

CM3 : Introduction au C

ISia2A3 - Algorithmique et programmation structurée

Thibault Liétard

2026-2027

Introduction

Invention

- Créé en 1972 pour développer UNIX
- Évolution du langage B
- influencé par FORTRAN et ALGOL
- A influencé C++, Java, Js,...

Utilisation

- langage de bas niveau
- contrôle de la mémoire
- efficacité d'exécution

Signification

Langage qui ne fournit que peu d'abstraction par rapport au jeu d'instructions du processeur de la machine, par opposition aux langages de haut niveau, qui permettent de créer un programme sans tenir compte des caractéristiques particulières (registres, etc) de l'ordinateur censé exécuter le programme.

Implications

- Chaque instruction est conçue pour être compilée en un nombre d'instructions machine prévisible
- Les types correspondent aux types processeur
- Usage intensif de l'adressage mémoire

Dans la suite du cours, on notera :

- `<exp>` : une expression quelconque
- `[<exp>]` : une expression optionnelle

Bases du langage

Identificateur

C'est le "nom" de la variable, d'une fonction, d'un type,...

- Commence par une lettre
- Peut contenir n'importe quel caractère alphanumérique ou '_'
- Sensible à la casse

Identification

Par convention, un identificateur de variable est en minuscules.

Principe

Une variable est un emplacement en mémoire qui possède :

- Un nom, pour l'identifier dans la table d'adressage
- Un type, qui définit sa taille en mémoire et les opérations applicables

Déclaration

`<type> <identificateur>[=valeur] [,<identificateur>[=valeur]]`

Structure générale

```
1  int main(){  
2      //déclarations  
3  
4      //instructions  
5  
6      return 0;  
7  }
```

Entiers

- `int i = 0, j;`
- codé sur 4 octets
- dans $[-2^{31}; 2^{31} - 1]$
- extension sur 8 octets : `long`

Réels

- `float x = 3.14;`
- codé sur 4 octets
- extension sur 8 octets : `double`

Booléen

- n'existe pas en C, représenté avec des entiers
- 0 : faux
- $\neq 0$: vrai
- peut être simulé avec *stdbool.h*
- Opérateurs :
 - ET : &&
 - OU : ||
 - NON : !
- Séquentiel : la deuxième opérande du ET et du OU n'est évaluée que si nécessaire

Caractères

- `char c = 'a';`
- Entre quotes : `'_'`
- codé sur un octet (table ASCII étendue)
- Utilisation du symbole `\` pour :
 - Les caractères non-affichables : `\n` (nouvelle ligne), `\t` (tabulation), `\b` (retour en arrière avec effacement),...
 - L'échappement de certains caractères : `\"` pour les guillemets, `\\` pour le backslash,...
- Opérateurs identiques aux entiers : `'a'+1⇒'b'`, `'c'<'d'`,...

Chaîne de caractères

- Un tableau de caractères

Syntaxe

`<var> = <expr>;`

Détails

- la variable `<var>` doit être déjà déclarée
- l'expression `<expr>` peut être n'importe quelle expression du type de `<var>`
- elle est évaluée avant que l'affectation soit faite
- doit impérativement se terminer par un `';`.

Trois utilisations du '='

a==b

comparaison entre a et b ; expression booléenne

a = b;

affectation de b à a

a = b

absence de ';' = expression; le résultat de l'évaluation de cette expression est la valeur de la partie droite

Idem dans l'expression *if* ($a=b$)...

Expression hétérogènes

- Exemple : $5*1.0/2$
- Relation d'ordre des types : $\text{char} < \text{int} < \text{float}$

Cast

- Exemple : $(\text{float})\ 5$
- force la conversion dans un type donné

Format

Une chaîne de caractère contenant des codes formats dans l'ordre souhaité : %d (int), %f (float), %c (char), ...

Écriture

`printf(<format>, <expr1>, <expr2>,...);`

- chaque expression correspond à un format
- Exemple : `printf("%d %d", a, b);`

Lecture

`scanf(<format>, &var1, &var2,...);`

- Nécessite des adresses sur des variables existantes
- Exemple : `scanf("%d", &a);`

Exemple

```
1 int main()
2 {
3     int a, b;
4     printf("Saisissez deux entiers : ");
5     scanf("%d%d", &a, &b);
6     printf("Somme de %d et %d = %d\n",a,b,a+b);
7     return 0;
8 }
```

Structures de contrôle

Principe

Permet de faire une instruction simple avec une séquence d'instructions

Propriété

- Entre accolades { }
- Surtout utilisé dans les structures de contrôle (*if, while, for,...*)
- utilisable aussi pour structurer les séquences d'instructions en sous-séquences
- variable déclarée dans un bloc : portée uniquement dans ce bloc

Conditionnel

```
1 int main()
2 {
3     int a, b;
4     printf("Saisissez deux entiers : ");
5     scanf("%d%d", &a, &b);
6     if (a<b){
7         printf("%d est le maximum", a);
8     }
9     else{
10        printf("%d est le maximum", b);
11    }
12
13    return 0;
14 }
```

