ALGORITHMIQUE AVANCÉE ET PROGRAMMATION

CHAP 1: LES BASES DE LA PROGRAMMATION C

Dr. Ikbal Chammakhi Msadaa

Plan du Chapitre

- Structure d'un programme C
- Création d'un programme C
- Les Types de base du langage C
- Les Entrées sorties:
  - La fonction **printf**
  - La fonction **scanf**
- Les opérateurs et les expressions du langage C
- Les instructions de contrôle
- Les instructions de branchement non conditionnel

Structure d'un programme C

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5

main()
{ int i ;
  float x ;
  float racx ;

  printf ("Bonjour\n") ;
  printf ("Je vais vous calculer %d racines carrées\n", NFOIS) ;

  for (i=0 ; i<NFOIS ; i++)
    { printf ("Donnez un nombre : ") ;
      scanf ("%f", &x)  ;
      if (x < 0.0)
       printf ("Le nombre %f ne possède pas de racine carrée\n", x) ;
       else
        { racx = sqrt (x) ;
          printf ("Le nombre %f a pour racine carrée : %f\n", x, racx) ;
        }
    }
  printf ("Travail terminé - Au revoir") ;
}
```

En-tête

déclarations

bloc

Source: C. Delannoy. Programmer en langage C

Structure d'un programme C

- main()
  - Un en-tête
  - Annonce le programme principal délimité par les accolades « { » et « } »
  - Le programme principal: comme une fonction dont le nom (main) est imposé.
- Les déclarations
  - Sont obligatoires; Elles doivent être regroupées au début du programme.
  - int: entiers relatifs;
  - float: approximation de nombres réels.

Structure d'un programme C

- La fonction printf
  - Fonction prédéfinie utilisée pour écrire des informations
  - Reçoit comme argument:
    - Une chaîne de caractères:
    - plusieurs arguments: un format + des arguments.
    - Le format précise comment afficher les informations données par les arguments suivants
    -
    - %: un *code format*; ici considérer NFOIS comme un entier et l'afficher en décimal (%d).
    - Toujours veiller à accorder le code de format au type de la valeur correspondante.

```
printf ("Bonjour\n") ;
```

```
printf ("Je vais vous calculer %d racines carrées\n", NFOIS) ;
```

Structure d'un programme C

- L'instruction for: `for (i=0 ; i<NFOIS ; i++)`
  - Une des façons en C pour réaliser une répétition (boucle) d'un bloc donné.
  - Avant de commencer la répétition: i = 0
  - Avant chaque nouvelle exécution, examiner la condition i < NFOIS
  - Si elle est satisfaite, exécuter le bloc indiqué sinon passer à la suite. A la fin de chaque exécution, réaliser i++ ⇔ i = i + 1

Structure d'un programme C

- La fonction scanf
  - Fonction prédéfinie pour lire une information au clavier. `scanf ("%f", &x) ;`
  - Possède 2 arguments: un format en chaîne de caractères ("%f") + **la valeur** de la variable à saisir.
  - & est un opérateur signifiant *adresse de*. Donc la valeur saisie sera rangée dans l'emplacement (adresse) correspondant à la variable x.

Structure d'un programme C

- L'instruction if

```
if (x < 0.0)
   printf ("Le nombre %f ne possède pas de racine carrée\n", x) ;
 else
   { racx = sqrt (x) ;
     printf ("Le nombre %f a pour racine carrée : %f\n", x, racx) ;
   }
```

- Notez que C dispose de 3 types d'instructions
  - Les instructions simples, terminées par ;
  - Les instructions de structuration: ex. if ou for
  - Les blocs; délimités par « { » et « } »

Structure d'un programme C

- Les directives à destination du préprocesseur:

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5
```

  - Doivent obligatoirement être écrite à raison d'une par ligne.
  - Doivent obligatoirement commencer en début de ligne.
  - Utilisées pour:
    - Demander d'introduire (avant compilation) des instructions situées dans des fichiers en-têtes: stdio.h pour printf et scanf et math.h pour sqrt.
    - Demander de remplacer systématiquement le symbole NFOIS par 5.

