Тема: Потоки ввода/вывода

- Ввод/вывод в **C** и **C++**
- Предопределенные объекты
- Иерархия потоков
- Классы ***ios, ostream, istream***
- Классы ***ofstream, ifstream***
- Ввод/вывод для класса ***String***

Ввод/вывод в **C** и **C++**

- Стандартная библиотека ввода/вывода **C**
  - большой набор функций (трудно запомнить)
  - нет контроля типов при форматном вводе/выводе (***printf, scanf***)
  - трудно реализовать для нового типа
  - стандартные потоки ***stdin, stdout, stderr***
- Стандартная библиотека классов потоков ввода/вывода **C++**
  - перегруженные операции “**>>**” и “**<<**”
  - контроль типов
  - легко наращивать для новых типов (классов)
  - стандартные потоки ***cin, cout, cerr, clog***

Предопределенные объекты-потоки

- Стандартная библиотека классов потоков ***iostream*** имеет четыре предопределенных объекта, ассоциированных со стандартным вводом/выводом

  cin – стандартный ввод (клавиатура)

  cout – стандартный вывод (экран монитора)

  cerr – стандартное устройство ошибок (экран) с небуферизованным выводом

  clog – стандартное устройство ошибок (экран) с буферизованным выводом

Операции помещения в поток и извлечения из потока

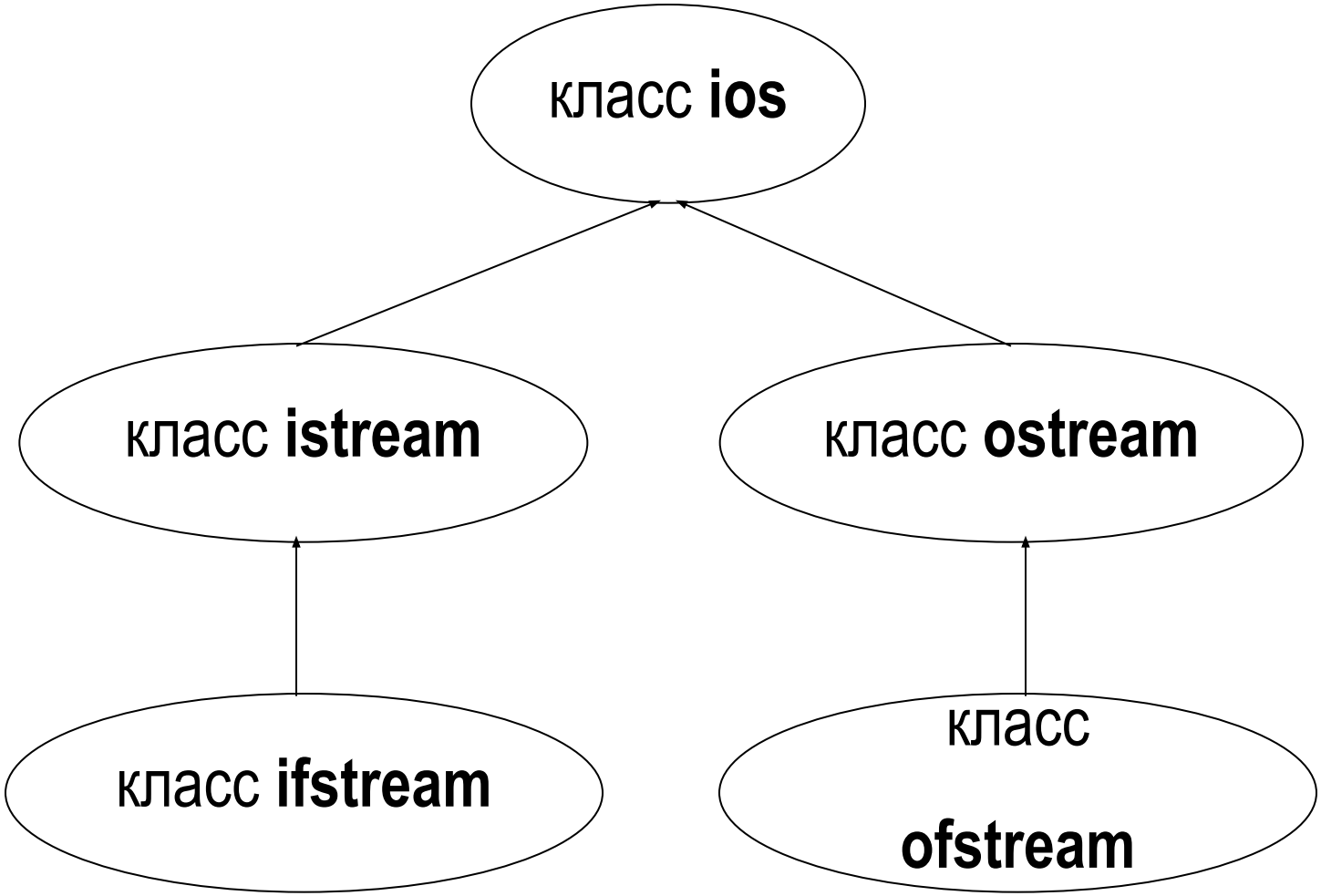
- Вывод данных обеспечивается классом ***ostream,*** который имеет перегруженную операцию левого сдвига “<<”

```
#include <iostream.h>
void main(void) {
        cout << “Hello!”;
}
```

