|  |  |  |
| --- | --- | --- |
|  | **INSTITUT SUPERIEUR DES ETUDES TECHNOLOGIQUES DE NABEUL** | |
|  | Travaux Pratiques N° 1 : Débuter avec le langage C | |
| **Classe : TI 1** | | **Enseignante : Sana REFAI** |

## 1. Objectifs

Ce TP permet de:

* Se familiariser avec l’IDE Code blocks
* Connaitre la structure d’un programme en C
* Déclarer des variables de types simples en C
* Déclarer des constants
* Manipuler les fonctions d’entrée/Sortie

## 2. Structure d’un programme en C

En langage C le programme principal et les sous programmes sont définies comme des fonctions. Lorsqu'un programme est chargé, son exécution commence par l'appel d'une fonction spéciale du programme. Cette fonction doit impérativement s'appeler « main » (principal en anglais) pour que le compilateur puisse savoir que c'est cette fonction qui marque le début du programme.

* Les commentaires sont utilisés pour des raisons de lisibilité et de compréhension du programme. Un commentaire est une chaîne de caractères comprise entre /\* et \*/. Cette chaîne est ignorée par le compilateur C et peut déborder sur plusieurs lignes.
* Le langage C distingue les minuscules, des majuscules. Les mots réservés (exemples : for, while, return, struct …) du langage C doivent être écrits en minuscules.
* Toute instruction C se termine par un point-virgule ( ;) ou une accolade fermante si elle avait débuté par une accolade ouvrante

|  |  |
| --- | --- |
| **Syntaxe** | **Exemple 1** |
| **<include des bibliothèques>**  **<type\_fonction>**  **main()**  **{**  **<déclaration des variables>**  **<instructions>**  **}** | **/\* déclaration des bibliothèques\*/**  **#include <stdio.h>**  **/\* fonction principale \*/**  **main()**  **/\* Notre premier programme en C \*/**  **{**  **/\* Instructions du programme \*/**  **printf("hello, world\n");**  **}** |

. ***Les bibliothèques en C***

Le langage C dispose de bibliothèques de fonctions prédéfinies qui sont disponibles sous leur forme pré-compilée avec une extension .LIB

Pour pouvoir utiliser ces fonctions prédéfinies, il faut inclure des fichiers entête dans les programmes avec une extension .H :

#include <NomFichierEntête>

La commande **#include** sert à inclure un fichier. En règle générale, si le nom est entre ***<>*** il s’agit d’un fichier du système. S’il est entre guillemets, il s’agit par contre d’un fichier saisi par un utilisateur.

**Exemple:**

Pour faire appel à des fonctions mathématiques prédéfinies, il faudrait inclure la

bibliothèque:

MATHS.LIB: #include <maths.h>

Pour faire appel à des fonctions d'entrés-sorties standard, il faudrait inclure la bibliothèque:

STDIO.LIB :

#include <stdio.h>

Après la compilation et lors de l’édition de liens, les fonctions pré-compilées des bibliothèques seront ajoutées au programme compilé afin d’obtenir un programme complet exécutable

