|  |
| --- |
| **Page 1** |

**лекция 13**

**Тема лекции:** Потоки. Форматирования вывода. Использование манипуляторов потока. Функции-члены классов IOS, ISTREAM, STREAM. Флаги форматирования объектов класса IOS. Строчной-ориентированное ввода / вывода. Символические константы режима открытия потока.

Перегрузка оператора вывода в поток

***Потоки ввода-вывода***

В языке С ++ как и в С, отсутствуют встроенные операторы ввода-вывода. В программах на С и на С ++ ввода и вывода обычно осуществляются при помощью функций ввода-вывода, содержащихся в библиотеках.

Стандарт языка С определяет набор функций ввода-вывода *(stdio.h),* которые обязательно должны входить в состав стандартной библиотеки С. Прототипы стандартных функций ввода-вывода содержатся в заголовочный файл *stdio.h.* Хотя все стандартные функции С также доступны в С ++, при программировании на С ++ во многих случаях лучше использовать

определенная стандартная библиотека потокового ввода / вывода *iostream,* что

содержит большой набор классов и функций для буферизированная и небуферизованого

ввода / вывода данных как для файлов, так и для устройств.

Что же такое, по сути, потоки ввода / вывода? В полном соответствии с

логикой С ++ поток определяется как некоторый абстрактный тип данных,

представляет какую-то последовательность элементов данных, направленную от источника к

потребителя. Количество элементов в потоке называют его длиной,

порядковый номер доступного в некоторый момент элемента называют текущей

позицией. Каждый поток имеет один из трех возможных режимов доступа: только

для чтения, только для записи, чтения и записи. библиотека *iostream*

содержит две параллельные семьи классов. Первая ведет свое начало от класса

*streambuf,* а вторая - от *ios.* Класс *streambuf* и два класса, производных от него -

*filebuf*

и

*strstreambuf*

обеспечивают

буферизацию

потокового

|  |
| --- |
| **Page 2** |

ввода / вывода и абстрагируют обращения к физическим устройствам

ввода / вывода. Класс *ios* дает начало семье классов, предназначенных для

реализации форматированного и неформатированного ввода / вывода на высоком

уровне с возможностью контроля и изменения состояния потока. *Ios* содержит член *ios :: bp* -

указатель на *streambuf,* с помощью которого и общается с устройствами

ввода / вывода. Описание класса *ios* расположен в файле *iostream.h,*

включение которого в программу, использующую ввода / вывода,

обязательна.

Среди классов, производных от *ios* назовем следующие:

*istream* - введение со стандартного устройства ввода. содержит

перегружен оператор форматированного ввода ">>" и ряд функций

неформатированного введения;

*ostream* - вывод на стандартное устройство вывода. содержит

перегружен оператор форматированного вывода "<<" и ряд функций

неформатированного вывода;

*iostream* - смесь классов *istream* и *ostream;*

*iostream\_withassign*

- *iostream*

с перегруженным оператором

присвоения. Его объектами являются стандартные объекты ввода / вывода. В С ++

это объекты *cin,* *cout,* *cerr,* *clog.*

В заголовочный файл *IOSTREAM.H* содержатся такие объявления

одного потока ввода и трех потоков вывода:

*•* in - объект потока ввода;

*•* out - объект потока вывода;

*•* err - объект потока вывода ошибок;

*•* log - объект буферизированная потока вывода ошибок.

Поточные классы выполняют отформатировано ввода-вывода с

встроенной поддержкой обработки ошибок. Потоки ввода / вывода

поддерживают базовые типы данных, подобные *char,* *short,* *int,* *long,* *float,* *double,* *long*

*double,* *char \** и *void \** (значение - адрес).

*Файловое ввода-вывода.* Рассмотрим подробнее использования класса

*ifstream* и *ofstream:*

|  |
| --- |
| **Page 3** |

*ifstream* - производный от *istream* класс, связывает файл с прикладной

программой для выполнения только введение.

*ofstream* - производный от *ostream* класс, связывает файл с прикладной

программой для выполнения только вывод.

