Variables et fonctions

Programmation séquentielle en C, 2021-2022

Orestis Malaspinas (A401), ISC, HEPIA

2021-10-05

Inspirés des slides de F. Glück

Les variables

Qu'est-ce qu'une variable?

Les variables

Qu'est-ce qu'une variable?

- Un **identifiant**

Les variables

Qu'est-ce qu'une variable?

- Un **identifiant**
- pour un **espace de stockage**

Les variables

Qu'est-ce qu'une variable?

- Un **identifiant**
- pour un **espace de stockage**
- contenant un **valeur**.

En général, une variable possède:

- Un **type** (`int`, `double`, ...);
- une **adresse mémoire** (voir ci-après).

Représentation des variables en mémoire (1/3)

La mémoire est :

- ... un ensemble de bits,
- ... accessible via des adresses,

bits	00110101	10010000	....	00110011	....	....
addr	2000	2001	....	4000	....	....

- ... gérée par le système d'exploitation.
- ... séparée en deux parties: **la pile** et **le tas**.

Pile vs tas

- Pile structurée, tas non-structuré;
- Pile ordonnée, tas amas informe;
- Pile taille connue à la compilation, tas dynamique.

Représentation des variables en mémoire (2/3)

Une variable, `type a = valeur`, possède:

- un type (`char`, `int`, ...),
- un contenu (une séquence de bits qui encode `valeur`),
- une adresse mémoire (accessible via `&a`),
- une portée.

En fonction du type les bits ne représentent pas la même chose!

Représentation des variables en mémoire (3/3)

