Вычислительная техника и компьютерное моделирование в физике

Лекция 4

Зинчик Александр Адольфович

zinchik_alex@mail.ru

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.

- 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).

- 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

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

- Для поддержки потоков библиотека C++ содержит иерархию классов, построенную на основе класса — ios.
- Класс ios содержит общие для ввода и вывода поля и методы. От этого класса наследуется класс istream для входных потоков и ostream — для выходных.
- Два последних класса являются базовыми для класса iostream, реализующего двунаправленные потоки.

ios	базовый класс потоков
istream	класс входных потоков
ostream	класс выходных потоков
iostream	класс двунаправленных потоков
ifstream	класс входных файловых потоков
ofstream	класс выходных файловых потоков
fstream	класс двунаправленных файловых потоков

- 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, определенными пользователем. Для этого требуется переопределить операции потоков.

- К недостаткам потоков можно отнести снижение быстродействия программы, которое в зависимости от реализации компилятора может быть весьма значительным.

Стандартные потоки

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

cin	istream	Связывается с клавиатурой (стандартным буферизованным вводом)
cout	ostream	Связывается с экраном (стандартным буферизованным выводом)

- В классах istream и ostream **операции извлечения из потока >> и помещения в поток <<** определены путем перегрузки операций сдвига.

Пример:

```
#include <iostream.h>
int main(){
    int i;
    cin >> i;
    cout << "Вы ввели " << i;
    return 0;
}
```

- 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

- Величины при вводе должны разделяться пробельными символами (пробелами, знаками табуляции или перевода строки). Извлечение прекращается, если очередной символ оказался недопустимым.
- Операции << и >> перегружены для всех встроенных типов данных, что позволяет автоматически выполнять ввод и вывод в соответствии с типом величин.

Например:

```
#include <iostream.h>
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;
}
```

Форматирование данных

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

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 , 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
unitbuf	0x2000	Выгружать буферы всех потоков после каждого вывода
stdio	0x4000	Выгружать буферы потоков stdout и stderr после каждого вывода

- 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

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.
- 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;
int x_fill	символ заполнения поля вывода.

int ios::width(); int ios::width(int);	Возвращаетзначение ширины поля вывода;
int ios::precision(); int ios::precision(int);	возвращает (устанавливает) значение точности представления при выводе вещественных чисел;
char fill(); char fill(char);	возвращает (устанавливает) значение текущего символа заполнения, возвращает старое значение символа.

- 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)

```
#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;
}
```

- 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*.

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;

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;
setw(int)	устанавливает максимальную ширину поля вывода.

- Пример 1:
- cout << 13 << hex << ' ' << 13 << oct << ' ' << 13 << endl;
- Пример 2:

```
#include <iostream.h>
#include <iomanip.h>
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;
}
```

Методы обмена с потоками

- В потоковых классах наряду с операциями извлечения >> и включения << определены методы для неформатированного чтения и записи в поток
- 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 с;

get(buf,num,lim='\n')	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.
getline(buf, num, lim='\n')	аналогична функции get, но копирует в buf и символ lim;
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;
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)
- ios::end (от конца файла);

tellg()	возвращает текущую позицию чтения потока;

- 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:
tellg()	возвращает текущую позицию записи потока;
write(buf, num)	записывает в поток num символов из массива buf и возвращает ссылку на поток.

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

```
#include "iostream.h"
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;
}
```

Файловые потоки

- Под файлом обычно подразумевается именованная информация на внешнем носителе, например, на жестком или гибком магнитном диске. Логически файл можно представить как конечное количество последовательных байтов, поэтому такие устройства, как дисплей, клавиатуру и принтер также можно рассматривать как частные случаи файлов.

- По способу доступа файлы можно разделить на *последовательные*, чтение и запись в которых производятся с начала байт за байтом, и *файлы с произвольным доступом*, допускающие чтение и запись в указанную позицию.

Стандартная библиотека содержит три класса для работы с файлами

ifstream	класс входных файловых потоков;
ofstream	класс выходных файловых потоков;
fstream	класс двунаправленных файловых потоков.

Эти классы являются производными от классов istream, ostream и iostream соответственно, поэтому они наследуют перегруженные операции << и >>, флаги форматирования, манипуляторы, методы, состояние потоков и т. д.

Использование файлов в программе предполагает следующие операции:

- создание потока;
- открытие потока и связывание его с файлом;
- обмен (ввод/вывод);
- уничтожение потока;
- закрытие файла.

Каждый класс файловых потоков содержит **конструкторы**, с помощью которых можно создавать объекты этих классов различными способами.

