Introduction générale au langage C

I. Introduction

Le langage C a été créé par Dennis Ritchie aux Laboratoires Bell de Téléphonie en 1972. L'objectif du langage était de développer le système d'exploitation Unix déjà présent sur de nombreux ordinateurs. Très tôt, le langage se révéla puissant et souple, d'où sa facile adoption par les programmeurs professionnels.

Pour faire un programme en langage C (comme dans d'autres langages), une logique fondamentale serait de :

- Définir les objectifs du programme ;
- Choisir les méthodes que l'on veut utiliser pour écrire ce programme ;
- Créer le programme ;
- Enfin, l'exécuter et observer les résultats.

Le grand succès du langage C s'explique par les avantages suivants; C'est un langage:

- Universel

Le langage C n'est pas orienté vers un domaine d'applications spéciales.

- Compact

Le langage C est basé sur un noyau de fonctions et d'opérateurs limités, qui permettent la formulation d'expressions simples, mais efficaces.

- Près de la machine

Il offre des opérateurs qui sont très proches de ceux du langage machine et des fonctions qui permettent un accès simple et direct aux fonctions internes de l'ordinateur (gestion de la mémoire).

- Rapide

Il est possible de développer des programmes efficients et rapides.

- Portable

En respectant le standard ANSI-C, il est possible d'utiliser le même programme sur tout autre système, simplement en le recompilant.

- Extensible

Le langage C est animé par des bibliothèques de fonctions privées ou livrées par de nombreuses maisons de développement.

II. Présentation du langage C

1. Cycle de développement

Les différentes étapes de développement d'un programme C sont:

- la création du code source;
- la compilation ;
- et la création du fichier exécutable.

a. Création du code source

Il s'agit de la phase de saisie, éventuellement de modification du texte du programme. Le texte saisi est constitué d'une série de commandes et de déclarations et est appelé *code source*. C'est la première étape du développement ; le code source est créé à l'aide d'un éditeur.

Exemple 1

```
#include<stdio.h>
main()
{
   printf (" le langage C vous souhaite la bienvenue");
}
```

Le code source sera sauvegardé dans un fichier dont le nom aura l'extension .C.

Par exemple, on peut donner au programme précédent le nom *premier.c*.

b. Compilation du code source

L'ordinateur ne peut pas comprendre le code source C. Il ne comprend que des instructions binaires (langage machine). Le code source devra alors être transmis au *compilateur* qui le transforme en langage machine. On parle de compilation.

Le résultat de la compilation est l'obtention d'un code objet qui sera rangé dans un fichier d'extension .obj (ici *premier.obj*). Le code objet contiendra les mêmes instructions que le code source.

