture formatée de données : scantf()

- **Exemples** :
  - int jour, mois, annee ;
    scanf("**%i %i %i**", &jour, &mois, &annee) ;
  - Cette instruction lit 3 entiers séparés par les espaces, tabulations ou interlignes. Les valeurs sont attribuées respectivement aux 3 variables : jour, mois et annee.
  - int i ;
  - float x ;
    scanf("**%d %f**", &i, &x) ;
  - Si lors de l'exécution, on entre 48 et 38.3e-1 alors scanf affecte 48 à i et 38.3e-1 à x.

Chap. 4 : Lire & Ecrire des données

4.2 Lecture formatée de données : scanf()

- **Remarques** :
  - Lors de l'évaluation des données, **scanf** s'arrête si la chaîne de format a été travaillée jusqu'à la fin ou si une donnée ne correspond pas au format indiqué.
  - **scanf** retourne comme résultat le nombre d'arguments correctement reçus et affectés.
  - **Exemples** :
    - int jour, mois, annee, recu ;
    - recu = scanf("**%i %i %i**", &jour, &mois, &annee) ;

Données introduites	*Affichage à l'écran*			
	recu	*jour*	*mois*	*annee*
12 4 1980	3	12	4	1980
12/4/1980	1	12	indéfinie	indéfinie
12.4 1980	1	12	indéfinie	indéfinie
12 4 19.80	3	12	4	19

Chap. 4 : Lire & Ecrire des données

4.2 Ecriture d'un caractère : putchar()

- **putchar** permet d'afficher un caractère sur l'écran.
- **putchar(c)** ; est équivalente à **printf("%c", c)** ;
- **Forme générale** :

  putchar(<caractere>) ;
- Elle reçoit comme argument la valeur d'un caractère convertie en entier.

Chap. 4 : Lire & Ecrire des données

4.2 Ecriture d'un caractère : putchar()

Exemples

- char a = 63 ;
- char b = '\n' ;
- int c = '\a' ;

```
putchar('x') ; /* affiche la lettre x */
putchar('?') ; /* affiche le symbole ? */
putchar(b) ;  /* retour à la ligne */
putchar(65) ; /* affiche le caractère de code ASCII = 65  c.-à-d. la lettre A */
putchar(a) ;  /* affiche le caractère de code ASCII = 63  c.-à-d. le symbole ? */
putchar(c) ;  /* beep sonore */
```

- **Remarque :**

putchar retourne la **valeur du caractère** écrit toujours considéré comme un entier, ou bien la **valeur -1 (EOF)** en cas d'erreur.

Chap. 4 : Lire & Ecrire des données

4.2 Lecture d'un caractère : getchar()

- Permet de lire un **caractère** depuis le **clavier**.
- **c=getchar(c)** ; est équivalente à **scanf("%c",&c)** ;
- ***Forme générale*** :

  <Caractere> = getchar() ;
- **Remarques** :
  - **getchar** retourne le caractère lu (un entier entre 0 et 255), ou bien la valeur -1 (EOF).
  - **getchar** lit les données depuis le clavier et fournit les données après confirmation par la touche **"entrée"**
- **Exemple** :
  - int c ;

    c = getchar() ; /* attend la saisie d'un caractère au clavier */

Chap. 5 : **Structures de contrôle**

- On appelle structure de contrôle toute instruction qui permet de contrôler le fonctionnement d'un programme.
- Parmi les structures de contrôle, on distingue :
  - structures de choix et
  - structures répétitives

Chap. 5 : Structures de contrôle

5.1 Structures de choix

- Les structures de choix permettent de déterminer quelles instructions seront exécutées et dans quel ordre.
- En langage C, les structures de choix peuvent être exprimées par :
  - L'instruction de branchement conditionnels : **if...else**
  - l'instruction de branchement multiple : **switch**

Chap. 5 : Structures de contrôle

5.1 Structures de choix

5.1.1 Branchement conditionnel (if … else)

Format :

```
if (expression) bloc-instruction-1
else  bloc-instruction-2
```

où

- expression : est une expression quelconque. Après évaluation, si elle est vraie, alors le 1er bloc d'instructions est exécuté, sinon c'est le 2ème bloc qui est exécuté.
- bloc d'instructions : peut désigner **une suite d'instructions délimitées par des accolades** ou une seule instruction.

