

Cours Programmation I (chapitres 3&4)

Licence Fondamentale SMI (semestre 3)

Pr. Mouad BEN MAMOUN

ben_mamoun@fsr.ac.ma

Année universitaire 2014/2015

Chapitre 3

Entrées-sorties

Les instructions de lecture et d'écriture

- Il s'agit des instructions permettant à la machine de dialoguer avec l'utilisateur
 - Dans un sens la **lecture** permet à l'utilisateur d'entrer des valeurs au clavier pour qu'elles soient utilisées par le programme
 - Dans l'autre sens, **l'écriture** permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran (ou en les écrivant dans un fichier)
- La bibliothèque standard **<stdio>** contient un ensemble de fonctions qui assurent la lecture et l'écriture des données. Dans ce chapitre, nous allons en discuter les plus importantes:
 - **printf()** écriture formatée de données
 - **scanf()** lecture formatée de données

Ecriture formatée de données: printf ()

- la fonction **printf** est utilisée pour afficher à l'écran du texte, des valeurs de variables ou des résultats d'expressions.
- Syntaxe : **printf("format", expr1, expr2, ...);**
 - **expr1,...** : sont les variables et les expressions dont les valeurs sont à représenter
 - **Format** : est une chaîne de caractères qui peut contenir
 - du texte
 - des séquences d'échappement ('\\n', '\\t', ...)
 - des spécificateurs de format : un ou deux caractères précédés du symbole %, indiquant le format d'affichage

Rq : Le nombre de spécificateurs de format doit être égale au nombre d'expressions!

Spécificateurs de format

SYMBOLE	TYPE	AFFICHAGE COMME
<i>%d ou %i</i>	<i>int</i>	entier relatif
<i>%u</i>	<i>unsigned int</i>	entier naturel non signé
<i>%c</i>	<i>char</i>	caractère
<i>%o</i>	<i>int</i>	entier sous forme octale
<i>%x ou %X</i>	<i>int</i>	entier sous forme hexadécimale
<i>%f</i>	<i>float, double</i>	réel en notation décimale
<i>%e ou %E</i>	<i>float, double</i>	réel en notation exponentielle
<i>%s</i>	<i>char*</i>	chaîne de caractères

Séquences d'échappement

- l'affichage du texte peut être contrôlé à l'aide des *séquences d'échappement* :
 - **\n** : nouvelle ligne
 - **\t** : tabulation horizontale
 - **\a** : signal sonore
 - **\b** : retour arrière
 - **\r** : retour chariot
 - **\v** : tabulation verticale
 - **\f** : saut de page
 - **** : back slash (\)
 - **\'** : apostrophe
 - **\''** : guillemet

Exemples de printf()

```
#include<stdio.h>

main()
{ int i=1 , j=2, N=15;
  printf("la somme de %d et %d est %d \n", i, j, i+j);
  printf(« N= %x \n" , N);
  char c='A' ;
  printf(" le code Ascii de %c est %d \n", c, c);
}
```

Ce programme va afficher :

la somme de 1 et 2 est 3
N=f
le code Ascii de A est 65

Remarque : Pour pouvoir traiter correctement les arguments du type long, il faut utiliser les spécificateurs %ld, %li, %lu, %lo, %lx

Exemples de printf()

```
#include<stdio.h>
main()
{ double x=10.5, y=2.5;
  printf("%f divisé par %f égal à %f \n", x, y, x/y);
  printf("%e divisé par %e égal à %e\n", x, y, x/y);
}
```

Ce programme va afficher :

10.500000 divisé par 2.500000 égal à 4.200000

1.050000e+001 divisé par 2.500000e+000 égal à 4.200000e+000

Remarque : Pour pouvoir traiter correctement les arguments du type long double, il faut utiliser les spécificateurs %lf et %le

Remarques sur l'affichage

- Par défaut, les entiers sont affichés sans espaces avant ou après
- Pour agir sur l'affichage → un nombre est placé après % et précise le nombre de caractères **minimum à utiliser**

- Exemples : `printf("%4d", n);`

`n = 20` → `~~20` (~ : espace)

`n=56123` → `56123`

`printf("%4X", 123);` → `~~7B`

`printf("%4x", 123);` → `~~7b`

Remarques sur l'affichage

- Pour les réels, on peut préciser la *largeur minimale* de la valeur à afficher et le nombre de chiffres après le point décimal.
- La précision par défaut est fixée à six décimales. Les positions décimales sont arrondies à la valeur la plus proche.
- Exemples :