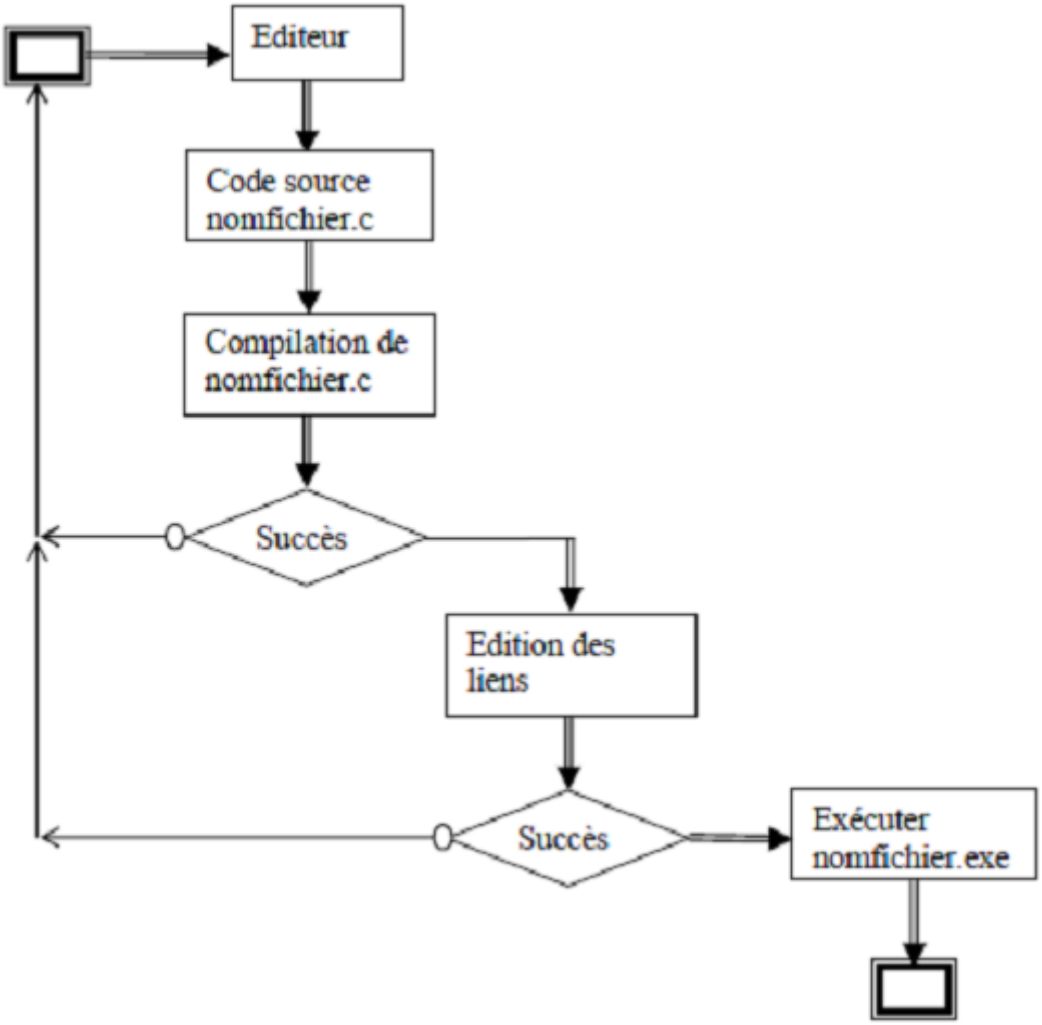
c. Edition des liens

Le fichier objet n'est pas directement exécutable ; son contenu n'est pas complet.

On le complète par l'incorporation des modules objets des routines et fonctions prédéfinies que le programme utilise.

Un compilateur C dispose de routines ou fonctions telle que *printf()*, réalisant rapidement les opérations les plus fréquentes. L'opération d'incorporation est réalisée par un programme dit de chaînage (linker) ou d'édition des liens, qui combine les instructions de programmes présentes dans le fichier objet avec les routines fournies par le compilateur, pour aboutir enfin au code exécutable. On obtiendra alors un fichier exécutable de même nom que le fichier objet mais d'extension .exe (ici *premier.exe*).

2. Schéma récapitulatif



3. Structure d'un programme C

Un programme écrit en C comporte deux parties:

- L'entête;
- Et le corps du programme.

a. En-tête

Utilisation des bibliothèques de fonctions

Il est fréquent d'avoir besoin d'une partie déjà implémentée dans une autre unité de compilation, ou même d'avoir besoin de fonctions de la bibliothèque standard. L'inclusion de source nous permet alors d'y accéder.

La commande est la suivante :

#include<nom du fichier>

Cette commande comme nous l'avons vue, est placée en début du programme.

Cette ligne sera remplacée par le pré-*processeur* qui mettra à cette place le fichier correspondant.

Exemple:

#include <stdio.h>

b. Corps du programme

i. Déclaration des Variables

Les variables contiennent les valeurs qui sont utilisées pendant l'exécution du programme. Les noms des variables sont des identificateurs quelconques.

ii. Déclaration des fonctions

En C, le programme principal et les sous-programmes sont définis comme fonctions. Il n'existe pas de structures spéciales pour le programme principal ni les procédures (comme en Pascal ou en langage algorithmique).

iii. Programme principal

La fonction main est la fonction principale des programmes en C: Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction main. Il est délimité par les accolades « { » et « } ». On dit que les instructions situées entre ces accolades forment un « bloc ».

