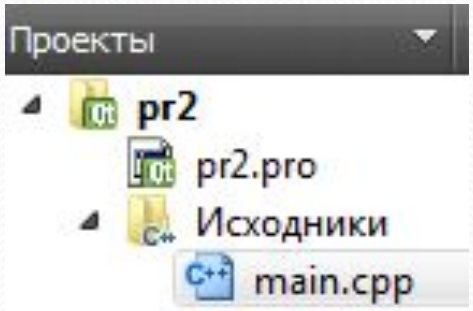
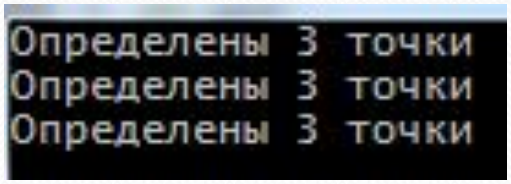
Программирование на C++

Тема1.2 C++. Введение в классы и объекты

Статические поля класса

Пример 7: `// main.cpp`






```
#include <iostream>
using namespace std;
class TPoint
  {  double x,y;
    public:
      static int N;  //количество точек
      TPoint(double x1=0.0,double y1=0.0)
        { N++; x = x1; y = y1;}
  };
int TPoint::N=0; //определение и инициализация статич. поля
int main(void)
 { TPoint A(1.0,2.0); TPoint B(4.0,5.0);
   TPoint *C = new TPoint(7.0,8.0);
   cout << "Определены " << TPoint::N << " точки\n";
   cout << "Определены " << A.N << " точки\n";
   cout << "Определены " << C->N << " точки\n";
   delete C;}
```

- Статические поля класса объявляются внутри класса с использованием ключевого слова **static**. Специфика статических полей:

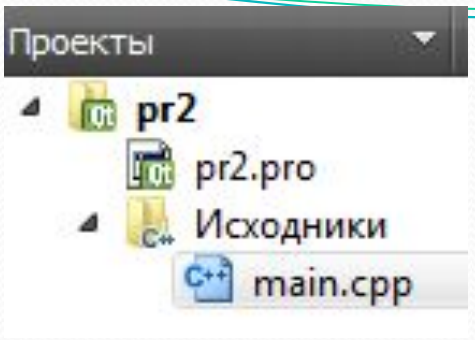
  а) **размещаются в статической области памяти** на стадии компиляции (когда еще не создано ни одного объекта!) независимо от того, где создается сам объект.

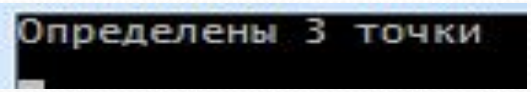
  б) **существуют в единственном экземпляре** (то есть компилятор выделяет под нее память только один раз!) независимо от того, сколько создано объектов. Это означает, что для полей класса с ключевым словом static при создании объекта память не резервируется.

  - Статическое поле класса является общим для всех объектов этого класса. Необходимость в таких переменных продиктована рядом практических соображений, и их использование во многих случаях оправдано. Например, в программе нужно контролировать количество созданных на данный момент объектов определенного класса. В этом случае создается статический член класса, значение которого определяется количеством объектов в работе.
  - Для того чтобы компилятор отвел под статическую переменную место в статической памяти ее необходимо определить и инициализировать вне функций и вне объявления класса. Статические переменные независимо от спецификатора доступа определяются и инициализируются одинаково.
  - Статическая переменная класса по сути является глобальной, заключенной в пространство имен класса , поэтому **к public статической переменной можно обращаться посредством имени класса и операции области видимости** (cout << TPoint::N). С другой стороны, формально статическая переменная является членом класса, поэтому ничто не мешает обращаться к ней посредством объекта: cout << A.N << C->N.
  - Обращение к статической переменной внутри методов класса для программиста ничем не отличается от обращения к обычной переменной, компилятор же подставляет обращение по адресу статической переменной, поэтому статические переменные никогда не должны инициализироваться в конструкторе.

