ЛЕКЦИЯ 19 (КСР)
Потоковый ввод-вывод

Потоковые классы

Потоки являются абстрактным понятием, введенным в языке C++ для однообразного описания и организации ввода-вывода. Иными словами, с помощью понятия потока можно как организовать ввод-вывод различных типов данных, так и обеспечить работу с различными внешними устройствами.

Реализация механизма потоков выполнена средствами ООП, а потоковые классы и связанные с ними операции и функции представляют собой часть *стандартной библиотеки классов.*

Поток представляет собой последовательность примитивов ввода-вывода и реализован с помощью шаблонов, где в качестве параметра шаблона задается тип примитива **`charT`**. С помощью оператора **`typedef`** описаны потоки для типов **`char`** и **`wchar_t`**. В рамках настоящей темы будем рассматривать реализацию потоков для типа **`char`** и, следовательно, считать, что поток представляет собой последовательность символов.

Чтение данных из потока называется ***извлечением***, а их запись – ***помещением*** в поток. Для увеличения скорости передачи данных используется **механизм буферизации:** данные помещаются или извлекаются в специальную область памяти – буфер, а физический обмен данными происходит по заполнению/опустошению буфера.

Потоки можно классифицировать по следующим критериям:
по направлению обмена данными – на входные, выходные и двунаправленные;
по виду внешних устройств – на стандартные, файловые и строковые:
стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея;
файловые – для обмена информацией с файлами на внешних носителях;
строковые потоки предназначены для работы с данными, хранящимися в оперативной памяти. Работа со строковыми потоками будет рассмотрена позднее.

Ввод-вывод данных с использованием потоков может быть ***форматированным*** или ***неформатированным***. В первом случае используются перегруженные операторы >> и <<, которые позволяют помещать в поток и извлекать из потока данные любых типв, в том числе и пользовательских. Во втором случае (неформатированный ввод-вывод) никакого преобразования данных не происходит, и работа с потоками во многом напоминает уже описанную ранее работу с файлами.

- **ios** – базовый класс для всех потоков;
- **istream, ostream, iostream** – классы для стандартных потоков;
- **ifstream, ofstream, fstream** – классы для файловых потоков;
- **istringstream, ostringstream, stringstream** – классы для строковых потоков.

Форматированный ввод-вывод

Для форматированного ввода-вывода используются перегруженные операторы >> и <<. Суть работы оператора << состоит в следующем: данные преобразуются в последовательность символов, которая и заносится в выходной файл. Так, целочисленные данные (типы **int**, **long** и т.д.) могут быть преобразованы к одной из систем счисления (десятичной, восьмеричной или 16-ричной).

Числа с дробной частью (**float**, **double** и т.д.) выводятся только в десятичной системе счисления, но могут быть представлены в фиксированной и экспоненциальной форме. Символы (тип **char**) и строки (т.е. данные типа **char *)** выводятся в соответствии с принятой кодировкой (в консольных приложениях – это т.н. кодировка OEM, соответствующая кодовой странице 866).

Для разделения пробелом двух строк можно записать как конструкцию

```
cout << s1 << ' ' << s2;
```

так и

```
cout << s1 << " " << s2;
```

Оператор >>, наоборот, анализирует последовательность данных во входном потоке и в случае правильной интерпретации преобразует данные во внутреннее представление того типа, который указан в качестве второго операнда. При этом разделителями между операндами является, среди всего прочего, и пробел, и, как следствие, возникают вопросы при вводе строк: окончанием строки является не нажатие клавиши Enter, а первый встреченный пробел. Для решения этой проблемы можно использовать неформатированный ввод.

Флаги можно устанавливать с помощью функции **setf()**, а сбрасывать - с помощью функции **unsetf()**.