Chap. 5 : Structures de contrôle

5.1 Structures de choix

5.1.1 Branchement conditionnel (if … else)

Remarques :

- On notera que l'expression conditionnelle doit être entre parenthèses.
- La partie else est optionnelle :
  if (expression) bloc-instruction
- Lorsque plusieurs instructions if sont imbriquées, il est convenu que chaque else se rapporte au *dernier* if qui ne possède pas de partie else.

Chap. 5 : Structures de contrôle

5.1 Structures de choix

5.1.1 Branchement conditionnel (if … else)

Exemples :

- Calcul du maximum de deux entiers
- Teste si un caractère saisie depuis le clavier , est une lettre majuscule , si oui il affiche cette lettre en minuscule, sinon il affiche un message d'erreur.
- Résolution de l'équation du second degré : $a.x^2 + b.x + c = 0$

Chap. 5 : Structures de contrôle

5.1 Structures de choix

5.1.2 Branchement multiple (switch)

On l'appelle aussi l'*instruction d'aiguillage. Elle t*este si une expression prend une valeur parmi *une suite de constantes*, et effectue le branchement correspondant si c'est le cas.

Format :

```
switch ( expression )
{
  case  constante1 :  suite d'instructions 1
  case  constante2 :  suite d'instructions 2
  ...
  case  constanteN : suite d'instructions N
  default :  suite d'instructions
 }
```


Chap. 5 : Structures de contrôle

5.1 Structures de choix

5.1.2 Branchement multiple (switch)

Remarques **:**

- Le fonctionnement de cette instruction est le suivant :
  - expression est évaluée ;
  - s'il existe un énoncé case avec une constante qui égale la valeur de expression, le contrôle est transféré à l'instruction qui suit cet énoncé;
  - si un tel case n'existe pas, et si énoncé default existe, alors le contrôle est transféré à l'instruction qui suit l'énoncé default ;
  - si la valeur de expression ne correspond à aucun énoncé case et s'il n'y a pas d'énoncé default, alors aucune instruction n'est exécutée.

- **Attention**. Lorsqu'il y a branchement réussi à un case, toutes les instructions qui le suivent sont exécutées, jusqu'à la fin du bloc ou jusqu'à une instruction de rupture (break).

Chap. 5 : Structures de contrôle

5.1 Structures de choix 5.1.2 Branchement multiple (switch)

Exemples :

```
#include <stdio.h>
main()
{   int  a,b ,
    char operateur ;
         printf("Entrez un opérateur (+, -, * ou /) : ") ;
        scanf("%c", &operateur) ;
        printf("Entrez deux entiers : ") ;
        scanf("%d  %d", &a,&b) ;
       switch (operateur)
       {
          case '+' : printf("a + b = %d\n",a+b) ; break ;
          case '-'  : printf("a - b = %d\n",a-b) ; break ;
          case '*' : printf(" a * b = %d\n",a*b) ; break ;
          case '/' : printf("a / b = %d\n",a/b) ; break ;
         default  : printf("opérateur inconnu\n") ; break ;
       }
       return 0 ;
}
```

Soit le code C suivant :

```
int i , j;
... //initialisation de i
j = 0;
switch (i)
{
     case 3: j+ +;
     case 2: j+ +;
     case 1: j+ +;
}
```

- Si on suppose que i ne peut prendre que les valeurs 0 ... 3.
- Que fait ce programme?

Chap. 5 : Structures de contrôle

5.2 Structures répétitives (Boucles)

- Les structures répétitives (ou Boucles) permettent de répéter une série d'instructions tant qu'une certaine condition reste vraie.
- On appelle parfois ces structures instructions d'itérations.
- En langage C, les structures répétitives peuvent être exprimées par :
  - les instructions while et do ... while
  - l'instruction for

Chap. 5 : Structures de contrôle

5.2.1 while et do … while

- Les instructions **while** et **do … while** représentent un moyen d'exécuter plusieurs fois la même série d'instructions.
- La syntaxe :

```
while ( condition )
{
        liste d'instructions
}
```

```
do
{
        liste d'instructions
}
while ( condition );
```

- **Dans la structure while on vérifie la condition avant d'exécuter la liste d'instructions, tandis que dans la structure do … while on exécute la liste d'instructions avant de vérifier la condition**

Chap. 5 : Structures de contrôle

5.2.1 while et do … while

Exemples

1. Programme pour imprimer les entiers de 1 à 9.

```
i = 1;

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

2. Programme pour saisir au clavier un entier entre 1 et 10 :