Статические методы класса

Пример 8: // main.cpp






```
#include <iostream>
using namespace std;
class TPoint
  {    double x,y;
       static int N;//количество точек
     public:
       TPoint(double x1=0.0,double y1=0.0)
        { N++; x = x1; y = y1;}
       static int& count(){return N;}
  };
int TPoint::N; //определение и инициализация
               // статического поля (по умолчанию 0)
int main(void)
 { TPoint A(1.0,2.0); TPoint B(4.0,5.0);
   TPoint *C = new TPoint(7.0,8.0);
   cout << "Определены " << TPoint::count() << " точки\n";
   delete C;}
```

- **Статический метод** – это по сути глобальная функция, область видимости которой ограничена именем класса, поэтому (если она public) фактически вызвать ее можно, указывая имя класса и спецификатор области видимости, но формально она является членом класса, поэтому вызывать ее можно также посредством объекта или указателя на объект.
- Основные *отличия статического метода класса от обычного*:
  - указатель this в статических методах не существует;
  - Чтобы обратиться к нестатическим полям класса в статическом методе нужно им передать объект как параметр;
- Статические методы используются для доступа private или protected static-данным класса (см. пример на слайде).

Указатель на функцию

- **Объявление указателя на функцию:**

```
тип (*имя_указателя)(типы параметров функции)
```

- **Вызов функции по указателю:**

```
(*имя_указателя)(список фактических параметров)
```

Пример 9:

```
// main.cpp
#include <iostream>
using namespace std;
//суммирование двух  чисел
float sum(float a, int b)
{  return a + b;}

int main(void)
{ float (*ptr)(float, int);  // описание указателя на функцию
  ptr = sum;                 // присвоение указателю адреса функции
  float x = sum(0.5, 1);          // вызов функции по имени
  float y = (*ptr)(1.5, 2);       // вызов функции по указателю
  cout << "x = " << x << ' ' << "y = " << y << endl;
}
```

```
x = 1.5 y = 3.5
```

- В определении указателя на функцию количество и тип параметров должны совпадать с соответствующими типами в определении функции, на которую ставится указатель.
- Присвоение указателю на функцию адреса функции не требует операции взятие адреса (&), поскольку имя функции само по себе обозначает этот адрес.

Указатель на функцию как параметр функции

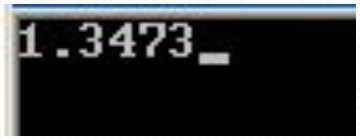
Пример 10:

```
//Нахождение корня нелинейного уравнения методом
// деления пополам
#include <iostream>
#include <math.h>
using namespace std;

double func(double x);
double bisect(double a, double b, double eps,
              double (*f)(double))
{ double c, y;
  do { c = 0.5*(a+b); y = f(c);
     if (fabs(y) < eps) break; //корень найден. Выход из цикла
  // Если на концах отрезка [a, c] функция имеет разные знаки
     if (f(a)*y < 0.0) b = c; // значит, корень здесь.
                              // Переносим точку b в точку c
        // В противном случае:
        else a = c; // переносим точку a в точку c
      // Продолжаем, пока отрезок [a, b] не станет мал
   }while(fabs(b-a) >= eps);
  return c;
}
```

Указатель на функцию как параметр функции

Продолжение примера 10:

```
int main(void)
{ double a = 0.0, b = 1.5, eps = 1e-5;
  cout << bisect(a, b,eps,func)<< endl;
}

double func(double x)
{ return x*x*x - 3*x*x + 3; }
```



Функция qsort

Прототип функции qsort (stdlib.h (C), cstdlib (C++):

```
void qsort(void* first, size_t num, size_t size,
           int (* comp) (const void*, const void*));
```

- **Параметры:**
  - **first** - указатель на первый элемент сортируемого массива;
  - **num** - количество элементов в сортируемом массиве, на который ссылается указатель first;
  - **size** - размер одного элемента массива в байтах;
  - **comp –** указатель на функцию, которая сравнивает два элемента массива. Функция должна иметь следующий прототип:

```
int compare(const void* v1, const void* v2)
```