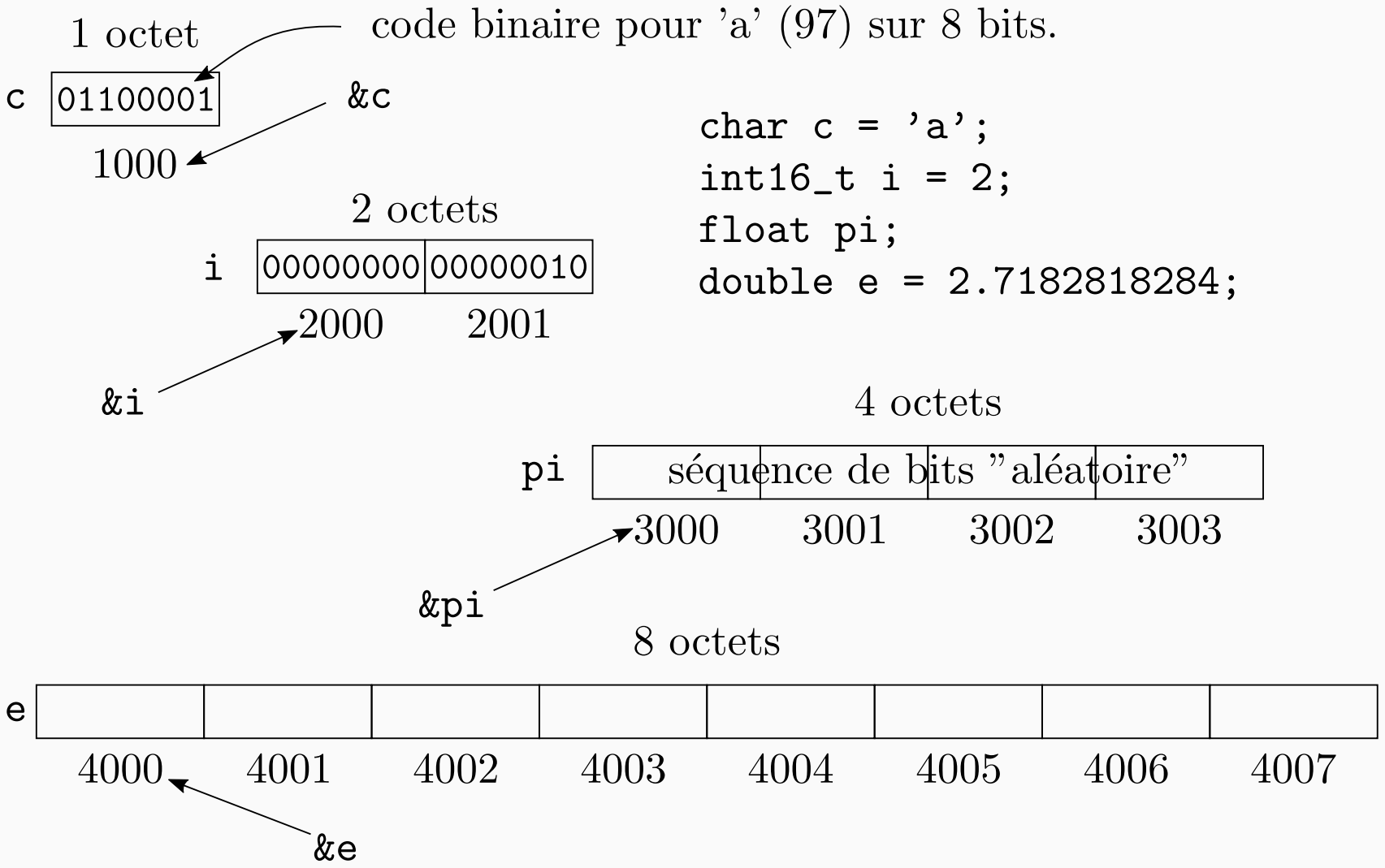

Figure 1: Les variables en mémoire.

Les fonctions (1/7)

- Les parties indépendantes d'un programme.
- Permettent de modulariser et compartimenter le code.
- Syntaxe:

```
type identificateur(paramètres) {
    // variables optionnelles
    instructions;
    // type expression == type
    return expression;
}
```

Les fonctions (2/7)

Exemple

```
int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}

int main() {
    int c = max(4, 5);
}
```

Les fonctions (3/7)

- Il existe un type `void`, “sans type”, en C.
- Il peut être utilisé pour signifier qu'une fonction ne retourne rien, ou qu'elle n'a pas d'arguments.
- `return` utilisé pour sortir de la fonction.
- Exemple:

```
void show_text(void) { // second void optionnel
    printf("Aucun argument et pas de retour.\n");
    return; // optionnel
}

void show_text_again() { // c'est pareil
    printf("Aucun argument et pas de retour.\n");
}
```

Les fonctions (4/7)

Prototypes de fonctions

- Le prototype donne la **signature** de la fonction, avant qu'on connaisse son implémentation.
- L'appel d'une fonction doit être fait **après** la déclaration du prototype.

```
int max(int a, int b); // prototype

int max(int a, int b) { // implémentation
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

Les fonctions (5/7)

Arguments de fonctions

- Les arguments d'une fonction sont toujours passés **par copie**.
- Les arguments d'une fonction ne peuvent **jamais** être modifiés.

```
void set_to_two(int a) { // a: nouvelle variable
    // valeur de a est une copie de x
    // lorsque la fonction est appelée, ici -1
    a = 2; // la valeur de a est fixée à 2
} // a est détruite
int main() {
    int x = -1;
    set_to_two(x); // -1 est passé en argument
    // x vaudra toujours -1 ici
}
```

Les fonctions (6/7)

Arguments de fonctions: pointeurs

- Pour modifier un variable, il faut passer son **adresse mémoire**.
- L'adresse d'une variable, `x`, est accédé par `&x`.
- Un **pointeur** vers une variable entière a le type, `int *x`.
- La syntaxe `*x` sert à **déréférencer** le pointeur (à accéder à la mémoire pointée).

Les fonctions (7/7)

Exemple

```
void set_to_two(int *a) {
    // a contient une copie de l'adresse de la
    // variable passée en argument

    *a = 2; // on accède à la valeur pointée par a,
            // et on lui assigne 2
} // le pointeur est détruit, pas la valeur pointée
int main() {
    int x = -1;
    set_to_two(&x); // l'adresse de x est passée
    // x vaudra 2 ici
}
```

Quiz: Les fonctions

Comment lire une signature?

Que peut-on déduire de la signature de la fonction suivante:

```
int32_t bissect(double a1, double b1, double epsilon,
                double *zero);
```

Comment lire une signature?

Que peut-on déduire de la signature de la fonction suivante:

```
int32_t bissect(double a1, double b1, double epsilon,
                double *zero);
```

Une fonction prenant trois `double` en argument, et un pointeur de `double` (il est donc modifiable) et retournant un entier.

Un site permettant de faire ce genre de traductions:
https://cdecl.org/