В составе класса **ios** определены поля:

long x_flags, содержащее набор битовых флагов, каждый из которых отвечает за тот или иной режим ввода-вывода и имеет свое имя, определенное в классе **ios**;

int x_width – минимальная ширина вывода (по умолчанию – 0);
int x_precision – количество цифр в дробной части при выводе с фиксированной точкой или число значащих цифр при выводе в экспоненциальной форме (по умолчанию – 6);
char x_fill – символ-заполнитель поля вывода (по умолчанию - пробел).

Поля **x_flags, x_fill** и **x_precision,** будучи один раз установленными, остаются таковыми до следующего явного изменения. Однако значение поля **x_width** сбрасывается в ноль после каждого нового вывода.

Флаги форматирования

skipws - при вводе пробельные литеры пропускаются;
left - выводимые данные выравниваются по левому краю с дополнением символами-заполнителями по ширине поля;
right - выводимые данные выравниваются по правому краю с дополнением символами-заполнителями по ширине поля (установлен по умолчанию);
internal - знак числа выводится по левому краю, а само число – по правому краю.
Промежуток между знаком и цифрами заполняется символом **x_fill;**
dec - целые числа выводятся по основанию 10 (установлен по умолчанию); устанавливается также манипулятором **dec;**
oct - целые числа выводятся по основанию 8; устанавливается также манипулятором **oct;**
hex - целые числа выводятся по основанию 16; устанавливается также манипулятором **hex** ;
boolalpha - перевод логического 0 и 1 соответственно в false или **true;**
showbase - при выводе целых чисел отображается префикс, указывающий на основание системы счисления;
showpoint - при выводе чисел с плавающей запятой всегда отображается десятичная точка, а хвостовые нули не отбрасываются;
uppercase - шестнадцатеричные цифры от A до F, а также символ экспоненты E отображаются в верхнем регистре;
showpos - при выводе положительных чисел отображается знак плюс;
scientific - числа с плавающей запятой отображаются в научном формате (с экспонентой) ;
fixed - числа с плавающей запятой отображаются в фиксированном формате (без экспоненты);
unitbuf - при каждой операции вывода буфер потока должен очищаться;
stdio - при каждой операции вывода буферы потоков **stdout** и **stderr**

должны очищаться.

Форматирующие методы

Для доступа к полям форматирования могут быть использованы следующие методы класса **ios**.

```
long flags(long);
```

Этот метод устанавливает значение поля **x_flags**, например:

```
cout.flags(ios::fixed|ios::left);
//остальные флаги сброшены
```

```
long setf(long);
```

Этот метод устанавливает флаги, указанные в параметре и не меняет остальные, например:

```
cout.setf(ios::fixed|ios::left);
//остальные флаги не изменены
```

```
long unsetf(long);
```

Этот метод сбрасывает флаги, указанные в параметре и не меняет остальные.

Некоторые из формально независимых флагов не могут быть установлены одновременно. Для работы с такими флагами можно использовать функцию

```
long setf(long, long);
```

которая сбрасывает флаги, заданные во втором параметре и устанавливает только те из них, которые заданы в первом параметре.

Кроме того, можно использовать определенные в классе **ios** константы

```
adjustfield=left|right|internal;
basefield = dec | oct | hex;
floatfield = fixed | scientific;
```

Так, для гарантированного задания значения **hex** и неизменности других флагов можно записать

```
cout.setf(ios::hex, ios::basefield);
```

Функция

```
int width(int);
```

устанавливает значение ширины поля вывода (т.е. поля **x_width**),

а функция

```
int precision (int);
```

– значение поля **x_precision**.

Функция
char fill(char);
задает значение символа заполнения.