- Ввод данных обеспечивается классом ***istream,*** который имеет перегруженную операцию правого сдвига “>>”

```
#include <iostream.h>
void main(void) {
        char ch;
        cin >> ch;
}
```

Классы потоковой библиотеки



Упрощенный класс *ios*

```
class ios {
  public:
    int eof( );
    int fail( );
    int bad( );
    int good( );
    enum io_state {goodbit=0,  eofbit=1, failbit=2, badbit=4,
          hardfail=010 };
    int rdstate( );
    void clear( int i=0 );
    operator void*( );
    int operator!( );
    enum open_mode { in=1, out=2, ate=4, app=010, trunc=020,
      nocreate=040, noreplace=0100 , binary=0200 };
    enum seek_dir { beg=0, cur=1, end=2 };
    // другие функции, включающие доступ к буферу
};
```

Константы состояния потока

- Состояние потока характеризуется *флагом состояния* – значением типа ***ios::io_state***

  ios::goodbit – все в порядке

  ios::eofbit – достигнут конец файла

  ios::failbit – ошибка форматирования или преобразования (поток можно использовать, сбросив бит)

  ios::badbit – серьезная ошибка обмена (поток, скорее всего, испорчен)

  ios::hardfail – неисправимая ошибка, связанная, например, с неисправностью оборудования

Константы доступа к потоку

- Доступ к потоку характеризуется *флагом доступа* – значением типа ***ios::open_mode***

  ios::in – открыть файл на чтение

  ios::out – открыть файл на запись

  ios::ate – после открытия файла поместить указатель в его конец

  ios::app – открыть файл на дозапись

  ios::trunc – открыть файл с усечением длины

  ios::nocreate – открыть существующий файл

  ios::noreplace – открыть несуществующий файл

  ios::binary – открыть нетекстовый файл

Константы позиционирования файлового указателя

- Перемещение файлового указателя по потоку определяется *флагом направления* – значением типа ***ios::seek_dir***

  ios::beg – смещение указателя от начала файла

  ios::cur – смещение указателя от его текущего положения в файле

  ios::end – смещение указателя от конца файла

Методы класса ***ios***

int *eof*( ) – Возвращает ненулевое значение, если установлен флаг ***ios::eofbit***

int *fail*( ) – Возвращает ненулевое значение, если установлен один из флагов ***ios::failbit, ios::badbit,*** или ***ios::hardfail***

int *bad*( ) – Возвращает ненулевое значение, если установлен один из флагов ***ios::badbit,*** или ***ios::hardfail***

int *good*( ) – Возвращает ненулевое значение, если сброшены все биты ошибок

int *rdstate*( ) – Возвращает текущее состояние

void *clear*(int=0) – Если параметр равен 0 (по умолчанию), все биты сбрасываются. Иначе он задает состояние ошибки

operator void*( ) – Возвращает нулевой указатель, если установлен один из битов ***ios::failbit, ios::badbit,*** или ***ios::hardfail*** (так же, как и ***fail*( )**)

int operator!( ) – Возвращает ненулевое значение, если установлен один из битов ***ios::failbit, ios::badbit,*** или ***ios::hardfail*** (так же, как и ***fail*( )**)

Упрощенный класс ***ostream***

```
class ostream : public ios {
  public:
    ostream( streambuf* );
    virtual ~ostream( );
    ostream& flush( );
    ostream& seekp( streampos p );
    ostream& seekp( streamoff o, seek_dir d );
    streampos tellp( );
    ostream& put( char );
    ostream& operator<<( char );
    ostream& operator<<( unsigned char );
    ostream& operator<<( const char* );
    ostream& operator<<( int );
    ostream& operator<<( long );
    ostream& operator<<( double );
    // и так далее …
};
```

