Лекция 4

Функциональные объекты. Работа с последовательностями

Возвращаясь к алгоритму *sort*

В предыдущей версии алгоритма *sort* предполагалось, что использовался оператор сравнения “меньше чем” <. Рассмотрим алгоритм *sort* с третьим аргументом, который задает функцию сравнения.

```
// dsort1.cpp: Сортировка в нисходящем порядке
#include <iostream>
#include <algorithm>
using namespace std;
bool comparefun(int x, int y)
{   return x > y;
}
```

Функция *comparefun(a[i], a[j])* равняется *true* тогда и только тогда, когда после сортировки *a[i]* должно предшествовать *a[j].* Функции, возвращающие значение типа *bool*, называются *предикатами.*

Возвращаясь к алгоритму *sort*

```
int dsort1()
{   const  int N =  8;
int a[N]={1234,5432,8943,3346,9831,7842,8863,9820};
cout << "Before  sorting: \n";
copy(a, a+N, ostream_iterator<int> (cout, " ") ) ;
cout  <<  endl;
sort (a, a+N, comparefun);
cout << "After sorting in descending order:\n";
copy(a, a+N, ostream_iterator<int>(cout, " "));
cout  <<  endl;
return  0;
}
```

Программа создает следующий вывод:

1234 5432 8943 3346 9831 7842 8863 9820

9831 9820 8943 8863 7842 5432 3346 1234

Функциональные объекты

Функциональным объектом **называется класс, где определен оператор вызова, который записывается как *operator().* От класса не требуется наличия каких-либо других членов.**

```
#include <iostream>
class compare
{       public:
    bool operator() (int x, int y) const
        {   return x > y; }
};
int funobj()
{   compare v;
cout << v(2, 15) << endl; // Вывод:  0
cout << compare()(5, 3) << endl;  // Вывод:  1
cout << endl;
return 0;
}
```

Замечания

В классе *compare* определен оператор вызова функции, *operator(),* с двумя параметрами типа int,=> можно использовать выражение *v(2,15),* где

***v* – переменная этого класса. Это сокращенная форма записи *v.operator()(2,15),* которая приводит к вызову функции-члена *operator()* класса *compare,* возвращающей значение 0, поскольку 2 <15.**

Второй вызов *compare()(5, 3)* выглядит довольно необычно. Первая часть этой записи, *compare(),* представляет собой вызов конструктора по умолчанию класса *compare.* Т.е. выражение *compare()* представляет объект типа *compare,* и за ним,

как и у *v* может следовать список аргументов, в этом примере *(5, 3).*

Использование функционального объекта

```
// dsort2 .cpp: Сортировка в нисходящем порядке
class  compare
{     public:
      bool operator()(int x, int y)const
            {   return x > y; }
};
int dsort2()
{   const int N = 8;
int a[N] ={1234,5432,8943,3346,9831,7842,8863,9820};
cout << "Before sorting:\n";
copy(a, a+N, ostream_iterator<int>(cout,  " "));
cout << endl;
sort (a, a+N, compare());
cout << "After sorting in descending order:\n";
copy(a, a+N, ostream_iterator<int>(cout,  " "));
cout << endl;
return 0; }
```

Работа с последовательностями. Алгоритм *accumulate*

```
// Вычисление суммы элементов последовательности
#include <iostream>
#include <numeric>   // для вычислений
using namespace std;
int accum1_massiv()
{   const int N = 8;
int a[N] = {4, 12, 3, 6, 10, 7, 8, 5}, sum = 0;
sum = accumulate(a, a+N, sum);
cout << "Sum of all elements:  " << sum << endl;
cout << "1000 + a[2] + a[3] + a[4]  = "
 << accumulate(a+2, a+5, 1000) << endl;
return 0;
}
```

Sum of all elements: 55

1000 + a[2] + a[3] + a[4] = 1019

Алгоритм *accumulate*

// Вычисление произведения

Шаблон *multiplies<int>()* аналогичен шаблону *greater<int>().* Мы используем его для вычисления произведения вместо суммы:

```
#include <iostream>
#include <numeric>
#include <algorithm>
#include <functional> // Функциональные объекты,
                      // определенные в STL
using namespace std;
int accum2_massiv()
{   const int N = 4;
int a[N] = {2, 10, 5, 3}, prod =  1;
prod = accumulate(a, a+N, prod, multiplies<int>());
cout << "Product of all elements: " << prod << endl;
return  0;
}  // Вывод программы составляет 300 (=1x2x10x5x3).
```

Алгоритм *accumulate* (с функциональным объектом)

Для заданного массива a, содержащего четыре элемента, вычисляется следующее значение:

1 *a[0] + 2 * a[1] + 4 * a[2] + 8 * a[3]

Кроме функции operator() наш функциональный объект содержит член типа int, который хранит последовательные степени 1, 2, 4 и 8, а также конструктор для инициализации этого члена класса:

```
#include <iostream>
#include <numeric>
using namespace  std;
class  fun
{    public:
         fun(){i = 1;}
         int  operator()(int x, int y)
         {   int u = x + i * y;   i *= 2;   return u; }
     private:
         int  i;
};
```

