Série de TP 0 : langage C

Introduction

Le langage C est un langage évolué et structuré, assez proche du langage machine destiné à des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel ...). Les compilateurs C possèdent les taux d'expansion les plus faibles de tous les langages évolués (rapport entre la quantité de codes machine générée par le compilateur et la quantité de codes machine générée par l'assembleur et ce pour une même application);

Le langage C possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

Exemples: math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

Mise en place de l'environnement

Pour pouvoir programmer en langage C, vous auriez besoin de :

- **Un compilateur :** pour traduire le programme écrit dans un langage évolué (C) en un programme de bas niveau que la machine peut exécuter.

- **Un éditeur du texte :** pour taper la séquence de code de votre programme.

Pour installer le compilateur **GCC**, on utilise la commande : **sudo apt install build-essential**

Compilation et Exécution des Programmes C

Si vous disposez d'un **IDE,** vous trouverez déjà dans l'interface les options de compilation et exécution.

Pour installer **Code ::Blocks IDE**, on utilise la commande : **sudo apt-get install codeblocks.**

Sinon, si vous utilisez un simple éditeur du texte, vous pouvez compiler vos programmes C par le biais du terminal. Pour cela :

- Tapez **touch nom-de-votre-programme.c** afin de créer votre fichier.
-Tapez **gcc nom-de-votre-programme.c -o nom-de-votre-exécutable** afin de compiler votre programme et créer votre exécutable.
- Tapez **./ nom-de-votre-exécutable** afin d'exécuter votre programme.
- Visualisez ensuite le résultat de votre programme sur le terminal.

Remarque : Le langage C distingue les minuscules, des majuscules. Les mots réservés du langage C doivent être écrits **en minuscules.**

Structure générale d'un programme en C

```
<Directives de compilation>
<Déclaration de variables externes>
<Déclaration de fonctions>
main ( )
{
        Corps du programme
        (Commentaires, déclaration de variables et constantes, instructions)
}
<Définition de fonctions>
```

Directives de compilation

Nous en citons les directives **include** et **define:**

#include < nom_fichier>

Indique le fichier de bibliothèque C à inclure dans le programme. Ce fichier contient les définitions de certaines fonctions prédéfinies utilisées dans le programme.

#define expression_à_remplacer expression_de_remplacement

Permet de remplacer un symbole par une constante ou un type ou de faire des substitutions avec arguments dans le cas des macros.

Exemples :

```
#include <stdio.h>  /*ce fichier contient les fonctions d'entrées/sorties comme printf*/
#define pi 3.14      /*pi sera remplacé par la valeur 3.14*/
#define entier int    /*entier sera remplacé par le type prédéfini int*/
#define somme(x,y) x+y     /*la macro somme(x,y) sera remplacée par x+y*/
```

Variables et constantes

1. Déclaration des variables

Type identificateur;
Ou
Type identificateur1, identificateur2, … , identificateur n;

- **Différents types de variables**

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	entier	**16 bits**
short(ou short int)	entier court	**16 bits**
long (ou long int)	entier en double longueur	**32 bits**
char	caractère	**8 bits**
float(ou short float)	réel	**32 bits**
double(ou long float)	réel double précision	**64 bits**
long double	réel en très grande précision	**80 bits**
unsigned	Non signé (positif)	**16 bits**

2. **Déclaration des constantes**

const Type Identificateur = Valeur ;

Opérateurs et Expressions : La liste des opérateurs disponibles est :

- **Arithmétique :** +, -, *, /, %
- **Affectation :** =, +=, -=, *=, /=, %=
- **Incrémentation/décrémentation:** ++, --
- **Comparaison:** <, <=, >, >=, ==, !=
- **Logique:** !, &&, ||
- Mais également: **sizeof** renvoie le nombre d'octets réservés en mémoire pour chaque type d'objet.
  Exemple: n = sizeof(char); /* n vaut 1 */

Structures algorithmiques

- **for** (i=0; i<n; i++) { ... }
- **while** (cond) {...}
- **do** { ...; } **while** (cond);
- **if** (cond) { ... } **else** { ... }

Fonctions d'entrées-sorties

- **Fonction d'écriture « printf » :** **printf("texte", var, cons, expr,…);**
  - le texte à afficher
  - Les spécificateurs de format
  - Caractère d'échappement

Exemples :

printf("Donnez le prix unitaire"); /*le programme affiche Donnez le prix unitaire */
printf("la valeur de i est %d\n ",i); /*le programme affiche la valeur de i et retourne à la ligne*/

Spécificateurs de format	Type
%d	int
%ld	long
%f	float ou double
%c	char

Caractères d'échappement	
'\n'	interligne
'\t'	tabulation horizontale
'\v'	tabulation verticale
'\r'	retour charriot
'\f'	saut de page
'\\'	backslash
'\''	cote

- **Fonction de saisie (lecture) « scanf » :** **scanf("format-1,…", &v1,…);**
  Exemple: scanf("%d",&i); /*le programme lit une valeur entière et l'affecte à i*/

Chaines de caractères

- **Initialisation**

char Identificateur [Taille constante] = "Texte\0" ;
• Le caractère '\0' indique la fin de la chaîne de caractères. Il est conseillé de ne pas l'omettre.

