Fonctions d'ordre supérieur

Programmation séquentielle en C, 2022-2023

Orestis Malaspinas (A401) et un tout petit peu Michaël El Kharroubi

2023-03-21

Informatique et Systèmes de Communication, HEPIA

Tribute

Rendons à Cesar:

- Ces slides ont été écrits par Michaël El Kharroubi
- J'arrive pas à changer l'auteur simplement sur un slide donc....
- Merci à lui pour ses efforts et qu'il soit crédité comme il se doit!

Présentation du problème

- Imaginons que nous ayons la structure d'un vecteur en 3 dimensions suivante

```
typedef struct _vec3 {
    double x;
    double y;
    double z;
} vec3;
```

- On souhaite implémenter 3 opérations différentes
  - La somme
  - La soustraction
  - Le produit de Hadamard (produit composantes à composantes)

Présentation du problème (suite)

On a donc les fonctions suivantes

- Addition

```
vec3 add(vec3 lhs, vec3 rhs){
    vec3 res;
    res.x = lhs.x + rhs.x;
    res.y = lhs.y + rhs.y;
    res.z = lhs.z + rhs.z;
    return res;
}
```

Présentation du problème (suite)

- Soustraction

```
vec3 sub(vec3 lhs, vec3 rhs){
    vec3 res;
    res.x = lhs.x - rhs.x;
    res.y = lhs.y - rhs.y;
    res.z = lhs.z - rhs.z;
    return res;
}
```

Présentation du problème (suite)

- Produit de Hadamard

```
vec3 mul(vec3 lhs, vec3 rhs){
    vec3 res;
    res.x = lhs.x * rhs.x;
    res.y = lhs.y * rhs.y;
    res.z = lhs.z * rhs.z;
    return res;
}
```

- Quel est le problème avec ces trois fonctions?

Présentation du problème (suite)

- Le problème avec ces fonctions c'est la **répétition**.
- La seule chose qui change, c'est l'opérateur (+,-,*).
- Problèmes possibles
  - Tentation de copier-coller du code (donc risque d'erreurs)
  - Faible résilience au changement (imaginons que je veuille des vecteurs 2d, 4d, nd)

Présentation du problème (solution)

- Vecteur de taille dynamique

```
typedef struct _vecn {
    int size;
    double *xs;
} vecn;
```

- Règle le problème de résilience du code, mais ne règle pas le problème de répétition...

Fonction d'ordre supérieur (solution au problème)

- Pour notre problème, nous aimerions donc découpler l'opération (opération entre deux termes : +,-,*) de l'itération sur les composantes.
- Ce qui nous donne conceptuellement en pseudo c

```
// Attention pseudo c, ne compile pas !!!!!
vec3 apply_operator(operator op, vec3 lhs, vec3 rhs){
    vec3 res;
    res.x = lhs.x op rhs.x;
    res.y = lhs.y op rhs.y;
    res.z = lhs.z op rhs.z;
    return res;
}
```

Fonction d'ordre supérieur (solution au problème)

- Avec notre fonction conceptuelle `apply_operator`, on pourrait faire (toujours en pseudo c)

```
// Attention pseudo c, ne compile pas !!!!!
vec3 add(vec3 lhs, vec3 rhs){
    return apply_operator(+, lhs, rhs);
}
vec3 sub(vec3 lhs, vec3 rhs){
    return apply_operator(-, lhs, rhs);
}
vec3 mul(vec3 lhs, vec3 rhs){
    return apply_operator(*, lhs, rhs);
}
```

- En fait, on vient de créer ce qu'on appelle une fonction d'ordre supérieur.

Fonction d'ordre supérieur (définition)

- Une fonction d'ordre supérieur est une fonction qui prend en paramètre et/ou retourne une(des) autre(s) fonction(s).
- Si on essayait de définir `operator`, c'est en fait une fonction qui prend deux paramètres (un terme de gauche et un terme de droite). On s'en aperçoit clairement avec la notation préfix (polonaise).
  - `L + R -> + L R`
  - `L - R -> - L R`
  - `L * R -> * L R`
- Comment l'implémenter concrètement en C?

