ЛЕКЦИЯ 4.

Вопросы:

- Операторы языка С++.
- Выражения.
- Приведение типов.

Операторы языка С++

разделяются по количеству операндов на унарные, бинарные и один тернарный (три операнда) оператор. Унарные операторы, кроме того, разделяются по расположению знаков относительно операнда на префиксные и постфиксные.

Термину «оператор языка С» соответствует термин «операция языка Паскаль», а вместо привычного «оператор языка» будут применяться слова «инструкция языка». Это связано с проблемами, возникающими при переводе английских слов ***operator*** и ***statement***.

Понятие L-выражений (L-values)

Некоторые операторы (присваивания, декремент, инкремент и т.д.) изменяют значение одного из своих операндов. Не всякое выражение может служить изменяемым операндом таких операций. На этом месте должна стоять конструкция, однозначно адресующая некоторый участок памяти (например, имя переменной). Подобного рода выражения получили **название L-выражений (L-values).**

Арифметические операторы

- вычитание и унарный минус: -x, x-y
+ сложение и унарный плюс: +x, x+y
***** умножение: x*y
/ деление: x/y
% получение остатка (деление по модулю) : x%y
++ инкремент: ++x, x++ (x = x+1)
-- декремент: --x, x-- (x = x-1)

Операторы - + * / применимы к целочисленным и вещественным типам данных. Если оператор деления (**/**) применяется к целому числу (или символу), дробная часть отбрасывается.

Оператор деления по модулю (**%**) возвращает остаток целочисленного деления. Его нельзя применять к вещественным числам (числам с плавающей точкой).

Унарный минус меняет знак своего операнда на противоположный (умножает свой операнд на -1).

Унарный плюс не меняет знак своего операнда.

```
int a=3,b=11,c=0,d;
d = b % a; // результат: 2
d = a % b; // результат: 3
d = b % c; // результат: сообщение об ошибке
d = b / a; // результат: 3
d = a / b; // результат: 0
d = b / c; // результат: сообщение об ошибке
float a=3, b=11, c=0, d;
d = b % a; // сообщение об ошибке
d = b % c; // сообщение об ошибке
d = b / a; // результат: 3.666667
d = a / b; // результат: 0.272728
d = b / c; // сообщение об ошибке

int a=3,b=11;
float x=3;
cout << (b/a) << endl << (b/x);

результат:
 3
 3.666667
```

Операторы инкремента и декремента

Унарные операторы **(++)** и **(--)** (инкремент и декремент) изменяют на единицу свой операнд. **Префиксные** (до) операторы указывают, что сначала увеличить (или уменьшить) значение переменной, а затем использовать это значение.

Постфиксные (после) указывают, что сначала использовать значение переменной, а затем увеличить (или уменьшить) его.

В программе вместо строки

```
x + 1;
```

можно записать строку

```
x++;
```

или

```
++x;
```

В подобной ситуации, когда оператор ++ является единственным в выражении, не имеет значения место его расположения: до имени переменной или после. Ее значение в любом случае увеличится на единицу.

```
int i = 3, j, k = 0;
```

Теперь проанализируем, как изменятся значения переменных в следующих выражениях (предполагается, что они выполняются не

последовательно, а по отдельности):

```
k = ++i; // i = 4, k = 4
k = i++; // i = 4, k = 3
k = --i; // i = 2, k = 2
k = i--; // i = 2, k = 3
i = j = k--; // i = 0, j = 0, k = -1

int x=5;
--x+x   // значение выражения=8, x=4
x+x--   // значение выражения=10, x=4
```

Побочные эффекты

```
int c, d, t = 3;
c = ++t+1;       // c=5, t=4
d = t++ + t++;  // не определен
```

Чтобы избежать ошибок

💣 Не применяйте операции увеличения или уменьшения к переменной, присутствующей в более чем одном аргументе функции.

💣 Не применяйте операции увеличения или уменьшения к переменной, которая входит в выражение более одного раза.