✓ Les unités lexicales

Maintenant que nous avons la structure générale d'un programme, nous allons voir les types d'unités lexicales qu'utilisent le langage.

Il s'agit principalement:

✓ Des identificateurs

Ils permettent de nommer les entités du programme (variables, fonctions,...);

✓ Règles de construction d'un identifiant:

- Ils doivent être pris à partir de l'alphabet :'a - z' 'A - Z' '0 - 9' '_';
- Doivent commencer par une lettre ou par _.
- La longueur ne doit pas dépasser 31 pour les identificateurs internes (dans ce cas, le compilateur tronque les caractères significatifs);
- La distinction entre majuscules et minuscules n'est pas garantie pour les identificateurs externes.

i, Premier, _d, h1, var, Locale sont des identificateurs valides.

Contrairement à 1i, i!h, p:1

✓ Des mots clés

Ils sont réservés et ne doivent pas être utilisés comme identificateurs:

- Auto
- enum
- restrict
- unsigned
- break
- if
- inline
- do
- switch
- else
- extern
- return
- void
- case
- float
- static
- struct
- double
- register
- short
- volatile
- char
- for
- signed
- _Complex
- _Imaginary
- long
- union
- while
- const
- goto
- sizeof
- _Bool
- default
- int
- typdef

III. Données et Types de base

1. Introduction

Le but d'un programme est de manipuler une ou des donnée(s) afin de produire un ou des résultats.

Les variables et les constantes sont les données principales qui peuvent être manipulées par un programme.

Un objet qui a une valeur fixe est une *constante*; par contre si la valeur est modifiable, elle est appelée *variable*.

Une variable est caractérisée par :

- Son nom, qui indique une zone mémoire de l'ordinateur;
- Sa valeur;
- Et son type (ensembles des valeurs que peut prendre la variable). Utiliser son nom, revient à adresser la donnée qui y est stockée.

Les *déclarations* introduisent les variables qui sont utilisées, fixent leur type et parfois aussi leur valeur de départ. Les *opérateurs* contrôlent les actions que subissent les valeurs des

données. Pour produire de nouvelles valeurs, les variables et les constantes peuvent être combinées à l'aide des opérateurs dans des *expressions*.

2. Les types d'une variable

Le *type* d'une variable détermine l'ensemble des valeurs admissibles, le nombre d'octets à réserver en mémoire et l'ensemble des opérateurs qui peuvent y être appliqués.

Il existe en C plusieurs types de variables:

a. Types simples

Ce sont les ensembles *de nombres et leurs représentations*.

b. Les types entiers

Avant de pouvoir utiliser une variable, nous devons nous intéresser à deux caractéristiques de son type numérique:

(1) le domaine des valeurs admissibles
(2) le nombre d'octets qui est réservé pour une variable

Le tableau suivant résume les caractéristiques des types numériques entiers de C :

définition	*description*	*domaine min*	*domaine max*	*nombre d'octets*
Char	caractère	-128	127	1
Short	entier court	-32768	32767	2
int	entier standard	-32768	32767	2
long	entier long	-2147483648	2147483647	4

Si on ajoute le préfixe *unsigned* à la définition d'un type de variables entières, les domaines des variables sont déplacés comme suit:

définition	*description*	*domaine min*	*domaine max*	*nombre d'octets*
unsigned char	caractère	0	255	1
unsigned short	entier court	0	65535	2
unsigned int	entier standard	0	65535	2
unsigned long	entier long	0	4294967295	4

- **Remarque :** En toute rigueur, chacun des trois types (short, int et long) peut être nuancé par l'utilisation du qualificatif unsigned (non signé). Dans ce cas, il n'y a plus de bit réservé au signe et on ne représente plus que des nombres positifs. Son emploi est réservé à des situations particulières.

