ТЕМА 9 – КЛАССЫ STRING, FSTREAM.

КЛАСС **STRING**

В C++ существует два типа строк:

```
1. char s[10];  // массив символов
```

Переменная хранит в себе только 1 символ, элементы массива – отдельные объекты, сложно работать со строками переменной

```
2. string s;  // символьная строка
```

Это специальный класс **string**
Для его подключения в начале программы нужно подключить :
#include <string>

```
char cs[ ] = "Napoleon";  // C-string
string s = "Napoleon";    // C++ - string

cout << s << " has " <<  s.length() << " characters.\n";
s.replace(5, 2,"ia");  //changes s to "Napolian
```

Символьные строки

Начальное значение:

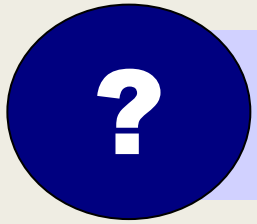
```
string s = "Привет!";
```

Присваивание:

```
s = "Привет!";
```

Вывод на экран:

```
cout << s;
```



А если массив символов?

Символьные строки

Ввод с клавиатуры:

```
cin >> s;
```

только до пробела!

```
getline ( cin, s );
```

до перевода строки (Enter)

Отдельный символ:

```
s[4] = 'a';
```

Символы в строке нумеруются с нуля!

Длина строки:

```
int n;
...
n = s.size();
```

метод для объектов типа `string`

Символьные строки

Задача: заменить в строке все буквы 'а' на буквы 'б'.

```
#include <iostream>
using namespace std;
int main()
{
  setlocale(0,"rus");
   string s;

   cout << "Введите строку: ";
   getline ( cin, s );
   for (int i = 0; i < s.size(); i++ )
      if ( s[i] == 'a' )
         s[i] = 'b';
   cout << s;
}
```

цикл по всем символам строки

Операции со строками

Конструкторы строк	**string()** - конструктор по умолчанию (без параметров) создает пустую строку. **string(string & S)** - копия строки S **string(size_t n, char c)** - повторение символа с заданное число n раз. **string(size_t c)** - строка из одного символа с.
s.append(str)	добавляет в конец строки строку str. Можно писать как s.append(переменная), так и s.append("строка")
s.assign(str)	присваивает строке s значение строки str. Аналогично записи **s=str**
s.clear()	как следует из названия, очищает строку. Т.е. удаляет все элементы в ней
s.compare(str)	сравнивает строку s со строкой str и возвращает 0 в случае совпадения (на самом деле сравнивает коды символов и возвращает их разность) **Перегруженные операторы: ==, !=** - посимвольное сравнение. **<, >, <=, >=** - лексикографическое сравнение.
s.copy **(куда, сколько, начиная с какого)**	- копирует из строки s в куда (там может быть как строка типа стринг, так и строка типа char). Последние 2 параметра не обязательные (можно использовать функцию с 1,2 или 3 параметрами)

Операции со строками

bool b=s.empty()	если строка пуста, возвращает true, иначе false
s.erase(откуда, сколько)	удаляет n элементов с заданной позиции
s.find(str,позиция)	ищет строку str начиная с заданной позиции
s.insert(позиция,str, начиная beg, count)	вставляет в строку s начиная с заданной позиции часть строки str начиная с позиции beg и вставляя count символов
int len=s.length()	записывает в len длину строки
s.push_back(symbol)	добавляет в конец строки символ
s.replace(index, n,str)	берет n первых символов из str и заменяет символы строки s на них, начиная с позиции index
str=s.substr(n,m)	возвращает m символов начиная с позиции n
s.swap(str)	меняет содержимое s и str местами
s.size()	возвращает число элементов в строке

Операции со строками

Объединение (*конкатенация*):

```
string s, s1, s2;
s1 = "Привет";
s2 = "Вася";
s = s1 + ", " + s2 + "!";
```

"Привет, Вася!"

