Table des matières

Chapitre 1: Présentation du langage C Eléments de Base

1. Introduction

Le C a été développé conjointement au système d'exploitation UNIX, dans les Laboratoires BELL, par Brian W Kernigham et Dennis M Ritchie, qui ont défini en 78, dans "The C Language", les règles de base de ce langage. Le but principal était de combiner une approche structurée (et donc une programmation facile) avec des possibilités proches de celles de l'assembleur (donc une efficacité maximale en exécution, quitte à passer plus de temps de programmation), tout en restant standard (c'est à dire pouvoir être implanté sur n'importe quelle machine). Puis ce langage a été normalisé en 88 (norme ANSI), cette norme apportant un nombre non négligeable de modifications au langage.
Le C est un langage compilé, c'est à dire qu'il faut :
entrer un texte (le *programme source*) dans l'ordinateur (à l'aide d'un programme appelé EDITEUR),
le traduire en langage machine (c'est à dire en codes binaires compréhensibles par l'ordinateur) : c'est la compilation et, si plusieurs modules ont été compilés séparément, l'édition de liens (LINK ou BIND),
l'exécuter.
Contrairement à un langage interprétés (Basic autrefois, javascript...), l'exécution sera beaucoup plus rapide puisqu'il n'y a plus de traduction à effectuer, mais la phase de mise au point sera plus complexe.

6.2 Premier programme

Regardons ce petit programme :

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
 {

  printf("Salam\n");
 }
```

On trouve dans ce programme :

- des directives du pré processeur (commençant par #)

#include : inclure les fichiers qu'on appelle librairies définissant les fonctions standards d'entrées/sorties (en anglais STanDard In/Out), qui feront le lien entre le programme et la console (clavier/écran). Dans cet exemple il s'agit de printf.
- une **entête de fonction.** Dans ce cas on ne possède qu'une seule fonction, la fonction principale **main**. Cette ligne est obligatoire en C, elle définit le "point d'entrée" du programme, c'est à dire l'endroit où débutera l'exécution du programme.
- un **bloc d'instructions**, délimité par des accolades **{ }**, et comportant : des déclarations de variables, sous la forme : type listevariables, et des instructions, toutes terminées par un **;** Une instruction est un ordre élémentaire que l'on donne à la machine, qui manipulera les données (variables) du programme.

2. Déclaration de variables

Une variable est un case mémoire de l'ordinateur, que l'on se réserve pour notre programme. On définit le nom que l'on choisit pour chaque variable, ainsi que son type. Les trois types scalaires de base du C sont l'entier (int), le réel (float) et le caractère (char).
Les noms donnés aux variables sont appelés des *identificateurs*. Ils doivent commencer par une lettre (majuscule ou minuscule non accentuée) mais peuvent contenir outre des lettres, des chiffres et le caractère souligné _ dans le reste du nom. En langage C, les caractères majuscules et minuscules ne sont pas équivalents, ainsi "batna" et "BATNA" sont deux identificateurs différents. On ne peut jamais utiliser de variable sans l'avoir déclarée auparavant. Une faute de frappe devrait donc être facilement détectée, à condition d'avoir choisi des noms de variables suffisamment différents (et de plus d'une lettre).
On déclare une variable comme suit:

<type> <identificateur>;

S'il y a plusieurs variables de même type, elles peuvent être déclarées sur la même ligne, séparées par des virgules comme on le voit dans cette déclaration:

int i, j, m1, m2, m3, m4, m5;

7.2 Les types simples de C

2.1.1 Les entiers

Le type entier associé au mot-clé **int**, signifie qu'une variable de ce type ne peut contenir que des nombres entiers.
Il existe d'autres types numériques. Ainsi les types **char** , **long int** et **short int**, que l'on peut abréger en **long** ou **short**, représentent des entiers tout comme **int** mais avec des valeurs minimales et maximales qui peuvent être différentes.
En principe un **int** est un entier représenté dans la taille la plus naturelle pour le microprocesseur utilisé, c'est-à-dire 32 bits sur un microprocesseur 32 bits et 64 bits sur

un microprocesseur 64 bits mais ceci dépend aussi du système d'exploitation voire du compilateur (on peut avoir des **int** de 32 bits sur une machine 64 bits). Dans tous les cas un **short** fait au moins 16 bits, un **long** au moins 32.

Exemple

