Лабораторная работа №6.
Потоковые классы

Цель работы: Изучить методы организации чтения/записи данных в файл и вывода информации на экран.

Содержание

Понятие потока

В стандартной библиотеке ввода/вывода стандартного Си (заголовочный файл библиотеки - <stdio.h>) имеются внешние переменные-указатели на дескрипторы файлов - стандартных устройств ввода-вывода.

```
extern FILE *stdin, *stdout, *stderr, *stdaux, *stdprn;
```

(стандартный ввод, стандартный вывод, регистрация ошибок, дополнительное устройство, устройство печати).

Эти файлы открываются библиотекой автоматически перед выполнением функции main и по умолчанию назначаются на терминал (stdin - клавиатура, stdout, stderr - экран), последовательный порт (stdaux) и принтер (stdprn). stdin и stdout могут быть переназначены в командой строке запуска программы на любые другие файлы. Потоковые классы представляют объектно-ориентированный вариант функций ANSI-C. Поток данных между источником и приемником при этом обладает следующими свойствами.

- Источник или приемник данных определяется объектом потокового класса.
- Потоки используются для ввода-вывода высокого уровня.
- Потоковые классы делятся на три группы (шаблонов классов):
  - basic_istream, basic_ostream - общие потоковые классы, которые могут быть связаны с любым буферным объектом;
  - basic_ifstream, basic_iostream - потоковые классы для считывания и записи файлов;

- basic_istringstream, basic_ostringstream - потоковые классы для объектов-строк.

- Каждый потоковый класс поддерживает буферный объект, который предоставляет память для передаваемых данных, а также важнейшие функции ввода/вывода низкого уровня для их обработки.
- Базовым шаблоном классов basic_ios (для потоковых классов) и basic_streambuf (для буферных классов) передаются по два параметра шаблона:
  - первый параметр (charT) определяет символьный тип;
  - второй параметр (traits) - объект типа ios_traits (шаблон класса), в котором заданы тип и функции, специфичные для используемого символьного типа;
  - для типов char и wchar_t образованы соответствующие объекты типа ios_traits и потоковые классы.

Пример шаблона потокового класса.

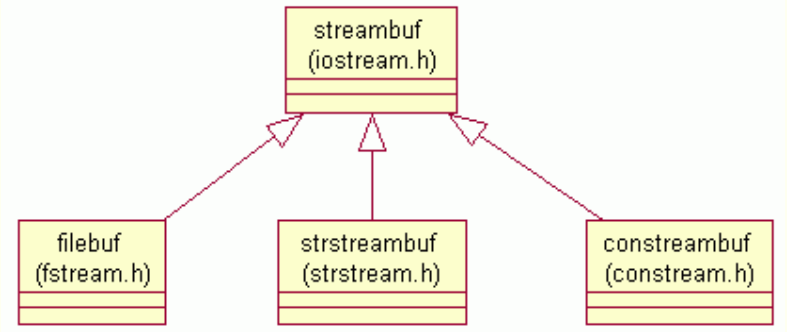
```
template<class charT, class traits = ios_traits<charT> >
class basic_istream : virtual public basic_ios<charT, traits>;
```

[В начало]