Порядок вычисления операторов

( )	слева направо
- (унарный)	слева направо
*** / %**	слева направо
+ - (вычитание)	слева направо
=	справа налево

Побитовые операторы

Побитовые операторы выполняются над целыми числами: над каждым битом операндов. Результатом является целое число.

~ - отрицание: ~x

& - побитовое “и” (and)

| - побитовое “или” (or)

^ - побитовое “исключающее или” (xor)

a	b	AND a & b	OR a | b	XOR a ^ b
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Оператор ~ меняет в битовом представлении операнда 0 на 1, а 1 – на 0.
Например:

```
unsigned short y = 0xFFFF;  //ffff
y = ~y;
cout << hex << y << endl;   //0
```

```
 short int a=0x45ff;
~ a;    //  -0x3a00
//0100 0101 1111 1111
//1011 1010 0000 0000
//-0x3a00
```

Оператор & сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если оба бита равны 1, то соответствующий бит результата равен 1, иначе 0.
Оператор & часто используется для выделения некоторого множества битов.

```
234567 & 255
```

В этом примере выделяется последний байт числа.
Упр. Чему он равен?

С помощью операции & можно определить остаток от деления операнда типа **unsigned int** на 2, 4, 8, 16 и т.д.

Для этого достаточно применить операцию **&** к делимому с масками **0x01, 0x03, 0x07, 0x0f, 0x1f** и т.д.
Например:
127&0x03 дает **3 (остаток от деления 127 на 4).**

```
127=0x7f=01111111
01111111&11 // 00000011=3
```

Оператор | сравнивает каждый бит 1-го операнда с соответствующим

битом 2-го операнда; если хотя бы один из них равен 1, то соответствующий бит результата равен 1, иначе 0.

Операция **|** используется для включения битов:

x | 7 устанавливает в 1 три последних бита переменной x (число 7 =00000111_2), остальные биты переменной не изменяются.

Оператор ^ сравнивает каждый бит первого операнда с соответствующим битом второго операнда; если значения битов одинаковое, то соответствующий бит результата устанавливается в 0, если разное –бит результата устанавливается в 1.

Оператор ^ часто используется для обнуления переменных:

```
n=n^n;  // значение переменной n будет равно 0
```

Используя этот оператор можно обменять значения двух переменных, не используя третью:

```
x=x^y;
y=x^y;
x=x^y;
```

Упр. Разобрать след. примеры:

```
short int a=0x45ff, b=0x00ff;
~ b;    //  -0x7f00
a | b   //  0x45ff
a & b   //  0x00ff
a ^ b   //  0x4500
```

Операторы сдвига

Оператор << (сдвиг влево) выполняет побитовый сдвиг влево левого операнда на количество разрядов, соответствующее значению правого операнда. Сдвиг на 1 бит эквивалентен умножению на 2. Результатом является целое число.

Оператор >> (сдвиг вправо): выполняет побитовый сдвиг вправо левого операнда на количество разрядов, соответствующее значению правого операнда. Если число без знака, то левые биты = 0. Сдвиг на 1 бит вправо эквивалентен делению на 2. Результатом является целое число.

Сдвиг вправо может быть арифметическим (т. е. освобождающиеся слева разряды заполняются значениями знакового разряда) или логическим в зависимости от реализации, однако гарантируется, что при сдвиге вправо целых чисел без знака освобождающиеся слева разряды будут заполняться нулями.

Побитовый сдвиг влево или вправо эквиалентен умножению и делению на степени 2:

```
unsigned short m=2000;
m = m>>4;
//2000₁₀ =0x7D0=111 1101 0000₂
//0111 1101 0000;
//0111 1101=0x7D=7*16+13=125
//2000:2⁴ =2000:16=125

int m1, m2, m=2000;
m2=m<<4;  // 32000 (2000*2⁴)
m1=~m;    // -2001

2000₁₀ =0x7D0=111 1101 0000₂
2000₁₀ &4 =111 1101 0000₂ &100₂=0
2000₁₀ |4 =111 1101 0000₂ |100₂=2004₁₀
2000₁₀ ^4 =111 1101 0000₂ ^100₂=2004₁₀
```

Операторы присваивания

В C++ представлены как простой оператор присваивания, так и составные операторы присваивания, совмещающие выполнение арифметической операции или побитовой операции и присваивания.