Описания этих классов содержатся в заголовочный файл *fstream.h.* он

автоматически подключает к программе и файл описания классов потоков *iostream.h.*

Классы *ifstream* и *ofstream* являются производными от *istream* и *ostream* и наследуют

операции ***"<<"*** и ***">>".***

Для открытия файла для записи используется запись:

ofstream foo ( "имя файла");

Для записи в файл

foo << "Hello";

Чтение информации из файла происходит аналогично записи:

ifstream foo ( "имя\_файла")

char buffer [100];

foo >> buffer; // читает текст с файла

Перед использованием классов должен быть включен заголовочный файл:

#include <fstream.h>

Когда разгружается переменная потока, поток, соответствующий файла,

закрывается. Его можно закрыть до того, как поток будет исчерпан,

используя для этого функцию-член *close ():*

foo.close ();

Обычно это делают, если хотят использовать тот же поток для

обслуживание нескольких файлов. В этом случае нужно закрыть первый

файл и открыть другой файл с помощью функции-члену *open ():*

foo.open ( "file. txt")

Для того, чтобы узнать, осталось ли еще что-нибудь в файле, нужно

воспользоваться функцией-членом *eof (),* которая возвращает значение *"истина"* при

выявлении конца файла.

Советы для чтения и записи файлов:

• лучшее не писать и не читать одновременно один и тот же файл. хотя

это вполне законно, но можно запутаться в том, какая часть файла читается,

|  |
| --- |
| **Page 4** |

а какая пишется.

• При записи чисел в файл с помощью "<<" нужно помнить,

что числа хранятся в символьном формате (как текст), но без пробелов между

символами.

• Существует символ конца файла, сообщает оператору ">>" о том, что

файл закончился.

• Строки читаются, пока не будет встречен разделитель *(eof* или \ *n).*

Если последняя строка в файле не заканчивается символом *\ n,* то признак конца

файла достигается сразу же после завершения чтения последней строки.

• При чтении чисел из потока символ конца файла по последним числом

отсутствует. Поэтому может случиться так, что программист пытается прочитать на

одно число больше, чем их есть на самом деле в файле или поместить символ конца

файла в число. В этом случае ">>" ничего не будет делать, а, значит, число

не изменится. Следовательно, при чтении и записи в файл, необходима проверка

на конец файла.

***Использование манипуляторов***

Существуют некоторые атрибуты, указывая которые можно изменить способ чтения и

записи в потоке. Эти атрибуты называются манипуляторами потока. Сами они не

позволяют читать и записывать данные - они лишь влияют на средство

интерпретации данных при чтении и записи. некоторые манипуляторы

используются с параметрами.

Манипуляторы ввода / вывода определяются в заголовочный файлах

IOMANIP.H и IOSTREAM.H:

*endl* - Начать новую строку; вывести буфер потока.

*ends* - Вставить нулевой завершающий символ в строке.

*flush* - Выполнить вывода в поток.

*lock (ios & ir)* - Заблокировать дескриптор файла для посылки *ir* на поток

ввода / вывода.

*resetiosflags (long f)* - Очистить биты форматирования.

|  |
| --- |
| **Page 5** |

*setiosflags (long f)* - установить биты форматирования.

*setfill (int c)* - использовать символ «с» для заполнения при

выравнивании.

*setprecision (int n)* - установить точность вывода значений с плавающей

запятой равной *n*

*setw (int n)* - установить ширину поля *n.*

*unlock (ios & ir)* - Разблокировать дескриптор файла для посылки *ir* потоков

ввода / вывода.

*ws* - Вытянуть пустые символы.

*Skipws -* Игнорировать пустое пространство при вводе.

*Left -* "Прижать" вывода к левой стороне поля.

*Right -* "Прижать" вывод к правой стороне поля.

*Internal -* "Прижать" вывод после знака или основы системы счета.

*Showbase -* использовать индикатор основы системы счета при

вываживании.

*Showpoint -* Обязательно показывать десятичную точку при выводе

действительного числа.