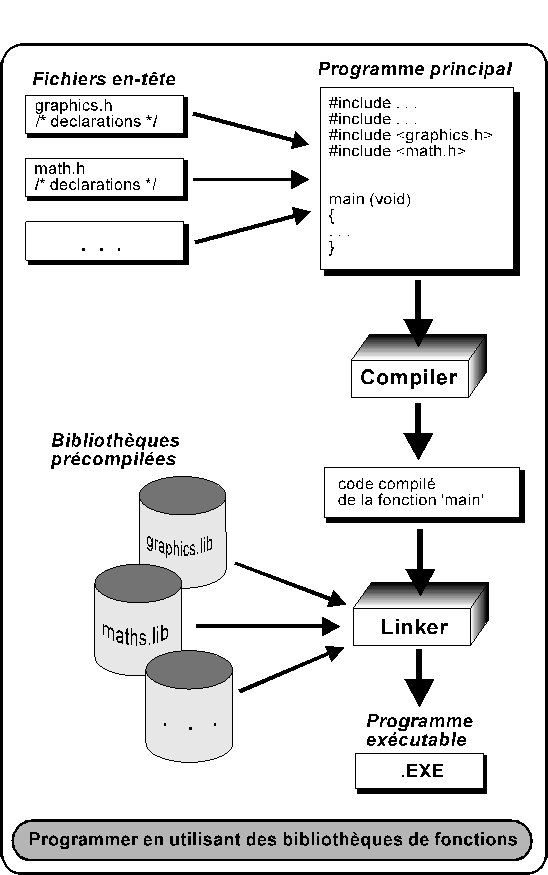


Figure 1 Bibliothèques de fonctions et compilation

## 3. Déclarations des variables en C

Les variables permettent de représenter les données d’un programme. Chaque variable est matérialisée par un emplacement mémoire identifié par **un** **nom** ou **un identificateur**.

Les variables contiennent les valeurs qui sont utilisées pendant l’exécution du programme.

Chaque variable possède :

* un **nom**
* un **type**
* une **valeur**

### 3.1. Les noms des variables en C

Les noms des variables en C sont composés de lettres et de chiffres :

* L’ensemble des symboles utilisables : **{0,1,2, ...9, A,B,...,Z,\_,a,b,....,z}**
* Le premier caractère doit être une lettre ou le symbole **‘ \_ ’**
* C distingue les majuscules et les minuscules. **Exemple**:‘ compteur ’ et ‘ Compteur’ sont deux variables différentes
* La longueur des identificateurs n’est généralement pas limitée (C distingue généralement les 31 premiers caractères)
* Le choix des noms des variables est important pour la lisibilité d’un programme.
* Le nom d'une variable doit être différents des mots réservés.

En voici la liste :

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

|  |  |  |
| --- | --- | --- |
| **Nom de variable correct** | **Nom de variable incorrect** | **Raison** |
| Variable | Nom de Variable | comporte des espaces |
| Nom\_De\_Variable | 123Nom\_De\_Variable | commence par un chiffre |
| nom\_de\_variable | toto@mailcity.com | caractère spécial @ |
| nom\_de\_variable\_123 | Nom-de-variable | signe - interdit |
| \_707 | goto | nom réservé |

### 3.2. Les types des variables en C

Le type nous indique de quelle sorte de variable il s’agit :entier, décimal, caractère, ...

Le type est une caractéristique restreignant les champs de valeurs possibles et les opérateurs applicables :

* Les types entiers :

- **int** : entier standard

- **short** : entier court

- **long** : entier long

- **char** : caractère

* Les types rationnels

-  **float** : décimal

- **double** : décimal long

On distingue deux caractéristiques concernant les types :

- Le **domaine** des valeurs admissibles

- Le **nombre d’octets** réservé pour une variable

#### 3.2.1. Les types entiers

Le langage C distingue plusieurs types d'entiers:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type de donnée** | **Signification** | **Taille (en octets)** | **Plage de valeurs acceptée** |
| **Char** | Caractère | 1 | -128 à 127 |
| **unsigned char** | Caractère non signé | 1 | 0 à 255 |
| **short int** | Entier court | 2 | -32 768 à 32 767 |
| **unsigned short int** | Entier court non signé | 2 | 0 à 65 535 |
| **Int** | Entier | 4 (32 bits) | -2 147 483 648 à 2 147 483 647 |
| **unsigned int** | Entier non signé | 4 (32 bits) | 0 à 4 294 967 295 |
| **long int** | Entier long | 4 | -2 147 483 648 à 2 147 483 647 |
| **unsigned long int** | Entier long non signé | 4 | 0 à 4 294 967 295 |

**Remarque** :

En langage C, le type **char** est un cas particulier du type entier : ***un caractère est un entier de 8 bits***

*Exemple :* Le **caractère** 'b' a pour valeur 98 (son code ASCII).

