Méthodologie de la programmation

Pablo Rauzy
pr@up8.edu
pablo.rauzy.name/teaching/mdlp

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS
UFR MITSIC / L1 informatique

Séance f

Aperçu du langage C : introduction

Aperçu du langage C : introduction

Introduction

- Le langage C est un langage impératif, statiquement et faiblement typé.
- C'est un langage compilé, assez bas niveau (“assembleur portable”).

Introduction

- Le langage C est un langage impératif, statiquement et faiblement typé.
- C'est un langage compilé, assez bas niveau ("assembleur portable").
- Dans ce cycle du cours (deux cours magistraux puis un TP) on va voir un aperçu du langage C et des outils de compilation qui y sont liés.

Variables

- On déclare une variable en donnant son type :
  - `char letter;`
  - `int age;`
  - `double height;`
- Optionnellement on peut lui affecter une valeur directement :
  - `char letter = 'c';`
  - `int age = 42;`
  - `double height = 1.75;`

Expressions

- Une *expression* est une combinaison d'éléments du langage qui retourne une valeur quand elle est *évaluée* :
  - `2 + 3`
  - `age > 20`
  - `height * 100`

Tableaux

- La syntaxe pour déclarer un tableau est la suivante :
  - `int list[3];`
  - `char name[6];`
- Optionnellement on peut lui affecter une valeur directement :
  - `int list[3] = { 1, 2, 3 };`
- En C les chaînes de caractères sont représentées par des tableaux de **char** dont le dernier élément est 0 (`0` ou `'\0'`) :
  - `char name[6] = { 'R', 'a', 'u', 'z', 'y', 0 };`
  - `char name[6] = "Rauzy";`

- On peut aussi déclarer des structures en C :

```
struct struct_name {
  decls // déclarations des membres et de leur type
};
```

- Exemple :

```
struct NamedPoint {
  char label;
  int x, y;
};
```

- On déclare une variable *instance* de cette structure comme avec un type de base :
  - `struct Point p;`
  - `struct Point triangle[3];`
- On accède aux *champs* (ou *membres*) d'une instance de la structure avec un `.` :
  - `p.x`
  - `triangle[0].label`

- On peut définir des raccourcis pour le nom des structures avec `typedef` :
  - `typedef existing_type new_name;`
- Par exemple :
  - `typedef struct Point Point;`

Instructions

- Une *instruction* se termine par un `;`.
- Il peut s'agir d'une déclaration, d'une affectation, d'une condition, d'une boucle, ou d'un appel de fonction.

Blocs

- Un *bloc* est délimité par { et }.
- Il peut contenir une ou plusieurs instructions.

Conditions

- En C les expressions booléennes renvoie `1` si elles sont vraies ou `0` si elles sont fausses.
- Les opérateurs booléens sont :
  - $==$ pour l'égalité,
  - $\neq$ pour la différence,
  - $<$, $\leqslant$, $>$, et $\geqslant$ pour les inégalités,
  - `&&` pour le "et",
  - `||` pour le "ou",
  - `!` pour le "non".
- La syntaxe de la condition est :

```
1 if (cond)
2   instr-or-block
3 else // optionnel
4   instr-or-block
```

- Il existe un autre type de construction conditionnelle en C :

```
switch (expr) {
  case expr:
    instr-or-block
    break; // si on veut sortir du switch
  // potentiellement d'autres case
  default: // optionnel
    instr-or-block
}
```

Boucles

- Il existe trois types de boucle en C.
- La boucle `while` :

```
while (cond)
  instr-or-block
```

- La boucle `do … while` :

```
do instr-or-block
while (cond);
```

- La boucle `for` :

```
for (init-expr; cond; iter-expr)
  instr-or-block
```

Fonctions

- La déclaration de fonction en C donne son type de retour et le type de ses arguments.
- La syntaxe est la suivante :

```
1 ret-type function_name (args)
2 block
```

- Où `args` est une liste séparée par des virgules de déclaration de variable qui seront les arguments de la fonction.
- Et `block` est un bloc d'instruction pouvant contenir l'instruction `return` qui renvoie une valeur.

- Certaines fonctions ne renvoient rien, dans ce cas on parle parfois de procédure.
- Leur type de retour est alors **`void`**.

- On ne peut appeler une fonction en C que si elle est déjà déclarée.
- Cependant il n'est pas obligatoire de l'avoir déjà définie.
- On peut simplement déclarer son existence avec son *prototype*.
- Exemple :
  - `int add (int a, int b);`

- En C les fonctions reçoivent leur arguments par *valeur*.
- C'est à dire que la fonction reçoit une *copie* des variables passées en argument.
- Il en va de même pour ce que les fonctions retournent.

