La généricité

Programmation séquentielle en C, 2023-2024

Orestis Malaspinas (A401)

2024-03-25

Informatique et Systèmes de Communication, HEPIA

Problématique

- En C on doit écrire chaque algorithme/structures de données pour des types précis (`int`, `double`, `char`, ...).

```
void int_sort(int size, int tab[size]);    // tri d'entiers
void double_sort(int size, int tab[size]); // tri de double
void char_sort(int size, char tab[size]);  // tri de char
```

- Duplication du code pour chaque type possible et imaginable.
- On aimerait un moyen pour pouvoir représenter “n’importe quel type” sans réécrire tout le code.

La généricité

Une "solution": `void *`

- En général, un pointeur connaît son **adresse** et le **type** des données sur lesquelles il pointe.

```
int *a = malloc(sizeof(*a));
int *b = malloc(sizeof(int));
```

- Un `void *` le connaît **que** son adresse, au programmeur de pas faire n'importe quoi.
- Vous avez déjà utilisé des fonctions utilisant des `void *`

```
void *malloc(size_t size);
void free(void *);
```

Attention danger

- Ne permet pas au compilateur de vérifier les types.
- Les données pointées n'ayant pas de type, il faut déréférencer avec précaution:

```
int a = 2;
void *b = &a; //jusqu'ici tout va bien
double c = *b; // argl!
```

- Une attention accrue est nécessaire.

Cas particuier: on sait pas comment libérer la mémoire

Exemple

```
struct tab {
    int *t;
}
struct tab *tmp = malloc(sizeof(*tmp));
tmp->t = malloc(10 * sizeof(*(tmp->t)));
free(tmp); // memory leak of tmp->t...
```

Cas particuier: on sait pas comment libérer la mémoire

Exemple

```
struct tab {
    int *t;
}
struct tab *tmp = malloc(sizeof(*tmp));
tmp->t = malloc(10 * sizeof(*(tmp->t)));
free(tmp); // memory leak of tmp->t...
```

Solution: tout faire à la main

```
free(tmp->t);
free(tmp);
```

Exemple simple

- On souhaite échanger deux pointeurs

```
int *a = malloc();
int *b = malloc();
swap(&a, &b);
```

- Comment écrire `swap()` pour que le code ci-dessus marche pour n'importe quel type?

Exemple simple

- On souhaite échanger deux pointeurs

```
int *a = malloc();
int *b = malloc();
swap(&a, &b);
```

- Comment écrire `swap()` pour que le code ci-dessus marche pour n'importe quel type?

```
void swap(void **a, void **b) {
    void *tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Cas d'utilisation (1/4)

- La somme d'un tableau de type arbitraire (facile non?)

```
void sum(void *tab, int length, size_t size_elem, void *zero,
    void (*add)(void *, void *)) {
    for (int i = 0; i < length; ++i) {
        void *rhs = (void *)((char *)tab + i * size_elem);
        add(zero, rhs);
    } // de combien on "saute" avec un void *?
}
```

- Pour des entiers

```
void int_add(void *lhs, void *rhs) {
    *((int *)lhs) += *((int *)rhs); // cast d'entiers
}
int zero  = 0;
int tab[] = {1, -2, 4, 5};
sum(tab, 4, sizeof(int), &zero, int_add);
printf("%d\n", zero);
```

Cas d'utilisation (2/4)

Que fait cette fonction?

```
void *foo(void *tab, int n_items, int s_items,
          bool (*bar)(void *, void *)) {
    if (n_items <= 0 || s_items <= 0 || NULL == tab) {
        return NULL;
    }
    void *elem = tab;
    for (int i = 1; i < n_items; ++i) {
        // void pointer arithmetics is illegal in C
        // (gcc is ok though)
        void *tmp_elem = (void *)((char *)tab + i*s_items);

        if (bar(elem, tmp_elem)) {
            elem = tmp_elem;
        }
    }
    return elem;
}
```

Cas d'utilisation (3/4)

Avec un tableau de `int`

```
bool cmp_int(void *a, void *b) {
    return (*(int *)a < *(int *)b);
}

int main() {
    int tab[] = {-1, 2, 10, 3, 8};
    int *a = foo(tab, 5, sizeof(int), cmp_int);
    printf("a = %d\n", *a);
}
```

Cas d'utilisation (4/4)

Avec un tableau de `double`

```
bool cmp_dbl(void *a, void *b) {
    return (*(double *)a < *(double *)b);
}

int main() {
    double tab[] = {-1.2, 2.1, 10.5, 3.6, 18.1};
    double *a = foo(tab, 5, sizeof(double), cmp_dbl);
    printf("a = %f\n", *a);
}
```