- Функция qsort выполняет сортировку num элементов массива, на который ссылается указатель first. Для каждого элемента массива устанавливается размер в байтах, который передается через параметр size. Последний параметр функции qsort— указатель comp на функцию сравнения, которая используется для определения порядка следования элементов в отсортированном массиве.
- Функция сравнения (compare) имеет два параметра - не типизированные указатели (void* v1, void* v2) на элементы массива. Эти параметры должны быть приведены к указателям на те типы данных, которые имеют элементы массива. Возвращаемое значение этой функции должно быть отрицательным, положительным или равным нулю. Если необходимо выполнить сортировку элементов массива по возрастанию, то если *v1 < *v2, тот функция должна вернуть отрицательное значение, если *v1 > *v2, тот функция должна вернуть положительное значение, если *v1 == *v2, тот функция должна вернуть ноль.

Пример 11.

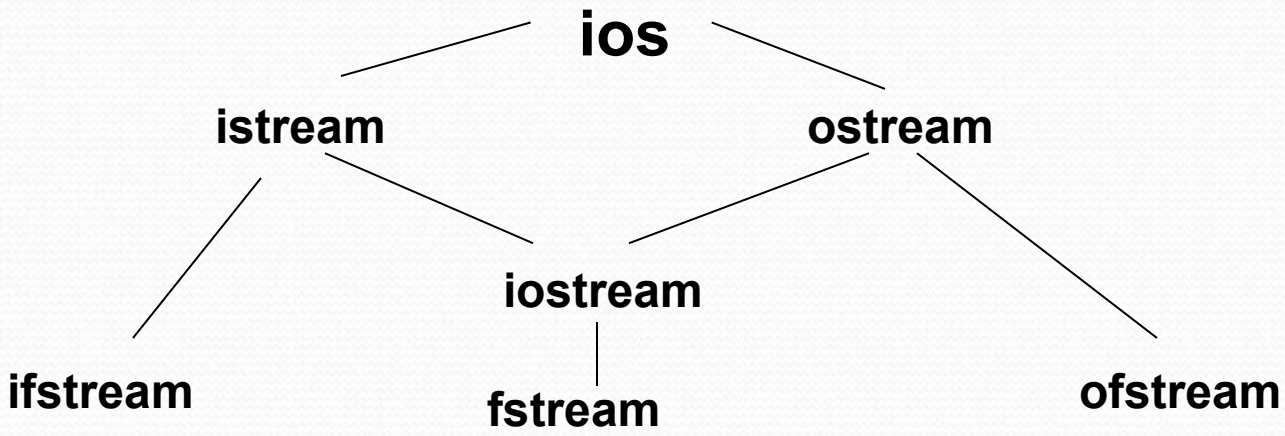
```
//пример использования функции qsort
#include <iostream>
#include <cstdlib>
using namespace std;

int vector[] = { 14, 10, 11, 19, 2, 25 };

int compare(const void * x1, const void * x2)
{if (*(int*)x1 < *(int*)x2) return -1;
   else if (*(int*)x1 > *(int*)x2) return 1;
           else return 0;
}
int main ()
{ qsort(vector, 6, sizeof(int), compare);
  for ( int i = 0; i < 6; i++)
      cout << vector[i] << " ";
  return 0;
}
```

```
2 10 11 14 19 25
```

Пространства имен

```
namespace имя
{ объявления и определения глобальных
  переменных, функций и классов
}
#include <iostream>
#include <cmath>
namespace n1
{ int pow(int x, int y) { return x + y;}
}
namespace n2
{ int pow(int x, int y) { return x - y;}
}
int main()
 { std::cout << n1::pow(2, 1) << std::endl
             << n2::pow(2, 1) << std::endl
             << std::pow(2, 1) << std::endl;
 }
```

