UNIVERSITÉ ALIOUNE DIOP DE BAMBEY
UFR SCIENCES APPLIQUÉES ET TECHNOLOGIE DE L'INFORMATION ET DE LA COMMUNICATION
DÉPARTEMENT INFORMATIQUE

Chapitre 2 - CONCEPTS DE BASE DU LANGAGE C

Programmation en LANGAGE C
AMRT 2
2016-2017

Le langage C est un langage compilé qui a les caractéristiques suivantes:

- langage typé: tout objet doit avoir un type (caractère, entier, réel...)

- langage déclaratif: tout objet doit être déclaré avant son utilisation.

- propose un certain nombre de fonctions prédéfinies dans des bibliothèques.

I. STRUCTURE D'UN PROGRAMME EN C

Un programme en C se compose d'un programme principal et éventuellement de sous programmes.

Il est écrit dans un fichier portant l'extension .c. Ce fichier, appelé fichier source du programme, contient généralement les parties suivantes:

- directives préprocesseur ou directives de compilation
- définition des structures et des types (s'il y en a)
- déclaration des variables globales (s'il y en a)
- définition des sous programmes (fonctions) (s'il y en a)
- programme principal (fonction main)

I-1. Programme principal

En C, toutes les fonctions ont la même structure :

- le type de retour
- le nom de la fonction
- la liste d'arguments mis entre parenthèses et séparés par des virgules
- le corps de la fonction contenu entre accolades.

Il doit y avoir une fonction main dans tout programme C. C'est la fonction qui démarre le programme. On l'appelle le programme principal.

Le programme principal suit cette structure, mais son type de retour, son nom et ses arguments sont imposés:

```
int main(void)
{
/* déclaration des variables utilisées dans la fonction
   main */
/* instructions du programme*/
}
```

Le nom du programme principal est main; son type de retour est entier (int).

Il ne prend en général pas d'arguments. Dans ce cas, on met le mot void entre les parenthèses (ou rien du tout)

Dans certains cas il peut en prendre des arguments.

I-2. Exemple de programme C

```
#include <stdio.h>
int main( ) /* programme principal*/
{
printf("Bonjour");
return 0;
}
```

Dans ce programme, #include <stdio.h> est une directive de compilation qui indique qu'on va utiliser des fonctions de la bibliothèque standard stdio.h (STandarD Input/Output).

Une bibliothèque (ou librairie) standard est un fichier dans lequel est défini un ensemble de fonctions prêtes à l'emploi facilitant ainsi le travail du programmeur.

Dans la fonction main, on appelle la fonction printf, qui est définie dans la bibliothèque stdio.h pour afficher à l'écran la chaîne de caractères Bonjour.

Comme toute instruction, l'appel à la fonction printf se termine par un point virgule.

L'instruction suivante termine la fonction main qui retourne le résultat 0.

On peut insérer des commentaires, soit entre les balises /* */, soit après // jusqu'à la fin de la ligne.

Les commentaires sont des parties du programme qui sont ignorées par le compilateur. Ils servent à rendre un programme compréhensible.

I-3. Les directives préprocesseur ou directives de compilation

Elles indiquent au compilateur de procéder à des opérations préalables au début de la compilation. Ces directives se situent en tout début du programme source.

La directive #include

Cette directive permet l'inclusion de librairies dont les éléments seront utilisés dans le programme source.

Le compilateur C fournit un ensemble de librairies mais le programmeur peut aussi créer ses propres librairies.

Syntaxe de cette directive :

#include <fichier>

ou

#include "fichier"

<*fichier*> indique l'utilisation d'une librairie, fournie par le compilateur C, se trouvant dans un répertoire particulier du système.

Les librairies les plus utilisées sont généralement :

- stdio.h qui contient les définitions des fonctions d'Entrée/Sortie (io pour Input/Output).
- string.h qui contient les définitions des fonctions traitant des chaînes de caractères.
- stdlib.h qui contient les définitions de fonctions traitant la conversion de nombres et l'allocation de mémoire.
- ctype.h qui contient les définitions de fonctions traitant la conversion de caractères.

