Объектно-ориентированное программирование

Потоки и файлы

Потоковые классы и их приемущество

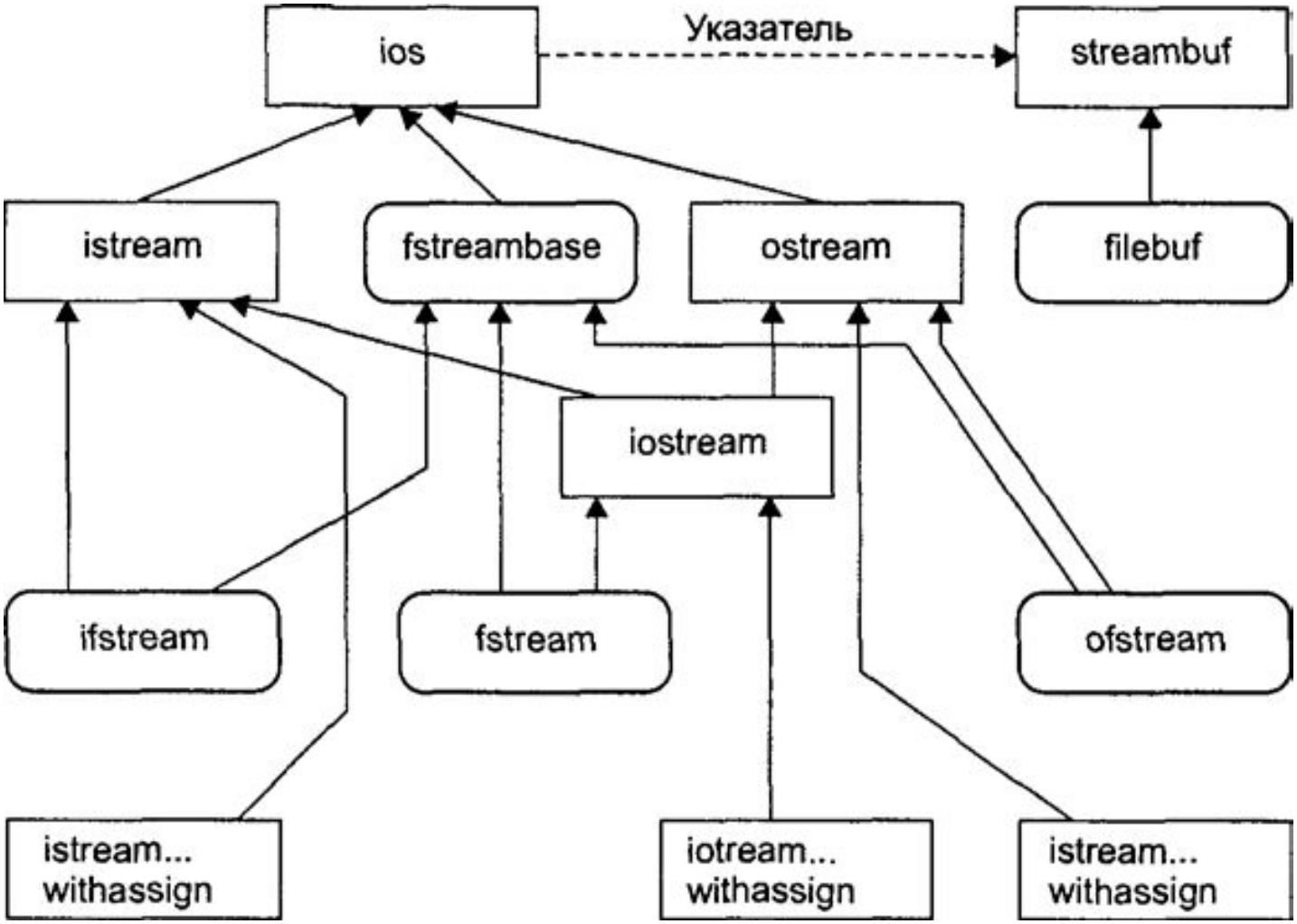
Поток — это общее название, обычно в C++ это поток. Потоковые объекты данных **cin** и **cout**

Одним из аргументов в пользу потоков является простота использования.

