USTHB Année 2021/2022
Faculté d'Electronique et d'Informatique MI
Département d'Informatique MEDJADBA

Chapitre 1 Les bases de la programmation en C

1. Les éléments du langage C :

Structure d'un programme C

Un programme écrit en C doit respecter les règles de base suivantes :

- ✓ La fonction main est la fonction principale des programmes en C. elle se trouve obligatoirement dans tous les programmes.
- ✓ Chaque instruction se termine par un point virgule.
- ✓ C distingue les majuscules et les minuscules.

La structure générale d'un programme C est la suivante :

ecrire

Structure générale d'un algorithme	***Structure générale d'un programme C***
Algorithme identificateur ; <Déclaration des constantes> <Déclaration des variables> ***Début*** <Bloc d'instructions> ***Fin.***	**Bibliothèque** **main( )** **{** **<Déclarations>** **<Instructions>** **}**

Bibliothèque :

La pratique en C exige l'utilisation de bibliothèques de fonctions. Ces bibliothèques sont disponibles dans leur forme précompilée (extension .LIB). Pour les utiliser, il faut inclure des fichiers en-tête (***header files, extension.h***) dans les programmes. Ces fichiers contiennent des prototypes des fonctions définies dans les bibliothèques et créent un lien entre les fonctions précompilés et nos programmes.

Exemple :

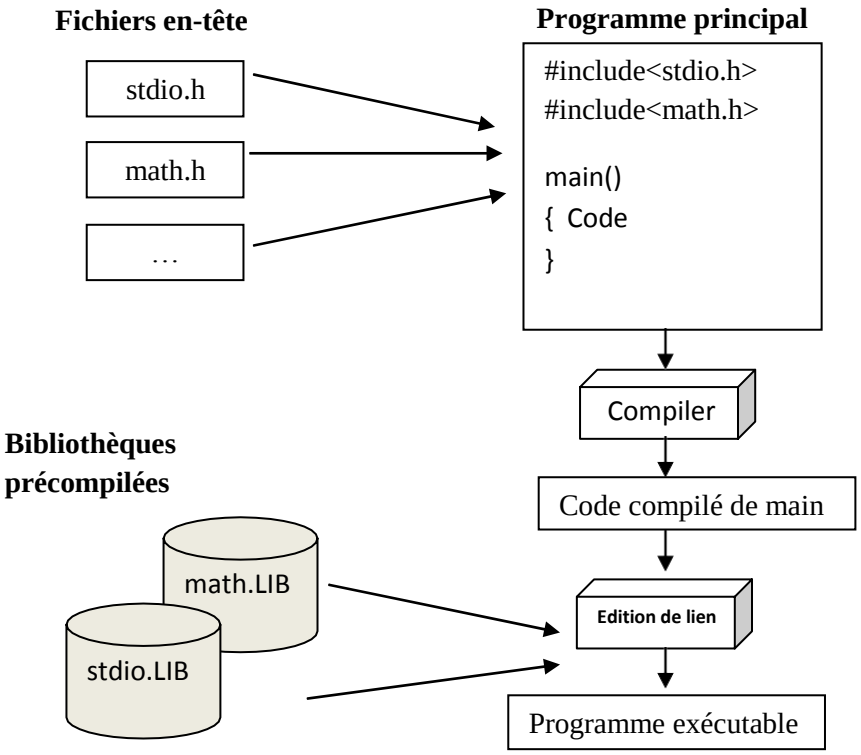
Si l'on écrit un programme qui fait appel à des fonctions mathématiques et des fonctions

d'entrée et de sortie prédéfinies. Pour pouvoir utiliser ces fonctions, le programme a besoin des bibliothèques : MATHS.LIB et STDIO.LIB.

Il faut donc inclure des fichiers en-tête correspondants dans le code source (.c) du programme à l'aide des instructions :

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

L'instruction #include insère les fichiers en-tête indiqués comme arguments dans le texte du programme au moment de la compilation. Après la compilation, les fonctions précompilées des bibliothèques seront ajoutées au programme pour former une version exécutable du programme (voir le schéma suivant).