"fichier" indique l'utilisation d'une librairie contenue dans "fichier". Cette librairie est souvent un ensemble de fonctions propres au programmeur ou un fichier de déclaration des variables du programme.

"fichier" peut contenir juste le nom de la librairie ou le chemin d'accès et le nom si elle ne se trouve pas dans le même répertoire que le programme source.

Exemples :

```
#include <stdio.h>
#include "C:\MesProgrammes\definitions.h"
```

La directive #define

Cette directive permet de remplacer dans le programme toutes les occurrences d'une suite de caractères par un nom de substitution.

Cette possibilité permet une meilleure lisibilité du programme source, évite de devoir réécrire à chaque fois une suite longue de caractères souvent utilisés au cours du programme source et permet de définir des constantes pour le programme ou des macros.

Syntaxe de cette directive :

#define nom_de_substitution suite_de_caractères

nom_de_substitution sera utilisé tout au long du programme source pour représenter la suite de caractères suite_de_caractères.

Exemple:

#define VRAI 1

Le mot VRAI sera utilisé pour représenter le chiffre 1 dans le programme source. Mais rien ne vous empêche d'utiliser le chiffre 1.

II. LES TYPES DE BASE DU LANGAGE C

A une variable est associée un type, qui permet de définir la taille de l'espace occupé en mémoire (nombre d'octets).

Dans le langage C, les types de base, dit aussi **types primitifs sont divisés en deux classes:** les entiers et les réels (ou flottants).

II-1. Les entiers

Type	Description	Taille en octets	Valeurs possibles	Format
int	Entier standard	2 ou 4	de -32768 à 32767 (sur 2 octets) de -2147483648 à 2147483647 (sur 4 octets)	%d
short	Entier court	2	de -32768 à 32767	%d
long	Entier long	4	de -2147483648 à 2147483647	%ld

Attention : ces tailles (sauf pour char) sont dépendantes du système d'exploitation, du compilateur ou du processeur utilisé.

L'opérateur **sizeof(un type)** retourne **le nombre d'octets utilisés pour stocker une valeur d'un type** donné.

Sur certains systèmes et compilateurs int est synonyme de short, sur d'autres il est synonyme de long.

Si l'on désire manipuler des entiers **non signés, alors on ajoutera le mot « unsigned » :**

Type	Taille	Valeurs possibles	Format
unsigned int	2 ou 4	de 0 à 65535 (sur 2 octets) de 0 à 4294967295 (sur 4 octets)	%u
unsigned short	2	de 0 à 65535	%u
unsigned long	4	de 0 à 4294967295	%lu

Le type **booléen n'existe pas en C. Par contre, la valeur 0 (au sens binaire de la représentation)** de n'importe quel type signifie faux. Toute autre valeur est vraie.

II-2. Les réels

Type	Description	Taille en octets	Format
float	Réel	4	%f
double	Réel double précision	8	%lf

II-3. Les caractères

Les caractères sont considérés **comme des entiers codés en mémoire sur 1 octet**. A chaque caractère est associé un nombre entier (son code ASCII). Ainsi, toutes les opérations s'appliquant aux entiers peuvent également s'appliquer aux caractères.

Type	Description	Taille en octets	Valeurs possibles	Format
char	Caractère (considéré comme un entier signé)	1	de -128 à 127	%c
unsigned char	caractère (considéré comme un entier non signé)	1	De 0 à 255	%c

En C, la notion de caractère dépasse celle de caractère imprimable, c'est à dire auquel est associé un graphisme (lettres, chiffres, ponctuation, caractères spéciaux,…). Par exemple, les retours chariot, les tabulations, les sauts de page sont aussi des caractères.

Les caractères non imprimables possèdent une représentation conventionnelle utilisant le caractère \ nommé antislash.

Exemples de caractères non imprimables :

- \n retour chariot avec saut de ligne
- \r retour chariot sans saut de ligne
- \t tabulation horizontale
- \v tabulation verticale

III. DÉCLARATION DES VARIABLES ET DES CONSTANTES

III-1. Déclaration d'une variable

En C, toute variable doit être déclarée avant sa première utilisation.

