

Chapitre I

INTRODUCTION AU LANGAGE C

Objectifs

Général

⇒ *Initiation au langage de programmation C.*

Spécifiques

- ❑ *Connaître la structure d'un programme C ;*
 - ❑ *Connaître les composantes élémentaires d'un programme ;*
-

1. Introduction

1.1. Historique

Le langage C a été mis au point par D.Ritchie et B.W.Kernighan au début des années 70. Leur but était de permettre de développer un langage qui permettrait d'obtenir un système d'exploitation portable. Ils se sont inspirés d'autres langages, pour créer un nouveau langage: le langage C.

La première définition de ce langage a été donnée dans leur livre commun « The C programming language ».

Toutefois, suite à l'apparition de nombreux compilateurs C, l'ANSI (abréviation de *American National Standards Institute*) a décidé de normaliser ce langage pour donner ce que l'on appelle le C-ANSI. Suite à cette norme, Ritchie et Kernighan ont sorti une deuxième édition du livre en intégrant les modifications apportées par l'ANSI.

1.2. Avantages

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

→ Universel

C n'est pas orienté vers un domaine d'applications spéciales, comme par exemple FORTRAN (applications scientifiques et techniques) ou COBOL (applications commerciales ou traitant de grandes quantités de données).

→ Compact

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

→ Moderne

C est un langage structuré, déclaratif et récursif; il offre des structures de contrôle et de déclaration comparables à celles des autres grands langages de ce temps (FORTRAN, COBOL, PASCAL, etc.).

→ Près de la machine

Comme C a été développé en premier lieu pour programmer le système d'exploitation UNIX, 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.

→ Rapide

Comme C permet d'utiliser des expressions et des opérateurs qui sont très proches du langage machine, il est possible de développer des programmes efficaces et rapides.

→ Indépendant de la machine

Bien que C soit un langage près de la machine, il peut être utilisé sur n'importe quel système en possession d'un compilateur C. Aujourd'hui C est devenu le langage de programmation standard dans le domaine des micro-ordinateurs.

→ Portable

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

→ Extensible

C ne se compose pas seulement des fonctions standards ; le langage est animé par des bibliothèques de fonctions privées ou livrées par de nombreuses maisons de développement.

1.3. Désavantages

Evidemment, rien n'est parfait ; C a les désavantages suivants :

→ Efficience et compréhensibilité

En C, nous avons la possibilité d'utiliser des expressions compactes et efficaces. D'autre part, nos programmes doivent rester compréhensibles pour nous-mêmes et pour d'autres. Mais, ces deux exigences peuvent se contredire réciproquement.

Les formulations 'courtes' représentent le bon style dans C et sont de loin préférables aux deux autres. Nous constatons donc que :

- La programmation efficace en C nécessite beaucoup d'expérience et n'est pas facilement accessible à des débutants.
- Sans commentaires ou explications, les programmes peuvent devenir incompréhensibles, donc inutilisables.

→ Discipline de programmation

Le programmeur a donc beaucoup de libertés, mais aussi des responsabilités : il doit veiller lui-même à adopter un style de programmation propre, solide et compréhensible.

2. L'environnement et les bibliothèques

Pour le travail pratique en C, il faut utiliser un compilateur et un éditeur facile à utiliser. A titre d'exemple on décrit ici l'utilisation de l'environnement Borland C++ (Version 3.1).

Ce programme nous offre une surface de programmation confortable et rapide. Tout autre compilateur qui permet la programmation selon le standard ANSI-C fait aussi bien l'affaire.

2.1. Description de l'environnement Borland C

Borland C est une implémentation complète du standard C. Le compilateur Borland C est capable de produire du code C selon le standard ANSI-C. D'autre part, Borland C nous offre toute une série de bibliothèques, qui permettent d'exploiter les capacités du PC.

On se limite à une brève description des menus de Borland C.

- ✓ **FILE** : gestion des fichiers, retour au DOS.
- ✓ **EDIT** : commandes d'édition du texte.
- ✓ **SEARCH** : recherche d'un texte, d'une déclaration de fonction, de la position d'une erreur dans le programme.
- ✓ **RUN** : exécution d'un programme.
- ✓ **COMPILE** : traduction et/ou enchaînement des programmes.
- ✓ **DEBUG** : détection d'erreurs en inspectant les données.
- ✓ **PROJECT** : gestion de projets.
- ✓ **OPTIONS** : changement et sauvetage des réglages par défaut:
 - des menus,
 - des options du compilateur,
 - de l'environnement.
- ✓ **WINDOW** : visualisation et disposition des différentes fenêtres sur l'écran.
- ✓ **HELP** : système d'aide

2.2. Les bibliothèques de fonctions prédéfinies

▪ Utilisation des bibliothèques de fonctions

La pratique en C exige l'utilisation de bibliothèques de fonctions. Ces bibliothèques sont disponibles dans leur forme précompilée (extension: **.LIB**). Pour pouvoir les utiliser, il faut inclure des fichiers en-tête (*header files* - extension **.H**) dans nos programmes.

