Лекция N 1

Язык программирования C++

C++ был разработан на основе языка программирования C и сохраняет его как подмножество.

Язык программирования C был разработан Денисом Ричи в лаборатории фирмы Bell в начале семидесятых годов. Одна из первоначальных целей создания языка C - замена ассемблера в задачах системного программирования. Поэтому в этом языке есть такие средства, как указатели, побитовые операции, операции поразрядного сдвига. На этом языке была разработана операционная система UNIX для мини-ЭВМ PDP-11.

Однако именно те средства, которые позволяют на C писать мощные и компактные программы делают его беззащитным для целого класса ошибок, от которых есть защита в других языках программирования.

C++ является расширением языка C. Основная цель этого расширения - поддержка объектно-ориентированного программирования. Ключевым понятием C++ является класс. Кроме того там есть еще ряд средств, которые не относятся непосредственно к объектно-ориентированному программированию, такие как перегрузка операций и функций, inline-функции и так далее. C++ более типизирован и более защищен от ошибок, чем C.

Изложение языка C++ будет сделано в 2 этапа: вначале быстрое, не совсем формальное введение в C++ на примерах задач, которое позволит сразу же начать программирование, а затем последующее изложение остального.

Следующая программа осуществляет вывод на экран монитора сообщения "Hello, hello!".

```
#include <stdio.h>
void main(void)
{
  printf("Hello, hello!\n");
}
```

Прокомментируем эту программу. Все строки, которые начинаются с символа "#" - это директивы препроцессора. Препроцессор – это программа, которая предварительно преобразует (препроцессирует) текст программы, после чего передает его компилятору. В данном случае эта директива сообщает препроцессору, что вместо нее необходимо вставить содержимое файла stdio.h (header файла, заголовочного файла). Все функции в языке C разбиты на отдельные группы. Для каждой группы функций необходим свой заголовочный файл, в котором находятся прототипы этих функций (то есть описания правильного обращения к функциям), определения необходимых структур данных, различных констант. Файл stdio.h поддерживает работу функций, обеспечивающих стандартный ввод-вывод высокого уровня.

Вторая строка сообщает компилятору, что это основная (главная) функция п при запуске программы именно ей будет передано управление. Первое ключевое слово void говорит о том что эта функция не возвращает после окончания работы

никакого значения, второе слово – что при запуске функции из командной строки ей не осуществляется передача аргументов. Тело функции ограничено фигурными скобками.

Четвертая строка - это оператор-выражение, обращение к функции вывода printf. Эта функция выводит форматную строку, заключенную в "" на дисплей. Строка выводится в то место, где в данный момент находится курсор. Тот же самый результат был бы и в следующем случае:

```
printf("Hello, ");
printf("hello!\n");
```

где \n - это управляющий символ. Символ "\" всегда рассматривается вкупе со следующим за ним символом. Наиболее используемые управляющие символы:

\n - перевод строки;
\r - возврат каретки;
\t - горизонтальная табуляция;
\v - вертикальная табуляция;
\f - перевод формата.

В языке С каждый оператор завершается символом ";". В языке С прописные и строчные символы - это разные символы.

Следует заметить, что в языке С ввод-вывод поддерживается не операторами, а функциями.

Более подробное описание ввода-вывода приведено в лекции 8 и приложении 01.

Вывод в этой программе может быть реализован и по другому: с помощью тех средств, которые появились в С++.

```
#include <iostream.h>
void main (void)
{
  cout << "Hello, World!\n";
}
```

Файл iostream.h содержит описания классов (типов данных), перегруженных операций, необходимых для поддержки стандартных потоков ввода-вывода С++. Без этих описаний выражение

```
cout << "Hello, World!\n"
```

не имело бы смысла. Перегруженная операция << (в языке С - это операция поразрядного сдвига влево) записывает значение своего второго параметра в первый параметр. В данном случае строка "Hello, World!\n" записывается в стандартный выходной поток cout (console out). Строка - это последовательность символов, заключенная в двойные кавычки.

