Лекция 4. Операции, выражения и операторы.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Операции в Си

Операции – в программировании, это способ выполнения какого-либо действия, которое имеет строгий синтаксис и строгую семантику.

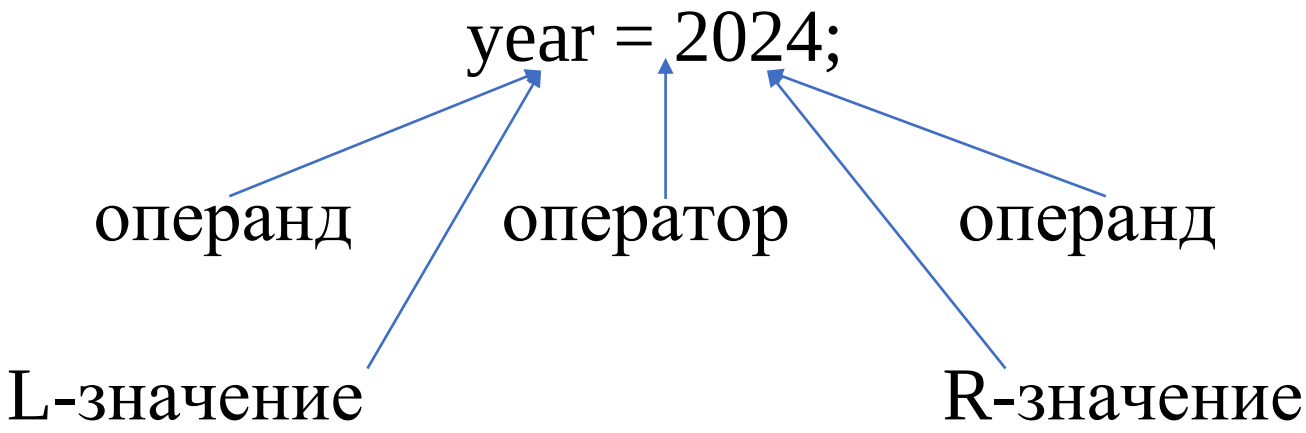
Операции делятся на:

- по смыслу
    1. Арифметические
    2. Логические

    И. пр. по типу действий который они выполняют… ~~(длиннный списоок)~~
- по количеству аргументов:
    1. Унарные
    2. Бинарные
    3. Тернарные

Операции, операнды, l и r значения

Переменная – с точки зрения смысла, есть **объект данных**, что позволяет выполнять любые действия с удержанным в ней значением.

