

# Programmation impérative

ENSIIE

Semestre 1 — 2020–21

# Organisation du cours

- ▶ Cours magistral : présentation des concepts, des structures de données et leurs outils
- ▶ TP : manipulation, mise en œuvre **TOUT** refaire
- ▶ TD : aspects théoriques, intérêt

## Évaluation :

- ▶ 1 projet  $\left(\frac{1}{3}\right)$
- ▶ contrôle continu  $\left(\frac{1}{3}\right)$ 
  - interrogations « surprises »
  - TP(s) noté(s)
- ▶ examen final  $\left(\frac{1}{3}\right)$

# Introduction

# Programmation

Exécutable : suite de 0 et 1 directement compréhensible par la machine

```
001001000100010011
110100010101001010
111110010101001101
```

Programme : abstraction d'un exécutable pour le rendre plus facile à comprendre et à écrire

```
int main() {
    printf("Bonjour !");
    return 0;
}
```

## Paradigme de programmation

Différents types d'abstractions possibles

⇒ différents paradigmes de programmation

- ▶ Assembleur : très proche de l'exécutable
- ▶ Impérative : modification de la mémoire, effets de bord
- ▶ Fonctionnelle : évaluation de fonctions mathématiques, persistance des données
- ▶ Orientée Objet : structures de données et traitements associés regroupés au sein d'un « objet »
- ▶ ...

Pas de paradigme meilleur que les autres

Plus adapté dans un certain contexte

Problème  $\rightsquigarrow$  bonne structure  $\rightsquigarrow$  langage adapté

# Compilation

Traduit le programme (abstrait) en exécutable (code machine)

```
int main() {  
    return 0; }  
}
```

↔

```
001001000100010011  
110100010101001010  
111110010101001101
```

Autre possibilité : interprétation

- ▶ un exécutable (interpréteur) lit le programme et effectue les opérations attendues

# Avertissement

Ceci N'est PAS un cours de C

Néanmoins on ne peut pas apprendre la programmation sans la pratiquer

On utilisera donc le C pour illustrer le cours

# La référence du C

*The C Programming Language*

Brian W. Kernighan et Dennis M. Ritchie

2<sup>e</sup> édition



Défini la norme ANSI/ISO pour le langage C

Les compilateurs C sont censés respecter cette norme

Des traductions existent en français

# Concepts de base

# Programmation impérative

Principe de base : modification de l'état mémoire

Utilisation d'effets de bord

Deux mêmes appels à des endroits différents peuvent avoir des comportements différents

État mémoire abstrait

# Comment décrire un programme impérative ?

- ▶ État mémoire attendu avant l'exécution du programme  
= **précondition** @requires
- ▶ Éléments de la mémoire potentiellement modifié par le programme  
= **assignements** @assigns
- ▶ État mémoire résultant de l'exécution du programme  
= **postcondition** @ensures

# Abstraction de la mémoire

```
000100101010101010101001100101000011000101...
```

Adresse : position sur le ruban, regroupé par octets

0	1	2	3	4	5
00010010	11010100	11010100	11001010	00011000	101...

Variable : emplacement (case) dans la mémoire

	x		y	
00010010	00101010	01010010	0100001100010	1011...

## Type d'une variable

- ▶ taille de la case
- ▶ comment le contenu est interprété

000100101010101010	<sup>x</sup> 00101010	0101000011000101...
--------------------	--------------------------	---------------------

char x :

- ▶ x tient sur 1 octet (8 bits)
- ▶ x est interprété comme un entier (42 en l'occurrence, ou encore '\*')

## Types de base en C

- ▶ Langage peu typé
- ▶ Conversion implicites
- ▶ La plupart : dépendants du compilateur/de la machine

Faire **très** attention

Trois familles de base :

- ▶ entiers
- ▶ flottants
- ▶ pointeurs

# Types entiers

Arithmétique modulo  $2^{\text{nb. bits}}$

Plusieurs tailles :

