Потоковые классы (продолжение)

Ошибки потоков

В базовом классе ios определено поле state, которое представляет собой состояние потока в виде совокупности битов:

```
enum io_state   {
    goodbit     = 0x00,     // Нет ошибок
    eofbit      = 0x01,     // Достигнут конец файла
    failbit     = 0x02,     // Ошибка форматирования или преобразования
    badbit      = 0x04,     // Серьезная ошибка, после которой
                            // пользоваться потоком невозможно
    hardfail    = 0x08      // Неисправность оборудования
};
```

Ошибки потоков

Состоянием потока можно управлять с помощью перечисленных ниже методов и операций:

`int rdstate()` — возвращает текущее состояние потока;

`int eof()` — возвращает ненулевое значение, если установлен флаг `eofbit`;

`int fail()` — возвращает ненулевое значение, если установлен один из флагов `failbit`, `badbit` или `hardfail`;

`int bad()` — возвращает ненулевое значение, если установлен один из флагов `badbit` или `hardfail`;

`int good()` — возвращает ненулевое значение, если сброшены все флаги ошибок;

`void clear(int = 0)` — параметр принимается в качестве состояния ошибки, при отсутствии параметра состояние ошибки устанавливается 0;

`operator void*()` — возвращает нулевой указатель, если установлен хотя бы один бит ошибки;

`operator !()` — возвращает ненулевой указатель, если установлен хотя бы один бит ошибки.

Ошибки потоков

Далее приведены часто используемые операции с флагами состояния потока.

```
// Проверить, установлен ли флаг flag:
if(stream_obj.rdstate() & ios::flag)
// Сбросить флаг flag:
stream_obj.clear(rdstate() & ~ios::flag)
// Установить флаг flag:
stream_obj.clear(rdstate() | ios::flag)
// Установить флаг flag и сбросить все остальные:
stream_obj.clear( ios::flag)
// Сбросить все флаги:
 stream_obj.clear()
```

Ошибки потоков

Операция void*() неявно вызывается всякий раз, когда поток сравнивается с 0. Это позволяет записывать циклы вида:

```
while (stream_obj){
   // Все в порядке, можно производить ввод/вывод
}
```

Ошибки потоков

В приведенном ниже примере показана работа функции rdstate(). Программа выводит на экран содержимое текстового файла, имя которого задается в командной строке. При наличии ошибки функция сообщает об этом посредством CheckStatus().

```
#include <iostream.h>
#include <fstream.h>
void CheckStatus(ifstream &in);
int main(int argc, char *argv[ ]){
   if(argc != 2){
      cout << "Usage: <program_name> <file_name>" << endl;
      return 1;
   }
   ifstream in(argv[1], ios::in|ios::nocreate);
   if(!in){
      cout << "Cannot open file" << argv[1] << endl;
      return 1;
   }
   char c;
   while(in.get(c)){
      cout << c; CheckStatus(in);
   }
```

Ошибки потоков

```
    CheckStatus(in); // контроль финального состояния
    in.close();
    return 0;
}
void CheckStatus(ifstream &in){
    int i;
    i = in.rdstate();
    if(i & ios::eofbit)
       cout << "EOF is occured" << endl;
    else if(i & ios::failbit)

          cout << "Not fatal input/output error" << endl;
       else if(i & ios::badbit)
          cout << "Fatal input/output error" << endl;
 }
```

Файловые потоки

Под файлом обычно подразумевается именованная информация на внешнем носителе, например, на жестком или гибком магнитном диске. Логически файл можно представить как конечное количество последовательных байтов, поэтому такие устройства, как дисплей, клавиатуру и принтер также можно рассматривать как частные случаи файлов.

Файловые потоки

По способу доступа файлы можно разделить на *последовательные,* чтение и запись в которых производятся с начала байт за байтом, и *файлы с произвольным доступом,* допускающие чтение и запись в указанную позицию

Файловые потоки

Стандартная библиотека содержит три класса для работы с файлами:

ifstream — класс входных файловых потоков;
ofstream— класс выходных файловых потоков;
fstream — класс двунаправленных файловых потоков.

Файловые потоки

Эти классы являются производными от классов istream, ostream и iostream соответственно, поэтому они наследуют перегруженные операции « и » , флаги форматирования, манипуляторы, методы, состояние потоков и т. д.

Файловые потоки

Использование файлов в программе предполагает следующие операции:

- создание потока;
- открытие потока и связывание его с файлом;
- обмен (ввод/вывод);
- уничтожение потока;
- закрытие файла

Файловые потоки

Каждый класс файловых потоков содержит конструкторы, с помощью которых можно создавать объекты этих классов различными способами.

• Конструкторы без параметров создают объект соответствующего класса, не связывая его с файлом:

ifstream();

ofstream();

fstream();

Файловые потоки

• Конструкторы с параметрами создают объект соответствующего класса, открывают файл с указанным именем и связывают файл с объектом:

ifstream(const char *name, int mode = ios::in);

ofstream(const char *name, int mode = ios::out | ios::trunc);

fstream(const char *name, int mode = ios::in | ios::out);

Файловые потоки

Вторым параметром конструктора является режим открытия файла.

Если установленное по умолчанию значение не устраивает программиста, можно указать другое, составив его из битовых масок, определенных в классе ios:

Файловые потоки

```
enum open_mode{
    in          = 0x01,     // Открыть для чтения
    out         = 0x02,     // Открыть для записи
    ate         = 0x04,     // Установить указатель на конец файла
    app         = 0x08,     // Открыть для добавления в конец
    trunc       = 0x10,     // Если файл существует, удалить
    nocreate    = 0x20,     // Если файл не существует, выдать ошибку
    noreplace   = 0x40,     // Если файл существует, выдать ошибку
    binary      = 0x80,     // Открыть в двоичном режиме
};
```

Файловые потоки

В таблице 10.2 приведено соответствие между битовыми масками класса ios и режимами открытия файла, описанными в <stdio.h>.

Таблица 10.2. Режимы открытия файла

Комбинация флагов ios					Эквивалент
binary	**in**	**out**	**trunc**	**app**	**stdio**
		+			"w"
		+		+	"a"
		+	+		"w"
	+				"r"
	+	+			"r+"
	+	+	+		"w+"
+		+			"wb"
+		+		+	"ab"
+		+	+		"wb"
+	+				"rb"
+	+	+			"r+b"
+	+	+	+		"w+b"

Файловые потоки

Открыть файл в программе можно с использованием либо конструкторов, либо метода open, имеющего такие же параметры, как и в соответствующем конструкторе, например:

```
ifstream inpf ("input.txt", , ios::in|ios::nocreate);    // Использование
конструктора
if (!inpf){
   cout << "Невозможно открыть файл для чтения"; return 1;

}
ofstream f;
f.open("output.txt"); // Использование метода open
if (!f){
   cout << "Невозможно открыть файл для записи";
   return 1;
}
```

Файловые потоки

Чтение и запись выполняются либо с помощью операций чтения и извлечения, аналогичных потоковым классам, либо с помощью методов классов.

Файловые потоки

Пример использования методов (программа выводит на экран содержимое файла):

```
#include <fstream.h>
int main(){
   char text[81], buf[81];
   cout << "Введите имя файла:";
   cin >> text;
   ifstream f(text, ios::in|ios::nocreate);
   if (!f){
      cout << "Ошибка открытия файла"; return 1;
   }
   while (!f.eof()){
      f.getline(buf, 81);
      cout << buf << endl;
   }
   return 0;
}
```

Файловые потоки

Для закрытия потока определен метод close(), но поскольку он неявно выполняется деструктором, явный вызов необходим только тогда, когда требуется закрыть поток раньше конца его области видимости.

Строковые потоки

Строковые потоки позволяют считывать и записывать информацию из областей оперативной памяти так же, как из файла, с консоли или на дисплей.

Строковые потоки

В стандартной библиотеке определено три класса строковых потоков:

istringstream — входные строковые потоки;
ostringstream — выходные строковые потоки;
stringstream — двунаправленные строковые потоки

Строковые потоки

Эти классы определяются в заголовочном файле <sstream> и являются производными от классов istream, ostream и iostream соответственно, поэтому они наследуют перегруженные операции « и » , флаги форматирования, манипуляторы, методы, состояние потоков и т. д.

Строковые потоки

Участки памяти, с которыми выполняются операции чтения и извлечения, по стандарту определяются как строки C++ (класс string).

Строковые потоки

Строковые потоки создаются и связываются с этими участками памяти с помощью конструкторов:

```
explicit istringstream(int mode = ios::in);
explicit istringstream(const string& name, int mode = ios::in);
explicit ostringstream(int mode = ios::out);
explicit ostringstream(const string& name, int mode = ios::out);
explicit stringstream(int mode = ios::in | ios::out);
explicit stringstream(const string& name, int mode = ios::in | ios::out);
```

Строковые потоки

Строковые потоки являются некоторым аналогом функций sscanf и sprintf библиотеки C и могут применяться для преобразования данных, когда они заносятся в некоторый участок памяти, а затем считываются в величины требуемых типов. Эти потоки могут применяться также для обмена информацией между модулями программы.

Строковые потоки

В строковых потоках описан метод str, возвращающий копию строки или устанавливающий ее значение:
string str() const;
void str(const string & s);
Проверять строковый поток на переполнение не требуется, поскольку размер строки изменяется динамически.

Строковые потоки

В приведенном ниже примере строковый поток используется для формирования сообщения, включающего текущее время и передаваемый в качестве параметра номер:

```
#include <sstream>
#include <string>
#include <iostream>
#include <ctime>
using namespace std;
string message( int i){
   ostringstream os;
   time_t t;
   time(&t);
   os << " time: " << ctime(&t) << " number: " << i << endl;
   return os.str();
}
int main(){
   cout << message(22);
   return 0;
}
```

Потоки и типы, определенные пользователем

Для ввода и вывода в потоках используются перегруженные для всех стандартных типов операции чтения и извлечения « и » .

При этом выбор конкретной операции определяется типом фактических параметров.

Потоки и типы, определенные пользователем

Для того чтобы вводить и выводить величины типов, определенных пользователем, требуется перегрузить эти операции.

Это бинарные операции, левым операндом которых является объект-поток, а правым — объект, который требуется извлечь или поместить в этот поток.

Возвращаемое значение должно быть ссылкой на поток, чтобы можно было организовывать цепочки операций, как и в случае стандартных типов.

Потоки и типы, определенные пользователем

Пусть, например, в программе определен класс MyClass:

```
class MyClass{
    int   x;
    float y;
    …
}
```

Потоки и типы, определенные пользователем

Для того чтобы вводить и выводить объекты этого класса, требуется определить в классе MyClass операции следующего вида:

```
// Вывод:
friend ostream& operator << (ostream& out, MyClass& C){
   return out << "x = " << C.x << "  y = " << C.y;
}
// Ввод:
friend istream& operator >> (istream& in, MyClass& C){
   cout << "Введите x: ";in >> C.x;
   cout << "Введите y: ";in >> C.y;
   return in;
}
```

Потоки и типы, определенные пользователем

После этого в программе можно использовать объекты класса MyClass в операциях ввода и вывода наряду с величинами стандартных типов:

```
#include <iostream.h>
class MyClass{
      int   x;
      float y;
 public:
      MyClass(int nx = 1, float ny = 0.01){x = nx; y = ny;}
      friend ostream& operator<< (ostream& out, MyClass& C){
         return out << "x = " << C.x << "  y = " << C.y;
      }
      friend istream& operator>> (istream& in, MyClass& C){
         cout << "Введите x: ";   in >> C.x;
         cout << "Введите y: ";   in >> C.y;
         return in;
```

Потоки и типы, определенные пользователем

```
        }
};
int main(){
    MyClass C;
    cout << C << endl;
    MyClass C1(100, 100);
    cout << C1 << endl;
    MyClass C2;
    cin >> C2; cout << C2 << endl;
    return 0;
}
```

Экземпляр C класса MyClass создается с параметрами конструктора по умолчанию, поэтому на экран будет выведено:

```
x = 1 y = 0.01
```

Экземпляр C1 класса MyClass создается с параметрами 100, 100:

```
x = 100 y = 100
```

После создания экземпляра C2 будет выведено приглашение ко вводу x и y, а затем введенные с клавиатуры значения будут выведены на экран.