```
int nombre;
long X1;X2;
```

Pour éviter qu'une variable ne prenne une valeur indéterminée jusqu'à sa première affectation, on peut spécifier sa valeur au moment de sa déclaration. Cette construction s'appelle une *initialisation*:

```
int nombre = 5;
```

On peut également appliquer les qualificatifs **signed** et **unsigned** aux types **char**, **int**, **short** et **long**. Ces qualificatifs indiquent si le type est signé ou non signé ce qui change l'arithmétique et les valeurs maximales autorisées pour une variable du type considéré. Ainsi, une variable de type **signed char** peut contenir desvaleurs allant de -128 à +127 alors qu'une variable de type **unsigned char** peut contenir des valeurs allant de 0 à 255. Par défaut les types sont signés et le mot-clé **signed** est donc très peu utilisé en pratique, le mot-clé **unsigned** est par contre très fréquent.

2.1.1 Les réels

Pour manipuler des nombres réels, non entiers, on utilise les types **float**, qui signifie nombre à virgule flottante ou plus simplement flottant, et **double**, qui signifie flottant à double précision. Un nombre à virgule flottante est une quantité codée sur 32 bits, comprenant au moins six chiffres significatifs, et comprise entre 10-38 et 10+38 environ. Une variable de type **float** ou **double** peut ainsi contenir par exemple des valeurs très proches de (mais jamais exactement égales à) π ou $\sqrt{2}$. Un flottant peut posséder des chiffres après la virgule, la virgule étant représentée par un point conformément à l'usage des pays anglo-saxons.
Exemple:

```
 #include <stdio.h>
void main ()
{
float nombre_reel;
nombre_reel = 5.679 + 6.120;
}
```

2.1.1 Les constantes

Les constantes entières tapées dans un programme telles que 125 sont par défaut de type **int**. On peut toutefois demander qu'elles soient considérées de type **long** en ajoutant un **l**

ou **L** à la fin: 125L. Une constante entière trop grande pour tenir dans **int** est considérée comme un **long** même si elle n'a pas de **l** ou **L** à la fin.
Pour désigner une constante non signée on utilise le suffixe **u** ou **U** que l'on peut combiner avec le suffixe **l** ou **L** pour désigner une constante de type **unsigned long**: 1245UL.

Les constantes contenant une virgule (125.7) ou un exposant (1e2) sont considérées de type double par défaut. On peut cependant demander que de telles constantes soient considérées soit comme un **float** en ajoutant un suffixe **f** ou **F**, soit comme un **long** double en ajoutant un **l** ou **L**.

Une constante de type caractère (**char**) s'écrit sous forme d'un caractère entre apostrophes: 'a'. La valeur numérique d'une telle constante est la valeur du caractère dans le jeu de caractères de la machine (le plus souvent le code ASCII). Ainsi la valeur numérique de la constante caractère '0' est 48. Dans les calculs, les caractères sont traités exactement comme des entiers même si on s'en sert le plus souvent pour les comparer à d'autres caractères.

Remarque
Dans de nombreux programmes, il est agréable et plus clair d'utiliser une constante sous son nom habituel (pi, g...). Pour cela on utilise la directive **#define**, généralement à l'extérieur du bloc de la fonction main:

```
#include <stdio.h>
#define pi 3.1415626
#define g 9.81
void main ()
{
double rayon, diametre;
rayon = 15.0;
diametre = 2*pi*rayon;
}
```

Dans le programme, les constantes s'utilisent comme les variables, à part le fait qu'elles ne peuvent évidemment pas apparaître dans le membre de gauche d'une affectation.

Constantes énumérées. Une énumération est une suite de valeurs entières constantes auxquelles on donne des noms symboliques. Par exemple:

```
enum fruits { pomme, poire, banane };
```

Le premier nom d'une énumération vaut zéro, le suivant un, et ainsi de suite, à moins que l'on précise des valeurs explicites. Si l'on ne donne que certaines valeurs, les suivantes se déduisent par incréments successifs de un:

enum mois { jan=1,fev,mar,avr,mai,jun,jul,aou,sep,oct,nov,dec};

Dans cet exemple fev vaut 2, mar vaut 3 et ainsi de suite.

Les noms définis à l'intérieur d'une énumération doivent être distincts.

3. Commentaires

Il est indispensable de mettre des commentaires dans les programmes car il est très difficile de retrouver ce que fait un programme quand on n'a que la liste d'instructions. En C, il est possible de mettre des textes entre les séquences /* et */ ou après la séquence //, pour documenter les programmes directement dans les sources. Ces commentaires ne font pas partie des instructions et sont totalement ignorés par le compilateur.

