

# Standard Template Library (STL)

Введение:

Обработка данных

# Понятия данных и обработки данных

*Данные* — зарегистрированная информация, представление фактов, понятий или инструкций в форме, приемлемой для общения, интерпретации, или обработки человеком или с помощью автоматических средств.

*Обработка данных* — [data processing, information processing] процесс приведения данных к виду, удобному для использования.

*Информационная технология* — это процесс, использующий совокупность средств и методов сбора, обработки и передачи данных для получения информации нового качества о состоянии объекта, процесса или явления.

# Области применения технологий обработки данных

список примеров:

Астрономия

Бухгалтерский учёт

Биотехнологии

Издательское дело

Компьютерная графика

Криптография

Уфология

Экспериментальная

психология

а также:

Нанотехнологии

Обработка результатов  
экспериментов

Обработка сигналов

Обучение

Прикладная статистика

Экономическая  
кибернетика

*Несложно назвать еще много примеров поскольку это -  
практически все области жизни современного  
общества*

Standard Template Library (**STL**)

Стандартная библиотека  
шаблонов

Краткий обзор библиотеки

# В библиотеке STL выделяют пять основных компонентов:

1. **Контейнер** (англ. *container*) — хранение набора объектов в памяти.
2. **Итератор** (англ. *iterator*) — обеспечение средств доступа к содержимому контейнера.
3. **Алгоритм** (англ. *algorithm*) — определение вычислительной процедуры.
4. **Адаптер** (англ. *adaptor*) — адаптация компонентов для обеспечения различного интерфейса.
5. **Функциональный объект** (англ. *functor*) — сокрытие функции в объекте для использования другими компонентами.

# Контейнеры STL

Наиболее часто используемым функционалом STL являются контейнерные классы («контейнеры»).

Контейнеры STL делятся на три основные категории:

Последовательные контейнеры

Ассоциативные контейнеры

Адаптеры

# Последовательные контейнеры

реализуют структуры данных с  
возможностью последовательного доступа к  
НИМ.

**vector** - динамический непрерывный массив

**list** - список

**deque** - очередь

Определяющая характеристика: можно  
вставить свой элемент в любое место  
контейнера.

# Некоторые особенности последовательных контейнеров

**Вектор (vector)** представляет собой тип последовательного контейнера, который используется в большинстве случаев.

**Список (list)** используется при частых операциях вставки и удаления в произвольной позиции.

**Дек (deque)** выбирается в случае, если удалений нет, а вставки производится только в конце последовательности элементов».



# Вектора в C++

# Вектор создание и инициализация

Создание вектора определенного типа  
(синтаксис):

`vector<тип_элемента> имя_вектора;`

Пример:

```
#include <vector>
```

Подключение  
библиотеки

```
using namespace std;
```

Объявление  
пространства  
имен

```
vector<int> V; //пустой вектор
```

```
vector<int> V(10); // вектор размером 10  
//элементов
```

```
vector <int> V(10, 0); //вектор размером 10  
//элементов равных 0
```

# Методы класса vector

`push_back()` - добавление нового элемента в конец вектора;

`size()` – определение размера вектора (количество элементов);

`pop_back()` — удалить последний элемент;

`clear()` — удалить все элементы вектора;

`empty()` — проверить вектор на пустоту.

Для доступа к элементам вектора можно использовать квадратные скобки [], также, как для обычных массивов

## Пример процедуры вывода на консоль элементов вектора

Размер вектора определяется методом `size()`:

**`имя_вектора.size();`**

Пример: **`int n=V.size();`**

**//Пример процедуры вывода вектора**

```
void pr_vec(vector<int> A)
{
    for(int i = 0; i < A.size(); i++)
        cout << A[i] << ' ';
        cout<<endl;
}
```



# Некоторые операции с векторами

Изменение размера вектора (количества элементов):

**имя\_вектора.resize(новый\_размер);**

Добавить новый элемент в конец вектора:

**имя\_вектора.push\_back(новый\_элемент);**

# Пример: играем с размерами вектора

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main()
6 {
7 //setlocale(LC_ALL, "Russian");
8 setlocale(0, "");
9 vector<int>V(3); //Создаем вектор 3 элемента
10 cout<<"размер вектора "<<V.size()<<endl;
11 V.resize(10); //Увеличиваем размер до 10 элементов
12 cout<<"размер вектора "<<V.size()<<endl;
13 V.resize(7); //Уменьшаем размер до 7 элементов
14 cout<<"размер вектора "<<V.size()<<endl;
15 V.push_back(7); //Добавляем в конец 1 элемент
16 cout<<"размер вектора "<<V.size()<<endl;
17 system ("pause");
18 }
```

# Значения вектора