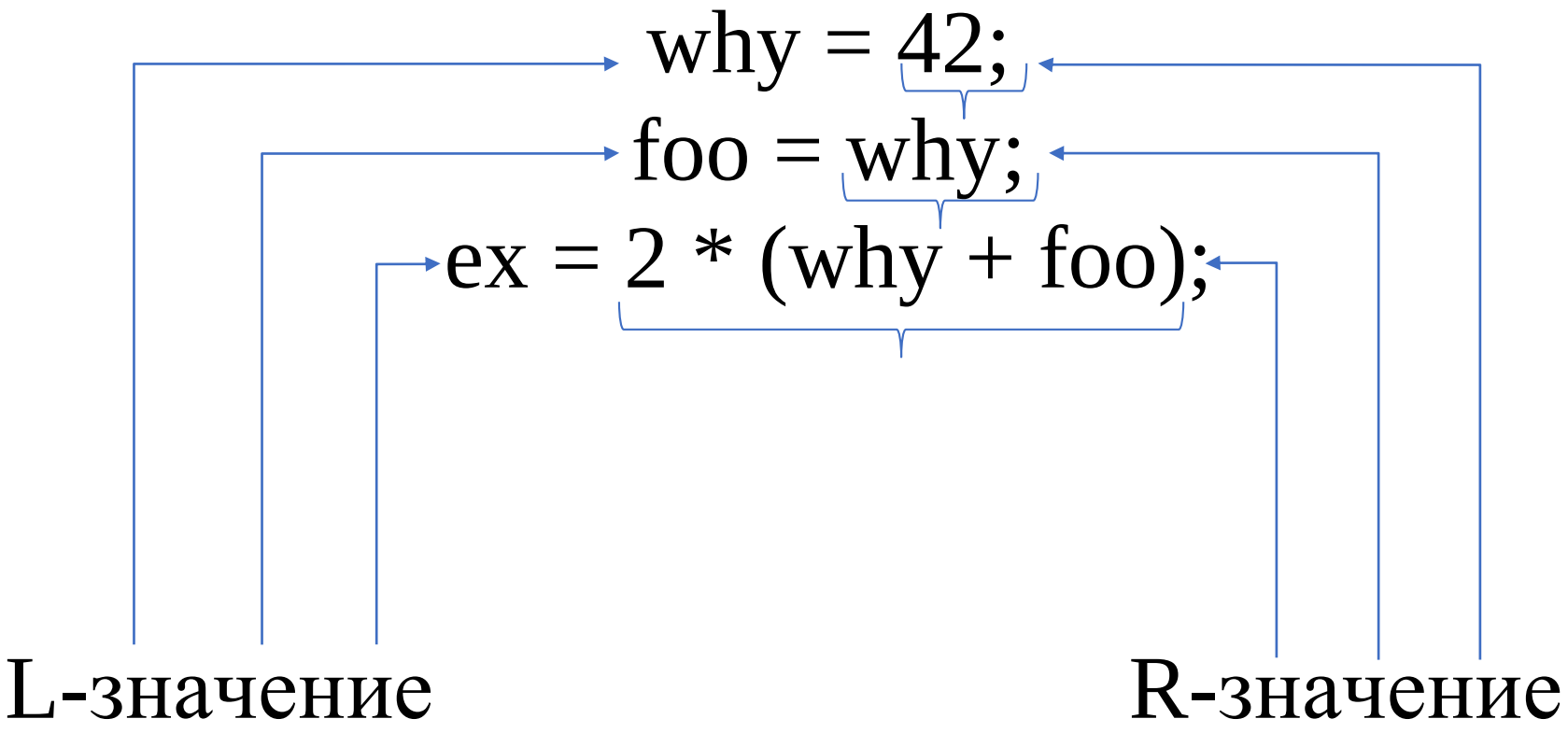


Операнд – один из аргументов операции.
L-значение – (от англ, left) левая часть, **ссылается** на адрес в памяти.
R-значение – (от англ, right) правая часть, величина которая может быть присвоена модифицируемым L-значениям. Иначе называют **выражением**.

