# Итераторы

# Итераторы

Обеспечивают:

- Получение значений элементов
- Перемещение по последовательности

# Категории итераторов

| Тип | Дополнительные операции | Какие контейнеры предоставляют |
|---|---|---|
| входной (input) | x = *i;  ++i;  i++; | ??? |
| выходной(output) | *i = x;  ++i;  i++; | ??? |
| прямой или однонаправленный (farward) | x = *i; *i = x;  ++i;  i++; | ??? |
| двунаправленный (bidirectional) | как у farward и<br>–i; i--; | ??? |
| **Произвольного доступа (random access)** | как у bidirectional и<br>i+n; i-n; i+=n; i-=n; i<j; i>j;<br>i<=j; i>=j;<br>При этом: если для вектора желательно сохранить “общность”, то лучше ???,<br>повысить эффективность – ??? | ??? |

# Эмуляция алгоритма copy

Функция должна переписывать значения элементов из **любой** последовательности в **любую** другую

# Эмуляция алгоритма copy

```
template< typename In, typename Out>
                        void copy( In first, In last, Out to)
{
        while(first != last)
        {
                *to = *first;
                ++to;
                ++first;
        }
}
```

# Потоковые итераторы (Stream Iterators )

- #include <iterator>
- Цель - представить входные и выходные потоки как **последовательности,** чтобы можно было использовать потоки ввода/вывода в обобщенных алгоритмах, также как списки, вектора…
- => специализированные **потоковые итераторы**

# ostream_iterator

```
template <class Type, //тип элемента, выводимого в поток
        class CharType= char, // тип кодировки (char, wchar_t)
        class Traits = char_traits<CharType> >
        //специализация для конкретного символьного типа
class ostream_iterator:public iterator<…>{
```

# ostream_iterator

Специфика:

- operator++() – реализован как заглушка
- operator= выводит в поток
- конструктор с одним параметром – принимает в качестве параметра указанный объект потока вывода
- конструктор с двумя параметрами принимает в качестве второго параметра – разделитель (строка!)

# Пример

```
ostream_iterator<int> os(cout);
*os = 1;
//++os;
*os = 2;
```

# Пример – вывод с разделителем

```
ostream_iterator<int> os(cout, " * ");
*os=1;
//++os;
*os=2;
//++os;//1 * 2 *
```

# Пример – вывод с разделителем

```
ostream_iterator<int> os(cout, " * ");
*os=1;
//++os;
*os=2;
//++os;//1 * 2 *
```

# Пример – вывод элементов контейнера с помощью copy()

```
vector<int> v(10,1);
```

???

# Пример – вывод элементов контейнера с помощью copy()

```
vector<int> v(10,1);
ostream_iterator<int>  it(cout, " * ");
copy(v.begin(),  v.end(), it);
```

# Пример – вывод в файл

**vector<int> v;**

//формируем значения

???

# Пример – вывод в файл

```
vector<int> v;
//формируем значения
ofstream f(“my.txt”);
ostream_iterator<int> fit (f, ” “);
copy(v.begin(),  v.end(), fit);
```

# istream_iterator

```
template <  class Type,
        class CharType = char,
        class Traits = char_traits<CharType>,
        class Distance= ptrdiff_t  >
 class istream_iterator :   public iterator<…>
```

# Специфика:

- operator* извлекает очередной объект
- operator++() смещает на следующую позицию (НЕ заглушка!)
- программист должен обеспечить место
- default конструктор формирует признак конца ввода
- конструктор с одним параметром принимает ссылку на объект потокового ввода
- разделители (пробел, tab, enter) отфильтровываются

# Пример

```
istream_iterator<int> it(cin);
int x = *it;
++it;        // подготовили к следующему вводу!
int y = *it;
```

# Вывод значений из файла на консоль

???

