Le langage C

François Trahay

Contents

Présentation du module

Objectifs du module:

- Maîtriser le langage C
- Savoir s'adresser au système d'exploitation depuis un programme

Modalités:

- Un peu de théorie
- Beaucoup de pratique

Contenu du module

Partie *Programmation*

- CI 1 – Le langage C
- CI 2 – Les types composés / qu'est-ce qu'une adresse ?
  - Exercice Hors-Présentiel
- CI 3 – Faire des programmes modulaires en C
- CI 4 – Les pointeurs
- CI 5 – Debugger un programme

Partie *Système*

- CI 6 – Les fichiers
- CI 7 – Les processus
- CI 8 – Appels système et Sémaphores
- CI 9 – Signaux

Evaluation

- CI 10 – Exercice de synthèse
- CF1 (sur papier) – questions sur l'exercice de synthèse

Déroulement d'une séance

Système de *classe inversée*. Pour chaque séance :

- **Avant** la séance
  - Etude de la partie cours de la séance à venir
- **Pendant** la séance:
  - Mini-évaluation de la partie cours (Kahoot!)
  - Explications sur les points mal compris
  - Travaux pratiques : expérimentations sur les concepts vus en cours

Attention ! Cela ne fonctionne que si vous travaillez sérieusement **avant** la séance.

Hypothèse: les étudiants suivant ce cours sont des adultes responsables.

Ressources disponibles

Pour vous aider, vous avez à votre disposition:

- Le poly contenant l'ensemble des transparents commentés
- Les transparents en version pdf

- La documentation des fonctions C standard (`man 2 <fonction>` ou `man 3 <fonction>`)
- Une équipe enseignante de choc !

C vs. Java

- langage de *bas niveau* vs. *haut niveau*
- En C, manipulation de la mémoire et de ressources proches du matériel
- *"Un grand pouvoir implique de grandes responsabilités"*[1]
- programmation impérative vs. programmation objet

[1] *B. Parker, Amazing Fantasy, 1962*

Mon premier programme en C

Fichier ***.c**

```c
/* hello_world.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
  printf("Hello World!\n");
  return EXIT_SUCCESS;
}
```

Compilation/execution:

```
$ gcc hello_world.c -o hello_world -Wall -Werror
$ ./hello_world
Hello World!
```

- Les `#include <stdio.h>` indiquent que le programme a besoin des outils `stdio`. Il s'agit donc d'un équivalent du `import package` de Java
- Pour afficher un message dans le terminal, on utilise la fonction `printf("message\n");`
- Le `return EXIT_SUCCESS;` à la fin du `main` permet de spécifier le code retour du programme (accessible depuis le shell en faisant `echo $?`). En cas d'erreur, on peut retourner `EXIT_FAILURE` à la place de `EXIT_SUCCESS`.

Déclaration de variable

Pour les types simples, déclaration identique à Java:

```
int var1;
int var2, var3, var4;
int var5 = 42;
```

Types disponibles: * pour les entiers: `int`, `short`, `long`, `long long` * pour les flottants: `float`, `double` * pour les caractères: `char`

Pour les entiers: possibilité de préfixer le type par `unsigned`. Les variables sont alors non-signées (ie. positives).

La taille d'une variable entière (ie. le nombre de bits/octets) dépend de l'implémentation. Le standard C ne spécifie que la taille minimum. Ainsi, un `int` doit faire au moins 16 bits, alors que la plupart des implémentations modernes utilisent 32 bits pour les `int`. Il convient donc de ne pas se reposer sur ces types lorsqu'on a besoin d'un nombre précis de bits/octets.

Pour cela, il est préférable d'utiliser les types fournis par `stdint.h`: `uint8_t` (8 bits), `uint16_t` (16 bits), `uint32_t` (32 bits), ou `uint64_t` (64 bits).

Comme en Java, les variables déclarées dans une fonction sont *locales* à la fonction (elles disparaissent donc dès la sortie de la fonction). Les variables déclarées en dehors d'une fonction sont *globales*: elles sont accessibles depuis n'importe quelle fonction.