Ces fichiers contiennent des '*prototypes*' des fonctions définies dans les bibliothèques et créent un lien entre les fonctions précompilées et nos programmes.

▪ L'instruction **#include**

L'instruction **#include** insère les fichiers en-tête indiqués comme arguments dans le texte du programme au moment de la compilation.

▪ Identification des fichiers

Lors de la programmation en Borland C, nous travaillons donc avec différents types de fichiers qui sont identifiés par leurs extensions:

*.C	fichiers source
*.OBJ	fichiers compilés (versions objet)
*.EXE	fichiers compilés et liés (versions exécutables)
*.LIB	bibliothèques de fonctions précompilées
*.H	fichiers en-tête (<i>header files</i>)

2.3. Phases de compilation

La compilation se fait généralement en plusieurs phases:

→ Le compilateur transforme le code source en code objet, et le sauvegarde dans un **fichier objet**, c'est-à-dire qu'il traduit le fichier source en langage machine.

→ Le compilateur fait ensuite appel à un **éditeur de liens** (en anglais **linker**) qui permet d'intégrer dans le fichier final tous les éléments annexes (fonctions ou bibliothèques) auquel le programme fait référence mais qui ne sont pas stockés dans le fichier source.

→ Puis, il crée un **fichier exécutable** qui contient tout ce dont il a besoin pour fonctionner de façon autonome, le fichier ainsi créé possède l'extension **.exe**.

Dans le cas du langage C, une phase supplémentaire apparaît, il s'agit du traitement du fichier par le préprocesseur C, un programme permet d'inclure dans le fichier source les éléments référencés par les instructions situées au début du fichier source (instructions précédées du caractère **#**).

Les phases de compilation dans le cas d'un compilateur C sont donc les suivantes:



3. 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 cette section, nous allons discuter un petit programme en mettant en évidence les structures fondamentales d'un programme en C.

3.1. Hello C !

Suivons la tradition et commençons la découverte de C par l'inévitable programme « Hello World ». Ce programme ne fait rien d'autre qu'imprimer les mots suivants sur l'écran: « Hello World ». Comparons d'abord la définition du programme en C avec celle en langage algorithmique.

HELLO_WORLD en langage algorithmique

Algorithme HELLO_WORLD

Début

| Ecrire ("hello, world")

Fin

HELLO_WORLD en C

```
#include <stdio.h>
main( )
{
    printf("hello, world\n");
    return 0;
}
```

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

3.2. Les composantes d'un programme en C

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.

▪ 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 (comme en langage algorithmique).

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>
finfonction
```

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>, ..., <TypeParN><NomParN>: 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 à d: des données qui sont uniquement connues à l'intérieur de la fonction).
- ✓ <instructions>: la liste des instructions qui définissent 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), 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");
```

- **La fonction main**

La fonction **main** est la fonction principale des programmes en C (se trouve obligatoirement dans tous les programmes). L'exécution d'un programme entraîne automatiquement l'appel de la fonction **main**.

Définition du programme principal en langage algorithmique

```
Algorithme <NomAlgo>  
| <déclarations>  
| <instructions>  
finAlgo
```

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.

- **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 */
```

▪ 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 suivant.

▪ 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.

→ Les espaces ne sont pas autorisés.

→ Les noms de variables ne peuvent pas être un noms réservés :

- auto, break, case, continue, default, goto,
- do, switch, else, for, if, while,
- enum, extern, union, unsigned void, volatile,
- float, int, char, const, double, long, short,
- register, return,
- signed, sizeof, static, struct, typedef.

Exemples

<i>Identificateurs corrects</i>	<i>Identificateurs incorrects</i>
nom1	1nom
nom_2	nom.2
_nom_3	-nom-3
Nom_de_variable	Nom de variable
deuxieme_choix	deuxième_choix
mot_francais	mot_français

Exercice 1.1

Lesquels des identificateurs suivants sont acceptés par C ?

- a) fonction-1
- b) _MOYENNE_du_MOIS_
- c) 3e_jour
- d) limite_inf.
- e) lim_supérieure
- f) __A_

Solution

fonction-1	pas accepté (contient le caractère spécial '-')
------------	---

<code>_MOYENNE_du_MOIS_</code>	accepté
<code>3e_jour</code>	pas accepté (chiffre au début de l'identificateur)
<code>limite_inf.</code>	pas accepté (contient le caractère spécial '.')
<code>lim_supérieure</code>	pas accepté (contient le caractère spécial 'é')
<code>_A_</code>	accepté

4. Discussion de l'exemple 'Hello_World'

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

4.1. *HELLO_WORLD en C*

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

4.2. *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*.
- 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*'. 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 '\'.

4.3. *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>
```