Простой оператор присваивания

обозначается символом **=** (знак равенства),
имеет два операнда:

операнд1 = операнд2

В результате ***операнду1*** присваивается значение ***операнда2*** и вырабатывается значение, равное значению, полученному первым операндом.

Операнд2 может быть выражением.

Оператор присваивания в правой части выражения

Оператор присваивания (=) не обязан стоять в отдельной строке и может входить в более крупные выражения:

```
int v1,v2;
v1 = 8 * (v2 = 5);//v1=40, v2=5
```

```
int i=0; double x=0;
x = ( i = 5.25) + 6.5;
// i=5  x=11.5 (5+6.5)
```

Множественные присваивания

В C++ допустимо множественное присваивание

```
i=j=k;    x=y=z=0;
```

Множественное присваивание - это отнюдь не присваивание нескольким переменным одного и того же значения. Множественное присваивание работает по тем же правилам, как любое использование оператора присваивания в правой части выражения.

Например:

```
int i=0; float x=0, y=0;
bool b=false;
x = b = i = y = 4.567;
// y=4.567, i=4, b=1, x=1.0
```

и

```
float x1=0, y1=4.5, z1=3.2;
x1 = (y1 = z1) = 6.5;
// z1=3.2, y1=6.5,  x1=6.5
```

Составные операторы присваивания

Формат:

переменная оператор выражение

Имеется 10 составных операторов, каждый из которых связан с арифметической или побитовой операцией, требующей двух операндов.

Алгоритм выполнения составного оператора:

- взять значение первого (левого) операнда;
- выполнить требуемую операцию, используя в качестве первого операнда это значение левого операнда, а в качестве второго - выражение в правой части оператора присваивания;
- присвоить полученное значение пременной в левой части оператора присваивания.

Арифметические составные операторы присваивания

+= Сложение с замещением

-= Вычитание с замещением

***=** Умножение с замещением

/= Деление с замещением

%= Деление по модулю с замещением

Например:

x+=1; //x=x+1;

y*=z; //y=y*z;

Побитовые составные операторы присваивания

&= Побитовое И с замещением

|= Побитовое ИЛИ с замещением

^= Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ с замещением

>>= Сдвиг вправо с замещением

<<= Сдвиг влево с замещением

Например**:**

unsigned short y = 0x00FF;

cout << hex << y << endl; ***// ff***

y &= 0xAA80;

cout << hex << y << endl; ***// 80***

Преобразование типов при присваивании

При выполнении оператора присваивания, когда типы операндов не совпадают, значение правой части преобразовывается к типу левой части. Правила такого преобразования приведены в следующей таблице:

lvalue =	rvalue	Возможные потери
char, signed char	unsigned char	Если rvalue > 127, результатом будет отрицательное число
unsigned char	char, signed char	Если rvalue < 0, результатом будет положительное число
char	short, short int	Старшие 8 бит
char	int, long int	Старшие 24 бит
short, short int	int, long int	Старшие 16 бит
short, short int, int, long int	float, double	Дробная часть и, возможно, что-то еще
float	double	Точность; возможны переполнение или потеря порядка

Операторы сравнения

Сравнивать можно операнды любого типа, но
либо они должны быть оба одного и того же встроенного типа,
либо между ними должна быть определена соответствующая операция

сравнения,

сравнение на равенство и неравенство может быть для двух величин любого типа.

Результат сравнения – логическое значение **true** или **false**.

Операторы сравнения и логические операторы не изменяют значения своих операндов, а только вычисляют значение ложь или истина.

Любое значение, не равное нулю, считается истиной, равное нулю – ложью.