Срез (подстрока):

```
s = "0123456789";
s1 = s.substr( 3, 5 );    // «23456"
```

откуда

с какого символа

сколько символов

5

```
s = "0123456789";
s1 = s.substr( 3 );    // "3456789"
```

Операции со строками

Удаление:

```
s = "0123456789";
s.erase ( 3, 6 ); // "0129"
```

с какого символа

сколько символов

Вставка:

```
s = "0123456789";
s.insert( 3,"ABC" ); // "012ABC3456789"
```

куда

с какого символа

что

Поиск символа в строке

```
string s = "Здесь был Вася.";
int n;
n = s.find( 'c' );  // 3
```

find – искать

Вернёт -1, если не нашли!

```
if ( n >= 0 )
      cout << "Номер символа 'c': "
           << n << endl;
else cout << "Символ не найден.\n";
```

Поиск подстроки

```
string s = "Здесь был Вася.";
int n;
n = s.find( "Вася" );  // 10
```

```
if ( n >= 0 )
  cout << "Слово начинается с s["
       << n << "]\n";
else
  cout << "Слово не найдено.\n";
```

`s.rfind()` – поиск с конца строки!

Пример обработки строк

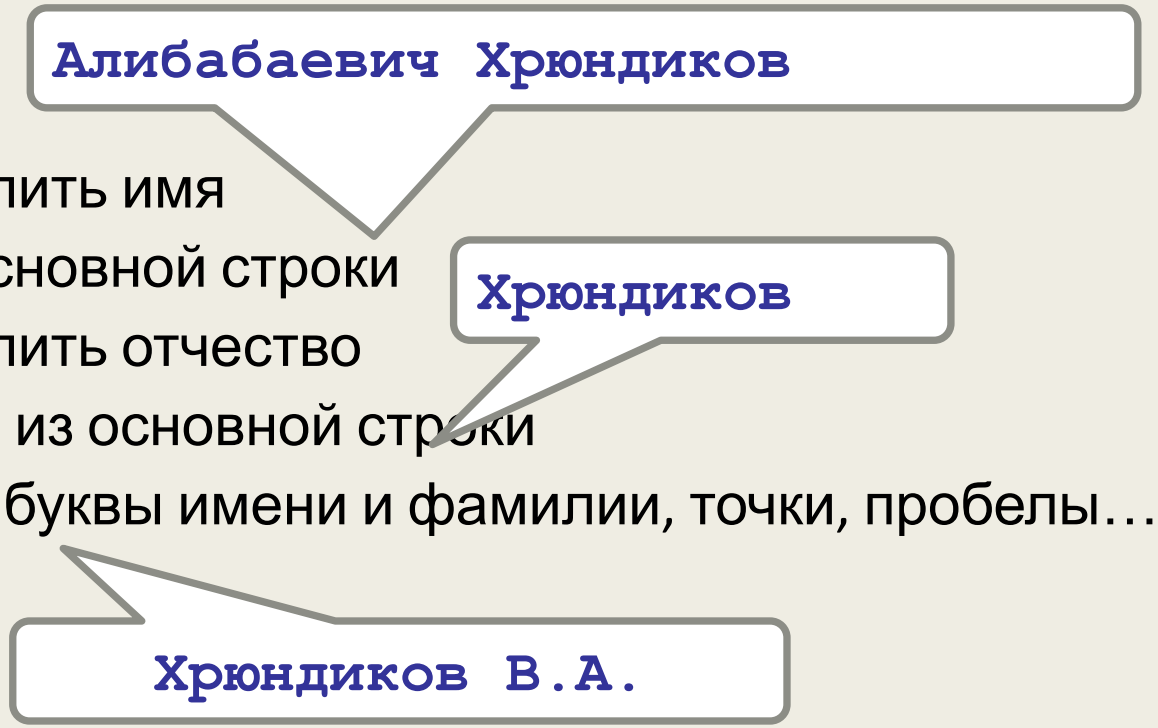
Задача: Ввести имя, отчество и фамилию. Преобразовать их к формату «фамилия-инициалы».

