Лекция ПКШ2018\_Л6

# Работа с потоками в С++

## Потоковые классы

## Понятие потока

Поток – это последовательность данных, передаваемых от источника к приемнику.   
Потоки в С++ реализованы в виде шаблонов потоковых классов.   
Каждый потоковый класс поддерживает буферный объект, который предоставляет память для передаваемых данных, а также важнейшие функции ввода/вывода низкого уровня для их обработки.

Потоки используются для высокоуровневого ввода-вывода, то есть программа инициализирует выполнение операций ввода-вывода (указывает источник или приемник данных, выполняет обмен данными между ОП и буфером потока), а непосредственный обмен данными между буфером и источником или приёмником выполняют функции операционной системы.  
Данные могут быть представлены символами одного из двух типов:   
*char* и *wchar\_t*.

Источник или приемник данных определяется объектом потокового класса.  
 Потоковые классы (шаблоны классов) делятся на три группы:   
- общие потоковые классы, которые могут быть связаны с любым объектом, использующим буфер для обмена данными;   
- потоковые классы для считывания и записи файлов;   
- потоковые классы для объектов-строк, размещённых в ОП.

## 2. Структура стандартной библиотеки ввода-вывода.

В языке программирования С++ стандартная библиотека ввода-вывода организована как иерархия шаблонов классов. Библиотека 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. Однако он может быть 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 ОП. Они часто используются для преобразования типов с помощью перегруженных операций ‘>>’ и ‘<<’.  
Файловые потоки (*ifstream, ofstream, fstream* ) служат для работы с файлами.

Потоковые классы, их методы и данные становятся доступными в программе, если в неё включен нужный заголовочный файл.   
Ниже перечислены заголовочные файлы, которые содержат интерфейсы классов из стандартной библиотеки ввода-вывода:

<ios> ios\_base, ios

<istream> istream

<ostream> ostream

<iostream> iostream, cin, cout, cerr

<fstream> ifstream, fstream, ofstream, filebuf

<sstream> istringstream, stringstream, ostringstream, stringbuf

<streambuf> streambuf

## 3. Базовые классы ввода-вывода istream и ostrem

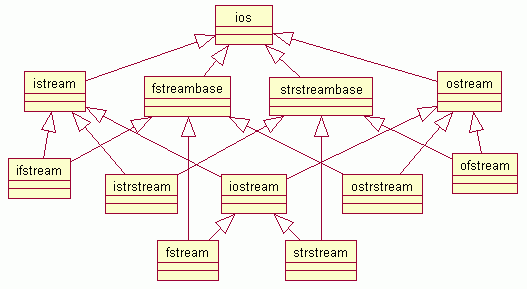


Рисунок 6.1. Диаграмма классов потокового ввода-вывода.

Для ввода из 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, или когда скопированы size символов, или 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 &ch);  
  Читает символ из istream в ch. В случае ошибки ch 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 ch);  
  Если в области get объекта streambuf есть свободное пространство, то туда помещается символ С.
* 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 ch);  
  Помещает в ostream символ ch.
* 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 sizeсимволов. 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 р от начала потока.
* istream &seekg(long p, seek\_dir point);  
  Указывается начальная точка перемещения.  
  enum seek\_dir { beg, curr, end };  
  Положительное значение р перемещает указатель get вперед (к концу потока), отрицательное значение р - 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 р 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 . Перегрузка операции >> и << не изменяет их приоритета.

Чтобы использовать операции >> и << с данными типов, определяемых пользователем, необходимо перегрузить для них эти операции. Первым параметром операции-функции должна быть ссылка на объект потокового типа, вторым - ссылка или объект пользовательского типа

При вводе-выводе можно выполнять форматирование данных.

## 4. 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.

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);

## 5. 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.

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. |

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 функцииios::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). Он 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.

## 5. 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 0Х00

eofbit, // конец файла 0Х01

failbit, // последняя операция не выполнилась 0Х02

badbit, // попытка использования недопустимой операции 0Х04

hardfail // фатальная ошибка 0Х08

};

Флаги, определяющие результат последней операции с 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.

## 6. 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. Для p мы будем использовать значение по умолчанию.  
  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 - указатель 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 функцией

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

Для проверки правильности открытия файла в fstream добавлена (по сравнению с базовым классом iostream) функция is\_open().