#### 3.2.2. Les types réels

En informatique, les rationnels sont souvent appelés des 'flottants'. Ce terme vient de *'en virgule flottante'* et trouve sa racine dans la notation traditionnelle des rationnels :

**<+|-> *<mantisse> \* 10<exposant>***

* **<+|->:** est le signe positif ou négatif du nombre.
* ***<mantisse>* :** est un décimal positif avec un seul chiffre devant la virgule.
* ***<exposant>:*** est un entier relatif

Le langage C distingue 3 types de réels :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type de donnée** | **Signification** | **Mantisse** | **Taille (octets)** | **Plage de valeurs acceptée** |
| **float** | Flottant (réel) | 6 | 4 | 3.4\*10-38 à 3.4\*1038 |
| **double** | Flottant double | 15 | 8 | 1.7\*10-308 à 1.7\*10308 |
| **long double** | Flottant double long | 19 | 10 | 3.4\*10-4932 à 3.4\*104932 |

### 3.3. Déclaration de variables

Pour être utilisée, une variable doit être déclarée :

Syntaxe:

**<Type> <NomVar1>,<NomVar2>,...,<NomVarn> ;**

**int compteur, x, y;**

Exemple:

**float racine;**

**char touche;**

**/\* Déclaration avec initialisation \*/**

**int nombre = 1;**

**char lettre = ‘ A ’;**

**Remarque** :

La déclaration d’une variable se traduit par une réservation de place mémoire dont le contenu est imprévisible !

## 4. Les constantes en C

Les constantes sont utilisées pour calculer des valeurs, pour initialiser des variables…

* **Les constantes entières**

• 100 : int

• 0144 : int (en octal)

• 0x64 : int (en hexadécimal)

* **Les constantes rationnelles**

• 123.4 : float (notation décimale)

• 1234e-1 : float (notation exponentielle)

* **Les caractères constants** (entourés d’apostrophes)

• ‘A ’ désigne la lettre A (sa valeur est 65 = code ASCII).

*(Voir Annexe D: codes ASCII pour tous les caractères).*

• On distingue des caractères non imprimables qui sont identifiés par leurs codes internes commençant par **\** :

‘ \0 ’ : caractère NUL

‘ \n ’ : nouvelle ligne

‘ \t ’ : tabulation horizontale

‘ \b ’: espacement arrière

‘ \f ’ : saut de page

‘ \r ’ : retour chariot

‘ \v ’ : tabulation verticale

‘ \\ ’ : back slach

‘ \″ ’ : guillemet

‘ \’ ’ : apostrophe

* **Constante chaîne de caractères** (entourée de guillemets)

• Une chaînepeut ne comporter aucun caractère : " " (exemple:"message").

• Le compilateur insert automatiquement un caractère nul \0 à la fin de chaque chaîne de caractères.

* **Définition de constantes**

Syntaxe:

**#define <NomConstante> <Valeur>**

Exemple:

**#define PI 3.14**

**#define LETTRE\_1 ‘A’**

## 5. Affectation des variables

L’affectation est une instruction qui permet de placer une **valeur** dans une **variable** qui doit être préalablement déclarée.

Syntaxe:

**<nom\_variable> = <expression> ;**

**Exemple:**

« Affecter le nombre **5** à la variable **A**» est équivalent à « Ranger dans la zone mémoire **A** l’entier **5**» s’écrit en C : **A=5;**

**Schématiquement:**

**5**

5

**A A**

On distingue plusieurs types d’affectations :

* **Affectation avec des valeurs constantes :** placement d’une valeur dans une *variable.*

Exemple:

**Montant = 6;**

* **Affectation avec des valeurs de variables :** la valeur à placer dans une variable

peut provenir d’une autre variable.

Exemple:

**A = 2;**

**B = A;**

* **Affectation avec des valeurs d’expressions :** On peut ranger dans une variable

le résultat d’un calcul.

**A = 2;**

Exemple:

**B = A + 2;**

* **Initialisation:** La déclaration d'une variable ne fait que "réserver" un emplacement mémoire où stocker la variable. Tant que l'on ne lui a pas affecté une donnée celle-ci contient ce qui se trouvait précédemment à cet emplacement, que l'on appelle *garbage* (en français: *détritus*).

On peut donc affecter une valeur initiale à la variable lors de sa déclaration, on parle alors d'initialisation:

Syntaxe:

**type Nom\_de\_la\_variable = donnee;**

Exemple:

**float Toto = 125.36;**

## 6. Les opérateurs

En langage C, on distingue plusieurs types d’opérateurs.

