Base VI

Programmation séquentielle en C, 2020-2021

Orestis Malaspinas (A401), ISC, HEPIA

2020-10-28

Inspirés des slides de F. Glück

Types énumérés

Type `enum` (1/2)

- Un **type énuméré** est un ensemble de valeurs constantes: les *variantes*.
- En `C` les variantes sont des entiers numérotés à partir de 0.

```
enum days {
    monday, tuesday, wednesday,
    thursday, friday, saturday, sunday
};
```

- On peut aussi donner des valeurs “custom” `C`

```
enum days {
monday = 2, tuesday = 8, wednesday = -2,
thursday = 1, friday = 3, saturday = 12, sunday = 9
};
```

Type `enum` (2/2)

- Très utile dans les `switch ... case`

```
enum days d = monday;
switch (d) {
    case monday:
        // trucs
        break;
    case tuesday:
        printf("0 ou 1\n");
        break;
}
```

- Le compilateur vous prévient qu'il en manque!

Pointeurs de pointeurs

Doubles pointeurs

- Un pointeur étant un type comme un autre, on peut définir un pointeur sur un pointeur:

```
// pointeur sur un pointeur d'entier 32bits
int32_t **double_ptr;
```

- Cela peut servir à deux choses principalement

  1. Allouer un tableau de tableau.
  2. Modifier un pointeur en argument dans une fonction.

- On peut même aller plus loin et avoir ds pointeurs de pointeurs de pointeurs ... de pointeurs.

```
int32_t **********************a;
```

Exemple: tableau de tableaux

- Cas pratique

```
int32_t **p = malloc(3 * sizeof(int32_t *));
p[0] = malloc(3 * sizeof(int32_t));
p[1] = malloc(5 * sizeof(int32_t));
p[2] = malloc(8 * sizeof(int32_t));
```

- Faites un dessin de ce qui se passe en mémoire!

Exemple: argument d'une fonction

- Modification d'un pointeur en argument à une fonction

```
void alloc_ptr(int32_t **p, int32_t size) {
    *p = malloc(size * sizeof(int32_t));
}
```

- Que se passe-t-il si on utilise pas un `int32_t **p` mais un `int32_t *p`.

Pointeurs avancés

Pointeurs de fonctions (1/3)

- Considérons la fonction `max` retournant la valeur maximale d'un tableau

```
int32_t max(int32_t *t, int32_t size) {
    int32_t val_max = t[0];
    for (int32_t i = 1; i < size; ++i) {
        if (t[i] > val_max) {
            val_max = t[i];
        }
    }
    return val_max;
}
```

- L'appel à `max`, retourne l'adresse de la fonction en mémoire.
- On peut affecter cette valeur à un pointeur.

Pointeurs de fonctions (2/3)

- Le type de la fonction `max` est

```
int32_t (*pmax)(int32_t *, int32_t);
```

- Le type doit être déclaré avec la signature de la fonction.
- On peut alors utiliser l'un ou l'autre indifféremment

```
int32_t (*pmax)(int32_t *, int32_t);
pmax = max;
int32_t tab[] = {1, 4, -2, 12};
printf("%d", max(tab, 4)); // retourne 12
printf("%d", pmax(tab, 4)); // retourne 12 aussi
```

Pointeurs de fonctions (3/3)

- On peut passer des fonctions en argument à d'autres fonctions

```
int32_t reduce(int32_t *tab, int32_t size, int32_t init,
               int32_t (*red_fun)(int32_t, int32_t))
{
    for (int32_t i = 0; i < size; ++i) {
        init = red_fun(init, tab[i]);
    }
    return init;
}
```

- Ici une fonction de *réduction* `sum()`

```
int32_t sum(int32_t lhs, int32_t rhs) {
    return lhs + rhs;
}
int32_t tab[] = {1, 4, -2, 12};
int32_t red = reduce(tab, 4, 0, sum);
printf("%d", red); // affiche 15
```