Алгоритм *accumulate* (с функциональным объектом)

```
int accum3()
{   const int N = 4;
int a[N] = {7, 6, 9, 2}, prod = 0;
prod =  accumulate(a, a+N, prod, fun());
cout << prod << endl;
return  0;
}
```

Эта программа выведет значение:

71 (=1x7 + 2x6 + 4x9 + 8x2)

Алгоритм *count*

count **подсчитывает, какое количество элементов последовательности равно заданному значению.**

```
//  count_e.cpp:  Подсчет количества букв  'e'.
#include <iostream>
#include <string>
#include <algorithm>
using namespace  std;

int count_e()
{   char  *p  =
"This demonstrates the Standard Template Library";
int n = count (p, p + strlen(p),  'e');
cout << n << " occurrences of 'e' found. \n";
return  0;
}
```

Алгоритм *count*

```
    // Сосчитать, сколько раз гласные a, e, i, o, u
//встречаются в заданной строке (первая версия).
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int countvwl()
{   char *p =
"This demonstrates the Standard Template Library",
*q = p + strlen(p) ;
int n = count(p, q, 'a') +  count(p, q, 'e') +
      count(p, q, 'i')  +   count(p, q, 'o') +
      count(p, q, 'u');
cout << n << " vowels (a, e, i, o, u) found. \n";
     // n = 13
     return 0;
}
```

Алгоритм *count_if*

```
    // Сосчитать, сколько раз гласные a, e, i, o, u
//встречаются в заданной строке (улучшенная версия).
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
bool found(char ch)
{   return ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u'; }
int countvw2()
{   char *p =
"This demonstrates the Standard Template Library";
int n = count_if(p, p + strlen(p), found);
cout << n << " vowels (a, e, i, o, u) found.\n";
    // n = 13
return  0;
}
```

Функциональные объекты, определенные в STL

Функции, например *found*, возвращающие *true* *или* *false* в зависимости от соблюдения некоторого условия, называются *предикатами*. Ранее уже встречалось выражение *greater<int>,* которое также является предикатом, но определенным в STL в виде шаблона.

Мы использовали это выражение в вызове:

sort (a, a+N, greater<int> ());

Другой предикат *multiplies<int>()* встречался при подсчёте произведения:

int prod = accumulate (a, a+N, 1, multiplies<int>());

Полный список шаблонов

Шаблоны (определенные в заголовке *functional*), соответствуют стандартным *бинарным операциям*:

plus<T>	**minus<T>**	
multiplies<T>	**divides<T>**	**modulus<T>**
equal_to<T>	**not_equal_to<T>**	
greater<T>	**less<T>**	
greater_equal<T>	**less_equal<T>**	
logical_and<T>	**logical_or<T>**	

Существуют шаблоны, соответствующие унарным операторам - (-x) и (!):

negate<T>	**logical_not<T>**

Все эти шаблоны (с парой скобок ()) являются функциональными объекты, определенными в библиотеке STL, например, *plus<T>()* .

Более сложный пример

Рассмотрим класс *Man*, в котором (кроме прочего) содержится два поля:

- string *name;*

- int *age;*

Создадим вектор *men,* содержащий объекты класса *Man.* Будем производить сортировку по разным критериям.

В классе *Man* определён предикат – операция *operator<(),* в соответствии с которым по умолчанию производится сортировка по возрастанию значений поля *name.*

Также определён функциональный объект *LessAge,* с помощью которого сортируется по возрастанию значение поля *age.*

```
class Man
{
public:    //Man(){ name = ""; age = 0;}
           //Описание пустого конструктора
     Man (string _name, int _age):
     name(_name), age(_age) {}
// Предикат, задающий сортировку по умолчанию
      bool operator<(const Man& m) const
      {   return name < m.name; }
// Далее – оператор вывода
friend ostream& operator<< (ostream&, const Man&);
friend class LessAge;
private:
     string name;
     int age;
};
```

