

Programmation en C

Perfectionnement à la programmation en C

Samuele Giraudo

`samuele.giraudo@univ-eiffel.fr`

`https://igm.univ-mlv.fr/~giraudo`

Université Gustave Eiffel

LIGM, bureau 4B162

2021–2022

Introduction

L'objectif de ce module est d'approfondir les concepts de base et d'approcher certains concepts plus avancés de la programmation en C.

Celui-ci est organisé en trois axes.

1. **Axe 1 : écrire un projet maintenable.**

Bonnes habitudes de programmation, documentation, pré-assertions, gestion des erreurs, programmation modulaire, compilation.

2. **Axe 2 : comprendre les mécanismes de base.**

Pointeurs, allocation dynamique, types, types structurés, entrées/sorties.

3. **Axe 3 : utiliser quelques techniques avancées.**

Opérateurs bit à bit, mémoïsation, génération aléatoire, pointeurs de fonction, généricité.

Contenu du cours

Axe 1.

- 1. Bases
- 2. Habitudes
- 3. Modules
- 4. Compilation

Axe 2.

- 5. Allocation dynamique
- 6. Entrées et sorties
- 7. Types
- 8. Types structurés

Axe 3.

- 9. Opérateurs
- 10. Pointeurs de fonction
- 11. Génération aléatoire
- 12. Mémoïsation

Pré-requis

Ce cours demande les pré-requis suivants :

- des bases en **programmation générale** (notion de programme, d'instruction, de compilation);
- des bases en **programmation impérative** (notion de variable, de structures de contrôle, de fonction);
- des bases en **algorithmique** (manipulation de structures de données élémentaires comme les tableaux, les chaînes de caractères, les listes);
- des bases en **programmation en C** (écriture de fonctions, manipulation de tableaux, de chaînes de caractères, connaissance des types numériques, des types structurés, notion d'allocation dynamique, de pointeur).

Quelques dates

Le langage C est apparu en 1972 dans les laboratoires Bell. Il a été créé par **D. Ritchie** et **K. Thompson** au même moment que l'apparition des systèmes UNIX.

Il est influencé par le langage B qui, contrairement au C, ne possédait pas de système de type.

Quelques dates sur l'évolution de la spécification du langage :

- 1978 : première description complète du langage (traditionnel);
- 1989 : **publication de la norme ANSI C** (ou C89);
- 1995 : évolution du langage C94 et C95;
- 1999 : évolution du langage C99;
- 2011 : nouvelle version du standard C11.

Quelques caractéristiques

Le C est un **langage de bas niveau** : il offre la possibilité de se placer « proche » de la machine. Il permet en effet de manipuler finement des données en mémoire, des adresses et des suites de bits.

Ce langage a été initialement pensé pour la **conception de systèmes d'exploitation**.

Il est cependant suffisamment expressif pour s'adapter à un large éventail d'utilisations.

Il se situe à la croisée des chemins du monde des langages de programmation : beaucoup de langages modernes sont traduits en C pour être compilés.

L'un de ses compilateurs, gcc, fruit de nombreuses optimisations, fait que le C est un langage d'une **efficacité extrême**.

Axe 1 : écrire un projet maintenable

1. Bases
2. Habitudes
3. Modules
4. Compilation

Plan

1. Bases

Plan

1. Bases

1.1 Généralités

1.2 Expressions et instructions

1.3 Constructions syntaxiques

1.4 Variables

1.5 Fonctions et pile

1.6 Commandes préprocesseur

La notion de programme

Un programme en C est avant tout un texte (contenu dans un **fichier**) qui suit certaines règles (dites de **syntaxe**).

C'est une collection

- de déclarations de fonctions ;
- de définitions de fonctions ;
- de déclarations de types ;
- de déclarations de variables globales ;
- de commandes pré-processeur.

Tout programme possède une **fonction principale** nommée `main`. C'est par elle que commence l'exécution du programme. On appelle ceci le point d'entrée du programme.

Compiler un programme

Compiler un programme signifie **traduire** le fichier le contenant en un langage compréhensible et exécutable par la machine cible.

On compile un fichier `Prog.c` par la commande

```
gcc -o NOM Prog.c
```

Ceci produit un exécutable nommé `NOM`.