- **Remarque :** La norme C99 introduit le type long long, ainsi que des types permettant de choisir :
    - soit la taille correspondante, par exemple int16 pour des entiers codés sur 16 bits ou int32 pour des entiers codés sur 32 bits ;
    - soit une taille minimale, par exemple int_least32_t pour un entier d'au moins 32 bits.

c. Les types rationnels

En C, nous avons le choix entre trois types de rationnels: *float*, *double* et *long double*.

définition	*précision*	*mantisse*	*domaine min*	*domaine max*	*nombre d'octets*
float	simple	6	3.4 * 10-38	3.4 * 1038	4
double	double	15	1.7 * 10-308	1.7 * 10308	8
long double		19	3.4 * 10-4932	1.1 * 104932	10

min et *max* représentent les valeurs minimales et maximales positives. Les valeurs négatives peuvent varier dans les mêmes domaines.

mantisse indique le nombre de chiffres significatifs de la mantisse

3. Déclaration d'une variable

Toute variable doit être déclarée avant son utilisation.

La déclaration de variable indiquera au compilateur le nom et le type de la variable. Si on utilise une variable non déclarée, alors le compilateur génère un message d'erreurs.

a. Syntaxe de la déclaration :

nom_type nom_variable ;

nom_type : indique le type de la variable et doit faire partie des mots-clés répertoriés.

nom_variable : est le nom de la variable déclarée.

Exemple :

int nombre;

Cette déclaration précise que la variable nommée nombre est de type int, c'est-à-dire qu'elle est destinée à contenir des nombres entiers (relatifs).

float taux, note, moyenne ;

b. Remarque

Une déclaration de variable peut s'accompagner de l'initialisation de la variable.

Exemple :

int nombre =10;
float taux = 0.05, note, moyenne ;
L'instruction réserve un emplacement mémoire pour une variable mais de plus, elle y range une valeur.

NB :

Il ne faut initialiser une variable avec une valeur qui ne correspond pas au type déclarée. Le programme pourra donner des résultats erronés sans détection d'erreurs par le compilateur. Les deux initialisations suivantes sont incorrectes :

int nombre =10.000 ;

unsigned int prix = -2510 ;

Remarque :

On peut également définir des types ''synonymes'' avec le mot-clé typedef.

Exemple :

La déclaration

typedef int entier ;

signifie que entier est ''synonyme'' de int, de sorte que les déclarations suivantes soient équivalentes

int n, p ;
entier n, p ;

Remarque :

typedef ne crée pas un nouveau type de donnée ; il permet seulement d'utiliser un nom différent pour un type de donnée déjà défini.

typedef permet de faciliter la lecture des programmes et n'est pas une simple substitution.

Remarque :

Suivant la norme C99, une déclaration peut figurer à n'importe quel emplacement, pour peu qu'elle apparaisse avant que la variable correspondante ne soit utilisée.

4. Les constantes

Une *constante* est un emplacement mémoire utilisé par un programme. Contrairement à une variable, la valeur stockée dans une *constante* ne peut pas changer pendant l'exécution du programme.

Le langage C possède deux types de *constante*: les *constantes littérales* et les *constantes symboliques*.

a. Les constantes littérales

Une *constante littérale* est une valeur qui est introduite directement dans le code source.

Exemple :

int nombre =10 ;
float taux =0.05;

10 et 0.05 sont des *constantes littérales*.

Une constante avec virgule flottante est considérée par le compilateur C comme un nombre *double précision*. Elle peut être représentée avec une notation décimale standard :

2.31 E 2 (= 1.23 10^2 = 123)
8.04 e 6 (= 8.04 10^6 = 8 040 000)

b. Les constantes symboliques

Le langage C offre deux possibilités de représentations des *constantes symboliques*.

- l'ordre (directive) *#define*
- le mot-clé *const*

Cas 1 :

#define nom_constante litterale

La *constante symbolique* est représentée par son nom (symbole) *nom_constante* et par sa valeur *litterale* qui est une *constante littérale*.

L'instruction crée une constante appelée *nom_constante* de valeur *litterale* .

Exemple :

#define PI 3.14159

Convention :

On écrira les noms des *constantes symboliques* en lettres majuscules et les noms des variables en lettres minuscules.

L'instruction *#define X Y* ne se termine pas par un point-virgule (;).

Cas 2 :

const type nom_constante =valeur;

Exemple :

const int nombre =20 ;
const float pi=3.14159 ;
const long compte =21 000 000 , float taux =.05;

nombre, *pi*, *compte* et *taux* sont des *constantes symboliques*.