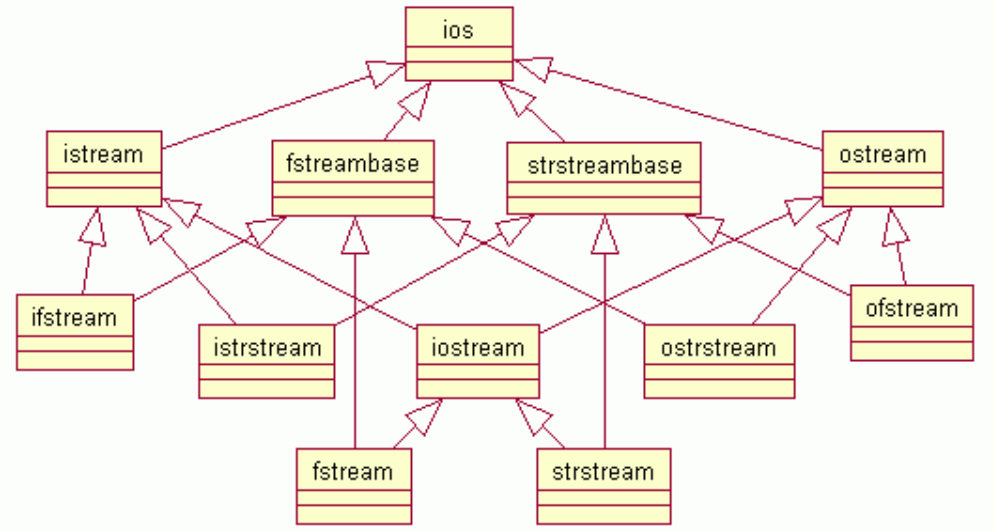
Потоковые классы в C++

Библиотека потоковых классов C++ построена на основе двух базовых классов: ios и streambuf .

Класс streambuf обеспечивает организацию и взаимосвязь буферов ввода-вывода, размещаемых в памяти, с физическими устройствами ввода-вывода. Методы и данные класса streambuf программист явно обычно не использует. Этот класс нужен другим классам библиотеки ввода-вывода. Он доступен и программисту для создания новых классов на основе уже существующих.



Класс ios содержит средства для форматированного ввода-вывода и проверки ошибок.



Стандартные потоки (istream, ostream, iostream) служат для работы с терминалом. Строковые потоки (istrstream, ostrstream, strstream) служат для ввода-вывода из строковых буферов, размещенных в памяти. Файловые потоки (ifstream, ofstream, fstream) служат для работы с файлами.

- ios базовый потоковый класс
- streambuf буферизация потоков
- istream потоки ввода
- ostream потоки вывода
- iostream двунаправленные потоки
- iostream_withassign поток с переопределенной операцией присваивания
- istrstream строковые потоки ввода
- ostrstream строковые потоки вывода
- strstream двунаправленные строковые потоки
- ifstream файловые потоки ввода
- ofstream файловые потоки вывода
- fstream двунаправленные файловые потоки

Следующие объекты-потоки заранее определены и открыты в программе перед вызовом функции main:

```
extern istream cin;    // Стандартный поток ввода с клавиатуры
extern ostream cout;   // Стандартный поток вывода на экран
extern ostream cerr;   // Стандартный поток вывода сообщений об ошибках (экран)
extern ostream clog;   // Стандартный буферизованный поток вывода
                       // сообщений об ошибках (экран)
```

Потоковые классы, их методы и данные становятся доступными в программе, если в неё включен нужный заголовочный файл.

- iostream.h - для ios, ostream, istream .
- strstream.h - для strstream, istrstream, ostrstream .
- fstream.h - для fstream, ifstream, ofstream.

[В начало]

Базовые потоки ввода-вывода

.

Для ввода с потока используются объекты класса istream, для вывода в поток - объекты класса ostream .

В классе istream определены следующие функции:

- istream &get(char *buffer, int size, char delimiter = '\n'); Эта функция извлекает символы из istream и копирует их в буфер. Операция прекращается при достижении конца файла, либо когда скопированы size символов, либо при обнаружении указанного разделителя. Сам разделитель не копируется и остается в streambuf. Последовательность прочитанных символов всегда завершается нулевым символом.
- istream &read(char *buffer, int size); Не поддерживает разделителей, и считанные в буфер символы не завершаются нулевым символом.
- istream &getline(char *buffer, int size, char delimiter ='\n'); Разделитель извлекается из потока, но в буфер не заносится. Это основная функция для извлечения строк из потока. Считанные символы завершаются нулевым символом.
- istream &get(streambuf &s, char delimiter = '\n'); Копирует данные из istream в streambuf до тех пор, пока не обнаружит конец файла или символ-разделитель, который не извлекается из istream. В s нулевой символ не записывается.
- istream get(char &C); Читает символ из istream в C. В случае ошибки C принимает значение 0XFF.
- int get(void); Извлекает из istream очередной символ. При обнаружении конца файла возвращает EOF.
- int peek(void); Возвращает очередной символ из istream, не извлекая его из istream.
- int gcount(void); Возвращает количество символов, считанных во время последней операции неформатированного ввода.
- istream &putback(char C); Если в области get объекта streambuf есть свободное пространство, то туда помещается символ C.
- istream &ignore(int count = 1, int target = EOF); Извлекает символ из istream, пока не произойдет следующее:
  - функция не извлечет count символов;
  - не будет обнаружен символ target;
  - не будет достигнуто конца файла.