### 6.1. Les operate urs simples

* **Opérateurs arithmétiques sur les réels**

**+ : Addition**

**- : Soustraction**

**\* : Multiplication**

**/ : Division (rationnelle)**

Attention : 2/4 vaut 0 mais 2.0/4 vaut 0.5

* **Opérateurs arithmétiques sur les entiers**

**+ : Addition**

**- : Soustraction**

**\* : Multiplication**

**/ : Quotient de la division**

**% : Reste de la division.**

2/4 vaut 0 et 2%4 vaut 2

**char c,d;**

Exemple:

**c = 'G';**

**d = c+'a'-'A';**

Les caractères sont des entiers sur 8 bits, on peut donc effectuer des opérations. Sur cet exemple, on transforme la lettre majuscule G en la lettre minuscule g.

* **Opérateurs logiques sur les entiers**

**& : ET**

**| : OU**

**^ : OU EXCLUSIF**

**~ : COMPLEMENT A UN**

**« : DECALAGE A GAUCHE**

**» : DECALAGE A DROITE**

Exemple:

**p = n « 3; /\* p est égale à n décalé de 3 bits à gauche \*/**

**p = n » 3; /\* p est égale à n décalé de 3 bits à droite \*/**

* **L'opérateur sizeof(type)**

Renvoie le nombre d'octets réservés en mémoire pour chaque type d'objet.

Exemple:

**n = sizeof(char); /\* n vaut 1 \*/**

* **L'opérateurs de comparaison:** (1: vrai et 0: faux)

**== : égale à**

**!= : différent de**

**<, <=, >, >= : plus petit, plus petit ou égal, ...**

**Remarques:**

* + 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**

**0 && 2**  **0 (faux)**

**!2 0 (faux)**

**5||!(32 > 4) 1 (vrai)**

### 6.2. Liste des opérateurs d’affectation

Le langage C autorise des écritures simplifiées lorsqu'une même variable est utilisée de chaque côté du signe = d'une affectation. Ces écritures sont à éviter lorsque l'on débute l'étude du langage C car elles nuisent à la lisibilité du programme.

**expr1 op= expr2**

*Liste des opérateurs d’affectation:*

**a = a+b; est équivalent à a+= b;**

**a = a-b; est équivalent à a-= b;**

**a = a \*b; est équivalent à a\*= b;**

**a = a %b; est équivalent à a%= b;**

**a = a /b; est équivalent à a/= b;**

### 6.3. Les opérateurs d’incrémentation et de décrémentation

Les affectations les plus fréquentes sont du type :

**i = i + 1;**

**et**

**i = i - 1;**

Le langage C autorise des écritures simplifiées pour l'incrémentation et la décrémentation de variables :

**i = i+1; est équivalent à i++;**

**i = i-1; est équivalent à i--;**

* Incrémenter ou bien décrémenter de 1 une variable

**i++; et ++i;**

**ou bien**

**i--; et --i;**

* Incrémenter ou bien décrémenter de 1 une variable et en même temps affecter sa valeur à une autre variable
  + **x = i++;** passe la valeur de i à x et incrémente après i
  + **x = i--;** passe la valeur de i à x et décrémente après i
  + **x = ++i;** incrémente d’abord i et passe la valeur à x
  + **x = --i;**  décrémente d’abord i et passe la valeur à x

Exemple:

**i = 5; a = ++i; /\* i = 6 et a = 6 \*/**

**i = 5; a = i++; /\* i = 6 et a = 5 \*/**

### *6.4. La priorité des opérateurs*

Les constantes et les variables sont ***des expressions***. Les expressions peuvent être combinées entre elles par des opérateurs et former ainsi des expressions plus complexes.

L’ordre d’évaluation des différentes parties d’une expression est en principe le même qu’en mathématiques.

**x = 2\*A+3\*B+4\*c; (\*,+,=)**

Classe de priorité:

Priorité 1 (la plus forte) : **( )** G->D

Priorité 2 : **! ++ --** *D->G*

Priorité 3 : **\* / %** G->D

Priorité 4 : **+ -** G->D

Priorité 5 : **< <= > >=** G->D

Priorité 6 : **== !=**  G->D

Priorité 7 : **&&** G->D

Priorité 8 : **||** G->D

Priorité 9 : **= += -= \*= /= %=** *D->G*

**Conversion de type**

En langage C, on distingue deux types de conversion :

Conversion de type automatique

Si les opérandes sont de différents types, le résultat de l’opération est converti vers le type ‘ le plus grand ’ ( pas de perte de précision).

