CHAPITRE 2

GÉNÉRALITÉS SUR LE LANGAGE C

Sommaire

Dans ce chapitre, nous vous proposons une première approche d'un programme en langage C, basée sur deux exemples commentés. Vous y découvrirez comment s'expriment les instructions de base (déclaration, affectation, lecture et écriture), ainsi que deux des structures fondamentales (boucle avec compteur, choix). Nous dégagerons ensuite quelques règles générales concernant l'écriture d'un programme. Enfin, nous vous montrerons comment s'organise le développement d'un programme en vous rappelant ce que sont l'édition, la compilation, l'édition de liens et l'exécution.

2.1 Un exemple de programme en langage C

Voici un exemple de programme en langage C, accompagné d'un exemple d'exécution. Avant d'en lire les explications qui suivent, essayez d'en percevoir plus ou moins le fonction-

nement.

Programme C :

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5
main()
{ int i ;
float x ;
float racx ;
printf ("Bonjour\n") ;
printf ("Je vais vous calculer %d racines carrées\n", NFOIS) ;
for (i=0 ; i<NFOIS ; i++)
{ printf ("Donnez un nombre : ") ;
scanf ("%f", &x) ;
if (x < 0.0)
printf ("Le nombre %f ne possède pas de racine carrée\n", x) ;
else
{ racx = sqrt (x) ;
printf ("Le nombre %f a pour racine carrée : %f\n", x, racx) ;
}
}
printf ("Travail terminé - Au revoir") ;
}
```

Après exécution

```
Bonjour
Je vais vous calculer 5 racines carrées
Donnez un nombre : 4
Le nombre 4.000000 a pour racine carrée : 2.000000
Donnez un nombre : 2
Le nombre 2.000000 a pour racine carrée : 1.414214
Donnez un nombre : -3
Le nombre -3.000000 ne possède pas de racine carrée
Donnez un nombre : 5.8
Le nombre 5.800000 a pour racine carrée : 2.408319
Donnez un nombre : 12.58
Le nombre 12.580000 a pour racine carrée : 3.546829
Travail terminé - Au revoir
```

2.2 Structure d'un programme en langage C

2.2.1 Les directives à destination du préprocesseur

Les trois premières lignes de notre programme :

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5
```

sont appelées des **directives**. Ces directives seront prises en compte avant la traduction (compilation) du programme. Contrairement au reste du programme, elles doivent être écrites à raison d'une par ligne et doivent obligatoirement commencer en début de ligne.
Les deux premières directives

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

demandent en fait d'introduire (avant compilation) des instrucions (en langage C) situées dans les fichiers stdio.h et math.h .
il est nécessaire d'incorporer de tels fichiers, nommés « fichiers en-têtes », qui contiennent des déclarations appropriées concernant cette fonction : stdio.h pour printf et scanf , math.h pour sqrt .
La troisième directive

```
#define NFOIS 5
```

demande simplement de remplacer systématiquement, dans toute la suite du programme, le symbole NFOIS par 5 . Autrement dit, le programme qui sera réellement compilé comportera ces instrutions :

```
printf ("Je vais vous calculer %d racines carrées\n", 5) ;
for (i=0 ; i<5 ; i++)
```

2.2.2 Programme principal

La ligne

```
main()
```

se nomme un « en-tête ». Elle précise que ce qui sera décrit à sa suite est en fait le programme principal (main).
Le programme (principal) proprement dit vient à la suite de cet en-tête. Il est délimité par les accolades « { » et « } ». On dit que les instructions situées entre ces accolades forment un « bloc ». Ainsi peut-on dire que la fonction main est constituée d'un en-tête et d'un bloc.

2.2.3 Déclaration

Les trois instructions :

```
int i ;
float x ;
float racx ;
```

sont des **déclarations**.
La première déclaration *int i* précise que la variable nommée i est de type int , c'est-à-dire qu'elle est destinée à contenir des nombres entiers (relatifs). Nous verrons qu'en C il existe plusieurs types d'entiers.
Les deux autres déclarations *float x* et *float racx* précisent que les variables x et racx sont de type float , c'est- à-dire qu'elles sont destinées à contenir des nombres flottants (approximation de nombres réels). Là encore, nous verrons qu'en C il existe plusieurs types flottants.
En C, **les déclarations des types des variables sont obligatoires et doivent être regroupées au début du programme**.

2.2.4 Pour écrire des informations : la fonction printf

L'instruction :

```
printf ("Bonjour\n") ;
```

appelle en fait une fonction prédéfinie (fournie avec le langage, et donc que vous n'avez pas à écrire vous-même) nommée printf . Ici, cette fonction reçoit un argument qui est : "Bonjour $\backslash n$" Les guillemets servent à délimiter une « chaîne de caractères » (suite de caractères). La notation $\backslash n$ est conventionnelle : elle représente un caractère de fin de ligne, c'est-à-dire un caractère qui, lorsqu'il est envoyé à l'écran, provoque le passage à la ligne suivante.
L'instruction suivante :

```
printf ("Je vais vous calculer %d racines carrées\n", NFOIS) ;
```

ressemble à la précédente avec cette différence qu'ici la fonction printf reçoit deux arguments. Pour comprendre son fonctionnement, il faut savoir qu'en fait le premier argument de printf est ce que l'on nomme un « format »; il s'agit d'une sorte de guide qui précise comment afficher les informations qui sont fournies par les arguments suivants.
Ici, on demande à printf d'afficher suivant ce format : "Je vais vous calculer %d racines carrées $\backslash n$" la valeur de NFOIS , c'est-à-dire, la valeur 5 .