В классе ostream определены следующие функции:

- ostream &put(char C); Помещает в ostream символ C.
- ostream &write(const char *buffer, int size); Записывает в ostream содержимое буфера. Символы копируются до тех пор, пока не возникнет ошибка или не будет скопировано size символов. Буфер записывается без форматирования. Обработка нулевых символов ничем не отличается от обработки других. Данная функция осуществляет передачу необработанных данных (бинарных или текстовых) в ostream.
- ostream &flush(void); Сбрасывает буфер streambuf.

Для прямого доступа используются следующие функции установки позиции чтения - записи. При чтении

- istream &seekg(long p); Устанавливает указатель потока get (не путать с функцией) со смещением p от начала потока.

- istream &seekg(long p, seek_dir point); Указывается начальная точка перемещения. enum seek_dir { beg, curr, end } Положительное значение p перемещает указатель get вперед (к концу потока), отрицательное значение p - назад (к началу потока).
- long tellg(void); Возвращает текущее положение указателя get.

При записи

- ostream &seekp(long p); Перемещает указатель put в streambuf на позицию p от начала буфера streambuf.
- ostream &seekp(long p, seek_dir point); Указывает точка отсчета.
- long tellp(void); Возвращает текущее положение указателя put.

Помимо этих функций в классе istream перегружена операция >>, а в классе ostream - <<. Операции << и >> имеют два операнда. Левым операндом является объект класса istream (ostream), а правым - данное, тип которого задан в языке.

Для того чтобы использовать операции << и >> для всех стандартных типов данных используется соответствующее число перегруженных функций operator << и operator >>. При выполнении операций ввода-вывода в зависимости от типа правого операнда вызывается та или иная перегруженная функция operator .

Поддерживаются следующие типы данных: целые, вещественные, строки (char *). Для вывода - void * (все указатели, отличные от char *, автоматически переводятся к void *). Перегрузка операции >> и << не изменяет их приоритета.

Функции operator << и operator >> возвращают ссылку на тот потоковый объект, который указан слева от знака операции. Таким образом, можно формировать "цепочки" операций.

```
cout << a << b << c;
cin >> i >> j >> k;
```

При вводе-выводе можно выполнять форматирование данных.

Чтобы использовать операции >> и << с данными пользовательских типов, определяемых пользователем, необходимо расширить действие этих операций, введя новые операции-функции. Первым параметром операции-функции должна быть ссылка на объект потокового типа, вторым - ссылка или объект пользовательского типа.

```
// объявление класса Vector
class Vector {
  public:
  Vector(unsigned);

  ~Vector();

  protected:
    unsigned v; // размер вектора
    float *pv; // указатель на 1-й элемент

  friend ostream &operator <<(ostream &, const Vector &);
  friend istream &operator >>(istream &, Vector &);
};

// определение некоторых компонентных функций
```

```
   ostream &operator <<(ostream &strm, const Vector &V) {
   strm << endl;
   for(unsigned i = 0; i < V.v; i++) {
     strm.width(5);
     strm << V.pv[i] << endl;
   }
   return strm;
}

 istream &operator <<(istream &strm, Vector &V) {
   for (unsigned i = 0; i < V.v; i++) {
     strm >> V.pv[i];
   }
   return strm;
}
```

[В начало]

Форматирование

Непосредственное применение операций ввода << и вывода >> к стандартным потокам cout, cin, cerr, clog для данных базовых типов приводит к использованию "умалчиваемых" форматов внешнего представления пересылаемых значений.

Форматы представления выводимой информации и правила восприятия данных при вводе могут быть изменены программистом с помощью флагов форматирования. Эти флаги унаследованы всеми потоками из базового класса ios. Флаги форматирования реализованы в виде отдельных фиксированных битов и хранятся в protected компоненте класса long x_flags. Для доступа к ним имеются соответствующие public функции.

Кроме флагов форматирования используются следующие protected компонентные данные класса ios :

int x_width - минимальная ширина поля вывода.

int x_precision - точность представления вещественных чисел (количество цифр дробной части) при выводе;

int x_fill - символ-заполнитель при выводе, пробел - по умолчанию

. Для получения (установки) значений этих полей используются следующие компонентные функции:

```
int width(void);
  int width(int);
  int precision(void);
  int precision(int);
  char fill(void);
  char fill(char);
```

[В начало]

Манипуляторы

Несмотря на гибкость и большие возможности управления форматами с помощью компонентных функций класса ios, их применение достаточно громоздко. Более простой способ изменения параметров и флагов форматирования обеспечивают манипуляторы.