# Вывод значений из файла на консоль

```
ifstream f(“my.txt”);
istream_iterator<int> it(f);
while( it != istream_iterator<int> () )
{
        cout<<*it<<‘ ‘;
        ++it;
}
f.close();
```

# Непосредственная работа с буферами ввода/вывода

- ostreambuf_iterator
- istreambuf_iterator

# Итераторы - адаптеры

- Реверсивные итераторы
- Итераторы вставки (в случае необходимости умеют **расширять** контейнер)

# Реверсивные итераторы

**C<T>::reverse_iterator**

**C<T>::const_reverse_iterator**

# Пример

```
vector<int> v;
//формирование значений
…
//С помощью алгоритма copy() вывести значения
//а) – в прямом порядке
…
//б) – в обратном порядке
…
```

# Пример

```
vector<int> v;
//формирование значений
…
ostream_iterator<int> it(cout);
copy(v.begin(),v.end(),it);
cout<<endl;

copy(v.rbegin(),v.rend(),it);
```

# Итераторы вставки
# Insert Iterators

- back_insert_iterator
- front_insert_iterator
- insert_iterator

# Итераторы вставки

Для формирования итераторов вставки STL предоставляет шаблоны глобальных функций:

- **back_inserter(cont& c);**
- **front_inserter(cont& c);**
- **inserter(cont& c, iterator it);**

# Специфика

Операторы вставки при выполнении

*it=<значение>;

Вызывают соответствующий метод контейнера:

- **back**_inserter_iterator- **push_back()**
- **front**_insert_iterator – **push_front()**
- **insert**_iterator – **insert()**

# Пример

```
vector<int> v(10);
back_insert_iterator<vector<int>> it =
        back_inserter(v);
 *it = 1; //???
// ++it; //заглушка
 *it = 2; //???
```

# Пример

```
int ar[5] = {1,2,3,4,5};
vector<int> v = {5,6,7};

//добавить значения из массива в вектор (copy())
                    ...
```

# Пример

```
int ar[5] = {1,2,3,4,5};
vector<int> v;

copy(ar, ar+5, back_inserter(v));

//в обратном порядке?
copy(ar, ar+5, front_inserter(v));  //???
```

# Пример

```
int ar[5] = {1,2,3,4,5};
list<int> l;
copy( ar, ar+2, front_inserter(l) ); //???
```

# Пример

```
int ar[5] = {1,2,3,4,5};
list<int> l(4,5);
list<int>::iterator it = l.begin();
++it;
copy(ar,ar+2,inserter(l,it)); //???
```

# Пример

vector<int> v;

//С помощью алгоритма copy() ввести значения. Формирование признака окончания ввода – Ctrl+D

…

# Защита от некорректного ввода

```
cin.exceptions(ios::failbit);
istream_iterator<int> is(cin);
vector<int> v;
try{
        copy(is, istream_iterator<int>(), ???);
}
catch(ios_base::failure& f)
{
        cout<<f.what();
        while(cin.rdbuf()->in_avail()) { cin.get(); }
        cin.clear();
}
cin.exceptions(ios::goodbit);
```

# Чтение из файла

???

# Пример – чтение из файла

vector<int> v;

ifstream f(“my.txt”); //пусть в файле хранятся целые

//с помощью алгоритма copy() считать данные из файла в вектор

???

# Пример – чтение из файла

```
vector<int> v;
ifstream f(“my.txt”); //пусть в файле хранятся целые


copy(istream_iterator<int>(f),
        istream_iterator<int>(),
        back_inserter(v));
f.close();
```

# Чтение из файла текста

```
ifstream f(“my.txt”);
vector<int> v(istreambuf_iterator<int> (f),
                istreambuf_iterator<int> () );
```

# Пример

vector<int> v;

//С помощью алгоритма copy() ввести значения. Формирование признака окончания ввода – Ctrl+D

…

# Пример

```
vector<int> v;
istream_iterator<int> is(cin);

copy(is, istream_iterator<int>(), back_inserter(v));

cin.clear(); //!!! сброс флагов состояния
while (cin.rdbuf()->in_avail()) //fflush(stdin);
{
        char c = cin.get();
}
```

#include <algorithm>

namespace std

# ОБОБЩЕННЫЕ АЛГОРИТМЫ СПОСОБЫ ЗАДАНИЯ ПРЕДИКАТОВ

Полубенцева М.И.

# Определение

В контексте стандартной библиотеки C++ **обобщенный алгоритм** –

**это шаблон функции, которая умеет работать с разными источниками данных:**

- Контейнерами
- Массивами
- Потоками ввода/вывода (в частности с файлами)

# Цели обобщения

- Реализация наиболее часто встречающихся задач (итерация по последовательности, поиск, сортировка, трансформация…) независимо от внутренней реализации
- Распространить обобщение на работу с разными источниками данных

# Средства реализации

- Шаблоны функций
- Итераторы
- Предикаты
- Псевдонимы типов

# for_each()

Инкапсулирует цикл:

- перебирает элементы последовательности
- и вызывает заданный предикат, отправляя в качестве параметра очередной элемент последовательности

# for_each()

???

