# PROGRAMMATION IMPERATIVE EN C

Johann PAVIS (selon source : D. JANIN)

# Table des matières

# Chapitre 1

# INTRODUCTION A LA PROGRAMMATION EN C

## 1.1 Machines programmables

$$\text{prg} \rightarrow \underset{\text{Exécution}}{\boxed{\text{Ordinateur}}} \rightarrow \left\{ \begin{array}{l} \text{- calculs (scientifiques)} \\ \text{- interaction (systèmes d'exploitation)} \\ \text{- contrôle (train, voiture, ...)} \end{array} \right.$$

Plus en détails :

$$\begin{array}{c} \text{PRG en} \\ \text{langage évolué} \end{array} \rightarrow \boxed{\begin{array}{c} \text{Compilateur /} \\ \text{Traducteur} \end{array}} \begin{array}{c} \text{Programmes en code} \\ \rightarrow \\ \text{machine (illisible)} \end{array}$$

$$\rightarrow \boxed{\text{Machine}} \rightarrow \text{Comportement}$$

Mais ATTENTION, un programmeur peut quand même lors de la lecture du programme l'exécuter de tête, sans passer par le code machine. Il ne faut pas non plus confondre l'exécution du programme (propriétés dynamiques) et sa compilation (analyse statique de ses propriétés).

## 1.2 Structure générale d'un programme

Un programme est une liste de commandes, d'instructions :
1. Calculer 2+3
2. Faire la lessive
3. Ne rien faire à partir de là
4. Hurler "I am the kimg of the world"
5. Lever la jambe
6. Recommencer au début

On voit apparaître :
→ des instructions élémentaires (n$^\circ$ 1)
→ des instructions de contrôle de flot d'exécution (n$^{\circ s}$3 et 6)
→ des instructions complexes (n$^\circ$ 2) qu'on peut voir comme des sous-programmes.

### 1.2.1 Structure dynamique

A un instant donné, c'est l'état d'un programme, c'est à dire l'état de la mémoire associée au programme, qui est structurée comme suit (architecture de Von Neumann)

**Zone de données** (ou segment de données)
Elle est accessible en lecture et en écriture.

**Zone de texte**
Elle contient la liste des instructions et est accessible en lecture seulement.

**Zone de contrôle**
Elle contient des registres de contrôle d'exécution, dont *PC (program counter)* qui contient le n$^\circ$ du programme (exécution élémentaire) en cours.

### 1.2.2 Algorithme d'exécution d'un programme

PC :=0
Répéter

$$\left[\begin{array}{l}\text{PC :=PC+1}\\ \text{Exécuter Instruction [PC-1]}\end{array}\right.$$

### 1.2.3 Instructions possibles (plus en détail)

- Calcul
- E / S mémoire
- Branchement / saut (contrôle)

  → PC :=PC + $x$ (branchement **relatif**)
  → PC :=a (branchement **absolu**)

**Attention** : En C, on ne manipule **jamais** explicitement le *program counter*. On le fait indirectement via les boucles (`Repeat,While,For,...`), les conditionnelles (`if,else,...`), les appels de sous-programmes, ...

## 1.3 Premiers pas en C

Un programme en C est une liste de méthodes, appelées `fonction`, dont on distingue une méthode particulière : la fonction de départ, nommée `main` (en C).

Exemple :

```
#include <stdio.h>
int main()
 {
 printf(``Bienvenue \ n'') ;
 return(EXIT_SUCCESS) ;
 }
```

# Chapitre 2

# PREMIERS PAS EN C

Un programme en C est une liste de `fonctions`, pouvant éventuellement s'appeler mutuellement, dont on distingue la fonction initiale : `main`.

Ces fonctions manipulent des valeurs typées. Un type est un domaine de valeurs possibles et des opérations associées.

**Exemple** : `int` + `addition,multiplication,comparaison,...` `char` + `‘+‘,‘<‘,...`

**Remarque** : Ces valeurs sont souvent stockées dans des *variables*.

**Exemple** :

| | | |
|---|---|---|
| `int x=2;` | → | initialisation de x à la valeur 2 |
| `x=3;` | → | affection de la valeur 3 à x |
| `y=x;` | → | copie de la valeur de x dans y |

Une variable est donc définie par :
- une valeur du type correspondant (appelée `rvalue` par gcc)
- un emplacement mémoire dans lequel cette valeur est stockée (appelée `lvalue` par gcc)

**Exemple** : Dans l'affection `x=y`, on a besoin de la valeur droite de y et de la valeur gauche de x.

Les fonctions sont déclarées / définies de la faç on suivante :

```
[type renvoyé] [nom fonction] ([argument(s)])
{
[liste des instructions]
}
```

Un *appel de fonction* est une copie des valeurs des arguments d'appel dans les variables de la fonction appelée, suivie d'une exécution de la fonction appelée.

Autrement dit : les arguments d'une fonction lors de sa définition ont un statut de **variables locales** à la fonction qui sont initialisées lors de l'appel.

**Attention** : En C, chaque variable à une certaine visiblité : le corps de la fonction pour les arguments, le bloc (et les sous-blocs) de définition pour les variables locales.

La visibilité d'une variable peut être réduite par des phénomènes de masquage.

**Exemple**

```
int multiplication_lente (int x , int y)
     {
       int z;
       int r;
       z=x;
          {
            int z=0;
            while (x>0)
               {
                 z=z+y;
                 x=x-1;
               }
            r=z;
          }
       return r;
     }
```

## 2.1 Les pointeurs

On a déjà vu (ou moins pour les variables) que chaque donnée se caractérise par :

- une valeur droite (valeur typée et donc prenant une certaine taille en mémoire)
- une valeur gauche (l'adresse mémoire à laquelle est stockée cette donnée)

En C, comme dans la plupart des langages, ces adresses sont elles-mêmes des données appelées **pointeurs** (ou référence, dans d'autres langages).