Méthodologie de conception et de programmation

Cours 2

N. de Rugy-Altherre - Vincent Demange

Compilation

Compilation

Le langage C est *compilé* :

- Le *code source* C est écrit dans un fichier texte d'extension `.c`.
- Il est traduit en *code objet* dans un fichier d'extension `.o`.
- Plusieurs codes objets sont ensuite assemblés en un fichier directement *exécutable*.

La compilation d'un code source C se déroule en 4 étapes :

1. **Le préprocesseur :** transforme le source C en source intermédiaire en traitant des directives dédiées, p. ex. remplacement des macros (**`#define`**), inclusion des fichiers (**`#include`**), compilation conditionnelle (**`#ifdef`**).
2. **La compilation :** traduit le code intermédiaire en code assembleur, complété d'optimisations fichier `.s`.
3. **L'assemblage :** transforme le code assembleur en fichier objet écrit en langage machine i.e. instructions binaires pour le processeur.
4. **L'édition des liens :** produit un fichier exécutable en réunissant fichiers objets et bibliothèques nécessaires.

Compilation

Le compilateur C sous UNIX s'appelle `cc`. On utilisera de préférence celui du projet GNU : `gcc`.

Compilation sous Linux

```
gcc [options] fichier.c [options]
```

Options les plus importantes (cf. `$ man gcc`) :

- `-o fichier`
  spécifie le nom du fichier produit (`a.out` par défaut) ;
- `-v`
  affichage *verbeux*, détail des commandes exécutées ;
- `-Wall`
  affiche tous les messages d'avertissement (*all warnings*) ;
- `-E` applique préprocesseur ;
- `-S` génère code assembleur.

Compilation

Utilisation standard UNIX

```
$ gcc -Wall fichier.c -o prog
$ ./prog
```

`./prog` exécute le fichier exécutable généré.

Utilisation Windows 10-11

- Dans la barre de recherche, saisir `cmd`
- Dans le terminal qui s'ouvre, taper `bash`
- Faire comme sous UNIX (plus ou moins)

IDE

Définition

Un IDE (integrated developpment environment) est un environnement de développement intégré : un logiciel proposant un éditeur de text, un compilateur, un débuggeur voir même un gestionnaire de mémoire et une aide à la programmation

Par exemple : Visual Studio, Eclipse.
Ces environnements sont des grandes aides aux développeurs et vous en verrez en L2 (IntelliJ par exemple).
Néanmoins en méthodo I ils seront interdit car :

- il faut apprendre les détails qu'un IDE gère
- La prise en main d'un IDE est longue...

Introduction : Python et références

Le code :

```
def modiInt(i):
    i = i + 1

def modiTab(T):
    T[0] = T[0] + 1

n = 0
modiInt(n)
print("n =",n)
T = [0]
modiTab(T)
print("T =", T)
```

Produit l'affichage :

Introduction : Python et références

Le code :

```
def modiInt(i):
    i = i + 1

def modiTab(T):
    T[0] = T[0] + 1

n = 0
modiInt(n)
print("n =",n)
T = [0]
modiTab(T)
print("T =", T)
```

Produit l'affichage :

```
n = 0
```

Introduction : Python et références

Le code :

```
def modiInt(i):
    i = i + 1

def modiTab(T):
    T[0] = T[0] + 1

n = 0
modiInt(n)
print("n =",n)
T = [0]
modiTab(T)
print("T =", T)
```

Produit l'affichage :

```
n = 0
T = [1]
```

Introduction : Python et références

Le code :

```
def modiInt(i):
    i = i + 1

def modiTab(T):
    T[0] = T[0] + 1

n = 0
modiInt(n)
print("n =",n)
T = [0]
modiTab(T)
print("T =", T)
```

Produit l'affichage :

```
n = 0
T = [1]
```

n est passé à `modiInt` par *valeur*.

T est passé à `modiTab` par *référence*.

Introduction

Mémoire

1 bit = 0 ou 1
1 octet = 8 bits (par exemple 00110101)
1 byte = espace de mémoire élémentaire (généralement 1 octet)

La mémoire centrale (mémoire vive) d'un ordinateur est composée de milliards de cases. Chaque case mémorise un bit. Les cases sont organisées en bytes (ici on supposera 1 byte = 1 octet).

