Тема 1.3 Операции в С

- Преобразование типов явное и неявное.
- Операции арифметического типа. Преобразование типа в операции присвоения.
- Операции логического типа. Одноместные и двуместные операции.
- Адресные операции. Указатели в адресных операциях.

Операции в С

Над объектами в языке Си могут выполняться различные операции:

- операции присваивания;
- операции отношения;
- арифметические;
- логические;
- сдвиговые операции.

Результатом выполнения операции является число.

Операции могут быть бинарными или унарными. ***Бинарные*** операции выполняются над двумя объектами, ***унарные*** — над одним.

Приоритеты (ранги) операций

Ранг	Операции	Ассоциативность
1	0 [ ] -> .	→
2	! ~ + _ ++ — & * *(тип)* sizeof	←
3	* / % (мультипликативные бинарные)	→
4	+ - (аддитивные бинарные)	→
5	<< >> (поразрядного сдвига)	→
6	< <= >= > (отношения)	→
7	= != (отношения)	→
8	& (поразрядная конъюнкция "И")	→
9	^ (поразрядное исключающее "ИЛИ")	→
10	| (поразрядная дизъюнкция "ИЛИ")	→
11	&& (конъюнкция "И")	→
12	|| (дизъюнкция "ИЛИ")	→
13	?: (условная операция)	←
14	= *= /= %= += -= &= ^= |= <<= >>=	←
15	, (операция "запятая")	→

Приоритеты (ранги) операций

Операции ***ранга 1*** имеют наивысший приоритет. Операции одного ранга имеют одинаковый приоритет, и если их в выражении несколько, то они выполняются в соответствии с правилом ассоциативности либо ***слева направо*** (->), либо ***справа налево*** (<-). Если один и тот же знак операции приведен в таблице дважды (например, знак *), то первое появление (с меньшим по номеру, т.е. старшим по приоритету, рангом) соответствует унарной операции, а второе - бинарной.

Основные бинарные операции, расположенные в порядке уменьшения приоритета:

- умножение *****;
- деление ***/***;
- сложение **+**;
- вычитание **-**;
- остаток от целочисленного деления **%**.

Основные унарные операции:

- инкрементирование (увеличение на 1) **++**;
- декрементирование (уменьшение на 1) **--**;
- изменение знака **-**.

! Результат вычисления выражения, содержащего операции инкрементирования или декрементирования, зависит от того, где расположен знак операции (до объекта или после него):

Пример:

```
int a=2;
int b=3;
int c;
c = a*++b; // c=8, поскольку в
операции умножения уже b=4

int a=2;
int b=3;
int d;
d = a*b++; // d=6, поскольку в операции умножения b=3,
следующим действием будет b=4
```

Операции отношения:

Основные операции отношения:

- == эквивалентно — проверка на равенство;
- != не равно — проверка на неравенство;
- < меньше;
- > больше;
- <=меньше или равно;
- >= больше или равно.

! Операции отношения используются при организации условий и ветвлений. Результатом этих операций является 1 бит, значение которого равно 1, если результат выполнения операции - истина, и равно 0, если результат выполнения операции - ложь.

Логические операции

Логические операции делятся на две группы:

- условные;
- побитовые.

Условные логические операции

- Основные условные логические операции:
- **&&** - И (бинарная) — требуется одновременное выполнение всех операций отношения;
- **||** - ИЛИ (бинарная) — требуется выполнение хотя бы одной операции отношения;
- ! - НЕ (унарная) — требуется невыполнение операции отношения.

Условные логические операции чаще всего используются в операциях проверки условия if и могут выполняться над любыми объектами. Результат условной логической операции:

- 1 если выражение истинно;
- 0 если выражение ложно.

! Все значения, отличные от нуля, интерпретируются условными логическими операциями как истинные.

Побитовые логические операции

- Основные побитовые логические операции в языке Си:
- **&** конъюнкция (логическое И) - бинарная операция, результат которой равен 1 только когда оба операнда единичны (в общем случае - когда все операнды единичны);
- **|** дизъюнкция (логическое ИЛИ) - бинарная операция, результат которой равен 1 когда хотя бы один из операндов равен 1;
- **~** инверсия (логическое НЕ) - унарная операция, результат которой равен 0 если операнд единичный, и равен 1, если операнд нулевой;
- **^** исключающее ИЛИ - бинарная операция, результат которой равен 1, если только один из двух операндов равен 1 (в общем случае если во входном наборе операндов нечетное число единиц).

! ***Побитовые*** логические операции оперируют с битами, каждый из которых может принимать только два значения: 0 или 1.