Использование класса ***ostream***

```
#include <iostream.h>
void main(void) {
    cout << 42 ;
    cout << “hello world \n” ;
    cout << 3.1415 ;
    float a=4.56 ;
    int b=7 ;
    cout << “сумма ” << a << “и” << b
     << “ равна ” << a+b << “.\n”;
    if( cout.good() ) cout << “C I/O все в порядке.\n” ;
    else cerr << “C I/O что-то не в порядке.\n”
};
```

Упрощенный класс ***istream***

```
class istream : public ios {
  public:
    istream( streambuf* );
    virtual ~istream( );
    istream& seekg( streampos p );
    istream& seekg( streamoff o, seek_dir d );
    streampos tellg( );
    istream& get( char& );
    istream& getline( char*, int lim, char delim='\n');
    int get( );
    int peek( );
    istream& putback( char );
    istream& operator>>( char& );
    istream& operator>>( char* );
    istream& operator>>( int& );
    // и так далее …
};
```

Использование класса ***istream***

```
#include <iostream.h>
void main(void) {
    int i ;
    char buffer[256];
    double d ;
    cin >> i ;          // operation>>(int&)
    cin >> buffer ; // operation>>(char*)
    cin >> d ;          // operation>>(double&)
    cin >> i >> buffer >> d ;   // тот же результат
    if( cin ) cout << "i=" <<  i << "buffer:" << buffer
        << "d=" << d ;
    else cerr << "Ввод ошибочен.\n"
};
```

Упрощенный класс ***ofstream***

```
class ofstream : public ostream {
  public:
    ofstream( );
    ofstream( const char* name, int mode=ios::out,
          int prot=filebuf::openprot );
    ofstream( int fd );
    ~ofstream( );

    void open( const char* name, int mode=ios::out,
          int prot=filebuf::openprot );
    void close( );
};
```

Упрощенный класс ***ifstream***

```
class ifstream : public istream {
  public:
    ifstream( );
    ifstream( const char* name, int mode=ios::in,
          int prot=filebuf::openprot );
    ifstream( int fd );
    ~ifstream( );

    void open( const char* name, int mode=ios::in,
          int prot=filebuf::openprot );
    void close( );
};
```

Использование файловых потоков

```
#include <fstream.h>
void main(void) {
    ifstream in("\tmp\source") ;
    ofstream out("\tmp\dest") ;
    int i ;
    double d ;
    if( in.good() && out.good() ) {
     in >> i >> d ;
     out << "read integer: " <<  i
          << " and double: " << d << ".\n";
    }
};
```

Пересмотренный класс *String*

```
#include <iostream.h>
class String {
    public:
        String() ;
        String( const char* ) ;
        String( const String& ) ;
       ~String() ;
        operator const char*( ) const;
        String& operator= ( const String& );
         int  length( ) const ;
         friend istream& operator>>(istream&, String&);
         friend ostream& operator<<(ostream&, const String&);
         char& operator[ ] ( int );
         const char& operator[ ] ( int ) const;
         String substring (int start, int len) const;
         friend String operator+ (const String&, const String& );
    private:
      char* text ;
      static void error_msg( const String&  ) ;
};
```

Вывод объектов типа *String*

```
#include "mystring.h"
ostream& operator<<(ostream& out, const String& s)
{
    return out << s.text ;
}
```

Ввод объектов типа *String*

```
#include “mystring.h”
istream& operator>>( istream& in, String& s ) {
    char nextch;
    int size=0;
    // освобождаем занятую память, выделяем новую память
    while(1) {
     in.get(nextch);
     if( !in || nextch == ‘\n’ ) {
          s.text[size] = ‘\0’ ;
          return in ;
     }
     s.text[size++] = nextch ;
     // при необходимости вновь выделить больше памяти
    }
}
```

Использование операций ввода/вывода

```
#include “mystring.h”
#include <fstream.h>
void main(void) {
    String  firstname , lastname ;
    cerr << “Enter your first name: ”;
    cin >> firstname;
    cerr << “ and now your last name: ”;
    cin >> lastname;
    cout<<“Your name is:”<<firstname+lastname<<“.\n”;
    cin >> lastname;
    ofstream namefile(“\tmp\name”, ios::out | ios::app) ;
    namefile<<“Name: ”<<firstname+“ ”+lastname<<“.\n”;
}
```

РЕЗЮМЕ

- Потоки ввода/вывода обладают надежностью за счет контроля типов
- Просты в использовании
- Большая гибкость и возможности представления данных
- Свойства и методы можно осваивать по мере необходимости
- Легко расширяются на типы (классы) пользователя