Boucle TantQue

```
1  int main()
2  {
3      int a, b;
4      printf("Saisissez deux entiers : ");
5      scanf("%d%d", &a, &b);
6
7      while(a<b){
8          a++;
9          printf("a vaut maintenant : %d", a);
10     }
11
12     return 0;
13 }
```

```
1  int main()
2  {
3      int a, b;
4      printf("Saisissez deux entiers : ");
5      scanf("%d%d", &a, &b);
6
7      for(int i=0; i<b; i++){
8          printf("%d\n", a+i);
9      }
10
11     return 0;
12 }
```

Boucle Faire-TantQue

```
1  int main()
2  {
3      int a, b, i=0;
4      printf("Saisissez deux entiers : ");
5      scanf("%d%d", &a, &b);
6
7      do{
8          i++;
9          printf("%d\n", a+i);
10     } while(i<b);
11
12     return 0;
13 }
```

Compilation et exécutable

Principe

Programme qui transforme un code source en un code objet

ligne de commande

gcc -o exec exec.c -Wall

Principes

Archivage de fichiers objets (fichiers .o) contenant des définitions de fonctions standards

Exemples

- libc
 - Fonctions d'entrées sorties (scanf, printf, ...)
 - Fonctions de traitement de caractères et chaînes de caractères
 - Fonctions de gestion mémoire
- libm
 - Bibliothèque de fonctions mathématiques
 - sin, cos, tan, sqrt, exp, log,...

Fichier header .h

À chaque bibliothèque correspond un ou plusieurs fichiers d'en-tête (qu'on appelle fichier *header*) contenant les déclarations de constantes, fonctions, etc...

Bibliothèques courantes

- *stdio.h* (entrées-sorties)
- *string.h* (traitement des chaînes de caractères)
- *stdbool.h* (simulation des booléen)
- *malloc.h* (gestion mémoire)
- *math.h* (pour libm)

Inclure un .h

Syntaxe

`#include <fichier.h>`

Exemple

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a, b;
6      printf("Saisissez deux entiers : ");
7      scanf("%d%d", &a, &b);
8      printf("Somme de %d et %d = %d\n", a, b, a+b);
9      return 0;
10 }
```

Créer son propre .h

Principe

Créer un fichier .h contenant uniquement les déclarations des constantes et des fonctions, puis créer un .c contenant les traitements.

Exemple

calculs.h

```
/* Fonction x puissance n */  
double puissance(double x, int  
n);  
  
/* Fonction factorielle n */  
double factorielle(int n);  
  
/* Fonction x puissance n divise  
par factorielle n */  
double puisFact(double x, int n);
```

calculs.c

```
/* Fonction x puissance n */  
double puissance(double x, int n){  
    double P = 1;  
    int i;  
  
    for (i=1; i<=n; i++) P = P * x;  
    return P;  
}  
...
```

Le préprocesseur traite les directives de précompilation (ie les lignes commençant par #)

#include <fichier.h>

Inclusion d'une librairie; remplacé dans le source C par le contenu du fichier lors de la précompilation

#include "fichier.h"

Inclusion d'un fichier utilisateur; remplacé dans le source C par le contenu du fichier lors de la précompilation

#define NOM valeur

Définition d'une constante. Par convention, les noms de constantes sont en majuscule.

Commande

`gcc -c main.c fichier.c`

Remarques

- Gère l'expansion du code (remplacement des constantes, association des .h,...)
- Création d'un fichier binaire .o
- Le compilateur doit connaître tous les chemins nécessaires pour trouver les fichiers d'entête

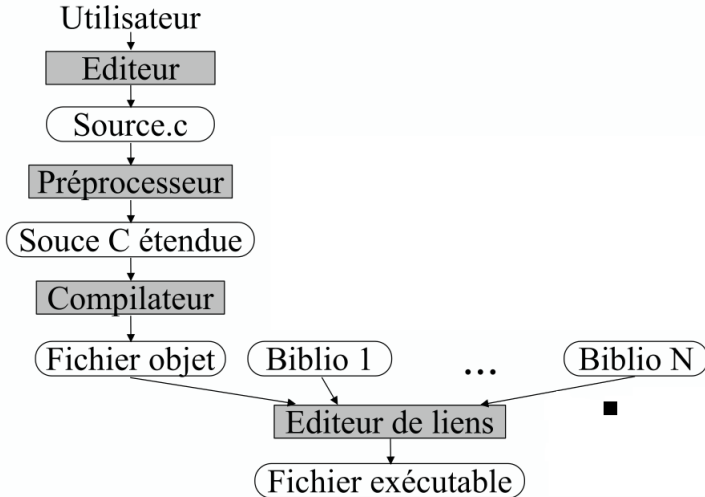
Commande

`gcc -o exec main.o fichier.o`

Remarques

- Production d'un exécutable à partir d'un ensemble de fichiers objet par agrégation de chaque fichier objet avec les éventuels fichiers binaires des bibliothèques qui sont utilisées
- L'option `-o` permet de donner le nom que l'on veut au fichier exécutable
- Par défaut : `a.out`

Schéma global



Tout en même temps

On peut réaliser toutes les étapes précédentes en une seule fois :

```
gcc -o exec main.c fichier.c
```

-Wall

man : This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning)

```
gcc -o exec main.c fichier.c -Wall
```