Конспект по C++

Черепанов Валерий

18 марта 2016 г.

Часть I
Лекция -1

1 Исправление некоторых неточностей

1.1 reserve

Для вызова `vector.reserve` не требуется вызвать какие-либо конструкторы (даже по умолчанию). Там используется placement new, то есть просто выделяется кусок памяти с помощью new и sizeof, а потом пишется `new(adress) Obj` где `adress` — это выделенная память.

1.2 string

Раньше string старались быть ленивыми и при создании старались сделать ссылку на уже созданное. Но плохо было то, что мы могли с легкостью получить линию при обращении с помощью `[]` (строка копировалась в новую память, чтобы ее можно было модифицировать), да к тому же мы могли инвалидировать итератор.

1.3 Манипуляторы потоков

Манипуляторы вроде `hex` как правило просто выставляют необходимый флаг с помощью `setf`. Для их использования перегружен оператор <<, он принимает поток и указатель на функцию.

2 STL

2.1 Кое-что про map

Как уже говорилось, `map` — это по сути `set` пар. Оператор `[]` у `map` создает новый элемент (вызывая стандартный конструктор), если мы обратились по несуществующему ключу, потому что он не знает, хотим ли мы лишь получить значение по ключу или изменить его.

2.2 Кое-что про set

```
class Person {
    string name;
    int age;

    bool operator <(const Person& p) {
        return name < p.name;
    }
}
```

Если мы создадим `set<Person>`, то элементы будут отсортированы по полю `name`. Но мы можем захотеть сортировать по `age`. Как этого добиться? На помощь приходят функторы!

```
struct by_age {
    bool operator()(const Person& p1, const Person& p2) {
        return p1.age < p2.age;
    }
}

set<Person, by_age> s; // Используем так
```

Внутри это устроено примерно так:

```
template<typename T, class comparator>
class set {
    insert(...) {
        if (comparator()(n1, n2)) { ... } // Анонимный объект
    }
}
```

Если мы не передаем в `set` второй шаблонный параметр, то используется стандартный функтор `less`. Он выглядит примерно так:

```
template<typename T>
struct less {
    bool operator()(const T& t1, const T& t2) {
        return t1 < t2;
    }
}
```

Замечание 1. *`multimap` и `multiset` — это тоже деревья, но обычно в узле хранится список элементов.*

2.3 `algorithm`

Итераторы — обертки над указателями. Их идея заключается в том, что алгоритм может работать с разными структурами данных, поддерживающими одинаковые операции (от структуры требуется лишь предоставить итераторы и методы работы с ними).

Некоторые алгоритмы STL:

1. `swap(T& a, T& b)`, `max(T& a, T& b)`
2. `count[_if](It a, It b, const T& x)`
   `x` — значение в `count`, функция или функтор в `count_if` (можно считать сложные функции или экономить время на сравнение).
3. `equal(It a, It b, It it)` *`// Сравнивает [a, b) с [it, ...)`*
4. `sort`, `min_element`, `nth_element`, `reverse`, `...`

2.4 Чем хороши функторы?

Ну и чем же?

2.5 Чем плохи итераторы?

У них слишком маленький базовый интерфейс (`++`, `--`, `*`, `->`), поэтому они "из коробки" не подходят для адекватной реализации многих алгоритмов (даже для бинарного поиска). Кроме того, мы не знаем тип объекта итератора, а он может нам понадобиться. Для решения этих проблем были придуманты `iterator_traits`.

Как часть решения первой проблемы алгоритмы обарачивают некоторые обращения к итераторам используя функции `distance` и `advance`.

С точки зрения перемещения итераторы бывают:

1. Forward
2. Bidirectional
3. Random Access