Программирование в C++

Преподаватель
Колотова Людмила Павловна

Содержание 10-го занятия по практике

Потоки и файлы.

- Потоковые классы. Класс istream. Класс ostream.
- Потоковый ввод/вывод дисковых файлов
- Файловый ввод/вывод с помощью методов

Потоковые классы

Поток — это общее название потока данных. В C++ поток представляет собой объект некоторого класса. Именно поэтому вы могли встретить в листингах потоковые объекты **cin** и **cout**. Разные потоки предназначены для представления разных видов данных.

Преимущества потоков

Одним из аргументов в пользу потоков является простота использования. Каждый объект сам знает, как он должен выглядеть на экране. Это избавляет программиста от одного из основных источников ошибок. Другой причиной является то, что можно перегружать стандартные операторы и функции вставки (<<) и извлечения (>>) для работы с создаваемыми классами. Это позволяет работать с собственными классами как со стандартными типами, что, опять же, делает программирование проще и избавляет от множества ошибок. Оказывается, потоковый ввод/вывод, нужен. Потому что это лучший способ записывать данные в файл, лучший способ организации данных в памяти для последующего использования при вводе/выводе текста в окошках и других элементах графического интерфейса пользователя (GUI).

Иерархия потоковых классов

Потоковые классы имеют довольно сложную иерархическую структуру. Операция извлечения >> является методом класса istream, операция вставки << — методом класса ostream. Оба этих класса являются наследниками ios. Некоторые манипуляторы описаны в IOMANIP, а некоторые классы для работы с объектами «в памяти» определены в STRSTREAM. Класс ios является базовым для всей иерархии. Он содержит множество констант и методов, общих для операций ввода/вывода любых видов. Класс istream содержит функции: get(), getline(), read() и перегружаемую операцию извлечения (>>). Класс ostream содержит функции: put(), write() и перегружаемую операцию вставки (<<). Класс iostream — наследник одновременно классов istream и ostream (пример множественного наследования). Его производные классы могут использоваться при работе с объектами – дисковые файлы, которые могут быть открыты одновременно для записи и чтения. Классы: istream_withassign, ostream_withassign,iostream_withas-sign — являются наследниками istream, ostream и iostream соответ-ственно. Они добавляют к этим классам операторы присваивания

Иерархия потоковых классов

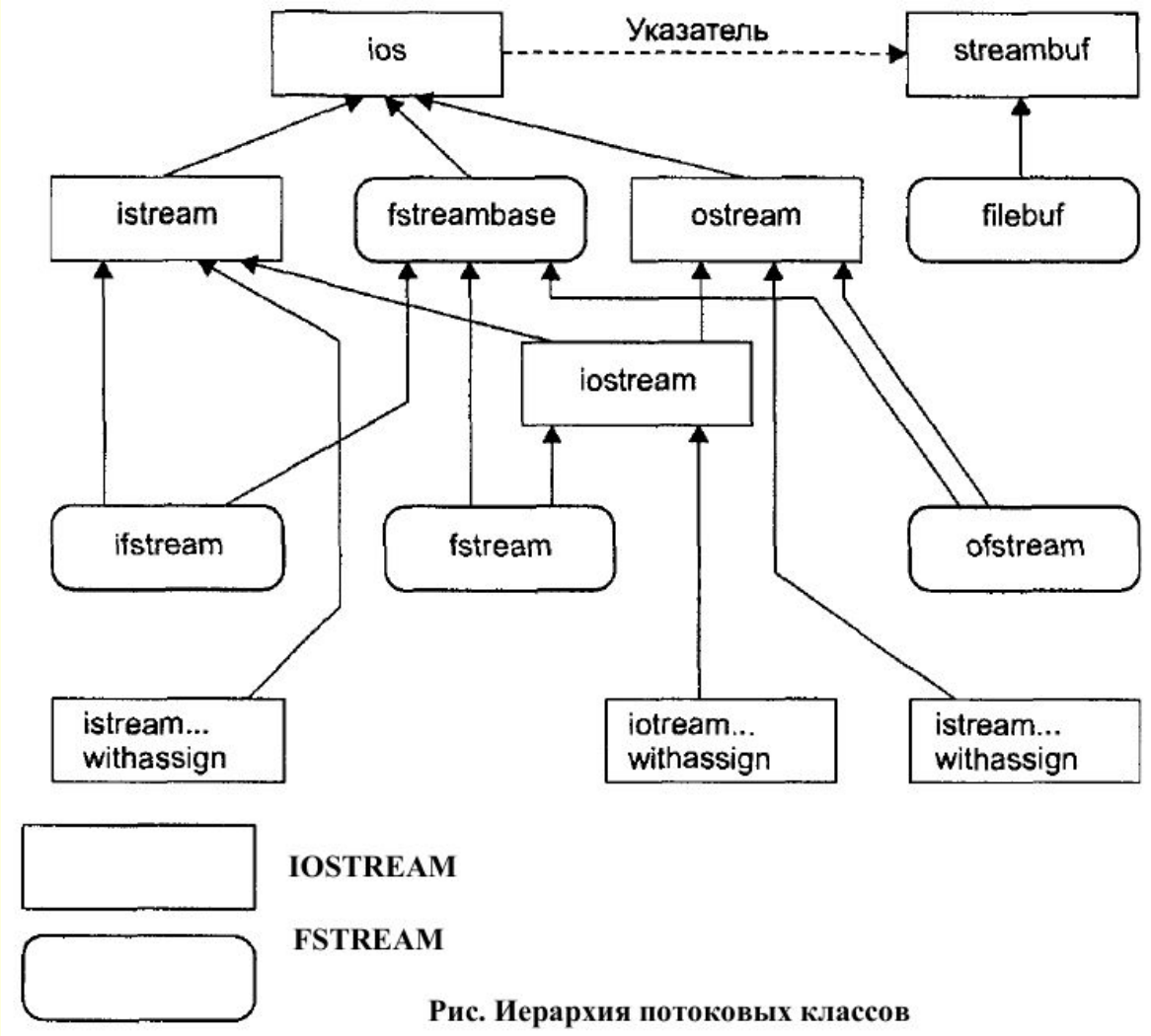


Рис. Иерархия потоковых классов

Класс ios

ios является дедушкой всех потоковых классов и обладает большинством особенностей, без которых работа с потоками была бы невозможна. Три его главных направления — это флаги форматирования, флаги ошибок и режим работы с файлами.

Класс ios. Флаги форматирования

Таблица. Флаги форматирования класса ios	
Флаг	**Значение**
skipws	Пропуск пробелов при вводе
left	Выравнивание по левому краю
right	Выравнивание по правому краю
internal	Заполнение между знаком или основанием числа и самим числом
dec	Перевод в десятичную форму
oct	Перевод в восьмеричную форму
hex	Перевод в шестнадцатеричную форму
boolalpha	Перевод логического 0 и 1 соответственно в «false» и «true»
showbase	Выводить индикатор основания системы счисления (0 для восьмеричной, 0x для шестнадцатеричной)
showpoint	Показывать десятичную точку при выводе
uppercase	Переводить в верхний регистр буквы X, E и буквы шестнадцатеричной системы счисления (ABCDEF) (по умолчанию — в нижнем регистре)
showpos	Показывать «+» перед положительными целыми числами
scientific	Экспоненциальный вывод чисел с плавающей запятой
fixed	Фиксированный вывод чисел с плавающей запятой
unitbuf	Сброс потоков после вставки
stdio	сброс stdout, sterror после вставки

Класс ios. Флаги форматирования

Есть несколько способов установки флагов форматирования, для каждого свои. Так как они являются компонентами класса ios, обычно к ним обращаются посредством написания имени класса и оператора явного задания (например, **ios::skipws**). Все без исключения флаги могут быть выставлены с помощью методов **setf()** и **unsetf()**. Примеры:

```
cout.setf(ios::left);       // выравнивание текста по левому краю
cout << "Этот текст выровнен по левому краю"
cout.unsetf(ios::left);   // вернуться к прежнему форматированию
```

Класс ios. Манипуляторы

Таблица . Манипуляторы ios без аргументов

Манипулятор	Назначение
ws	Включает пропуск пробелов при вводе
dec	Перевод в десятичную форму
oct	Перевод в восьмеричную форму
hex	Перевод в шестнадцатеричную форму
endl	Вставка разделителя строк и очистка выходного потока
ends	Вставка символа отсутствия информации для окончания выходной строки
flush	Очистка выходного потока
lock	Закрыть дескриптор файла
unlock	Открыть дескриптор файла

Манипуляторы — это инструкции форматирования, которые вставляются прямо в поток. Например **endl**, который посылает символ разделителя строк в поток и сбрасывает буфер.

Класс ios. Манипуляторы

Таблица. Манипуляторы ios с аргументами

Манипулятор	Аргумент	Назначение
setw()	ширина поля (int)	Устанавливает ширину поля для вывода данных
setfill()	символ заполнения (int)	Устанавливает символ заполнения (по умолчанию, пробел)
setprecision()	точность (int)	Устанавливает точность (число выводимых знаков)
setiosflags()	Флаги форматирования (long)	Устанавливает указанные флаги форматирования
resetiosflags()	Флаги форматирования (long)	Сбрасывает указанные флаги форматирования

Следует иметь в виду, что манипуляторы действуют только на те данные, которые следуют за ними в потоке, а не на те, которые находятся перед ними. На то он и поток.