Поэтому вместо сравнения величины с нулем

```
if (a != 0)
```

можно использовать

```
if (a)
```

Пример:

```
int i=2, j=-3, k=5, l=1;
char c1='S', c2='U';
bool b;
b = i > 3; cout << b << endl; // 0
b = i >=3; cout << b << endl; // 0
b = i < 3; cout << b << endl; // 1
b = k !=2; cout << b << endl; // 1
b = j == i - k;
cout << b << endl;  // 1
b = 'U' < 'S';
cout << b << endl;  // 0
b = l == (k > i);
cout << b << endl;  // 1
```

Логические операторы

&& логическое “и” (конъюнкции)

|| логическое “или” (дизъюнкции)

! логическое “НЕ “(отрицание)

Операнды - логические значения, результат – тоже логическое значение.

Примеры:

(x < y) && (x > y) // =0

(x < y) || (x > =y) // =1

! (1 < 2) // =0

a	b	a && b	a || b	!a
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Условный оператор

Имеет вид:

условие ? выражение1 : выражение2

Его результат равен значению выражения1, если условие истинно, и значению выражения2 в противном случае. Это тернарная операция: первый операнд должен быть логическим значением, второй и третий операнды могут быть любого, но одного и того же, типа, а результат будет того же типа, что и третий операнд.

Использование этого оператора позволяет изящно записать несложные сравнения, например: **s = a>0 ? a : 0;**

Переменная s получит значение переменной **a**, если **a>0** и значение **0** в противном случае.

```
int a = 1,  b = 0;
c = (a>b)  ?  2  :  3;     // c = 2
```

Что вычисляется в следующем операторе?

```
X = (X >=  0.0)  ?  X  :  -X;
```

```
#include <iostream>
using namespace std;
int main()
{
   setlocale(LC_ALL,".1251");
   float balance, payment;
   cout << "введите сумму заема: ";
   cin >> balance;
   cout << "\nведите сумму погашения: ";
   cin >> payment;
   cout << "\nваш баланс: ";
   cout << ((payment>balance) ? "переплачено на $" :
```

```
        "уплачено $");
   cout << ((payment>balance) ? payment - balance :
        payment);
   cout << " по заему на сумму $" << balance << ".\n";
   return 0;
}
```

Оператор последовательного вычисления (запятая)

Служит для перечисления нескольких выражений в одной инструкции. Выражения вычисляются слева направо. Значением всего выражения является значение последнего из выражений.

Синтаксис:

выражение, ***выражение***[,…,***выражение***];

Оператор запятая имеет самый низкий приоритет.

```
int i=0;
float  x=0, y=0;
x=(y=3, y+1);
  // переменная x получит значение 4
x=(y=3.5, i=2, x=y+i, i=x);
  // переменная x получит значение 5
    int n=2,m=3;
    int z;
    z=(m++,m+n); // z=6
```

или

```
    m++,z=m+n;   // z=6
```

но

```
    z=m++,m+n;   // z=3
```

Оператор запятая используется обычно там, где по синтаксису должно быть одно выражение

```
    for (I=1,J=5; I<=10;I++,--J )
```

Оператор измерения размера

имеет две формы:

sizeof *выражение* и

sizeof(*тип*)

В первом случае вычисляется размер для значения выражения, во втором – размер любой переменной заданного типа. Рассмотрим примеры (предполагается, что переменная t имеет тип **int,** а переменная f – тип **float**).

```
int t;
float f;
```

```
t = sizeof(int);
//результат для Microsoft Visual C++: 4
t = sizeof(float); // результат: 4
t = sizeof f;         // результат: 4
t = sizeof(f+0.1);
/* результат: 8, т.к. константа имеет  тип double, скобки
   нужны для задания приоритета */
```

Остальные операторы будут рассмотрены позднее.

Выражения

В C++ выражением считается любая допустимая комбинация операторов, констант, функций и переменных.

Выражение, после которого стоит точка с запятой – это ***инструкция-выражение***. Ее смысл состоит в том, что компьютер должен выполнить все действия, записанные в данном выражении, иначе говоря, вычислить выражение.