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 fstream (ifstream, ofstream)

fstream stream;

Открывается файл, который связывается через filebuf с потоком

stream.open("имя", ios::in);

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

fstream stream("имя", ios::in);

3. Создается объект   
 filebuf fbuf;

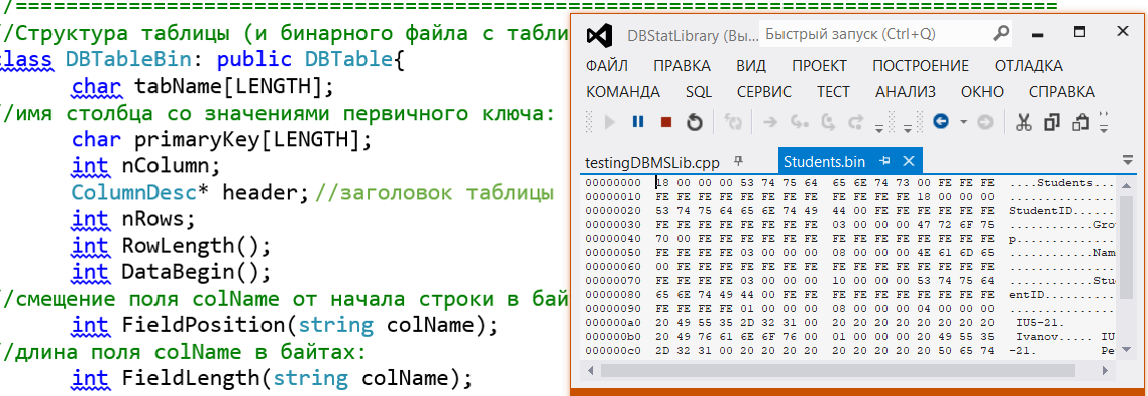
Объект filebuf связывается с устройством (файлом)

fbuf.open("имя", ios::in);

Создается поток и связывается с filebuf

istream stream(&fbuf);

# Бинарные файлы прямого доступа



void ResizeData(int deltaRows);

char\*\* data;  
//число строк в data равно nRows+DELTA

int maxRows;

// В файл не записывается

char fileName[80];//Используется в методах для

//обращения к файлу

void CreateTableMaket(Strip \*&strips,  
 int &nStrips,int screenWidth);

public:

DBTableBin(){}

DBTableBin(string tableName);

DBTableBin(string tableName,Header hdr,string primKey);

~DBTableBin(){}

Header GetHeader();

vector<int> IndexOfRecord(void\* keyPtr,string keyName);

//по значению ключа определяет

//порядковый номер (индекс) строки (записи) в data.

// keyPtr-значение ключа (указатель на string, int или DBDate)

// keyName-имя ключевого столбца (первичный или вторичный ключ)

//в таблице может быть несколько записей с одинаковым значением вторичнего ключа.

int FieldPosInFile(int rowIndex,string columnName);

. . .

<интерфейс DBTable>

. . .

};

//============================================================================

//------Запись данных таблицы в бинарный файл tabName+".bin"--------------

void DBTableBin::WriteDBTable(string fileName){

//извлечение имени таблицы из fileName (путь к файлу из текущей папки)

string tableName=GetTabNameFromPath(fileName);

if(tableName!=tabName){

cout<<"Имя таблицы в fileName "<<tableName<<

" не соответствует имени таблицы "<<tabName<<  
 ",указанному в DBTableBin\n";

system("pause");

return;

}

ofstream fout;//открытие файлового потока на вывод

fout.open(fileName,ios::binary|ios::out);

if (!fout.is\_open()){

cout<<"Ошибка открытия файла "<<fileName<<endl;

system("pause");

return;

}

int len=LENGTH;

fout.write((char\*)&len,sizeof(int));

fout.write(tabName,LENGTH);//запись имени таблицы

fout.write((char\*)&len,sizeof(int));

fout.write(primaryKey,LENGTH);//запись имени столбца с первичным ключом

fout.write((char\*)&nColumn,sizeof(int));//число столбцов в таблице

fout.write((char\*)header,nColumn\*sizeof(ColumnDesc));//запись заголовка . . .  
 . . .

}

}

//--------------------------------------------------------------

void DBTableBin:: ReadDBTable(string fileName){//!!! fileName - путь из текущей папки.

//При чтении из двоичного файла в DBTableBin преобразование типов не требуется.

//Извлечение имени таблицы из fileName (путь к файлу из текущей папки)

string tableName=GetTabNameFromPath(fileName);

SetTableName(tableName);

ifstream fin;//открытие файлового потока на ввод

fin.open(fileName,ios::binary|ios::in);

if (!fin.is\_open()) {

cout<<"Ошибка открытия файла "<<fileName<<endl;

system("pause");

return;

}

fin.seekg(sizeof(int));

char tabNm[LENGTH];

fin.read(tabNm,LENGTH);

if(tableName!=tabNm){

cout<<"Имя таблицы в fileName "<<tableName<<

" не соответствует имени, "<<tabNm <<  
 ", указанному в теле файла\n";

system("pause");

return;

}

fin.seekg(sizeof(int),SEEK\_CUR);

fin.read(primaryKey,LENGTH);//'\0' помещается в файл при записи

// считывание заголовка в ColumnDesc\* header:

nColumn=0;

fin.read((char\*)&nColumn,sizeof(int));

header=new ColumnDesc[nColumn];

fin.read((char\*)header,nColumn\*sizeof(ColumnDesc));

//считывание данных из файла

. . .   
. . .  
}