Класс ios. Функции

Таблица. Функции ios

Функция	Назначение
ch = fill();	Возвращает символ заполнения (символ, которым заполняется неиспользуемая часть текстового поля; по умолчанию — пробел)
fill(ch);	Устанавливает символ заполнения
p = precision();	Возвращает значение точности (число выводимых знаков для формата с плавающей запятой)
precision(p);	Устанавливает точность p
w = width();	Возвращает текущее значение ширины поля (в символах)
width(w);	Устанавливает ширину текущего поля
setf(flags);	Устанавливает флаг форматирования (например, ios::left)
unsetf(flags);	Сбрасывает указанный флаг форматирования
setf(flags, field);	Очищает поле, затем устанавливает флаги форматирования

Класс **ios** содержит набор функций, с помощью которых можно выставлять флаги форматирования и выполнять некоторые другие действия. Эти функции вызываются для нужных потоковых объектов обычным способом — через точку. Например: **cout.width(14); cout.fill('*');**

Класс istream

Таблица. Функции istream

Функция	Назначение
>>	Форматированное извлечение данных всех основных (и перегружаемых) типов из потока
get(ch)	Извлекает один символ в ch
get(str)	Извлекает символы в массив str до ограничителя '\n'
get(str, MAX)	Извлекает до MAX числа символов в массив

Класс istream

Функция	Назначение
`get(str, DELIM)`	Извлекает символы в массив str до указанного ограничителя (обычно '\n'). Оставляет ограничитель в потоке
`get(str, MAX, DELIM)`	Извлекает в массив str до MAX символов или до символа DELIM. Оставляет ограничитель в потоке
`getline(str,MAX, DELIM)`	Извлекает в массив str до MAX символов или символа DELIM. Извлекает ограничитель из потока
`putback(ch)`	Вставляет последний прочитанный символ обратно во входной поток
`ignore(MAX,DELIM)`	Извлекает и удаляет до MAX числа символов до ограничителя включительно (обычно '\n'). С извлеченными данными ничего не делает
`peek(ch)`	Читает один символ, оставляя его в потоке
`count = gcount()`	Возвращает число символов, прочитанных только что встретившимися вызовами get(), getline() или read()
`read(str, MAX)`	(Для файлов.) Извлекает вплоть до MAX числа символов в массив str
`seekg()`	Устанавливает расстояние (в байтах) от начала файла до файлового указателя
`seekg(pos,seek_dir)`	Устанавливает расстояние (в байтах) от указанной позиции в файле до указателя файла. seek_dir может принимать значения ios::beg, ios::cur, ios::end
`pos = tellg(pos)`	Возвращает позицию (в байтах) указателя файла от начала файла

Класс istream

Нам уже встречались некоторые из этих функций, например **get()**. Большинство из них рассчитаны на работу с объектом **cin**, обычно представляющим собой поток данных, вводимых с клавиатуры. Последние четыре функции предназначены только для работы с дисковыми файлами.

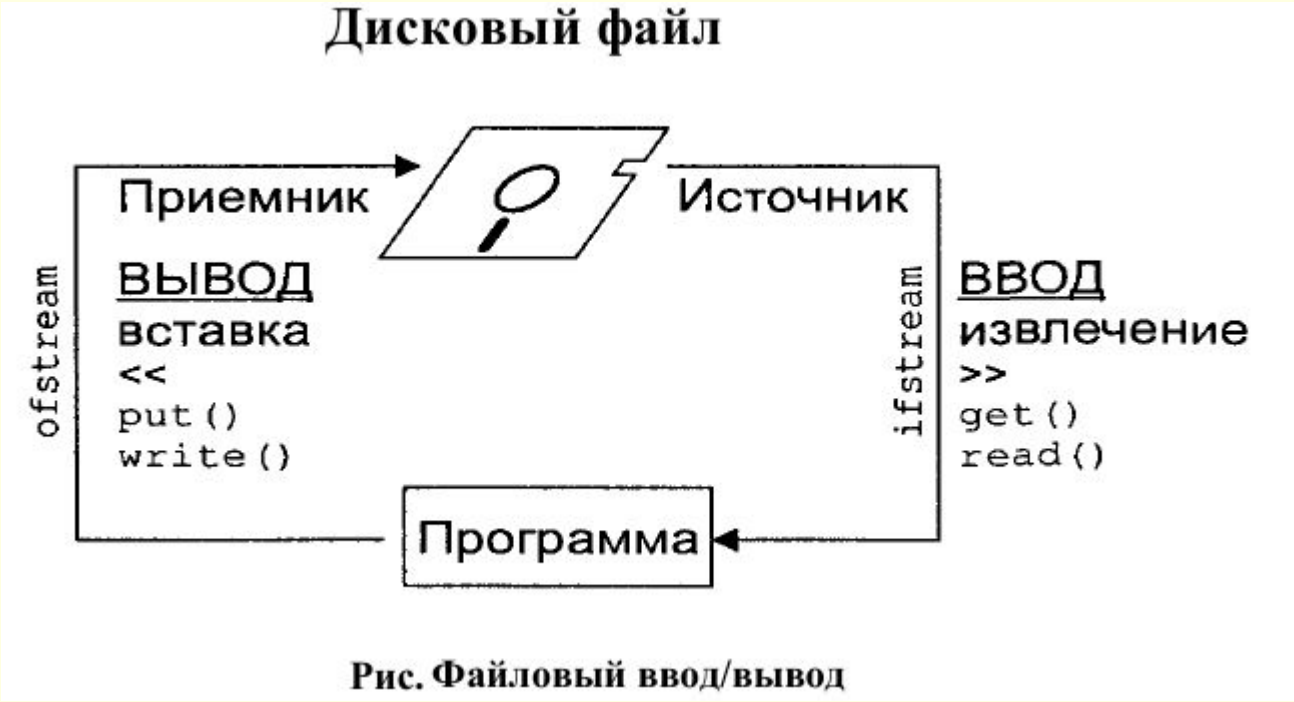


Рис. Файловый ввод/вывод

Класс ostream

Таблица. Функции ostream

Функция	Назначение
`<<`	Форматированная вставка данных любых стандартных (и перегруженных) типов
`put(ch)`	Вставка символа ch в поток
`flush()`	Очистка буфера и вставка разделителя строк
`write(str, SIZE)`	Вставка SIZE символов из массива str в файл
`seekp(position)`	Устанавливает позицию в байтах файлового указателя относительно начала файла
`seekp(position, seek_dir)`	Устанавливает позицию в байтах файлового указателя относительно указанного места в файле, seek_dir может принимать значения ios::beg, ios::cur, ios::end
`pos = tellp()`	Возвращает позицию указателя файла в байтах

Класс **ostream** предназначен для вывода (вставки в поток) данных. Как и в предыдущем случае, последние четыре функции работают только с файлами.

Классы iostream и _withassign

Класс **iostream** является порожденным по отношению к **istream** и **ostream**. Его единственное предназначение — выступать в качестве базового класса для других специфических классов вида **iostream_withassign**. У него нет собственных методов, кроме конструктора и деструктора. Его порожденные классы могут осуществлять как ввод, так и вывод данных. Существуют три класса вида **_withassign**:

- **istream_withassign** — наследник istream;
- **ostream_withassign** — наследник ostream;
- **iostream_withassign** — наследник iostream.

Эти классы очень похожи на своих предков. Разница лишь в том, что в них, в отличие от породивших их классов, имеются перегружаемые операции присваивания, благодаря чему их объекты могут быть скопированы.

Предопределенные потоковые объекты

Использования двух предопределенных потоковых объектов, порожденных классами вида **_withassign**: **cin** и **cout** мы уже знакомы. Обычно они связаны с клавиатурой и монитором соответственно. Еще двумя предопределенными объектами являются **cerr** и **clog**:

- **cin**, объект **istream_withassign**, используется для операций ввода с клавиатуры;
- **cout**, объект **ostream_withassign**, используется для операций вывода на экран монитора;
- **cerr**, объект **ostream_withassign**, используется для сообщений об ошибках;
- **clog**, объект **ostream_withassign**, используется для ведения журнала.

Объект **cerr** часто используется для сообщений об ошибках и программной диагностики. Поток, посланный в него, немедленно выводится на экран, минуя буферизацию. Этим **cerr** отличается от **cout**. К тому же этот поток не может быть перенаправлен.

