Plan

1 Introduction
- Langage de programmation
- Le langage C

Langage de programmation

Definition

Un langage de programmation est un système de notation permettant de décrire de manière simple une suite d'actions ordonnée qu'un ordinateur pourra exécuter.

Différents niveaux de langage:

- **langage machine**:
  codage binaire propre à chaque processeur
- **langage assembleur**:
  codage alphanumérique du langage machine
  $\hookrightarrow$ traduit en langage machine par un assembleur
- **langage haut niveau**:
  codage alphanumérique proche du langage naturel
  $\hookrightarrow$ traduit en langage assembleur par un compilateur

Langage assembleur

- Langage proche de l'architecture de la machine
- Loin de la vision algorithmique

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _main
        .align  4, 0x90
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp2:
        .cfi_def_cfa_offset 16
Ltmp3:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp4:
        .cfi_def_cfa_register rbp
        mov     eax, 0
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], 10
        pop     rbp
        ret
        .cfi_endproc

.subsections_via_symbols
```

Langage assembleur (et machine)

- Langages composés d'instructions bas-niveau :
  - chargement de valeur dans un registre
  - déplacement de valeur de registre à zone mémoire (et vice-versa),
  - calcul binaire sur registre (addition, comparaison...)
  - poursuite de l'exécution à une autre point du programme ;

  $\hookrightarrow$ Pas adapté à l'écriture simple de programmes.
- Langages propres à chaque processeur

  $\hookrightarrow$ Nécessité de reprogrammer pour exécuter la même tâche sur une autre architecture de machine : code non portable !

Langage haut niveau

- Langage adapté à la manière dont le programmeur voit l'exécution de son programme, proche d'une description algorithmique
- Différents paradigmes de programmation :
  - impératif/procédural : vision d'un ensemble structuré d'opérations modifiant l'état de la machine *: C, Pascal*
  - fonctionnel : vision d'un calcul mettant en œuvre un certain nombre de fonctions mathématiques *: Lisp, Caml*
  - objet : vision d'un ensemble de briques logicielles dotées d'un comportement et capable d'interagir *: C++, Java, OCaml*

Langage haut niveau

- La même tâche que précédemment mais écrit en C.

```c
int main()
{
    int a;
    a = 10;
    return 0;
}
```

- Un langage de haut niveau n'est pas exécutable directement par un ordinateur. Deux solutions possibles :
  - compilation : traduire en langage machine puis exécuter
  - interprétation : traduire et exécuter étape par étape

  ↪ Généralement un langage est soit compilé, soit interprété .
- Nécessité de disposer d'un compilateur/interpréteur du langage haut niveau adapté à sa machine.

Plan

1 Introduction
- Langage de programmation
- Le langage C

Historique

Le langage C

- est un langage purement impératif
- est créé en 1972 par D. Ritchie et K. Thompson pour développer UNIX
- normalisation ANSI (1989) et ISO (1990,1991,2011)

Caractéristiques importantes

- faiblement typé (conversion implicite des types)
- langage compilé
- arithmétique sur les adresses mémoires et sur les bits

Programmer en C

Au préalable, un travail algorithmique de spécification et conception du programme précisant les structures de données à mettre en œuvre doit être réalisé.

Trois étapes ordonnées:

1. écrire dans un fichier texte le code du programme en C
2. compiler le programme
3. exécuter le programme

Écrire un programme C

Il suffit d'écrire avec un éditeur de texte simple (sans formatage) quelques lignes de code.

exemple.c

```c
#include <stdio.h>

int main() {
   int a,b,c;
   scanf("%d %d", &a, &b);
   c = a/b;
   printf("%d + %d = %d\n", a, b, c);
   return 0;
}
```

Un exemple de compilation

Pour compiler l'exemple (exemple.c), on tape dans un terminal :

```
gcc -Wall exemple.c -o exemple
```

ce qui permet de créer un programme exécutable nommé exemple.

Syntaxe générale : `gcc -Wall source.c -o execu`

- gcc est le nom du compilateur C
- `-Wall` est une option classique qui demande d'afficher les *warning* lors de la compilation : bon programme ⇒ pas de warning
- `source.c` est le fichier contenant le programme en C.
- `-o execu` précise que le fichier produit en langage machine sera nommé `execu`

Notion de programme en C

Un programme est constitué de plusieurs fonctions (procédures) qui peuvent s'appeler les unes les autres dont une (unique) spécifique, nommée **`main`**, qui constitue le point d'entrée du programme : le début de l'exécution.

Point d'entrée d'un programme C :

```
1 int main() {
2   // debut du programme
3   ...
4   // fin du programme
5   return 0;
6 }
```

Les erreurs de programmation

- Erreur de syntaxe
  - non-respect des règles d'écriture du langage
  - elles sont détectées par le compilateur qui les décrit et souvent dit comment les corriger.
- Erreur de conception
  - le programme ne fait pas ce qu'on attend
  - non détectable par le compilateur $\Rightarrow$ faire des tests !
- Erreur à l'exécution
  - le programme s'interrompt anormalement lors de l'exécution
  - non détectable par le compilateur mais est révélateur d'une erreur d'écriture ou d'une erreur de conception $\Rightarrow$ déboguer !

Une erreur de syntaxe

syntax-error.c

```
1 int main() {
2   int a
3   return 0;
4 }
```

Compilation

```
gcc -Wall syntax-error.c -o syntax-error
```

```
syntax-error.c: In function 'main':
syntax-error.c:3:3: error: expected '=', ',', ';', 'asm'
 or '__attribute__' before 'return'
    3 |   return 0;
      |   ^~~~~~
```

il y a une erreur ligne 3 avant le mot clé return $\rightarrow$ il manque un ;

Une erreur à l'exécution : la division par 0 (due à erreur de conception)

exemple.c

```c
#include <stdio.h>

int main() {
  int a,b,c;
  scanf("%d %d", &a, &b);
  c = a/b;
  printf("%d / %d = %d\n", a, b, c);
  return 0;
}
```

Pas d'erreur de compilation, mais l'exécution avec $b = 0$ provoque l'erreur :

Exception en point flottant (core dumped)

```
$ gcc -Wall exemple.c -o exemple

$ ./exemple
12 5
12 / 5 = 2

$ ./exemple
12 0
Exception en point flottant (core dumped)
```