Arithmetic Operators in C

CMSC 104 Section 02
February 26, 2024

Administrative Notes

From last Wednesday: the “fathoms” program

```
#include <stdio.h>

int main() {
float fathoms;// the depth in fathoms
float feet;// the depth in feet
float inches; //the depth in inches

/* Ask the user for the depth in fathoms */
printf(“Enter the depth in fathoms: “);
scanf(“%f”, &fathoms);
/* Convert depth from fathoms to inches */
feet = 6.0 * fathoms;
inches = 12.0 * feet;

/* Display The Results */
printf ( ”Its depth at sea:\n” );
printf( ”\t%1.2f fathoms\n” , fathoms);
printf( ”\t%1.2f feet\n”, feet);
printf (”\t%1.2f inches\n”, inches);
return 0 ; }
```

We had you start doing math in C

- Now we’re going to explain how it works

C supports two basic types of mathematics

Integer math

- all operands are integers
- All results are integers

Floating point math

- Operands can be integers or floating point numbers
- All results are floating point numbers

The basic operations are similar, but the differences are important

Arithmetic Operators

Name	Operator	Example
Addition	+	first_integer+second_integer
Subtraction	-	initial_amount - spent
Multiplication	*	fathoms*6
Division	/	sum/count
Modulus	%	m%n - gives the remainder

Note that blank spaces in these expressions don’t matter. You can use no spaces, one space, or more - whatever makes the code more readable

Rules for operations

Addition:

- Int + int = int
- Int + float = float
- Float + float = float

Subtraction:

- Int - int = int
- Int - float or float - int = float
- Float - float = float

Rules for operations

Multiplication

- Int * int = int
- Int * float = float
- Float * float = float

Rules for division

Division is a little different - you have to be careful

```
z = x/y;
```

If both x and y are ints, C will perform the integer division, even if z is declared to be a float

Integer division

In English:

The quotient is the result of integer division

Now that you're in college, "remainder" is called "modulus"

```
dividend/divisor = quotient + remainder
```

Where all of the above are whole numbers (integers)

5/2 = 2; 5%2 = 1

A very quick demonstration

```
#include <stdio.h>

int main() {

  /*  first we'll show straight up integer division */
  int dividend = 9;
  int divisor = 2;
  int quotient;
  int modulus;

  quotient = dividend/divisor;
  modulus = dividend%divisor;
  printf("THe quotient is %d\n\n",quotient);
  printf("The modulus is %d\n\n", modulus);

  // What if we told the system that quotient should be a float
  float fquotient;
  fquotient = dividend/divisor;
  printf("When quotient is a floating point the answer is %1.4f\n\n", fquotient);

  // but what if at least one of the operands is a float?
  float f_dividend = 9.0;
  fquotient = f_dividend/divisor;
  printf("With at least one floating point operand you get %1.4f\n\n", fquotient);


}
```

A word on modulus

Modulus ONLY works for integer operands

- If you try to use % and one of the operands is a float, it won’t work. You’ll get an error message.

Modulus

The expression m%n yields the integer remainder after m is divided by n.

Examples:

- 17 % 5 = 2
- 6 % 3 = 0
- 9 % 2 = 1
- 5 % 8 = 5

Uses for Modulus

Used to determine if an integer is even or odd:

- 5 % 2 == 1 /* odd */
- 4 % 2 == 0 /* even */

Modulus by 2 of an integer results in a 1 if odd, or zero if even.

Used to see if some number is divisible by another.

- 25 % 5 == 0 /* divisible by 5 */
- 49 % 7 == 0 /* divisible by 7 */

Operator Precedence

C works like standard arithmetic. Operations are evaluated in a specific order:

- Step 1: anything inside parentheses comes first.
  - If there are multiple sets of parentheses, go left to right
  - If there are nested parentheses, evaluate the innermost expression first
- Step 2: then evaluate all *, /, and % operations
  - If there are multiple, go left to right
- Step 3: then evaluate all + and - operations
  - If there are multiple, go left to right
- Step 4: then evaluate all equality operations (the = sign)
  - If there are multiple, go right to left

Good Programming Practice

- It’s best not to take the “big bang” approach to coding.
- Use an incremental approach by writing your code in incomplete, yet working, pieces. For example, for your projects:
  - ▸ Don’t write the whole program at once.
  - ▸ Just write enough to display the user prompt on the screen.
  - ▸ Get that part working first (compile and run).
  - ▸ Next, write the part that gets the value from the user, and just print it out.
  - ▸ Get that part working (compile and run).
  - ▸ Next, change the code so you use the value in a calculation. Print answer.
  - ▸ Get that part working (compile and run).
  - ▸ Repeat until the requirements are complete.
- This also helps you find any semantic errors sooner.
- Always have a working version of your program!