Пример : *вывести на экран матрицу 5x5 из данных типа float с точностью 0.001, если известно, что данные не превосходят по модулю 500.*

```
float a[5][5];
// задание матрицы a
cout.setf(ios::fixed,ios::floatfield);
cout.precision(3);
for (int i=0; i<5; i++)
{
 for (int j=0; j<5; j++)
 {
  cout.width(9);
  cout << a[i][j];
 }
 cout << endl;
}
```

Манипуляторы

Работа с манипуляторами во многом похожа на использование форматирующих методов, но зачастую более проста и понятна. Манипуляторы делятся на простые и параметризированные (последние требуют задания в скобках уточняющих значений). Перечислим основные манипуляторы:

dec, oct, hex - смотри описание соответствующих флагов
endl - включает в поток при выводе символ новой строки и сбрасывает буфер
ends - включает в поток при выводе нулевой символ
flush - сбрасывает буфер
setiosflags(long) - устанавливает те флаги, значения которых заполнены в параметре
resetiosflags(long) - сбрасывает те флаги, значения которых заполнены в параметре
setprecision(int) - устанавливает значение поля **x_precision**
setw(int) - устанавливает значение поля **x_width**

setfill(char) - устанавливает значение поля **x_fill**

```
float a[5][5];
// задание матрицы a
cout << resetiosflags(ios::floatfield) <<
setiosflags(ios::fixed);
cout << setprecision(3);
for (int i=0; i<5; i++)
{
 for (int j=0; j<5; j++)
  cout << setw(9) << a[i][j];
 cout << endl;
}
```

Неформатированный потоковый ввод-вывод

В потоковых классах наряду с операциями извлечения из потока и помещения в поток определены методы для неформатированного чтения-записи. Для потока могут быть определены две текущих позиции: одна используется для чтения, другая – для записи в поток.

Перечислим основные функции неформатированного ввода-вывода:

Методы класса istream

get()
извлекает из потока символ и возвращает его код или **EOF**.

getc(ch)
извлекает из потока символ и помещает его в **ch**. Функция возвращает ссылку на текущий поток .

get(buf, num, lim='\n')
считывает **num-1** символов (или пока не встретится символ **lim**) в буфер, адрес которого содержится в параметре **buf** типа **char *,** формирует нуль-терминированную строку, оставляет **lim** в потоке. Функция возвращает ссылку на текущий поток .

getline(buf, num, lim='\n')
то же, но символ **lim** извлекается из потока.

read(buf, num)
считывает **num** символов в буфер, адрес которого содержится в параметре **buf** типа **char *.** Функция возвращает ссылку на текущий поток.

readsome(buf, num)
то же, но возвращает действительное количество прочитанных символов.

ignore(num=1, lim=EOF)

пропускает либо **num** символов из потока, либо символы до тех пор, пока не встретится символ-ограничитель **lim.**

peek()
возвращает текущий символ потока (или **EOF**) без его удаления из потока.

putback(ch)
помещает в поток символ **ch**. Этот символ становится текущим.

tellg()
возвращает текущую позицию чтения потока.

seekg(pos)
задает новое значение **pos** для текущей позиции чтения потока.

seekg(offs, org)
перемещает текущую позицию чтения потока на величину **offs** относительно **org**. Допустимые значения для org: **ios::beg, ios::end, ios::cur.**

Методы класса ostream

put(ch)
помещает в поток символ **ch** .

write(buf, num)
помещает в поток **num** символов из буфера, адрес которого содержится в параметре **buf** типа **char *.** Функция возвращает ссылку на текущий поток .

tellg()

seekg(pos)

seekg(offs, org)
аналогичны функциям класса **istream**, но работают с текущей позицией записи в поток.

flush()
принудительно сбрасывает буфера и выводит их содержимое на физическое устройство.

Особенности работы с файловыми потоками

Создание файлового потока :

```
ifstream(const char *имяфайла, [ int режим =ios::in ])
ofstream(const char * имяфайла,
  [ int режим =ios::out | ios::trunc])
fstream(const char * имяфайла,
[ int режим =ios::out | ios::in ])
```

Эти конструкторы одновременно с созданием потокового объекта

открывают соответствующий файл. Метод **open** с такими же параметрами открывает файл после того, как поток уже определен.

Закрытие файла, связанного с потоком, выполняется либо при уничтожении объекта-потока во время работы деструктора, либо с помощью вызова метода **close**.