Ошибки потоков. Биты статуса ошибки

Таблица. Флаги статуса ошибок

Название	Значение
goodbit	Ошибок нет (флаги не установлены, значение = 0)
eofbit	Достигнут конец файла
failbit	Операция не выполнена (пользовательская ошибка, преждевременный EOF)
badbit	Недопустимая операция (нет ассоциированного streambuf)
hardfail	Неисправимая ошибка

Таблица. Функции для флагов ошибок

Функция	Назначение
int = eof();	Возвращает true, если установлен флаг EOF
int = fail();	Возвращает true, если установлены флаги failbit, badbit или hardfail
int = bad();	Возвращает true, если установлены флаги badbit или hardfail
int = good();	Возвращает true, если ошибки не было
clear(int = 0);	При использовании без аргумента снимает все флаги ошибок, в противном случае устанавливает указанный флаг, например clear(ios::failbit)

Задача 10-1

Ввод чисел

Обработка ошибки при вводе чисел. При чтении числа с клавиатуры или с диска, проверяется значение **goodbit**, и сообщается об ошибке, если его значение не равно **true**, и предоставляется возможность пользователю ввести корректное число.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
using namespace std;
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    int i;
    while(true)                 // Цикл до тех пор, пока
    {                           // ввод не будет корректным
       cout <<"Введите целое число: ";
       cin >> i;
       if( cin.good() )              // если нет ошибок
          { cin.ignore(10, '\n'); // удалить разделитель строк
             break;
          }                          // выйти из цикла
       cin.clear();               // Очистить биты ошибок
       cout << "Неправильный ввод данных\n";
       cin.ignore(10, '\n');    // Удалить разделитель строк
    }
    cout << "целое число: " << i; //целое без  ошибок
    cout << endl;    system("PAUSE"); return EXIT_SUCCESS;
}
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_1.exe
Введите целое число: два
Неправильный ввод данных
Введите целое число: 2
целое число: 2
Для продолжения нажмите любую клавишу
```

Задача 10-2

Ввод целого числа и дробного

Обработка ошибки при вводе чисел. При чтении числа с клавиатуры или с диска, проверяется значение **goodbit**, и сообщается об ошибке, если его значение не равно **true**, и предоставляется возможность пользователю ввести корректное число

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <cstdlib>       // для atoi(), atof()
using namespace std;
int isFeet(string str)       // true если введена строка
{                           // с правильным значением футов
  int slen = str.size();     // получить длину
  if(slen==0 || slen > 5)    // не было или слишком много
                             // данных
    return 0;                // не целое
  for(int j=0; j<slen; j++) // проверить каждый символ
                             // если не цифра или минус
  if((str[j] < '0' || str[j] > '9')&&str[j] != '-')
      return 0;              // строка неправильных футов
  double n = atof( str.c_str() ); // перевод в double
  if( n<-999.0 || n>999.0 )  // вне допустимых значений?
    return 0;                 // если да, неправильные футы
  return 1;                   // правильные футы
}
class Distance           // Класс английских расстояний
{ private:    int feet;    float inches;
  public: Distance(){feet=0;inches=0.0;} // конструктор (без аргументов)
    Distance(int ft,float in){feet=ft;inches=in;} // конструктор (2 арг.)
    void showdist(){ cout << feet << "\'-" << inches << '\"';}
    void getdist(); // запросить длину у пользователя
};
```

Задача 10-2

Ввод целого числа и дробного (продолжение)

```
void Distance::getdist()     // получение длины от пользователя
{ string instr;              // для входной строки
  while(true)       // цикл, пока футы не будут правильными
  { cout << "Введите футы: ";
    cin >> instr;            // получить футы как строку
    if( isFeet(instr) )      // правильное значение?
    { cin.ignore(10, '\n'); // съесть символы,включая разделитель строк
      feet = atoi( instr.c_str() ); // перевести значение в целочисленное
      break;  }              // выход из цикла 'while'
                            // нет, не целое
    cin.ignore(10, '\n');    // съесть символы, включая разделитель строк
    cout << "Футы должны быть целыми < 1000\n";
  }  //конец цикла while для футов
  while(true)                // цикл проверки дюймов
  { cout << "Введите дюймы: ";
    cin >> inches;           // получить дюймы (тип float)
    if(inches>=12.0 || inches<0.0)
    { cout << "Дюймы должны быть между 0.0 и 11.99\n";
      cin.clear(ios::failbit); //"искусственно" установить флаг ошибки
    }
    if( cin.good() )         // все ли хорошо с cin
    { cin.ignore(10, '\n'); // съесть разделитель
      break;       }         // Ввод корректный, выйти из 'while'
    cin.clear();            // ошибка; очистить статус ошибки
    cin.ignore(10, '\n'); // съесть символы с разделителем
    cout << "Неверно введены дюймы\n";  //заново
  }  //конец while для дюймов
}
```

Задача 10-2

Ввод целого числа и дробного (окончание)

```
int main(int argc, char *argv[])
{    SetConsoleCP(1251); SetConsoleOutputCP(1251);
     Distance d;              //создать объект Distance
     char ans;
     do { d.getdist();          // получить его значение
          cout << "\nРасстояние = ";
          d.showdist();         // вывести его
          cout << "\nЕще раз (y/n)? ";
          cin >> ans;
          cin.ignore(10,'\n' ); // съесть символы и разделитель
     } while(ans != 'n');  // цикл до 'n'
     system("PAUSE");
     return EXIT_SUCCESS;
}
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_2.exe
Введите футы: пять
Футы должны быть целыми < 1000
Введите футы: 5
Введите дюймы: 22
Дюймы должны быть между 0.0 и 11.99
Неверно введены дюймы
Введите дюймы: 11.99

Расстояние = 5'-11.99"
Еще раз (y/n)? y
Введите футы: -10
Введите дюймы: 5.5

Расстояние = -10'-5.5"
Еще раз (y/n)? n
Для продолжения нажмите любую клавишу
```

Потоковый ввод/вывод дисковых файлов

Большинству программ требуется сохранять данные на диске и считывать их. Работа с дисковыми файлами подразумевает наличие еще одного набора специальных классов: **ifstream** для ввода и **ofstream** для вывода. Класс **fstream** осуществляет и ввод, и вывод.

Форматированный файловый ввод/вывод. При форматированном вводе/выводе числа хранятся на диске в виде серии символов. Таким образом, число 6.02 вместо того, чтобы храниться в виде четырехбайтного значения типа float или восьмибайтного double, хранится в виде последовательности символов '6', '.', '0', '2'.

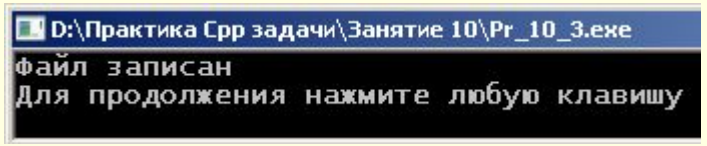
Символы и строки хранятся в файлах в привычной форме.

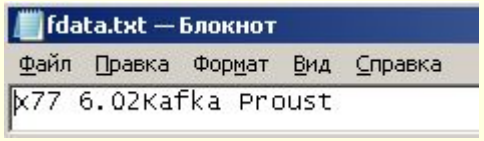
Задача 10-3

Запись данных

Следующая программа демонстрирует запись **символа**, **целого** числа, числа типа **double** и двух объектов типа **string** в дисковый файл. Вывод на экран не производится.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    char ch = 'x';  int j = 77;  double d = 6.02;
    string str1 = "Kafka";  // строки без
    string str2 = "Proust"; // пробелов
    ofstream outfile("fdata.txt"); //создать объект ofstream
    outfile << ch           // вставить (записать) данные
        << j
        << ' '              // пробелы между числами
        << d
        << str1
        << ' '              // пробелы между строками
        << str2;
    cout << "Файл записан\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```



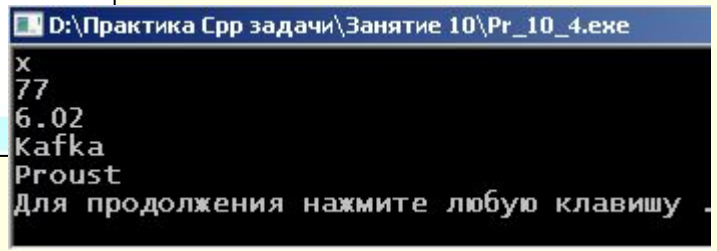


Задача 10-4

Чтение данных

Прочитать файл, если он уже создан, можно с использованием объекта типа **ifstream**, инициализированного именем файла. Файл автоматически открывается при создании объекта. Затем данные из него можно считать с помощью оператора извлечения (>>).

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>                  // для файлового ввода/вывода
using namespace std;
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    char ch;  int j;  double d;  string str1;  string str2;
    ifstream infile("fdata.txt"); // создать объект ifstream
                                  // извлечь (прочесть) из него данные
    infile >> ch >> j >> d >> str1 >> str2;
    cout << ch << endl             // вывести данные
      << j << endl
      << d << endl
      << str1 << endl
      << str2 << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

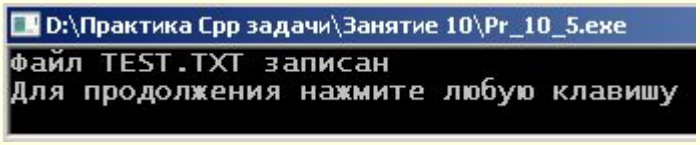


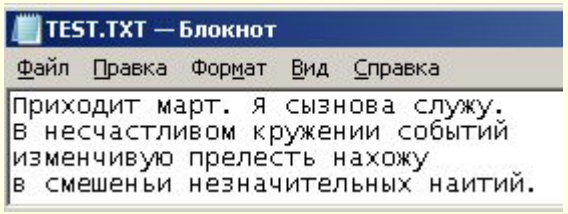
Задача 10-5

Строки с пробелами (запись)

Следующая программа выводит строки с пробелами – после каждой строки нужно писать специальный символ-ограничитель и использовать функцию **getline()** вместо оператора извлечения.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    ofstream outfile("TEST.TXT"); // создать выходной файл
                                  // отправить текст в файл
    outfile << "Приходит март. Я сызнова служу.\n";
    outfile << "В несчастливом кружении событий \n";
    outfile << "изменчивую прелесть нахожу \n";
    outfile << "в смешеньи незначительных наитий.\n";
    cout << "Файл TEST.TXT записан \n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```





Строки стихотворения И. Бродского записаны в файл TEST.TXT

Задача 10-6

Строки с пробелами (чтение)

Следующая программа считывает строки с пробелами используя функцию **getline()**, результат чтения затем выводится на экран после считывания каждой строки.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    const int MAX = 80;             // размер буфера
    char buffer[MAX];               // буфер символов
    ifstream infile("TEST.TXT"); // создать входной файл
    while( !infile.eof() )          // цикл до EOF
    {  infile.getline(buffer, MAX); // читает строку текста
       cout << buffer << endl;      // и выводит ее
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_6.exe
Приходит март. Я сызнова служу.
В несчастливом кружении событий
изменчивую прелесть нахожу
в смешеньи незначительных наитий.

