Les fonctions recursives

Une fonction récursive est une fonction qui s'appelle elle-même. Cela est donc équivalent aux boucles et permettent de répéter une suite d'instructions.

Dans un code, il suffit donc de mettre un appel vers une fonction dans le corps de cette fonction.

```
void f() {
    f();  // appel récursif de la fonction f
}
```

Même si ce n'est pas à proprement parlé des fonctions récursives, il est possible de considérer aussi les fonctions qui ne s'appellent pas elle-même directement, via d'autres fonctions intermédiaires, et qui forment des boucles d'appels de fonctions (récursivité croisée). Par exemple :

```
void g();  // declaration anticipee

void f() {
    g();
}

void g() {
    f();
}
```

Dans ce code, la fonction `f` appelle la fonction `g`, qui appelle la fonction `f`, qui appelle la fonction `g`, et ainsi de suite.

Declaration anticipee
Si vous lisez le code ligne par ligne, comme le fait le compilateur, vous remarquez que dans la fonction `f`, la fonction `g` n'est pas encore connue

par le compilateur. Vous aurez donc un message d'erreur “g n'est pas déclarée” :

```
main.cpp: In function 'void f()':
main.cpp:2:7: error: 'g' was not declared in this scope
     g();
       ^
```

Si vous définissez la fonction `g` avant la fonction `f`, vous aurez le même probleme inverse (“f n'est pas connu”).

Une déclaration anticipée permet de dire au compilateur que la fonction `g` existe, mais qu'elle sera définie plus tard. Et c'est suffisant pour le compilateur.

Une déclaration anticipée est simplement la signature d'une fonction, avec le corps de la fonction qui est remplacée par un point-virgule.

Si vous n'écrivez que la déclaration anticipée et oubliez de définir la fonction, vous aurez une erreur de lien (“référence indéfinie”) :

```
In function `f()': undefined reference to `g()'
```

Condition d'arret et continuation

Comme pour les boucles classiques, il est important de faire attention aux conditions d'arrêt, pour éviter les boucles infinies. Dans les codes précédents, il n'y a pas de condition d'arrêt, ce qui implique que ces boucles tourneront à l'infini, jusqu'à ce que le programme plante.

Contrairement aux instructions itératives vues précédemment, il n'y a pas de syntaxe explicites dans une fonction récursive pour arrêter la récursion. Vous devez donc utiliser un test conditionnel pour tester la condition d'arrêt et choisir de continuer ou non la récursivité.

Classiquement, cela veut dire que vous devez utiliser un test `if`, avec une condition d'arrêt associé à une clause `return` pour terminer la fonction, ou avec une condition de continuation associé à un appel

récursif de la fonction.

Par exemple, avec une condition d'arrêt :

```
void f() {
    if (CONDITION_ARRET) {
        return;
    }
    f();  // appel récursif
}
```

Dans ce code, lorsque la condition est vrai, l'instruction `return` est exécutée, ce qui termine la fonction `f`. Dans le cas contraire, la fonction `f` est appelée récursivement.

Un autre exemple, avec une condition de continuation :

```
void f() {
    if (CONDITION_CONTINUATION) {
        f();  // appel récursif
    }
}
```

Dans ce code, lorsque la condition est vraie, la fonction `f` est appelée récursivement. Dans le cas contraire, la fonction s'arrête.

Complexite et limitation

Selon la complexité des algorithmes, il n'est pas toujours simple de garantir qu'un algorithme récursif se termine toujours correctement, quelques soit les conditions. La démonstration qu'un algorithme est correct peut être rapproché des raisonnement par récurrence en mathématique.

performances

Un compteur récursif

Pour être concret, voici un exemple simple de fonction récursive, pour réaliser un compteur.

```
#include <iostream>

void f(int i) {
    std::cout << i << std::endl;
    if (i > 0) {
        f(i - 1);
    }
}

int main() {
    f(5);
}
```

affiche

```
5
4
3
2
1
0
```

Dans ce code, la fonction `f` prend en paramètre un compteur entier. Ce paramètre est passé par valeur, ce qui signifie que la valeur sera copiée lors de chaque appel à la fonction `f`.

