CHAPITRE 1 : NOTIONS DE BASE

Avant de pouvoir comprendre ou même écrire des programmes, il faut connaître la composition des programmes dans le langage de programmation. Dans ce chapitre, nous allons discuter un petit programme en mettant en évidence les structures fondamentales d'un programme en C.

I) Hello C ! :

Ce programme ne fait rien d'autre qu'imprimer les mots suivants sur l'écran :
Comparons d'abord la définition du programme en C avec celle en langage algorithmique.

HELLO_WORLD en langage algorithmique :

```
programme HELLO_WORLD
|   (* Notre premier programme en C *)
|   écrire "hello, world"
fprogramme
```

HELLO_WORLD en C :

```
#include <stdio.h>
main()
/* Notre premier programme en C */
{
   printf("hello, world\n");
   return (0);
}
```

Dans la suite du chapitre, nous allons discuter les détails de cette implémentation.

II) Les composantes d'un programme en C :

Programmes, fonctions et variables :

Les programmes en C sont composés essentiellement de fonctions et de variables. Pour la pratique, il est donc indispensable de se familiariser avec les caractéristiques fondamentales de ces éléments.

1) Les 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 .

Le programme principal étant aussi une 'fonction', nous devons nous intéresser dès le début à la définition et aux caractéristiques des fonctions en C. Commençons par comparer la syntaxe de la définition d'une fonction en C avec celle d'une fonction en langage algorithmique :

Définition d'une fonction en langage algorithmique :

```
fonction <NomFonct> (<NomPar1>, <NomPar2>, ...):<TypeRés>
| <déclarations des paramètres> | <déclarations locales>

|   <instructions>
ffonction
```

Définition d'une fonction en C :

```
<TypeRés> <NomFonct> (<TypePar1> <NomPar1>, <TypePar2> <NomPar2>, ... )
          {
             <déclarations locales>
             <instructions>
          }
```

En C, une fonction est définie par :

- une ligne déclarative qui contient :
  <TypeRés> - le type du résultat de la fonction
  <NomFonct> - le nom de la fonction
  <TypePar1> <NomPar1>, <TypePar2> <NomPar2>, ...
  les types et les noms des paramètres de la fonction
- un bloc d'instructions délimité par des accolades { }, contenant : *<déclarations locales>* - les déclarations des données locales (c'est-à-dire : des données qui sont uniquement connues à l'intérieur de la fonction)
  <instructions> - la liste des instructions qui définit l'action qui doit être exécutée

Résultat d'une fonction :

Par définition, *toute fonction en C fournit un résultat* dont le type doit être défini. Si aucun type n'est défini explicitement, C suppose par défaut que le type du résultat est **int** (integer).

Le retour du résultat se fait en général à la fin de la fonction par l'instruction **return**.

Le type d'une fonction qui ne fournit pas de résultat (comme les procédures en langage algorithmique ou en Pascal), est déclaré comme **void** (vide).

Paramètres d'une fonction :

La définition des paramètres (arguments) d'une fonction est placée entre parenthèses **( )** derrière le nom de la fonction. Si une fonction n'a pas besoin de paramètres, les parenthèses restent vides ou contiennent le mot **void**. La fonction minimale qui ne fait rien et qui ne fournit aucun résultat est alors :

```
void dummy() {}
```

Instructions :

En C, *toute* instruction simple est terminée par un point virgule **;** (même si elle se trouve en dernière position dans un bloc d'instructions). Par exemple :

```
printf("hello, world\n");
```

2) La fonction main :

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**.

Dans les premiers chapitres, nous allons simplement 'traduire' la structure programme du langage algorithmique par une définition équivalente de la fonction **main** :

Définition du programme principal en langage algorithmique :

```
programme <NomProgramme>
|   <déclarations>
|   <instructions>
fprogramme
```

Définition de la fonction main en C :

```
main()
{
   <déclarations>
   <instructions>
   return (0);
}
```

Résultat de main :

En principe tout programme devrait retourner une valeur comme code d'erreur à son environnement. Par conséquent, le type résultat de **main** est toujours **int**. En général, le type de **main** n'est pas déclaré explicitement, puisque c'est le type par défaut. Nous allons terminer nos programmes par l'instruction :

```
return (0);
```

qui indique à l'environnement que le programme s'est terminé avec succès, sans anomalies ou erreurs fatales.

Paramètres de main :

- Si la liste des paramètres de la fonction **main** est vide, il est d'usage de la déclarer par **()**.
- Si nous utilisons des fonctions prédéfinies (par exemple : **printf**), il faut faire précéder la définition de **main** par les instructions **#include** correspondantes.