Если вам приходилось когда-нибудь использовать символ управления форматом %d. Ничего подобного в потоках вы не встретите, ибо 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, не говоря уж об эстетическом удовлетворении.

Иерархия потоковых классов



Класс **istream** содержит функции, как get(), getline(), read() и перегружаемую операцию извлечения (>>)

Класс **ostream** содержит функции put(), write() и перегружаемую операцию вставки (<<)

Класс **ios** и флаги форматирования

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, Ox 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

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

Манипуляторы ios

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

```
cout << "Jedem Das Seine." << endl;
cout << hex << var;
```

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

Функции ios

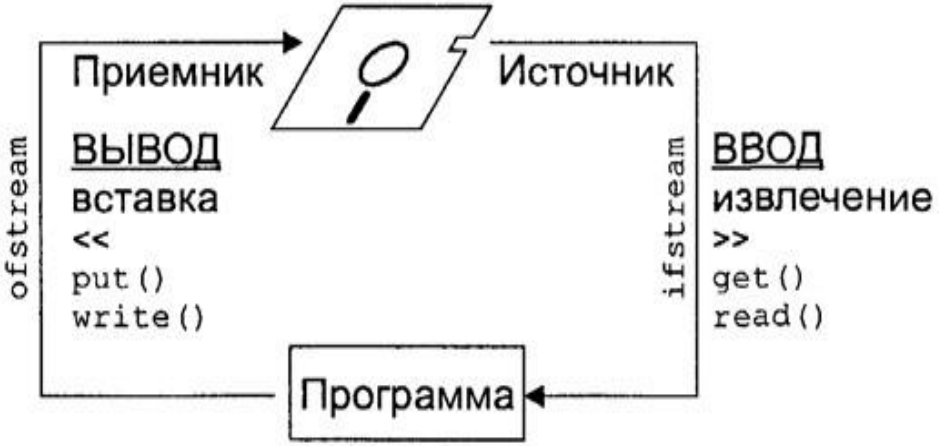
Функция	Назначение
ch=fill();	Возвращает символ заполнения (символ, которым заполняется неиспользуемая часть текстового поля; по умолчанию — пробел)
fill(ch);	Устанавливает символ заполнения
p=precision();	Возвращает значение точности (число выводимых знаков для формата с плавающей запятой)
precision(p);	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

Первый аргумент: устанавливаемые флаги	Второй аргумент: сбрасываемые флаги
dec, oct, hex	basefield
left, right, internal	adjustfield
scientific, fixed	floatfield

```
cout.width(14);
cout.fill('*');
cout.setf(ios:: left);
cout.unsetf(ios::left);
```

Класс **istream**

Класс istream, наследник класса ios, выполняет специфические действия по вводу данных — извлечение.



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

Функции **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

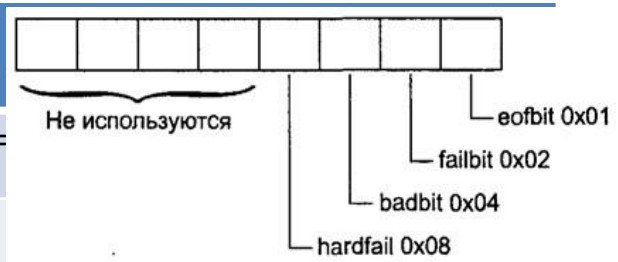
Класс **ostream**

Класс ostream предназначен для вывода (вставки в поток) данных.

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

Ошибки потоков. Биты статуса ошибки

Название	Значение
goodbit	Ошибок нет (флаги не установлены, значение =
eofbit	Достигнут конец файла



Название	Значение
failbit	Операция не выполнена (пользовательская ошибка, преждевременный EOF)
badbit	Недопустимая операция (нет ассоциированного streambuf)
hardfail	Неисправимая ошибка

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)