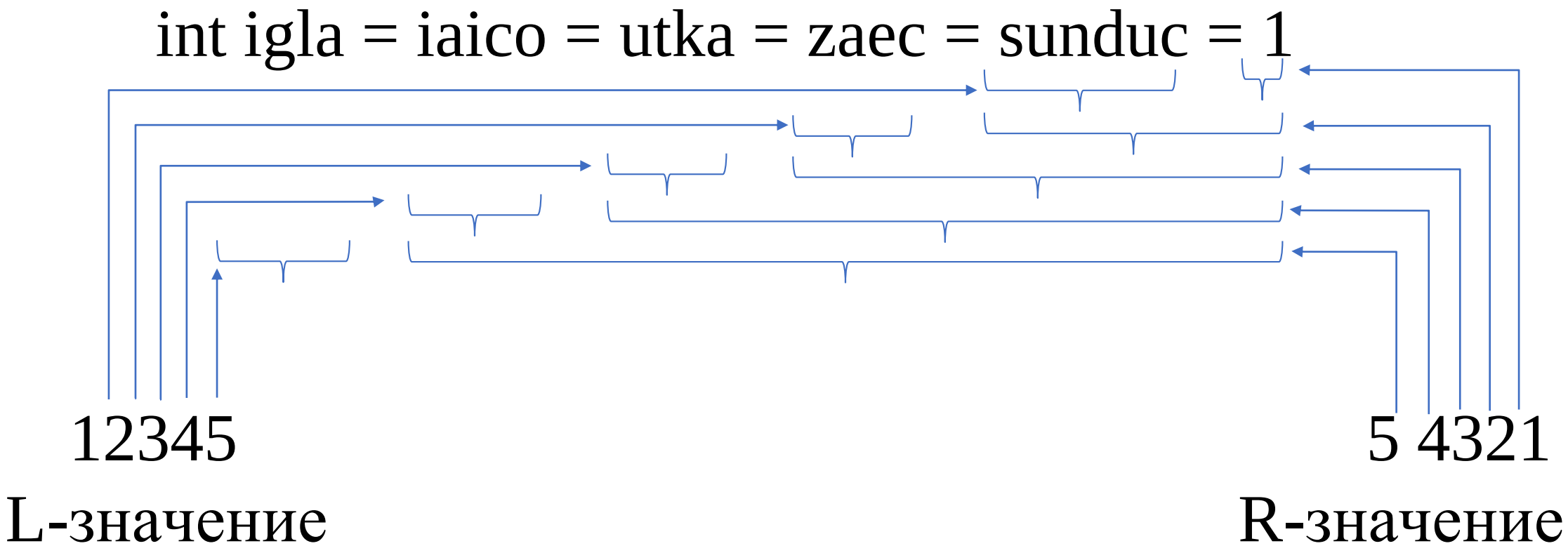
Оператор присваивания

```
int why, foo, ex;

why = 42;
foo = why;
ex = 2 * (why + foo);
```



lr-пары при присваивании



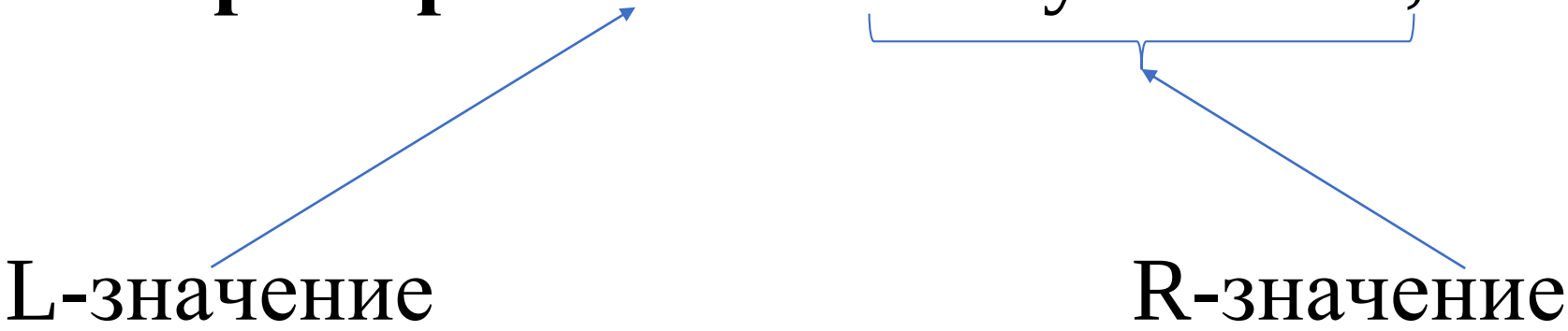
Операция сложения

Приводит к **сумме** двух значений с обеих сторон знака +.

Пример: printf("%d",41+1);

Результат выполнения программы: 42 , а не 41+1

Пример: income = salary + bribes;



Операция вычитания

Вызывает **вычитания** числа, следующего за знаком **и**, из числа находящегося перед этим знаком.

Пример: money = 20000 – 5000;

Операции называются **бинарными** или **двухместными** если они требуют указания двух **операндов.**

Пример: операции сложения и вычитания и пр.