```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 void pr_vec(vector<int> A)
5 {.....}
6
7 int main()
8 {
9  setlocale(0, "");
10 vector<int>V(3,1); //Создаем вектор 3 элемента
11  pr_vec(V);
12 V.resize(10); //Увеличиваем размер до 10 элементов
13  pr_vec(V);
14 V.resize(7); //Уменьшаем размер до 7 элементов
15  pr_vec(V);
16 V.push_back(7); //Добавляем в конец 1 элемент
17  pr_vec(V);
18  system ("pause");
19 }
```

Показать

# Итератор ы



# Итераторы

**Итератор** — нечто, указывающее на элемент вектора.

Объявление итератора:

```
vector<тип>::iterator имя_итератора;
```

Итератор **имя\_вектора.begin()** указывает на первый элемент соответствующего вектора.

Итератор **имя\_вектора.end()** указывает на фиктивный элемент вектора, расположенный за последним его элементом

# Действия с итераторами

Для итератора на *vector* вы можете:

- Выполнять операцию разыменования (обращаться к значению элемента на которое указывает итератор), как мы это делали с указателем: `int x = *it;`
- Использовать инкремент (`it++`, `++it`) и декремент (`it--`, `--it`).
- Применять арифметические операции. Например, сместить итератор на пять элементов вправо, вот так: `it += 5;`
- Сравнивать на равенство `if (it == it2) { ...`
- Передать переменной разницу итераторов  
`int x = it - it2;`

# Алгоритмы

# Использование итераторов

Требования:

```
#include <vector>
```

Пространство имен: `std`

Векторы и итераторы позволяют воспользоваться большим количеством библиотечных функций, которые называются алгоритмами

```
#include <algorithm>
```

# Функции для тестовой программы

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <algorithm>
5  using namespace std;
```

## Формирование вектора из файла

```
15 void vec_from_file(vector<int> &A)
16 { ifstream file("vec-file.txt");
17   if (!file.is_open()) cout <<"Error"<<endl;
18   int value;
19   while ( file >> value ) A.push_back(value); }
20 //
```

## Вывод вектора по итератору

```
7 void print(vector<int> A)
8 {
9   vector<int>::iterator it;
10  for(it = A.begin(); it != A.end(); ++it)
11    cout<<*it<<" ";
12  cout<<endl;
13 }
```

## Начало тестовой программы

```
21 int main()
22 {
23     system("chcp 1251"); system("cls");
24     vector<int>V;
25     vec_from_file(V);
26     vector<int>::iterator it,it1;
27     print(V);
```

9 2 3 4 7 6 12 8

## Минимальный и максимальный элементы

```
28     it = min_element(V.begin(), V.end());
29     cout<<"минимальный элемент " << *it << " ";
30     cout<<"его номер " << it-V.begin() << "\n";
31     it = max_element(V.begin(), V.end());
32     cout <<"максимальный элемент " << *it << " ";
33     cout<<"его номер " << it-V.begin() << "\n";
```

минимальный элемент 2 его номер 1  
максимальный элемент 12 его номер 6

Функции возвращают итератор.

## Поиск заданного элемента

```
34 | it = find(V.begin(), V.end(), 7); // ищем в векторе число 7
35 | cout<<"нашли число 7 ";
36 | cout<<"его номер "<<it-V.begin()<<"\n";
```

```
нашли число 7 его номер 4
```

## Вставка элементов в вектор:

**имя\_вектора.insert(куда, что)**

Здесь:

**куда** — итератор, указывающий на элемент, непосредственно перед которым вставляется новый элемент.

**что** — элемент, который нужно вставить.

```
37 | it=V.insert(it, 21); //Вставка элемента
38 | print(V);
39 | cout<<"итератор на номере "<<it-V.begin()<<"\n";
```

```
9 2 3 4 21 7 6 12 8
итератор на номере 4
```

**insert** возвращает итератор, указывающий на только что вставленный элемент.

## Удаление заданного элемента

```
40 |     it+=2; V.erase(it);  
41 |     print(V);
```

```
9 2 3 4 21 7 6 12 8  
итератор на номере 4  
9 2 3 4 21 7 12 8
```

## Сортировка и реверс

```
42 |     sort(V.begin(), V.end());  
43 |     print(V);  
44 |     reverse(V.begin(), V.end());  
45 |     print(V);
```

```
2 3 4 7 8 9 12 21  
21 12 9 8 7 4 3 2
```



## Удаление диапазона элементов

```
46 | it=V.begin()+2; it1=V.end()-2;  
47 | V.erase(it,it1); //Удалить диапазон значений  
48 | print(V);
```

21 12 9 8 7 4 3 2

21 12 3 2

## Удаление элементов вектора по значению:

имя\_вектора.erase( remove(имя\_вектора.begin(),  
имя\_вектора.end(), значение), имя\_вектора.end());

```
25 | vec_from_file(V);  
26 | print(V);  
27 | V.erase(remove(V.begin(),V.end(),3),V.end());  
28 | print(V);
```

9 2 3 4 7 3 12 8  
9 2 4 7 12 8

# Потоковые итераторы

Суть применения потоковых итераторов в том, что они превращают любой поток в итератор, используемый точно так же, как и прочие итераторы: перемещаясь по цепочке данных, считывает значения объектов или присваивает им другие значения.

