Introduction au Langage C

I – Opérations de bases et premier programme

A - Premier pas

Extension de type :

.c (fichier de code)

.h (fichier d'entête)

Compilation :

création d'un fichier exécutable permettant de lancer l'application

Syntaxe :

- transformer code .c en a.out

  gcc nom_du_fichier.c

- transformer code .c en .exec

  gcc -o mon_fichier.exe nom_de_fichier_de_code.c

Premier programme (afficher Hello Word à l'écran)

+ Code :

```
#include <stdio.h>

int main(){
    printf(« Hello World \n ») ;
    return 0 ;
}
```

!!! ‘ ‘ pour un caractère

+ Exécution :

$ gcc **-Wall -Wpedentic** hello.c => **a.out**

./a.out

!!! pour renommer le fichier a.out :

gcc -o hello hello.c

B - Opérations de bases

Structure d'un fichier

```
/**
        Commentaire
**/

* Programme principal

int main (){
        instruction ;
        instruction ;
        return 0 ;
}
```

Le langage C est typés, la déclaration est obligatoire avant toute utilisation de variable.

Déclaration de variable

```
type nom_de_variable ;
```

Affectation

```
nom_de_variable = valeur ;
```

Type élémentaire

Type caractère : char
- → signe (negatif) : -128 …
- → unsigned (positif) : 0 … 255

Type entier : short (2 octets), int (4 octets), long (8 octets)
Type réel : float, double, long, double

- pas de type boolean => 0 pour faux et non nul pour vrai
- pas de type chaîne de caractère

Affichage

- Chaîne de caractère avec retour chariot :

```
puts(« Hello ») ; => affiche et retourne à la ligne
puts(« Hello \n ») ;
```

- entier i non signé

```
printf(« L'entier est %d \n »,i) ;
```

- entier cours (short) i non signé

```
printf(« L'entier est %hu\n »,i) ;
```

- réel x

```
printf(« L'entier est %f \n »,x) ;
```

- réel x avec 4 chiffres avant la virgule et 2 après

```
printf(« L'entier est %4,2f \n »,x) ;
```

Lecture du clavier

- 1 caractère :

```
char car ;
scanf(«%c »,&car) ;
```

- 1 entier :

```
int i ;
scanf(«%d »,&i) ;
```

- plusieurs entier :

```
int i,j ;
scanf(«%d%d »,&i,&j) ;
```

- 1 réel :

```
float x ;
scanf(«%f »,&x) ;
```

C - Instruction conditionnelle

```
int a=5 ;
if ( a > 2 ){
        printf(« a > 2 \n ») ;
}
else{
        printf(« a <= 2 \n ») ;
}
```

D - Instruction d'aiguillage (switch)

```
int a ;
scanf(«%d »,&a) ;
switch(a){
        case 0 : printf(« ...0 … \n ») ;
        case 1 : printf(« ...1 … \n ») ;
                 break; => important sinon il fait le cas suivant
        default : printf(« Autre \n ») ;
 }
```

E - Instruction boucle (while)

=> s'exécute tant que la condition est vraie

```
int i=0 ;
while(i<5){
        puts(« Tapez un nombre >=5 ») ;
        scanf(«%d »,&i) ;
}
```

F - Le Do While

Pareil que le while, mais le code est exécuté une fois avant de faire le test.

```
int i =13 ;

do{
        puts(« Tapez un nombre  >=5 ») ;
        scanf(«%d »,&i) ;
}while(i<5) ;
```

G - Instruction FOR

```
        int i ;
        for(i=0;i<10;i++){
                printf(«%d »,i) ;
        }
        printf(« «\n ») ;
```

H - Instruction de débranchement

- break : quitte la boucle avant la fin
- continue : aller à la fin des instructions

I - Opérations logiques

Mathématiques : +, -, *, /, %, ++, --
Relationnelle : ==,!=, >, <, <=, >=
Logique : &&, ||, ! (ou exclusif)
Bits à bits : >>, <<, b, ^, |,
Affectation composé : += , -=, *=, /=, %=, <<=, >>=, ^=, |=

