ЛЕКЦИЯ 5.

Вопросы: **Инструкции.**

Инструкции (Statements)

Любое синтаксически правильно составленное предложение языка С, которое заканчивается символом ‘ ; ‘, называется инструкцией.

Любое число инструкций, объединённых в { }, называется блоком.

Любое выражение может быть преобразовано в инструкцию добавлением к нему точки с запятой. Запись вида: **выражение;**
является инструкцией. Значение выражения игнорируется. Действие такой инструкции состоит в создании побочного эффекта вычислением значения выражения.

Инструкции служат основными строительными блоками программы. Программа состоит из последова-тельности инструкций с добавлением знаков пунктуации.

`timer = 24` - это всего лишь выражение, которое может быть частью другого выражения.

`timer = 24;` - это инструкция присваивания, которая служит указанием компьютеру присвоить число 24 переменной timer.

Метки инструкций

Перед любой инструкцией может стоять метка, состоящая из идентификатора и символа “**:**”.

Управляющие инструкции С++

Инструкции присваивания

Особенности операторов С++ приводят к тому, что вместо обычного для других языков оператора присваивания используется более мощное понятие: ***инструкция-выражение*****.**

Записанный отдельной инструкцией оператор присваивания является лишь частным случаем такой инструкции.

Инструкция сложное присваивание

```
y+=2; /* Увеличение переменной y на 2*/
p+=n;  //    p=p+n;
x-=3;  //    x=x-3;
x*=x;  //    x=x*3;
x/=2;  //    x=x/3;
x%=10; //    x=x%3;
x>>=4; //    x=x>>4;
x<<=1; //    x=x<<1;
```

```
remitems &= mask;
Control ^= seton;
additems |= mask;
```

Пример 1.

```
/*Определить площадь треугольника, если известны три стороны*/
#include <iostream>
#include <cmath>
using namespace std;
float a=10, b=20, c=15;
double p, s;
int main()
{
     setlocale(LC_ALL,".1251");
     p = (a+b+c)/2;
     s = sqrt(p*(p-a)*(p-b)*(p-c));
     cout << "Площадь  = " << s << endl;
     return 0;
}
```

Пример 2.

```
/* Вычислить среднее арифметическое и среднее геометрическое трех чисел a,b,c */
#include <iostream>
#include <cmath>
using namespace std;
double a=10, b=20, c=15, p, s;
int main()
{
  setlocale(LC_ALL,".1251");
  s = (a+b+c)/3;
  p = pow((a*b*c),(1.0/3));
  cout << "ср.арифм.= " << s <<endl;
  cout << "ср.геом. = " << p <<endl;
  return 0;
}
```

Инструкции выбора if и if …else

Синтаксис:

```
if (условие)
     инструкция1;
[else
     инструкция2;]
```

Сначала вычисляется условие, указанное в круглых скобках. Если полученный результат равен **true** (не равен нулю), то выполняется **инструкция1**. Если результат вычисления условия равен **false** (равен нулю), то выполняется **инструкция2,** если есть часть **else** или инструкция, следующая за if-оператором.

Условие в интрукциях if может быть простым и сложным выражением.

Примеры простых условий:

```
1)if (x>0) sgrtx=sqrt(x);

2)if (a>b)
       max=a;
   else
       max=b;

3)if (a!=b)
    {
     r = r – a;
     k = k + b
    };
```

Пример сложного условия:

```
if (((a1==a2)&&(a1==a3)&&(a1!=a4))
 ||((a1==a2)&&(a1==a4)&&(a1!=a3))
 ||((a1==a3)&&(a1==a4)&&(a1!=a2))
 ||((a2==a3)&&(a2==a4)&&(a2!=a1)))
   cout << "среди этих чисел ровно 3 одинаковые\n";
```

Вложение else if

можно составлять так называемые **else-if**-конструкции, которые могут осуществить проверку сразу нескольких выражений, не используя сложные условия .

```
if (условие) инструкция;
else if (условие) инструкция;
     else if (условие) инструкция;
…                  else инструкция;
```

КАК ЭТО работает? Разобрать решение квадратного уравнения на практике.