Практически итератор потока вывода используется для отображения данных на экране.

Итератор потока ввода — это удобный программный интерфейс, обеспечивающий доступ к любому потоку, из которого требуется считать данные.

Потоковые итераторы имеют одно существенное ограничение — в них нельзя возвратиться к предыдущему элементу. Единственный способ сделать это - заново создать итератор потока.

# Потоковые итераторы

являются либо итератором **ВХОДНОГО ПОТОКА**, либо итератором **ВЫХОДНОГО ПОТОКА**.

Классы для этих итераторов:

*istream\_iterator* и *ostream\_iterator*.

Примеры :

**istream\_iterator cin\_it (cin)** - это итератор для потока **cin**.

**ostream\_iterator cout\_it (cout)** - это итератор для потока **cout**.

**ostream\_iterator cout\_it (cout, " ")** - это итератор для потока **cout** с разделителем.

# Примеры использования потоковых итераторов



Алгоритм: вводим целые числа

Они после ввода отображаются на экране (выводятся).

Если введено число 111, работа программы завершается.

```
1 #include <iostream>
2 #include <vector>
3 #include <iterator>
4 using namespace std;
5
6 int main() {
7     system("chcp 1251"); system("cls");
8     cout<<"введите число ";
9     istream_iterator<int> cin_it (cin);
10    ostream_iterator<int> cout_it (cout, " еще число ");
11    //копирование из потока ввода в поток вывода
12    while (( *cin_it) != 111)
13    {
14        *cout_it = *cin_it;
15        cin_it++; cout_it++;
16    }
17    system("pause"); return 0;
18 }
```

# Функция сору в С++

Часто приходится вывести некоторое количество элементов или добавить ячейки из одного контейнера в другой. Часто для это используется цикл, но есть средство лучше — метод **сору()**.

**сору** — это метод который имеет три области применения.

- Первая — это выводить элементы от n-го итератора до j.
- Вторая — это добавлять элементы из контейнера А в контейнер В.
- Третья — это копирование диапазона ячеек и вставка его позицию Х.

Синтаксис:

**сору**(<первый>, <последний>, <операции>);

**Первый** и **последний** — это диапазон ячеек, с которыми будут выполняться операции

В **операции** входит:

[вывод элементов](#)

[добавление элементов](#)

[копирование элементов](#)

# Примеры использования потоковых итераторов



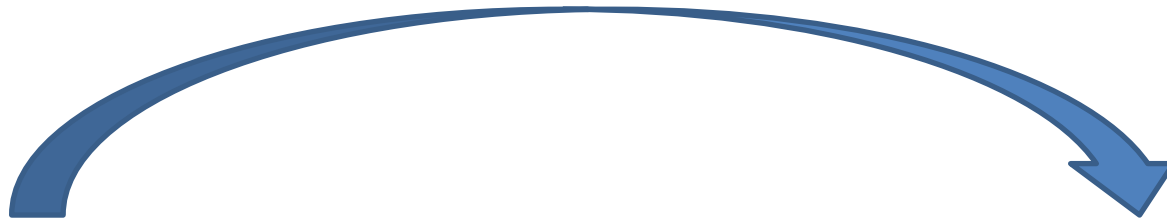
```
1 //Копирование данных из входного потока в выходной поток
2 #include <iostream>
3 #include <iterator>
4 using namespace std;
5 int main()
6 {
7     system("chcp 1251"); system("cls");
8     // Потоковые итераторы
9     istream_iterator<int> cin_it(cin); //пот.итератор ввода
10    istream_iterator<int> eos; //конец ввода
11    ostream_iterator<int> cout_it(cout, " "); //пот.итератор вывода
12    copy(cin_it, eos, cout_it);
13    system("pause"); return 0;
14 }
```

Имена

По умолчанию eos = ^Z (Ctrl Z)

## Добавление элементов в вектор

`inserter` — специальный тип итератора вывода который добавляет элементы из контейнера А в контейнер В.