Унарные операции сложения и вычитания

Операции называются **унарные** так как выполняется над одним **операндом.**

Оператор вычитания – может быть использован для **указания** или **изменения алгебраического знака** значения.

Пример: money = – 5000;
Пример: items = –(25 – 5);
Пример: best_temp = +30;

Операция умножения

Приводит к **произведению** двух значений с обеих сторон знака *.

Пример: cm = 2.54 * 27;

Оператор умножения используется для реализации функций **возведения в квадрат**.

~~Python: 2**10=1024~~

2*2*2*2*2*2*2*2*2*2-1024

Иначе требуется подключать библиотеку cmath.h

Операция деления

Значение слева от символа / **делится** на значение, указанное справа.

Пример: cm = 12.0 / 3.0;

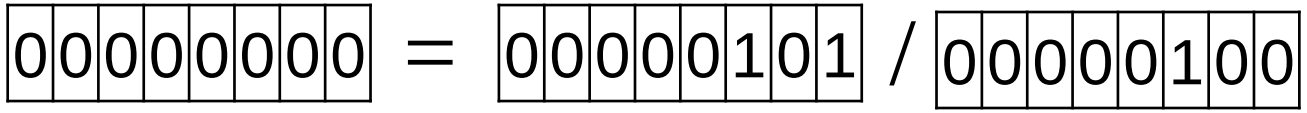
В зависимости от **типов значений** деление работает по разному.

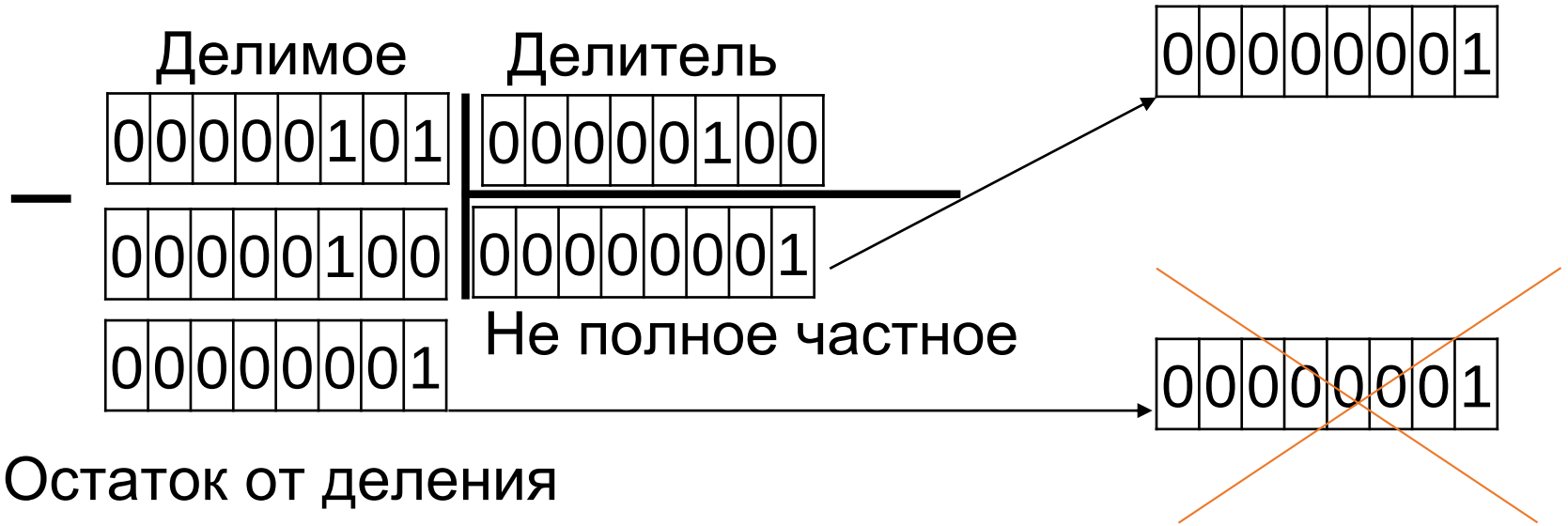
Операция деления целочисленные типы

Целочисленное деление **усекает** дробную часть.