*Uppercase -* привести к верхнему регистру вывода по основе системы

счета 16.

*Showpos -* Добавить + при выводе положительного *int.*

*Scientific -* использовать форму 1.2345Е2 для вывода действительных

значений.

*Fixed -* использовать форму 123.45 для вывода действительных значений.

*Untbuf -* Флешуваты все потоки после выполнения операции "<<".

*Stdio -* Флешуваты потоки *stdout, stderr* после выполнения операции *"<<".*

Манипуляторы-функции потоков вывода возвращают посылки на объект

*ostream.* Можно написать свои собственные манипуляторы, включив заголовочный

файл IOSTREAM.H и определив функцию типа *& ostream.* К примеру,

определим манипулятор звонка для потока вывода:

ostream & bell (ostream & os)

{

return os << "\ a"; // \ A - управляющий код звонка

|  |
| --- |
| **Page 6** |

}

Эту функцию можно использовать в каскадном операторе вывода в

поток:

cout << bell () << "Ding! ";

Можно вызвать функцию-член потокового объекта для задания

выравнивания. Например, для вывода значения целой переменной *v* с

выравниванием по правому краю в восьми позициям:

cout.width (8);

cout << v << '\ n';

Модификатор *width* влияет только на это выведено значение.

*Символьные константы режима открытия потока*

In

Открытие потока для чтения.

Out

Открытие потока для записи.

Ate

Перемотка до конца файла перед открытием.

App

Открытие в режиме пополнения.

Trunc

Открытие файла с усечением уже существующего содержания.

nocreate

Открытие потерпит неудачу, если файл не существует.

norepleace

Открытие потерпит неудачу, если файл существует.

binary

Открытие в двоичном режиме.

Конструкторы объектов классов *ifstream,* *ofstream,* *fstream* имеют три формы,

перегружаемых и различаются компилятором по числу и типу

параметров. Варианты конструкторов:

• создание уникального объекта, не соединен с файлом: *ifstream ();*

*fstream ();* *fstream ();*

• создание уникального объекта соединяется с открытым файлом,

заданным префиксом *fd:* *ifstream (int fd);* *ofstream (int fd);* *fstream (int fd);*

• создание уникального объекта соединяется с открытым файлом,

заданным префиксом *fd,* а для обмена использует буфер, на начало которого

указывает *char \** размером *int* байт (третий параметр): *ifstream (int fd, char \*, int)*

• создание уникального объекта, открытие файла, имя которого задает *char \**

*fname,* а режим открытия файла - второй параметр, третий параметр задает

защита файла (file protection) по умолчанию принимается значение,

|  |
| --- |
| **Page 7** |

задается константой *filebuf :: openprot.* Открытый файл соединяется с объектом:

ifstream (const char \* fname, int, int = filebuf :: openprot)

Используя нужен конструктор, можно выполнить посимвольно

или блок-ориентированный обмен с файлом. Обмен с файлом всегда

буферизуется и при создании объектов класса *ifstream, ofstream, fstream*

автоматически создается объект класса *filebuf.*

При организации обмена данными с файлом используют функции-члены

класса *filebuf.* Режимом открытия файла управляют символьные константы, описанные

для класса *ios.* Во всех случаях, когда приложение хочет изменить

установлен режим доступа к файлу по умолчанию, необходимо выполнить

создание объектов класса *ifstream, ofstream, fstream* с заданным вторым

аргументом. При этом константы режима могут объединяться с операцией

поразрядного логического *ИЛИ.*

Например, если файл *c: \ tools \ text.tmp* нужно открыть для записи в

режиме пополнения без создания нового файла, то в программе создается

объект класса *ofstream* следующим образом:

ofstream

my\_output ( "c: \ tools \ text.

tmp ", ios :: noreplace, ios :: app)

Если файл успешно открыт, то можно выполнить чтение и запись,

управления указателем потока функциями-членами базовых классов *istream* и

*ostream.* Для определения состояния файлового ввода / вывода используют

функции базового класса *ios.*

***Строчной-ориентированное ввода-вывода***

Введение строк выполняет оператор ">>". К примеру:

char string [M];

cin >> string;