Более подробное описание этого типа ввода-вывода приведено в приложениях 02 и 03.

Оператор цикла WHILE

Синтаксис

```
while (выражение)
    оператор;
```

Оператор выполняется до тех пор, пока значение выражения не станет ЛОЖЬ. В этом случае управление передается следующему оператору.

!!! В языке С значение выражения - ИСТИНА, если оно отлично от нуля. !!!

Если в цикле должны выполняться несколько операторов, то они объединяются в составной оператор с помощью фигурных скобок {}.

Все условные выражения в языке С++ обязательно заключаются в ().

Следующая программа осуществляет печать таблицы площадей кругов в зависимости от радиуса.

R = 1 (1) 20

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
 int r;
 float s;
 r=1:
 clrscr();
 while (r<=20)
 {
  s=3.1416*r*r;
  printf("For r=%2d   s=%7.2f\n",r,s);
  r=r+1;
 }
 getch();
}
```

Для функций clrscr() – очистка экрана и getch() – чтение кода нажатой клавиши необходим заголовочный файл conio.h. Функция getch() в данном примере используется для ее приостановки программы до нажатия любой клавиши.

Стандартная ошибка начинающих:

```
while (r<=20);
```

В этом случае в цикле выполняется пустой оператор и программа зацикливается.

Что нового появилось в этой программе?

В этой программе появились определения типов данных. В С существуют следующие основные встроенные типы данных.

Знаковые целочисленные типы данных:

Тип	Размер	Диапазон значений
[signed] char	1 байт	-128 … 127
[signed] short [int]	2 байта	-32 768 … 32 767
[signed] int	2 байта	-32 768 … 32 767
[signed] long [int]	4 байта	-2 147 483 648 … 2 147 483 647

[] обозначают необязательность написания данного элемента.

Беззнаковые целочисленные типы:

Тип	Размер	Диапазон значений
unsigned char	1 байт	0 … 255
unsigned short [int]	2 байта	0 … 65 535
unsigned [int]	2 байта	0 … 65 535
unsigned long [int]	4 байта	0 … 4 294 967 295

Вещественные типы:

Тип	Размер	Диапазон значений
float	4 байта	3.4e-38 … 3.4e+38
double	8 байт	1.7e-308 … 1.7e+308
long double	10 байт	3.4e4932 … 3.4e+4932

Для вещественных типов в таблице приведены абсолютные величины минимальных и максимальных значений.

Следует отметить, что на других платформах может быть другой размер типа int. Для его получения необходимо пользоваться операцией sizeof, результатом которой является размер типа в байтах. Например, в операционной системе MS-DOS sizeof(int) дает результат 2, а в Widows XP результатом будет 4.

Следующие арифметические операции можно использовать над любым сочетанием перечисленных типов:

+ (сложение)
++ (сложение инкрементальное)
- (вычитание)
-- (вычитание инкрементальное)
*** (умножение)**
/ (деление)
% (остаток от деления)

То же верно для операций отношения:

== (равно)
!= (не равно)

< (меньше чем)
<= (меньше или равно)
>= (больше или равно)

Результат операций отношения нормализован: 0 - ложь, 1 - истина, их значения можно использовать точно так же, как и прочие числовые. Например:

```
y=10+(x>1);
```

Результат будет 11, если x>1 и 10, в противном случае.

Символ = обозначает операцию присваивания, а == операцию проверки на равенство. При освоении Си часто вместо == пишут =, что приводит к логическим ошибкам, например:

```
while(x=1)
{ ... }
```

Этот цикл будет выполняться бесконечно, поскольку x присваивается значение 1 и значение условного выражения всегда будет "истина".

Операции инкремента ++ и декремента -- могут быть как префиксные, так и постфиксные. Если операция префиксная, то значение переменной изменяется на 1 до использования в выражении, если постфиксная, то после использования.

Например

```
int i=1;
cout<<++i;   //2
cout<<i;     //2
cout<<i--;   //2
cout<<i;     //1
```

В арифметических выражениях можно использовать числовые значения различных типов. При вычислении используются стандартные преобразования типов (смотри конец 2-ой лекции).

