Arithmetic Operators in C

Arithmetic operators in C are certain special symbols, predefined to perform arithmetic operations. We are familiar with the basic arithmetic operations − addition, subtraction, multiplication and division. C is a computational language, so these operators are essential in performing a computerised process.

In addition to the above operations assigned to the four symbols **+**, **−**, *****, and **/** respectively, C has another arithmetic operator called the **modulo operator** for which we use the **%**symbol.

The following table lists the arithmetic operators in C −

Operator	Description
+	Adds two operands.
−	Subtracts second operand from the first.
*	Multiplies both operands.
/	Divides numerator by denominator.
%	Modulus Operator and remainder of after an integer division.
++	Increment operator increases the integer value by one.
--	Decrement operator decreases the integer value by one.

The **++** and **--** operators are also listed in the above table. We shall learn about increment and decrement operators in a separate chapter.

Example: Arithmetic Operators in C

The following example demonstrates how to use these arithmetic operators in a C program −

Open Compiler

```
#include <stdio.h>

int main(){
   int op1 = 10;
   int op2 = 3;

   printf("Operand1: %d Operand2: %d \n\n", op1, op2);
   printf("Addition of op1 and op2: %d\n", op1 + op2);
   printf("Subtraction of op2 from op1: %d\n", op1 - op2);
   printf("Multiplication of op1 and op2: %d\n", op1 * op2);
   printf("Division of op1 by op2: %d\n", op1/op2);

   return 0;
}
```

Output

When you run this code, it will produce the following output −

```
Operand1: 10 Operand2: 3

Addition of op1 and op2: 13
Subtraction of op2 from op1: 7
Multiplication of op1 and op2: 30
Division of op1 by op2: 3
```

Type Casting in C

The first three results are as expected, but the result of division is not. You expect 10/3 to be a fractional number (3.333333). Is it because we used the **%d** format specifier to print the outcome of the division? If we change the last line of the code as follows −

```
printf("Division of op1 by op2: %f\n", op1/op2);
```

Now the outcome of the division operation will be "0.000000", which is even more surprising. The reason why C behaves like this is because the division of an integer with another integer always returns an integer.

To obtain floating-point division, at least one operand must be a float, or you need to use the typecast operator to change one of the integer operands to float.

Now, change the last printf statement of the given program as follows −

```
printf("Division of op1 by op2: %f\n", (float)op1/op2);
```

When you run run the code again after making this change, it will show the correct division −

```
Division of op1 by op2: 3.333333
```

Note: If you use **%d** format specifier for a floating-point expression, it will always result in "0".

Example

The result of arithmetic operations with at least one float (or double) operand is always float. Take a look at the following example −

Open Compiler

```
#include <stdio.h>

int main(){

   int op1 = 10;
   float op2 = 2.5;

   printf("Operand1: %d Operand2: %f\n", op1, op2);
   printf("Addition of op1 and op2: %f\n", op1 + op2);
   printf("Subtraction of op2 from op1: %f\n", op1 - op2);
   printf("Multiplication of op1 and op2: %f\n", op1 * op2);
   printf("Division of op1 by op2: %f\n", op1/op2);

   return 0;
}
```

Output

Run the code and check its output −

```
Operand1: 10 Operand2: 2.500000
Addition of op1 and op2: 12.500000
Subtraction of op2 from op1: 7.500000
Multiplication of op1 and op2: 25.000000
Division of op1 by op2: 4.000000
```

Explore our **latest online courses** and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Arithmetic Operations with char Data Type

In C, **char** data type is a subset of **int** type. Hence, we can perform arithmetic operations with **char** operands.

Example

The following example shows how you can perform arithmetic operations with two operands out of which one is a "char" type −

Open Compiler

```
#include <stdio.h>

int main(){

   char op1 = 'F';
   int op2 = 3;

   printf("operand1: %c operand2: %d\n", op1, op2);
   printf("Addition of op1 and op2: %d\n", op1 + op2);
   printf("Subtraction of op2 from op1: %d\n", op1 - op2);
   printf("Multiplication of op1 and op2: %d\n", op1 * op2);
   printf("Division of op1 by op2: %d\n", op1/op2);

   return 0;
}
```

Output

Run the code and check its output −

```
operand1: F operand2: 3

Addition of op1 and op2: 73
Subtraction of op2 from op1: 67
Multiplication of op1 and op2: 210
Division of op1 by op2: 23
```

Since a char data type is a subset of int, the **%c** format specifier returns the ASCII character associated with an integer returned by the **%d** specifier.

If any arithmetic operation between two char operands results in an integer beyond the range of char, the **%c** specifier displays blank.

Modulo Operator in C

The modulo operator (**%**) returns the remainder of a division operation.

Example

Take a look at the following example −

Open Compiler

```
#include <stdio.h>

int main(){

   int op1 = 10;
   int op2 = 3;

   printf("Operand1: %d Operand2: %d\n", op1, op2);
   printf("Modulo of op1 and op2: %d\n", op1%op2);

   return 0;
}
```

Output

Run the code and check its output −

```
Operand1: 10 Operand2: 3
Modulo of op1 and op2: 1
```

The modulo operator needs both the operands of int type. If not, the compiler gives a **type mismatch error**. For example, change the data type of "op1" to float in the above code and run the program again −

```
float op1 = 10;
int op2 = 3;
```

```
printf("Modulo of op1 and op2: %d\n", op1%op2);
```

Now, you will get a type mismatch error with the following message −

```
error: invalid operands to binary % (have 'float' and 'int')
```

It does allow char operands for modulo operations though.

Negation Operator in C

The increment and decrement operators represented by the symbols **++** and **--** are unary operators. They have been covered in a separate chapter. The "**−**" symbol, representing subtraction operator, also acts a unary negation operator.

Example

The following example highlights how you can use the negation operator in C −

Open Compiler

```
#include <stdio.h>

int main(){

   int op1 = 5;
   int op2 = -op1;

   printf("Operand1: %d Operand2: %d\n", op1, op2);

   return 0;
}
```

Output

When you run this code, it will produce the following output −

```
Operand1: 5 Operand2: -5
```

In the above example, the "−" symbol returns the negative value of op1 and assigns the same to op2.