II - Les fonctions

```
type_de_retour nom_de_fonction(type arg1, type arg2){
        instruction ;
        instruction ;
}
```

Remarques

- On peut avoir aucune valeur de retour (type de retour est void)
- On peut avoir plusieurs valeurs de retour
- Fonction principale du programme :
  int main (int argc, char * argv[])

Porté des variables

- visible à l'intérieur de son bloc d'instruction
- pour une vairable hors de toute fonction elle doit être déclaré avant tout bloc d'instruction et peut être utilisé partout

Les fonctions imbriquées

```
int f() ;
int g() ;
int f(){
        g() ; // permet d'utiliser la fonction g sans l'avoir définit
}
int g(){ … }
```

Les fichier d'entête (.h)

- contient la déclaration des fonctions (.h)
- le fichier .c contient le code et le fichier d'entête à l'aide de l'instruction (# include « nom_du_fichier.h »)

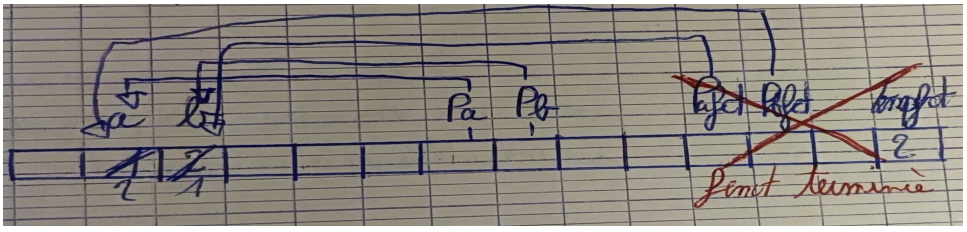
Exemple

```
# ifndef FICHIER_H
# define FICHIER_H
void fct1() ;
void fct2(int a, float b) ;
int fct3(float a) ;
#endif
```

III - Les pointeurs et la mémoire C

Le passage des arguments en C se fait par valeur. La valeur des variables d'entrée et copié dans une variable temporaire existant uniquement durant l'exécution de la fonction.

On peut échanger deux variables en utilisant les adresse des variables



Définition de pointeur

Le pointeur « pa » sur la variable « a » est l'adresse mémoire de la variable « a »

Déclaration

type_de_la_variable_pointé *nom_de_la_variable

Accès à l'adresse

&variable

Accès à la variable pointée

*pointeur

Exemple

```
int a = 10 ;
int *pa ;
pa=&a ;
*pa=11 ; // valeur de a est aussi changé
```

→ Comment retourner plusieurs valeur en C ?

En utilisant plusieur pointeurs en paramètre d'entiée

Exemple

```
int a ;
int b ;
ftc(&a,&b) ;
```

Le Scanf, explication

Le « & » devant les variables d'un scanf permet de le faire passer par adresse. Il permet ainsi de stocker le résultat dans la variable mis en argument.

Les pointeurs sur les fonctions

C'est un pointeur sur les instructions d'une fonction donnée.

Exemple

```
int ma_ftc(double *, int ){
        ...
}

int ( *pointeurSurFonction) (double*,int) ; // déclaration
pointeurSurFonction= ma_ftc ; // affectation (pas de &)
```

!!! int (*pf)() => pointeur sur une fonctions
int * f() => retour un pointeur

IV - Les tableaux

Définition

- Un tableau est une collection d'objet de même type
- Ce regroupement ce fait sous un même identifiant
- On accède au élément individuellement par un indice entier
- Un tableau peut contenir lui-même des tableaux comme un élèment

Exemple

{ 1, 22, 43, 78, 4 }
{ 'a', 'e', 43, 4 } => compris entre 0 et 255 c'est donc un caractère
{ { 1, 2 } , {3} }
{ « abcd », {'d','a'} }

Remarques

il existe 2 types de tableau en C :
- les **tableaux statiques** dont la dimension est connu au moment de la compilation
- les **tableaux dynamiques** dont la taille peut-être choisie à l'exécution

A - Les tableaux statiques

Déclaration

type mon_tab[dim] ;

La dimension « dim » ne peut pas être une variable. C'est un entier qui doit être fixé à la compilation.

