Курс c++

Царьков Олег

Описание тикета OTSN-13

Шаблон проектирования "функтор". Идея основана на том, что внутри класса можно переопределить оператор *operator*() и тогда вызов этого оператора будет похож на обычный вызов функции(пишется объект класса, рядом с ним круглые скобки, в которых что-то написано).

Пример:

```
#include <cstdlib>
#include <iostream>
#include <vector>

template <typename T>
class BinaryAction {
public:
    virtual T operator() (const T& a, const T& b) const = 0;
};

template <typename T>
class Sum : public BinaryAction<T> {
    T operator() (const T& a, const T&b) const {
        return a + b;
    }
};

template <typename T>
class Prod : public BinaryAction<T> {
    T operator() (const T& a, const T&b) const {
        return a * b;
    }
};

template <typename T>
T GroupAction(const std::vector<T>& vector, const BinaryAction<T>
& action) {
    T result = vector[0];
    for (int index = 1; index < vector.size(); ++index) {
        result = action(result, vector[index]);
    }
    return result;
```

```
}

int main () {
    std :: vector < int > vector;
    vector.push_back(1);
    vector.push_back(2);
    vector.push_back(3);
    std :: cout << GroupAction(vector, Sum < int > ()) << std :: endl;
    std :: cout << GroupAction(vector, Prod < int > ()) << std :: endl;
}
```

Здесь функторами являются Sum и $Prod$, эти функторы обозначают бинарное действие над двумя переменными. В итоге, вместо того, чтобы писать отдельно функцию, которая считает произведение всех элементов и функцию, которая считает сумму всех элементов, пришлось написать только одну функцию $GroupAction$, которая умеет делать и то и другое, и вообще все, что угодно.

В строчке

$$result = action(result, vector[index]);$$

был вызван $operator()$ у функтора.

Задание 1. По возможности, больше сюда не подсматривая, переписать этот код, чтобы он принимал не вектор, а два итератора на начало и конец куска памяти, где лежат элементы, с которыми надо сделать $GroupAction$.

Алгоритмы сортировки. Рассматривается задача сортировки по возрастанию чисел $a_1, ..., a_n$, заданных в произвольном порядке.

Задание 2. Написать любой алгоритм сортировки(можно даже неоптимальный по времени). При этом вместо сравнения пар чисел должен быть функтор, принимающий два числа и возвращающий истину, если первое число меньше второго и ложь, если это не так.

QSort. Проще всего писать сортировку рекурсивно, используя при этом $QSort$. Идея алгоритма такая: взять самое маленькое из чисел, и самое большое, и посчитать их среднее арифметическое $\frac{a_{min} + a_{max}}{2}$. После этого запихнуть в начало массива числа, меньшие его, в конец массива числа, большие его, то есть сделать так, чтобы числа $a_1, \ldots, a_k$ были меньше $\frac{a_{min} + a_{max}}{2}$, а числа $a_{k+1}, \ldots, a_n$ — больше. Теперь эти два куска массива можно сортировать отдельно, то есть можно вызвать функцию сортировки, которую мы и пишем, рекурсивно.

Задание 3. Сделать $QSort$. Для сравнения также использовать функтор. Написать функцию $Testing$, которая сравнивает работу этого алгоритма с обычным, написанным в задании 2, на случайных массивах чисел.

MergeSort. Тоже рекурсивный алгоритм, только все наоборот — сначала вызовем нашу функцию сортировки рекурсивно, чтобы она отсортировала элементы $a_1, \ldots, a_{n/2}$, потом вызовем еще раз, чтобы отсортировала числа $a_{n/2+1}, \ldots, a_n$. Теперь нужно слить эти два куска в один отсортированный массив, что тоже несложно: храним указатели на начало первого куска и на начало второго куска и применяем следующий алгоритм: возьмем два числа, на которые указывают указатели, сравним их и добавим меньшее из них в результирующий массив, и тот указатель, который указывал на добавленное число, сдвинем на 1, и так пока оба указателя не доедуд до конца обоих участков памяти.

Задание 4. Сделать $MergeSort$. Для сравнения использовать функтор. Написать функцию $Testing$, которая сравнивает работу этого алгоритма с обычным, написанным в задании 2, на случайных массивах чисел.

Сортировки деревьями. Все предложенные выше алгоритмы работают за время, не превосходящее $Cn\log(n)$, где n — размер массива, а C — константа, которая для всех n одинаковая.

Если что-то можно сделать за $\log(n)$, то это намекает на то, что это можно сделать, используя дерево.

Задание 5. Придумать и реализовать алгоритм сортировки, использующий $Splay$-дерево.