Langage C — notes de cours

K. El Omari
IUT GTE, UPPA

1 Présentation générale

1.1 Introduction

La programmation par le langage C (ainsi que par d'autres langages dit compilés) est basée sur :

1. la rédaction du texte du programme (codage) en respectant la syntaxe précise du langage, à l'aide d'un éditeur de texte ASCII pur (sans mise en page) ;
2. la compilation : transformation du code C en langage machine par un compilateur C ;
3. exécution du fichier binaire obtenu.

1.2 Premier programme

```
#include <stdio.h>
main()
{
    /* Déclaration de variables */
    int a,b,c;

    printf(" Bonjour, ça marche !\n");
    a = 1;      /* initialisation */
    b = 2;
    c = a + b;  /* affectation */
    printf(" Résultat = %d \n ",c);
    printf(" Au revoir !\n");
}
```

La fonction `main()`est la fonction principale du programme. Elle est la seule à être exécutée au lancement du programme ; elle fait appel éventuellement à d'autres fonctions. Dans l'exemple ci-dessus, elle appelle la fonction printf().

La fonction `printf()`est une fonction standard définie dans le fichier `stdio.h`inclus en début de programme. Elle affiche une chaîne de caractères à l'écran ainsi que certains types de variables prédéfinis.

Le corps des fonctions est toujours délimité par des {...}, et chaque instruction est terminée par un " ; ".

Les commentaires sont mis entre `/* ... */`, ainsi le compilateur les ignore.

2 Déclaration de variables

2.1 Noms des variables

En langage C, les variables doivent être déclarées en début de programme et leurs types doivent être explicités.

Les noms des variables peuvent contenir des lettres et des chiffres ainsi que le caractère de soulignement "_".

Le C distingue les minuscules des MAJUSCULES (ex : `a` est différente de `A`). Les noms de variables ne doivent pas commencer par un chiffre (ex : `1b` n'est pas valable).

Le compilateur peut considérer des noms de variables qui contiennent jusqu'à 31 caractères voire plus. Il est important alors d'utiliser des noms significatifs pour faciliter la relecture et la compréhension du programme.

Certains mots-clés du langage lui sont réservés et ne doivent pas être utilisés pour nommer les variables (ex : `int, float, if` ...).

2.2 Les types prédéfinis

Déclaration	Type	Exemples
`int`	entier	`1, -5`
`float`	réel à virgule flottante simple précision	`2.365, -0.25e-2`
`double`	réel à virgule flottante double précision	`10.2, .25E+5`
`char`	caractère	`'a' '1' ' ' '\n'`

2.3 Déclaration

La déclaration se fait à l'aide du mot-clé correspondant au type souhaité, suivi d'un nom de variable respectant les règles citées plus haut. ex :

```
float pourcentage_de_pommes_pourries = 10.36;
```

Cette variable a été initialisée dès sa déclaration.

2.4 Déclaration de constantes

la déclaration et l'initialisation d'une variable peuvent commencer par le qualificatif `const` qui indique que la valeur de cette variable ne pourra plus être modifiée.

```
const int N = 30;
```

Il est également possible de déclarer des constantes à l'aide de la commande du préprocesseur `#define` :

```
#define NMAX 10
```

Avant la compilation, le préprocesseur (une partie du compilateur) remplacera toutes les occurrences de `NMAX` par le chiffre `10`.

3 Les opérateurs

3.1 Les opérateurs arithmétiques

`x + y`	addition
`x - y`	soustraction
`x * y`	multiplication
`x / y`	division
`x % y`	reste de la division entière (modulo)

3.2 Les opérateurs de comparaison et les opérateurs logiques

Les opérateurs de comparaison sont : `> >= < <=`

Les opérateurs d'égalité : `== !=` .

Attention, `==` est différent de l'opérateur d'affectation `=`.

Les opérateurs logiques `&&` et || signifient "et" et "ou".

Ces opérateurs donnent un résultat "Vrai" ou "Faux" selon l'expression où ils figurent. En réalité ils retournent la valeur 0 pour "Faux" et 1 pour "Vrai".