Манипуляторами называются специальные функции, позволяющие модифицировать работу потока. Особенность манипуляторов состоит в том, что их можно использовать в качестве правого операнда операции >> или << . В качестве левого операнда, как обычно, используется поток (ссылка на поток), и именно на этот поток воздействует манипулятор. Манипуляторы указывают, например, ширину поля, точность при вычислении с плавающей точкой и т.п.

Для обеспечения работы с манипуляторами в классах istream и ostream имеются следующие перегруженные функции operator.

```
istream &operator >>(istream &(*_f)(istream &));
ostream &operator <<(ostream &(*_f)(ostream &));
```

При использовании манипуляторов следует включить заголовочный файл <iomanip.h>, в котором определены встроенные манипуляторы. Манипуляторы действуют на ввод-вывод в поток до внесения новых изменений.

dec	Устанавливает 10-тичную систему счисления. Воздействует на int и long. Потоки используют основание 10 по умолчанию.
hex	Устанавливает 16-ричную систему счисления.
oct	Устанавливает 8-ричную систему счисления.
ws	Выбирает из потока ввода символы пропуска. Поток будет читаться до появления символа, отличного от пропуска, или до возникновения ошибки потока.
endl	Вставляет в поток вывода символ новой строки и затем сбрасывает поток.
ends	Вставляет '\0' в поток вывода.
flush	Сбрасывает поток вывода.

setbase() устанавливает основание счисления к любому из четырех значений:

0	Основание по умолчанию. При выводе 10-тичное, при вводе - числа, начинающиеся с '0', считаются 8-ричными, начинающиеся с '0x', - 16-ричными. Во всех остальных случаях основание считается 10-тичным.
8	Для ввода-вывода используется основание 8.
10	Для ввода-вывода используется основание 10.
16	Для ввода-вывода используется основание 16.

Другие значения игнорируются. Библиотека iostream не поддерживает произвольных оснований, подобных 3, 12 и т.д. Если нужно представить значения по основанию, отличному от 8, 10 или 16, то соответствующее преобразование нужно выполнить явно.

resetiosflags(long) очищает один или более флагов форматирования в ios::x_flags

setiosflags(long) устанавливает один или более флагов форматирования в ios::x_flags

setfill(int) устанавливает символ - заполнитель.

Символ-заполнитель используется для заполнения поля тогда, когда ширина поля больше ширины выведенного значения. Заполнение не будет происходить, если пользователь не указал минимальной ширины поля с помощью манипулятора

setw(int) или функции ios::width(int). По умолчанию символом-заполнителем является пробел. Заполнение будет происходить справа, слева, или как-то еще, в зависимости от значения битов ios::adjustfield, установленных обращением к ios::setf(long)

setprecision() устанавливает число цифр после 10-тичной точки в числах с плавающей точкой. Этот манипулятор действует только на потоке вывода setw(int width) Устанавливает ширину следующей вставляемой в поток вывода переменной. Если значение следующей переменной требует для записи меньше места, чем указано, то будет осуществляться заполнение символом-заполнителем, установленным манипулятором setfill(int). Ширина автоматически сбрасывается в 0 после каждой вставки в поток.

[В начало]

Определение пользовательских манипуляторов

Порядок создания пользовательского манипулятора с параметрами, например для вывода, следующий:

1. Определить класс (MyManip) с полями: параметры манипулятора, указатель на функцию типа ostream &(*f) (ostream &, <параметры манипулятора>);
2. Определить конструктор этого класса (MyManip) с инициализацией полей.
3. Определить, в этом классе дружественную функцию - operator <<. Эта функция в качестве правого аргумента принимает объект класса MyManip, левого аргумента (операнда) поток ostream и возвращает поток ostream как результат выполнения функции *f. Например,

```
typedef ostream &(*PTF) (ostream &, int, int, char);
 class MyManip
 {
 public:
 // конструктор
 MyManip(PTF F, int W, int N, char FILL)
 : f(F),
 w(W),
 n(N),
 fill(FILL)
 {}

 protected:
 int w;
 int n;
 char fill;
 PTF f;

 friend ostream &operator <<(ostream &, MyManip);
 };

 ostream &operator <<(ostream &out, MyManip my)
 {
 return my.f(out, my.w, my.n, my.fill);
```

```
}
```