Elle permet au 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>**.

Exercice 1.2

Modifiez le programme 'hello world' de façon à obtenir le même résultat sur l'écran en utilisant plusieurs fois la fonction **printf**.

Solution

Voici une des solutions possibles:

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

Exercice 1.3

Expérimentez avec les séquences d'échappement que vous trouvez dans le tableau ci-dessous et complétez les colonnes vides.

<i>séq. d'échapp.</i>	<i>descr. anglaise</i>	<i>descr. française</i>
<code>\n</code>	new line	passage à la ligne
<code>\t</code>		
<code>\b</code>		
<code>\r</code>		
<code>\"</code>		
<code>\\</code>		
<code>\0</code>		
<code>\a</code>		

Solution

séq. d'échapp.	descr. anglaise	descr. française
<code>\n</code>	new line	passage à la ligne
<code>\t</code>	tabulator	tabulation
<code>\b</code>	back	curseur arrière
<code>\r</code>	return	retour au début de la ligne
<code>\"</code>	quotation marks	guillemets
<code>\\</code>	back-slash	trait oblique
<code>\0</code>	NUL	fin de chaîne
<code>\a</code>	attention (bell)	signal acoustique

Exercice 1.4

Ci-dessous, vous trouvez un simple programme en C. Essayez de distinguer et de classer autant que possible les éléments qui composent ce programme (commentaires, variables, déclarations, instructions, etc.)

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

```
/* Ce programme calcule la somme de 4 nombres entiers
introduits au clavier.*/
main( )
{
    int NOMBRE, SOMME, COMPTEUR;

    /* Initialisation des variables */
    SOMME = 0;
    COMPTEUR = 0;
    /* Lecture des données */
    while (COMPTEUR < 4)
    {
        /* Lire la valeur du nombre suivant */
        printf("Entrez un nombre entier :");
        scanf("%i", &NOMBRE);
        /* Ajouter le nombre au résultat */
        SOMME += NOMBRE;
        /* Incrémenter le compteur */
        COMPTEUR++;
    }
    /* Impression du résultat */
    printf("La somme est: %i \n", SOMME);
    return 0;
}
```

Solution

- **Commande au compilateur:** `#include<stdio.h>` pour pouvoir utiliser les fonctions **printf** et **scanf**.
- **Fonction main** n'a pas de paramètres (la liste des paramètres est vide) et fournit par défaut un résultat du type **int** (à l'environnement).
- **Commentaires** (toutes les chaînes de caractères mises entre `/* */`).
- **Variables utilisées:** **NOMBRE**, **SOMME**, **COMPTEUR** déclarées comme entiers (type **int**).
- **Fonctions utilisées:** **printf**, **scanf** de la bibliothèque `<stdio>`.
 - ✓ La fonction **scanf** est appelée avec deux paramètres: le format de saisie de la donnée (ici: `"%i"` pour lire un entier du type **int**) et l'adresse de la variable destination (ici: l'adresse de **NOMBRE**).
 - ✓ La fonction **printf** est appelée avec un respectivement avec deux paramètres: le premier paramètre est une chaîne de caractères, qui peut contenir une information pour le format d'affichage des variables indiquées dans la suite (ici: `"%i"` pour afficher la valeur du type **int** contenue dans **SOMME**). Les paramètres qui suivent la chaîne de caractères indiquent les noms des variables à afficher (ici: **SOMME**).
- **Structure de contrôle:** **while (<condition>)** {...} répète le bloc d'instructions aussi longtemps que la `<condition>` est remplie.
- **Opérateurs:**

+=	opérateur arithmétique d'affectation
++	opérateur arithmétique

<	opérateur de comparaison
=	opérateur d'affectation

→ L'instruction **return 0 ;** retourne la valeur zéro comme code d'erreur à l'environnement après l'exécution du programme.