Для продолжения нажмите любую клавишу
```

Определение признака конца файла (EOF)

Итак, объекты порожденных из ios классов содержат флаги статуса ошибок, с помощью которых можно проверить результат выполнения операций. При чтении файла условие окончания файла – сигнал EOF.

while(!infile.eof()) // Пока не EOF

Флаги ошибок **failbit** и **badbit** тоже могут возникнуть при работе программы, мы изменим условие цикла:

while(infile.good()) // Пока нет ошибок

Можно также проверять поток напрямую.
Запишем еще один вариант цикла while:

while(infile) // Пока нет ошибок

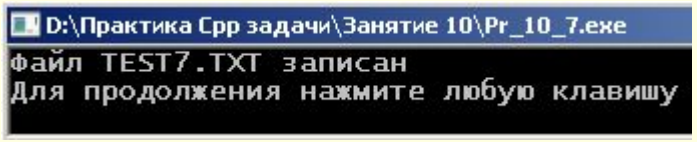
Задача 10-7

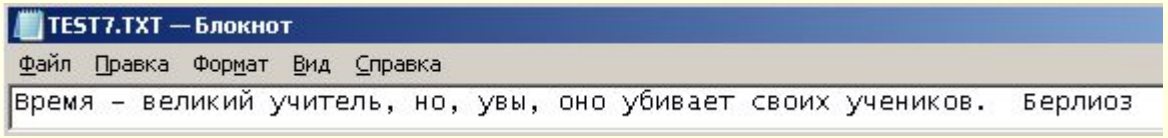
Ввод символов

Функция **put()** является методом **ostream**, может быть использована для ввода единичных символов.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;

int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    string str = "Время - великий учитель, но, увы, "
          "оно убивает своих учеников.  Берлиоз\n";
    ofstream outfile("TEST7.TXT");    // Создать выходной файл
    for(int j=0; j<str.size(); j++) // каждый символ
        outfile.put( str[j] );         // записывать в файл
    cout << "Файл TEST7.TXT записан\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```





Задача 10-8а

Вывод символов

Функция **get()** является методом **istream**, может быть использована для вывода единичных символов.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    char ch;                        // символ для считывания
    ifstream infile("TEST7.TXT"); // входной файл
    while( infile )                 // читать до EOF или ошибки
    {   infile.get(ch);              // считать символ
        cout << ch;                  // и вывести его
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

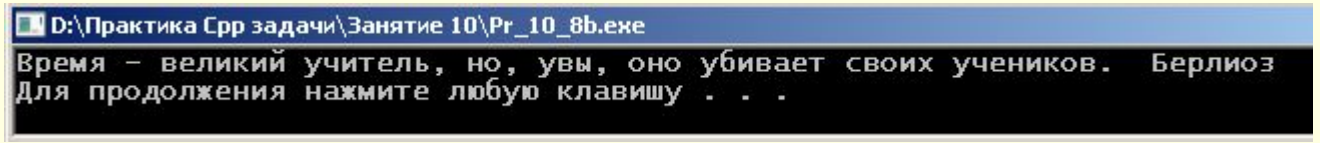
```
D:\Практика Cpp задачи\Занятие 10\Pr_10_8a.exe
Время - великий учитель, но, увы, оно убивает своих учеников.  Берлиоз
Для продолжения нажмите любую клавишу . . .
```

Задача 10-8в

Вывод символов

Есть и другой способ читать символы из файла — использовать функцию **rdbuf()** (она является методом класса ios).

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
int main(int argc, char *argv[])
{    SetConsoleCP(1251); SetConsoleOutputCP(1251);
     ifstream infile("TEST7.TXT"); // создать входной файл
     cout << infile.rdbuf();       // передать его буфер в cout
     system("PAUSE");
     return EXIT_SUCCESS;
}
```

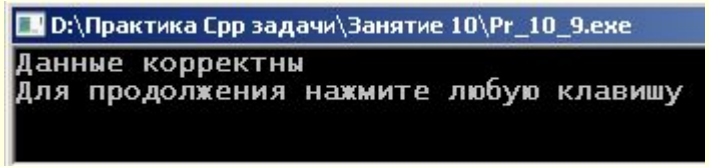


Задача 10-9

Двоичный ввод/вывод

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
const int MAX = 100;        // размер буфера
int buff[MAX];              // буфер для целых чисел
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    for(int j=0; j<MAX; j++) // заполнить буфер данными
    buff[j] = j;            // (0, 1, 2, ...)
                            // создать выходной поток
    ofstream os("edata.dat", ios::binary);
                            // записать в него
    os.write(reinterpret_cast<char*>(buff), MAX*sizeof(int) );
    os.close();               // должен закрыть его
    for(int j=0; j<MAX; j++)      // стереть буфер
    buff[j] = 0;
                            // создать входной поток
    ifstream is("edata.dat", ios::binary);
                            // читать из него
    is.read( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
    for(int j=0; j<MAX; j++)      // проверка данных
      if( buff[j] != j )
      { cerr << "Некорректные данные!\n"; system("PAUSE");return 1; }
    cout << "Данные корректны\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Массив целых чисел записывается в файл и читается из него. Используются две функции — **write**() (метод класса **ofstream**), **read**() (метод **ifstream**). Адрес вычислен с использованием **reinterpret_cast** относительно типа **char***. Второй параметр – длина в байтах.



Оператор reinterpret_cast

В задаче 10-9 использует оператор **reinterpret_ cast** для того, чтобы буфер данных типа **int** выглядел для функций **read**() и **write**() как буфер типа **char**.

```
is.read(reinterpret_cast<char*>(buff), MAX*sizeof(int));
```

reinterpret_cast изменяет тип данных в определенной области памяти, совершенно не задумываясь о том, имеет это смысл или нет. Поэтому вопрос целесообразности использования этого оператора остается целиком на совести программиста.

Закрытие файлов

До сих пор в наших примерах не нужно было вручную закрывать файлы — это делалось автоматически при окончании работы с ними. Соответствующие объекты запускали деструкторы и закрывали ассоциированные файлы. Но в задаче 10-9 оба потока, входной **is** и выходной **os**, связаны с одним и тем же файлом EDATA.DAT, поэтому первый поток должен быть закрыт до того, как откроется второй. Для этого используется функция **close**(). Можно запускать эту функцию каждый раз при закрытии файла, не полагаясь на деструктор потока.

Объектный ввод/вывод

Так как C++ — это все-таки объектно-ориентированный язык, рассмотрим, как происходит запись объектов в дисковые файлы и чтение из них.

Задача 10-10

Запись объекта на диск

При записи объекта обычно используем бинарный режим. В данной программе у пользователя запрашивается информация об объекте класса **person**, который потом записывается в файл PERSON.DAT.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
class person                // класс person
{ protected:  char name[80]; short age; // имя человека, возраст
  public:
    void getData()          // получить данные о человеке
      { cout << "Введите имя: "; cin >> name;
        cout << "Введите возраст: "; cin >> age;     }
};
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    person pers;              // создать объект
    pers.getData();           // получить данные
                              // создать объект ofstream
    ofstream outfile("PERSON.DAT", ios::binary);
                              // записать в него
    outfile.write(reinterpret_cast<char*>(&pers), sizeof(pers));
    cout << "Файл PERSON.DAT записан\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_10.exe
Введите имя: Артур
Введите возраст: 60
Файл PERSON.DAT записан
Для продолжения нажмите любую клавишу
```

Задача 10-11

Чтение объекта с диска

Для чтения используется метод **read**(). Из файла PERSON.DAT считывается информация объекта класса **person**, затем информация выводится на экран.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
class person              // класс person
{ protected: char name[80]; short age; // Имя человека, его возраст
  public:
    void showData()       // вывести данные
      { cout << "Имя: " << name << endl;
        cout << "Возраст: " << age << endl;       }
};
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    person pers;           // переменная типа person
    ifstream infile("PERSON.DAT", ios::binary);// создать поток
                           // чтение потока
    infile.read( reinterpret_cast<char*>(&pers), sizeof(pers));
    pers.showData();       // вывести данные
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_11.exe
Имя: Артур
Возраст: 60
Для продолжения нажмите любую клавишу
```