Пример: char rounds = 0;

rounds = 5/4;





Операция деления вещественные типы

Вещественное деление работает как **стандартное** деление.

Пример:

```
printf("%f", 10 / 3); //
printf("%f", 3. / 10);
printf("%f", 5./5.);
```

Деление работает как **вещественное**, если один из операндов **вещественного** типа данных.

Приоритеты операций

Таблица 5.1. Операции в порядке снижения приоритета

Операции	Ассоциативность
( )	Слева направо
+ – (унарные)	Справа налево
* /	Слева направо
+ – (бинарные)	Слева направо
=	Справа налево

Приоритеты операций

```
#include <stdio.h>
int main(void)
{
    int top, score;

    top = score = -(2 + 5) * 6 + (4 + 3 * (2 + 3));
    printf("top = %d, score = %d\n", top, score);

    return 0;
}
```

Приоритеты операций

```
#include <stdio.h>
int main(void)
{
    int top, score;
    top = score = -(2 + 5) * 6 + (4 + 3 * (2 + 3));
    printf("top = %d, score = %d\n", top, score);

    return 0;
}
```

9 8 2 1 6 7 5 4 3

Операция sizeof

Операция **sizeof** возвращает размер своего операнда в byte. **Операндом** может быть конкретный **объект данных**, **имя переменной** или **тип данных.**

printf("%ul“, **sizeof** int); //4

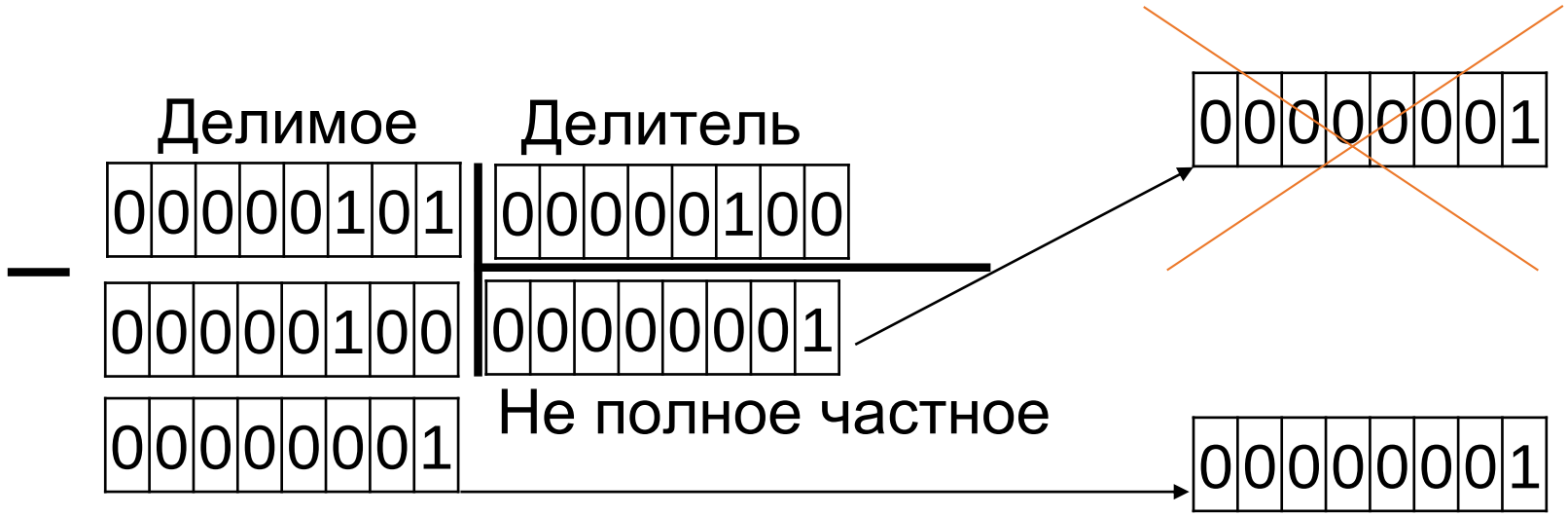
Результат выполнения **sizeof** это значение типа **size_t,** который представляет собой целочисленный тип без знака.