Пример:

```
Введите имя, отчество и фамилию:
Василий Алибабаевич Хрюндиков
Результат:
Хрюндиков В.А.
```

Алгоритм:

- найти первый пробел и выделить имя
- удалить имя с пробелом из основной строки
- найти первый пробел и выделить отчество
- удалить отчество с пробелом из основной строки
- «сцепить» фамилию, первые буквы имени и фамилии, точки, пробелы…



Пример обработки строк

```
main()
{
  string s, name, name2;
  int n;
  cout << "Введите имя, отчество и фамилию: ";
  //Татьяна Степановна Лапшун
  getline ( cin, s );
  name = s.substr(0,1) + '.';// начало имени    - Т.
  n = s.find(' ');           // найти пробел    - 7
  s = s.substr( n+1 );       // удалить имя     - Степановна Лапшун
  n = s.find(' ');           // найти пробел    - 10
  name2 = s.substr(0,1) + '.';// начало отчества - С.
  s = s.substr ( n+1 );      // осталась фамилия - Лапшун
  s = s + ' ' + name + name2;   // результат       = Лапшун Т. С.
  cout << s;
}
```

Задачи

1: Ввести адрес файла и «разобрать» его на части, разделенные знаком `'/'`. Каждую часть вывести в отдельной строке.

Пример:

```
Введите адрес файла:C:/Фото/2013/Поход/vasya.jpg
C:
Фото
2013
Поход
vasya.jpg
```

2: Напишите программу, которая заменяет во всей строке одну последовательность символов на другую.

Пример:

```
Введите строку: (X > 0) and (Y < X) and (Z > Y) and (Z<>5)
Что меняем: and
Чем заменить: &&
Результат
(X > 0) && (Y < X) && (Z > Y) && (Z<>5)
```

Преобразования «строка» – «число»

Из строки в число:

```
string s="123";
int N;
N=atoi( s.c_str() );     // N = 123
```

«12x3» → 12

в строку языка Си

```
string s="123.456";
float X;
X=atof( s.c_str() );   // X = 123.456
```

Преобразования «строка» – «число»

Из числа в строку:

Идея: направить выходной поток в строку!

```
#include <sstream>
```

строковые потоки

```
ostringstream ss;
string s;
int N=123;
ss << N;
s=ss.str();    // s = "123"
```

строковый поток вывода

из потока в строку

Преобразования «строка» – «число»

Вещественное число в строку:

```
ostringstream ss;
string s;
double X=123.456;
ss.width(10);     // ширина поля
ss.precision(3); // знаков в дробной части
ss << X;
s=ss.str();       // s ="   123.456"
```

Научный формат:

```
ss.str("");       // очистка потока
ss.width(10);     // ширина поля
ss.precision(6); // знаков в дробной части
ss << scientific << X; // научный формат
s=ss.str();       // s = "1.234560E+002"
```

Задачи

«A»: Напишите программу, которая вычисляет сумму трех чисел, введенную в форме символьной строки. Все числа целые.

Пример:

```
Введите выражение:
12+3+45
Ответ: 60
```

«B»: Напишите программу, которая вычисляет выражение, состоящее из трех чисел и двух знаков (допускаются только знаки «+» или «–»). Выражение вводится как символьная строка, все числа целые.

Пример:

```
Введите выражение:
12-3+45
Ответ: 54
```

КЛАСС **FSTREAM**

1. Открытие и закрытие файла

Для выполнения операций ввода-вывода над файлами необходимо включить в программу заголовок:

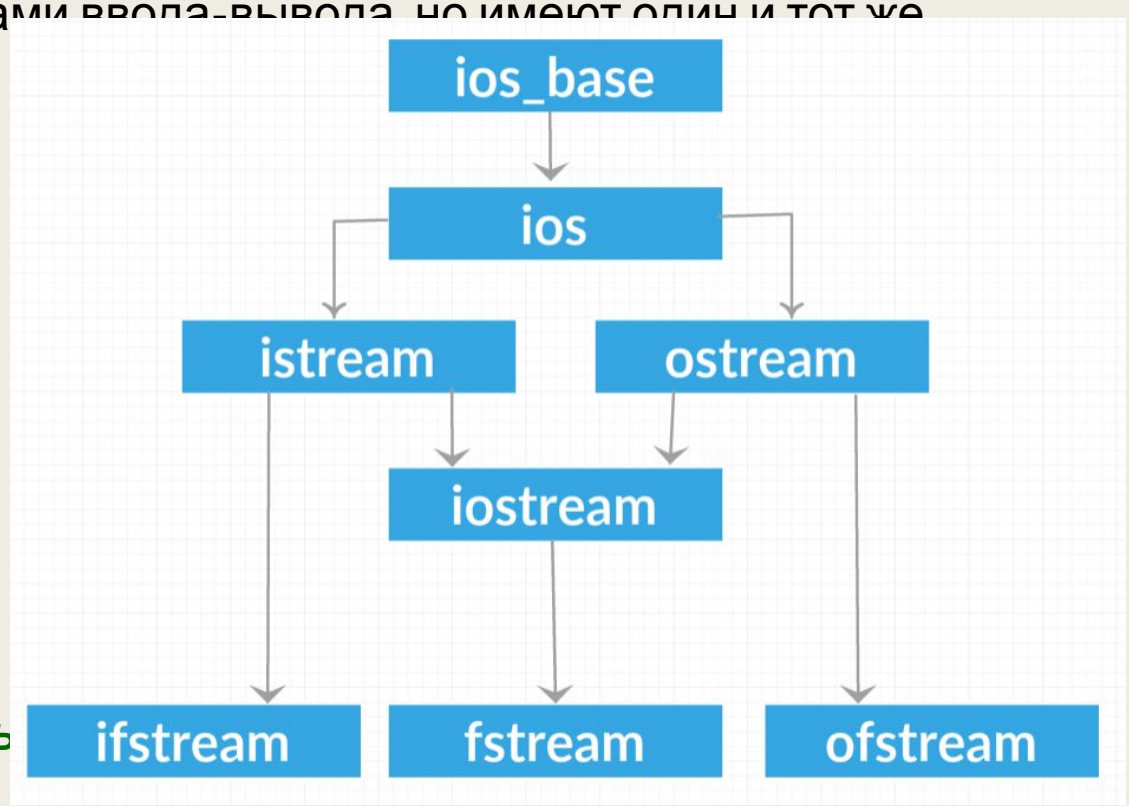
```
#include <fstream>
```

В C++ ввод-вывод осуществляется посредством логического интерфейса, называемого потоком.

Потоки связываются с различными устройствами ввода-вывода, но имеют один и тот же программный интерфейс.

Существуют три типа потоков:

- ввода (класс **ifstream**),
- вывода (класс **ofstream**) и
- ввода – вывода (класс **fstream**).



Пример создания потоков

(имена потоков задаются программистом):

```
ifstream in;   // in   - поток ввода/чтение
ofstream out;      // out  - поток вывода/запись
fstream both;// both - поток ввода-вывода
```

Созданный поток необходимо связать с файлом с помощью функции **open( )**.

Примеры:

```
in.open( "input.txt" );       //  для ввода
out.open( "output.txt" );      //   для  вывода
both.open( "inout.txt" ); //   для ввода-вывода
```

Можно совместить описание потоков с операцией связывания с файлом (обычно так и записывают):

```
ifstream  in( "input.txt" );          //  для ввода
ofstream  out( "output.txt" );          //   для  вывода
fstream both ("inout.txt" );        //   для ввода-вывода
```

Если функция open( ) завершилась неудачно, то поток имеет значение false.