Для каждого бита результат выполнения операции будет получен в соответствии с таблицей:

a	b	a & b	a | b	~a	a ^ b
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Пример:

```
unsigned char a = 14;     // a = 0000 1110
unsigned char b = 9;      // b = 0000 1001
unsigned char c, d, e, f;
c = a & b;                // c = 8 = 0000 1000
d = a | b;                // d = 15 = 0000 1111
e = ~a;                   // e = 241 = 1111 0001
f = a ^ b;                // f = 7 = 0000 0111
```

Побитовые операции позволяют осуществлять установку и сброс отдельных битов числа. С этой целью используется ***маскирование битов***. Маски, соответствующие установке ка

Бит	Маска
0	0x01
1	0x02
2	0x04
3	0x08
4	0x10
5	0x20
6	0x40
7	0x80

Для установки определенного бита необходимо соответствующий бит маски установить в 1 и произвести операцию побитового логического **ИЛИ** с константой, представляющей собой маску:

```
unsigned char a = 3;
a = a | 0x04;  // a = 7, бит 2 установлен
```

Для сброса определенного бита необходимо соответствующий бит маски сбросить в 0 и произвести операцию побитового логического **И** с константой, представляющей собой инверсную маску:

```
unsigned char a = 3;
a = a & (~0x02);  // a = 1, бит 1 сброшен
```

Объединение операций

! Бинарные арифметические операции могут быть объединены с операцией присваивания:

- объект *= выражение; // объект = объект * выражение
- объект /= выражение; // объект = объект / выражение
- объект += выражение; // объект = объект + выражение
- объект -= выражение; // объект = объект - выражение
- объект %= выражение; // объект = объект % выражение

! Бинарные побитовые логические операции могут быть объединены с операцией присваивания:

- объект &= выражение; // объект = объект & выражение
- объект |= выражение; // объект = объект | выражение
- объект ^= выражение; // объект = объект ^ выражение

Сдвиговые операции

Операции арифметического сдвига применяются в целочисленной арифметике и обозначаются как:

>> - сдвиг вправо;

<< - сдвиг влево.

Общий синтаксис осуществления операции сдвига:

объект = выражение сдвиг КоличествоРазрядов;

Пример:

```
unsigned char a=6;  // a = 0000 0110
unsigned char b;
b = a >> 1; // b = 0000 0110 >> 1 = 0000 0011 = 3
```

! Арифметический сдвиг целого числа вправо **>>** на 1 разряд соответствует делению числа на 2.
Арифметический сдвиг целого числа влево **<<** на 1 разряд соответствует умножению числа на 2.

Преобразование типов явное и неявное

- При явном приведении перед выражением следует указать в круглых скобках имя типа, к которому необходимо преобразовать исходное значение.
- При неявном приведении преобразование происходит автоматически, по правилам, заложенным в языке С.

ПРИМЕР ЯВНОГО ПРИВЕДЕНИЯ ТИПА

```
int x = 5;
double y = 15.3;
x = (int) y;
y = (double) x;
```

ПРИМЕР НЕЯВНОГО ПРИВЕДЕНИЯ ТИПА

```
int x = 5;
double y = 15.3;
y = x; //здесь происходит неявное приведение типа к double
x = y; //здесь происходит неявное приведение типа к int
```

НЕЯВНОЕ ПРИВЕДЕНИЕ ТИПА ПРИ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЯХ

типы операндов	тип результата
float / float	float
float / int	float
int / float	float
int / int	int

! В последнем случае (и только в нем) осуществляется **целочисленное деление с отбрасыванием остатка**.

Зачем нужно явное приведение типов если есть неявное?

Пример 1:

```
int x = 12;
int y = 7;
double z = x/y;
```

! В данном выражении разбор начнется с операции наиболее высокого приоритета — с деления, и только потом дело дойдет до присваивания.

Т.е. операция с точки зрения типов будет такой: **z = (double)((int)x / (int)y)**

Итого, **z = 1.0**

Пример 2:

```
int x = 12;
int y = 7;
double z = (double)x/y;
```

! В данном выражении разбор начнется с операции наиболее высокого приоритета — с унарной операции приведения типов, и только потом дело дойдет до деления.

Т.е. операция с точки зрения типов будет такой: **z = (double)x / (int)y**

Итого, **z = 1.714285714285714**

Операция присваивания

Операция присваивания обозначается символом = и выполняется в 2 этапа:

вычисляется выражение в правой части;

результат присваивается операнду, стоящему в левой части:

объект = выражение;

```
объект = (тип)выражение;
float a = 241.5;
// Перед вычислением остатка от деления a приводится к целому типу
int b = (int)a % 2;  // b = 1
```

! В случае если объекты в левой и правой части операции присваивания имеют разные типы используется операция явного приведения типа.