Opérateurs et Expressions

La liste des opérateurs disponibles est à peu près la même qu'en Java:

- arithmétique : `+`, `-`, `*`, `/`, `\%`
- affectation : `=`, `+=`, `-=`, `*=`, `/=`, `\%=`
- incrémentation/décrémentation: `++`, `{-}`
- comparaison: `<`, `<=`, `>`, `>=`, `==`, `!=`
- logique: `!`, `\&\&`, `||`

Mais également:

- `sizeof n`: donne le nombre d'octets qui constitue une variable/un type `n`

Opérateurs bit à bit

Possibilité de travailler sur des *champs de bits*.

- Opération sur les bits d'une variable
- décalage: `<<`, `>>`
- OR : `|`, AND :`&`, XOR : `^`, NOT : `~`
- affectation: `<<=`, `>>=`, `|=`, `&=`, `^=`, `~=`

```c
/* bits.h */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int main(int argc, char** argv) {
  uint32_t v = 1;
  int i;

  /* l'operateur << decale vers la gauche */
  for(i=0; i<32; i++) {
    /* v << i  decale les bits de v de i places vers la gauche
     * c'est equivalent à calculer v*(2^i)
     */
    printf("v<<%d = %u\n", i, v<<i);
  }

  v = 5;
  /* v | 3 effectue un OU logique entre les bits de v et la representation binaire de 3
   * 101 | 11 = 111 (7)
   */
  printf("%u | %u = %u\n", v, 3, v|3);

  /* v & 3 effectue un ET logique entre les bits de v et la representation binaire de 3
   * 101 & 11 = 001 (1)
   */
  printf("%u & %u = %u\n", v, 3, v&3);

  /* v ^ 3 effectue un XOR logique entre les bits de v et la representation binaire de 3
   * 101 ^ 011 = 110 (6)
   */
  printf("%u ^ %u = %u\n", v, 3, v^3);

  /* ~v effectue un NON logique des bits de v
   * ~ 00...00101 = 11..11010 (4294967290)
   */
  printf("~%u = %u\n", v, ~v);

  return EXIT_SUCCESS;
}
```

Remarque

Lorsqu'on opère un décalage (avec `<<`) sur une valeur signée (par exemple, un `int`), le bit de signe n'est pas modifié par le décalage. Par exemple, si les bits

d'un `int a` sont à `1010 0000 0000 0000 0000 0000 0000 0000`, le résultat de `a >> 1` est `1001 0000 0000 0000 0000 0000 0000 0000`.

Structures algorithmiques

Comme en Java:

- `for(i=0; i<n; i++) { ... }`
- `while(cond) {... }`
- `do { ... } while(cond);`
- `if (cond) { ... } else { ... }`

Affichage / Lecture

- Pour afficher: `printf("%d exemple de %f format \n", v1, v2);`
- Pour lire: `scanf("%d-%f", &v1, &v2);`

```c
/* formats.c */
#include <stdio.h>

int main(int argc, char** argv) {
  int v;
  printf("Entrez la valeur de v:\n");
  scanf("%d", &v);
  printf("v = %d (en decimal)\n", v);
  printf("v = %u (en decimal non signe)\n", v);
  printf("v = %x (en hexadecimal)\n", v);
  printf("v = %o (en octal)\n", v);
  printf("v = %c (en ASCII)\n", v);

  double a;
  scanf("%lf", &a);
  printf("a = %f (en flottant)\n", a);
  printf("a = %lf (en flottant double precision)\n", a);
  printf("a = %e (en notation scientifique)\n", a);

  char *chaine = "Bonjour";
  printf("chaine = %s\n", chaine);
  printf("chaine = %p (adresse)\n", chaine);

  printf("On peut aussi afficher le caractère %%\n");
}
```

Fonctions

Déclaration:

```
type_retour nom_fonc(type_param1 param1, type_param2 param2) {
/* déclaration des variables locales */
/* instructions à exécuter */
}
```

En C, il est d'usage de nommer les fonctions en minuscule, en séparant les mots par `_`. Par exemple, l'équivalent en C de la fonction Java `calculerLeMinimum()` sera `calculer_le_minimum()`.