2.2.5 Pour lire des informations : la fonction scanf

L'instruction :

```
scanf ("%f", &x) ;
```

est un appel de la fonction prédéfinie scanf dont le rôle est de lire une information au clavier. Comme printf , la fonction scanf possède en premier argument un format exprimé sous forme d'une chaîne de caractères, ici : %f qui correspond à une valeur flottante.

2.2.6 Pour faire une répétition : l'instruction for

Comme nous le verrons, en langage C, il existe plusieurs façons de réaliser une répétition (on dit aussi une « boucle »). Ici, nous avons utilisé l'instruction for :

```
for (i=0 ; i<NFOIS ; i++)
```

Son rôle est de répéter le bloc (délimité par des accolades « { » et « } ») figurant à sa suite, en respectant les consignes suivantes :

- avant de commencer cette répétition, réaliser :
  $i = 0$
- avant chaque nouvelle exécution du bloc (tour de boucle), examiner la condition :
  $i < NFOIS$
  si elle est satisfaite, exécuter le bloc indiqué, sinon passer à l'instruction suivant ce bloc : à la fin de chaque exécution du bloc, réaliser :
  $i++$
  Il s'agit là d'une notation propre au langage C qui est équivalente à :
  $i = i + 1$

2.2.7 Pour faire des choix : l'instruction if

Les lignes :

```
if (x < 0.0)
printf ("Le nombre %f ne possède pas de racine carrée\n", x) ;
else
{ racx = sqrt (x) ;
printf ("Le nombre %f a pour racine carrée : %f\n", x, racx) ;
}
```

constituent une instruction de choix basée sur la condition $x < 0.0$. Si cette condition est vraie, on exécute l'instruction suivante, c'est-à-dire :

```
printf ("Le nombre \%f ne possède pas de racine carrée\n", x) ;
```

Si elle est fausse, on exécute l'instruction suivant le mot else , c'est-à-dire, ici, le bloc :

```
{ racx = sqrt (x) ;
printf ("Le nombre \%f a pour racine carrée : %f\n", x, racx) ;
}
```

Notez qu'il existe un mot *else* mais pas de mot *then* . La syntaxe de l'instruction if (notamment grâce à la présence de parenthèses qui encadrent la condition) le rend inutile.

2.3 Quelques règles d'écriture

Ce paragraphe expose un certain nombre de règles générales intervenant dans l'écriture d'un programme en langage C. Nous aborderons les éléments suivants :

- les identificateurs
- les mots-clés
- le format libre
- les séparateurs
- les commentaires

2.3.1 Les identificateurs

Les identificateurs servent à désigner les différents « objets » manipulés par le programme. Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur peut désigner :

— un nom de variable ou de fonction,
— un type défini par typedef, struct, union ou enum,
— une étiquette.

Un identificateur est une suite de caractères parmi :

— les lettres (minuscules ou majuscules, mais non accentuées),
— les chiffres,
— le "blanc souligné" (_).

2.3.2 Les mots-clés

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

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	satitic	union	while

que l'on peut ranger en catégories :

- les spécificateurs de stockage
  auto register static extern typedef
- les spécificateurs de type
  char double enum float int long short signed struct union unsigned void
- les qualificateurs de type
  const volatile
- les instructions de contrôle
  break case continue default do else for goto if switch while

- divers
  return sizeof

2.3.3 Les séparateurs

Dans un programme, deux identificateurs successifs entre lesquels la syntaxe n'impose aucun signe particulier (tel que : , = ; * ( ) [ ] { }) doivent impérativement être séparés soit par un espace, soit par une fin de ligne. En revanche, dès que la syntaxe impose un séparateur quelconque, il n'est alors pas nécessaire de prévoir d'espaces supplémentaires (bien qu'en pratique cela améliore la lisibilité du programme). Ainsi, vous devrez impérativement écrire :

- int x,y
  et non :
- intx,y
  En revanche, vous pourrez écrire indifféremment :
- int n,compte,total,p
  ou plus lisiblement :
- int n, compte, total, p

2.3.4 Le format libre

Le langage C autorise une mise en page parfaitement libre. En particulier, une instruction peut s'étendre sur un nombre quelconque de lignes, et une même ligne peut comporter autant d'instructions que vous le souhaitez. Les fins de ligne ne jouent pas de rôle particulier, si ce n'est celui de séparateur, au même titre qu'un espace, sauf dans les « constantes chaînes » où elles sont interdites ; de telles constantes doivent impérativement être écrites à l'intérieur d'une seule ligne. Un identificateur ne peut être coupé en deux par une fin de ligne, ce qui semble évident.

2.3.5 Les commentaires

Comme tout langage évolué, le langage C autorise la présence de commentaires dans vos programmes source. Il s'agit de textes explicatifs destinés aux lecteurs du programme et qui n'ont aucune incidence sur sa compilation.
Ils sont formés de caractères quelconques placés entre les symboles /* et */ . Ils peuvent apparaître à tout endroit du programme où un espace est autorisé. En général, cependant, on se limitera à des emplacements propices à une bonne lisibilité du programme.
Voici quelques exemples de commentaires :

```
] /* programme de calcul de racines carrées */
/* commentaire fantaisiste &ç§{<>} ?%!!!!!! */
/* commentaire s'étendant
sur plusieurs lignes
de programme source
```

```
*/
/* ============================================
*
commentaire quelque peu esthétique
*
*
et encadré, pouvant servir,
*
*
par exemple, d'en-tête de programme
*
============================================ */
```