Инструкция выбора switch

Синтаксис:

```
switch (целочисленное_выражение)
{
     case константа1: инструкции;
     case константа2: инструкции;
     ...
     case константаN: инструкции;
     [default: инструкция;]
}
```

Схема выполнения оператора **switch** следующая:
- вычисляется выражение в круглых скобках;
- вычисленное значение последовательно сравнивается с константными выражениями, следующими за ключевыми словами **case**;
- если одно из константных выражений совпадает со значением выражения, то управление передается на инструкцию, помеченную соответствующим ключевым словом **case**;
- если ни одно из константных выражений не равно выражению, то управление передается на инструкцию, помеченную ключевым словом **default**, а в случае его отсутствия управление передается на следующую после **switch** инструкцию;
- после выполнения инструкций выбранного **case** начинают выполняться инструкции следующего **case** и т. д. , если не было выхода по **break**;
-для того, чтобы выполнить одни и те же действия для различных значений выражения, можно пометить одну и ту же инструкцию несколькими ключевыми словами **case**.

Использование оператора **break** позволяет в необходимый момент прервать последовательность выполняемых инструкций в теле **switch**, путем передачи управления инструкции, следующей за **switch**.

В теле **switch** можно использовать вложенные инструкции **switch**, при этом в ключевых словах **case** можно использовать одинаковые константные выражения.

Следует обратить особое внимание на то, что выполнение инструкции **switch** отличается от аналогичного оператора **case** в Паскале.

- если одна ветвь **switch** состоит из нескольких инструкций, то их не нужно заключать в блок (в Паскале – нужно).

- Во-вторых, после завершения работы одной ветви в C++ начинает выполняться следующая ветвь (в Паскале происходит выход из оператора). Для того, чтобы обеспечить завершение работы инструкции **switch**, необходимо записать специальную инструкцию **break** в конце ветви:

```
switch (выражение)
{
 case константа_1: инструкции; break;
 case константа_2: инструкции; break;
 ...
 case константа_N: инструкции; break;
 [default: инструкция;]
}
```

Сравним Паскаль и C++

```
case n of
   0: writeln('ноль');
   1: writeln('один');
   2: writeln('два');
   3: writeln('три');
else writeln('много');
end;
```

```
switch (n)
{
   case 0: cout << "ноль\n"; break;
   case 1: cout << "один\n"; break;
   case 2: cout << "два\n"; break;
   case 3: cout << "три\n"; break;
   default: cout << "много\n";
}
```

```
int a=2, b=0;
switch(a)
{
    case 0:
    case 1: b++;
    case 2: b+=2;
    case 3: b+=3;
    default: a=0;
}
```

```
//b=5;

// простейший калькулятор
int a, b, y;
char  c;
switch(c)
{
  case '+': y=a+b; break;
  case '-': y=a-b; break;
  case '*': y=a*b; break;
  case '/': y=a/b; break;
 default : cout<<"Неверный знак операции";
}

switch(c)
{
  case '+': case '-': y=a-b+a; break ;
  case '*': case '/': y=a/b*a;
}
```

Пример вложенных инструкций:

```
switch (a)
{
     case 1: b=c; break;
     case 2:
            switch (d)
            {   case 0:  f=s;  break;
                case 1:  f=9;  break;
                case 2:  f-=9; break;
            }
     case 3: b-=c; break;
}
```

Организация циклов

Инструкция while (цикл с предусловием)

Формат:

```
while (выражение)
     инструкция;  //тело цикла
```

В качестве выражения допускается использовать любое выражение языка Си, а в качестве инструкции (тела цикла) любую инструкцию, в том числе пустую

или составную.

Схема выполнения цикла while следующая:

1. Вычисляется выражение.

2. Если выражение ложно, то выполнение оператора **while** заканчивается и выполняется следующая за циклом инструкция. Если выражение истинно, то выполняется тело цикла.

3. Процесс повторяется с пункта 1.

Таким образом, цикл выполняется до тех пор, пока выражение истинно.