- Конструкторы без параметров создают объект соответствующего класса, не связывая его с файлом:
- ifstream();
- ofstream();
- fstream();

- Конструкторы с параметрами создают объект соответствующего класса, открывают файл с указанным именем и связывают файл с объектом:

- ifstream(const char *name, int mode = ios::in);
- ofstream(const char *name, int mode = ios::out | ios::trunc);
- fstream(const char *name, int mode = ios::in | ios::out);

- Вторым параметром конструктора является режим открытия файла.

in	0x01	Открыть для чтения
out	0x02	Открыть для записи
ate	0x04	Установить указатель на конец файла
app	0x08	Открыть для добавления в конец
trunc	0x10	Если файл существует, удалить
nocreate	0x20	Если файл не существует, выдать ошибку
noreplace	0x40	Если файл существует, выдать ошибку
binary	0x80	Открыть в двоичном режиме

- Открыть файл в программе можно с использованием либо конструкторов, либо метода open, имеющего такие же параметры, как и в соответствующем конструкторе, например:

```
ifstream inpf ("input.txt");      // Использование конструктора
if (!inpf){
    cout << "Невозможно открыть файл для чтения"; return 1;
}
ostream f;
f.open("output.txt", ios::out); // Использование метода open
if (!f){
    cout << "Невозможно открыть файл для записи";
    return 1;
}
```

```
#include <fstream.h>
int main(){
    char text[81], buf[81];
    cout << "Введите имя файла:";
    cin >> text;
    ifstream f(text);
    if (!f){
     cout << "Ошибка открытия файла";
     return 1;
    }
```

```
  while (!f.eof()){
   f.getline(buf, 81);
   cout << buf << endl;
  }
  return 0;
}
```

- Для закрытия потока определен метод close(), но поскольку он **неявно выполняется деструктором**, явный вызов необходим только тогда, когда требуется закрыть поток раньше конца его области видимости.

Структуры (struct)

В отличие от массива, все элементы которого однотипны, структура может содержать элементы разных типов.

Элементы структуры называются *полями структуры* и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него.

```
struct [ имя_типа ] {
    тип_1 элемент_1;
    тип_2 элемент_2;

    тип_n элемент_n;
} [ список_описателей ];
```

Определение массива структур и указателя на структуру:

```
struct {
    char fio[30];
    int date, code;
    double salary;
}stuff[100], *ps;
```

Если список отсутствует, описание структуры определяет новый тип, имя которого можно использовать в дальнейшем наряду со стандартными типами, например:

```
struct Worker{        // описание нового типа char
   fio[30];
   int date, code;
   double salary;
};// описание заканчивается точкой с запятой
```

Имя структуры можно использовать сразу после его объявления (определение можно дать позднее) в тех случаях, когда компилятору не требуется знать размер структуры, например:

```
struct List;    // объявление структуры List;

struct Link{
    List *p;
    Link *prev, *succ;
};
struct List { /* определение структуры List */};
```

Это позволяет создавать связные списки структур.

Для **инициализации структуры** значения ее элементов перечисляют в фигурных скобках в порядке их описания:

```
struct{
    char fio[30];
    int age, code;
    double salary;
}worker = {“Иванов", 31, 512, 1234.56};
```

При инициализации массивов структур следует заключать в фигурные скобки каждый элемент массива (учитывая, что многомерный массив — это массив массивов):

```
struct complex{
    float real, im;
} compl [2][3] = {
{{1, 1}, {1, 1}, {1, 1}},     // строка 1, то есть
    массив compl[0]
{{2, 2}, {2, 2}, {2, 2}}      // строка 2, то есть
                              массив compl[1]
};
```

- Для переменных одного и того же структурного типа определена **операция присваивания**, при этом происходит поэлементное копирование. Структуру можно передавать в функцию и возвращать в качестве значения функции.
- Размер структуры не обязательно равен сумме размеров ее элементов, поскольку они могут быть выровнены по границам слова.

Доступ к полям структуры выполняется с помощью операций выбора . (точка) при обращении к полю через имя структуры и

-> при обращении через указатель, например:

```
Worker stuff[100], *ps, worker;
worker.age = 25;
stuff[8].code = 215;
ps->salary = 0.12;
```

Если элементом структуры является другая структура, то доступ к ее элементам выполняется через две операции выбора:

```
struct A {int a; double x;};
struct B {A a; double x;} x[2];
x[0].a.a = 1;
x[1].x = 0.1;
```

Заключение

- Спасибо за внимание!
- Вопросы???