Structure d'un programme C

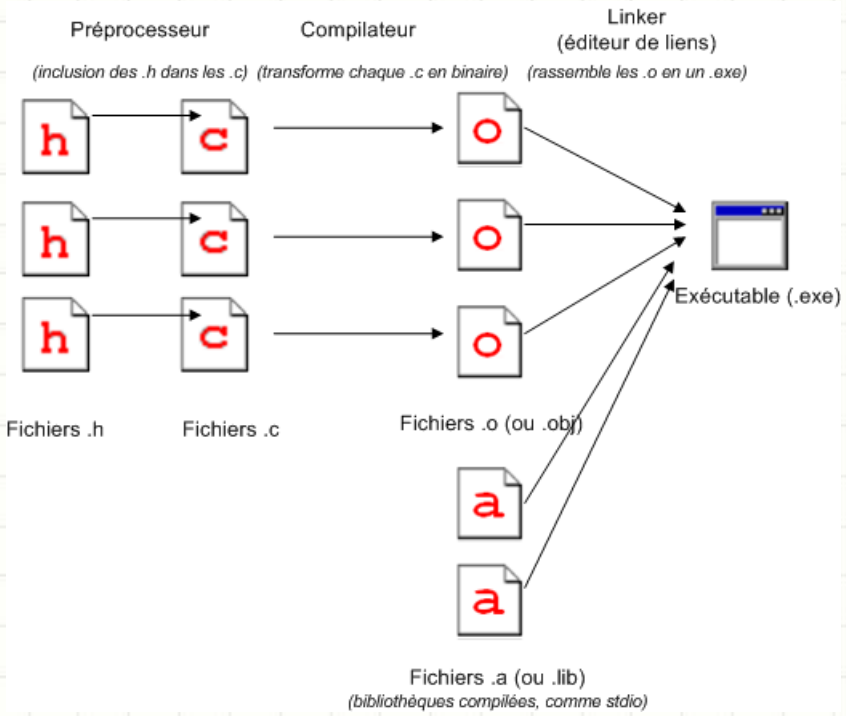
- Quelques règles d'écriture
  - Les identificateurs
  - Les mots clés: réservés par le langage C

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

Source: C. Delannoy. Programmer en langage C

  - Les séparateurs
  - Les commentaires

Création d'un programme C



- **Préprocesseur** :
  - un programme qui démarre avant la compilation.
  - Son rôle est d'exécuter les instructions spéciales qu'on lui a données dans des directives de préprocesseur (#).
- **Compilation** :
  - consiste à transformer les fichiers source en code binaire compréhensible par l'ordinateur. Le compilateur compile chaque fichier .c un à un .o
- **Édition de liens** :
  - le *linker* (ou « éditeur de liens » en français) est un programme dont le rôle est d'assembler les fichiers binaires .o.
  - Il va assembler les .o (temporaires) avec les bibliothèques compilées dont on a besoin (.a ou .lib selon le compilateur).
  - Il les assemble en un seul fichier : l'exécutable final. Cet exécutable a l'extension .exe sous Windows.

Les types de base du langage C

- Les types entiers:
  - short int (ou short)
  - int
  - long int
  - Sur PC, short int et int correspondent à 16 bits; long correspond à 32 bits.
  - Avec 16 bits: de -32 768 à 32 767
  - Avec 32 bits: -2 147 483 648 à 2 147 483 647

Les types de base du langage C

- Les types flottants
  - Permettent de représenter, de manière approchée, une partie des nombres réels.
  - C prévoit 3 types de flottants:
    - float
    - double
    - long double
  - 2 notations pour les constantes flottantes:
    - Décimale: 19.56 -0.42 -.42 9. .36
    - Exponentielle: 6.75E4 6.75e+4 67.5E3
      56e12 56.e12 56.0E12

Les types de base du langage C

- Les types caractères
  - Codés sur un octet
  - L'ensemble des caractères représentables dépend de l'environnement; on est toujours certain de disposer des lettres (majuscules et minuscules), des chiffres, des signes de ponctuation et des différents séparateurs.
  - !! Les caractères nationaux ou semi-graphiques ne figurent pas dans tous les environnements.

Les types de base du langage C

- En plus des caractères imprimables, il y a les « caractères de contrôle » (codes ASCII entre 0 et 31).
  - Exemples:
    - \n (saut de ligne),
    - \t (tabulation horizontale)
    - \a cloche ou bip
    - \\ \' \" \?
- Notation des constantes caractères
  - `a` `Y` `+` `$`
  - `\a` `\x07` `\x7` `\07`

Les entrées sorties: la fonction **printf**

- Les principaux codes de conversion:
  - **c char :** caractère affiché « en clair » (convient aussi à short ou à int compte tenu des conversions systématiques)
  - **d int:** entier (convient aussi à char ou à int, compte tenu des conversions systématiques)
  - **u unsigned int:** entier non signé (convient aussi à unsigned char ou à unsigned short, compte tenu des conversions systématiques)
  - **ld** long
  - **lu** unsigned long
  - **f** double ou float (compte tenu des conversions systématiques float -> double) écrit en notation décimale avec six chiffres après le point (par exemple : 1.234500 ou 123.456789)