С точки зрения смысла, **size_t** это псевдоним **unsingned long**.

Операция деление по модулю

Операция **% (деление по модулю)** применима только лишь в целочисленной арифметике, а его результатом является **остаток от делителя**.

Пример: char rounds = 0;
rounds = 5%4;



Операция деление по модулю

Оператор % в Python

```
print(7%3+"\t"+-7%3+"\t"+7%-3+"\t"+-7%-3)
```

1 2 -2 -1

Оператор % в Си

```
printf("%d \t%d\t%d\t%d", 7%3, -7%3, 7%-3, -7%-3);
```

1 -1 1 -1

Операция инкремента и декрементирование

Increment – (с англ.) увеличение.

Оператор **инкремента** ++ решает простую задачу, увеличивает значение своего **операнда** на 1.

Существую две разновидности этой операции

Префиксная	i=0	Постфиксная
++i		i++
temp = ++i *10;		temp = i++*10;
	printf("%d",temp)	
10		0

Операция инкремента и декрементирование

Decrement – (с англ.) уменьшение.

Оператор **декрементирование --** решает простую задачу, уменьшает значение своего **операнда** на 1.

Существую две разновидности этой операции

Префиксная	i=10	Постфиксная
--i		i--
temp = --i *10;		temp = i--*10;
printf("%d",temp)		
90		100

Не стоит умничать

```
int num = 5;
пока(num < 21)
{
     printf("%10d %10d\n", num, num*num++);
}
```

Какой будет результат?

Кто отвечает за этот результат?

Кто принимает решение за функционирование программы?

Не стоит умничать

6	30
7	42
8	56
9	72
10	90
11	110
12	132
13	156
14	182
15	210
16	240
17	272
18	306
19	342
20	380
21	420

Чем проще код, тем он понятнее и тем легче его сопровождать. (с)

Выражение и операторы

Выражение – комбинация операций и операндов.

Выражение	Значение
`-4 + 6`	2
`c = 3 + 8`	11
`5 > 3`	1
`6 + (c = 3 + 8)`	17

Операторы – основные действия выполняемые программой.

Программа – последовательность операторов с необходимыми знаками пунктуации.

Примеры операторов

```
1)      1;              //Оператор выражения
2)      2+1;            //Оператор выражения
3)      x = 25;         //Оператор присваивания
4)      ++x;            //Оператор инкремента
5)      x = sqrt(x);    //Оператор присваивания
6)      ;               //Пустой оператор
7)      return 0;       //Оператор возврата
```

Преобразования типов

Ниже описаны базовые правила преобразования типов данных.

1. Находясь в выражении, типы `char` и `short` (как `signed`, так и `unsigned`) автоматически преобразуются в `int` или при необходимости в `unsigned int`. (Если тип `short` имеет такой же размер, как у `int`, то размер типа `unsigned short` больше, чем `int`; в этом случае `unsigned short` преобразуется в `unsigned int`.) В K&R C, но не в текущей версии языка тип `float` автоматически преобразуется в `double`. Поскольку они являются преобразованиями в большие по размеру типы, они называются *повышением*.

2. Если в любую операцию вовлечены два типа, оба значения приводятся к более высокому из этих двух типов.