Примеры выражений:

```
(x + y) – 12*sin(c)
Sqrt(c*c – a*a)
Abs(s1–s2)/(3*x+7*y*z)–1/(sqr(a)+sqr(b))
Sin(x*x)+cos(2*pi–alpha)
```

Приоритет операций

Приоритетом операции называется очередность выполнения операции в выражении, в котором должно быть выполнено несколько операций. Так как в C++ операции задаются с помощью операторов, можно говорить о приоритете операторов. Далее приведены основные операторы C++, сгруппированные в порядке убывания приоритетов:

разрешение области действия **::**

индексация **[ ]**
вызов функции и подвыражение **( )**
доступ к члену структуры **.**
доступ к члену структуры через указатель **->**
префиксный инкремент **++**
префиксный декремент **--**

выделение памяти **new**
освобождение памяти **delete**
освобождение памяти из-под массива **delete[ ]**
разыменование *****
взятие адреса **&**
унарный **+**
унарный **-**
логическое отрицание **!**
поразрядное логическое НЕ **~**
получение размера **sizeof**
получение типа **typeid**
преобразование типа: **(type), const_cast, dynamic_cast, reinterpret_cast, static_cast**

умножение *****
деление ***/***
остаток от деления **%**

сложение **+**
вычитание **–**

поразрядный сдвиг влево **<<**
поразрядный сдвиг вправо **>>**

меньше **<**
больше **>**
меньше или равно **<=**
больше или равно **>=**

равно **==**
не равно **!=**

поразрядное логическое И **&**

поразрядное исключающее ИЛИ **^**

поразрядное логическое ИЛИ **|**

логическое И **&&**

логическое ИЛИ **||**

тернарный условный оператор **? :**

присваивание **=**
составные операторы присваивания
= /= %= += -= <<= >>= &= ^=

операция последовательного вычисления запятая **,**
постфиксный инкремент **++**
постфиксный декремент **--**

Сокращенный вариант:

Приоритет операторов

1. Вычисления в круглых скобках
2. Вычисления значений функций
3. Унарные операторы:
! -(изменение знака) +
4. Операторы типа умножение: * / %
5. Операторы типа сложения: + -
6. Операторы сдвига: << >>
7. Операторы отношения:
< <= > >= = = !=
8. Оператор &
9. Оператор ^
10.Оператор |
11.Оператор &&
12.Оператор ||
13.Условный оператор ? :
14. Оператор ,

Порядок вычисления операторов

В С++ все операторы выполняются (имеют ассоциативность) слева направо, кроме операторов:
разрешения области действия **::**
выделение памяти **new**
освобождение памяти **delete**
освобождение памяти из-под массива **delete[ ]**
разыменование ***

взятие адреса **&**
унарный **+**
унарный **–**
логическое отрицание **!**
поразрядное логическое НЕ **~**
получение размера **sizeof**
получение типа **typeid**
преобразование типа (type) **const_cast dynamic_cast reinterpret_cast static_cast**

Эти операторы имеют асссоциативность справа налево.

Приведение типов в выражениях

Если в выражение входят константы и перемненные различных типов, они последовательно, по ходу выполнения вычислений, преобразуются к одному, "покрывающему" типу (типу с наибольшим диапазоном значений.). Последовательность типов в C++ от "покрывающих" к "покрываемым":

Тип данных	старшинство
long double (в MS VS совпадает с double)	**Высший**
double	
float	
unsigned long (в MS VS совпадает с unsigned int)	
long (в MS VS совпадает с int)	
unsigned int	
int	
short (short int)	
unsigned char	
char	
bool	**Низший**

Такое приведение типов называется неявным. Стиль приведения на С поощряет неявное преобразование типов данных.
Любое арифметическое выражение, включающее операнды, типы которых не превышают int, перед вычислением всегда преобразовывается в тип int.

Для явного преобразования типа выражения используется оператор преобразования типа:

Синтаксис:

(тип) выражение

где в скобках – один из простых типов данных.

```
(int)(1.5 / 0.3);// тип int
(float)(1.5 / 0.3);   // тип float
```

В выражении приоритет преобразования типов данных приравнивается к приоритету унарных операторов.