```
/* Commentaire
qui s'étend sur
plusieurs lignes */
// Commentaire sur une ligne
```

4. Affichage des variables

Pour afficher le contenu de variables on utilise l'instruction **printf**. Jusqu'à présent nous avons utilisé cette fonction avec comme seul paramètre une simple chaîne de caractères. Mais la fonction **printf** permet également d'afficher le contenu de variables. Pour cela, la fonction a besoin de savoir quel est le type de ces variables (entier, flottant, simple caractère, etc...) et quel format d'affichage utiliser (combien afficher de chiffres après la virgule, utiliser la notation décimale habituelle ou la notation scientifique en puissances de 10, compléter les nombres trop petits par des espaces ou des zéros à gauche, etc...) Ces deux informations, type des variables à afficher et format d'affichage sont indiquées au moyen de séquences de caractères spéciales, appelées *spécificateurs de format*, insérées dans la chaîne de caractères passée comme premier paramètre à la fonction **printf**. Cette chaîne, dite chaîne de format (*format string*), est suivie du nom des variables à afficher, séparées par des virgules.
Ainsi l'instruction

```
printf("%d",A);
```

affiche le contenu de la variable A sur l'écran.
Le spécificateur de format %d, placée dans la chaîne de format indique que la variable A est de type **int**. Le tableau suivant résume quelques uns des spécificateurs de format les plus courants reconnus par la fonction **printf**.

Format Type de variable correspondant

%c	Caractère de type **char**
%d ou %i	Entier de type **int**
%ld	Entier de type **long**

%u	Entier non signé de type **unsigned int**
%f	Nombre à virgule flottante de type **float**
%lf	Nombre à virgule flottante de type **double**
%e ou %E	Nombre à virgule flottante affiché en notation exponentielle

Spécificateurs de format de la fonction printf

Il est également possible d'afficher le contenu de plusieurs variables avec la même instruction **printf** en insérant pour chacune un spécificateur dans la chaîne de format et en séparant les différentes variables par des virgules dans la chaîne de paramètres. Par ailleurs il est également possible d'insérer du texte à afficher autour des spécificateurs de format.
Ainsi, si les variables réelles long et larg contiennent respectivement les valeurs 31.23 et 5.34, l'instruction

```
printf(‘‘longueur = %f largeur = %f’’, long, larg);
```

Contrôle de la longueur de champ d'affichage

Pour améliorer la lisibilité de nombres de longueur variable affichés en colonne, la fonction **printf** permet d'ajouter aux spécificateurs de format un nombre indiquant la largeur minimale de champ d'affichage, les nombres sont alors affichés alignés à droite dans des champs de longueur correspondante. Ainsi si l’on place un nombre au milieu du spécificateur **%d** dans la chaîne de format de l'instruction **printf**, comme suit:

```
printf(‘‘%6d’’, A);
```

la variable A sera affichée dans un champ de 6 caractères. Si la valeur contenue A comporte, par exemple, 4 chiffres alors deux espaces seront insérées avant d'afficher A. Si, par contre, A comporte 6 chiffres ou plus alors aucune espace supplémentaire n'est inséré.

Exemple

```
#include <stdio.h>
void main ()
{
int nbr;
nbr = 12; printf("%4d\n", nbr);
nbr = 1234; printf("%4d\n", nbr);
nbr = 31645; printf("%4d\n", nbr);
}
```

Ecran

```
  12
```

1234
31645

La largeur minimale de champ s'applique aussi aux variables de type float ou double. De plus, pour celles-ci, il est également possible d'indiquer une précision, c'est-à-dire combien de chiffres après la virgule doivent être affichés (sans indication de précision, 6 décimales sont affichées). La précision est indiquée à la suite de la largeur de champ, séparée par un point.
Ainsi l'instruction :

```
printf(‘‘%12.3f’’,x)
```

affiche la valeur de la variable flottante x avec 3 décimales au plus dans un champ de 12 caractères minimum complété à gauche par des espaces si besoin. Si x=1125.43567

Ecran
1125.435

5. Lecture au clavier

Il est possible de demander au programme de s'arrêter à un endroit de son exécution et d'attendre qu'on lui donne une valeur au clavier. L'instruction qui fait cela, nommée **scanf**, a le même effet qu'une affectation.
Tout se passe comme si l'on pouvait écrire: `A =` "valeur tapée au clavier". **scanf** est un peu le symétrique de **printf** et s'utilise de façon assez similaire. Ainsi, le premier argument passé à **scanf** doit être une chaîne de format indiquant le type des paramètres suivants. Cette chaîne est suivie d'une liste de variables dont le contenu sera affecté d'après ce que l'utilisateur du programme aura tapé au clavier.