4. Определить функцию типа *f(fmanip), принимающую поток и параметры манипулятора и возвращающую поток. Эта функция собственно и выполняет форматирование. Например,

```
ostream &fmanip(ostream &s, int w, int n, char fill)
 {
 s.width(w);
 s.flags(ios::fixed);
 s.precision(n);
 s.fill(fill);
 return s;
 }
```

5. Определить собственно манипулятор (wp) как функцию, принимающую параметры манипулятора и возвращающую объект MyManip, поле f которого содержит указатель на функцию fmanip. Например,

```
MyManip wp(int w, int n, char fill)
 {
 return MyManip(fmanip, w, n, fill);
 }
```

Для создания пользовательских манипуляторов с параметрами можно использовать макросы, которые содержатся в файле <iomanip.h>:

```
OMANIP(int)
 IMANIP(int)
 IOMANIP(int)
```

[В начало]

Состояние потока

Каждый поток имеет связанное с ним состояние. Состояния потока описываются в классе ios в виде перечисления enum.

```
public:
 enum io_state {
 goodbit, // нет ошибки 0X00
 eofbit, // конец файла 0X01
 failbit, // последняя операция не выполнилась 0X02
 badbit, // попытка использования недопустимой операции 0X04
 hardfail // фатальная ошибка 0X08
 };
```

Флаги, определяющие результат последней операции объектом ios, содержатся в переменной state. Получить значение этой переменной можно с помощью функции int rdstate().

Кроме того, проверить состояние потока можно следующими функциями:

```
int bad(void); // 1, badbit или hardfail
 int eof(void); // 1, если eofbit
 int fail(void); // 1, если failbit, badbit или hardfail
 int good(void); // 1, если goodbit
```

Если операция >> используется для новых типов данных, то при её перегрузке необходимо предусмотреть соответствующие проверки.

[В начало]

Файловый ввод-вывод

Потоки для работы с файлами создаются как объекты следующих классов:
ofstream - запись в файл;
ifstream - чтение из файла;
fstream - чтение/запись.
Для создания потоков имеются следующие конструкторы:

- fstream(); создает поток, не присоединяя его ни к какому файлу.
- fstream(const char *name, int mode, int p = filebuf::openprot); создает поток, присоединяет его к файлу с именем name, предварительно открыв файл, устанавливает для него режим mode и уровень защиты p. Если файл не существует, то он создается. Для mode = ios::out, если файл существует, то его размер будет усечен до нуля.

Флаги режима определены в классе ios и имеют следующие значения:
in - для чтения
out - для записи
ate - индекс потока помещен в конец файла. Чтение больше не допустимо, выводные данные записываются в конец файла;
app - поток открыт для добавления данных в конец. Независимо от seekp() данные будут записываться в конец; trunc - усечение существующего потока до нуля;
nocreate - команда открытия потока будет завершена неудачно, если файл не существует;
noreplace - команда открытия потока будет завершена неудачно, если файл существует;
binary - поток открывается для двоичного обмена.
Если при создании потока он не присоединен к файлу, то присоединить существующий поток к файлу можно функцие

```
void open(const char *name, int mode, int p = filebuf::openprot);
```

Функция void fstreambase::close(); сбрасывает буфер потока, отсоединяет поток от файла и закрывает файл. Эту функцию необходимо явно вызвать при изменении режима работы с потоком. Автоматически она вызывается только при завершении программы.
Таким образом, создать поток и связать его с файлом можно тремя способами:

1. Создается объект filebuf

```
filebuf fbuf;
```

Объект filebuf связывается с устройством (файлом)

```
fbuf.open("имя", ios::in);
```

Создается поток и связывается с filebuf

```
istream stream(&fbuf);
```

2. Создается объект fstream (ifstream, ofstream)

```
fstream stream;
```

Открывается файл, который связывается через filebuf с потоком

```
stream.open("имя", ios::in);
```

3. Создается объект fstream, одновременно открывается файл, который связывается с потоком

```
fstream stream("имя", ios::in);
```

[В начало]

Индивидуальное задание

Пересмотреть проект, разработанный при выполнении лабораторной работы №1 (№8). Разработать и программно реализовать методы объектов для вывода данных на экран, чтение из файла и запись в файл.

[В начало]