3. Порядок типов от высшего к низшему выглядит так: `long double`, `double`, `float`, `unsigned long long`, `long long`, `unsigned long`, `long`, `unsigned int` и `int`. Возможно одно исключение, когда `long` и `int` имеют одинаковые размеры; в этом случае `unsigned int` превосходит `long`. Типы `short` и `char` в этом списке отсутствуют, т.к. они уже должны были повыситься до `int` или, возможно, до `unsigned int`.

Преобразования типов (продолжение)

4. В операторе присваивания финальный результат вычислений преобразуется к типу переменной, которой присваивается значение. Процесс может привести к повышению типа, как описано в правиле 1, или к *понижению* типа, при котором значение преобразуется в более низкий тип.
5. При передаче в качестве аргументов функции типы `char` и `short` преобразуются в `int`, а `float` — в `double`. Это автоматическое повышение переопределяется прототипированием функций,

Значение не помещается в целевой тип.

1. Когда целевым является одна из форм целочисленного типа без знака, а присвоенное значение представляет собой целое число, лишние биты, делающие значение слишком большим, игнорируются. Например, если целевой тип – 8-битный `unsigned char`, то присвоенным значением будет результат деления исходного значения по модулю 256.
2. Если целевым типом является целый тип со знаком, а присвоенное значение – целое число, то результат зависит от реализации.
3. Если целевой тип является целочисленным, а присвоенное значение представляет собой значение с плавающей запятой, то поведение не определено.

Операции приведения

Следует стараться избегать автоматического преобразования типов, особенно ситуаций *понижения*.

Операция приведения ***(тип_данных)*** – желаемый тип данных будет установлен операнду.

```
mice = 1.6 + 1.7;
mice = (int) 1.6 + (int) 1.7;
```

Функции с аргументами

```
/* pound.c -- определяет функцию с аргументом */
#include <stdio.h>
void pound(int n);          // объявление прототипа функции согласно стандарту ANSI
int main(void)
{
    int times = 5;
    char ch = '!';          // ASCII-код равен 33
    float f = 6.0f;
    pound(times);           // аргумент типа int
    pound(ch);              // эквивалентно pound((int)ch);
    pound(f);               // эквивалентно pound((int)f);
    return 0;
}

void pound(int n)           // заголовок функции в стиле ANSI, который указывает,
{                           // что функция принимает один аргумент int
    while (n-- > 0)
        printf("#");
    printf("\n");
}
```

Операции в C

Операция присваивания

`=` Присваивает переменной, указанной слева от знака операции, значение, заданное справа от него.

Арифметические операции

`+` Добавляет значение справа от знака операции к значению слева от знака.

`-` Вычитает значение справа от знака операции из значения слева от знака.

`-` Как унарная операция, изменяет знак значения, указанного справа от знака операции.

`*` Умножает значение справа от знака операции на значение слева от знака.

`/` Делит значение слева от знака операции на значение справа от знака.
Если оба операнда являются целочисленными, результат усекается.

`%` Выдает остаток от деления значения слева от знака на значение справа от знака (только для целочисленных значений).

`++` Добавляет 1 к значению переменной справа от знака операции (префиксная форма) или к значению слева от знака операции (постфиксная форма).

`--` Подобна `++`, но вычитает 1.

Операции различного назначения

`sizeof` Возвращает размер в байтах операнда, указанного справа. Операндом может быть спецификатор типа в круглых скобках, например, `sizeof (float)`, либо имя конкретной переменной, массива и т.д. без скобок, например, `sizeof foo`.

`(`*тип*`)` Как операция приведения, преобразует следующее за ней значение в тип, указанный внутри круглых скобок. Например, `(float) 9` преобразует целочисленное значение `9` в число с плавающей запятой `9.0`.

Задачи на практику

Читать: Главу 6. Операции, выражения и операторы

- Разобраться с тем как использовать while (примеров много)
- Разобрать листинг 5.14

Смотреть вопросы для самоконтроля (193 стр, page 183)
Упражнения по программированию (196-197 стр, page 187-188)

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)