Ввод чисел и обработка ошибок

```
1  while(true) // Цикл до тех пор. пока
2  // ввод не будет корректным
3  {
4      cout <<"\nВведите целое число: ";
5      cin >> i;
6      if( cin.good() ) // если нет ошибок
7      {
8          cin.ignore(10, '\n'); //удалить разделитель строк
9          break;
10     } // выйти из цикла
11     cin.clear(); // Очистить биты ошибок
12     cout << "Неправильный ввод данных";
13     cin.ignore(10, '\n'); // Удалить разделитель строк
14 }
15 cout << "целое число: " << i; //целое без ошибок
```

Ввод при отсутствии данных

```
1  cout << "\nВведите целое число: ";
2  cin.unsetf(ios::skipws);//не игнорировать разделители
3  cin >> i;
4  if( cin.good() )
5  {
6      //ошибок нет
7  }
8  //ОШИБКА!
```

Уж теперь если пользователь и нажмет Enter, забыв ввести данные, то будет установлен флаг **failbit** и тем самым сгенерирован признак ошибки. После этого можно попросить пользователя ввести данные повторно или перемещать курсор так, чтобы экран не прокручивался.

Проверка ввода целых и дробных чисел

```
void Distance::getdist() // получение длины от пользователя
{
    string instr; // для входной строки
    while(true) // цикл, пока футы не будут правильными
    {
        cout << "\n\nВведите футы: ";
        cin.unsetf(ios::skipws);// не проп
        // разделители
        cin >> instr; // получить футы как
        if( isFeet(instr) ) // правильное
        { //да
            cin.ignore(10, '\n'); // съест
            // включая разделитель строк
            feet = atoi( instr.c_str() );
            // значение в целочисленное
            break; // выход из цикла 'whil
        } // нет, не целое
        cin.ignore(10, '\n'); // съесть си
        // разделитель строк
        cout << "Футы должны быть целыми < 1000\n";
    } //конец цикла while для футов
    while(true) // цикл проверки дюймов
    {
        cout << "Введите дюймы: ";
         cin.unsetf(ios::skipws);// не пропускать
        // разделители
        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
        { //(обычно вводят не цифры)
            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
}
```

```
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
}
```

Форматированный файловый ввод/вывод

Запись данных

```
#include <fstream> // для файлового ввода/вывода
#include <iostream>
#include <string>
using namespace std;
int main()
{
    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";
    return 0;
}
```

```
x
77
6.02
Kafka
Proust
```

Форматированный файловый ввод/вывод

Чтение

```
1  #include <fstream> // для файлового ввода/вывода
2  #include <iostream>
3  #include <string>
4  using namespace std;
5  int main()
6  {
7      char ch;
8      int j;
9      double d;
10     string str1;
11     string str2;
12     ifstream infile("fdata.txt"); // создать объект ifstream
13     // извлечь (прочесть) из него данные
14     infile >> ch >> j >> d >> str1 >> str2;
15     cout << ch << endl // вывести данные
16              << j << endl
17              << d << endl
18              << str1 << endl
19              << str2 << endl;
20     return 0;
21 }
```

```
x
77
6.02
Kafka
Proust
```

Строки с пробелами

```cpp
#include <fstream> // для операций
// файлового ввода/вывода
using namespace std;
int main()
{
    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";
    return 0;
}
```

```cpp
#include <fstream> // для файловых функций
#include <iostream>
using namespace std;
int main()
{
    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
    }
    return 0;
}
```

```cpp
while( !infile.eof() ) //Пока не EOF

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

while( infile ) //Пока нет ошибок
```

Ввод/вывод символов

```cpp
#include <fstream> // для файловых функций
#include <iostream>
#include <string>
using namespace std;
int WriteChars()
{
    string str = "Время - великий учитель, но, увы, "
    "оно убивает своих учеников. Берлиоз";
    ofstream outfile("TEST.TXT"); // Создать выходной файл
    for(int j=0; j<str.size(); j++) // каждый символ
        outfile.put( str[j] ); // записывать в файл

    cout << "Файл записан\n";
    return 0;
}

int ReadChars()
{
    char ch; // символ для считывания
    ifstream infile("TEST.TXT"); // входной файл
    while( infile ) // читать до EOF или ошибки
    {
        infile.get(ch); // считать символ
        cout << ch; // и вывести его
    }
    cout << endl;
    return 0;
}

int ReadBufChars()
{
    ifstream infile("TEST.TXT"); // создать входной файл
    cout << infile.rdbuf(); // передать его буфер в cout
    cout << endl;
    return 0;
}
```