Если в теле цикла нужно выполнить несколько инструкций, то используется блок {…}.

Для того, чтобы цикл когда-нибудь завершился, выражение должно стать ложным, иначе будет бесконечный цикл.

Так как в цикле while вначале происходит проверка условия, то его удобно использовать в ситуациях, когда тело цикла может не выполниться ни разу.

```
// сумма квадратов первых 10 чисел
#include <iostream>
int main()
{
   int s,i;
   s=0;
   i=1;
   while (i<=10)
   {
     s=s+i*i;
     i++;
   }
   std::cout << "summa=" << s << std::endl;
   return 0;
}
```

Пример бесконечного цикла

```
while (true)
   {
   ...
   }
```

Инструкция do…while (цикл с постусловием)

Формат:

```
do
  инструкция;
while (выражение);
```

используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз.

Схема выполнения цикла **do while** :

1. Выполняется тело цикла (которое может быть блоком {…}).
2. Вычисляется выражение.
3. Если выражение ложно, то выполнение цикла **do while** заканчивается и выполняется следующая за ним инструкция. Если выражение истинно, то выполнение продолжается с пункта 1.

Пример бесконечного цикла

```
do
{
   ...
}
while (true);
```

Циклы **while** и **do while** могут быть вложенными.

Инструкция for

Оператор for - это наиболее общий способ организации цикла.

Формат:

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

```
int x;
for (x=0;x<100;x++)
     std::cout << x << std::endl;
```

Схема выполнения цикла for:

1. Вычисляется выражение1.
2. Вычисляется выражение2.
3. Если значения выражения2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение3 и осуществляется переход к пункту 2, если выражение2 равно нулю (ложь), то управление передается на инструкцию,

следующую за циклом for.

Таким образом,

выражение1 вычисляется один раз перед началом цикла;

выражение3 вычисляется после каждой операции цикла;

условие (выражение2) проверяется каждый раз перед выполнением инструкции.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

Пример:

```
 int main()
 {
   int i,s=0;
   for (i=1; i<11; i++)
       s=s+i*i;
   std::cout<<"summa=”<<s<<
        std::endl;
   return 0;
  }
```

В этом примере вычисляется сумма квадратов чисел от 1 до 10.

Сумма первых десяти натуральных чисел

Для вычисления можно написать три различных цикла for.

1)

```
int count,s=0;
for (count=1; count < =10; count++)
    s=s+count;
```

2)

```
int count=10, s=0;
 for (; count>0; count--)
   s=s+count;
```

3)

```
int count=10,int s=0;
for (; count; count--)
   s=s+count;
```

цикл закончит выполнение при count=0, т.к. условие цикла будет ложным.

Некоторые варианты использования оператора for повышают его гибкость за счет возможности использования нескольких переменных, управляющих циклом.

Пример:

```
 int main()
 {
     int top,  bot;
      char string[100],  temp;
      for ( top=0, bot=99 ; top < bot ; top++, bot-- )
      {  temp=string[top];
          string[top]= string[bot];
          string[bot]=temp;
      }
      return 0;
  }
```

В этом примере, реализующем запись строки символов в обратном порядке, для управления циклом используются две переменные top и bot. Отметим, что на месте выражение1 и выражение3 здесь используются несколько выражений, записанных через запятую, и выполняемых последовательно.

Бесконечный цикл for

Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют дополнительное условие и оператор **break**.

```
for (;;)
{ ...
  ...
  break;
  ...
}
```

так как согласно синтаксису языка Си оператор может быть пустым, тело оператора for также может быть пустым. Такая форма оператора может быть использована для организации поиска.

```
for (i=0; t[i]<10 ; i++);
```

В данном примере переменная цикла i принимает значение номера первого элемента массива t, значение которого больше 10.

Инструкции передачи управления

```
goto;
break;
continue;
return;
```

Инструкция break

Прекращает работу блоков switch и циклов.

```
int a=-2, b=1;
while (a)
{
     a++;
     b--;
     if (!b)
          break;
}
//a=-1;
```

После выполнения оператора break управление передается оператору, следующему за прерванным. При вложенных циклах – работает с самым внутренним из объемлющих его циклов.