Поэтому можно проконтролировать успешность открытия файла:

```
if( !inp )
        cout << "Нет файла\n";
else
        cout << "Файл открыт\n";
```

Для закрытия файла используется функция close( ).

```
inp.close( ); // закрытие файла
out.close( ); // закрытие файла
```

Запись данных в файл:

```
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
    setlocale(0, "");
    double  q = 1.23e-25,  p = 45.6;
    char ch = 'Z' ;
    char str[20] = "Строка";
    int i = 789;

    ofstream out("output.txt"); // создание и открытие файла для записи
    if(!out)     // проверка успешности открытия файла
       {cout << "Нет файла"; return 1;}
    out << q << '\n' << p << '\n'  << ch << "  " << str << "  " << i; // запись данных в файл
    out.close( );    // закрытие файла

    return 0;
}
```

В результате в файл **output.txt** будет записан

```
1.23e-25
45.6
Z   Строка   789
```

Модификатор ate сохраняет содержимое файла, если он существует, и устанавливает указатель в конец файла при открытии;
app - то же самое, с той лишь разницей, что устанавливает указатель в конец файла перед каждой записью в файл (т.е. записывать можно только в конец файла).

Дополнение информации в файл

```
// открытие файла для дополнения
    ofstream out("output.txt", ios::app);
```

Режимы открытия

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове метода `open()`.

```
ofstream fout("file.txt", ios::app);
fout.open("file.txt", ios::app);
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции **ИЛИ** `|`, например:

`ios::out | ios::in` - открытие файла для записи и чтения.

Константа	Описание
`ios::in`	открыть файл для чтения
`ios::out`	открыть файл для записи
`ios::ate`	при открытии переместить указатель в конец файла
`ios::app`	открыть файл для записи в конец файла
`ios::trunc`	удалить содержимое файла, если он существует
`ios::binary`	открытие файла в двоичном режиме

Чтение данных из файла:

```
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
    setlocale(0, "");
    double  q, p;
    char ch ;     char str[20];   int i;

    ifstream inp("Input.txt"); // открытие файла для чтения
    if(!inp)      // проверка успешности открытия файла
      { cout << "Нет файла"; return 1;}
    inp >> q >> p >> ch >>  str >> i; // ввод информации из файла
    inp.close( );     // закрытие файла

    // вывод данных на экран для контроля
    cout << " q=" << q << "  p =" << p <<  "  ch = " << ch
         <<  "  str = " << str << "  i = " << i <<endl;
    return 0;
}
```

```
1.23e-25
45.6
Z   Строка  789
```

```
q=1.23e-025   p =45.6   ch = Z   str = Строка   i = 789
```

Механизм чтения данных из файла

- Смещаем текущую позицию на начало читаемой порции;
- Читаем очередную порцию

После чтения порции A

После чтения порции B

Чтение текста вместе пробелами:

Если строка читается с помощью операции **>>** , то чтение прекращается на первом пробеле.

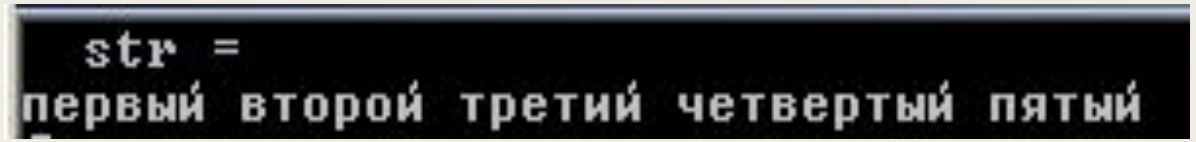
Чтение строк с пробелами осуществляется функцией:

getline(имя_потока, строковая_переменная**)**

```
#include <iostream>   #include <fstream>     #include <string>
using namespace std;
int main( )
{
      setlocale(0,"");
      string  str;

      ifstream inp("Input.txt"); // открытие файла для чтения
      if(!inp)    cout << "Нет файла";
      getline(inp, str); // ввод информации из файла
      inp.close( );    // закрытие файла