La valeur du compteur est affichée dans un premier temps avec `std::cout`. Ensuite, cette valeur est testée : si elle est superieur à zéro, la fonction `f` est appelée récursivement. Dans le cas contraire, la fonction s'arrête.

Le point important à noter est que le compteur est décrémenté à chaque appel récursif à la fonction `f`. Pour bien comprendre ce qui se passe, il faut “dérouler” la boucle :

- la fonction `main` appelle la fonction `f` avec la valeur `5` en argument ;

- dans le premier appel à la fonction `f`, le test `if` est vrai (5 est superieur à 0), la fonction `f` est appelée récursivement, avec la valeur 5 - 1 = 4 ;
- dans le deuxième appel à la fonction `f`, le test `if` est vrai (4 est superieur à 0), la fonction `f` est appelée récursivement, avec la valeur 4 - 1 = 3 ;
- dans le troisième appel à la fonction `f`, le test `if` est vrai (3 est superieur à 0), la fonction `f` est appelée récursivement, avec la valeur 3 - 1 = 2 ;
- dans la quatrième appel à la fonction `f`, le test `if` est vrai (2 est superieur à 0), la fonction `f` est appelée récursivement, avec la valeur 2 - 1 = 1 ;
- dans la quatrième appel à la fonction `f`, le test `if` est vrai (1 est superieur à 0), la fonction `f` est appelée récursivement, avec la valeur 1 - 1 = 0 ;
- dans la quatrième appel à la fonction `f`, le test `if` est faux (0 n'est pas superieur à 0), la fonction `f` se termine.

Diviser pour régner

https://fr.wikipedia.org/wiki/Diviser_pour_r%C3%A9gner_(informatique)

divide and conquer en anglais

grande classe de resolution de problemes via recurision. Idee : diviser en sous preobleme, resoudre les sous probleme, combiner les resultat.

Exemple concret : vous avez une pile de feuilles, que vous devez relire et corriger les fautes d'orthographe. Pour proceder :

- vous diviser le paquet en 2
- vous donner la moitiee a une autre personne, pour qu'elle corrige la moitie et vous vous occuper de l'autre moitie
- vous divisez encore en 2 le paquet qu'il vous reste et vous donner la moitie a une troisieme personne. La deuxieme

personne fait la meme chose et donne la moitie de son paquet a une quatrieme personne ;
- vous recommancez : chacun divise son paquet en 2 et donne la moitie a 4 autres personnes
- et ainsi de suite, jusqu'a temps que vous n'ayez pas trop de pages a corriger.

Exemples : recherche, tri, arbres,

Note : principe application aussi aux structures de donnees. Par exemple, au lieu d'avoir un vector de N elements, vous pouvez avoir 2 vector de N/2 elements. Ou 4 vectors de N/4 elements, etc.

Ces structures de donnees “recursives” presentent des avantages tres interessants pour certaines problematiques (std::map utilise par exemple en interne ce type de structure de donnees). Et les algorithmes recurisves sont donc en toute logique particulierement adpatee pour travailler sur de telles structures.

arbres, graph, etc.

Exercices

Aide : https://en.wikipedia.org/wiki/Recursion_(computer_science) et https://fr.wikipedia.org/wiki/Algorithme_r%C3%A9cursif

1. Ecrire une fonction récursive qui calcul la fonctionnelle d'un nombre entier : n! = 1 * 2 * 3 * ... * n.

2. Ecrire une fonction récursive qui calcul le plus grand commun diviseur (PGCD) de deux nombres entiers. Le PGCD est le plus grand entier qui divise ces deux nombres simultanément.

3. https://fr.wikipedia.org/wiki/Partition_d%27un_entier

4. Résoudre le jeu “les tours de Hanoi” par une fonction récursive. https://fr.wikipedia.org/wiki/Tours_de_Hano%C3%AF

5. recherche dichotomique.

Chapitre précédent	Sommaire principal	Chapitre suivant