Задача 10-12

Ввод/вывод множества объектов

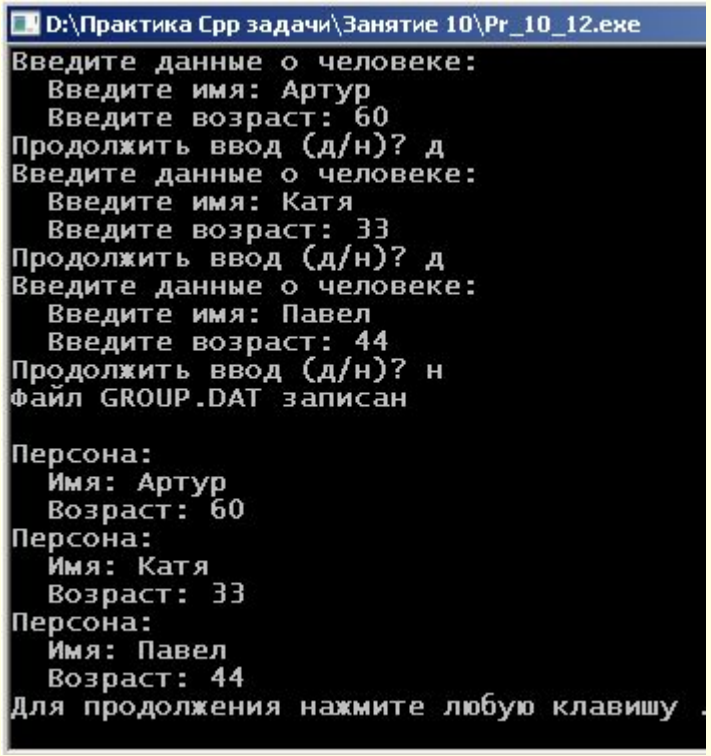
```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
class person               // класс person
{ protected: char name[80]; int age; // имя человека, его возраст
  public: void getData()       // получить данные о человеке
 {cout<<"  Введите имя: ";cin>>name;cout<<"  Введите возраст: ";cin>>age;}
          void showData()      // вывод данных о человеке
 { cout << "\n  Имя: " << name; cout << "\n  Возраст: " << age;     }
};
int main(int argc, char *argv[])
{   SetConsoleCP(1251); SetConsoleOutputCP(1251);
    char ch;  person pers;           // создать объект person
    fstream file; file.open("GROUP.DAT", ios::app |ios::out |ios::binary);
    do {  cout << "Введите данные о человеке:\n";
       pers.getData();  // получить данные и записать их в файл
       file.write( reinterpret_cast<char*>(&pers), sizeof(pers) );
       cout << "Продолжить ввод (д/н)? ";      cin >> ch;
    }  while(ch=='д');        // выход по 'н'
    file.close();    cout << "Файл GROUP.DAT записан" << endl;
    file.open("GROUP.DAT", ios::in |ios::binary);// создать поток
    file.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
    while( file.good() )  // Выход по EOF или ошибке
    {    cout << "\nПерсона:"; //вывести данные
         pers.showData();       //считать данные о следующем
         file.read(reinterpret_cast<char*>(&pers),sizeof(pers));
    }    cout << endl;
    system("PAUSE");     return EXIT_SUCCESS;
}
```

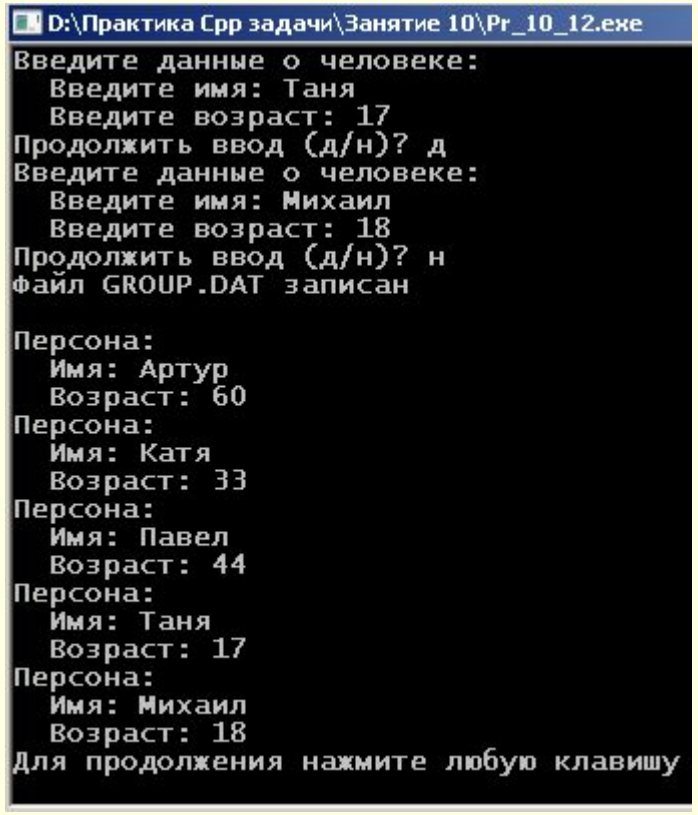
В следующем примере в файл записывается произвольное число объектов. Затем они считываются, и на экран выводится целиком содержимое файла.

Задача 10-12

Ввод/вывод множества объектов

В следующем примере в файл записывается произвольное число объектов. Затем они считываются, и на экран выводится целиком содержимое файла.





Биты режимов

Таблица. Биты режимов

Бит режима	Результат
in	Открытие для чтения (по умолчанию для ifstream)
out	Открытие для записи (по умолчанию для ofstream)
ate	Чтение, начиная с конца файла (AT End)
app	Запись, начиная с конца файла (AAPend)
trunc	Обрезать файл до нулевой длины, если он уже существует (TRUNCate)
nocreate	Не открывать несуществующий файл
noreplace	Не открывать для вывода существующий файл, если не установлены ate или app
binary	Открыть в бинарном (не текстовом) режиме

Указатели файлов

У каждого файлового объекта есть два ассоциированных с ним значения, называемые **указатель чтения** и **указатель записи**. Их также называют **текущая позиция чтения** и **текущая позиция записи**. Или просто **текущая позиция**. Функции **seekg**() и **tellg**() позволяют устанавливать и проверять текущий указатель **чтения**, а функции **seekp**() и **tellp**()— выполнять те же действия для указателя **записи**.

Задача 10-13

Вычисление сдвига

В следующем примере вычисляется количество записей person, по запросу выводится информация о персоне.

```
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>          // для файлового ввода/вывода
using namespace std;
class person               // класс person
{ protected: char name[80]; int age; // имя человека, его возраст
  public: void getData()       // получить данные о человеке
 {cout<<"  Введите имя: ";cin>>name;cout<<"  Введите возраст: ";cin>>age;}
          void showData()      // вывод данных о человеке
 { cout << "\n  Имя: " << name; cout << "\n  Возраст: " << age;     }
};
int main(int argc, char *argv[])
{    SetConsoleCP(1251); SetConsoleOutputCP(1251);
     person pers; ifstream infile;       // создать входной файл
     infile.open("GROUP.DAT",ios::in|ios::binary); // открыть файл
     infile.seekg(0, ios::end); //устан.указатель на 0 байт от конца файла
     int endposition = infile.tellg();       // найти позицию
     int n = endposition / sizeof(person);  // число человек
     cout << "\nВ файле " << n << " человек(а)";
     cout << "\nВведите номер персоны: ";     cin >> n;
     int position = (n-1) * sizeof(person);
     infile.seekg(position);  // число байт от начала
     infile.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
     pers.showData();                          //вывести одну персону
     cout << endl;     system("PAUSE");     return EXIT_SUCCESS;
}
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_13.exe

В файле 7 человек(а)
Введите номер персоны: 2

  Имя: Катя
  Возраст: 33
Для продолжения нажмите любую клавишу
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_13.exe

В файле 7 человек(а)
Введите номер персоны: 5

  Имя: Михаил
  Возраст: 18
Для продолжения нажмите любую клавишу
```

Задача 10-14

Реагирование на ошибки

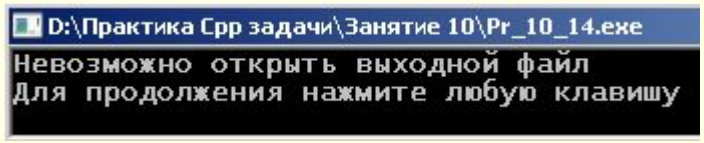
```
// Обработка ошибок ввода/вывода
#include <cstdlib>
#include <iostream>
#include <fstream>              // для файловых потоков
#include <windows.h>
//#include <process.h>           // для exit()
using namespace std;
const int MAX = 1000;
int buff[MAX];
int main()
{ SetConsoleCP(1251); SetConsoleOutputCP(1251);
  for(int j=0; j<MAX; j++) buff[j] = j;     // заполнить буфер данными
  ofstream os;                  // создать выходной поток
  os.open("a:edata14.dat", ios::trunc | ios::binary);  // открыть его
  if(!os) {cerr<<"Невозможно открыть выходной файл\n";system("PAUSE");exit(1);}
  cout << "Идет запись...\n"; // записать в него содержимое буфера
  os.write( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
  if(!os)  { cerr << "Запись в файл невозможна\n"; system("PAUSE"); exit(1); }
  os.close();                   // надо закрыть поток
  for(int j=0; j<MAX; j++) buff[j] = 0;         // очистить буфер
  ifstream is;                  // создать входной поток
  is.open("a:edata14.dat", ios::binary);
  if(!is) {cerr<<"Невозможно открыть входной файл\n";system("PAUSE");exit(1);}
  cout << "Идет чтение...\n"; // чтение файла
  is.read( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
  if(!is)  { cerr << "Невозможно чтение файла\n"; system("PAUSE"); exit(1); }
  for(int j=0; j<MAX; j++)          // проверять данные
    if(buff[j]!=j) {cerr<<"\nДанные некорректны\n"; system("PAUSE");exit(1); }
  cout << "Данные в порядке\n";
  system("PAUSE");     return EXIT_SUCCESS;
}
```

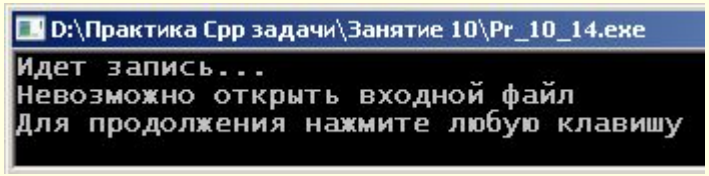
Обработка ошибок. Все дисковые операции проверяются после их выполнения Программа открывает выходной потоковый объект, записывает целый массив целых чисел единственным вызовом **write**() и закрывает объект. Затем открывает входной потоковый объект и считывает массив вызовом функции **read**().