Si le programme essaie de modifier une variable, alors le compilateur génère un message d'erreurs.

Caractères non imprimables :

\a	sonnerie	\\	trait oblique
\b	curseur arrière	\?	point d'interrogation

\t	tabulation	\'	apostrophe
\n	nouvelle ligne	\"	guillemets
\r	retour au début de ligne	\f	saut de page (imprimante)
\0	NUL	\v	tabulateur vertical

5. Instructions d'Entrée / Sortie

Il existe des d'instructions pour permettre à la machine de dialoguer (communiquer) avec l'utilisateur : les *instructions d'entrées/sorties* ou tout simplement de *lecture/écriture*.

Les deux instructions de base de la communication, que l'on utilisera dans l'écriture des algorithmes, sont :

- La lecture d'informations à fournir par l'utilisateur (le plus souvent au clavier) et leur mémorisation sous un nom choisi par l'utilisateur, que l'on notera : scanf .

Syntaxe :
scanf("format", argument_1, argument_2, argument_3);

L'*adresse d'une variable* est indiquée par le nom de la variable précédé du signe &.

Tableau de formats pour scanf

%d	Entier décimal (int)		%u	Décimal (non signé)
%hd	Décimal (short)		%hu	Décimal (short non signé)
%ld	Décimal (long)		%lu	Décimal (long non signé)
%o	Entier octal (int)		%f	Réel (float)
%ho	Octal (short)		%lf ou %Lf	Réel (double, ou long d.)
%lo	Octal (long)		%e ou %E	Réel (float en exponentiel)
%x	Entier hexadécimal (int)		%le ou %LE	Réel (double ou long d.)
%hx	Hexadécimal (short)		%g ou %G	Réel(float)
%lx	Hexadécimal (long)		%lg ou %LG	Réel (double ou long d.)
%c	Caractère		%s	Chaîne de caractères

Exemple :

scanf("%d" ,&a) ;

Cette instruction est un appel de la fonction prédéfinie scanf dont le rôle est de lire une information au clavier.

- L'écriture d'informations (le plus souvent à l'écran) à destination de l'utilisateur (qui pourra alors les lire). L'information (ou les informations) à écrire peuvent être des messages "textes", placés alors entre guillemets, ou des noms de données (variable) faisant référence à des informations mémorisées ou calculées. Les guillemets servent à délimiter une « chaîne de caractères » (suite de caractères). La nota- tion \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.

Syntaxe :

printf("format", argument_1,argument_2,-----,argument_3);

La chaîne qui donne le format est composée de % et d'une lettre qui donne le type de l'argument à afficher.

Exemple :

Int i=20;

printf("%d",i);

Cette instruction appelle en fait une fonction prédéfinie (fournie avec le langage, et donc que vous n'avez pas à écrire vous-même) nommée printf.

printf("%d", 4+6);

Remarque:

Il vous faudra toujours veiller à accorder le code de format au type de la valeur correspondante. Si vous ne respectez pas cette règle, vous risquez fort d'afficher des valeurs totalement fantaisistes.

Tableau de formats pour printf

Symbole	Type décrit
%d	Entier (entier décimal)
%o	Entier (entier octal)
%x ou %X	Entier (entier hexadécimal)
%u	Entier (entier non signé)

%f	Réel
%e ou %E	Réel (format exponentiel)
%g ou %G	Réel (soit e ou f)
%c	Caractère
%s	Chaîne de caractère

IV. Opérateurs et expressions

1. Notion d'instruction

Une instruction représente une tâche à accomplir par l'ordinateur. En langage C, on écrit une instruction par ligne et elle se termine par un point-virgule (à l'exception de #define et #include).