Pour l’affectation la donnée à droite est convertie dans le type de celle de gauche, il y a perte de précision si le type de la destination est plus faible.

**int i = 8;**

Exemple:

**float x = 12.5;**

**double y;**

**y = i \* x; /\* Pas de perte de précision \*/**

Conversion de type forcée (casting)

On peut convertir explicitement une valeur en forçant la transformation.

Syntaxe:

**<NomVariable> = (<Type>)<Expression>;**

**int i = 2, j = 4;**

Exemple1:

**float y, z;**

**y = i / j; /\* y = 0.0 \*/**

**z = (float)i / j; /\* y= 0.5 \*/**

Exemple2:

**#include <stdio.h>**

**#include <conio.h>**

**void main()**

**{**

**int i=0x1234,j;**

**char d,e;**

**float r=89.67,s;**

**j = (int)r;**

**s = (float)i;**

**d = (char)i;**

**e = (char)r;**

**printf("Conversion float -> int: %5.2f -> %d\n",r,j);**

**printf("Conversion int -> float: %d -> %5.2f\n",i,s);**

**printf("Conversion int -> char: %x -> %x\n",i,d);**

**printf("Conversion float -> char: %5.2f -> %d\n",r,e);**

**printf("Pour sortir frapper une touche ");getch();**

**}**

## 7. Lecture et écriture

L’instruction de **lecture** permet de fournir des informations (**données**) à notre programme par l’intermédiaire d’un périphérique (clavier).

L’instruction d**’écriture** permet à un programme de communiquer des informations (**résultats**) par l’intermédiaire d’un périphérique (écran).

La bibliothèque standard <**stdio**> contient un ensemble de fonctions permettant la communication de l’ordinateur avec le monde extérieur.

Les principales fonctions de lecture et d’écriture en langage C sont **scanf** et **printf**. Ces deux fonctions font parties de la bibliothèque standard <**stdio**>.

**Ordinateur**

**Données**

**Résultats**

**Programme**

**Lecture des données**

**Écriture des résultats**

### 7.1. Ecriture des données

### 7.1.1. La fonction printf

La fonction, formatée d’écriture de données, **printf** permet de transférer du texte, des valeurs de variables ou des résultats d’expressions vers l’écran. Elle exige l’utilisation de formats de sortie.

**Syntaxe**

**printf (“<format> “, <expr1>, <expr2>, ...);**

* **<format>:** texte, séquence d’échappement, spécificateur de format.
* Autant de spécificateurs de formats que d’expressions.

**Spécificateurs de format**

**%[justification][largeur\_min][.précision\_pour\_les\_rationnels] caractère\_du\_type**

***Caractère du type : (obligatoire)***

|  |  |  |
| --- | --- | --- |
| **Caractère** | **Signification** | **Exemple** |
| D | Conversion en décimale | int a =64; printf(“%d”,a); 64 |
| O | Conversion en octale | printf(“%o”,a); 100 |
| X | Conversion en hexadécimale (de 0 à f) | printf(“%x”,a); |
| X | Conversion en hexadécimale (de 0 à F) |  |
| U | Conversion en un entier non signé |  |
| C | Conversion en caractère | int a =65; printf(“%c”,a); A |
| S | Conversion en chaîne de caractères |  |
| E | Conversion sous la forme m.nnnexx |  |
| E | Conversion sous la forme m.nnnExx |  |
| F | Conversion sous la forme m.nnn |  |

***Largeur minimale : (facultatif)***

Nombre minimal de caractères à afficher avec remplissage par des espaces.

Exemple : ( **\_** <=> position libre)

|  |  |  |
| --- | --- | --- |
| **printf("%4d", 123);** | **==>** | **\_123** |
| **printf("%4d", 1234);** | **==>** | **1234** |
| **printf("%4d", 12345);** | **==>** | **12345** |
| **printf("%4u", 0);** | **==>** | **\_\_\_0** |
| **printf("%4X", 123);** | **==>** | **\_\_7B** |
| **printf("%4x", 123);** | **==>** | **\_\_7b** |

***Justification : (facultatif)***

Indicateurs de justification en sortie :

**- :** justifie a gauche en remplissant à droite avec des espaces sinon spécifié, justifie à droite en remplissant a gauche par des espaces.