```
#include <stdio.h>
void main ()
{
int A;
printf("Donnez un nombre: ");
scanf("%d", &A);
}
```

Lors de l'exécution du programme ci-dessus, l'ordinateur affichera "Donnez un nombre:" puis, lorsqu'il arrivera à l'instruction `scanf(‘‘%d’’, &A)`, il attendra que l'utilisateur tape une valeur entière au clavier. Cette valeur sera déposée dans la variable `A`.
On peut mettre plusieurs variables dans l'instruction `scanf`. Lorsqu'on les tape au clavier, il faut les séparer par des blancs ou des retours de ligne.

```
#include <stdio.h>
void main ()
{
int A; float X1;
```

```
printf("Donnez un nombre entier et un nombre réel: ");
scanf("%d %f", &A, &X1);
}
```

Remarque
Attention de ne pas oublier le signe **&** devant le nom des variables numériques passées en argument.

6. Calcul en C et Expressions arithmétiques

Les opérateurs arithmétiques du langage C sont les suivants: +, -, *****, /, **%**, représentant respectivement l'addition, la soustraction (ou le changement de signe: -x), la multiplication, la division et finalement le reste de la division entière (modulo).
Les quatre opérateurs de base opèrent sur des entiers aussi bien que des réels. Si les opérandes sont tous deux entiers alors l'opération a lieu en arithmétique entière. Si l'un des deux opérandes au moins est réel, alors l'opération a lieu en arithmétique réelle. Il faut dès lors prendre garde au fait qu'un nombre écrit dans un programme est par défaut considéré entier s'il n'a pas de point décimal. De plus, si l'on affecte à une variable entière un nombre réel (ou une expression, telle une division de réels, dont le résultat est un réel), alors le compilateur effectue automatiquement l'arrondissement. Par exemple si i est une variable entière, l'expression i=100.0/45.0; vaut 2. Voir tableau suivant:

Instruction	**Signification**
i = 10 / 3;	i de type **int** reçoit la valeur 3, résultat de la division entière
i = 10.0 / 3;	i de type **int** reçoit la valeur 3 par arrondi automatique de la division réelle
x = 10 / 3;	x de type **float** reçoit la valeur 3, résultat de la division entière
x = 10/3 * 3;	x reçoit la valeur 9.0, le calcul est fait en entiers puis transformé en réel.
x = 10.0/3 * 3;	x reçoit la valeur 10.0 la division et la multiplication étant effectués en réels.
x=i/j;	Si les entiers i et j valent 7 et 2, x de type **flo at** reçoit la valeur 3, résultat de la division entière
x=(float)i/j;	Si les entiers i et j valent 7 et 2, x de type **float** reçoit la valeur 3.5, résultat de la division réelle de i, préalablement converti explicitement en réel, par j automatiquement converti en réel

Arithmétique entière et réelle

Exemple

```
#include <stdio.h>
void main ()
{
int i, j, m1, m2, m3, m4, m5;
printf("Donnez deux nombres: ");
scanf("%d %d", &i, &j);
m1 = i + j;
m2 = i - j;
m3 = i * j;
m4 = i / j;
m5 = i % j;
printf("i + j = %8d\n", m1);
printf("i - j = %8d\n", m2);
printf("i * j = %8d\n", m3);
printf("%1d = %1d * %1d + %1d\n", i, j, m4, m5);
}
```

Ce programme illustre l'emploi des opérateurs sur les nombres entiers.
Dans les expressions mathématiques, les parenthèses obéissent aux mêmes règles qu'en algèbre. L'instruction `y = 2 * z1 + 5 * z2;` a la même signification que
`y = (z1 * 2) + (z2 * 5);.`
Comme nous l'avons déjà indiqué, une variable peut recevoir plusieurs valeurs de suite au cours de l'exécution du même programme. On peut même changer sa valeur à partir d'un calcul qui la contient:

```
i = 7; i = i + 5;
```

Cette dernière affectation (attention il s'agit bien d'une affectation et non d'une égalité mathématique) modifie la première valeur de i et réécrit 12 par-dessus.

6.1 Constructions abrégées

Le C offre un certain nombre de raccourcis d'écriture pour des instructions fréquentes. Le tableau suivant résume quelques-unes parmi les plus utilisées. Dans ce tableau `a`, `b` et `c` sont des variables numériques entières ou non:

Construction abrégée	Construction equivalente
a++;	a = a+1;
++a;	a = a+1;
a--;	a = a-1;
--a;	a = a-1;
a += b;	a = a+b;
a -= b;	a = a-b;
a *= b;	a = a*b;
a /= b;	a = a/b;

Constructions abrégées

Ces deux notations, préfixée et postfixée, incrémentent (resp. décrémentent) toutes deux la variable considérée mais présentent une différence fondamentale: l'expression ++a incrémente a *avant* de prendre sa valeur alors que a++ incrémente a *après* avoir pris sa valeur. Donc si a vaut 5, l'expression b = a++; met la valeur 5 dans b et incrémente ensuite a qui passe à 6 alors que l'expression b = ++a; incrémente d'abord a qui passe donc à 6 et met ensuite cette valeur dans b qui vaut alors 6.

Exemple

```
long  nb =2;
nb + = 4;      /* nb vaut 6  */
nb - = 3;      /* nb vaut 3  */
nb *= 5;       /* nb vaut 15  */
nb /= 3;       /* nb vaut 5  */
nb % =3;       /* nb vaut 2 car 5=1*3+2  */
```

6.2 Fonctions Mathématiques

Pour pouvoir utiliser les fonctions mathématiques, il est indispensable de mettre la directive de préprocesseur suivante en haut du programme:

```
#include <math.h>
```

Les fonctions mathématiques suivantes sont disponibles en C (parmi d'autres), elles sont déclarées dans le fichier `math.h`:

syntaxe	**fonction**
`sin(x)`	sinus en radians
`cos(x)`	cosinus en radians
`arctan(x)`	arc tangente en radians
`pow(x,y)`	`x` élevée à la puissance `y`
`sqrt(x)`	racine carrée
`abs(x)`	valeur absolue
`log(x)`	logarithme naturel
ceil(x)	renvoie le plus petit entier supérieur ou égal à l'argument réel x;
floor(x)	renvoie le plus grand entier inférieur ou égal à l'argument réel x;

Fonctions mathématiques courantes

On peut donc créer une expression du genre:

```
y = cos(sqrt(y+5.0)) + abs((arctan(z));

double R=0, nb=52.71;
R=ceil(nb);  /* R vaudra 53*/
```

```
double R=0, nb=2;
R= pow(nb,3);  /* R vaudra 2^3 */
```

7. Exercices d'applications

Exercice 1. Ecrire un programme qui permet de calculer et d'afficher le reste de la division d'un nombre par un autre.

Exercice 2. Ecrire un programme pour qu'il lise les 3 notes au clavier et qui calcule la moyenne exacte des notes 9,5 8,5 et 8, la moyenne arrondie à la note la plus proche et la note arrondie à 0,5 près.

Solutions

Solutions exercice 1:

```
#include <stdio.h>
void main ()
{
int nb1,nb2, quotient, reste;

/* lecture des données */
printf("donnez deux nombres entiers positifs: \n");
scanf("%d %d",&nb1,&nb2);

/* calcul */
quotient = nb1/nb2;
reste = nb1%nb2;

/* affichage */
printf("Le quotient de la division %d/%d=%d\n",nb1,nb2,quotient);
printf("Le reste de la division entière %d%%d=%d\n",nb1,nb2,reste);
}
```

Solutions exercice 2:

```
#include <stdio.h>
#include <math.h>
void main ()
{
float nt1,nt2,nt3,moy, moyent, moyreel;

/* lecture des données */
printf("Entrez 3 notes: \n");
scanf("%f %f %f",&nt1,&nt2,&nt3);

/* calcul */
moy = (nt1+nt2+nt3)/3.0;
```

```
moyent = floor(moy+0.5);
moyreel = (floor(moy*2+0.5))/2.0;

/* affichage */
printf("Moyenne : %f\n",moy);
printf("Moyenne arrondie à l'entier le plus proche :%d\n",moyent);
printf("Moyenne arrondie à 0.5 : %.2f\n",moyreel);
}
```