On compilera obligatoirement avec les **options** `-ansi`, `-pedantic` et `-Wall` au moyen de la commande

```
gcc -o NOM -ansi -pedantic -Wall Prog.c
```

- `-ansi -pedantic` : empêche un programme non compatible avec la norme ANSI C de compiler ;
- `-Wall` : active tous les messages d'avertissement.

Plan

1. Bases

- 1.1 Généralités
- 1.2 Expressions et instructions**
- 1.3 Constructions syntaxiques
- 1.4 Variables
- 1.5 Fonctions et pile
- 1.6 Commandes préprocesseur

Expressions

Une expression est définie récursivement comme étant soit

1. une constante;
2. une variable;
3. une combinaison d'expressions et d'opérateurs;
4. un appel de fonction qui renvoie une valeur, c.-à-d., de type de retour autre que `void`.

Une expression n'est donc rien d'autre qu'un **assemblage de symboles** qui vérifie des règles **syntaxiques** et **sémantiques**.

Toute expression possède une **valeur** et un **type**. Le processus qui consiste à déterminer la valeur d'une expression se nomme l'évaluation.

Expressions

– Exemples –

Les entités suivantes sont des expressions :

■ 0

Valeur : 0, type : int

■ 'a'

Valeur : 97, type : int

■ "abc"

Valeur : "abc", type : char *

■ x

Valeur : x, type : le type de la variable x

■ a == 2

Valeur : 0 ou 1, type : int

■ b = 3

Valeur : 3, type : int

■ f(5)

Valeur : la valeur renvoyée par f(5), type : le type de retour de f

■ (a == 0) && (b >= 3)

Valeur : 0 ou 1, type : int

■ a > 'a' ? a : -a

Valeur : a ou -a, type : le type de a

■ 1. + 7 * 2

Valeur : 15., type : float

Instructions

Une instruction est définie récursivement comme étant soit :

1. une expression `E`; terminée par un point-virgule;
2. un bloc `{I1 I2 ...Ik}` où `I1`, `I2`, ..., `Ik` sont des instructions;
3. une conditionnelle `if (E) I1 else I2`, où `E` est une expression et `I1` et `I2` sont des instructions;
4. toute autre construction similaire (`switch`, `while`, `do while`, `for`);
5. une déclaration `T x`; de variable, où `T` est un type et `x` est un identificateur.

À la différence des expressions, c'est souvent l'**effet** d'une instruction qui est sa raison d'être (et non plus uniquement sa valeur ou son type).

En effet, une instruction peut p.ex. afficher un élément, allouer une zone mémoire, lire dans un fichier, *etc.* La valeur de l'expression sous-jacente à l'instruction est d'importance moindre dans certains cas.

Instructions

– Exemples –

Les entités suivantes sont des instructions :

- `;` (instruction vide)

- `1;`

- `a = b;`

- `while (1) a += 1;`

- `printf("abc\n");`

- `a++;`

- `malloc(64);`

- `int a;`

- `int tab[64];`

- `tab[3] = 9;`

- `tab[3];`

- `return 31;`

Expressions à effet secondaire

Une **expression** est à effet secondaire (ou encore, improprement à « effet de bord ») si son évaluation modifie la mémoire.

– Exemples –

■ `0,`

■ `'c',`

■ `(a + 21) << 3,`

■ `tab[8],`

sont des expressions qui ne sont pas à effet secondaire.

– Exemples –

En revanche,

■ `a = 31,`

■ `printf("abc\n"),`

■ `char c,`

■ `malloc(64),`

sont des expressions à effet secondaire (noter que la déclaration de variable n'est pas une expression mais une instruction).

Expressions à effet secondaire

En règle générale, ce sont les éléments suivants dans les expressions qui produisent des effets secondaires :

- les affectations ;
- les allocations de mémoire ;
- la sollicitation au système de fichiers.

En revanche, les éléments suivants dans les expressions ne produisent pas d'effet secondaire :

- lectures de constantes et de variables ;
- calculs arithmétiques, logiques ou bit à bit ;
- comparaisons.

1. Bases

- 1.1 Généralités
- 1.2 Expressions et instructions
- 1.3 Constructions syntaxiques**
- 1.4 Variables
- 1.5 Fonctions et pile
- 1.6 Commandes préprocesseur

Blocs

Un bloc est une suite d'instructions délimitée par des accolades. Il est constitué d'une partie consacrée à la déclaration de variables et d'une partie consacrée aux instructions.

```
{  
    D  
    I  
}
```

Ici, **D** est une section de **déclarations** et **I** est une section d'**instructions**.