# Предикат – глобальная функция

- без модификации последовательности

```
int main()
{
        vector<int> v;
        //заполнение вектора
        //Вывести куб каждого элемента последовательности ???
}
```

# Предикат – глобальная функция

```
void PrintCube(int n) { cout << n * n * n << ‘ ‘ ; }

int main()
{
        vector<int> v;
        //заполнение вектора
        for_each(v.begin(), v.end(), PrintCube) ;
}
```

# Предикат – глобальная функция

- модификация последовательности

```
int main()
{
        list<Point> l;
        //заполнение списка
        //Поменять знак координат ???
}
```

# Предикат – глобальная функция

```
void Neg(Point& pt)
{
        pt=-pt;
}

int main()
{
        list<Point> l;
        //заполнение списка
        for_each(l1.begin(), l1.end(), Neg);
}
```

# Предикат – шаблон глобальной функции

???

# Предикат – шаблон глобальной функции

```
template<typename T> void Neg(T& t)
{
	t=-t;
}

int main()
{
	list<Point> l;
	//заполнение списка
	for_each(l.begin(), l.end(), Neg<Point>);
	int ar[] = {5,-3,-10,33};
	for_each(ar, ar+sizeof(ar)/sizeof(int), Neg<int>);

}
```

# find(), find_if()

Найти элемент последовательности, значение которого равно значению параметра

???

# find()

```
int main()
{
        vector<Point> v;
        …
        vector<Point>::iterator it = find(v.begin(),
                                    v.end(),Point(1,1));
        //что должно быть определено в классе Point ???
        if(???) cout<<”Not found”;
}
```

# Эмуляция алгоритма find_if()

???

# Пример использования алгоритма find_if()

Найти **первую** точку (Point), которая находятся в окружности с радиусом ==10 от начала координат

# Пример

```
bool FindPoint(const Point& pt){
        int tmp = pt.m_x*pt.m_x + pt.m_y*pt.m_y;
        return tmp < 10*10;
}
 int main()
{
   vector<Point> v;
        //…
   vector<Point>::iterator it =  find_if(v.begin(), v.end(),
                                        FindPoint) ;
    if(<такой точки нет>)        …
}
```

# Пример использования алгоритма find_if()

Найти **все** точки (Point), которые находятся в окружности с радиусом ==r от начала координат

# Пример использования алгоритма find_if()

```
 int main()
{
        vector<Point> v;
        //сформировали значения
        list<Point>  l; //для результата
        auto it = v.begin();
        while(it != v.end())
        {
                it= find_if(it, v.end(), FindPoint) ;
                if(it == v.end() ) break;
                 l.push_back(*it);
        }
        …
}
```

# count(), count_if()

# Эмуляция count_if()

???

# Пример count_if(). Предикат – функциональный объект

Преимущество функционального объекта – можно запаковать **любое** количество информации

Задание: посчитать точки, которые отстоят меньше, чем на заданное значение от начала координат

???

# Предикат – функциональный объект

```
class COUNT_IF{
        int m_d;
public:
        COUNT_IF(int d){m_d = d;}
        bool operator()(const Point& pt){
                        return …<m_d*m_d);}
};
```

# Предикат – функциональный объект

```
vector<Point> v;
//…
size_t n = count_if(v.begin(), v.end(),
                    COUNT_IF(2));
```

# Алгоритмы копирования

copy()

copy_if()

# copy()

```
vector <int> v1, v2;
    //Заполнили значениями v1;

//copy( v1.begin( ), v1.begin( ) + 3, v2.begin( )  );

copy( v1.begin( ), v1.begin( ) + 3,...);//???
```

# copy()

```
vector <int> v1, v2;
   //Заполнили значениями v1;

//copy( v1.begin( ), v1.begin( ) + 3, v2.begin( )  );

copy( v1.begin( ), v1.begin( ) + 3, front_inserter(v2));//???
copy( v1.begin( ), v1.begin( ) + 3, inserter(v2,v2.begin( )));
copy( v1.begin( ), v1.begin( ) + 3, back_inserter(v2));
```

# Реализация copy_if()

???