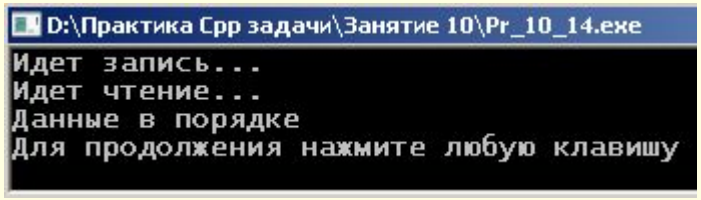
Задача 10-14

Реагирование на ошибки

Обработка ошибок. Все дисковые операции проверяются после их выполнения. Программа открывает выходной потоковый объект, записывает целый массив целых чисел единственным вызовом write() и закрывает объект. Затем открывает входной потоковый объект и считывает массив вызовом функции read().







Как классы записывают и читают сами себя

Статические функции

Одним из способов записать за один сеанс множество объектов является употребление статической функции, которая применяется ко всему классу в целом, а не к отдельным его объектам. Такая функция действительно может записать все объекты сразу. Она обратится к массиву указателей на объекты, который можно хранить в виде статической переменной. При создании каждого объекта его указатель заносится в этот массив. Статический элемент данных также может свято хранить память о том, сколько всего объектов было создано. Статическая функция **write**() *откроет* файл, в цикле пройдет по всем ссылкам массива, записывая по очереди все объекты, и после окончания записи *закроет* файл.

Как классы записывают и читают сами себя

Размеры порожденных объектов

Объекты, хранящиеся в памяти, имеют разные размеры. Класс **employee**, выступающий как базовый для классов **manager**, **scientist** и **laborer**. Объекты этих трех порожденных классов имеют разные размеры, так как содержат разные объемы данных.

Пусть имеется массив указателей arrap[], хранящий ссылки на объекты типа employee. Указатели могут ссылаться, таким образом, и на все три порожденных класса. При использовании *виртуальных* функций можно использовать выражения типа

arrap[]->putdata();

Тогда будет поставлена на исполнение во время работы программы версия **putdata**(), соответствующая объекту, на который ссылается указатель, а не та, что соответствует базовому классу.

Как классы записывают и читают сами себя

Использование функции **typeid**()
Можно использовать ее для определения класса объекта, подставляя имя этого класса в качестве аргумента **sizeof**(). Чтобы использовать **typeid**(), надо включить параметр компилятора RTTI. (это существенно только для компилятора Microsoft Visual C++)
Наш следующий пример показывает, как все вышеописанное работает. Узнав размер объекта, мы можем использовать его в функции **write**() для записи в файл. К программе 10_15 добавился несложный пользовательский интерфейс. Некоторые специфические методы сделаны виртуальными, чтобы можно было пользоваться массивом указателей на объекты. В программу также включены некоторые приемы обнаружения ошибок. Программа демонстрирует многие приемы, которые могут быть использованы в настоящем приложении для работы с базами данных. Она также показывает реальные возможности ООП.

Задача 10-15

```
// Файловый ввод/вывод объектов employee
#include <cstdlib>
#include <iostream>
#include <windows.h>
//#include <clocale>      // русские буквы
#include <fstream>       // для потоковых файловых функций
#include <typeinfo>      // для typeid()
#include <process.h>     // для exit()
using namespace std;
const int LEN = 32;      // Максимальная длина фамилий
const int MAXEM = 100;   // максимальное число работников
enum employee_type {tmanager, tscientist, tlaborer};
////////////////////////////////////////////////////////////
class employee                  // класс employee
  {
  private:
    char name[LEN];             // фамилия работника
    unsigned long number;       // номер работника
    static int n;               // текущее число работников
    static employee* arrap[]; //массив указателей на
                                // класс работников
  public:
    virtual void getdata()
      {
      cin.ignore(10, '\n');
      cout << "  Введите фамилию: "; cin >> name;
      cout << "  Введите номер: ";    cin >> number;
      }
    virtual void putdata()
      {
      cout << "\n  Фамилия: " << name;
      cout << "\n  Номер: " << number;
      }
    virtual employee_type get_type(); // получить тип
    static void add();                // добавить работника
    static void display();            // вывести данные обо всех
    static void read();               // чтение из файла
    static void write();              // запись в файл
  };
```

Классы записывают и читают сами себя

Код программы - начало

Задача 10-15

```
//статические переменные
int employee::n=0;           // текущее число работников
employee* employee::arrap[MAXEM]; // массив указателей
                                  // на класс работников
class manager : public employee    //класс manager (менеджеры)
  {
  private:
    char title[LEN];  // титул ("вице-президент" и т. п.)
    double dues;      // Налоги гольф-клуба
  public:
    void getdata()
      {
      employee::getdata();
      cout << "  Введите титул: ";       cin >> title;
      cout << "  Введите налоги: "; cin >> dues;
      }
    void putdata()
      {
      employee::putdata();
      cout << "\n  Титул: " << title;
      cout << "\n  Налоги гольф-клуба: " << dues;
      }
  };
class scientist : public employee      //класс scientist (ученые)
  {
  private:
    int pubs;               // число публикаций
  public:
    void getdata()
      {
      employee::getdata();
      cout << "  Введите число публикаций: "; cin >> pubs;
      }
    void putdata()
      {
      employee::putdata();
      cout << "\n  Число публикаций: " << pubs;
      }
  };
```

Классы записывают и читают сами себя

Код программы – продолжение 1

Задача 10-15

```
class laborer : public employee      //класс laborer (рабочие)
  {  };
void employee::add() //добавить работника в список (хранится в ОП)
  {
  char ch;
  cout << "'6' для добавления менеджера"
       "\n'7' для добавления ученого"
       "\n'8' для добавления рабочего"
       "\nВаш выбор: ";
  cin >> ch;
  switch(ch)
    {          //создать объект указанного типа
    case '6': arrap[n] = new manager;  break;
    case '7': arrap[n] = new scientist; break;
    case '8': arrap[n] = new laborer;  break;
    default: cout << "\nНеизвестный тип работника\n";
             system("PAUSE"); return;
    }
  arrap[n++]->getdata();  //Получить данные от пользователя
  }
void employee::display()   //Вывести данные обо всех работниках
  {
  for(int j=0; j<n; j++)
    {
    cout  << (j+1);                 // вывести номер
    switch( arrap[j]->get_type() ) //вывести тип
      {
      case tmanager:   cout << ". Тип: Менеджер";  break;
      case tscientist: cout << ". Тип: Ученый"; break;
      case tlaborer:   cout << ". Тип: Рабочий";  break;
      default: cout << ". Неизвестный тип";
      }
    arrap[j]->putdata();            // Вывод данных
    cout << endl;
    }
  }
```

Классы записывают и читают сами себя

Код программы – продолжение 2

Задача 10-15
Классы записывают и читают сами себя

```
employee_type employee::get_type()  //Возврат типа объекта
  {
  if( typeid(*this) == typeid(manager) )
    return tmanager;
  else if( typeid(*this)==typeid(scientist) )
    return tscientist;
  else if( typeid(*this)==typeid(laborer) )
    return tlaborer;
  else
    { cerr << "\nНеправильный тип работника";
    system("PAUSE");exit(1); }
  return tmanager;
  }
void employee::write() //Записать все объекты, из памяти, в файл
  {
  int size;
  cout << "Идет запись " << n << " работников.\n";
  ofstream ouf;            // открыть ofstream в двоичном виде
  employee_type etype;    // тип каждого объекта employee

  ouf.open("EMPLOY15.dat", ios::trunc|ios::binary);
  if(!ouf)
    { cout << "\nНевозможно открыть файл\n";return; }
  for(int j=0; j<n; j++)  // Для каждого объекта
    {                     // получить его тип
    etype = arrap[j]->get_type();
                          // записать данные в файл
    ouf.write( (char*)&etype, sizeof(etype) );
    switch(etype)         // find its size
      {
      case tmanager:  size=sizeof(manager); break;
      case tscientist: size=sizeof(scientist); break;
      case tlaborer:  size=sizeof(laborer); break;
      }      //запись объекта employee в файл
      ouf.write( (char*)(arrap[j]), size );
    if(!ouf)
      { cout << "\nЗапись в файл невозможна\n";
        system("PAUSE");return; }
    }
    ouf.close();
  }
```