**int a =64;**

**printf(“a=%-4dDT”,a); a=64 DT**

**+ :** le résultat convertit commence par un signe plus (+) ou moin (-).

**Blank** : si la valeur n’est pas négative, la sortie commence par un espace et non un plus ; les valeurs négatives commencent toujours par un signe moins (-).

***Précision pour les rationnels :***

Pour les rationnels, nous pouvons indiquer la *largeur minimale* de la valeur à afficher et la *précision* du nombre à afficher. La précision par défaut est fixée à six décimales. Les positions décimales sont arrondies à la valeur la plus proche.

Exemples

|  |  |  |
| --- | --- | --- |
| **printf("%f", 100.123);** | **==>** | **100.123000** |
| **printf("%12f", 100.123);** | **==>** | **\_\_100.123000** |
| **printf("%.2f", 100.123);** | **==>** | **100.12** |
| **printf("%5.0f", 100.123);** | **==>** | **\_\_100** |
| **printf("%10.3f", 100.123);** | **==>** | **\_\_\_100.123** |
| **printf("%.4f", 1.23456);** | **==>** | **1.2346** |

**Séquences d’échappement :**

‘ \n ’ : nouvelle ligne

‘ \t ’ : tabulation horizontale

‘ \b ’: espacement arrière

‘ \r ’ : retour chariot

‘ \a’ : bip sonore

‘ \\ ’ : back slach

‘ \″ ’: guillemet

**Expressions :**

Une expression peut être :

* Une valeur : 255, 1.2, ‘a’,
* Une variable : a, moy,..
* Combinaison entre variables et opérateurs : (a+b)/2, 2.0\* b,..

### 7.1.2. Autres fonctions de sorties

* **Affichage d’un caractère:**

La fonction **putchar** permet d’afficher un caractère : **c** étant une variable de type **char,** l’écriture **putchar(c);** est équivalente à **printf("%c\n",c);**

Exemple:

**char A = 225;**

**char B = '\a';**

**int C = '\a';**

**putchar('x'); /\* afficher la lettre x \*/**

**putchar('?'); /\* afficher le symbole ? \*/**

**putchar('\n'); /\* retour à la ligne \*/**

**putchar(65); /\* afficher le symbole avec le code 65 (ASCII: 'A')\*/**

**putchar(A); /\* afficher la lettre avec le code 225 (ASCII: 'ß')\*/**

**putchar(B); /\* beep sonore \*/**

**putchar(C); /\* beep sonore \*/**

* **Affichage d’un texte:**

La fonction **puts** permet d’afficher un texte : l’écriture **puts("Bonjour");** est équivalente à **printf("Bonjour\n");**

*REMARQUE***:**

Il vaut mieux utiliser **puts** et **putchar** si cela est possible, ces fonctions, non formatées, sont d'exécution plus rapide, et nécessitent moins de place en mémoire lors de leur chargement.

### 7.2. Lecture des données

### 7.2.1 La fonction printf

Permet de lire à partir du clavier des données. La saisie s'arrête avec ***"RETURN"***, les éléments saisis s'affichent à l'écran.

**scanf (“<format> “, <Adrv1>, <Adrv2>, ...);**

* **<format>:** format de lecture des données
* Autant de format que de données à lire
* **<Adrv>**: &NomVariable
* Spécificateur de format : **%** [largeur\_max] **caractère\_ de\_type**

Exemple:

**char alpha;**

**int i ;**

**float r;**

**scanf("%c",&alpha); /\* saisie d'un caractère \*/**

**scanf("%d",&i); /\* saisie d'un entier en décimal \*/**

**scanf("%x",&i); /\* saisie d'un entier en hexadécimal\*/**

**scanf("%f",&r); /\* saisie d'un réel \*/**

**int jour, mois, annee;**

**scanf("%d %d %d", &jour, &mois, &annee);**

**/\*lit trois entiers relatifs, séparés par des espaces, tabulations ou interlignes. Les valeurs sont attribuées respectivement aux trois variables JOUR, MOIS et ANNEE.\*/**

**Remarque** :

* Si l'utilisateur ne respecte pas le format indiqué dans **scanf**, la saisie est ignorée. Aucune erreur n'est générée.