Exemple

```
int tab[5] ;
float t[1] ;
```

Initialisation des tableaux

```
int tableau [5] = { 1, 5, 45, 3, 9} ;
int tableau [] = { 1, 5, 45, 3, 9} ;
int tableau [512] = { 0 } ;
```

Accès aux éléments d'un tableau

Si le tableau « tab » à été déclaré, on peut accéder à une case « i » en lecture et écriture : tab[i] ;

Remarques

- La première case est la case d'indice 0.
- La porté d'un tableau statique est la même que pour une variable classique

Exemple

Création d'un tableau contenant les chiffres de 0 à 9.

```
int tab[10] ;
for ( int i=0 ; i < 10 ; i++ ){
        tab[i]=i ;
}
```

!!! vérifier que l'on ne sort pas du tableau sinon il y aura des erreur difficile à détecter (buffer over flow)

B - Les tableaux et les pointeurs

Un tableau en C est un pointeur sur la première case du tableau.

On peut utiliser un tableau comme un pointeur:

```
tab[0]= *tab ;
*(tab+5)==tab[5]
```

```
void f ( int t [] ){
        /* ... */
}
```

```
void f ( int *t ) {
        /* ... */
}
```

Les tableaux dynamiques

!!! BILIOTHEQUE A UTILISE : #include <stdlib,h>

Déclaration

Un tableau dynamique est un pointeur sur une zone mémoire qui va être alloué au moment de l'exécution. Il se déclare comme un pointeur.

```
int *tab ;
```

Allocation de la mémoire

Une fois déclaré un tableau dynamique ne peut pas être utilisé,il réserve un espace mémoire.

```
tab= (int *)malloc( 5 * sizeof( int )) ;
tab= (int *)calloc( 5, sizeof(int)) ;  => initialise à zéro tout les blocs
```

Il a la même utilisation que les tableaux statiques

Libération de la mémoire

- Le tableau dynamique est persistant.
- **La mémoire reste réservé tant qu'elle n'est pas libéré et même si on a perdu la variable permettant d'accéder au tableau => pour chaque malloc / calloc il faut un free(mon_tab) ;**

Chaîne de caractères

- Pas de type String en C
- **Une chaîne de caractères est un tableau de caractères**
- Le dernier caractère d'une chaîne de caractère est nécessairement le caractère de fin de chaîne « \o » ;

Exemple

```
char *chaine1= « Ceci est une chaîne » ;
char chaine2[]= « Ceci est une autre chaîne » ;
char chaine3[]= {'C','e','c','i','','e','s','t','\o'} ;
```

V - Les tableaux 2D

Déclaration des tableaux 2D statiques

```
int matrice [2][3]={ {1, 2, 3}, {4, 5, 6}} ;
int matrice [2][3]={ 1, 2, 3, 4, 5, 6} ;
```

Accés

```
tab[i][j]
```

Déclaration des tableaux 2D dynamiques

```
int **tab ;
```

Allocation de mémoire pour les tableau 2D

→ Solution peut coûteuse peut de malloc à faire

```
int nbLigne=10 ;
int nbColonne=2 ;

int **tab ;
int *tabTmp ;

tab=(int **)malloc( nbLigne * sizeof( int *)) ;
tabTmp=(int *)malloc( nbLigne * nbColonne* sizeof(int)) ;

for(int i=0; i < nbLigne ; i++){
        tab[i]=tabTmp+i * nbColonne
}

…
free(tab[0]) ;
free(tab) ;
```