Код программы – продолжение 3

Задача 10-15

Классы записывают и читают сами себя

```
void employee::read() //чтение всех данных из файла в память
  {
  int size;                 // размер объекта employee
  employee_type etype;      // тип работника
  ifstream inf;             // открыть ifstream в двоичном виде
  inf.open("EMPLOY15.dat", ios::binary);
  if(!inf)
    { cout << "\nНевозможно открыть файл\n";return; }
  n = 0;                    // В памяти работников нет
  while(true)
    {                       // чтение типа следующего работника
    inf.read( (char*)&etype, sizeof(etype) );
    if( inf.eof() )         // выход из цикла по EOF
      break;
    if(!inf)                // ошибка чтения типа
      { cout << "\nНевозможно чтение типа\n";return; }
    switch(etype)
      {                     // создать нового работника
      case tmanager:        // корректного типа
        arrap[n] = new manager;
        size=sizeof(manager);
        break;
      case tscientist:
        arrap[n] = new scientist;
        size=sizeof(scientist);
        break;
      case tlaborer:
        arrap[n] = new laborer;
        size=sizeof(laborer);
        break;
      default: cout << "\nНеизвестный тип в файле\n"; return;
      }                     // чтение данных из файла
    inf.read( (char*)arrap[n], size  );
    if(!inf)                // ошибка, но не EOF
      { cout << "\nЧтение данных из файла невозможно\n"; return; }
    n++;                    // счетчик работников увеличить
    }  //end while
  cout << "Идет чтение " << n << " работников\n";
  inf.close();
  }
```

Код программы – продолжение 4

Задача 10-15

Классы записывают и читают сами себя

```
int main()
{ SetConsoleCP(1251); SetConsoleOutputCP(1251);
  char ch;
  while(true)
    {
    cout << "'1' - добавление сведений о работнике"
          "\n'2' - вывести сведения обо всех работниках"
          "\n'3' - записать все данные в файл"
          "\n'4' - прочитать все данные из файла"
          "\n'5' - выход"
          "\nВаш выбор: ";
    cin >> ch;
    switch(ch)
      {
      case '1':           // добавить работника
        employee::add(); break;
      case '2':           // вывести все сведения
        employee::display(); break;
      case '3':           // запись в файл
        employee::write(); break;
      case '4':           // чтение всех данных из файла
        employee::read(); break;
      case '5': system("PAUSE"); exit(0);  // выход
      default: cout << "\nНеизвестная команда";
      }  //end switch
    }  //end while
  cout<< endl;system("PAUSE");return 0;
}  //end main()
```

Код программы – конец

Задача 10-15

Как классы записывают и читают сами себя

Код типа объекта

Мы умеем определять класс объекта, находящегося в памяти, но как узнать класс объекта, если мы собираемся читать его из файла? Поэтому при записи данных на диск необходимо записывать код (перечисляемую переменную типа **employee_type**) прямо перед данными объекта. До начала чтения из файла надо прочитать его значение и создать объект соответствующего типа.

aptr_to_obj = new aClass; // arrap[n] = new manager;

И только после этого можно копировать данные из файла в этот новый объект. При корректном способе создания объекта запускается его конструктор. Это необходимо даже тогда, когда вы не определяете собственный, а используете конструктор по умолчанию. Ведь объект — это нечто более глобальное, нежели просто область памяти, в которой хранятся данные. Не забывайте, что это еще и набор методов, некоторые из них вы даже не видите!

Задача 10-15

Как классы записывают и читают сами себя

Взаимодействие с программой

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_15.exe
'1' - добавление сведений о работнике
'2' - вывести сведения обо всех работниках
'3' - записать все данные в файл
'4' - прочитать все данные из файла
'5' - выход
Ваш выбор: 4
Идет чтение 3 работников
'1' - добавление сведений о работнике
'2' - вывести сведения обо всех работниках
'3' - записать все данные в файл
'4' - прочитать все данные из файла
'5' - выход
Ваш выбор: 2
1. Тип: Менеджер
  Фамилия: Иванов
  Номер: 111
  Титул: Президент
  Налоги гольф-клуба: 222
2. Тип: Ученый
  Фамилия: Петров
  Номер: 333
  Число публикаций: 99
3. Тип: Рабочий
  Фамилия: Сидоров
  Номер: 555
'1' - добавление сведений о работнике
'2' - вывести сведения обо всех работниках
'3' - записать все данные в файл
'4' - прочитать все данные из файла
'5' - выход
Ваш выбор: 5
```

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_15.exe
'1' - добавление сведений о работнике
'2' - вывести сведения обо всех работниках
'3' - записать все данные в файл
'4' - прочитать все данные из файла
'5' - выход
Ваш выбор: 4
Идет чтение 4 работников
'1' - добавление сведений о работнике
'2' - вывести сведения обо всех работниках
'3' - записать все данные в файл
'4' - прочитать все данные из файла
'5' - выход
Ваш выбор: 2
1. Тип: Менеджер
  Фамилия: Иванов
  Номер: 111
  Титул: Президент
  Налоги гольф-клуба: 222
2. Тип: Ученый
  Фамилия: Петров
  Номер: 333
  Число публикаций: 99
3. Тип: Рабочий
  Фамилия: Сидоров
  Номер: 555
4. Тип: Ученый
  Фамилия: Баринов
  Номер: 777
  Число публикаций: 77
'1' - добавление сведений о работнике
'2' - вывести сведения обо всех работниках
'3' - записать все данные в файл
'4' - прочитать все данные из файла
'5' - выход
Ваш выбор: 5
```

Перегрузка операторов извлечения и вставки

Перейдем к изучению следующей темы, связанной с потоками. Данный раздел будет посвящен перегрузке операторов извлечения и вставки. Это мощный прием C++, который позволяет поддерживать ввод/вывод пользовательских типов наравне со стандартными, такими, как float и int. Например, если имеется объект класса **crawdad**, называющийся **cd1**, его можно вывести на экран простым выражением

```
cout << "\ncd1 =" << cd1;
```

Как видите, оно ничем не отличается от подобных конструкций для стандартных типов. Операторы извлечения и вставки могут быть перегружены и для работы с консольным вводом/выводом (с клавиатуры и на экран — **cin** и **cout**). С оглядкой, но можно перегрузить их и для работы с дисковыми файлами. Рассмотрим примеры всех этих ситуаций.

Задача 10-16

Перегрузка cout и cin

```
// Перегружаемые операции << и >>
#include <cstdlib>
#include <iostream>
#include <windows.h>
using namespace std;
class Distance                          // класс английских расстояний
{ private:    int feet;    float inches;
  public:
    Distance() :feet(0),inches(0.0) {  } // конструктор (0 арг.)
                                        // конструктор (2 аргумента)
    Distance(int ft, float in) : feet(ft), inches(in) {  }
    friend istream & operator >> (istream& s, Distance& d);
    friend ostream & operator << (ostream& s, Distance& d);
};
istream & operator >> (istream & s, Distance & d) //получить
  {                                  // значение от пользователя
  cout << "\nВведите футы: ";  s >> d.feet;  // используется
  cout << "Введите дюймы: ";  s >> d.inches; // перегруженный
  return s; }                                 // оператор >>
ostream& operator << (ostream & s, Distance & d)       // вывести
  {                                          // расстояние
  s << d.feet << "\'-" << d.inches << '\"'; // используется
  return s;  }                                // перегруженный <<
int main()
{  SetConsoleCP(1251); SetConsoleOutputCP(1251);
  Distance dist1, dist2;     //Определение переменных
  Distance dist3(11, 6.25);  // определение, инициализация dist3
  cout << "\nВведите два значения расстояний:";
  cin >> dist1 >> dist2;     // Получить значения от пользователя
  cout << "\ndist1 = " << dist1 << "\ndist2 = " << dist2;
  cout << "\ndist3 = " << dist3 << endl;
  system("PAUSE");  return 0;
}
```

Приведем пример, в котором операторы извлечения и вставки для класса Distance перегружены для работы с **cout** и **cin**.

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_16.exe

Введите два значения расстояний:
Введите футы: 10
Введите дюймы: 10

Введите футы: 11
Введите дюймы: 11

dist1 = 10'-10"
dist2 = 11'-11"
dist3 = 11'-6.25"
Для продолжения нажмите любую клавишу
```

Задача 10-17

Перегрузка << и >> для файлов

```
// перегружаемые операции << и >> могут работать с файлами
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <fstream>
using namespace std;
class Distance              // класс английских расстояний
{ private:    int feet;    float inches;
  public:    Distance() : feet(0), inches(0.0)  {  }
    Distance(int ft, float in) : feet(ft), inches(in) {  }
    friend istream& operator >> (istream& s, Distance& d);
    friend ostream& operator << (ostream& s, Distance& d);
};
istream& operator >> (istream& s, Distance& d) //получить данные
{ char d1; // из файла или с клавиатуры с пом.перегруженного >>
  s >> d.feet >> d1 >> d1 >> d.inches >> d1;  return s;  }
ostream& operator << (ostream& s, Distance& d) // послать данные
  { s << d.feet << "\'-" << d.inches << '\"';   return s; }
int main()
{ SetConsoleCP(1251); SetConsoleOutputCP(1251);
  char ch;  Distance dist1;  ofstream ofile; ofile.open("DIST17.TXT");
  do { cout << "\nВведите расстояние: ";  cin >> dist1; ofile << dist1;
       cout << "Продолжать (y/n)? ";    cin >> ch;
  } while(ch != 'n');
  ofile.close();           // закрыть выходной поток
  ifstream ifile; ifile.open("DIST17.TXT");
  cout << "\nСодержимое файла:\n";
  while(true) { ifile >> dist1;  if( ifile.eof())break;
    cout << "Расстояние = " << dist1 <<endl; }     //вывод расстояний
  system("PAUSE");  return 0;
}
```

Следующий пример продемонстрирует, как перегружаются операторы **<<** и **>>** в классе **Distance** для работы с файловым вводом/выводом.

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_17.exe

Введите расстояние: 10'-11.1"
Продолжать (y/n)? y

Введите расстояние: -15'-10.5"
Продолжать (y/n)? n

Содержимое файла:
Расстояние = 10'-11.1"
Расстояние = -15'-10.5"
Для продолжения нажмите любую клавишу
```