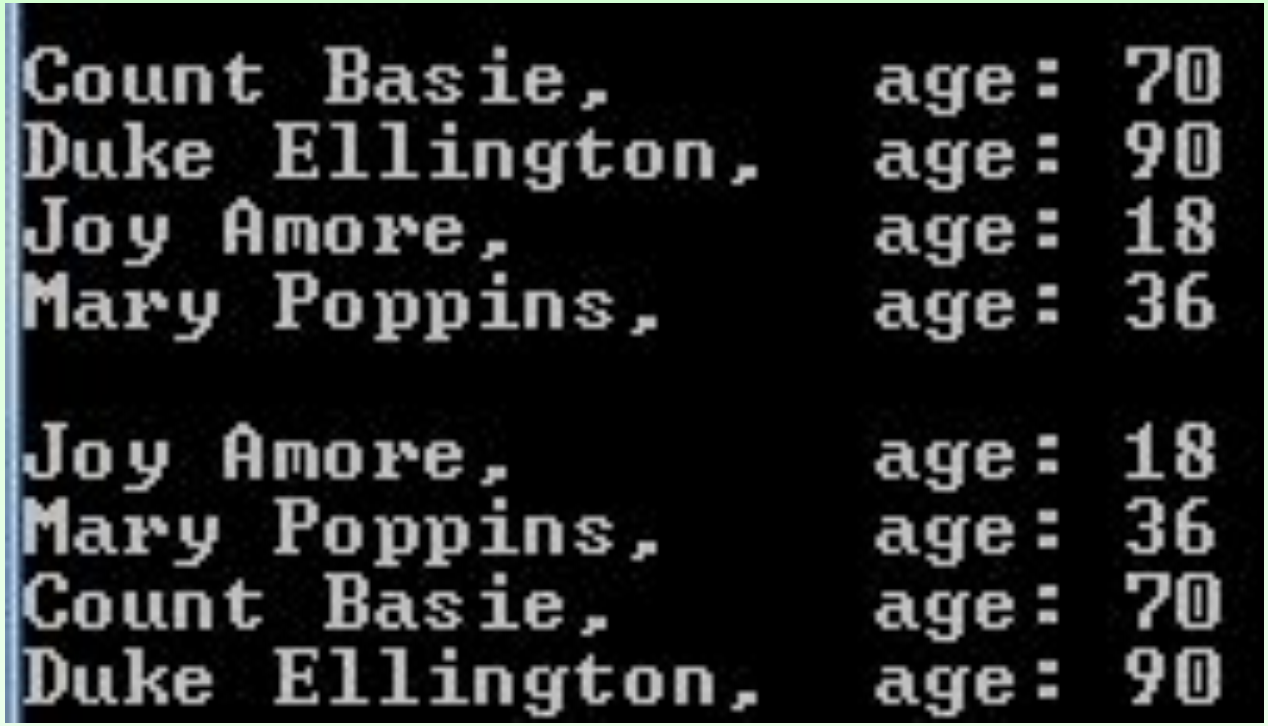
```
// Перегрузка оператора вывода (такой у него синтаксис)
ostream& operator<< (ostream& os, const Man& m)
{
    return os << endl << m.name << ",\t age: " << m.age;
}

// Функциональный класс для сравнения по возрасту
class LessAge
{
public:
    bool operator() (const Man& a, const Man& b)
    {
        return a.age < b.age;
    }
};
```

```
int FunObj_LessAge()
{    int i;
     Man ar[ ] = {Man("Mary Poppins", 36),
                  Man("Count Basie", 70),
                  Man("Duke Ellington", 90),
                  Man("Joy Amore", 18)    };
     int size = sizeof(ar) / sizeof(Man);
     vector<Man> men(ar, ar + size);
     //Сортировка по имени (по умолчанию)
     sort(men.begin(), men.end());
     print(men);
          //Сортировка по возрасту
     sort(men.begin(), men.end(), LessAge());
     print(men);
     return 0;
}
```

```
// Шаблонная функция print() для вывода
//содержимого контейнера
template <class T> void print(T& cont)
{
    typename T::const_iterator p = cont.begin();
    if (cont.empty())
        cout << "Container is empty";
    for (p; p != cont.end(); ++p)
        cout << *p << ' ';
    cout << endl;
 }
```

Эта функция выводит на печать содержимое любого контейнера.

Результат работы:



Алгоритм find_if

Алгоритмы семейства find осуществляют поиск в последовательности заданного значения.

Алгоритмы find_if выполняет поиск значения, заданного с помощью предиката.

В качестве предиката используем функциональный объект.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class In_10_50
{
public:
    bool operator() (int x)
    {return x > 10 && x < 50;}
};
```

Алгоритм find_if

```
int main_find()
{   int a[10] = {43,11,26,79,11,55,31,59,8,0};
    vector<int> v(a,a+10);
    vector<int>::iterator i;
    for (i=v.begin(); i != v.end();  ++i)
    cout << *i << " ";
    cout << endl;
    // Поиск элемента, равного 11
    cout << *find(v.begin(), v.end(), 11) << endl;
    // Поиск элемента по условию 10<x<50
    cout << *find_if(i, v.end(), In_10_50()) << endl;
    return 0;
}
```

find и find_if находят только первый элемент

Результат:

```
43 11 26 79
11 55 31 59 8 0
 11
 43
```

Алгоритм find_if

Модифицируем программу, чтобы найти все элементы, удовлетворяющие условию.

```
int main_find()
{    int a[] = {11,26,79,11,55,31,59,18,20,143};
     int m;   vector<int> v(a,a+10);
     vector<int>::iterator i,k;
     for (i=v.begin(); i != v.end();  ++i)
     cout << *i << " ";   cout << endl;
     // Поиск элемента, равного 31
     cout << *find(v.begin(), v.end(), 31) << endl;
     // Поиск всех элементов при условию 10<x<50
     for (i=v.begin(),m=0; i != v.end();  ++i,++m)
     {    k = find_if(v.begin() + m, v.end(), In_10_50());
          if (k == v.begin() + m)  cout << *k << endl;
          else    cout << "not find" << endl;
     }
return 0;
}
```

Результат работы программы:

```
11 26 79 11 55 31 59 18 20 143
31
11
26
not find
11
not find
31
not find
18
20
not find
Для продолжения нажмите любую клавишу . . .
```