Les entrées sorties: la fonction **printf**

- **e** double ou float (compte tenu des conversions systématiques float -> double) écrit en notation exponentielle (mantisse entre 1 inclus et 10 exclu) avec six chiffres après le point décimal, sous la forme **x.xxxxxxe+yyy ou x.xxxxxxe-yyy** pour les nombres positifs et **-x.xxxxxxe+yyy ou -x.xxxxxxe-yyy** pour les nombres négatifs
- **s chaîne de caractères** dont on fournit l'adresse (notion qui sera étudiée ultérieurement)

Les entrées sorties: la fonction **printf**

- Action sur le gabarit d'affichage
  - Par défaut, les entiers sont affichés avec le nombre de caractères nécessaires (sans espaces avant ou après). Les flottants sont affichés avec six chiffres après le point (aussi bien pour le code e que f).
  - Un nombre placé **après %** dans **le code de format** précise un gabarit d'affichage, c'est-à-dire un nombre **minimal** de caractères à utiliser. Si le nombre peut s'écrire avec moins de caractères, printf le fera précéder d'un nombre suffisant **d'espaces** ; en revanche, si le nombre ne peut s'afficher convenablement dans le gabarit imparti, printf utilisera le nombre de caractères nécessaires.

Les Entrées sorties: la fonction **printf**

- **Exemples:**
  - **printf ("%3d", n) ;** /* entier avec 3 caractères minimum */
    - n = 20
    - n = 3
    - n = 2358
    - n = -5200
  - **printf ("%f", x) ;** /* notation décimale gabarit par défaut (6 chiffres après point) */
    - x = 1.2345
    - x = 12.3456789
  - **printf ("%10f", x) ;** /* notation décimale - gabarit mini 10 (toujours 6 chiffres après point) */
    - x = 1.2345
    - x = 12.345
    - x = 1.2345E5
  - **printf ("%e", x) ;** /* notation exponentielle - gabarit par défaut: (6 chiffres après point) */
    - x = 1.2345
    - x = 123.45
    - x = 123.456789E8
    - x = -123.456789E8

Les entrées sorties: la fonction **printf**

- **Actions sur la précision**
  - Pour les types flottants, on peut spécifier un nombre de chiffres (éventuellement inférieur à 6) après le point décimal (aussi bien pour la notation décimale que pour la notation exponentielle).
  - Ce nombre doit apparaître, précédé d'un point, avant le code de format (et éventuellement après le gabarit).

Les entrées sorties: La fonction **printf**

- Exemples
  - **printf ("%10.3f", x) ;** /* notation décimale, gabarit mini 10 et 3 chiffres après point */
    - x = 1.2345
    - x = 1.2345E3
    - x = 1.2345E7
  - **printf ("%12.4e", x) ;** /* notation exponentielle, gabarit mini 12 et 4 chiffres après point */
    - x = 1.2345
    - x = 123.456789E8

Les entrées sorties: La fonction **scanf**

- Pour chaque code de conversion, nous précisons le type de la *lvalue* correspondante:
  - **c char**
  - **d int**
  - **u unsigned int**
  - **hd short int**
  - **hu unsigned short**
  - **ld long int**
  - **lu unsigned long**
  - **f ou e** float écrit indifféremment dans l'une des deux notations : décimale (éventuellement sans point, c'est-à-dire comme un entier) ou exponentielle (avec la lettre e ou E)
  - **lf ou le double** avec la même présentation que ci-dessus
  - **s** chaîne de caractères dont on fournit l'adresse (notion qui sera étudiée ultérieurement)

Les entrées sorties: La fonction **scanf**

- Les codes de format correspondant à un nombre entraînent:
  - l'avancement éventuel du pointeur jusqu'au premier caractère différent d'un séparateur.
  - Puis scanf prend en compte tous les caractères suivants jusqu'à la rencontre d'un séparateur (si aucun gabarit n'est précisé)

Les Entrées sorties: La fonction **scanf**

- Exemples:
  - **scanf ("%d%d", &n, &p) ;**
  - **scanf ("%c%d", &c, &n) ;**
  - **scanf ("%d%c", &n, &c) ;**
- Imposition d'un gabarit maximal
  - **scanf ("%3d%3d", &n, &p)**
  - **scanf ("%d^%c", &n, &c) ;**

/* ^ désigne un espace
%d^%c est différent de %d%c */

- 12^a@
- 12^^^a@
- 12@a@

- 12^25@
- ^12^^25^^@
- 12@
@
^25@

- a25@
- a^^25@
- 12 a@

- 12^25@
- ^^^^^12345@
- 12@
25@

putchar et getchar (de stdio.h)

- L'expression :

c = getchar();

joue le même rôle que :

scanf ("%c", &c);

- La 1ère est toutefois plus rapide puisqu'elle ne fait pas appel au mécanisme d'analyse de format.