Двоичный ввод/вывод

```
1  const int MAX = 100; // размер буфера
2  int buff[MAX]; // буфер для целых чисел
3
4  int main()
5  {
6      for(int j=0; j<MAX; j++) // заполнить буфер данными
7          buff[j] = j; // (0, 1, 2, ...)
8      // создать выходной поток
9      ofstream os("edata.dat", ios::binary);
10     // записать в него
11     os.write(reinterpret_cast<char*>(buff), MAX*sizeof(int) );
12     os.close(); // должен закрыть его
13     for(j=0; j<MAX; j++) // стереть буфер
14         buff[j] = 0;
15     // создать входной поток
16     ifstream is("edata.dat", ios::binary);
17     // читать из него
18     is.read( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
19     for(j=0; j<MAX; j++) // проверка данных
20         if( buff[j] != j )
21         { cerr << "Некорректные данные!\n"; return 1; }
22     cout << "Данные корректны\n";
23     return 0;
24 }
```

Объектный ввод/вывод

```
class person // класс person
{
protected:
    char name[80]; // имя человека
    short age; // возраст
public:
    void getData() // получить данные о человеке
    {
        cout << "Введите имя: "; cin >> name;
        cout << "Введите возраст: "; cin >> age;
    }
    void showData() // вывести данные
    {
        cout << "Имя: " << name << endl;
        cout << "Возраст: " << age << endl;
    }
};
////////////////////////////////////////////////////////////
int ReadObject()
{
    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));
    return 0;
}

int WriteObject()
{
    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
    return 0;
}
```

Для корректной работы программы чтения и записи объектов должны иметь в виду один класс объектов. Например, объекты класса person имеют длину ровно 82 байта, из которых 80 отведено под имя человека, 2 — под возраст в формате short. Если бы программы не знали длину полей, то одна из них не смогла бы корректно прочитать из файла то, что записала другая.

Ввод/вывод множества объектов

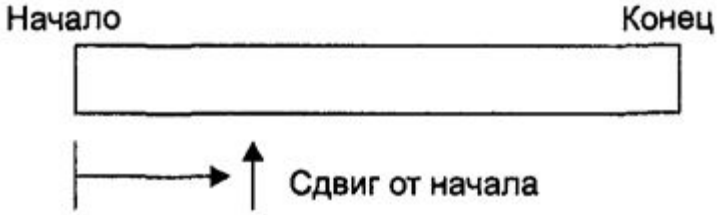
```
class person // класс person
{
protected:
    char name[80]; // имя человека
    int age; // его возраст
public:
    void getData() // получить данные о человеке
    {
        cout << "\n Введите имя: "; cin >> name;
        cout << " Введите возраст: "; cin >> age;
    }
    void showData() // вывод данных о человеке
    {
        cout << "\n Имя: " << name;
        cout << "\n Возраст: " << age;
    }
};
```

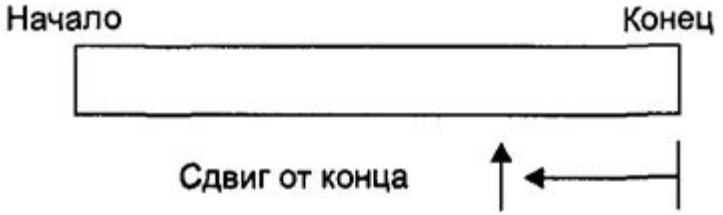
```
int main()
{
    char ch;
    person pers; // создать объект person
    fstream file; // создать входной/выходной файл
    // открыть для дозаписи
    file.open("GROUP.DAT", ios::app | ios::out |
    ios::in | ios::binary );
    do // данные от пользователя - в файл
    {
        cout << "\nВведите данные о человеке:";
        pers.getData(); // получить данные
        // записать их в файл
        file.write( reinterpret_cast<char*>(&pers), sizeof(pers) );
        cout << "Продолжить ввод (y/n)? ";
        cin >> ch;
    }
    while(ch=='y'); // выход по 'n'
    file.seekg(0); // поставить указатель на начало файла
    // считать данные о первом человеке
    file.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
    while( !file.eof() ) // Выход по EOF
    {
        cout << "\nПерсона:"; //вывести данные
        pers.showData(); //считать данные о следующем
        file.read(reinterpret_cast<char*>(&pers),sizeof(pers));
    }
    cout << endl;
    return 0;
}
```