Память как поток

Область памяти можно считать потоком и записывать в нее данные точно так же, как в файл. Это требуется, когда нужно выводить данные в определенном формате (например, оставлять только два знака после запятой) и одновременно использовать функцию текстового вывода, которая в качестве аргумента требует строку. Так обычно делают, вызывая функции вывода, при разработке приложений в GUI-средах, например в Windows, поскольку там этим функциям нужно передавать именно строку в качестве аргумента. Программисты на C, вероятно, сразу вспомнят, как они использовали **sprintf**() для этих целей. Семейство потоковых классов поддерживает такое форматирование данных в памяти. Для вывода в память существует специальный класс **ostrstream**, порожденный классом **ostream**. Для ввода из памяти служит класс **istrstream**, порожденный, соответственно, классом **istream**. Для объектов, которым требуется осуществлять одновременно ввод и вывод, создан класс **strstream** — наследник **iostream**.

Задача 10-18

Память как поток

```
// Запись форматированных данных в память
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <strstream>
#include <iomanip>          // для функции setiosflags()
using namespace std;
const int rr = 80;        // размер буфера
int main()
{ SetConsoleCP(1251); SetConsoleOutputCP(1251);
  char ch = 'x';           // тестовые данные
  int j = 77;
  double d = 67890.12345;
  char str1[] = "Kafka";
  char str2[] = "Freud";
  char membuff[rr];              // буфер в памяти
  ostrstream omem(membuff, rr); // создать потоковый объект
  omem << "ch=" << ch << endl       // вставить форматированные данные
       << "j=" << j << endl         // в объект
       << fixed   // формат с десятичной запятой (фиксированной)
       << setprecision(2)          // оставлять два знака после запятой
       << "d=" << d << endl
       << "str1=" << str1 << endl
       << "str2=" << str2 << endl
       << ends;           // закончить буфер символом '\0'
  cout << membuff;       // вывод содержимого буфера
  system("PAUSE");
  return 0;
}
```

Данный пример показывает, как это реально применить на практике. Создаем бу-фер данных в памяти. За-тем создайте объект **ostrstream**, используя этот буфер (его адрес и размер) в качестве аргументов конструктора потока. Теперь можно выводить форматированный текст в буфер, как если бы он был

```
D:\Практика Cpp задачи\Занятие 10\Pr_10_18.exe
ch=x
j=77
d=67890.12
str1=Kafka
str2=Freud
Для продолжения нажмите любую клавишу
```

Аргументы командной строки

Если вы использовали когда-нибудь старый добрый **MS DOS**, вам должно быть знакомо понятие аргументов командной строки, использующихся при запуске программ. Их типичное применение — передача имени файла с данными в приложение. Например, вы запускаете программу редактирования текстов и хотите сразу открыть документ, с которым будете работать. Для этого вы пишете: **C:\>wordproc afile.doc**

Здесь аргументом командной строки является **afile.doc**. Нам надо каким-то образом заставить программу на C++ распознавать такие обращения.

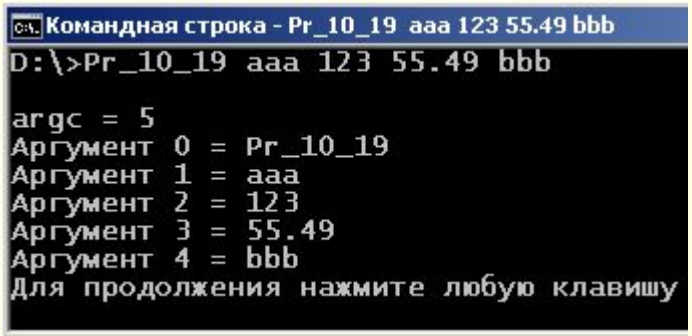
Задача 10-19

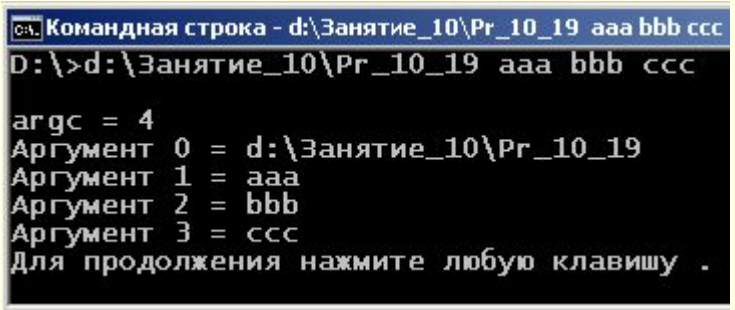
Аргументы командной строки

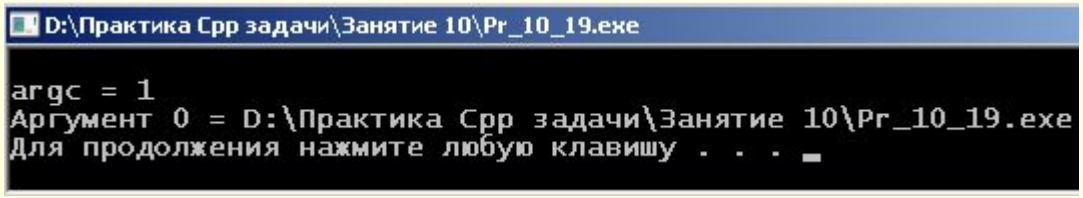
Следующая программа, приводимая в качестве примера, считывает и выводит на экран столько аргументов командной строки, сколько вы сможете напечатать (они отделяются пробелами)

```
// Демонстрация работы с аргументами командной строки
#include <cstdlib>
#include <iostream>
#include <windows.h>
using namespace std;

int main(int argc, char *argv[])
{ SetConsoleCP(1251); SetConsoleOutputCP(1251);
  cout << "\nargc = " << argc << endl; // число аргументов
  for(int j=0; j<argc; j++)             // вывести аргументы
    cout << "Аргумент " << j << " = " << argv[j] << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```







Список задач 10-го занятия

1. Ввод чисел
2. Ввод целого числа и дробного
3. Запись данных
4. Чтение данных
5. Строки с пробелами (запись)
6. Строки с пробелами (чтение)
7. Ввод символов
8. Вывод символов
9. Двоичный ввод/вывод

Список задач 10-го занятия

10. Запись объекта на диск
11. Чтение объекта с диска
12. Ввод/вывод множества объектов
13. Вычисление сдвига
14. Реагирование на ошибки
15. Как классы записывают и читают сами себя
16. Перегрузка cout и cin
17. Перегрузка << и >> для файлов
18. Память как поток
19. Аргументы командной строки

Резюме 10-го занятия

- В этой главе мы ознакомились с иерархией потоковых классов и показали, как обрабатывать различного рода ошибки ввода/вывода. Затем мы рассмотрели некоторые варианты файлового ввода/вывода. Файлы в C++ связаны с объектами различных классов: класс **ofstream** используется для файлового вывода, **ifstream** — для ввода, **fstream** — для ввода и вывода одновременно. Методы этих или базовых классов предназначены для выполнения операций ввода/вывода. Такие операции и функции, как **<<**, **put**() и **write**(), используются для вывода, а **>>**, **get**() и **read**() — для ввода.

Резюме 10-го занятия

- Функции **read**() и **write** работают с данными в двоичном режиме. Поэтому можно записывать в файлы объекты целиком, вне зависимости от типов данных, которые они содержат. Могут храниться как отдельные объекты, так и массивы и другие структуры, составленные из множества объектов. Файловый ввод/вывод может обрабатываться с использованием методов. За него могут отвечать как конкретные объекты, так и классы (с помощью статических функций).

Резюме 10-го занятия

- Проверка на наличие ошибок должна осуществляться после выполнения каждой файловой операции. Сам файловый объект принимает нулевое значение, если возникает какая-либо ошибка. К тому же для определения некоторых видов ошибок используются методы классов. Операции извлечения (**>>**) и вставки (**<<**) перегружаются для работы с пользовательскими типами данных. Память может представляться в виде потока, а данные в нее могут посылаться так же, как если бы это был файл.

Наиболее «популярные» ошибки

xxx.h: No such file or directory	не найден заголовочный файл 'xxx.h' (неверно указано его имя, он удален или т.п.)
'xxx' undeclared (first use this function)	функция или переменная 'xxx' неизвестна
missing terminating " character	не закрыты кавычки "
expected ;	нет точки с запятой в конце оператора **в предыдущей строке**
expected }	не закрыта фигурная скобка

Литература

1. Роберт Лафоре. Объектно-ориентированное программирование в C++
2. В. И. Шупляк. C++ Практический курс
3. В. В. Подбельский. Язык C++. Базовый курс

Конец