Exemple

```
#define taille1 3 /*taille1: constante de valeur 3*/
#define taille2 4 /*taille2: constante de valeur 4*/
main( ) {
        char t [taille1]="ali";     /*t chaîne de caractères initialisée au mot ali*/
        char a [taille2]="ali\0"; /*a chaîne de caractères initialisée au mot ali et terminée
                                    par le caractère '\0'*/
        }
```

- **Lecture et affichage**

scanf("%s", Chaîne de caractères);
printf("%s", Chaîne de caractères);
gets(Chaîne de caractères);
puts(Chaîne de caractères);
• **scanf** amène uniquement le texte introduit avant le premier blanc (espace) dans la variable à lire.
• **gets** amène tout le texte introduit jusqu'au retour chariot (retour à la ligne) dans la variable à lire.

- **Fonctions sur les chaines de caractères**
  Des fonctions prédéfinies appliquées aux chaînes de caractères sont définies dans le fichier **"string.h"**. Nous en citons :

Fonction	Appliquée à	Retourne	Rôle
strlen	Une chaîne de caractères	Un entier	Retourne la longueur d'une chaîne de caractères.
strcmp	Deux chaînes de caractères	Un entier	Compare deux chaînes et retourne 0 si elles sont égales, une valeur différente sinon.
strcpy	Deux chaînes de caractères	Rien (void)	Copie une chaîne en deuxième argument dans une autre en premier argument.

Fonctions

- **Déclaration**

```
Void ou Type_Résultat Nom_Fonction (Void ou Type Param1,…,Type Paramn);
{
  Corps de la fonction (instructions)
}
```

- **Appel de fonctions**

  - **Si la fonction retourne un résultat et admet des paramètres :**
    Variable = Nom_Fonction (Paramètres effectifs);
  - **Si la fonction retourne un résultat et n'admet pas de paramètres :**
    Variable = Nom_Fonction ( );
  - **Si la fonction ne retourne rien et admet des paramètres**
    Nom_Fonction (Paramètres effectifs);
  - **Si la fonction ne retourne rien et n'admet pas de paramètres**
    Nom_Fonction ( );

Tableaux

- **Déclaration**

Tableaux à une dimension :

Type Identificateur [Taille] ;
La **Taille** du tableau est le nombre de ses éléments. Elle doit être une constante définie avant ou au moment de la déclaration.

Tableaux à deux dimensions :

Type Identificateur [Taille1] [Taille2] ;

- **Initialisation**

Tableaux à une dimension :

Type Identificateur [Taille] = {Valeur1, Valeur2,…,Valeurn};

Tableaux à deux dimensions :

Type Identificateur [Taille1] [Taille2] = {Liste1, Liste 2} ;
Où Liste i = {Valeur1, Valeur2,…,Valeurn}

- **Lecture, écriture et affectation**
  Exemples :

```
#include<stdio.h>
#define  taille 20
main( ){
  int i, t [taille];
 for(i=0;i<taille;i++)
            scanf ("%d",&t[i]); /*Lecture*/
 for(i=0;i<taille;i++)
            printf ("%d\n",t[i]); /*Ecriture*/
 for(i=0;i<taille;i++)
           t[i]=i; /*Affectation*/
}
```

```
#include<stdio.h>
#define  taille1 8
#define taille2 11
main( ) {
    int t [taille1][taille2];
    int i, j;
    for(i=0;i<taille1;i++)
        for (j=0; j<taille2;j++)
           scanf ("%d",&t[i][j]); /*Lecture*/
              ……..
}
```

Structures : La définition d'un type structure peut être effectuée au moyen des mots réservés **struct** ou **typedef**.

a) Définition par struct

```
struct Nom_Structure
{
  Type1 Champ1;
     ….
   Typen Champn;
} ;
struct Nom_Structure Nom_Variable;
```

b) Définition par typedef

```
typedef struct
{
  Type1 Champ1;
     ….
   Typen Champn;
}Nom_Type_Structure;
Nom_Type_Structure Nom_Variable;
```

Exemples

```
struct date{
          int jour;
          int mois;
          int annee;
} ;
struct date d1 ;
```

```
typedef struct{
          int jour;
          int mois;
          int annee;
}date ;
date d1 ;
```

- **Accès aux champs d'une structure : nom_variable_structure.nom_champ**

  Exemple : d1.jour=1;

  d1.mois=3;

  d1.annee=2000;

Exercice 01 : (Tableaux)

Ecrire un programme C qui lit un entier n. Puis n autres entiers inférieurs à 100, dans un tableau. Et affiche le nombre d'occurrences de chaque élément du tableau de la façon suivante :
Si le tableau est : 1 2 5 2 1 2, on affiche :
1 est répété 2 fois.
2 est répété 3 fois.
5 est répété 1 fois.
*Pas nécessairement dans un ordre précis, mais chaque élément ne doit être cité qu'une seule fois.

Exercice 02: (Structures)

Ecrire un programme C, qui lit un ensemble de villes avec leur nombre d'habitants dans un tableau de structures, les trie et les affiche dans l'ordre croissant des nombres d'habitants.

Exercice 03 : (Structures)

Ecrire un programme C, qui lit les noms complets des étudiants et leurs moyennes dans un tableau de structures. Puis actualise ces moyennes en ajoutant un bonus de:

- *1* point pour les étudiants ayant une note strictement inférieure à *10*.
- *0.5* point pour les étudiants ayant une note entre *10* et *15* incluses.

N.B.: la structure doit avoir deux éléments: une chaîne de caractères et un réel.