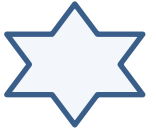


```
copy(<первый>, <последний>, inserter(<имя контейнера>, <итератор>));
```

`back_inserter` — добавляет значения в конец STL контейнера.

`front_inserter` — добавляет элементы в начала контейнера  
(не работает с вектором)

# Вывод и ввод вектора без цикла



Начало то же, только добавить

```
#include <vector>
```

```
10 // Создаем потоковые итераторы
11 istream_iterator<int> cin_it(cin); //пот.итератор ввода
12 istream_iterator<int> eos; //конец ввода
13 ostream_iterator<int> cout_it(cout, " "); //пот.итератор вывода
14
15 vector<int>V; //Создаем пустой вектор
16 //Копируем из входного потока в вектор(Ctrl-z)
17 copy(cin_it, eos, back_inserter(V));
18
19 //Вывод вектора на экран
20 copy(V.begin(), V.end(), cout_it);
```




# Копирование элементов из одного вектора в другой



```
9   vector<int>V1(3,1),V2(4,2); //Создаем 2 вектора
10  // Создаем потоковый итератор
11  ostream_iterator<int> cout_it(cout, " "); //пот.итератор вывода
12  //выводим вектора
13  copy(V1.begin(), V1.end(), cout_it);
14  cout<<"\n";
15  copy(V2.begin(), V2.end(), cout_it);
16  cout<<"\n";
17  //добавляем V1 к V2 после второго элемента
18  copy(V1.begin(), V1.end(), inserter(V2,V2.begin()+2));
19  //Выводим V2
20  copy(V2.begin(), V2.end(), cout_it);
```

# Добавление элементов в конец вектора

```
9   vector<int>V1(5,1),V2(4,2); //Создаем 2 вектора
10  // Создаем потоковый итератор
11  ostream_iterator<int> cout_it(cout, " "); //пот.итератор вывода
12  //выводим вектора
13  copy(V1.begin(), V1.end(), cout_it);
14  cout<<"\n";
15  copy(V2.begin(), V2.end(), cout_it);
16  cout<<"\n";
17  //добавляем часть вектора V1 в конец V2
18  copy(V1.begin()+1, V1.end()-1,back_inserter(V2));
19  //Выводим V2
20  copy(V2.begin(), V2.end(), cout_it);
21  system ("pause");
22  return 0;
23 }
```



## Копирование элементов в другой вектор

Для вставки копируемых элементов в контейнер нужно третьим аргументом передавать — итератор. От него начнут изменяться ячейки на значения другого контейнера.

```
17 //Заменяем часть вектора V2 элементами вектора V1
18 copy(V1.begin(), V1.end(), V2.begin()+6);
```



## Доступ к элементам вектора

```
8 vector<int> V(5, 9);
9 ostream_iterator<int> cout_it(cout, " "); //пот.итератор вывода
10 copy(V.begin(), V.end(), cout_it); cout<<"\n";
11 V.at(3) = 5; // изменяем значение одного элемента
12 copy(V.begin(), V.end(), cout_it);
```

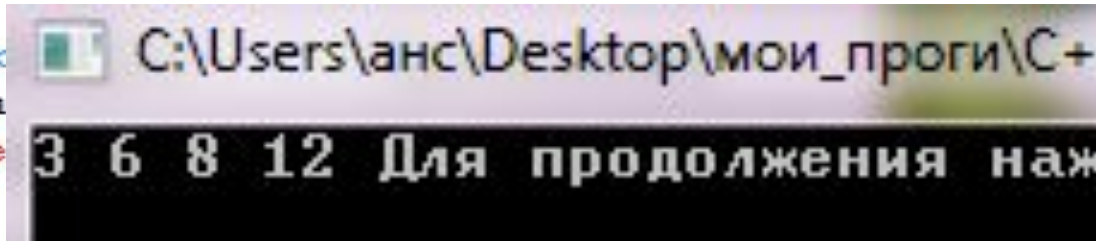




# Работа с файлом



```
1 #include <fstream>
2 #include <iostream>
3 #include <iterator>
4 #include <vector>
5 using namespace std;
6 int main()
7 {
8     setlocale(0, "");
9     vector<int> V;
10    ifstream fin("input_int.txt");
11    // Потокoвые итераторы
12    istream_iterator<int> fin_it(fin);
13    istream_iterator<int> eos;
14    ostream_iterator<int> cout_it(cout, " ");
15
16    // Копируем элементы из файла в вектор
17    copy(fin_it, eos, back_inserter(V));
18
19    // Сортировка вектора
20    sort(V.begin(), V.end());
21
22    // Вывод вектора
23    copy(V.begin(), V.end(), cout_it);
24    system("pause");
25    return 0;
26 }
```



**На сегодня**



**всё**