Pointeurs

- En C on peut manipuler les *adresses* des variables dans la mémoire.
- Pour déclarer un pointeur on utilise une `*` devant le nom de la variable.
- Pour récupérer l'adresse d'une variable on utilise `&` devant celle-ci.
- Pour *déréférencer* un pointeur, c'est à dire accéder à la valeur de la variable vers laquelle il pointe, on utilise `*` devant le pointeur.
- Exemples :

```c
char *name;  // name est un pointeur sur un char
int a = 42;  // a est un entier qui vaut 42
int *n = &a; // n est un pointeur sur un entier qui pointe vers a (sa valeur est l'adresse de a)
*n;          // cette expression vaut 42
*n = 13;     // a vaut maintenant 13
```

- La constante `NULL` est un pointeur particulier, de type **`void *`**, qui est compatible avec tous les types de pointeurs.
- Elle est utilisée comme valeur “ne pointe vers rien” d’un pointeur, pour le différencier d’un pointeur valide et initialisé.

- Quand on incrémente un pointeur (qu'on lui ajoute 1), celui-ci est en fait incrémenté de la taille du type vers lequel il pointe.
- Les tableaux en C peuvent être vu comme des pointeurs.

```c
int list[3] = { 1, 2, 3 };
int *l = list;  // pas besoin du &

list[2] == *(l + 2) // pourtant sizeof(int) est 4
```

Programmes

- Un programme en C est une suite de déclaration de fonctions et variables (mais il vaut mieux éviter l'utilisation de variables *globales*).
- Une fonction spéciale `main` est le *point d'entrée* du programme.
- La fonction `main` renvoie un entier :
  - si le programme termine normalement, celui-ci doit être 0 ;
  - sinon, celui-ci est un code d'erreur.
- La fonction `main` reçoit deux arguments :
  - un entier égal au nombre d'arguments reçus par le programme,
  - un tableau de chaînes de caractères avec les valeurs des arguments.

```c
int main (int argc, char *argv[])
{
  // le programme

  return 0;
}
```

- Comme on l'a vu il faut déclarer les fonctions avant de les utiliser.
- Les bibliothèques fournissent des fichiers *entêtes* (`.h` pour "header") qui déclarent les fonctions qu'elles définissent.
- On peut inclure ces entêtes avec la directive `#include`.
- Sa syntaxe est :
  - `#include "file.h"` quand il s'agit d'un fichier `file.h` dans votre projet,
  - `#include <file.h>` quand il s'agit d'un entête fournit par une bibliothèque installée sur le système, typiquement la bibliothèque standard de C.
- Exemple de programme complet :

```c
/* stdio.h est l'entête des fonctions d'entrées/sorties
   de la bibliothèque standard */
#include <stdio.h>

int main (int argc, char *argv[])
{
  printf("Hello, world!\n");

  return 0;
}
```

Entrées / sorties

- La bibliothèque standard de C définie entre les fonctions `printf` et `scanf`.
- Ces fonctions manipulent des chaînes qui contiennent des *formats* :
  - `%d` pour un entier,
  - `%x` pour un entier en hexadécimal,
  - `%c` pour un caractère,
  - `%s` pour une chaîne,
  - `%f` pour un double,
  - etc.
- Elles prennent donc une chaîne contenant des en premier argument et ensuite autant d'arguments qu'il y a de formats dans la chaîne.

- La fonction `printf` affiche la chaîne en ayant remplacé les formats par les valeurs des arguments correspondant :

```c
#include <stdio.h>

int main (int argc, char *argv[])
{
  int i;

  printf("J'ai %d argument%s :\n", argc, (argc > 1 ? "s" : ""));

  for (i = 0; i < argc; i++) {
    printf(" - %s\n", argv[i]);
  }

  return 0;
}
```

- La fonction `scanf` lit la chaîne et place les valeurs dans les arguments correspondants.

- La fonction `scanf` lit la chaîne et place les valeurs dans les arguments correspondants.
- Il faut donc lui passer des pointeurs vers les variables dans lesquelles on veut que les valeurs lues soient stockées.

```c
#include <stdio.h>

int main (int argc, char *argv[])
{
  char name[100];
  int age;

  printf("Nom : ");
  scanf("%s", name);

  printf("Age : ");
  scanf("%d", &age);

  printf("%s a %d ans\n", name, age);

  return 0;
}
```