Введение в язык C++

Углич П. С.[1]

[1]Южный федеральный университет

Лекция 4

Outline

Векторы

```
#include <vector>
```

Вектор можно объявить следующим образом:

```
std::vector<int> myVector; // мы создали пустой вектор типа
myVector.reserve(10);      // тут мы зарезервировали память
```

Кроме того, вектор можно объявить и в одной строке:

```
std::vector<int> myVector(10);
```

Такой способ объявления вектора не просто выделяет память, но и еще инициализирует все элементы вектора нулями.

Векторы

```
vector<int> myVector2(myVector1);
```

при объявлении второго вектора, копируется - первый.

Проверка наличия элементов

```
vecIntFirst.empty()?cout<<"\n vecIntFirst is empty\n"
:cout<<"vecIntFirst is not empty\n";
```

Векторы

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
vector<int> array1;
// инициализируем элементы вектора array1
array1.push_back(4);
array1.push_back(2);
array1.push_back(1);
vector<int> array2{4,2,1};
if (array1 == array2) {
cout << "array1 == array2" << endl;
}
}
```

Размер вектора

```
#include <iostream>
#include <vector>
#include <iterator> // заголовочный файл итераторов
using namespace std;
int main()
{   vector<int> array1{1,2,3};
    for(int i = 0; i < array1.size(); i++) {
        cout << array1[i] << ' ';
    }
    return 0;
}
```

Очистка вектора

```
    array.clear();
```

Итераторы

```
#include <iostream>
#include <vector>
#include <iterator> // заголовочный файл итераторов
using namespace std;
int main()
{   vector<int> array1; // создаем пустой вектор
    array1.insert(array1.end(), 4);
    array1.insert(array1.end(), 3);
    array1.insert(array1.end(), 1);
    copy( array1.begin(),   // итератор начала массива
          array1.end(),     // итератор конца массива
          ostream_iterator<int>(cout," ")
          //итератор потока вывода
        );
    return 0;
}
```

Итераторы

Объявление итератора

```
vector<int>::iterator it;///создаем итератор it
vector<int>vecInt(10);///выделили место под 10 элементов
cout<<"Vector contains: ";
for(int i=0;i<10;++i) vecInt[i]=i;
 for(it=vecInt.begin();it<vecInt.end();++it)
{
    cout<<*it<<ends;///вывод на экран вектора
}
```

Итераторы

```
it=vecInt.begin();
vecInt.emplace(it,10);
///вставляем нулевым элементом число 10 в вектор
 it=vecInt.begin()+2;
 ///итератор указывает на второй элемент вектора
 vecInt.emplace(it,10);
 ///вставляем вторым элементом число 10 в вектор
 vecInt.emplace_back(10);
 ///вставляем в конец вектора число 10
```

Итераторы

Удаление элементов

```
   vecIntSecond.erase(vecIntSecond.begin()+1);
   ///стираем 1 элемент вектора
   vecIntSecond.erase(vecIntSecond.begin()+2,
vecIntSecond.begin()+6);
   ///стираем 3-6 элементы вектора, 7 элемент не стирается
```

Итераторы

Рассмотрим пример:
Реализовать функцию void IntFilter(vector<int> &vec, Pred p), которая принимает вектор целых чисел и параметр-предикат pred типа typedef bool(*Pred)(int) (предикат одного целочисленного аргумента). Функция преобразует вектор таким образом, что в нем остаются лишь те элементы, которые удовлетворяют предикату p (сохраняя их относительный порядок).

Итераторы

Решение задачи:

```
void IntFilter(vector<int> &vec, Pred p)
{
auto it = vec.begin();
while (it != vec.end())
if (p(*it)) {
it = vec.erase(it);
}
else {
++it;
}
}
```

Итераторы

```
void IntFilter(vector<int> &vec, Pred p)
{
auto it = vec.begin();
while (it != vec.end())
if (p(*it)) {
it = vec.erase(it);
}
else {
++it;
}
}
```

Итераторы

Функция insert

```
vector <int> vecInt(3,100);
 ///Создаем вектор из 3 элементов
 //и заполняем его значением 100
vector <int>::iterator it;
it = vecInt.begin(); ///Итератор указывает на vec[0]
 ///Вектор расширяется теперь до 4 элементов
vecInt.insert (it,200);
 ///И первым элементом записывается 200
```

Итераторы

```
 ///Вектор расширяется теперь до 5 элементов
it = vecInt.begin() + 3;
///Вектор указывает на 4 элемент (0-элемент+3-элемента)
vecInt.insert(it,300);
///И четвертым элементом записывается 300
vecInt.insert(it+1,900);
///Вектор расширяется теперь до 6 элементов
//и 5 элементом записывается 900
```

Итераторы

Функция swap