En général, toute valeur non nulle est évaluée comme "Vrai".

L'opérateur de négation `!` inverse les valeurs logiques des expression.

```
int a,b;
a = 2;
b = 3;
a > b;           /* vaut 0 */
a < b;           /* vaut 1 */
!b;              /* vaut 0 */
a == b;          /* vaut 0 */
b != a;          /* vaut 1 */
a > b && a < b; /* vaut 0 */
a > b || a < b; /* vaut 1 */
```

3.3 Les opérateurs d'incrémentation et de décrémentation

```
int i = 0;
int a;
i++;          /* équivalent à i = i + 1 */
i--;          /* équivalent à i = i - 1 */
```

3.4 Les opérateurs d'affectation

```
a = i;        /* a prend la valeur de i *
a += i;       /* équivalent à a = a + i */
a -= i;       /* équivalent à a = a - i */
a *= i;       /* équivalent à a = a * i */
a /= i;       /* équivalent à a = a / i */
```

4 Les structures de contrôle

4.1 Instruction `if` (si)

```
if(a > b) i = 1;
```

```
if(a != b)
{
    i = 1;
    i++;
}
```

4.2 Instruction `if-else` (si-sinon)

```
if(a > b) i = 1;
else i++;
```

```
if(a != b)
{
    i = 1;
    i++;
    /* ... */
}
else
{ /* faire autre chose ... */ }
```

4.3 Instruction `switch`

```
char x;
int i;
switch(x){
    case 'a':
        i++;
        break;
    case 'b':
        i+=2;
        break;
    case 'c': case 'd':
        i+=3;
        break;
    default:
        i*=2;
        j+=3;
        break;
}
```

ou

```
char x;
int i;
if(x == 'a')
    i++;
else if(x == 'b')
    i+=2;
else if(x == 'c' || x == 'd')
    i+=3;
else
{
    i*=2;
    j+=3;
}
```

La même chose peut être réalisée par une combinaison d'instructions `if-else`.

Après `if` ou `else`, les accolades sont indispensables dans le cas de plusieurs instructions.

4.4 Boucle `for` : (*pour*)

Elle s'écrit en général sous la forme suivante :

`for(<< initialisation >> ; << condition de continuation >> ; << incrémentation >>)`

Les trois expressions sont facultatives, exemple : `for( ; ; )` boucle infinie.

```
int i;
for(i = 0; i<4 ; i++)
{
    int j = i*i;
    printf("i = %d, au carré = %d \n",i,j);
}
```

affiche :

```
i = 0, au carré = 0
i = 1, au carré = 1
i = 2, au carré = 4
i = 3, au carré = 9
```

4.5 Boucle `while` : (*tant que*)

La boucle précédente peut également s'écrire à l'aide de `while` comme ceci :

```
int i = 0;
while( i< 4)
{
    int j = i*i;
    printf("i = %d, au carré = %d \n",i,j);
    i++;
}
```

4.6 Boucle `do-while` : (*exécuter - tant que*)

```
int i = 0;
do
{
    int j = i*i;
    printf("i = %d, au carré = %d \n",i,j);
    i++;
}while(i<4);
```

Ici, la condition est évaluée à la fin de l'exécution de la première itération, contrairement à l'instruction `while` où la première itération n'est exécutée que si la condition est préalablement vraie.

4.7 Instructions `break` et `continue`

`break` provoque la sortie de la boucle à partir de l'emplacement où elle se trouve . `continue` renvoie au début de la boucle sans exécuter les instructions qui la suivent.

5 Les fonctions

En langage C, les fonctions correspondent à des sous-programmes. Elle englobent des parties de code aux quelles ont peut faire appel et éviter ainsi les redondances inutiles des mêmes instructions.

Chaque fonction doit être déclarée avant d'être utilisée (prototype).