Exemple:

**char alpha;**

**scanf("%d",&alpha);**

* **Si l'utilisateur saisie 97 tout va bien, alpha devient le caractère dont le code ASCII vaut 97.**
* **Si l'utilisateur saisi a, sa saisie est ignorée.**
* **scanf** retourne comme résultat le nombre de données correctement reçues (type **int**).
* **Indication de la largeur maximale**

Pour tous les spécificateurs, nous pouvons indiquer la *largeur maximale* du champ à évaluer pour une donnée. Les chiffres qui passent au-delà du champ défini sont attribués à la prochaine variable qui sera lue !

Soient les instructions:

**int A,B;**

**scanf("%4d %2d", &A, &B);**

Si nous entrons le nombre **1234567**, nous obtiendrons les affectations suivantes:

**A=1234**

**B=56**

Le chiffre 7 sera gardé pour la prochaine instruction de lecture.

* **Les signes d'espacement**

Lors de l'entrée des données, une suite de signes d'espacement (espaces, tabulateurs, interlignes) est évaluée comme un seul espace. Dans la chaîne de format, les symboles **\t**, **\n**, **\r** ont le même effet qu'un simple espace.

Pour la suite d'instructions

**int JOUR, MOIS, ANNEE;**

**scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);**

Les entrées suivantes sont correctes et équivalentes:

**12 4 1980**

ou

**12 004 1980**

ou

**12**

**4**

**1980**

* **Formats 'spéciaux'**

Si la chaîne de format contient aussi d'autres caractères que des signes d'espacement, alors ces symboles doivent être introduits exactement dans l'ordre indiqué.

La suite d'instructions :

**int JOUR, MOIS, ANNEE;**

**scanf("%i/%i/%i", &JOUR, &MOIS, &ANNEE);**

|  |  |
| --- | --- |
| accepte les entrées: | rejette les entrées: |
| **12/4/1980** | **12 4 1980** |
| **12/04/01980** | **12 /4 /1980** |

* **Nombre de valeurs lues**

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.

La suite d'instructions :

**int JOUR, MOIS, ANNEE, RECU;**

**RECU = scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);**

Réagit de la façon suivante ( **-** valeur indéfinie):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Introduit: |  | **RECU** | **JOUR** | **MOIS** | **ANNEE** |
| **12 4 1980** | **==>** | **3** | **12** | **4** | **1980** |
| **12/4/1980** | **==>** | **1** | **12** | **-** | **-** |
| **12.4 1980** | **==>** | **1** | **12** | **-** | **-** |
| **12 4 19.80** | **==>** | **3** | **12** | **4** | **19** |

### 7.2.2. Autres fonctions de lectures

* **Fonction GETCH**

La fonction **getch**, appartenant à la bibliothèque **conio.h**, permet la saisie clavier d'un caractère alphanumérique **sans écho écran**. La saisie s'arrête dès que le caractère a été frappé.

On peut utiliser **getch** de deux façons:

* **Sans retour de variable au programme**:

**printf("POUR CONTINUER FRAPPER UNE TOUCHE ");**

Exemple:

**getch();**

* **Avec retour de variable au programme** :

Exemple:

**char alpha;**

**printf("ENTRER UN CARACTERE (PAS DE RETURN) ");**

**alpha = getch();**

**printf("\nVOICI CE CARACTERE: %c",alpha);**

* **Fonction GETCHE**

La fonction **getche**, appartenant à la bibliothèque **conio.h**, permet la saisie clavier d'un caractère alphanumérique **avec écho écran**, elle est semblable a getch avec une seul différence que le caractère saisie avec getche s'affiche a l'écran.

* **Fonction GETCHAR**

La fonction **getchar** permet la saisie d'un caractère (char). Elle appartient à la bibliothèque **stdio.h**.