Chaque octet de la mémoire est identifié par une adresse (un nombre).

Pointeur

Exemple

Une mémoire de 256 o aura donc 256 bytes. On peut adresser ces zones par des nombres de 0 à 255 :

Adresses			Bytes	Bits
Décimal	hexa	binaire		
$0d000$	$0x00$	$0b0000\,0000$	1^{er} byte	0 à 7
$0d001$	$0x01$	$0b0000\,0001$	2^{e} byte	8 à 15
$0d002$	$0x02$	$0b0000\,0010$	3^{e} byte	16 à 23
...	...	...	...	...
$0d255$	$0xFF$	$0b0011\,1111$	255^{e} byte	2039 à 2047

Introduction : mémoire et adressage

Prenons l'exemple d'une variable entière $i = 42$ à l'adresse $0x03$ et un tableau T dont l'adresse de la première valeur est $0x10$.

Sur l'exemple Python

- L'appel `modiInt(i)` copie la valeur de `i` située à l'adresse $0x03$ dans une nouvelle variable locale à la fonction à une autre adresse, par exemple $0xA0$. C'est un **passage par valeur**. C'est la case $0xA0$ que la fonction va modifier, et pas $0x03$.
- L'appel `modiTab(T)` copie l'adresse $0x10$ de `T` dans une nouvelle variable locale à la fonction. Une instruction `T[0]` accède directement à $0x10$. Pour un tableau il s'agit d'un **passage par référence**.

En Python il y a une différence de traitement selon le type des paramètres.

Mémoire et adressage C

Adresse

Soit **int** i ;

- i permet d'accéder à la valeur de la variable i ;
- &i est l'adresse de la variable i.

Si i est une variable contenant la valeur 42 à l'adresse 0x03, alors i vaut 42 et &i vaut 0x03.

Exemple

```
int i = 42;
printf("i : %i, adresse de i : 0x%p \n",i,(int)&i);
```

Pointeur en C

Pointeur

Soit `int i` une variable. Un pointeur `p` vers un entier est une variable contenant l'adresse d'un entier :

- type de `p` : `int *`
- déclaration : `int * p;`
- initialisation : `p = &i;`

Notations

Soient `i` et `p` comme ci-dessus :

- `p` désigne une adresse, ici l'adresse de `i` ;
- `*p` est la valeur pointée par `p`, ici la valeur de `i`.
  On dit que le pointeur `p` est *déréférencé*.

Pointeurs : exemples

Valeurs de i et j ?

```
int i,j;
int * p1,p2;
i = 10;
j = 6;
p1 = &i;
p2 = &j;
*p1 = *p2;
```

Pointeurs : exemples

Valeurs de i et j ?

```
int i,j;
int * p1,p2;
i = 10;
j = 6;
p1 = &i;
p2 = &j;
*p1 = *p2;
```

À regarder dans ArtEoz.

Pointeurs : exemples

Valeurs de i et j ?

```
int i,j;
int * p1,p2;
i = 10;
j = 6;
p1 = &i;
p2 = &j;
p1 = p2;
```

Pointeurs : exemples

Valeurs de i et j ?

```
int i,j;
int * p1,p2;
i = 10;
j = 6;
p1 = &i;
p2 = &j;
p1 = p2;
```

À regarder dans ArtEoz.

Pointeurs

À quoi ça sert ?

- Contrôler le passage par valeur ou par référence.
- Identifier des paramètres en sortie (d'où plusieurs sorties).
- Gérer efficacement (sans recopie inutile) des structures de données complexes.
- Allouer dynamiquement la mémoire.

Pointeur : erreurs de segmentation (segfault)

Définition

Une erreur de segmentation (*segmentation fault*) arrive lorsqu'un pointeur est déréférencé vers une zone mémoire non allouée.

Par exemple :

```
int* p;
p = 0;
printf("%p\n",*p);
```

La zone mémoire à l'adresse 0 n'est pas accessible.