Потоковые классы C++



*

- В C++ ввод-вывод можно организовать с использованием процедурно-ориентированной библиотеки <stdio.h> (<cstdio> в стандарте C++). или объектно-ориентированной библиотеки <iostream>. В этих библиотеках реализуется концепция ввода-вывода, независимого от устройств. Несмотря на то, что устройства ввода-вывода имеют разные характеристики, система ввода-вывода C++ предоставляет программисту единый удобный интерфейс. Для этого используется абстрактное понятие «поток», относящееся к любому переносу данных от источника к приемнику. Поток – последовательность символов. Программа имеет дело не с устройствами или с файлами, а с потоками. Ввод информации осуществляется из входного потока, вывод программа производит в выходной поток. А уж поток можно связать и с устройством, и с файлом.
- В программе поток представлен потоковой переменной определенного типа: в <stdio.h>, например, указателем на структуру FILE, а в <iostream> объектами потоковых классов (например, istream, ostream, iostream, …).
- Потоки подразделяют на: входные и выходные; из входного потока информация считывается; в выходной поток данные записываются; двунаправленный поток реализует чтение и запись;
- В соответствии с особенностями «устройства», к которому «присоединен» поток, их делят на стандартные, файловые и строковые.
- Стандартные потоки соответствуют передаче данных «от клавиатуры» и «к экрану». Стандартные потоки бывают только однонаправленными: либо информация только читается из потока, либо – только пишется в поток. Если символы потока размещаются на внешнем носителе данных (диске и т.д.) – это файловый поток. Если символы потока в совокупности образуют символьный массив (строку) в основной памяти, то это строковый поток. Файловые и строковые потоки могут быть и двунаправленными: из потока можно вводить информацию и в тот же поток – выводить.

- Потоки подразделяют на: буферизуемые и не буферизуемые; при обмене с потоком часто используется вспомогательный участок основной памяти – буфер потока; при буферизованном потоке вывод выполняется не на устройство, а в буфер; использование буфера как промежуточной ступени при обменах с внешними устройствами повышает скорость передачи данных, так как реальные пересылки осуществляются только тогда, когда буфер уже заполнен (при выводе) или пуст (при вводе); данные помещаются в буфер, перед тем как они будут переданы к внешнему устройству (при выводе данных) или перед тем, как они будут переданы в область памяти выполняемой программы; заполненный буфер выводится на устройство, как правило, без непосредственного указания в программе; однако программа может потребовать вывести неполный буфер; при вводе буфер обычно заполняется при выполнении первой операции ввода, специально программировать это не требуется; обычно буферизация реализуется по умолчанию, но в библиотеках есть средства, позволяющие управлять назначением буферов.
- Потоки подразделяют на: форматируемые и не форматируемые; форматируемость означает, что при операциях ввода-вывода выполняется преобразование информации: при вводе, как правило, преобразование данных из внешнего представления (символьного вида) в двоичное (внутреннее) представление, а при выводе – наоборот; форматирование всегда выполняется для стандартных потоков;
- С потоком связывается внутренний указатель, значение которого задает позицию для выполнения следующей операции чтения или записи;
- Каждый поток (в том числе и стандартный) в каждый момент времени находится в некотором состоянии; эти состояния называются good, eof , bad, fail, hardfail;
- Средства ввода-вывода C++ могут обеспечить последовательный и прямой доступ к данным,

Стандартные потоки

- **cin** – объект класса **istream**, соответствующий стандартному потоку ввода, по умолчанию связан с клавиатурой;
- **cout –** объект класса **ostream**, соответствующий стандартному потоку вывода, по умолчанию связан с экраном;
- **cerr –** объект класса **ostream**, соответствующий стандартному не буферизованному потоку вывода для ошибок, по умолчанию связан с экраном;
- **clog –** объект класса **ostream**, соответствующий стандартному буферизованному потоку вывода для ошибок, по умолчанию связан с экраном;

Операции ввода - вывода

- **>>** - операция ввода (чтения) из потока;
- **<<** - операция вывода (записи) в поток;
- **int k = 5; cout << k; // cout.operator<<(k)**