Implémentation

- Si on reprend la signature de notre fonction d'exemple, on a

```
vec3 apply_operator(operator op, vec3 lhs, vec3 rhs);
```

- Nous avons déterminé que les `operator` étaient des fonctions qui prennaient deux paramètres.
- Pour passer une fonction en paramètre en C, nous devons la passer par référence, c'est à dire à l'aide d'un pointeur de fonction.

Pointeur de fonctions

- Un pointeur de fonction se définit ainsi

```
<type retour> (*<nom ptr fonc>)(<type params(s)>);
```

- Ou encore avec un `typedef`

```
typedef <type retour> (*<nom ptr fonc>)(<type params(s)>);
```

- Dans notre cas, nous avons donc un type de fonction nommé `operator`, qui prend en entrée deux `double` et qui retourne un `double`. Ce qui nous donne

```
typedef double (*operator)(double, double);
```

Implémentation (suite)

- En reprenant notre fonction `apply_operator`, on a donc

```
vec3 apply_operator(operator op, vec3 lhs, vec3 rhs){
    vec3 res;
    res.x = op(lhs.x, rhs.x);
    res.y = op(lhs.y, rhs.y);
    res.z = op(lhs.z, rhs.z);
    return res;
}
```

- NB : On voit que pour appeler notre fonction passée en paramètre, nous avons pu le faire comme avec n'importe quelle fonction.

Résultat

```
typedef double (*operator)(double, double);
vec3 apply_operator(operator op, vec3 lhs, vec3 rhs){
    vec3 res;
    res.x = op(lhs.x, rhs.x);
    res.y = op(lhs.y, rhs.y);
    res.z = op(lhs.z, rhs.z);
    return res;
}
double add_dbl(double lhs, double rhs){
    return lhs + rhs;
}
vec3 add(vec3 lhs, vec3 rhs){
    return apply_operator(add_dbl, lhs, rhs);
}
```

Fonctions d'ordre supérieur appliquées aux tableaux

- Comment appliquer des opérations sur un vecteur de taille n?
  - Map (application d'une fonction)
    - `add_one`, `square`
  - Filter (discrimination selon un prédicat)
    - `is_even`, `is_lower_than_five`
  - Reduce (réduction d'un vecteur à un seul élément)
    - `sum`, `multiply`

Le map

- Exemple d'application

```
typedef double (*operator)(double);
double *map(operator op, double *tab, size_t size) {
    double *res = malloc(sizeof(*res) * size);
    for (int i = 0; i < size; ++i) {
        res[i] = op(tab[i]);
    }
    return res;
}
double add_one(double val) {
    return val + 1;
}
double sqr(double val){
    return val * val;
}
double tab[] = {1.0, 2.0, 3.0};
double *square = map(sqr, tab, 3);
double *and_one = map(add_one, square, 3);
```

Le map

- Permettrait le chaînage.

```
double *sqr_and_one = map(add_one, map(sqr, tab, 3), 3);
```

Le map

- Permettrait le chaînage.

```
double *sqr_and_one = map(add_one, map(sqr, tab, 3), 3);
```

- Problème?

Le map

- Permettrait le chaînage.

```
double *sqr_and_one = map(add_one, map(sqr, tab, 3), 3);
```

- Problème?
- Allocation dynamique... fuite mémoire.

Le map

- Permettrait le chaînage.

```
double *sqr_and_one = map(add_one, map(sqr, tab, 3), 3);
```

- Problème?
- Allocation dynamique... fuite mémoire.
- Solution?

Le map

- Permettrait le chaînage.

```
double *sqr_and_one = map(add_one, map(sqr, tab, 3), 3);
```

- Problème?
- Allocation dynamique... fuite mémoire.
- Solution?

```
```c
typedef double (*operator)(double);
double *map(operator op, double *tab, size_t size) {
    double *res = malloc(sizeof(*res) * size);
    for (int i = 0; i < size; ++i) {
        res[i] = op(tab[i]);
    }
    free(tab);
    return res;
}
```
```

- Problème potentiel?
- **Attention au double free!**