Une fonction peut renvoyer une valeur (exemple un entier) ou ne rien renvoyer (`void`).

```c
#include <math.h>

double norm(double a, double b, double c)
{
    double n = sqrt(a*a + b*b + c*c);
    return n;
}
```

Elle peut également ne pas prendre de paramètre.

```c
#define N 10
void affiche_produit(int,int); /* prototypes */
void trace_ligne(void);

main()
{
    int i,j;
    trace_ligne();
    printf("   X ||");
    for(i=1; i<=N; i++) printf("%4d ||",i);
    trace_ligne();
    for(i=1; i<=N; i++)
    {
        printf("%4d ||",i);
        for(j=1; j<=N;j++)
        {
            affiche_produit(i,j);
        }
        trace_ligne();
    }
}

void affiche_produit(int a,int b)
{
  printf("%4d ||",a*b);
}

void trace_ligne(void)
{
    int i;
    printf("\n");
    for(i = 0;i<(N+1)*7;i++) printf("=");
    printf("\n");
}
```


6 Tableaux

Un tableau est un espace réservé en mémoire pour stocker une série d'éléments du même type (ex : `int`). Déclaration d'un tableau :

```c
#define N 100
/* ... */
int Tab[N];
```

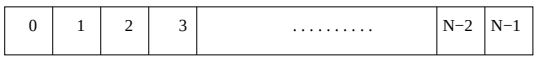

Tableau de taille N

Les emplacements d'un tableau de taille N sont numérotés de 0 à N-1 en langage C[1]. L'accès aux différents emplacements s'effectue grâce à leurs numéros. L'accès à l'emplacement `i` du tableau `Tab` est réalisé par `Tab[i]`.

```
#define N 100
main()
{
    int Tab[N],i;
    for(i=0; i<N;i++)
    {
        Tab[i] = i*i;
        printf(" Tab[%d] = %d \n",i,Tab[i]);
    }
}
```

Il est à noter que `Tab` représente un pointeur vers l'adresse dans la mémoire du premier élément du tableau. Il représente l'adresse de `Tab[0]`.

7 Les entrées-sorties

Les entrées-sorties en langage C utilisent la notion de fichier. Ainsi, la sortie standard (l'écran) est considérée comme un fichier où le programme peut écrire (afficher) de la même façon que sur un fichier sur disque.

7.1 Les entrées-sorties standard

Les entrées-sorties standard sont le clavier et l'écran.

Pour la lecture et l'écriture d'un caractère on peut utiliser les fonctions `getchar()` et `putchar()` :

```
main()
{
  char c;
  while((c = getchar()) != 'o')
    putchar(c);
}
```

Pour des entrées-sorties avec mise en forme on peut utiliser `scanf` et `printf` :

[1] de 1 à N en Fortran

```
main()
{
  char c, C[512];
  int i;
  float f;
  double x;
  printf(" Donner un caractère : ");
  scanf("%c",&c); printf("  %c\n",c);
  printf(" Donner une chaîne de caractères sans espaces: ");
  scanf("%s",C);  printf("  %s\n",C);
  printf(" Donner un entier : ");
  scanf("%d",&i); printf("  %d\n",i);
  printf(" Donner un réel simple : ");
  scanf("%f",&f); printf("  %f\n",f);
  printf(" Donner un réel double : ");
  scanf("%lf",&x); printf("  %lf\n",x);
}
```

Pour acquérir des caractères il est préférable d'utiliser les fonctions spécialisées comme `getchar()`.

7.2 Les entrées-sorties fichiers

La fonction `fopen()` peut ouvrir un fichier et renvoyer un pointeur vers celui-ci afin de le rendre disponible pour des opérations de lecture/écriture.

```
#include <stdio.h>
main()
{
  char C[256] = "Bonjour!", A[256];
  int i = 2, j;
  float f = 3.5e-5, g;
  FILE *fp;                              /* pointeur sur fichier */
  fp = fopen("nom_fichier.out", "w");    /* ouverture en écriture */
  fprintf(fp,"%s \n %d \n %f \n",C,i,f);
  fclose(fp);                            /* fermeture du fichier */

  fp = fopen("nom_fichier.out", "r");    /* ouverture en lecture */
  fscanf(fp,"%s \n %d \n %f \n",A,&j,&g);
  fclose(fp);                            /* fermeture du fichier */
}
```