# Пример использования copy_if() Предикат – шаблон класса

- скопировать все элементы, кроме совпадающих с указанным значением

# Пример использования copy_if(). Предикат – шаблон класса

```
template<typename T> class COPY_IF{
        T m_t;
public:
        COPY_IF(const T& t) : m_t(t) {}
        bool operator()(const T& t)
                {return t!=m_t;}
};
```

# Пример использования copy_if(). Предикат – шаблон класса

```
vector<int> vi;
int ar[] = {1,5,-6,1};
//скопировать все значения, кроме «1»
```

# Пример использования copy_if(). Предикат – шаблон класса

```
//скопировать из массива в вектор
        vector<int> vi;
        int ar[] = {1,5,-6,1};
        copy_if(ar, ar+sizeof(ar)/sizeof(int),
                ???,
                COPY_IF<int>(1));

 //скопировать из вектора в список
        vector<Point> vPt; //сформировали значения
        list<Point>  l;
        copy_if(v.begin(), v.end(),
                ???,
                COPY_IF<Point>(Point(1,1))); //value_type
```

# Пример использования copy_if(). Предикат – шаблон класса

```
vector<Point> v;
//сформировать значения

//вывести на печать, кроме Point(1,1)
```

# Пример использования copy_if(). Предикат – шаблон класса

```
vector<Point> v;
//сформировать значения

//вывести на печать, кроме Point(1,1)
copy_if(v.begin(), v.end(),
        ostream_iterator<Point>(cout, " "),
        COPY_IF<Point>(Point(1,1)));
```

# Пример использования copy_if(). Предикат – шаблон класса

```
vector<int> v;
ifstream file("test.txt"); //в файле - целые

//скопировать все значения, кроме 33
```

???

# Пример использования copy_if(). Предикат – шаблон класса

```
vector<int> v;
ifstream file(“test.txt”); //в файле - целые
copy_if(istream_iterator<int> (file),
                    istream_iterator<int>(),
inserter(v1,v1.begin()), COPY_IF<int>(33));
```

# sort()

- Два перегруженных варианта
- Без предиката:
  - сортирует последовательность в порядке возрастания значений элементов
  - чтобы алгоритм работал с пользовательскими типами, в классе должен быть перегружен operator<
- Замечание: для списка сортировка реализована методом класса.

# Пример sort()

```
int ar[5] = {-5,8,1,-10,5};
sort(ar, ar+5);

vector<int> v(ar, ar+5);
sort(v.begin(),v.end()); //в возрастающем порядке
//в убывающем порядке???
sort(???);
```

# Пример

```
vector<Point> v;
//…

//Отсортировать по удалению от x,y
```

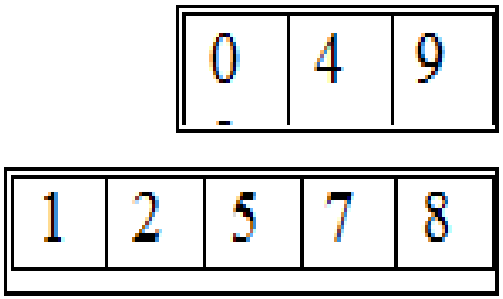
# Предикат

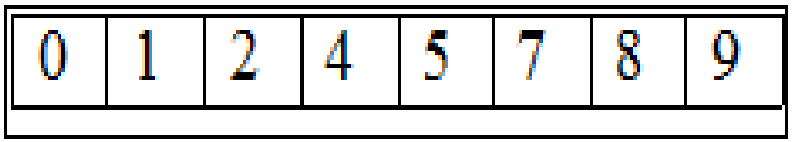
```
class CMP_IF{//по удалению от x,y
        int x,y;
public:
  CMP_IF(int a, int b){x=a; y=b;}
  bool operator()(const Point& pt1, const Point& pt2)const
        {
                return …
        }
};

sort(v.begin(),v.end(),CMP_IF(2,2));
```

# merge()

объединение двух **отсортированных**! последовательностей



# merge()

```
vector<int> v;
//…
sort(…);

int ar[5] = {1,2,-5,7,8};
sort(…);

//объединить в список ???
```

# merge()

```
vector<int> v;
//…
sort(…);

int ar[5] = {1,2,5,7,8};
sort(…);
```

а)
```
list<int> l (обеспечить достаточное место);
merge(v.begin(), v.end(), ar, ar+5, ???);//режим замены
```

б)
```
list<int> l;//пустой список
merge(v.begin(), v.end(), ar, ar+5, ???);//режим вставки
```