Démarche générale de la programmation en C

Déclarations des constantes et des variables :

Déclarations des constantes :

Une constante est un objet auquel on attribue une valeur à la déclaration, on ne peut pas la changer tout au long du programme. On peut définir les constantes à l'aide de la directive ***define*** ou le mot clé ***const***.

En algorithmique	*En C*	*Exemple*
< idf> <-- <valeur> ;	**#define <idf> valeur ;**	#define n 100 ;

Déclarations des variables :

a. Syntaxe : X: entier; float X;

En algorithmique	*En C*
<Identificateur> :<type> ; Ou <liste des identificateurs> :<type> ;	<type> <identificateur>; ou <type> <liste des identificateurs> ;

Remarque : dans la liste des identificateurs, ces derniers doivent être séparés par des virgules.

b. Type : Les types de bases sont :

Type	*Signification*
int	Entier
char	Caractère
float	Réel

Remarque 1: Une variable peut avoir une valeur dès sa déclaration (qui peut être changée par le programme).

Exemple : int i=1 ;

Remarque 2: Le type booléen n'existe pas en C, on pourrait utiliser n'importe quel type : le faux sera désigné par la valeur 0 et le vrai par la valeur 1.

c. Les identificateurs

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.
- Une étiquette.
- Un type prédéfini.

Un identificateur est une suite de caractères parmi :

- Les lettres (minuscules ou majuscules, mais non accentuées),
- Les chiffres,
- Le tiret (_).

Le premier caractère d'un identificateur ne peut pas être un chiffre.

Il est déconseillé d'utiliser _ comme premier caractère d'un identificateur car il est souvent employé pour définir les variables de l'environnement C.

Les commentaires

Il existe deux possibilités pour ajouter un commentaire en algorithmique ou en langage C :

- ✓ Elle commence toujours par les deux symboles /* et se termine par les symboles */.
- ✓ Elle commence par les deux symboles //.

Les fonctions d'entrées-sorties :

La fonction d'écriture ou de sortie printf :

Ce n'est pas une instruction du langage C, mais plutôt une fonction de la bibliothèque ***stdio.h***. Cette fonction permet d'afficher un message ou le contenu d'une variable et quand il s'agit d'une expression, elle est évaluée puis sa valeur est affichée.

a. Affichage d'un message :

Ecrire('Message') ;	**printf(''Message'') ;**

b. Affichage d'une valeur ou la valeur d'une expression :

Nombre de variables	***En algorithmique***	***En C***
Une seule variable	Ecrire(Var) ;	printf("format ",Var) ;
Plusieurs variables	Ecrire(Var1,Var2,…, Varn) ;	printf("format1 format2…formatn",var1,var2,…,varn) ;

c. Affichage d'un message et la valeur d'une variable :

Syntaxe	**Ecrire('Message',Nom_var)**	**Printf(**"Message format",var)

Le format d'une variable dépend de son type

Type	***Format***
int	%d
char	%c
float	%f

La fonction de saisie ou de lecture scanf :

Cette fonction permet la saisie de données au clavier, sa définition se trouve aussi dans le fichier 'stdio.h', sa syntaxe est la suivante :

Nombre de variables	***En algorithmique***	***En C***
Une seule variable	Lire(Var) ;	scanf(**"format"**,**&**Var) ;
Plusieurs variables	Lire(Var1,Var2,…,Varn) ;	scanf(**"format1 format2…formatn"**,**&**var1,**&**var2,…,**&**varn) ;

Les opérateurs arithmétiques :

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo(reste de la division)

Les opérateurs relationnels :

Ils permettent de former des expressions de contrôle en comparant deux opérandes.

infériorité stricte	<
infériorité large	<=
supériorité stricte	>
supériorité large	>=
Egalité	= =
Inégalité	!=

Les opérateurs logiques booléens :

Ils permettent de réaliser des expressions logiques ou booléennes, permettant ainsi de former des expressions complexes à partir d'expressions simples.