Однако, его использование может вызвать нежелательные посторонние эффекты,

если число введенных символов превышает объем зарезервированных для строки

символов. Для ввода строки с контролем числа принятых символов можно

использовать функцию-член класса *istream get (char \*, int, char).* К примеру,

|  |
| --- |
| **Page 8** |

для приема строки символов, для которого зарезервировано *M* байтов, надо сделать

так:

char string [M];

cin.get (string, M-1, '\ n');

Если введено более чем *M-1* символов, они остаются в буфере

ввода и будут переданы прикладной программе при последующем использовании

оператора ">>". В частности, всегда останется в буфере ввода символ

перевод строки. Чтобы устранить посторонние эффекты, при последующих операциях

введение надо освободить буфер ввода. Возможное решение - переместить

указатель потока наконец, используя функцию-член *seekg ():*

cin.seekg (0, ios :: end)

Вывод строки выполняет оператор "<<". Если необходимо вывести только

часть строки, используется функция-член класса *ostream write ().*

К примеру:

char string [M];

cout.write (string, 10);

выведет 10 первых символов строке *string.*

Пример. Программа принимает с клавиатуры произвольное число строк и записывает

их в файл. Имя файла задается при запуске программы первым параметром

командной строки. Работа программы завершается при получении с клавиатуры

условия *EOF,*

#include <fstream. h>

int main (int argc, char \*\* argv)

{

char ch;

if (argc <2)

{

cerr << "\ a Использование: \ n";

cerr << argv [0] << "имя файла назначь. \ n ";

return 255;

}

// Попытка создать объект ofstream

ofstream output\_to\_file (argv [1]);

if (! output\_to\_file)

|  |
| --- |
| **Page 9** |

{

cerr << "\ a Ошибка открытия" << argv [1];

cerr << "для записи \ n";

return 254;

}

while (! cin.eof ())

{

cin.get (ch);

output\_to\_file.put (ch);

}

output\_to\_file.close ();

return 0;

}

***Перегрузка оператора вывода в поток***

Обычно потоки вывода могут поддерживать только базовые типы

данных. С помощью перегрузки оператора вывода в поток << можно

заставить его выводить объекты классов.

Пример: в программе перегружается оператор вывода в поток.

class TPoint

{

private:

int x, y;

public:

TPoint ()

{

x = y = 0;

}

TPoint (int xx, int yy)

{

x = xx;

y = yy;

}

void PutХ (int xx)

{

|  |
| --- |
| **Page 10** |

x = xx;

}

void PutУ (int yy)

{

y = yy;

}

int GetХ (void)

{

return x;

}

int GetУ (void)

{

return y;

}

friend ostream & operator << (ostream & os, TPoint & p)

};

main ()

{

TPoint p;

cout << p << '\ n';

p.Put (100)

p.Put (200)

cout << p << '\ n';

return 0;

}

ostream & operator << (ostream & os, TPoint & p)

{

os << "x ==" << px << ", y ==" << py;

return os;

}

здесь

за

помощью

дружественной

функции-члена

*operator << ()*

перегружается оператор вывода в поток. Эта функция возвращает значение

типа *ostream &* и в ней объявлены два параметра: *os* типа *ostream &* и посылка *p* на

объект класса *TPoint.* Поскольку функция *operator << ()* является другом класса *TPoint,* в ней

можно сделать непосредственный доступ к данным членов *x* и *y* того объекта, на который

ссылается *p.* Функция возвращает посылку *os.* Запуск программы приведет к

|  |
| --- |
| **Page 11** |

следующих результатов:

x == 0, y == 0

x == 100, y == 200

То есть функция *operator << ()* возвращает посылку на *ostream.* можно

выводить несколько объектов в одном операторе вывода в поток:

cout << p1 << "," << p2 << "," << p3;

Также возможно и перегрузки оператора ввода из потока.