- При включении в программу заголовка <iostream> в программе автоматически становятся доступными стандартные потоки (глобальные потоковые объекты, см. слайд).
- Вместе с классами потоков и глобальными потоковыми объектами в библиотеке определены операция чтения (ввода) из потока **>>** и операция записи (вывода) в поток **<< .**
- Эти операции созданы перегрузкой операций побитового сдвига. Операция сдвига влево ‘<<’ перегружена для вывода в поток. Операция сдвига вправо ‘>>’ перегружена для ввода из потока.
- В классе ostream, например, определены перегруженные операции:

  ostream& operator<< (ostream&, short); //вывод целых

  ostream& operator<< (ostream&, int);

  ostream& operator<< (ostream&, long);

  ostream& operator<< (ostream&, char); //вывод символов

  ostream& operator<< (ostream&, double); //вывод вещественных

  ostream& operator<< (ostream&, const char *); //вывод строк

  и т.д.
- Если в программе встретится, например, оператор **cout << k;** компилятор выполнит:

  • сначала определит, что левый аргумент (объект **cout**) имеет тип **ostream**, а правый k– тип int;

  • в определении класса объекта **cout** найдет прототип соответствующего метода (функции_операции **operator<<()): ostream& operator<< (ostream&, int);**

  • для объекта **cout** вызовет метод **cout.operator<<(k);** в результате работы которого мы увидим на экране число 5; при этом обеспечивается преобразование значения встроенного типа int в строку символов с параметрами форматирования по умолчанию.
- Аналогично происходит работа со стандартным потоком ввода через объект cin, только возвращаемым значением должны быть ссылка на поток ввода **istream &.**

Стандартные потоки. Флаги форматирования

флаг	назначение
ios::boolalpha	представлять логические значения не в числовом, а в символьном виде, т.е. true и false
ios::dec	десятичная система счисления (установка по умолчанию);
ios::fixed	для вещественных чисел представление в формате с фиксированной точкой
ios::hex	шестнадцатеричная система счисления
ios::internal	символ-заполнитель пустых позиций помещается между символом знака (или символом основания системы счисления) и числовым значением
ios::left	выводимое значение выравнивается влево (используется символ заполнения – по умолчанию «пробел»)
ios::oct	выводимые числа форматируются как восьмеричные
ios::right	выводимое значение выравнивается вправо – это установка по умолчанию (используется символ заполнения)
ios::scientific	числа с плавающей точкой выводятся в экспоненциальной форме (научный формат)

Стандартные потоки. Флаги форматирования

ios::showbase	выводит основания системы счисления (0 – для восьмеричной, 0x – для шестнадцатеричной)
ios::showpoint	при выводе вещественных чисел печатать десятичную точку и и следующие за ней нули
ios::showpos	печатать знак положительного числа
ios::skipws	при чтении данных из входного потока игнорировать начальные пробельные символы
ios::unitbuf	очищать все потоки (выгружать содержимое буфера) после каждого ввода или вывода
ios::uppercase	при выводе использовать символы верхнего регистра
полезные значения битовой маски	
ios::adjustfield	маска для битов флага «дополнение» **internal** | **left** | **right** (внутри, слева, справа)
ios::basefield	маска для битов флага «основание системы счисления» **dec** | **hex** | **oct** (десятичная, шестнадцатеричная, восьмеричная)

Стандартные потоки. Методы работы с флагами

функция-член	назначение
flags() flags(флаги)	устанавливает или читает флаги формата
setf(флаги) setf(флаги, маска)	устанавливает флаги формата
unsetf (флаги)	сбрасывает флаги формата
width(w) w= width();	устанавливает ширину поля возвращает текущее значение ширины поля
fill(ch); ch=fill();	устанавливает символ-заполнитель читает символ-заполнитель
precision(P) P= precision();	устанавливает точность читает точность

Стандартные потоки. Примеры использования флагов и методов класса IOS

```
cout.flags (ios::hex | ios::uppercase);
cout <<15 << '\n';                                  //выводит F

cout.flags (ios:: oct | ios:: showbase);
cout <<8 << '\n';                                    //выводит 010

cout.flags (ios:: hex | ios:: uppercase| ios::showbase );
     cout <<15 << '\n';                                    //выводит 0XF

int nNum=16;
double fx=355.113;
long iFlags ;
cout.flags (ios::dec);
cout <<nNum << "(base 10) =";                              //16 (base 10)=

cout.flags (ios::hex | ios:: showbase);
cout <<nNum << "(base 16) =";                              //0x10 (base 16)=

cout.flags (ios:: oct | ios:: showbase);
```