Les deux écritures suivantes sont équivalentes:

**char c;**

**printf("ENTRER UN CARACTERE: ");**

**scanf("%c",&c);**

**char c;**

**printf("ENTRER UN CARACTERE: ");**

**c = getchar();**

**Remarque** :

Non formatée, la fonction **getchar** est moins gourmande en place mémoire que **scanf**. Il vaut mieux l'utiliser quand cela est possible.

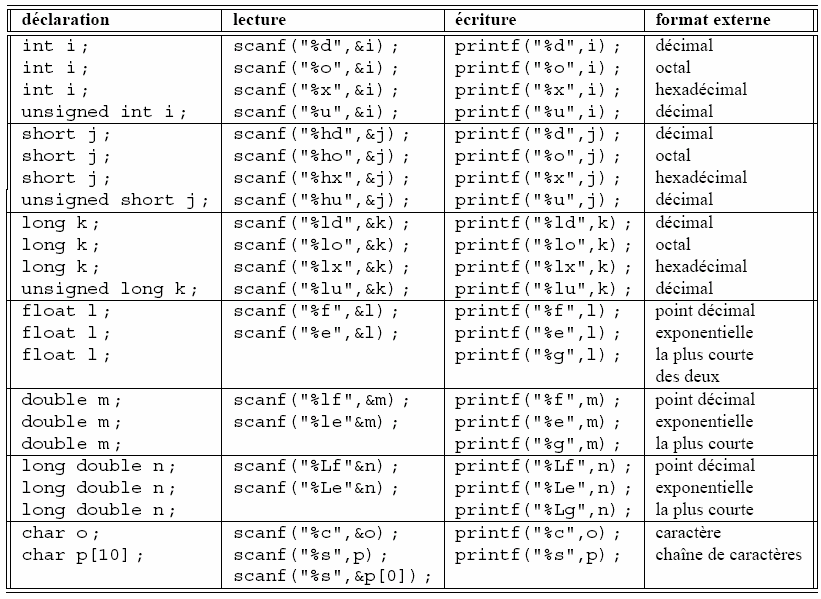
* **Fonction GETS**

La fonction **gets** permet la saisie d’un texte. On va détailler cette fonction en plus dans le chapitre chaîne de caractère.

**char s[50] ;**

**gets(s) ;**

Figure 1 Exemples de PRINTF et SCANF



## 8. Les Exercices

### 8.1 Exercice 1

Tester l’exemple 1.1 et donner le résultat d’exécution.

### 8.2 Exercice 2

Copier ce code sur votre machine et corriger les erreurs.

|  |
| --- |
| /\*Ceci est un commentaire \*/  #include Void main() {  /\* Mon programme principal  Printf("Bienvenue à ISET Nabeul ");  print("You are Weclome ")  printf("Marhaba bikom); |

### 8.3 Exercice 3

|  |
| --- |
| #include<stdio.h>  void main()  { int x,y,a,b,c ;  printf("donner la valeur de a,b et c ");  scanf("%d%d%d",&a,&b,&c);  x=a ;  y=b ;  a=c ;  b=x ;  c=y ;  printf( " la valeur de a est %d",a); printf( " la valeur de b est %d",b) ;  printf( " la valeur de c est %d",c);  } |

1. Donner une trace du programme pour a=5,b=8 et c=9.
2. Que fait ce programme ?

### 8.4. Exercice 4

Ecrire un programme C qui lit le rayon d’un cercle puis affiche son périmètre et sa surface

### 8.5. Exercice 5

Ecrire un programme c qui calcule le reste de la division entière entre deux entiers a et b. Exemple :

a=8

b=6

a/b=8/6=1 alors le reste=2

### 8.6. Exercice 6

Ecrire un programme C qui permet de convertir un entier exprimé en seconde à son équivalent en heure (h), minute (mn) et seconde (s). Exemple : v = 4000s => 1h : 6mn : 40s

### 8.7. Exercice 7

Ecrire un programme qui lit un caractère au clavier et affiche le caractère ainsi que son code numérique (code ASCII) : a) en employant getchar et printf, b) en employant getch et printf.

### 8.8. Exercice 8

Ecrire un programme C qui demande le nom, le prénom, l’âge et la taille en cm d’une personne puis répond comme suit : « Bonjour Nom prénom ! Tu as déjà Age ans et tu mesures taille à bientôt !!».