La syntaxe de déclaration d'une variable est:

type identificateur;

Exemple:

L'instruction int a ; permet de réserver un emplacement mémoire pour le stockage d'un entier (type int) qui sera nommé a dans la suite du programme.

Une déclaration peut se faire seule, ou accompagnée d'une initialisation (première valeur).

Exemple :

L'instruction

```
int b=10 ;
```

permet de déclarer un entier b et de lui donner 10 comme première valeur.

On peut déclarer plusieurs variables de même type dans une même instruction en séparant les identificateurs par des virgules.

Exemple :

```
int x, var=2, z ;
```

III-2. Déclaration d'une constante

La syntaxe de déclaration d'une constante est:

const type identificateur valeur;

Exemple :

const int MAX=32767;

On peut utiliser la directive de compilation #define pour définir une constante selon la syntaxe suivante:

#define identificateur valeur

Mais dans ce cas, il n'y a pas de réservation d'emplacement mémoire.

Exemple :

#define MAX 32767

III-3. Contraintes sur les identificateurs

Un identificateur est une suite de caractères parmi :

- les lettres (minuscules ou majuscules, mais non accentuées),
- les chiffres,
- le caractère « souligné » (_).

Le premier caractère d'un identificateur ne peut pas être un chiffre.

Le langage C fait la différence entre les majuscules et les minuscules.

Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. Le C standard (ANSI C) compte 32 mots clefs :

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

IV. LES OPÉRATEURS DU LANGAGE C

IV-1. Les opérateurs arithmétiques

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Reste de la division entière (entre deux entiers)

IV-2. Les opérateurs relationnels

Opérateur	Signification
>	Supérieur
>=	Supérieur ou égal
<	Inférieur
<=	Inférieur ou égal
==	Egal
!=	Différent

IV-3. Les opérateurs logiques

Opérateur	Signification
&&	Et logique
||	Ou logique
!	Négation logique

Exemples :

(a > 10) && (a <= 20) sera vrai si a > 10 et a <= 20

(a > 10)||(b <= 20) sera vrai si a > 10 ou si b <= 20

!(a > 10) sera vrai si a <= 10

V. LES INSTRUCTIONS ÉLÉMENTAIRES

V-1. L'affectation

Cette instruction permet de donner la valeur de l'expression de droite à la variable de gauche.

Le langage C utilise le symbole **=** pour l'affectation.

Pour affecter un caractère, il faut l'entourer d'apostrophes (sinon le compilateur pourrait le confondre avec un nom de variable). Pour une chaîne, il faut utiliser des guillemets.

Exemples :

```
x = 5;
```

La variable x prend pour valeur 5

```
i = i + 1;
```

La variable i prend pour valeur son ancienne valeur augmentée de 1

```
C = 'a';
```

La variable C prend pour valeur le caractère 'a'.

V-2. L'affichage

La fonction **printf** permet **d'afficher des informations à l'écran**

Syntaxe :
printf (" chaîne de caractères " , variable1, variable2, …)

Cette fonction, contenue dans la bibliothèque standard **stdio.h**, attend comme premier paramètre **la chaîne de caractère à afficher** avec éventuellement la description des formats d'affichage des variables à afficher dans cette chaîne :

Les paramètres suivants correspondent**, dans l'ordre, à chaque variable dont on souhaite afficher** la valeur.

Les formats d'affichage sont:

- %d ou %ld pour les entiers (int, short, long)
- %f ou %lf pour les réels (float, double)
- %s pour les chaînes de caractères
- %c pour les caractères

Exemple :

printf("La valeur des variables X et Y sont : %d et %d",X,Y) ;

Les deux %d seront respectivement remplacés à l'écran par la valeur de X et de Y.

La chaîne de caractères peut contenir des caractères spéciaux, par exemple :

\n pour saut de ligne

\t pour les tabulations

Exemple:

printf ("%d", y – x);

Cette instruction affiche 5 si y=10 et x=5.