Négation logique	!
Et logique	&&
Ou logique	||

Affectation :

Affectation simple :

Elle est symbolisée par le signe égale (=). Sa syntaxe est la suivante :

Nom_Var=<expression>

Affectation composée :

Une affectation composée est une affectation avec calcul arithmétique.

+=	i+=20	est équivalent à	i=i+20
-=	i-=k	est équivalent à	i=i-k
*=	i*=k	est équivalent à	i=i*k
/=	i/=k	est équivalent à	i=i/k
%=	i%=k	est équivalent à	i=i%k

Les instructions de branchement conditionnel :

Branchement conditionnel if---else :

Cette instruction peut être tronquée, simple ou imbriquée.

a. Les instructions conditionnelles tronquées :

En algorithmique (2ième possibilité)	*En C*
Si(condition) **Alors** **<Bloc d'Instructions>** **Fsi;**	**If(**condition**)** instr_1; **ou If(**Condition**)** {instr_1;…;instr_n;}

b. Les instructions conditionnelles simples (alternatives) :

En algorithmique (2ième possibilité)	*En C*
Si(condition) **Alors** **<Bloc d'Instructions1>** **Sinon** **<Bloc d'Instructions2>** **Fsi ;**	**if(**condition**)** instr_1; **else** instr_2; **ou if(**Cond**)** {indtr_1;…;instr_n;} **else** {instr_1;…;instr_m;}

4.13.3. Branchement multiple switch :

L'instruction **switch** permet des choix multiples uniquement sur des entiers ou des caractères. Sa syntaxe est la suivante :

En algorithmique	*En C*
cas variable de type caractère ou entier ***vaut*** valeur_1 : <Bloc Instructions1> ; valeur_2 : <Bloc Instructions2> ; **....** ***sinon*** <Bloc InstructionsN> ; ***fincas ;***	***switch (****variable de type caractère ou entier****)*** ***{*** ***case*** *valeur_1 : …… ;* *……;* ***break ;*** ***case*** *valeur_1 : …… ;* *……;* ***break ;*** . . ***Default:*** *…… ;* *…….;* ***}***

Les instructions itératives (les boucles) :

Boucle while (TantQue) :

Tant que la condition est vraie, l'instruction (ou le bloc d'instructions) est exécuté. La condition est évaluée avant l'exécution de la première instruction de la boucle.

En algorithmique (2ième possibilité)	*En C*
Tant que (Condition) ***Faire*** ***<Bloc d'Instructions>*** ***Fait ;***	**while** (cond) **instr;** ou **while** (Cond) **{**instr_1;instr_2;…;instr_n;**}**

Boucle do---while:

Consiste à exécuter l'instruction (ou le bloc d'instruction), tant que la condition reste vérifiée. Dans ce cas, la condition est évaluée en fin de boucle. Les instructions sont donc toujours exécutées au moins une fois.

En algorithmique	*En C*
Faire ***<Bloc d'Instructions>*** ***Tant que*** (Condition) ;	***do*** *instr ;* ***while(****Cond);* ***ou*** ***do*** ***{****instr_1 ;instr_2 ;…* *instr_n ;* ***} while (****cond);*

Boucle for(Pour) :

En algorithmique	*En C*
Pour var de exp1 **à** exp2 **Faire** **<Bloc d'Instructions>** **Fait ;**	**for(**exp1;exp2;exp3**)** **instr;** **ou** **for(**exp1;exp2;exp3**)** **{**instr1;instr2;…;instrn;**}**

- ✓ Exp_1: valeur de départ.
- ✓ Exp_2: condition pour effectuer les instructions de la boucle.
- ✓ Exp_3: pas d'incrémentation.

Exemple

```
for(i=1; i<5; i++)
      {printf("%d",i);}
```