Параметр «метод», задаваемый при открытии потока, представляет собой комбинацию из следующих констант – битовых масок, описанных в классе ios:

in - поток открывается для чтения
out - поток открывается для записи
ate - указатель текущей записи устанавливается на конец файла
app - поток открывается для добавления в конец
trunc - удалить существующий файл
nocreate - выдать ошибку, если файл не существует (только при использовании **iostream.h)**
noreplace - выдать ошибку, если файл существует (только при использовании **iostream.h)**
binary - открыть файл в двоичном режиме

Обработка состояния потока

В процессе работы с потоком могут возникнуть различные ситуации. Для этих целей в классе **ios** определено поле **state**, которое представляет собой комбинацию из следующих битовых масок:

eofbit - достигнут конец файла
failbit - ошибка при форматировании
hardfail - неисправность оборудования
badbit - прочая серьезная ошибка

Для обработки состояния потока можно использовать следующие методы:

int rdstate() - возвращает информацию о состоянии потока
int eof() - возвращает ненулевое значение, если установлен флаг **eofbit**
int good() - возвращает ненулевое значение, если не установлены флаги состояния
void clear(int= 0) - устанавливает новое значение состояния потока
int fail() - возвращает ненулевое значение, если последняя операция завершилась неудачно (например, пытались прочитать число, а в потоке была буква) Возможно восстановление
int bad() - возвращает ненулевое значение, если последняя операция

завершилась неудачно и причины серьезные, восстановление не возможно (например, при чтении произошел аппаратный сбой)

Кроме того, для удобства работы с потоком переопределены оператор ! и оператор приведения потока к типу **`void *.`** Последний оператор вызывается каждый раз при сравнении потока с нулевым значением (т.е. с **0**, **NULL** или **false**) и возвращает **NULL**, если функция **`fail()`** возвратила ненулевое значение. Соответственно, переопределенный оператор ! возвращает **false**, если поле **state** равно нулю. Этим фактом воспользуемся далее в примерах, записывая строки типа

```
while (f.getline(S, 101)!= NULL)
```

Пример 20.1. *Файл in1.txt содержит строки длиной не более 100 символов. Требуется посчитать в этом файле число строк, начинающихся с пробела.*

```
#include <fstream>
#include <iostream>
using namespace std;

int main()
{int N = 0;
 char *S = new char[101];
 ifstream f("in1.txt");
 while (f.getline(S,101)!= NULL)
  if (S[0] == ' ') N++;
 cout << N << endl;
 delete [] S;
 return 0;}
```

Пример 20.2. *//Переписать все непустые строки из файла* **`in1.txt`**, *описанного в предыдущем примере, в выходной текстовый файл* **`out1.txt`**

```
int main() {
 char *S = new char[101];
 ifstream f2("in1.txt");
 ofstream f1("out1.txt");
 while (f2.getline(S,101)!= NULL)
  if (S[0] != '\0')
  {
   f1.write(S, strlen(S));
   f1.put('\n');
```

```
   //или иначе  f1<<S<<"\n";
  }
 delete [] S;
 return 0;
}
```

Пример 20.3. *Строки текстового файла in4.txt состоят из слов, разделенных одним или несколькими пробелами. Длина слова не превышает 100 символов. Слова, которые начинаются с цифры, представляют собой десятичную запись натуральных чисел.*
Найти максимальное число, записанное в файле.

```
int main ()
{
    char *S = new char[101];
    char C;
    long max=-1, curr;
    fstream f4("in4.txt");
    while (f4 >> S)
    {
        C=S[0];
        if (isdigit(C))
        {
            curr=atoi(S);
            if (curr > max)
                max=curr;
        }
    }
    if (max == -1)
        cout << " No numbers entered" << endl;
    else
        cout << max << endl;
    delete [] S;
    return 0;
}
```

КОНЕЦ ЛЕКЦИИ