# merge()

```
vector<int> v;
//…
sort(…);

int ar[5] = {1,2,5,7,8};
sort(…);
```

а)
```
list<int> l(v.size() + sizeof(ar)/sizeof(int));//места достаточно
merge(v.begin(), v.end(), ar, ar+5, l.begin());//режим замены
```

б)
```
list<int> l;//пустой список
merge(v.begin(), v.end(), ar, ar+5, inserter(l,l.begin()));//режим вставки
```

# transform()

```
template<class FROM, class TO, class UnaryF>
TO transform(FROM First1, I FROM Last1,
                          TO Result, UnaryF f );

template<class FROM1, class FROM2, class TO,
class BinaryF>  TO
transform(FROM1 First1, FROM1 Last1,
                  FROM2 First2, TO Result, BinaryF f );
```

# Трансформация одной последовательности в другую

```
int a[] = {2,-5,…};
int b[<столько же>];
//Требуется: b[i] = -a[i];

transform(a, a+sizeof(a)/sizeof(int), b,
                                        negate<int>() );
```

# Трансформация двух последовательностей в третью

```
vector<int> v;
//сформировали значения
list<int> l;
//сформировали значения
deque<int> d;
//требуется: d[i] = v[i] + l[i];
//сколько элементов можно использовать?
transform(…, plus<int>());
```

# Задание предиката

vector<double> v;

//сформировали значения

list<double> l;

//требуется: l[i] = A*v[i]*v[i] + B*v[i] +C;

# Предикат – метод класса

mem_fun – для последовательностей, в которых хранятся указатели

mem_fun_ref – для последовательностей, в которых хранятся копии объектов

# mem_fun и mem_fun_ref

mem_fun_ref – это шаблон функции, которая:

- Принимает в качестве параметра указатель на метод класса
- А возвращает объект типа mem_fun_ref_t, в котором:
  - Сохраняется указатель на метод класса
  - Перегружен operator()

# Пример mem_fun и mem_fun_ref

```
class Point{
        int m_x, m_y;
public:
        Point (int x=0, int y=0){m_x = x; m_y = y;}
        bool MEM_F(){return (m_x<0  || m_y<0);}
};

        Point* arptr[] = {new Point(), new Point(1,2), new Point(3,4)};
        int n=count_if(arptr, arptr+sizeof(arptr)/sizeof(Point*),
                                        mem_fun(&Point::MEM_F));
        Point arpt [] = { Point(), Point(1,2), Point(3,4)};
        n = count_if(arpt, arpt+sizeof(arpt)/sizeof(Point),
                                mem_fun_ref(&Point::MEM_F));
```

# unary_function и binary_function

- Цель - шаблон вводит стандартные имена для типов аргументов
- Такой шаблон можно использовать в качестве базы для переопределения оператора ()

# unary_function

```
template<class Arg, class Result>
    struct unary_function {
    typedef Arg argument_type;
    typedef Result result_type;
    };
```

=> все унарные функции могут обращаться к своему единственному аргументу стандартным образом как к argument_type**,** а возвращать значение как result_type

# Пример

```
class MatchString : public unary_function<string, bool>{
    string m_string;
public:
    MatchString(const string& ss) : m_string(ss){}
    result_type operator()
                (const argument_type& str) const
                    {return m_string==str;}
};
```

# Пример

```
vector<string> strVect;
strVect.push_back("She");
strVect.push_back("Sells");
strVect.push_back("Sea");
strVect.push_back("Shells");
strVect.push_back("by");
strVect.push_back("the");
strVect.push_back("Sea");
strVect.push_back("Shore");

int result = count_if(strVect.begin(), strVect.end(),
                                        MatchString("Sea"));
```

# binary_function

template<class Arg1, class Arg2, class Result>
struct **binary_function** {
typedef Arg1 **first_argument_type**;
typedef Arg2 **second_argument_type**;
typedef Result **result_type**;
};

- Такой шаблон служит в качестве базы для переопределения оператора «()» в Вашем классе следующим образом: result_type operator()(first_argument_type, second_argument_type)
- => все бинарные функции могут обращаться к своему первому аргументу как к first_argument_type, ко второму как second_argument_type, а возвращать значение как result_type.

# bind1st и bind2nd

int ar[10] = {…};

int n = count_if(ar, ar+10, bind1st(less<int>(),5));

//создает объект binder1st, в котором сохраняется:

- объект типа less - pr
- 5 - val
- и перегружен

bool operator()( эл-т_последовательности_ x) {return pr(val, x);}