      // вывод данных на экран для контроля
      cout   <<  "  str = \n" << str <<endl;
      return 0;
}
```



Если в файле записано: «первый второй третий четвертый пятый» то будет выведено:

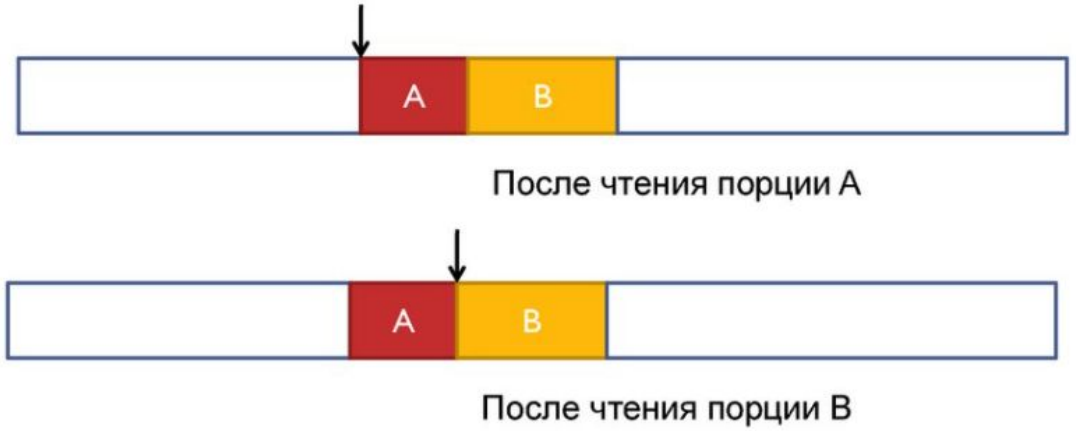
Обнаружение конца файла:

используется функция логического типа **eof( )**

Функция возвращает **true**, если считан символ окончания файла,
и **false** в противном случае.

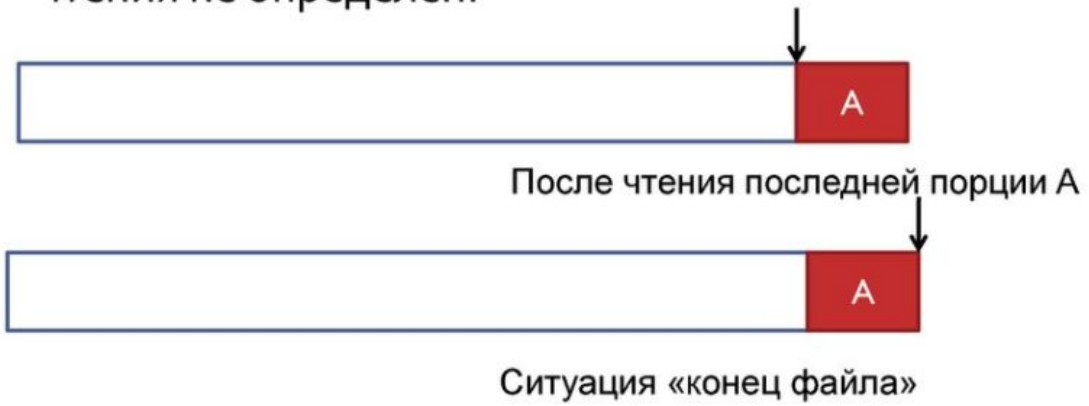
Механизм чтения данных из файла

- Смещаем текущую позицию на начало читаемой порции;
- Читаем очередную порцию



Определение конца файла

- Прочитав последнюю порцию данных, мы ещё не знаем, что она последняя!
- Попытка следующего чтения вызывает ситуацию «конец файла». Это – ещё не ошибка, но результат чтения не определён!



Программа для чтения из файла всех строк до окончания файла

```
#include <iostream>          #include <fstream>      #include <string>
using namespace std;

int main( ){
        setlocale(0,"");
        string str[30];  // символьный массив для хранения 30 строк

        ifstream inp( "Input.txt " ); // открытие файла для чтения
        if(!inp)      cout << "Нет файла";
        int k = 0;  // номер строки
        while( !inp.eof( ) )      // цикл до окончания файла
        {
                getline(inp, str[k]);   // чтение строки вместе с пробелами с извлечением
                 k++; // увеличение номера строки на единицу
        }
        inp.close( );    // закрытие файла

        // вывод данных на экран для контроля
        for(int i = 0;   i < k;   i++)  // здесь k кол-во строк, а i номер строки
                cout   << str[i] <<endl;
        return 0;
}
```