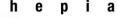
Langage C - Base II

2018

Florent Gluck - Florent.Gluck@hesge.ch

Version 0.2





Structures

- Une structure permet de combiner des variables (record en ADA).
- Les champs sont accessibles avec le sélecteur « . »

```
// Définition de la structure Complex
struct Complex {
   double real:
   double img;
};
// Déclaration variable num
struct Complex num;
// Initialisation variable num
num.real = 2.5;
num.img = 0.3;
// Initialisation variable num2
struct Complex num2 = { 2.5, 0.3 };
```

Définition de type

- L'utilisation de typedef permet de définir un nouveau type.
- Syntaxe: «typedef type nom»

```
// Définition du type uchar
typedef unsigned char uchar;
// Définition du type complex t
typedef struct Complex complex t;
uchar v = 123;
complex t var;
var.real = 5.9;
var.imag = 0.1;
complex t var2 = \{-0.7, 42\};
complex t var3 = var;
```

3

Enumérations

 En réalité, un type énuméré est de type entier; le compilateur assigne la valeur de 0 au premier élément, 1 au deuxième élément, etc.

Variables static

- Par défaut, une variable globale est visible (= utilisable) depuis n'importe quel fichier ("module").
- Une variable globale déclarée static est uniquement visible dans le fichier ("module) où elle est déclarée.
- Une variable locale déclarée static est en réalité une variable globale seulement visible dans la fonction où elle est déclarée.

Typecasting: forcer un type

- Il est possible de forcer le compilateur à intérpreter une variable selon le type spécifié.
- (type) x force la variable x à être du type type.
 - Ex: conversion double → int: prend la valeur entière.
- Très utilisé dans le cas de pointeurs génériques (void *) car ils permettent de créer des fonctions à arguments génériques.
- Quelques exemples de typecasts :

```
int a = 10, b = 4, x = 200, y = 260;
int c = a / b;
double d = a / b;
double e = (double)a/(double)b;
unsigned char car1 = (char)x;
unsigned char car2 = (char)y;

void func(void *data) {
  int *id = (int *)data;
}
Valeurs ?
```

Typecasting: forcer un type

- Il est possible de forcer le compilateur à intérpreter une variable selon le type spécifié.
- (type) x force la variable x à être du type type.
 - Ex: conversion double → int : prend la valeur entière.
- Très utilisé dans le cas de pointeurs génériques (void *) car ils permettent de créer des fonctions à arguments génériques.
- Quelques exemples de typecasts :

```
int a = 10, b = 4, x = 200, y = 260;
int c = a / b; // c = 2
double d = a / b; // d = 2
double e = (double)a/(double)b; // e = 2.5
unsigned char car1 = (char)x; // car1 = 200
unsigned char car2 = (char)y; // overflow! ⇒ car2 = 4

void func(void *data) {
   int *id = (int *)data;
}
```



Types de base (1)

- En C, pas de garantie sur la taille des types entiers :
 - dépend de l'architecture matérielle cible ;
 - dépend du compilateur C utilisé.
- Le fichier /usr/include/limits.h indique les limites des différents types.

Types de base (2)

Туре	C (officiel)	C (gcc sur x86-64)
char, unsigned char	Entier 8-bits	8-bits
short, unsigned short	Entier ≥ 16-bits	16-bits
int, unsigned int	Entier ≥ 16-bits	32-bits
long, unsigned long	Entier ≥ 32-bits	64-bits
long long, unsigned long long	Entier ≥ 64-bits (C99)	64-bits
float	Non défini, mais généralement IEEE 754 simple précision	32-bits
double	Non défini, mais généralement IEEE 754 double précision	64-bits
long double	Non défini	128-bits

Mot-clé sizeof

Le mot clé sizeof renvoie, en bytes, la taille d'un type ou d'une variable:

```
int main(void) {
  int n = 17;
  unsigned char c = 0;
  printf("size n = %d\n", sizeof(n));
  printf("size c = %d\n", sizeof(c));
  printf("char = %d\n", sizeof(char));
  printf("int = %d\n", sizeof(int));
  printf("long = %d\n", sizeof(long));
  printf("double = %d\n", sizeof(double));
  return 0;
}
```

Mot-clé sizeof

Le mot clé sizeof renvoie, en bytes, la taille d'un type ou d'une variable:

```
size n = 4
                                           size c = 1
int main(void) {
                                           char = 1
   int n = 17;
                                           int = 4
   unsigned char c = 0;
                                          lonq = 4
   printf("size n = %d n", sizeof(n));
                                           double = 8
   printf("size c = %d n", sizeof(c));
   printf("char = %d\n", sizeof(char));
   printf("int = %d\n", sizeof(int));
   printf("long = %d\n", sizeof(long));
   printf("double = %d\n", sizeof(double));
   return 0:
```

Code de retour du programme (1)

- Par définition, un programme qui se termine correctement doit renvoyer un code de retour de 0.
 - Ceci est réalisé par le mot-clé return dans la fonction main.
- Un code de retour différent de 0 signifie une erreur.
- Préférable d'utiliser les constantes EXIT_SUCCESS et EXIT_FAILURE définies dans le fichier header unistd.h

```
#include <unistd.h>
int main(void) {
    ...
    if (error)
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}
```



Code de retour du programme (2)

- Le code de retour d'un programme peut être lû dans le shell (bash) avec la variable \$?
- Cela permet de déterminer si le programme exécuté s'est terminé correctement ou pas.

```
gluckf@hal9000 ~ $ ./myprog

gluckf@hal9000 ~ $ if [ $? -eq 0 ]; then echo "OK" ;
else echo "ERROR"; fi
```