- L'expression :

putchar (c);

- joue le même rôle que :

printf ("%c", c);

- La 1ère est toutefois plus rapide puisqu'elle ne fait pas appel au mécanisme d'analyse de format.

Les opérateurs et les expressions

- Les opérateurs arithmétiques: +,-,*,/,% (reste de la division - modulo)
- Les opérateurs relationnels: **<**, **>**, **<=**, **>=**, **==**, **!=**
- Les opérateurs logiques booléen: **&&**, **||**, **!**
  - Vrai est représenté par 1 (toute valeur non nulle)
  - Faux est représenté par 0
  - Exemple: if(!n) ⇔ if(n==0)

Les opérateurs et les expressions

- Les opérateurs composés: += -= *= /= %= &= ^= |= <<= >>=
  - Pour tout opérateur op, l'expression

    expression-1 op= expression-2

  - est équivalente à

    expression-1 = expression-1 op expression-2

Les opérateurs et les expressions

- Les opérateurs d'incrémentation/décrémentation: ++, --
  - Exemple: (i=5)

```
n = ++i – 5;        // cas d'une préincrémentation
n = i++ - 5;        // cas d'une postincrémentation
```

- L'opérateur conditionnel
  - condition ? expression-1: expression-2
  - Exemple: max = a>b ? a:b ⇔ Si a>b alors a sinon b

Les instructions de contrôle

- L'instruction **if** (Branchement conditionnel)

```
if (expression)                          if (expression)
            instruction_1                            instruction_1
    else
        instruction_2
```

  - Un else se rapporte toujours au dernier if rencontré auquel un else n'a pas encore été attribué.

Les instructions de contrôle

- L'instruction **switch** (Branchement multiple)

```
switch (expression)
    { case constante_1 : [ suite_d'instructions_1 ]
      case constante_2 : [ suite_d'instructions_2 ]
                  ..............
      case constante_n : [ suite_d'instructions_n ]
      [ default        :   suite_d'instructions   ]
    }
```

  - Si la valeur de expression est égale à l'une des constantes, la suite d'instructions correspondante est exécutée.
  - Sinon la suite d'instructions correspondant à default est exécutée. L'instruction default est facultative.

Les instructions de contrôle

- L'instruction **do…..while**

```
do     instruction

     while  (expression) ;
```

  - Ici, instruction sera exécutée tant que expression est non nulle. Cela signifie donc que
  - instruction est toujours exécutée au moins une fois

Les instructions de contrôle

- L'instruction **while**

```
while  (expression)

       instruction
```

  - Tant que expression est vérifiée (i.e., non nulle), instruction est exécutée.
  - Si expression est nulle au départ, instruction ne sera jamais exécutée.
  - instruction peut évidemment être une instruction composée

Les instructions de contrôle

- L'instruction **for**

```
for  ( [ expression_1 ] ; [ expression_2 ] ; [ expression_3 ] )
            instruction
```

- Une version équivalente plus intuitive est :

Les instructions de branchement non conditionnel

- Branchement non conditionnel **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.
  - Elle permet **d'interrompre le déroulement de la boucle**, et passe à la première instruction qui suit la boucle. En cas de boucles imbriquées, break fait sortir de la boucle la plus interne
- Branchement non conditionnel **continue**
  - L'instruction continue permet de **passer directement au tour de boucle suivant**, sans exécuter les autres instructions de la boucle.

Les instructions de branchement non conditionnel

```
main()
{
   int i ;
   for ( i=1 ; i<=10 ; i++ )
      { printf ("début tour %d\n", i) ;
        printf ("bonjour\n")
        if ( i==3 ) break ;
        printf ("fin tour %d\n", i) ;
      }
   printf ("après la boucle") ;
}
```

Les instructions de branchement non conditionnel

```
main()
{   int i ;
    for ( i=1 ; i<=5 ; i++ )
       { printf ("début tour %d\n", i) ;
         if (i<4) continue ;
         printf ("bonjour\n") ;
       }
}
```

Les instructions de branchement non conditionnel

```
main()
{  int n ;
   do
      { printf ("donnez un nb>0 : ") ;
        scanf ("%d", &n) ;
        if (n<0) { printf ("svp >0\n") ;
                   continue ;
                 }
        printf ("son carré est : %d\n", n*n) ;
      }
   while(n) ;
}
```

Les instructions de branchement non conditionnel

- L'instruction **goto**
  - Elle permet le branchement en un emplacement quelconque du programme.

```
main()
{
   int i ;
   for ( i=1 ; i<=10 ; i++ )
      { printf ("début tour %d\n", i) ;
        printf ("bonjour\n") ;
        if ( i==3 ) goto sortie ;
        printf ("fin tour %d\n", i) ;
      }
   sortie : printf ("après la boucle") ;
}
```