Потоковые классы

Потоковые классы

Поток — это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Потоки C++, в отличие от функций ввода/вывода в стиле C, обеспечивают надежную работу как со стандартными, так и с определенными пользователем типами данных, а также единообразный и понятный синтаксис.

Потоковые классы

Чтение данных из потока называется *извлечением,* вывод в поток — *помещением,* или *включением.*

Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер).

Потоковые классы

Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти — *буфер.*

Фактическая передача данных выполняется при выводе после заполнения буфера, а при вводе — если буфер исчерпан.

Потоковые классы

По направлению обмена потоки можно разделить на *входные* (данные вводятся в память), *выходные* (данные выводятся из памяти) и *двунаправленные* (допускающие как извлечение, так и включение)

Потоковые классы

По виду устройств, с которыми работает поток, можно разделить потоки на стандартные, файловые и строковые.

Потоковые классы

Стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея, *файловые потоки* — для обмена информацией с файлами на внешних носителях данных (например, на магнитном диске), а *строковые потоки* — для работы с массивами символов в оперативной памяти.

Потоковые классы

Для поддержки потоков библиотека C++ содержит иерархию классов, построенную на основе двух базовых классов — ios и streambuf.

Класс ios содержит общие для ввода и вывода поля и методы, класс streambuf обеспечивает буферизацию потоков и их взаимодействие с физическими устройствами.

Потоковые классы

От этих классов наследуется класс istream для входных потоков и ostream — для выходных.

Два последних класса являются базовыми для класса iostream, реализующего двунаправленные потоки. Ниже в иерархии классов располагаются файловые и строковые потоки. Далее перечислены часто используемые классы потоков.

Потоковые классы

ios — базовый класс потоков;
istream — класс входных потоков;
ostream — класс выходных потоков;
iostream — класс двунаправленных потоков;
istringstream — класс входных строковых потоков;
ostringrstream — класс выходных строковых потоков;
stringstream — класс двунаправленных строковых потоков;
ifstream — класс входных файловых потоков;
ofstream — класс выходных файловых потоков;
fstream — класс двунаправленных файловых потоков.

Потоковые классы

Описания классов находятся в заголовочных файлах:
<ios> — базовый класс потоков ввода/вывода;
<iosfwd> — предварительные объявления средств ввода/вывода;
<istream> — шаблон потока ввода;
<ostream> — шаблон потока вывода;
<iostream> — стандартные объекты и операции с потоками ввода/вывода;
<fstream> — потоки ввода/вывода в файлы;
<sstream> — потоки ввода/вывода в строки;
<streambuf> — буферизация потоков ввода/вывода;
<iomanip> — манипуляторы

Потоковые классы

Подключение к программе файлов **<fstream>** и **<sstream>** автоматически подключает и файл **<iostream>,** так как он является для них базовым.

Потоковые классы

Основным преимуществом потоков по сравнению с функциями ввода/вывода, унаследованными из библиотеки C, является контроль типов, а также расширяемость, то есть возможность работать с типами, определёнными пользователем.
Для этого требуется переопределить операции потоков.

Потоковые классы

Кроме того, потоки могут работать с расширенным набором символов **wchar_t**.

Стандартные потоки

Заголовочный файл <iostream> содержит, кроме описания классов для ввода/вывода, четыре предопределенных объекта:

Объект	Класс	Описание
cin	istream	Связывается с клавиатурой (стандартным буферизованным вводом)
cout	ostream	Связывается с экраном (стандартным буферизованным выводом)
cerr	ostream	Связывается с экраном (стандартным небуферизованным выводом), куда направляются сообщения об ошибках
clog	ostream	Связывается с экраном (стандартным буферизованным выводом), куда направляются сообщения об ошибках

Стандартные потоки

Эти объекты создаются при включении с программу заголовочного файла <iostream>, при этом становятся доступными связанные с ними средства ввода/вывода.

