

C Operators

Comparison, Arithmetic, Logical
Operators, Expressions



SoftUni Team
Technical Trainers
Software University
<http://softuni.bg>



Table of Contents

1. Operators in C# and Operator Precedence
2. Arithmetic Operators
3. Logical Operators
4. Comparison Operators
5. Assignment Operators
6. Other Operators
7. Implicit and Explicit Type Conversions
8. Expressions

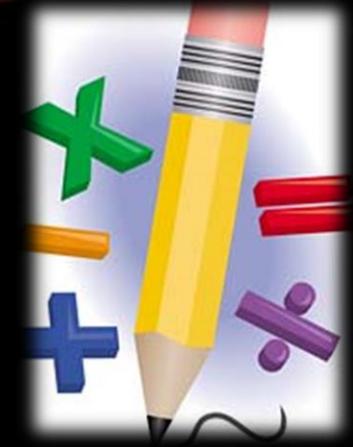


What is an Operator?

- Operator is an operation performed over data at runtime
 - Takes one or more arguments (operands)
 - Produces a new value
 - Example of operators:
`a = b + c;`
 - Operator "+"
 - Operator "="
- Operators have precedence
 - Precedence defines which will be evaluated first
 - Expressions are sequences of operators and operands that are evaluated to a single value, e.g. $(a + b) / 2$

Operators in C#

- Operators in C# :
 - Unary – take one operand
 - Binary – take two operands
 - Ternary (`? :`) – takes three operands
- Except for the assignment operators, all binary operators are left-associative
- The assignment operators and the conditional operator (`? :`) are right-associative



Operators in C

Arithmetic, Logical, Comparison,
Assignment, Etc.



Categories of Operators in C

Category	Operators
Arithmetic	+ - * / % ++ --
Logical	&& ^ !
Binary	& ^ ~ << >>
Comparison	== != < > <= >=
Assignment	= += -= *= /= %= &= = ^= <<= >>=
Type size	sizeof
Other	* & . -> ?:

Operators Precedence



Operators Precedence

Precedence	Operators
Highest	<code>()</code>
	<code>++ -- (postfix)</code>
	<code>++ -- (prefix) + - (unary) ! ~</code>
	<code>* / %</code>
	<code>+ -</code>
	<code><< >></code>
	<code>< > <= >= is as</code>
	<code>-- !=</code>
	<code>&</code>
Lower	<code>^</code>

Operators Precedence (2)

Precedence	Operators
Higher	
	&&
	? :
Lowest	= *= /= %= += -= <<= >>= &= ^= =

- Parenthesis operator always has highest precedence
- Note: prefer using **parentheses**, even when it seems stupid to do so

Arithmetic Operators



Arithmetic Operators

- Arithmetic operators **+**, **-**, ***** are the same as in math
- Division operator **/** if used on integers returns integer (without rounding) or exception
- Division operator **/** if used on real numbers returns real number or **inf** or **nan**
- Remainder operator **%** returns the remainder from division of integers
- The special addition operator **++** increments a variable

Arithmetic Operators – Examples

```
int squarePerimeter = 17;  
double squareSide = squarePerimeter / 4.0;  
double squareArea = squareSide * squareSide;  
printf("%f\n", squareSide); // 4.250000  
printf("%f\n", squareArea); // 18.062500
```

```
int a = 5;  
int b = 4;  
printf("%d\n", a + b); // 9  
printf("%d\n", a + b++); // 9  
printf("%d\n", a + b); // 10  
printf("%d\n", a + (++b)); // 11  
printf("%d\n", a + b); // 11
```

Arithmetic Operators – Example (2)

```
printf("%d\n", 12 / 3); // 4
printf("%d\n", 11 / 3); // 3
```

```
printf("%f\n", 11.0 / 3); // 3.666666667
printf("%f\n", 11 / 3.0); // 3.666666667
printf("%d\n", 11 % 3); // 2
printf("%d\n", 11 % -3); // 2
printf("%d\n", -11 % 3); // -2
```

```
printf("%f\n", 1.5 / 0.0); // inf
printf("%f\n", -1.5 / 0.0); // -inf
printf("%f\n", 0.0 / 0.0); // -nan
```

```
int x = 0;
printf("%f\n", 5 / x); // Floating point exception (core dumped)
```

Arithmetic Operators – Overflow Examples

```
#include <stdio.h>
#include <limits.h>

int main()
{
    int bigNum = 2000000000;
    int bigSum = 2 * bigNum; // Integer overflow!
    printf("%d\n", bigSum); // -294967296

    bigNum = INT_MAX;
    printf("%d\n", bigNum);
    bigNum = bigNum + 1;
    printf("%d\n", bigNum); // -2147483648

    return 0
}
```