Les parties **D** et/ou **I** peuvent être vides.

Sachant qu'un bloc est une suite d'instructions, il est possible de placer un bloc dans la section d'instructions d'un bloc et ainsi d'**imbriquer** plusieurs blocs.

Blocs

– Exemple –

```
int main() {  
    int b;  
    scanf("%d", &b);  
    {  
        int a;  
        a = 1 + b;  
    }  
}
```

La partie grise est un bloc au sein d'une fonction `main`.

Il y a des règles concernant la visibilité des variables (que nous verrons plus loin).

– Exemple –

```
{  
    int a;  
    int b;  
    scanf(" %d", &a);  
    {  
        printf("%d\n", a);  
    }  
    scanf(" %d", &b);  
}
```

Ceci est un bloc constitué d'une section de déclarations et d'une section d'instructions.

Cette dernière partie contient un bloc (en gris) dont la section de déclarations est vide.

Opérateur de test if

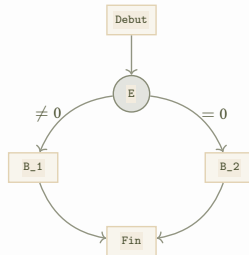
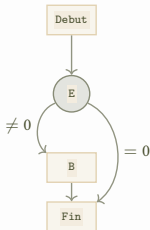
L'opérateur de test se décline en deux versions :

```
if (E)  
  B
```

```
if (E)  
  B_1  
else  
  B_2
```

Ici, **E** est une expression booléenne : elle est considérée comme **fausse** si elle s'évalue en zéro et comme **vraie** sinon. De plus, **B**, **B_1** et **B_2** sont des blocs d'instructions.

Diagrammes d'exécution :



Instruction de branchement `switch`

L'instruction de branchement admet la syntaxe

```
switch (E) {  
    case E_1 : I_1  
    case E_2 : I_2  
    ...  
    case E_N : I_N  
    default : I_D  
}
```

Ici, `E`, `E_1`, `E_2`, ..., `E_N` sont des expressions qui s'évaluent en des entiers.

Les `I_1`, `I_2`, ..., `I_N` et `I_D` sont des suites d'instructions.

L'exécution de cette instruction se passe ainsi. Soit `i` le plus petit entier (s'il existe) tel que les évaluations de `E` et `E_i` sont égales. Les instructions `I_i`, ..., `I_N` ainsi que `I_D` sont exécutées.

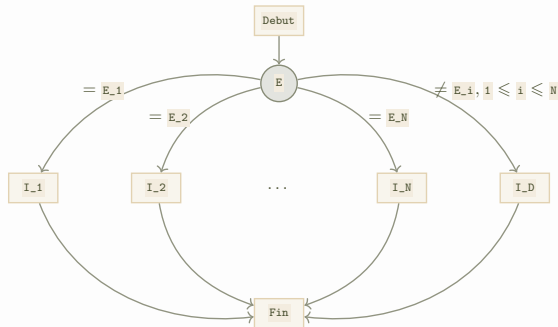
La ligne `default : I_D` est facultative. Si elle est présente, `I_D` est toujours exécutée.

Instruction de branchement `switch`

On s'impose de terminer chaque `I_j` et `I_D` par le mot-clé `break`.

Ceci permet de n'exécuter que le `I_i` tel que les évaluations de `E` et `E_i` sont égales. Dans ce cas, `I_D` n'est exécuté que si aucun des `E_i` ne l'a été.

On obtient ainsi le diagramme d'exécution



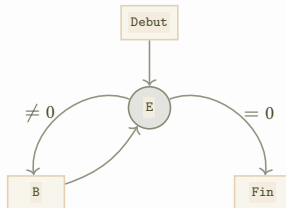
Instruction itérative `while`

L'instruction itérative `while` admet la syntaxe

```
while (E)  
  B
```

Ici, `E` est une expression booléenne et `B` est un bloc d'instructions.

Diagramme d'exécution :



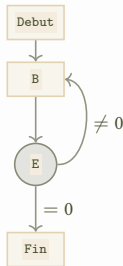
Instruction itérative `do while`

L'instruction itérative `do while` admet la syntaxe

```
do  
    B  
while (E);
```

Ici, `E` est une expression booléenne et `B` est un bloc d'instructions.