Exemple :

printf ("la valeur de x est %d et celle de y e st %d", x, y);

En supposant que x ait pour valeur 5 et y pour valeur 10, on aura à l'écran:

La valeur de x est 5 et celle de y est 10.

Pour inclure des sauts de ligne dans un texte, on utilise le caractère \n .

Exemple :

```
printf ( "Bonjour\nCa va?") ;
```

va donner à l'écran :

Bonjour
Ca va?

Exemple :

```
int i =1;
float x = 2 . 0 ;
printf ( " Bonjour \ n " ) ;
printf ( " i = %d \ n " , i ) ;
printf ( " i = %d , x = % f \ n " , i , x ) ;
```

affiche :

Bonjour
i = 1
i = 1, x=2.0

La fonction printf offre en plus la possibilité de définir le nombre d'emplacements à utiliser pour l'affichage d'une valeur.

Exemple :

printf(" valeur de X :%4.3f ", 3.5268) ;

donnera à l'écran valeur de X :¤¤¤3.527 (chaque ¤ représentant un espace)

V-3. La lecture

L'instruction **scanf** permet au programme **de lire des informations saisies au clavier par l'utilisateur.**

Syntaxe :

scanf (" chaîne de formatage " , &variable1 , &variable2, . . .)

Cette fonction, également contenue dans la bibliothèque standard **stdio.h**, attend comme premier paramètre la chaîne décrivant les formats de lecture des variables à lire au clavier. Les paramètres suivants sont, **dans l'ordre, l'adresse des variables dont on souhaite lire la valeur.**

Formats de lecture :

- %d ou %ld pour les entiers (int, short, long)
- %f ou %lf pour les réels (float, double)
- %s pour les chaînes de caractères
- %c pour les caractères

Exemple :

scanf(" %d %d",&X, &Y) ;

Le & permet d'accéder à l'adresse des variables.(**A omettre dans le cas de la lecture d'une chaîne de caractères**).

scanf lit au clavier les valeurs introduites successivement.

Par exemple:

scanf ("%d %d", &a , &b) ;

Si l'on écrit au clavier 2 puis 3, la variable a va prendre pour valeur 2 et b va prendre pour valeur

V-4. Lecture et affichage de caractères

Les fonctions **getchar** et **putchar** de la bibliothèque **stdio.h** permettent respectivement au programme **de lire au clavier** et **d'afficher à l'écran** des **caractères**. Il s'agit de fonctions d'entrées-sorties non formatées.

La fonction getchar retourne un entier (code ASCII) correspondant au caractère lu. Pour mettre le caractère lu dans une variable c de type caractère, on écrit

```
c = getchar( ) ;
```

La fonction putchar permet d'afficher un caractère à l'écran. La syntaxe pour afficher un caractère c à l'écran est la suivante :

```
putchar(c) ;
```

Remarque

La lecture avec la fonction getchar se fait par l'intermédiaire d'une zone mémoire appelée "buffer d'entrée" qui sert à contenir les caractères tapés au clavier par l'utilisateur et qui fonctionne selon le principe d'une "file" (premier arrivé = premier sorti).

Si le buffer d'entrée n'est pas vide, l'instruction **getchar(); lit le caractère le plus ancien** se trouvant dans le buffer d'entrée sans attendre l'utilisateur, et enlève ce caractère du buffer d'entrée.

Si le buffer d'entrée est vide, l'instruction **getchar(); stoppe le déroulement du programme jusqu'à** ce que le buffer d'entrée reçoive un ou plusieurs caractères. L'ajout de caractères dans le buffer d'entrée ne se fait qu'au moment où l'utilisateur tape un retour-chariot (touche **Entrée).**

Pour vider le buffer d'entrée, il faut utiliser l'intruction **fflush(stdin);**

Remarque

Certains compilateurs proposent une bibliothèque appelée **conio.h** qui contient deux fonctions de lecture et d'écriture de caractère. Ce sont respectivement les fonctions **getch** et **putch.**

```
c = getch( ) ;
putch(c) ;
```

La fonction getch lit directement le caractère tapé par l'utilisateur sans utiliser de buffer d'entrée.

Ces deux fonctions ne font pas partie du langage C standard.