Remarque avancée :

Il est possible de faire passer des arguments de la ligne de commande à un programme. Dans ce cas, la liste des paramètres doit contenir les déclarations correspondantes. Dans notre cours, nous n'allons pas utiliser des arguments de la ligne de commande. Ainsi la liste des paramètres de la fonction **main** sera vide (**void**) dans tous nos exemples et nous pourrons employer la déclaration suivante qui fait usage des valeurs par défaut :

```
main() { ... }
```

Voici l'exemple d'un programme utilisant des arguments de la ligne de commande,

Avec la forme :

```
int main(int argc, char *argv[])
```

on peut passer des parametres sur la ligne d'exécution Ainsi si on a le programme : **MyProg.c**

```
#include <stdio.h>

int main(int argc, char *argv[])
{
 int idx;
 for (idx = 0; idx < argc; idx++)
 {
  printf("parameter %d value is %s\n", argc, argv[idx]);
 }
 return (0);
}
```

Si j'exécute MyProg de la façon suivante :

```
MyProg param1 param2 param3
```

J'aurai comme résultat :

```
Parameter 0 value is MyProg
Parameter 1 value is param1

Parameter 2 value is param2
Parameter 3 value is param3
```

3) Les variables :

Les variables contiennent les valeurs qui sont utilisées pendant l'exécution du programme. Les noms des variables sont des identificateurs quelconques Les différents types de variables simples et les opérateurs admissibles sont discutés au chapitre 2.

4) Les identificateurs :

Les noms des fonctions et des variables en C sont composés d'une suite de lettres et de chiffres. Le premier caractère doit être une lettre. Le symbole '_' est aussi considéré comme une lettre.

* L'ensemble des symboles utilisables est donc
  : {0,1,2,...,9,A,B,...,Z,_,a,b,...,z}
* Le premier caractère doit être une lettre (ou le symbole '_')
* C distingue les majuscules et les minuscules, ainsi :
  '**Nom_de_variable**' est différent de '**nom_de_variable**'
* La longueur des identificateurs n'est pas limitée, mais C distingue 'seulement' les 31 premiers caractères.

Remarques :

- Il est déconseillé d'utiliser le symbole '_' comme premier caractère pour un identificateur, car il est souvent employé pour définir les variables globales de l'environnement C.
- Le standard dit que la validité de noms externes (par exemple noms de fonctions ou var. globales) peut être limité à 6 caractères (même sans tenir compte des majuscules et minuscules) par l'implémentation du compilateur, mais tous les compilateurs modernes distinguent au moins 31 caractères de façon à ce que nous pouvons généraliser qu'en pratique les règles ci-dessus s'appliquent à tous les identificateurs.

Exemples :

Identificateurs corrects :	*Identificateurs incorrects :*
nom1	1nom
nom_2	nom.2
_nom_3	-nom-3
Nom_de_variable	Nom de variable
deuxieme_choix	deuxième_choix
mot_francais	mot_français

5) Les commentaires :

Un commentaire commence toujours par les deux symboles '**/***' et se termine par les symboles '***/**'. Il est interdit d'utiliser des commentaires imbriqués.

Exemples :

```
/* Ceci est un commentaire correct */
/* Ceci est /* évidemment */ défendu */
```

III) Discussion de l'exemple 'Hello_World' :

Reprenons le programme 'Hello_World' et retrouvons les particularités d'un programme en C.

HELLO_WORLD en C :

```
#include <stdio.h>
main()
/* Notre premier programme en C */
{
 printf("hello, world\n");
 return (0);
}
```

Discussion :

- La fonction **main** ne reçoit pas de données, donc la liste des paramètres est vide.
- La fonction **main** fournit un code d'erreur numérique à l'environnement, donc le type du résultat est **int** et n'a pas besoin d'être déclaré explicitement.
- Le programme ne contient pas de variables, donc le bloc de déclarations est vide.
- La fonction **main** contient deux instructions :
  * l'appel de la fonction **printf** avec l'argument "hello, world\n";
    Effet : Afficher la chaîne de caractères "hello world\n".
  * la commande **return** avec l'argument 0 ;
    Effet : Retourner la valeur 0 comme code d'erreur à l'environnement.
- L'argument de la fonction **printf** est une chaîne de caractères indiquée entre les guillemets. Une telle suite de caractères est appelée *chaîne de caractères constante (string constant).*
- La suite de symboles '**\n**' à la fin de la chaîne de caractères "hello, world\n" est la notation C pour '*passage à la ligne*' (angl : new line). En C, il existe plusieurs couples de symboles qui contrôlent l'affichage ou l'impression de texte. Ces *séquences d'échappement* sont toujours précédées par le caractère d'échappement '\'.

printf et la bibliothèque <stdio> :

La fonction **printf** fait partie de la bibliothèque de fonctions standard <*stdio*> qui gère les entrées et les sorties de données. La première ligne du programme :

`#include <stdio.h>` instruit le compilateur d'inclure le fichier en-tête '**stdio.h**' dans le texte du programme.

Le fichier '**stdio.h**' contient les informations nécessaires pour pouvoir utiliser les fonctions de la bibliothèque standard <*stdio*>