```
int a;

do
{
  printf("\n Entrez un entier entre 1 et 10 : ");
  scanf("%d",&a);
}
while ((a <= 0) || (a > 10));
```

Chap. 5 : Structures de contrôle

5.2.1 for

- A l'instar des instructions **while** et **do … while, l'instruction for permet** d'exécuter plusieurs fois la même série d'instructions.
- La syntaxe de for est :

```
for ( expression1 ; expression2 ; expression3 )
{
        liste d'instructions
}
```

- Dans la construction de for :
  - **expression1** : effectue les initialisations nécessaires avant l'entrée dans la boucle;
  - **expression2** : est le test de continuation de la boucle ; le test est évalué avant l'exécution du corps de la boucle;
  - **expression3** : est évaluée à la fin du corps de la boucle.

Chap. 5 : Structures de contrôle

5.2.1 for

Remarques

- En pratique, expression1 et expression3 contiennent souvent plusieurs initialisations séparées par des virgules.
- Les expressions *expression*1 et *expression*3 peuvent être absentes (les points-virgules doivent cependant apparaître). **Par exemple** : for (; *expression*2 ;)
- Lorsque expression2 est absente, l'expression correspondante est considérée comme vraie. Par conséquent, **for** (; ;) est une boucle infinie.

Chap. 5 : Structures de contrôle

5.2.1 for

Remarques(suite)

- Par définition, la construction de for

```
for ( expression1 ; expression2 ; expression3 )
{
        liste d'instructions
}
```

est équivaut *strictement à* celle-ci :

```
expression1;
while (expression2)
{
   liste instructions;
   expression3;
}
```


Chap. 5 : Structures de contrôle

5.2.1 for

Exemples

1. Programme pour calculer la somme de 1 à 100 :

```
int n, total ;

for ( total  = 0, n = 1 ; n<101 ; n++ )
            total + = n ;

printf("La somme des nombres de 1 à 100 est %d\n", total) ;
```

2. Programme pour calculer la factorielle d'un entier n :

```
int n , i , fact ;

for ( i = 1, fact = 1 ; i < = n ; i++)
            fact *= i;

printf("%d ! = %d \n",n,fact);
```

Chap. 5 : Structures de contrôle

5.3 Instructions break et continue

5.3.1 l'instruction break

- On a vu le rôle de l'instruction break; au sein d'une instruction de branchement multiple switch.
- L'instruction break peut, plus généralement, être employée à l'intérieur de n'importe quelle boucle (for ; while ; do …while ; switch). Elle permet l'abandon de la structure et le passage à la première instruction qui suit la structure.
- En cas de boucles imbriquées, break fait sortir de la boucle la plus interne.

Chap. 5 : Structures de contrôle

5.3 Instructions break et continue 5.3.1 l'instruction break

Exemples (break)

1. Que fait ce programme?

```
for ( ; ; )
{
   printf("donne un nombre (0 pour sortir) : ");
   scanf("%d", &n);
  if (n == 0) break;
  ...
  exploitation de la donnée n
  ...
}
```

2. Que fait ce programme?

```
#include <stdio.h>
int main( )
{
  int i, j ;

  for (i = 1 ; i<=15 ; i++)
  {
   for (j = 1 ; j<=15 ; j++)
    {
      if (j = = 5)   break ;
      printf("%d\t", i * j) ;
    }
   printf("\n") ;
  }
  return 0 ;
}
```

Chap. 5 : Structures de contrôle

5.3 Instructions break et continue

5.3.1 l'instruction continue

- L'instruction continue peut être employée à l'intérieur d'une structure de type boucle (for ; while ; do …while).
- Elle produit l'abandon de l'itération courante et fait passer directement à l'itération suivante d'une boucle
- L'instruction continue concerne la boucle la plus proche.

Chap. 5 : Structures de contrôle

5.3 Instructions break et continue **5.3.2 l'instruction continue**

Exemple (continue)

1. Que fait ce programme?

```
int main()
{
   int i, j ;
   ... //initialisation de i et j
   for ( ; i>0 && j>0 ; i--, j-- )
   {
      if ( i == 5 )    continue ;
      printf("i : %d et j : %d\n, i, j) ;
      if ( j == 5 )
      break ;
   }
   return 0 ;
}
```

Valeurs introduites	Affichage
i = 2 et j = 3	i : 2 et j : 3 i : 1 et j : 2
i = 6 et j = 3	i : 6 et j : 3 i : 4 et j : 1
i = 3 et j = 5	i : 3 et j : 5