Exemple :

x = 2 + 3;

2. Les opérateurs

Un *opérateur* est un symbole qui décrit une opération à effectuer sur une ou plusieurs *opérandes*. En langage C, les opérandes sont toujours des expressions. Les opérateurs sont divisés en quatre catégories :

1. L'opérateur d'affectation ;
2. Les opérateurs mathématiques ;
3. Les opérateurs de comparaison ;
4. Les opérateurs logiques.

a. Opérateurs arithmétiques

Opération	Signe	Exemple
Addition	+	s = a + b
Soustraction	-	d = a - b
Multiplication	*	p = a * b
Division	/	q = a / b
Modulo (reste de la division)	%	S = a % b

Pré incrémentation	++ a	(5 * ++a) 20; a 4;
Post incrémentation	a ++	(5 * a++) 15; a 4;
Pré décrémentation	-- a	(5 * --a) 10; a 2;

Post décrémentation		(5 * a--) 15; a 2;

b. Opérateurs de comparaison

Opération	Signe
a == b	a égale b
a != b	a différent de b
a < b	a inférieur à b
a <= b	a inférieur ou égal à b
a > b	a supérieur à b
a >= b	a supérieur ou égal à b

c. Opérateurs logiques

Opération	Signe	Exemples
Et	&	--a = 0 1 0 1 --b = 0 1 1 0 a&b = 0 1 0 0
ou inclusif	|	..a = 0 1 0 1 ..b = 0 1 1 0 a|b = 0 1 1 1
ou exclusif *(xor)*	^	..a = 0 1 0 1 ..b = 0 1 1 0 a^b = 0 0 1 1
Complément à 1 (inversion des bits)	~	a = 0 1 0 1 ~a = 1 0 1 0
décalage à gauche de n bits	<<	b = a << n
décalage à droite de n bits	>>	b = a >> n

d. Opérateurs d'affectation

Opération	Signe	Exemples et équivalences
affectation simple	=	lvalue = expr;
multiplication et affectation	*=	x *= 3; x = x * 3;

division et affectation	/=	x /= 3; x = x / 3;
modulo et affectation	%=	x % = 3; x = x % 3;
addition et affectation	+=	x += 3; x = x + 3;
soustraction et affectation	-=	x -= 3; x = x - 3;
décalage gauche et affectation	<<=	x <<= 3; x = x << 3;
décalage droite et affectation	>>=	x >>= 3; x = x >> 3;
et bit à bit et affectation	&=	x &= 3; x = x & 3;
ou bit à bit et affectation	|=	x |= 3; x = x | 3;
xor bit à bit et affectation	^=	x ^= 3; x = x ^= 3;

e. Expression

En langage C, on appelle expression tout ce qui représente une valeur numérique. On distingue des expressions simples et des expressions complexes.

- L'expression la plus simple peut être constituée d'une seule variable, d'une constante littérale ou d'une constante symbolique.

Exemples :

Expression	Description
PI	Constante symbolique
20	Constante littérale
taux	Variable
-2.51	Constante littérale

- *Les expressions complexes* sont constituées de plusieurs expressions simples avec des opérateurs.

Exemples :

1. *7 + 12 ;* /* 7 et 12 sont 2 sous-expressions et + l'opérateur d'addition */

2. *-2.51 / 16+5 * taux * taux / nombre ;*
/* c'est expression avec plusieurs opérateurs. Son évaluation dépend de l'ordre dans lequel les opérations sont effectuées et des priorités des opérateurs. */

Remarque :

Une instruction d'affectation est elle-même une expression. Ainsi, on peut écrire :
y = x = a + b;
ou
x = 6 + (y = 4 + 5);

V. Les commentaires en C

Ils détaillent le programme et contribuent à la lisibilité et à la compréhension de celui-ci. Ils débutent par /* et se termine par */.
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.

Exemple
/*Ceci est un commentaire*/
NB : Les commentaires ne peuvent pas être imbriqués

Remarque :

La norme C99 autorise une seconde forme de commentaire, dit « de fin de ligne. Un tel commentaire est introduit par // et tout ce qui suit ces deux caractères jusqu'à la fin de la ligne est considéré comme un commentaire. En voici un exemple :

```
printf ("bonjour\n") ;   // formule de politesse
```

VI. Création d'un programme en langage C

La manière de développer et d'utiliser un programme en langage C dépend naturellement de l'environnement de programmation dans lequel vous travaillez. Nous vous fournissons ici quelques indications générales (s'appliquant à n'importe quel environnement) concernant ce que l'on pourrait appeler les grandes étapes de la création d'un programme, à savoir : édition du programme, compilation et édition de liens.

