

# Библиотека STL

Классы `string`, `vector`



# Что такое STL?

- STL - это библиотека стандартных шаблонов. Она содержит **часто встречающиеся способы организации данных** - так называемые контейнеры: динамические массивы, двунаправленные списки, стеки и др
- STL содержит множество **часто встречающихся алгоритмов**: сортировка (как на всем множестве, так и на части его), нахождение минимального и максимального значений и др.
- Каждый такой алгоритм работает **с разными типами контейнеров**. Т. е. вы, например, можете использовать один и тот же алгоритм сортировки как для динамического массива, так и для стека.
- STL состоит из **трёх частей** : контейнеры, алгоритмы и итераторы

# Контейнеры

- Первая часть - это динамические массивы, списки, очереди и др.
- Другая часть **ассоциативные** контейнеры. Основная их отличительная черта - это то, что хранящиеся в них значения ищутся по **ключам**. При этом ключ может быть самым разным. Аналогия такого контейнера из жизни - это телефонная книга. Там номера телефонов ищутся по фамилии владельца или названия фирмы.
- В каждом контейнере кроме собственно данных есть **методы для работы** с этими данными (для добавления, поиска, удаления и др.).

# Алгоритмы

- Алгоритмы не являются частью контейнеров, а образуют отдельную подсистему.
- Почти любой алгоритм может применяться к почти любому контейнеру. Вызывая метод для некоторого алгоритма, мы вызываем этот метод сам по себе, а не для экземпляра некоторого класса. Контейнер же, к которому применяется алгоритм, передается в качестве **параметра**.

# Итераторы

- В первом приближении итератор - это некоторый указатель, который может оббегать все элементы контейнера.
- Итераторы играют такую же роль, что и **индекс** у элемента массива. Через индекс массива мы можем получить некоторый элемент массива, и через итератор мы можем получить некоторый элемент контейнера.
- Итераторы бывают **разных типов**: для движения только вперед, для движения в обе стороны и др.
- В случае с указателями, добраться до элемента контейнера можно через **разыменованный** итератор

# Класс vector

- Класс vector является **динамическим** одномерным массивом - т. е. вы можете добавлять в него элементы, удалять их и т. п.
- С данным классом используются `push_back`, `pop_back`, `clear` и `empty`. Для доступа к отдельным элементам вектора используется оператор `[]` - как и для элементов массива.

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
void main()
```

```
{ vector <int> k; // Объявление вектора из целых.  
  // В конец вектора добавляем элементы  
  k.push_back(22);    k.push_back(11);  
  k.push_back(4);
```

# Класс vector

```
// Печать элементов вектора
for (int i = 0; i < k.size(); i++)
    cout << k[i] << "\n";
cout << "***\n";
k.pop_back(); // Удаление элемента с конца
вектора.
// Печать элементов вектора.
for (i = 0; i < k.size(); i++)
    cout << k[i] << "\n";
cout << "***\n";
k.clear(); // Удаление всех элементов вектора
if(k.empty) // Проверка, что вектор пуст.
    cout << "Vector is empty\n";
}
```

## Пример создания вектора

```
#include <iostream>
#include <vector>
using namespace std;
int main(){ // Вектор из 10 элементов типа int
vector<int> v1(10);
/* Вектор из элементов типа float с
неопределенным размером*/
vector<float> v2;
/* Вектор, состоящий из 10 элементов типа int по
умолчанию все элементы заполняются нулями */
vector<int> v3(10, 0);
return 0;}
```



## Методы класса *vector*

- Для добавления нового элемента в конец вектора используется метод `push_back()`. Количество элементов определяется методом `size()`. Для доступа к элементам вектора можно использовать квадратные скобки `[]`, также, как и для **обычных массивов**.
- `pop_back()` — удалить последний элемент
- `clear()` — удалить все элементы вектора
- `empty()` — проверить вектор на пустоту

# *Управление элементами вектора*

Создание вектора, в котором содержится произвольное количество фамилий студентов.

# Класс string

- Класс string предназначен для работы со строками. Он находится в пространстве имен **std** и для его использования надо подключить string.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{  string s0 = "abcde";
   string s1 = " fg";
.....
```

# Класс string

- Работать со строками через класс string достаточно удобно - вы можете делать конкатенацию (сложение) строк с помощью обычного оператора +, можете брать символ в определенном месте строки с помощью оператора [] (или другим способом - с помощью метода at), можете использовать привычные операторы =, ==, != для присваивания и сравнения строк. Также имеются методы для получения длины строки, для выяснения, не пустая ли это строка и др.
- С помощью метода getline можно прочесть строку из определенного потока (с клавиатуры).

# Примеры использования

- *// Конкатенация строк.*  
`string s = s0 + s1; cout<<s<<"\n";`
- *// Получаем символ на определенном месте.*  
`char ch0 = s0.at(1); cout<<ch0<<"\n";`
- `char ch1 = s0[3]; cout<<ch1<<"\n";`
- *// Выясняем, не пустая ли строка.*
- `if (s0.empty()) cout << "String is empty"<<"\n";`
- `else cout << "String isn't empty"<<"\n";`

# Примеры использования

- *// Обмен значения двух строк.*

```
swap(s0, s1);
```

- *// Присваивание и сравнение 2 строк.*

```
s1 = s0;
```

```
if(s1 == s0) cout << "Strings are equal" << "\n";
```

```
else cout << "Strings are not equal" << "\n";
```

- *// Чтение введенной с клавиатуры строки.*

```
getline(cin, s1); cout << s1;
```

- *// Получение длины строки.*

```
cout << s1.length();
```

# Управление элементами вектора

Создание вектора, в котором содержится произвольное количество фамилий студентов.

```
#include <iostream>
#include <vector>
#include <string>
Using namespace std
int main()
{ // Поддержка кириллицы в консоли Windows
  setlocale(LC_ALL, "");
  // Создание вектора из строк
  vector<string> students; // Буфер для ввода фамилии
  студента
  string buffer = "";
  cout << "Вводите фамилии студентов. " << "По окончании
  ввода введите пустую строку" << endl;
```

```
do {getline(cin, buffer);
    if (buffer.size() > 0)
        /*Добавление элемента в конец вектора */
        students.push_back(buffer);
}
while (buffer != ""); /* Сохраняем количество
элементов вектора */
    unsigned int vector_size = students.size();
// Вывод заполненного вектора на экран
cout << "Ваш вектор." << endl;
for (int i = 0; i < vector_size; i++)
    cout << students[i] << endl;
return 0;}
```