- ▶ char : 8 bits
- ▶ int : taille dépendant de la machine (32 bits, 64 bits)
- ▶ long int, short int

Signé ou non :

- ▶ unsigned char [0, 255]
- ▶ signed char [-128, 127]

## Types flottants

Nombres en virgule flottante ( $\neq$  réels)

- ▶ float, double

- ▶ 

$s$	$e$	$m$
1	10101001	1001001001001001001101

$$(-1)^s \times 1, m \times 2^{e-127}$$

Exemple :  $-0,25 = -1 \times 1,0000 \times 2^{125-127}$

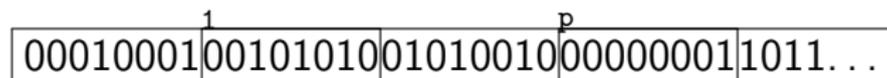
- ▶ 

$s$	$e$	$m$
1	01111101	000000000000000000000000



# Pointeurs

- ▶ le contenu est l'adresse d'une case mémoire
- ▶ exemple : `char* p`  
le contenu de `p` est l'adresse d'une case dont le contenu doit être interprété comme un `char`



- ▶ `p` pointe vers l'adresse 1
- ▶ `*p` est le contenu de la case pointée par `p`, interprété comme un `char`, i.e. 42

# Programme

Contenu d'un programme :

- ▶ déclarations de variables avec leur type
- ▶ bloc (liste) d'instructions

Instructions :

- ▶ contrôle et modifications
- ▶ utilisent des expressions

Expressions :

- ▶ valeurs

## Fonction principale

En C, on peut définir plusieurs fonctions  $\simeq$  sous-programmes  
Il faut nécessairement définir la fonction `main`

- ▶ point de départ du programme
- ▶ peut faire appel aux autres fonctions

Squelette :

```
int main() {  
    /* declarations de variables */  
    ...  
    /* instructions */  
    ...  
    return 0;  
}
```



## Instructions : affectation

$x = expr;$

$expr$  est évaluée, et sa valeur est mise dans la case mémoire identifiée par  $x$ .

- ▶ @requires : un état mémoire quelconque
- ▶ @ensures :  $x$  contient la valeur de  $expr$
- ▶ @assigns :  $x$ , mais aussi tout ce qui est modifié par l'évaluation de  $expr$

## Appel de procédure

```
nom_proc(expr1, . . . , exprn);
```

On fait appel à la procédure appelée *nom\_proc* en lui passant en paramètre les valeurs de expressions *expr*<sub>1</sub> . . . *expr*<sub>*n*</sub>

Entrées/sorties :

```
printf(format, arg1, arg2, . . .);
```

```
scanf(format, arg1, arg2, . . .);
```

# Bloc

```
{  
  instruction1  
  ...  
  instructionn  
}
```

Regroupe une séquence d'instructions  
Les exécute l'une après l'autre

# Conditionnelle

```
if (cond)  
    instruction_if  
suite...
```

```
if (cond)  
    instruction_if  
else  
    instruction_else  
suite...
```

► *cond* expression de type entier

► *val* . *valeur* ≠ 0

# Branchement

```
switch (expr) {  
    case constante1: instruction1 break;  
    case constante2: instruction2 break;  
    ...  
    default: instruction break;  
}
```

# Boucles

Faire quelque chose tant qu'une condition est vérifiée

- ▶ bornée
- ▶ non bornée

En C, sucre syntaxique  $\rightsquigarrow$  ici forte restriction

# Boucle bornée

Nombre d'itérations connu dès le départ

```
for (init ; condition ; mise à jour)  
    instruction
```

Dans ce cours :

- ▶ **interdit** de toucher à l'indice dans le bloc
- ▶  $\Rightarrow$  terminaison

## Boucle non bornée

Nombre d'itérations inconnu dès le départ

```
while (condition)  
    instruction
```

```
do  
    instruction  
    /* effectuée au moins une fois */  
while (condition)
```

Donner un argument de terminaison