Инструкция continue

Переход к следующей итерации цикла. Завершает текущую итерацию цикла и передает управление на вычисление условия выполнения цикла.

Формат :

```
continue;
```

При вложенных циклах – работает с самым внутренним из объемлющих его циклов.

```
int a=-2, b=1;
while (a)
{
     a++;
     if (!(a+b))
          continue;
     b--;
}// a=0, b=0

int a, b;
for (a=1,b=0;a<10;b+=a,a++)
{
     if (b%2)
          continue;
     std::cout << b << std::endl;
}
```

Упр. Определить результат работы этого фрагмента программы.

Инструкция goto

Формат:

```
goto имя-метки;
...
имя-метки: инструкция;
```

Выполняет безусловный переход к другой инструкции внутри того же файла.

Оператор **goto** передает управление на оператор, помеченный меткой имя-метки. Помеченный оператор должен находиться в той же функции, что и оператор goto, а используемая метка должна быть уникальной, т.е. одно имя-метки не может быть использовано для разных операторов программы.

Использование оператора безусловного перехода **goto** в практике программирования настоятельно не рекомендуется, так как он затрудняет понимание программ и возможность их модификаций.

Любой оператор в составном операторе может иметь свою метку. Используя оператор **goto**, теоретически можно передавать управление внутрь составного оператора. Но нужно быть осторожным при входе в составной оператор, содержащий объявления переменных с инициализацией, так как объявления располагаются перед выполняемыми операторами и значения объявленных переменных при таком переходе будут не определены.

И вообще – такой переход – нарушение принципов структурного программирования.

Не следует передавать управление внутрь операторов **if, switch и циклов**.

Это тоже – нарушение принципов структурного программирования.

Инструкция `return`

завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию, в точку, непосредственно следующую за вызовом. Функция **main** передает управление операционной системе.

Формат :

```
return [выражение] ;
```

Значение выражения, если оно задано, возвращается в вызывающую функцию в качестве значения вызываемой функции. Если выражение опущено, то возвращаемое значение не определено. Выражение может быть заключено в круглые скобки, хотя их наличие не обязательно.

Таким образом, использование оператора return необходимо либо для немедленного выхода из функции, либо для передачи возвращаемого

значения.

Пример:

```
int sum (int a, int b)
{
   return (a+b);
}
```

Функция sum имеет два формальных параметра a и b типа int, и возвращает значение типа int. Возвращаемое оператором return значение равно сумме фактических параметров.

Пример:

```
void prov (int a, double b)
{  double c;
   if (a<3) return;
   else
    if (b>10) return;
    else
    { c=a+b;
      if ((2*c-b)==11) return;                    }
}
```

В примере оператор return используется для выхода из функции в случае выполнения одного из проверяемых условий.

```
#include <iostream>
using namespace std;
int main()
{
   // простейший калькулятор
   setlocale(LC_ALL, ".1251");
   int a, b, y;
   char c;
   cout<<"Введите операнды"<<endl;
   cin >> a >> b;
   cout << "Введите знак операции"<< endl;
   cin>>c;
   switch(c)
   {
     case '+': y=a+b; break;
     case '-': y=a-b; break;
     case '*': y=a*b; break;
     case '/': y=a/b; break;
```

```
    default :
      cout << "Неверный знак операции" << endl; return 0;
   }
   cout << "результат = " << y << endl;
   return 0;
}
```

Стили выравнивания

1)
```
for (a=1,b=0;a<10;b+=a,a++)
{
    if (b%2)
        continue;
    std::cout << b << std::endl;
}
```
2)
```
for (a=1,b=0;a<10;b+=a,a++)
    {
        if (b%2)
            continue;
        std::cout << b << std::endl;
    }
```
3)
```
for (a=1,b=0;a<10;b+=a,a++){
    if (b%2)
        continue;
    std::cout << b << std::endl;
}
```

Ответ к упражнению:

Печать четных сумм чисел от 1 до 9:

0
6
10
28
36

КОНЕЦ ЛЕКЦИИ