Пример:

```
int a = 4; // переменной a присваивается значение 4
int b;
b = a + 2;   // переменной b присваивается значение 6,
             // вычисленное в правой части
```

Арифметические операции и указатели

С указателями можно выполнять следующие операции:

- разадресация, или косвенное обращение к объекту (*)
- присваивание,
- сложение с константой,
- вычитание,
- инкремент (++),
- декремент (—),
- сравнение,
- приведение типов.

Операция разадрееации, или разыменования, предназначена для доступа к величине, адрес которой хранится в указателе. Эту операцию можно использовать как для получения, так и для изменения значения величины (если она не объявлена как константа):

```
char a; // переменная типа char
char * p = new char; /* выделение памяти под указатель и под динамическую
переменную типа char */
*p = 'Ю'; a = *p; // присваивание значения обеим переменным
```

Операция присваивания

Указателю можно *присвоить адрес переменной*

p = &x, где***p***– указатель,***x***– имя переменной.

Указатели можно *присваивать один другому*, если оба указателя одного типа:

```
int x = 5; int* p = &x;
int* q = p;
cout<<*p<<' '<<*q<<endl; // 5 5
```

Сложение и ***вычитание*** указателей с константой **n** означает, что указатель перемещается по ячейкам памяти на столько байт, сколько занимает **n** переменных того типа, на который он указывает.

Допустим, что указатель имеет символьный тип и его значение равно **100**. Результат сложения этого указателя с единицей — **101**, так как для хранения переменной типа char требуется 1 байт. Если же значение указателя равно **100**, но он имеет целочисленный тип, то результат его сложения с единицей будет составлять

104, так как для переменной типа int отводится 4 байта.

```
double d,*p; int n;

p = &d;

n = 3;p = p +n; // в p адрес увеличился на24

p =n+ p; // в p адрес увеличился ещё на24
```

Вычитание указателей

При работе с указателями можно использовать *операцию вычитания* двух указателей:

имя_указателя1 – имя_указателя2

Операция имеет смысл, только если обе переменные являются указателями на один и тот же *набор*данных (например, массив). Результатом операции является целое число, которое показывает сколько элементов соответствующего типа можно расположить между адресами памяти, на которые указывают указатели *имя_указателя*1, *имя_указателя*2.

Инкремент (++)/Декремент (—)

Инкремент перемещает указатель к следующему элементу массива, *декремент* — к предыдущему. Значение указателя изменяется на величину sizeof (тип). Если указатель на определенный тип увеличивается или уменьшается на константу, его значение изменяется на величину этой константы, умноженную на размер объекта данного типа, **например:**

```
short * p = new short [5];
p++; // значение p увеличивается на 2
long * q = new long [5];
q++; // значение q увеличивается на 4
```

Операции сравнения

Указатели на данные одного и того же типа можно сравнивать с помощью обычных операций сравнения: **==, !=, >, >=, <, <=.**

При сравнении указателей сравниваются их значения (*хранимые в них адреса*), а не значения величин, на которые данные указатели указывают.

Результатом *операций сравнения* является целое число (**0**или**1**).

!Сравнение указателей с числовыми значениями как сравнение данных разных типов *не определено*.

Преобразование типа

Указатель на объект одного типа может быть преобразован в указатель на другой тип. При этом следует учитывать, что объект, адресуемый преобразованным указателем, будет интерпретироваться по-другому. Операция преобразования типа указателя применяется в виде **(<тип> *)<указатель>** :

```
int i, *ptr;
i = 0x8e41;
ptr = &i;
printf("%d\n", *ptr); // печатается значение int:-29119 (0x8e41)
printf("%d\n", *((char *)ptr)); // ptr преобразован к типу char,
                                // извлекается 1 байт и его двоичный код
                                // печатается в виде десятичного числа,
                                // печатается: 65 */
```

! Преобразование типа указателя чаще всего применяется для приведения указателя на неопределенный тип данных void к типу объекта, доступ к которому будет осуществляться через этот указатель.

Следующие операции недопустимы с указателями:

- сложение двух указателей;
- вычитание двух указателей на различные объекты;
- сложение указателей с числом с плавающей точкой;
- вычитание из указателей числа с плавающей точкой;
- умножение указателей;
- деление указателей;
- поразрядные операции и операции сдвига;