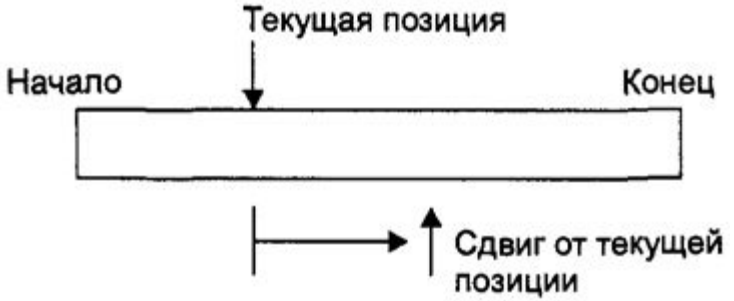
Биты режимов

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, позиция в файле







seekp(-10, ios::end);

beg означает начало файла
cur — текущую позицию указателя файла
end — это конец файла

Указатели файлов, позиция в файле

```
class person // класс person
{
protected:
    char name[80]; // имя человека
    int age; // его возраст
public:
    void getData() // получить данные о человеке
    {
        cout << "\n 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()
{
    person pers; // создать объект person
    ifstream infile; // создать входной файл
    infile.open("GROUP.DAT",ios::in|ios::binary); // открыть
    // файл
    infile.seekg(0, ios::end); // установить 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); // умножить размер
    // данных под персону на число персон
    infile.seekg(position); // число байт от начала
    // прочитать одну персону
    infile.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
    pers.showData(); //вывести одну персону
    cout << endl;
    return 0;
}
```

Обработка ошибок файлового ввода/вывода

```
1  const int MAX = 1000;
2  int buff[MAX];
3  int main()
4  {
5      for(int j=0; j<MAX; j++) // заполнить буфер данными
6      buff[j] = j;
7      ofstream os; // создать выходной поток
8      // открыть его
9      os.open("a:edata.dat", ios::trunc | ios::binary);
10     if(!os)
11         { cerr << "Невозможно открыть выходной файл\n"; exit(1); }
12     cout << "Идет запись...\n"; // записать в него содержимое
13     // буфера
14     os.write( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
15     if(!os)
16         { cerr << "Запись в файл невозможна\n"; exit(1); }
17     os.close(); // надо закрыть поток
18     for(j=0; j<MAX; j++) // очистить буфер
19         buff[j] = 0;
20     ifstream is; // создать входной поток
21     is.open("a:edata.dat", ios::binary);
22     if(!is)
23         { cerr << "Невозможно открыть входной файл\n";exit(1); }
24     cout << "Идет чтение...\n"; // чтение файла
25     is.read( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
26     if(!is)
27         { cerr << "Невозможно чтение файла\n"; exit(1); }
28     for(j=0; j<MAX; j++) // проверять данные
29         if( buff[j] != j )
30             { cerr << "\nДанные некорректны\n"; exit(1); }
31     cout << "Данные в порядке\n";
32     return 0;
33 }
```

Обработка ошибок файлового ввода/вывода

```
1  #include <fstream> // для файловых функций
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6      ifstream file;
7      file.open("a:test.dat");
8      if( !file )
9          cout << "\nНевозможно открыть GROUP.DAT";
10     else
11         cout << "\nФайл открыт без ошибок.";
12     cout << "\nfile = " << file;
13     cout << "\nКод ошибки = " << file.rdstate();
14     cout << "\ngood() = " << file.good();
15     cout << "\neof() = " << file.eof();
16     cout << "\nfail() = " << file.fail();
17     cout << "\nbad() = " << file.bad() << endl;
18     file.close();
19     return 0;
20 }
```