Имена этих объектов можно переназначить на другие файлы или символьные буферы

Стандартные потоки

В классах istream и ostream *операции извлечения из потока » и помещения в поток «* определены путем перегрузки операций сдвига. Пример:

```
int main(){
    int i;
    cin >> i;
    cout << "Вы ввели " << i;
    return 0;
}
```

Стандартные потоки

Операции извлечения и чтения в качестве результата своего выполнения формируют ссылку на объект типа istream для извлечения и ссылку на ostream — для чтения. Это позволяет формировать цепочки операций, что проиллюстрировано последним оператором приведенного примера. Вывод при этом выполняется слева направо.

Стандартные потоки

Как и для других перегруженных операций, для вставки и извлечения невозможно изменить приоритеты, поэтому в необходимых случаях используются скобки:

```
// Скобки не требуются – приоритет сложения больше, чем << :
cout << i + j;
// Скобки необходимы – приоритет операции отношения меньше, чем << :
cout << (i < j);
cout << (i << j);     // Правая операция << означает сдвиг
```

Стандартные потоки

Величины при вводе должны разделяться пробельными символами (пробелами, знаками табуляции или перевода строки). Извлечение прекращается, если очередной символ оказался недопустимым.

Стандартные потоки

Если в операции помещения в поток встречается выражение, изменяющее некоторую переменную, то она не должна присутствовать в цепочке операций более одного раза, поскольку в таком случае результат может зависеть от реализации компилятора.

Стандартные потоки

Операции « и » перегружены для всех встроенных типов данных, что позволяет автоматически выполнять ввод и вывод в соответствии с типом величин. Это означает, что при вводе последовательность символов преобразуется во внутреннее представление величины, стоящей справа от знака извлечения, а при выводе выполняется обратное преобразование.

Например:

Стандартные потоки

```
int main(){
    int i = 0xD;
    double d;
    // Символы из потока ввода преобразуются в double:
    cin >> d;
    // int и double преобразуются в строку символов:
    cout << i << ' ' << d;
    return 0;
}
```

Стандартные потоки

Рассмотрим, как обрабатываются с помощью этих операций данные различных типов.
Числовые значения можно вводить в десятичной или шестнадцатеричной системе счисления (с префиксом 0x) со знаком или без знака. Вещественные числа представляются в форме с фиксированной точкой или с порядком.
Например, если для предыдущего примера с клавиатуры вводится последовательность символов 1.53e-2, она интерпретируется как вещественное число с порядком и преобразуется во внутреннее представление, соответствующее типу double. При выводе выполняется обратное преобразование, и на экран выводятся символы:

Стандартные потоки

```
13 0.0153
```

Поскольку ввод буферизован, помещение в буфер ввода происходит после нажатия клавиши перевода строки, после чего из буфера выполняется операция извлечения из потока. Это дает возможность исправлять введенные символы до того, как нажата клавиша Enter.

При вводе строк извлечение происходит до ближайшего пробела (вместо него в строку заносится нуль-символ):

```
char str1[100], str2[100];
cin >> str1 >> str2;
```

Стандартные потоки

Если с клавиатуры вводится строка "раз два три четыре пять", переменные str1и str2 примут значения "раз" и "два" соответственно, а остаток строки воспринят не будет. При необходимости ввести из входного потока строку целиком (до символа '\n') пользуются методами get или getline.

Стандартные потоки

Значения указателей выводятся в шестнадцатеричной системе счисления. Под любую величину при выводе отводится столько позиций, сколько требуется для ее представления. Чтобы отделить одну величину от другой, используются пробелы:

```
cout << i << ' ' << d << "      " << j;
```

Стандартные потоки

Если формат вывода, используемый по умолчанию, не устраивает программиста, он может скорректировать его с помощью методов классов ввода/вывода, флагов форматирования и так называемых манипуляторов.

Форматирование данных

В потоковых классах форматирование выполняется тремя способами
- с помощью флагов, манипуляторов и форматирующих методов.

Флаги и форматирующие методы

Флаги представляют собой отдельные биты, объединенные в поле x_flags типа long класса ios.