<code>printf("%f", 100.123);</code>	→ 100.123000
<code>printf("%12f", 100.123);</code>	→ ~100.123000
<code>printf("%.2f", 100.123);</code>	→ 100.12
<code>printf("%5.0f", 100.123);</code>	→ ~100
<code>printf("%10.3f", 100.123);</code>	→ ~~~100.123
<code>printf("%.4f", 1.23456);</code>	→ 1.2346

Lecture formatée de données: scanf ()

- la fonction **scanf** permet de lire des données à partir du clavier
- Syntaxe : **scanf("format", AdrVar1, AdrVar2, ...);**
 - **Format** : le format de lecture de données, est le même que pour *printf*
 - **adrVar1, adrVar2, ...** : adresses des variables auxquelles les données seront attribuées. L'adresse d'une variable est indiquée par le **nom** de la variable **précédé** du signe **&**

Exemples de scanf()

```
#include<stdio.h>

main()
{ int i , j;
  scanf("%d%d", &i, &j);
  printf("i=%d et j=%d", i, j);
}
```

ce programme permet de lire deux entiers entrés au clavier et les afficher à l'écran.

Remarque : pour lire une donnée du type **long**, il faut utiliser les spécificateurs **%ld, %li, %lu, %lo, %lx**.

Exemples de scanf()

```
#include<stdio.h>
main()
{ float x;
  double y;
  scanf("%f %lf", &x, &y);
  printf("x=%f et y=%f", x,y);
}
```

ce programme permet de lire un réel simple et un autre double du clavier et les afficher à l'écran

Remarque : pour lire une donnée du type **double**, il faut utiliser **%le** ou **%lf** et pour lire une donnée du type **long double**, il faut utiliser **%Le** ou **%Lf**

Chapitre 4

Structures de contrôle

Structures de contrôle

- Les structures de contrôle définissent la façon avec laquelle les instructions sont effectuées. Elles conditionnent l'exécution d'instructions à la valeur d'une expression
- On distingue :
 - **Les structures alternatives (tests)** : permettent d'effectuer des choix càd de se comporter différemment suivant les circonstances (valeur d'une expression). En C, on dispose des instructions : ***if...else*** et ***switch***.
 - **Les structures répétitives (boucles)** : permettent de répéter plusieurs fois un ensemble donné d'instructions. Cette famille dispose des instructions : ***while***, ***do...while*** et ***for***.

L'instruction if...else

- Syntaxe : ***If (expression)***
bloc-instruction1
else
bloc-instruction2
 - *bloc-instruction* peut être une seule instruction terminée par un point-virgule ou une suite d'instructions délimitées par des accolades { }
 - *expression* est évaluée, si elle est vraie (valeur différente de 0), alors *bloc-instruction1* est exécuté. Si elle est fausse (valeur 0) alors *bloc-instruction2* est exécuté
- La partie *else* est facultative. S'il n'y a pas de traitement à réaliser quand la condition est fausse, on utilisera simplement la forme :
If (expression) *bloc-instruction1*

if...else : exemples

- *float a, b, max;*
 if (a > b)
 max = a;
 else
 max = b;
- *int a;*
 if ((a%2)==0)
 printf(" %d est paire" ,a);
 else
 printf(" %d est impaire ",a);

Imbrication des instructions if

- On peut imbriquer plusieurs instructions if...else
- Ceci peut conduire à des confusions, par exemple :
 - if (N>0)
 if (A>B)
 MAX=A;
 else MAX=B; (interprétation 1 : si N=0 alors MAX prend la valeur B)
 - if (N>0)
 if (A>B)
 MAX=A;
 else MAX=B; (interprétation 2 : si N=0 MAX ne change pas)
- En C un *else* est toujours associé au dernier *if* qui ne possède pas une partie *else* (c'est l'interprétation 2 qui est juste)

Imbrication des instructions if

- Conseil : pour éviter toute ambiguïté ou pour forcer une certaine interprétation dans l'imbrication des *if*, il vaut mieux utiliser les accolades
- ```
if(a<=0)
 {if(a==0)
 printf("a est nul ");
 else
 printf(" a est strictement négatif ");}
```
- Pour forcer l'interprétation 1: 

```
if (N>0)
 { if (A>B)
 MAX=A;
 }
else MAX=B;
```

# L'instruction d'aiguillage switch :

- Permet de choisir des instructions à exécuter selon la valeur d'une expression qui doit être de type entier

- la syntaxe est :

```
switch (expression) {
 case expression_constante1 : instructions_1; break;
 case expression_constante2 : instructions_2; break;
 ...
 case expression_constante n : instructions_n; break;
 default : instructions;
}
```