Стандартные потоки. Примеры использования флагов и методов класса IOS

```
cout << nNum << "(base 8) \n";                                    //020 (base 8)
cout.flags (ios:: scientific | ios:: showpos);
cout << "floating-point number is"<< fx<< '\n';
                                    //floating-point number is +3.551130e+002

cout.flags (ios:: fixed | ios:: showpos);
cout << "floating-point number is"<< fx<< '\n';
                                    //floating-point number is +355.113000

iFlags= cout.flags();  //получить флаги
if (iFlags & ios::dec)
cout << "integer output uses base 10\n";
else if(iFlags & ios::hex)
cout << "integer output uses base 16\n";
else
cout << "integer output uses base 8\n";     // integer output uses base 8
```

- Стандартные потоки являются форматируемыми. Для организации ввода-вывода с использованием стандартных потоков можно использовать параметры форматирования установленные по умолчанию или изменить их.
- Для изменения параметров форматирования, можно воспользоваться либо флагами и методами класса ios, либо манипуляторами ввода/вывода, представляющими собой функции, которые можно включать прямо в поток (файл <iomanip>).
- Флаги форматирования реализованы в виде отдельных фиксированных битов переменной (типа long) представления флагов. Поэтому несколько флагов с помощью логических битовых выражений можно объединять, тем самым по-разному комбинируя свойства потока. Список флагов форматирования представлен в таблице на слайдах 39-40.
- Проверить значения любых флагов, установить или сбросить их позволяют методы класса ios, доступные через объекты cin и cout (слайд 41).
- Примеры использования флагов и методов класса ios показаны на слайдах 42 и 43.
- Наиболее простой способ изменения параметров и флагов форматирования обеспечивают специальные функции-манипуляторы, непосредственно воздействующие на потоковый вывод. Особенность манипуляторов и их отличие от обычных функций состоит в том, что их имена (без параметров) и вызовы (с параметрами) можно использовать в качестве правого операнда для операции обмена << или >>. В качестве левого операнда в этом выражении, как обычно, используется поток (объект, представляющий поток), и именно на этот поток оказывает влияние манипулятор.
- Все изменения в потоке, внесенные манипулятором, сохраняются до следующей установки, за исключением setw(). См. слайды 45, 46, 47.

Стандартные потоки. Манипуляторы с параметрами

манипулятор	назначение
setw(int nWidth=0)	устанавливает минимальную ширину следующего поля (0 – по умолчанию)
setbase (int nBase=10)	устанавливает основание системы счисления (10 – по умолчанию)
setfill (char cFill=' ')	устанавливает символ-заполнитель (пробел – по умолчанию)
setprecision (int nPrec=6)	количество цифр дробной части;
setiosflags (long lFlags)	устанавливает флаги формата потока
resetiosflags (long lFlags)	сбрасывает флаги формата потока
ws	при вводе извлекает из входного потока (и игнорирует) пробельные символы

Стандартные потоки. Манипуляторы без параметров

манипулятор	назначение
binary	устанавливает двоичный режим потокового ввода/вывода
text	устанавливает текстовый режим потокового ввода/вывода
dec	при вводе и выводе устанавливает флаг 10 с/с
hex	при вводе и выводе устанавливает флаг 16 с/с
oct	при вводе и выводе устанавливает флаг 8 с/с
endl	при выводе включает в выходной поток символ новой строки и сбрасывает буфер (очищает выходной поток)
ends	при выводе включает в выходной поток нулевой признак конца строки
flush	при выводе очищает выходной поток

Стандартные потоки. Пример использования манипуляторов

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

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;
}
```

```
......1.2340
....-12.3457
....123.4568
.....-1.2340
......0.0000
Для продолжения нажмите любую клавишу . . .
```

Контрольные вопросы

1. Статические поля и методы класса: назначение, синтаксис описания и примеры использования
2. Классы и объекты стандартных потоков ввода-вывода C++. Как в C++ организуется форматируемый ввод-вывод данных с использованием флагов и манипуляторов? Примеры.
3. Пространства имен в C++: назначение, пример использования.