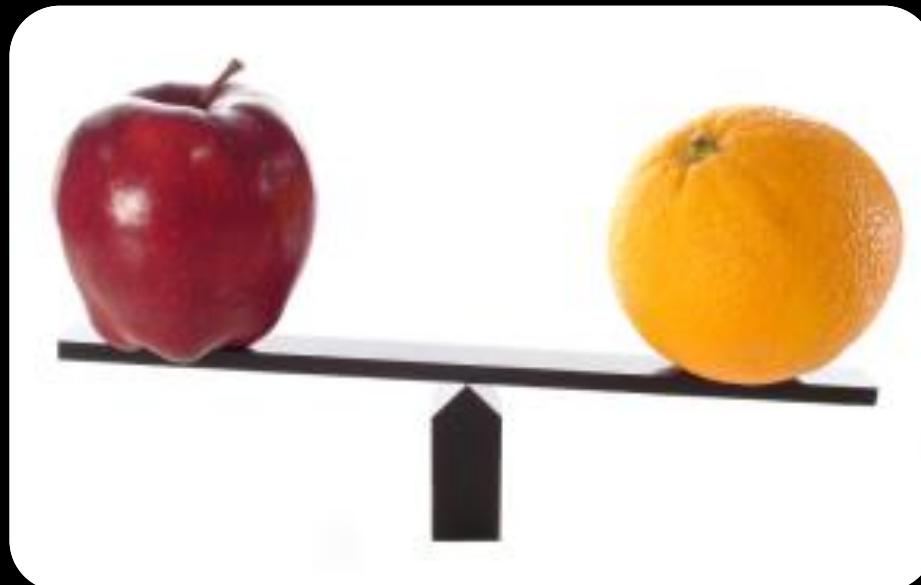
int varies according to
platform (2 or 4 bytes)

Arithmetic Operators

Live Demo



Comparison and Assignment Operators



Comparison Operators

- Comparison operators are used to compare variables
 - ==, <, >, >=, <=, !=
 - Return **1** if the comparison is true, **0** if it is false

```
int a = 5;
int b = 4;
int result = a >= b; // 1
result = a != b; // 1
result = a == b; // 0
result = a == a; // 1
result = a != ++b; // 0
result = a > b; // 0
```



Comparison and Assignment Operators

Live Demo



Logical Operators



Logical Operators

- In C all logical operators take integer operands and return integers as result (**1** → true, **0** → false)
- Operator **!** turns **1** to **0** and **0** to **1**
- Behavior of the operators **&&**, **||** and **^**
(1 == true**, **0 == false**):**

Operator					&&	&&	&&	&&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

Logical Operators – Example

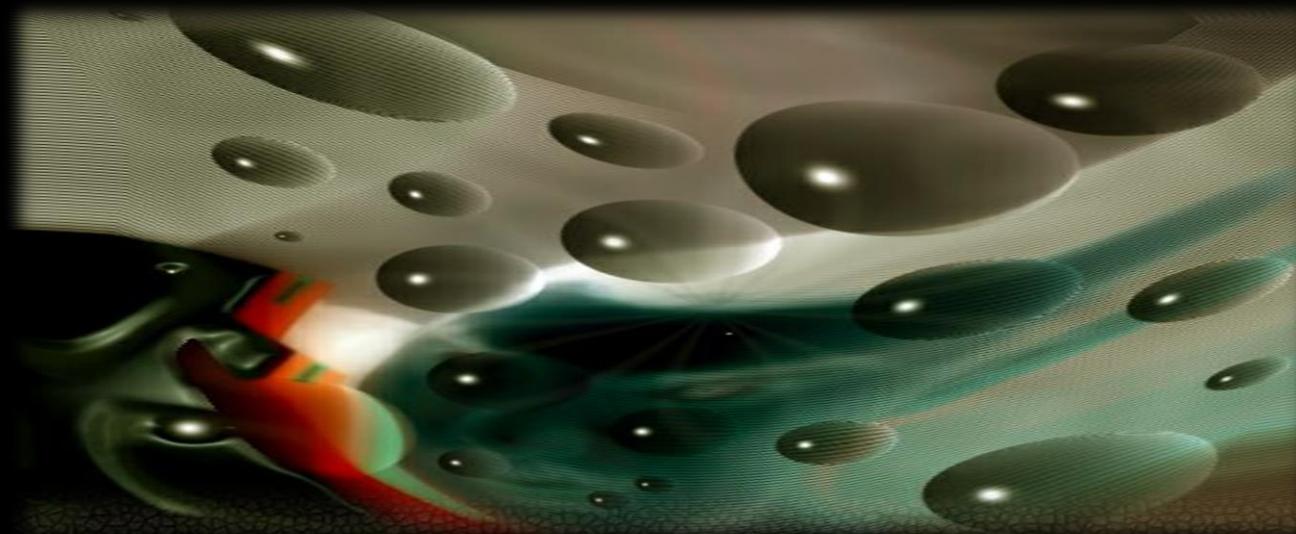
```
int a = 1;  
int b = 0;  
int result = a && b; // 0  
result = a || b; // 1  
result = a ^ b; // 1  
result = !b; // 1  
result = b || 1; // 1  
result = b && 1; // 0  
result = a || 1; // 1  
result = a && 1; // 1  
result = !a; // 0  
result = (5>7) ^ (a==b); // 0
```