Флаги перечислены в табл.1.

Флаг	Положение	Умолчание	Описание действия при установленном бите
skipws	0x0001	+	При извлечении пробельные символы игнорируются
left	0x0002		Выравнивание по левому краю поля
right	0x0004	+	Выравнивание по правому краю поля
internal	0x0008		Знак числа выводится по левому краю, число — по правому. Промежуток заполняется символами x_fill (см. ниже), по умолчанию пробелами
dec	0x0010	+	Десятичная система счисления
oct	0x0020		Восьмеричная система счисления
hex	0x0040		Шестнадцатеричная система счисления
showbase	0x0080		Выводится основание системы счисления (0x для шестнадцатеричных чисел и 0 для восьмеричных)
showpoint	0x0100		При выводе вещественных чисел печатать десятичную точку и дробную часть
uppercase	0x0200		При выводе использовать символы верхнего регистра
showpos	0x0400		Печатать знак при выводе положительных чисел
scientific	0x0800		Печатать вещественные числа в форме мантиссы с порядком
fixed	0x1000		Печатать вещественные числа в форме с фиксированной точкой (точность определяется полем x_precision, см. ниже)
unitbuf	0x2000		Выгружать буферы всех потоков после каждого вывода
stdio	0x4000		Выгружать буферы потоков stdout и stderr после каждого вывода

Таблица 1 Флаги форматирования

Флаги и форматирующие методы

ПРИМЕЧАНИЕ

Флаги (left, right и internal), (dec, oct и hex), а также (scientific и fixed) взаимно исключают друг друга, то есть в каждый момент может быть установлен только один флаг из каждой группы.

Флаги и форматирующие методы

Для управления флагами в классе ios есть *методы* flags, setf и unsetf:

`long ios::flags();`	— возвращает текущие флаги потока;
`long ios::flags(long);`	— присваивает флагам значение параметра;
`long ios::setf(long, long);`	— присваивает флагам, биты которых установлены в первом параметре, значение соответствующих битов второго параметра;
`long ios::setf(long);`	— устанавливает флаги, биты которых установлены в параметре;
`long ios::unsetf(long);`	— сбрасывает флаги, биты которых установлены в параметре.

Все функции возвращают прежние флаги потока.

Флаги и форматирующие методы

Кроме флагов, для форматирования используются следующие *поля* класса ios:

`int x_width` — минимальная ширина поля вывода;

`int x_precision` — количество цифр в дробной части при выводе вещественных чисел с фиксированной точкой или общее количество значащих цифр при выводе в форме с мантиссой и порядком;

`int x_fill` — символ заполнения поля вывода.

Для управления этими полями используются *методы* `width`, `precision` и `fill`:

`int ios::width()` — возвращает значение ширины поля вывода;

`int ios::width(int)` — устанавливает ширину поля вывода в соответствии со значением параметра;

`int ios::precision()` — возвращает значение точности представления при выводе вещественных чисел;

`int ios::precision(int)` — устанавливает значение точности представления при выводе вещественных чисел, возвращает старое значение точности;

`char fill()` — возвращает текущий символ заполнения;

`char fill(char)` — устанавливает значение текущего символа заполнения, возвращает старое значение символа.

Флаги и форматирующие методы

Перед установкой некоторых флагов требуется сбросить флаги, которые не могут быть установлены одновременно с ними. Для этого удобно использовать вторым параметром метода **setf** перечисленные ниже *статические константы* класса **ios**:

```
adjustfield  (left | right | internal)
basefield       (dec | oct | hex)
floatfield      (scientific | fixed)
```

Флаги и форматирующие методы

Пример форматирования при выводе с помощью флагов и методов:

```
#include <iostream.h>
int main(){
    long a = 1000, b = 077;
    cout.width(7);
    cout.setf(ios::hex | ios::showbase | ios::uppercase);
    cout << a;
    cout.width(7);
    cout << b << endl;
    double d = 0.12, c = 1.3e-4;
    cout.setf(ios::left);
    cout << d << endl;
    cout << c;
    return 0;
}
```

Флаги и форматирующие методы

В результате работы программы в первой строке будут прописными буквами выведены переменные a и b в шестнадцатеричном представлении, под каждую из них отводится по 7 позиций (функция width действует только на одно выводимое значение, поэтому ее вызов требуется повторить дважды). Значения переменных c и d прижаты к левому краю поля:

```
   0X3E8   0X3F
0.12
0.00013
```

Манипуляторы

Манипуляторами называются функции, которые можно включать в цепочку операций помещения и извлечения для форматирования данных.
Манипуляторы делятся на *простые,* не требующие указания аргументов, и *параметризованные.*
Пользоваться манипуляторами более удобно, чем методами установки флагов форматирования.

Простые манипуляторы

Ниже перечислены манипуляторы, не требующие указания аргументов.

dec — устанавливает при вводе и выводе флаг десятичной системы счисления;

oct — устанавливает при вводе и выводе флаг восьмеричной системы счисления;

hex — устанавливает при вводе и выводе флаг шестнадцатеричной системы счисления;

ws — устанавливает при вводе извлечение пробельных символов;

endl — при выводе включает в поток символ новой строки и выгружает буфер;

ends — при выводе включает в поток нулевой символ;

flush — при выводе выгружает буфер.

Изменения системы счисления действуют до следующего явного изменения.

Пример:

```
cout << 13 << hex << ' ' << 13 << oct << ' ' << 13 << endl;
```

Если другие значения флагов установлены по умолчанию, будет выведено:

```
13 d 15
```

Параметризованные манипуляторы

Ниже перечислены манипуляторы, требующие указания аргумента. Для их использования требуется подключить к программе заголовочный файл <iomanip>.

`setbase(int n)`	— задает основание системы счисления (n = 8, 16, 10 или 0). 0 является основанием по умолчанию (десятичное, кроме случаев, когда вводятся 8- или 16-ричные числа);
`resetiosflags(long)`	— сбрасывает флаги состояния потока, биты которых установлены в параметре;
`setiosflags(long)`	— устанавливает флаги состояния потока, биты которых в параметре равны 1;
`setfill(int)`	— устанавливает символ-заполнитель с кодом, равным значению параметра;
`setprecision(int)`	— устанавливает максимальное количество цифр в дробной части для вещественных чисел в форме с фиксированной точкой (`flnar fixed`) или общее количество значащих цифр для чисел в форме с мантиссой и порядком (`flnar scientific`);
`setw(int)`	— устанавливает максимальную ширину поля вывода.

Параметризованные манипуляторы

Пример использования параметризованных манипуляторов:

```
int main(){
   double d[] = {1.234, -12.34567, 123.456789, -1.234, 0.00001};
   cout  << setfill('.') << setprecision(4)
         << setiosflags(ios::showpoint | ios::fixed);
   for (int i = 0; i < 5; i++)
      cout << setw(12) << d[i] << endl;
   return 0;
}
```

Результат работы программы:

```
......1.2340
....-12.3457
....123.4568
.....-1.2340
......0.0000
```

Методы обмена с потоками

В потоковых классах наряду с операциями извлечения » и включения « определены методы для неформатированного чтения и записи в поток (при этом преобразования данных не выполняются).

Методы обмена с потоками

Ниже приведены функции чтения, определенные в классе `istream`.

`gcount()`	— возвращает количество символов, считанных с помощью последней функции неформатированного ввода;
`get()`	— возвращает код извлеченного из потока символа или EOF;
`get(c)`	— возвращает ссылку на поток, из которого выполнялось чтение, и записывает извлеченный символ в c;
`get(buf,num,lim='\n')`[1]	— считывает `num-1` символов (или пока не встретится символ `lim`) и копирует их в символьную строку `buf`. Вместо символа `lim` в строку записывается признак конца строки ('\0'). Символ `lim` остается в потоке. Возвращает ссылку на текущий поток;
`getline(buf, num, lim='\n')`	— аналогична функции `get`, но копирует в `buf` и символ `lim`;
`ignore(num = 1, lim = EOF)`	— считывает и пропускает символы до тех пор, пока не будет прочитано `num` символов или не встретится разделитель, заданный параметром `lim`. Возвращает ссылку на текущий поток;
`peek()`	— возвращает следующий символ без удаления его из потока или `EOF`, если достигнут конец файла;

Методы обмена с потоками

`putback(c)`	— помещает в поток символ c, который становится текущим при извлечении из потока;
`read(buf, num)`	— считывает `num` символов (или все символы до конца файла, если их меньше `num`) в символьный массив `buf` и возвращает ссылку на текущий поток;
`readsome(buf, num)`	— считывает num символов (или все символы до конца файла, если их меньше `num`) в символьный массив `buf` и возвращает количество считанных символов;
`seekg(pos)`	— устанавливает текущую позицию чтения в значение `pos`;
`seekg(offs, org)`	— перемещает текущую позицию чтения на `offs` байтов, считая от одной из трех позиций, определяемых параметром `org`: `ios::beg` (от начала файла), `ios::cur` (от текущей позиции) или `ios::end` (от конца файла);
`tellg()`	— возвращает текущую позицию чтения потока;
`unget()`	— помещает последний прочитанный символ в поток и возвращает ссылку на текущий поток.

Методы обмена с потоками

В классе ostream определены аналогичные функции для неформатированного вывода:

flush()	— записывает содержимое потока вывода на физическое устройство;
put(c)	— выводит в поток символ c и возвращает ссылку на поток;
seekg(pos)	— устанавливает текущую позицию записи в значение pos;
seekg (offs, org)	— перемещает текущую позицию записи на offs байтов, считая от одной из трех позиций, определяемых параметром org: ios::beg (от начала файла), ios::cur (от текущей позиции) или ios::end (от конца файла);
tellg()	— возвращает текущую позицию записи потока;
write(buf, num)	— записывает в поток num символов из массива buf и возвращает ссылку на поток.

Методы обмена с потоками

Пример 1. Программа считывает строки из входного потока в символьный массив.

```
int main(){
   const int N = 20, Len = 100;
   char str[Len][N];
   int i = 0;
   while (cin.getline(str[i], Len, '\n') && i<N){
      // _
      i++;
   }
   return 0;
}
```

Методы обмена с потоками

Пример 2. Программа записывает в файл (файловые потоки рассматриваются в следующем разделе) число с плавающей точкой и строку символов, а затем считывает их из файла и выводит на экран:

```
int main(){
// Запись в файл
   ofstream out("test");
  if(!out){
     cout << "Cannot open file 'test' for writing" << endl;
     return 1;
  }
  double num = 100.45;
  char str[ ] = "This is a test.";
  out.write(reinterpret_cast<char *>(&num), sizeof(double));
  out.write(str, strlen(str));
  out.close();
  // Чтение из файла
  ifstream in("test", ios::in|ios::nocreate);
  if(!in){
     cout << "Cannot open file 'test' for reading" << endl;
     return 1;
  }
```

Методы обмена с потоками

```
    double check_num;
    char check_str[60];
    in.read(reinterpret_cast<char *>(&check_num), sizeof(double));
    in.read(check_str, 60);
    int lstr = in.gcount();  // количество прочитанных символов
    check_str[lstr] = 0;     // занести нуль-символ в конец строки
    cout << check_num << ' ' << check_str << endl;
    in.close();
    return 0;
}
```

Приведение типа `reinterpret_cast<char *>` в вызове функций `write()` и `read()` необходимо в тех случаях, когда параметр не является символьным массивом.

Методы обмена с потоками

Пример 3. В приведенной ниже программе формируется файл test, в который выводится три строки.

```
int main(){
   // Запись в файл
   ofstream out("test");
   if(!out) {
      cout << "Cannot open file 'test' for writing" << endl;
      return 1;
   }
   char *str[ ] = {"This is the first line.",
                   "This is the second line.",
                   "This is the third line."};
   for (int i = 0; i<3; ++i){
      out.write(str[i], strlen(str[i]));
      out.put('\n');
   }
```

Методы обмена с потоками

```
    out.close();
// Чтение из файла
    ifstream in("test", ios::in|ios::nocreate);
    if(!in){
       cout << "Cannot open file 'test' for reading" << endl;
       return 1;
    }
    char check_str[3][60];
    for (i = 0; i<3; ++i){
       in.get(check_str[i], 60);
       in.get();}
// Контрольный вывод
    for (i = 0; i<3; ++i) cout << check_str[i] << endl;
    in.close();
    return 0;
}
```

После выполнения функции get(check_str[i], 60) символ-разделитель строк '\n' остается во входном потоке, поэтому необходим вызов get() для пропуска одного символа. Альтернативным способом является использование вместо функции get функции getline, которая извлекает символ-ограничитель из входного потока.

Методы обмена с потоками

Пример 4. Функции peek() и putback() позволяют упростить управление, когда неизвестен тип вводимой в каждый конкретный момент времени информации. Следующая программа иллюстрирует это. В ней из файла (файловые потоки рассматриваются в следующем разделе) считываются либо строки, либо целые. Строки и целые могут следовать в любом порядке.

```
int main(){
   char ch;
   // Подготовка файла
   ofstream out("test");
   if(!out) {
      cout << "Cannot open file 'test' for writing" << endl;
      return 1;
   }
   char str[80], *p;
   out << 123 << "this is a test" << 23;
   out << "Hello there!" << 99 << "bye" << endl;
   out.close();
   // Чтение файла
   ifstream in("test", ios::in|ios::nocreate);
   if(!in){
      cout << "Cannot open file 'test' for reading" << endl;
      return 1;
```

Методы обмена с потоками

```
    }
    do{
       p = str;
       ch = in.peek(); // выяснение типа следующего символа
       if(isdigit(ch)){
          while(isdigit(*p = in.get())) p++; // считывание целого
          in.putback(*p);        // возврат символа в поток
          *p = '\0';             // заканчиваем строку нулем
          cout << "Number: " << atoi(str);
       }
       else if(isalpha(ch)){     // считывание строки
             while(isalpha(*p = in.get())) p++;
             in.putback(*p);     // возврат символа в поток
             *p = '\0';          // заканчиваем строку нулем
             cout << "String: " << str;
          }
          else in.get();         // пропуск
       cout << endl;
    } while(!in.eof());
    in.close();
    return 0;
}
```

Методы обмена с потоками

Результат работы программы:

```
Number: 123
String: this

String: is

String: a

String: test
Number: 23
String: Hello

String: there

Number: 99
String: bye
```

Методы обмена с потоками

При организации диалогов с пользователем программы при помощью потоков необходимо учитывать буферизацию. Например, при выводе приглашения к вводу мы не можем гарантировать, что оно появится раньше, чем будут считаны данные из входного потока, поскольку приглашение появится на экране только при заполнении буфера вывода:

```
cout « "Введите x":
cin » x:
```

Методы обмена с потоками

Для решения этой проблемы в basic_ios определена функция tie(), которая связывает потоки istream и ostream с помощью вызова вида cin.tie(&cout). После этого вывод очищается (то есть выполняется функция cout.flush()) каждый раз, когда требуется новый символ из потока ввода.

Методы обмена с потоками

Использовать в одной программе потоки и функции библиотеки C, описанные в <cstdio> или <stdio.h>, не рекомендуется. Если это по каким-либо причинам необходимо, то до выполнения первой операции с потоками следует вызвать описанную в ios_base функцию sync_with_stdio(), которая обеспечит использование общих буферов. Вызов sync_with_stdio(false) разъединяет буфера (это может привести к увеличению производительности).