1. L'édition du programme

L'édition du programme (on dit aussi parfois « saisie ») consiste à créer, à partir d'un clavier, tout ou partie du texte d'un programme qu'on nomme « programme source ». En général, ce texte sera conservé dans un fichier que l'on nommera « fichier source ».

Chaque système possède ses propres conventions de dénomination des fichiers. En général, un fichier peut, en plus de son nom, être caractérisé par un groupe de caractères (au moins 3) qu'on appelle une « extension » (ou, parfois un « type ») ; la plupart du temps, en langage

C, les fichiers source porteront l'extension C.

2. La compilation

Elle consiste à traduire le programme source (ou le contenu d'un fichier source) en langage machine, en faisant appel à un programme nommé compilateur. En langage C, compte tenu de l'existence d'un préprocesseur, cette opération de compilation comporte en fait deux étapes :

● **traitement par le préprocesseur** : ce dernier exécute simplement les directives qui le concernent (il les reconnaît au fait qu'elles commencent par un caractère #). Il produit, en résultat, un programme source en langage C pur. Notez bien qu'il s'agit toujours d'un vrai texte, au même titre qu'un programme source : la plupart des environnements de programmation vous permettent d'ailleurs, si vous le souhaitez, de connaître le résultat fourni par le préprocesseur.

● **compilation** proprement dite, c'est-à-dire traduction en langage machine du texte en langage C fourni par le préprocesseur.
Le résultat de la compilation porte le nom de module objet.

3. L'édition de liens

Le module objet créé par le compilateur n'est pas directement exécutable. Il lui manque, au moins, les différents modules objet correspondant aux fonctions prédéfinies (on dit aussi « fonctions standard ») utilisées par votre programme (comme printf, scanf, sqrt).

C'est effectivement le rôle de l'éditeur de liens que d'aller rechercher dans la bibliothèque standard les modules objet nécessaires. Notez que cette bibliothèque est une collection de modules objet organisée, suivant l'implémentation concernée, en un ou plusieurs fichiers.

Le résultat de l'édition de liens est ce que l'on nomme un programme exécutable, c'est-à-dire un ensemble autonome d'instructions en langage machine. Si ce programme exécutable est rangé dans un fichier, il pourra ultérieurement être exécuté sans qu'il soit nécessaire de faire appel à un quelconque composant de l'environnement de programmation en C.

4. Les fichiers en-tête

Nous avons vu que, grâce à la directive #include, vous pouviez demander au préprocesseur d'introduire des instructions (en langage C) provenant de ce que l'on appelle des fichiers « en- tête ». De tels fichiers comportent, entre autres choses :

. ● des déclarations relatives aux fonctions prédéfinies,

. ● des définitions de macros prédéfinies. Lorsqu'on écrit un programme, on ne fait pas toujours la différence entre fonction et macro, puis- que celles-ci s'utilisent de la même manière. Toutefois, les fonctions et les macros sont traitées de façon totalement différente par l'ensemble « préprocesseur + compilateur + éditeur de liens».

. En effet, les appels de macros sont remplacés (par du C) par le préprocesseur, du moins si vous avez incorporé le fichier en-tête correspondant. Si vous ne l'avez pas fait, aucun remplacement ne sera effectué, mais aucune erreur de compilation ne sera détectée : le compilateur croira simplement avoir affaire à un appel de fonction ; ce n'est que l'éditeur de liens qui, ne la trouvant pas dans la bibliothèque standard, vous fournira un message. Les fonctions, quant à elles, sont incorporées par l'éditeur de liens. Cela reste vrai, même si vous omettez la directive #include correspondante ; dans ce cas, simplement, le compilateur n'aura pas disposé d'informations appropriées permettant d'effectuer des contrôles d'arguments (nombre et type) et de mettre en place d'éventuelles conversions ; aucune erreur ne sera signalée à la compilation ni à l'édition de liens ; les conséquences n'apparaîtront que lors de l'exécution : elles peuvent être invisibles dans le cas de fonctions comme printf ou, au contraire, conduire à des résultats erronés dans le cas de fonctions comme sqrt.