```
vecFirst.swap(vecSecond);
///меняем местами элементы векторов
swap (vecFirst.back(),vecSecond.back());
///поменяли местами последний элемент в обоих векторах

 cout << "\nVector max_size: " << vecInt.max_size();
 cout << "\nVector size: " << vecInt.size();
 ///Выводим размер вектора
 vecInt.resize(10);
 ///Увеличиваем размер до 10 элементов
vecFirst.shrink_to_fit();
```

Сортировка и удаление дубликатов:

```
 #include <algorithm>
 ...
//сортировка
 std::sort(v_str.begin(), v_str.end());

//сортировка по убыванию
 std::sort(v_str.begin(), v_str.end());
 std::reverse(v_str.begin(), v_str.end());

 //удаление дубликатов
 v_str.erase( std::unique(v_str.begin(), v_str.end())
 , v_str.end() );
```

Сортировка и удаление дубликатов:

```
 // Произвольно перемешиваем элементы
  random_shuffle(numbers.begin(), numbers.end());

// Получаем максимальный элемент, сложность O(n)
vector<int>::const_iterator largest =
max_element( numbers.begin(), numbers.end() );
cout << "Наибольший элемент " << *largest << endl;
cout << "Индекс этого элемента "
<< largest - numbers.begin() << endl;

// Находим позицию цифры 5 в векторе, сложность O(log n)
vector<int>::const_iterator five =
lower_bound(numbers.begin(), numbers.end(), 5);
```

Структуры и классы

Опередение структуры

```
struct <имя структуры> {
<поля и методы>
};
```

Определение класса

```
class <имя класса> {
<поля и методы>
};
```

Способ доступа по умолчанию: в классе - private, в структуре - public;

Структуры и классы

Существует три вида спецификаторов доступа:

- private (закрытый) — доступен только для членов класса;
- protected (защищенный) — доступен для членов класса и их наследников;
- public (открытый) — доступен для всех.

Эти спецификаторы определяют уровень доступа для всех объявлений, следующих за ними. После любого спецификатора должно стоять двоеточие.
В основном поля рекомендуется делать закрытыми, интерфейсные методы — открытыми, а служебные методы — закрытыми.

Структуры и классы

```
class B{
    int i, j, k;
public:
    int f();
    void g();
};
int B::f() {
    return i+j+k;
}
void B::g() {
    i=j=k=0;
}
```

Способ доступа по умолчанию: в классе - private, в структуре - public;

Структуры и классы

Пример: класс для описания геометрической фигуры «круг».
Заголовочный файл

```
//circle.h
#ifndef CIRCLE_H
#define CIRCLE_H
class circle {
private:
    double x,y,r;
public:
    //конструктор без параметров - конструктор по умолчанию
    circle();
    //конструктор с параметрами
    circle(double x1, double y1, double r1);
    //метод для вычисления площади
    double s();
```

Структуры и классы

```
    //методы для установки доступа к закрытым полям класса
    //задает значение радиуса
    void set_r(double r1);
    //получает значение радиуса
    double get_r();
    //метод для сравнения двух объектов на равенство
    bool equal_1(circle a);
    //перегруженная операция для сравнения объектов
    bool operator ==(circle a);
};
#endif
```

Структуры и классы

В случае, если явно не описан ни один конструктор, компилятор автоматически сгенерирует конструктор по умолчанию. Поведение конструктора по умолчанию будет таким же, как если бы он был объявлен явно без списка инициализации и с пустым телом.
Когда программа достигает точки выполнения, в которой определяется объект

```
circle a(3,1,4.2),b(0,0,1);
```

происходит автоматический вызов конструктора.

Структуры и классы

```
circle::circle(){
    x=10;
    y=10;
    r=10;
}
circle::circle(double x1, double y1, double r1){
    x=x1;
    y=y1;
    r=r1;
}
```

Структуры и классы

Списки инициализации

```
class Box {
public:
    Box(int width, int length, int height)
        : m_width(width), m_length(length),
        m_height(height) // member init list
    {}
    int Volume() {return m_width * m_length * m_height; }
private:
    int m_width;
    int m_length;
    int m_height;

};
```

Структуры и классы

Списки инициализации

```
circle(double x1, double y1, double r1):
x(x1),y(y1),z(z1) {}
```

Синтаксис деструктора в целом схож с синтаксисом конструктора: имя функции тоже определяется именем класса. Но чтобы деструктор отличался от конструктора, его имя начинается с префикса

```
~circle()
```

того, деструктор всегда вызывается без аргументов. Конструкторы и деструкторы обладают одной уникальной особенностью: они не имеют возвращаемого значения. В этом они принципиально отличаются от функций, возвращающих пустое значение

Структуры и классы

Для классов, размещающих данные в динамической памяти, необходим конструктор копии (или копирующий конструктор). Он создает копию объекта, передаваемого в конструктор в качестве параметра.

```
Array::Array (const Array &B) { //конструктор копии
    n=B.n;
    A=new int[n];
    for (int i=0; i<n; i++)
        A[i]=B.A[i];
}
```