Diagramme d'exécution :



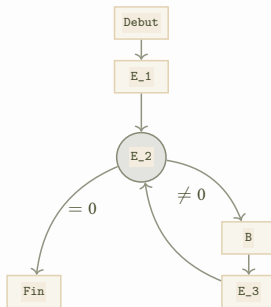
Instruction itérative `for`

L'instruction itérative `for` admet la syntaxe

```
for (E_1 ; E_2 ; E_3)  
  B
```

Ici, `E_2` est une expression booléenne (le test), `E_1` (l'initialisation) et `E_3` (l'incrément) sont des expressions et `B` est un bloc d'instructions.

Diagramme d'exécution :



Instruction itérative `for`

– Exemple –

```
for (i = 0 ; i < 3 ; ++i) {  
    printf("a");  
}
```

Affiche trois occurrences du caractère `'a'`.

– Exemple –

```
for ( ; *str != '\0'; str++) {  
    putchar(*str);  
}
```

Affiche la chaîne de caractères `str`. Il n'y a pas d'initialisation. L'incrémentacion fait pointer `str` sur le prochain caractère.

– Exemple –

```
for (x = lst ; x != NULL ; x = x->suiv) {  
    afficher(x->elem);  
}
```

Parcourt une liste simplement chaînée et l'affiche. La variable `x` est un pointeur sur la cellule courante.

– Exemple –

```
for ( ; *str != '\0'; putchar(*str++)) {  
    /* Rien. */  
}
```

Affiche la chaîne de caractères `str`. Il n'y a pas d'initialisation. L'affichage est réalisé comme effet secondaire de l'incrémentacion.

Instructions de court-circuit

Les instructions de court-circuit sont `break` et `continue`.

L'instruction `break` permet à l'exécution de sortir d'une structure `switch`, `while`, `do while` ou `for`.
L'exécution continue alors aux instructions qui suivent cette structure.

L'instruction `continue` permet à l'exécution de sauter à la fin du bloc `B` d'une structure `while`, `do while` ou `for`. L'exécution continue alors à l'évaluation de l'expression test.

Instructions de court-circuit

Dans une boucle `for`, l'instruction `continue` fait que l'expression d'incrément est tout de même évaluée.

– Exemple –

```
for (i = 1 ; i <= 7 ; ++i) {  
    if (i == 4)  
        continue;  
    printf("%d ", i);  
}
```

L'exécution de ces instructions produit sept tours de boucle et l'affichage `1 2 3 5 6 7`. Lorsque `i == 4`, le `continue` relance l'exécution à l'incrément du `for`.

Ce n'est pas le cas pour le `break`.

– Exemple –

```
for (i = 1 ; i <= 7 ; ++i) {  
    if (i == 4)  
        break;  
    printf("%d ", i);  
}
```

L'exécution de ces instructions produit quatre tours de boucle et l'affichage `1 2 3`. Lorsque `i == 4`, le `break` fait sortir de la boucle et `i` vaut `4`.

Plan

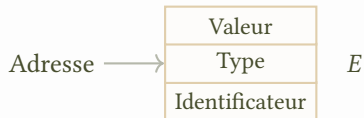
1. Bases

- 1.1 Généralités
- 1.2 Expressions et instructions
- 1.3 Constructions syntaxiques
- 1.4 Variables**
- 1.5 Fonctions et pile
- 1.6 Commandes préprocesseur

Variables

Une variable est une entité constituée des cinq éléments suivants :

1. un identificateur ;
2. un type ;
3. une valeur ;
4. une adresse ;
5. une portée lexicale.



Intuitivement, c'est une boîte qui peut contenir un objet (valeur) et qui dispose d'un nom (identificateur).

Une boîte ne peut contenir que des objets d'une certaine sorte (type).

Elle se situe de plus à un endroit bien précis dans la mémoire (adresse) et elle n'est visible qu'à partir de certains endroits du code (portée lexicale).

Identificateurs de variable

L'identificateur d'une variable est un mot commençant par une lettre ou bien '_', suivi par un nombre arbitraire de lettres, chiffres ou '_'.

De plus, aucun identificateur ne peut-être un mot réservé du langage. En voici la liste complète :

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

L'identificateur d'une variable est **attribué à sa déclaration**.