- *expression\_constantei* doit être une expression constante **entière**
- Instructions *i* peut être une instruction simple ou composée
- *break* et *default* sont optionnels et peuvent ne pas figurer

# Fonctionnement de switch

---

- *expression est évaluée*
- *si sa valeur est égale à une expression\_constante i, on se branche à ce cas et on exécute les instructions\_i qui lui correspondent*
  - On exécute aussi les instructions des cas suivants jusqu'à la fin du bloc ou jusqu'à une instruction break (qui fait sortir de la structure switch)
- si la valeur de l'expression n'est égale à aucune des expressions constantes
  - Si **default** existe, alors on exécute les instructions qui le suivent
  - Sinon aucune instruction n'est exécutée

# Switch : exemple

---

```
main()
{ char c;
 switch (c) {
 case 'a':
 case 'e':
 case 'i':
 case 'o':
 case 'u':
 case 'y': printf("voyelle\n"); break ;
 default : printf("consonne\n");
 }
}
```

# Les boucles while et do .. while

---

```
while (condition)
{
 instructions
}
```

```
do
{
 instructions
} while (condition);
```

- la condition (dite condition de contrôle de la boucle) est évaluée à chaque itération. Les instructions (corps de la boucle) sont exécutées tant que la condition est vraie, on sort de la boucle dès que la condition devient fausse
- dans la boucle while le test de continuation s'effectue avant d'entamer le corps de boucle qui, de ce fait, peut ne jamais s'exécuter
- par contre, dans la boucle do-while ce test est effectué après le corps de boucle, lequel sera alors exécuté au moins une fois

## Boucle while : exemple

---

Un programme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100

```
main()
{ int i, som;
 i =0; som= 0;
 while (som <=100)
 { i++;
 som+=i;
 }
 printf (" La valeur cherchée est N= %d\n ", i);
}
```

## Boucle do .. while : exemple

---

Contrôle de saisie d'une note saisie au clavier jusqu'à ce que la valeur entrée soit valable (comprise entre 0 et 20)

```
main()
{ int N;
 do {
 printf (" Entrez une note comprise entre 0 et 20 \n");
 scanf("%d",&N);
 } while (N < 0 || N > 20);
}
```

# La boucle for

---

```
for (expr1 ; expr2 ; expr3)
 {
 instructions
 }
```

- L'expression expr1 est évaluée une seule fois au début de l'exécution de la boucle. Elle effectue l'initialisation des données de la boucle
- L'expression expr2 est évaluée et testée avant chaque passage dans la boucle. Elle constitue le test de continuation de la boucle.
- L'expression expr3 est évaluée après chaque passage. Elle est utilisée pour réinitialiser les données de la boucle

# Boucle for : remarques

---

**for (expr1 ; expr2 ; expr3) équivaut à :**

```
{
 instructions
}
```

```
expr1;
while(expr2)
{ instructions
 expr3;
}
```

- En pratique, expr1 et expr3 contiennent souvent plusieurs initialisations ou réinitialisations, *séparées par des virgules*

## Boucle for : exemple

---

Calcul de  $x$  à la puissance  $n$  où  $x$  est un réel non nul et  $n$  un entier positif ou nul

```
main ()
{
 float x, puiss;
 int n, i;
 { printf (" Entrez respectivement les valeurs de x et n \n");
 scanf ("%f %d" , &x, &n);
 for (puiss =1, i=1; i<=n; i++)
 puiss*=x;
 printf (" %f à la puissance %d est égal à : %f", x,n,puiss);
 }
}
```

# L'instruction break

---

- L'instruction break peut être utilisée dans une boucle (for, while, ou do .. while). Elle permet d'arrêter le déroulement de la boucle et le passage à la première instruction qui la suit
- En cas de boucles imbriquées, break ne met fin qu' à la boucle la plus interne

- ```
{int i,j;  
    for(i=0;i<4;i++)  
        for (j=0;j<4;j++)  
            { if(j==1) break;  
              printf("i=%d,j=%d\n ",i,j);  
            }  
}
```

 résultat: i=0,j=0
i=1,j=0
i=2,j=0
i=3,j=0

L'instruction continue

- L'instruction continue peut être utilisée dans une boucle (for, while, ou do .. while). Elle permet l'abandon de l'itération courante et le passage à l'itération suivante
- ```
{int i;
 for(i=1;i<5;i++)
 {printf("début itération %d\n " ,i);
 if(i<3) continue;
 printf(" fin itération %d\n " ,i);
 }
}
```

résultat:    début itération 1  
             début itération 2  
             début itération 3  
             fin itération 3  
             début itération 4  
             fin itération 4