Logical Operators

Live Demo



Other Operators



Assignment Operators

- Assignment operators are used to assign a value to a variable:
 - `=, +=, -=, |=, ...`
- Assignment operators example:

```
int x = 6;  
int y = 4;  
y *= 2; // 8  
int z = y = 3; // y=3 and z=3  
printf("%d\n", z); // 3  
x += 3; // 9  
x /= 2; // 4
```



Other Operators

- **sizeof()** – returns the size (in bytes) of the type

```
int intSize = sizeof(int);    // 4
int longSize = sizeof(int);   // 8
int charSize = sizeof(char); // 1
```

- Square brackets [] are used with arrays for accessing index

```
char name[5] = "Asya";
char firstLetter = name[0]; // Get character at 0th index
printf("%c", firstLetter); // A
printf("%c", name[3]);    // y
printf("%c", name[4]);    // (\0 character)
printf("%c", name[5]);    // Illegal access!
```

Other Operators (2)

- **{type} *** is used for declaring pointers

```
char *text = "Hey, pedestrian!";
char *pointer = text; // pointer points to the memory of text
```

- **&{identifier}** returns a pointer to a variable
- ***{identifier}** dereferences a pointer (i.e. returns the value)

```
int a = 5;
int* b = &a; // b now points to the memory of a

printf("%d", *b); // Returns the memory where b points to: 5
```

Other Operators (3)

■ Member access operators for **struct / union**:

- . when working with the variable

```
struct Point point;  
point.x = 5;  
point.y = 10;
```

```
struct Point  
{  
    int x;  
    int y;  
};
```

- -> when working with pointer to the variable

```
struct Point point;  
struct Point *ptr = &point;  
ptr->x = 2;  
ptr->y = -3;  
printf("X: %d, Y: %d\n", ptr->x, ptr->y);
```

Other Operators (4)

- Conditional operator ?: has the form

```
b ? x : y
```

(if **b** is true then the result is **x** else the result is **y**)

- Example:

```
printf(5 > 4 ? "true" : "false")
```



Other Operators

Live Demo



Implicit and Explicit Type Conversions



Implicit Type Conversion

- Implicit type conversion
 - C allows implicitly to convert from one type to another

```
int a = 2185313921;  
long b = a;  
unsigned short c = b;  
  
printf("%hu", c); // 16001
```

short is too small
and overflows

- Converting to a smaller type may cause data loss

Explicit Type Conversion

- Explicit type conversion
 - Manual conversion of a value of one data type to a value of another data type
 - Allowed only explicitly by **(type)** operator
 - Required when there is a possibility of loss of data or precision
 - Example:

```
float a = 5.5f;  
int b = (int) a; // 5
```

Type Conversions

Live Demo



Expressions



Expressions

- Expressions are sequences of operators, literals and variables that are evaluated to some value

```
#define MATH_PI 3.14159265358979323846

int main()
{
    int r = (150 - 20) / 2 + 5; // r=70
    // Expression for calculating a circle area
    double surface = MATH_PI * r * r;
    // Expression for calculating a circle perimeter
    double perimeter = 2 * MATH_PI * r;
    return 0;
}
```

Expressions (2)

- Expressions have:

- Type (integer, real, int, ...)

- Value

- Examples:

Expression of type **int**.

Evaluated at compile time

Expression of type **int**.

Evaluated at runtime

```
int a = 2 + 3; // a = 5
int b = (a + 3) * (a - 4) + (2 * a + 7) / 4; // b = 12
int greater = (a > b) || ((a == 0) && (b == 0));
```

Expression of type **int**.

Evaluated at runtime



Expressions

Live Demo

C Programming – Operators



Questions?

SUPERHOSTING.BG



License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license



- Attribution: this work may contain portions from
 - "Programming Basics" course by Software University under CC-BY-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University @ YouTube
 - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg

