# Объектно-ориентированное программирование


Петрусевич Денис Андреевич

Доцент кафедры Проблем управления

petrusevich@mirea.ru

# Тема 3. Структура STL

Контейнеры

Адаптеры

Итераторы

Алгоритмы

Функторы (и лямбда-выражения)

# Итераторы

Средство для перечисления элементов в контейнере

Функции begin(), end() дают начало и конец контейнера (меньший и больший элемент в случае ассоциативного контейнера)

Итератор *p* перемещается по списку от начала к концу

```
list<char> lst;…
list<char>::iterator p;
p = lst.begin();
while(p!=lst.end())
   {
        cout<<*p<<" ";
        p++;
   }
```

# Итераторы

Итераторы ввода

Итераторы вывода

Однонаправленные итераторы

Двунаправленные итераторы

Итераторы произвольного доступа

Реверсивные итераторы

Итераторы потоков

Итераторы вставки

Константный итератор

# Итераторы

**Операторы:**

- **Оператор *** возвращает элемент, на который в данный момент указывает итератор (*p);
- **Оператор** ++ перемещает итератор к следующему элементу контейнера (p++);
- **Для некоторых итераторов возможен оператор** --, который перемещает итератор к предыдущему элементу контейнера (p--);
- **Операторы ==** и **!=** (p1==p2 или *p1==*p2);
- **Оператор =** (p = ? или *p = ?)

# Итераторы

Важные методы для работы с оператором =:

- **begin()** - итератор начала элементов контейнера.
- **end()** - итератор, направленный на место после последнего элемента в контейнере.
- **cbegin()** - константный (только для чтения) итератор, аналог begin().
- **cend()** - константный (только для чтения) итератор, аналог end().

# Итераторы

**Итераторы вывода**. Вывод элементов массива в поток

```
int Arr[] = {1, 2, 3, 4, 5}; //массив с данными
copy(Arr,Arr+5,ostream_iterator<int>(cout,"\n"));
//с помощью алгоритма выводим элементы массива в поток cout
```

# Итераторы

**Итераторы ввода**. Вводим данные из файла в вектор

```
char FName[]="D:/1.txt"; //Имя файла
   Fname
ifstream f(FName);
vector<char> v
((istreambuf_iterator<char>(f)),
istreambuf_iterator<char>());
```

# Итераторы

Объединение свойств итераторов ввода и вывода – **однонаправленный итератор** (есть операция движения по элементам контейнера в одну сторону: ++)

Если также есть возможность двигаться от текущего элемента к предыдущему (операция --), то **итератор двунаправленный**

```
list<char> lst;…
list<char>::iterator p = lst.begin();
while(p!=lst.end())
   {
        cout<<*p<<" "; p++;
   }
```

# Итераторы

**Итераторы произвольного доступа,** кроме движения вперёд и назад на 1 элемент, позволяют обратиться к произвольному элементу

```
vector <int> v;…
vector <int>::iterator it = v.begin();


cout << *( it + 4 ) << endl;
   //получаем произвольный элемент
   вектора с помощью итератора
   произвольного доступа
```

Получится ли это повторить с помощью итератора по связному списку?

# Итераторы

**Реверсивные операторы** перечисляют элементы от конца контейнера к началу

```
list<char>::reverse_iterator itr_l =
   lst.rbegin();
while (itr_l != lst.rend())
{
        cout << *itr_l << « ";
        itr_l++;
}
```

# Итераторы

**Итераторы вставки** позволяют произвести вставку в текущее место контейнера

```
//вставляем в голову списка lst
  элементы массива Arr
char Arr[] = { 'a', 'b', 'c' };
copy(Arr, Arr + 3,
front_inserter(lst));


//вставляем в хвост списка lst
элементы массива Arr2
char Arr2[] = { 'x', 'y', 'z' };
copy(Arr2, Arr2 + 3,
back_inserter(lst));
```

# Итераторы

**Константный итератор** запрещает изменение данных (но читать можно)

```
//константный итератор по списку
list<int>::const_iterator it;
```

# Функторы

Объект можно использовать как функцию. Лучше применять шаблоны

Сортировка списка *lst* (для сравнения элементов списка используется функция *compare_chars()*):

```
bool compare_chars(char c1, char c2)
{
…
}
…
lst.sort(compare_chars);
```

Предикат: функция возвращает bool.

# Алгоритмы

Часто используемые функции, работающие с разными контейнерами

Поиск; сортировка; подсчёт количества элементов, удовлетворяющих условию; применение ко всем элементам контейнера функции; …

Подключить библиотеку `#include<algorithm>`

Подсчёт количества букв ‘*a*’ или гласных в списке *lst*

```
int n1 = count(lst.begin(), lst.end(),
   'a');
int n2 = count_if(lst.begin(),
   lst.end(), glas);
```

# Алгоритмы

Полезные алгоритмы:

find, find_if

copy

sort

for_each

merge

count, count_if

swap

transform

…

# Алгоритмы

Алгоритм sort() может работать с разными последовательными контейнерами

Объекты в контейнере должны иметь операции сравнения

```
vector<int> vec;
…
sort(vec.begin(), vec.end());
```

# Алгоритмы

Алгоритмы find/find_if ищут определенный элемент или элемент, для которого истинен предикат

```
vector<int> v; …
if(find(v.begin(),  v.end(),  25)  !=
v.end())
{
        // число 25 найдено
}
else
{
         // число 25 не найдено
}
```

# Алгоритмы

Алгоритм for_each обходит элементы контейнера

```
void show(int i) {…}
void update(int &i) {…}


int main()
{
    int M[]={1,2,3,4,5};
    int len = 5;
    for_each(M,M+len,show);
    for_each(M,M+len,update);
    return 0;
}
```

# Алгоритмы

Алгоритм for_each обходит элементы контейнера

```
void show(int i) {…}
int main()
{
   vector <int> v; …
   for_each(v.begin(),v.end(),show);
   return 0;
}
```

# Литература


Шилдт. Самоучитель C++. Глава 14.

В любом учебнике по C++ есть глава, посвященная STL

Примеры работы итераторов:

https://habr.com/ru/post/122283/

https://cpp.com.ru/stl/5.html

http://www.realcoding.net/articles/iteratory.html

http://www.realcoding.net/articles/iteratory-biblioteki-stl.html

https://www.osp.ru/pcworld/1998/06/159178#part_2

https://ci-plus-plus-snachala.ru/?p=298

https://purecodecpp.com/archives/3717

# Спасибо за внимание!

Петрусевич Денис Андреевич

Доцент Кафедры Проблем управления

petrusevich@mirea.ru