```
double d=1;
int i=10;
short s=5;
 ...
d=d+(i=s+i);
```

Оператор цикла do-while

Синтаксис

```
do
   оператор;
while (выражение);
```

Оператор выполняется до тех пор, пока значение выражения не станет ЛОЖЬ (то есть 0). В этом случае управление передается следующему оператору.

Значение выражения проверяется после выполнения оператора. Поэтому оператор выполняется хотя бы один раз.

Оператор do-while проверяет условие в конце цикла.

Оператор while проверяет условие в начале цикла.

Пример:

```
  . . .
x=1;
do
   printf("%d\n",func(x,2));
while (++x<=7);
```

Оператор цикла for

Синтаксис

```
for (выражение1; выражение2; выражение3)
   оператор;
```

Выполнение оператора for эквивалентно выполнению последовательности следующих операторов:

```
выражение1;
while (выражение2)
{
   оператор;
   выражение3;
}
```

Ниже приведена программа для решения той же задачи, только с использованием оператора цикла for.

```
#include <stdio.h>
#include <conio.h>
{
  int r;
  float s;
  clrscr();
  for (r=1;r<=20;r++)
  {
    s=3.1416*r*r;
    printf("For r=%2d   s=%7.2f\n",r,s);
  }
  getch();
}
```

Данную программу можно записать более кратко, поместив все действия в оператор for, используя операцию “,” (последовательное вычисление подвыражений). Но лучше так не делать, так как текст стал гораздо менее читабельным.

```
 #include <stdio.h>
#include <conio.h>
void main(void)
{
  clrscr();
  for (int r=1;r<=20;printf("For r=%2d   s=%7.2f\n",r,3.1416*r*r),r++);
  getch();
}
```

В этой программе переменная r определена внутри цикла. Подобная конструкция является типичной для языка C++. Переменная, описанная в операторе цикла, видна только внутри этого цикла. В некоторых реализациях компиляторов видимость простирается до конца блока программы (в частности в Visual Studio), но это не является стандартом языка C++.

Условный оператор if - else

Синтаксис (1 вариант)

```
if (выражение)
   оператор;
```

Если значение выражения - истинно, то выполняется один единственный оператор (простой или составной), иначе этот оператор пропускается и выполняется следующий за ним оператор.

Синтаксис (2 вариант)

```
if (выражение)
   оператор1;
else
   оператор2;
```

В зависимости от истинности значения выражения выполняется или оператор1 или оператор2.

В следующем примере показано преобразование дюйма в сантиметр и обратно. Предполагается, что во входном потоке значение в сантиметрах завершается символом i, а значение в дюймах - символом c:

```
#include <iostream.h>
void main (void)
{
  float x, in, cm;
```

```
 char ch;
 cout << "enter length: ";
 cin >> x;              // ввод числа с плавающей точкой
 cin >> ch;             // ввод завершающего символа
 if (ch=='i')
 {                      // дюйм
    in=x;
    cm=x*2.54;
 }
 else if (ch=='c')
{                       // сантиметры
    in=x/2.54;
    cm=x;
}
 else
    in=cm=0;
 cout << in << "in = " << cm << "cm\n";
}
```

В этом примере появляется задание символьной константы, то есть константы, значение которой равняется коду символа. Символьная константа задается с помощью одинарных кавычек, ограничивающих конкретный символ. В данном примере это 'i'.

Оператор-переключатель switch

Синтаксис

```
switch (целочисленное выражение)
{
  case целочисленная_константа1:
     операторы
  [case целочисленная_константа2:
     операторы]
. . .
  [default:
     операторы]
}
```

[] обозначают необязательность конструкции, . . . – возможность многократного повторения.

Оператор switch (переключатель) сравнивает значение выражения с набором констант во всех ветвях case и передает управление первоиу оператору в той ветви, которая соответсвует значению выражения. Если такого соответствия нет, то управление передается ветви default, если она есть. Если ветви default не окажется, то никаких действий не будет выполнено. Ключевые слова case вместе с константами служат просто метками, и если будут выполняться операторы для некоторого варианта case, то далее будут выполняться операторы всех последующих

вариантов до тех пор, пока не встретиться оператор break. Это позволяет связывать одну последовательность операторов с несколькими вариантами.

Значения выражений и констант должны быть целочисленного типа.

if-else в предыдущем примере можно заменить на switch следующим образом:

```
   . . .
switch (ch)
{
  case 'i':
    in=x;
    cm=x*2.54;
    break;
  case 'c':
    in=x/2.54;
    cm=x;
    break;
  default:
    in=cm=0;
}
   . . .
```

Операторы break используются для выхода из переключателя. Пример связи операторов с несколькими ветвями приведен ниже:

```
   . . .
switch (x)
{
  case 'A':
    printf("CASE A\n");
    break;
  case 'B':
  case 'C':
    printf("CASE B or C\n");
    break;
  default:
    printf("NOT A, B or C\n");
}
   . . .
```

Оператор break

Синтаксис

```
break;
```

Прекращает выполнение ближайшего вложенного внешнего оператора switch, while, do или for. Этот оператор вызывает немедленный выход из самого внутреннего из объемлющих его циклов или переключателей. Управление передается первому

оператору, следующему за ними. Одно из назначений этого оператора - закончить выполнение цикла при достижении внутри тела цикла некоторого условия.

Например:

```
   . . .
for (i=0;i<n;i++)
  if ((a[i]=b[i])==0)
    break;
   . . .
```

В данном примере элементы массива b переписываются в массив a до тех пор, пока очередное переписываемое значение не окажется нулем.

Оператор continue

Синтаксис

```
continue;
```

Этот оператор в чем-то похож на break, но применяется гораздо реже. Он вынуждает ближайщий объемлющий ее цикл (for, while или do-while) начать следующий шаг итерации. Для while и do-while это означает немедленный переход к проверке условия, а для for - к приращению шага (то-есть к вычислению 3-го выражения).

Ниже приведен пример использования этого оператора:

```
   . . .
for (i=0;i<n;i++)
{
   if (a[i]!=0)
     continue;
   a[i]=b[i];
   . . .
}
```

В этом примере нулевые значения массива a заменяются значениями соответствующих элементов массива b.

Оператор-выражение

Любое выражение, заканчивающееся точкой с запятой (;), является оператором. Ниже приведены примеры операторов-выражений:

```
x=3;
printf("Для продолжения работы нажмите любую клавишу.\n");
getch();
```

Составной оператор

Составной оператор (блок) состоит из одного или более операторов любого типа, заключенных в фигурные скобки ({}). После закрывающей скобки не надо ставить «;» (хотя это можно и сделать и это не будет зафиксировано как ошибка). Пример:

```
{
  x=1;
  y=2;
  z=3;
}
```

Пустой оператор

Состоит только из точки с запятой (;). То есть если мы в предыдущем случае после «}» поставим «;» - это будет просто пустой оператор.

Метка оператора

Метка может стоять перед любым оператором, для того чтобы этому оператору можно было передать управление с помощью оператора goto. Метка состоит из идентификатора, за которым стоит двоеточие (:). Областью определения метки является данная функция. Пример метки:

```
ABCD2: x=3;
```

Оператор перехода goto

Синтаксис

```
goto метка;
```

Управление передается на оператор с меткой "метка". Область действия ограничена текущей функцией. Пример:

```
goto ABCD2;
// . . .
ABCD2:  оператор;
```

Метка и goto являются анахронизмами, пришедшими из языков неструктурного программирования, но в ряде случаев они бывают удобны. Например:

-- выход сразу из нескольких вложенных циклов;
-- переход из нескольких мест функции в одно.

Оператор возврата return

Синтаксис

return;

Прекращает выполнение текущей функции и возвращает управление вызвавшей программе без передачи значения.

Синтаксис

return выражение;

Прекращает выполнение текущей функции и возвращает управление вызвавшей программе с передачей значения "выражения". Пример:

return x+y;