C Programming

Comments and Documentation

Comments are used to insert an explanation for a piece of code

There are different ways to display a comment

1. **Single-line comment**

Using two forward-slash characters together, aka, a single-line comment

```
e.g., // This is a comment and is ignored by the compiler
// there is no corresponding double-forward slash at the end
```

2. **Multi-line comment**

To use a multi-line comment, you use the following characters

```
/*
    Write your multi-line
    comment between the /* and the corresponding
    */
    ……
    ……
*/
```

Data Types (continued)

Integers - int
Floating points (Floats) - float
Characters - char

Integer data types

Type	Size	Range
short	2 bytes	-32,000 → 32,000

e.g., **short** num1; delimiter: %sd		
int e.g., **int** num1; delimiter: %d	4 bytes	-2 billion → 2 billion
long e.g., **long** num1; delimiter: %ld	8 bytes	-Big number → Big number

Float data types

Type	**Size**	**Range**
float e.g., **float** num1; delimiter: %f	4 bytes	-2 billion → 2 billion
double e.g., **double** num1; delimiter: %lf	8 bytes	-Big number → Big number

Type	**Size**	**Range**
char e.g., **char** my_char; delimiter: %c	1 byte	n/a
boolean e.g., **bool(ean)** num1; delimiter: %b	1 byte	true or false 1 or 0

an unsigned variable

This takes the negative range of a numeric variable and places it in the positive range

e.g., **unsigned** int num1; // range: 0 → 4 billion

Use %**u**d for int, %**u**f for float, etc., as the delimiters for unsigned variables

Arithmetic Operations and Expressions

```
int num1;
int num2;
int num3;

// assign 20 to num1
num1 = 20;

// copy contents of num1 into num3 and then into num2
num2 = num3 = num1;
```

Arithmetic Operators

Operator	**Meaning**
+	Addition
-	Subtraction
*	Multiplication
/	Division

%	Modulus
	this gives the remainder of a division
	e.g.,
	int var = 0;
	
	var = 11 % 3;

Example:

```
/*
Program to demonstrate arithmetic operations
*/

#include <stdio.h>

int main()
{
    int var1;
    int var2;

    var1 = 0;
```

```
    var2 = 10;

    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    // Let's do some arithmetic operations
    var2 = var1 + 10;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    var1 = var2 * 3;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    var2 = var1 - 1;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    var2 = var1 / 5;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    var2 = var1 % 20;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    return 0;
}
```

Repl 2.1: https://replit.com/@michaelTUDublin/21-Arithmetic-Operations#main.c

Increment and Decrement Operator

These are used to add / subtract the value 1 (and only 1) to a variable

```c
/*
Program to demonstrate increment and decrement operators
*/

#include <stdio.h>

int main()
{
    int var1;
    int var2;
    char my_char;

    var1 = 1;
    var2 = 2;
    my_char = 'a';

    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    // Increment var1, i.e., add 1 to the contents of var1
    //var1 = var1 + 1;
    var1++;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    // Decrement var2, i.e., subtract 1 from the contents of var2
    // var2 = var2 - 1;
    var2--;
    printf("var1 contains %d, var2 contains %d\n", var1, var2);

    // Redundant Code
    // var1 = var1 * 1;
    // var1**;
```

```
    //
    // var1 = var1 / 1;
    // var1//;

    // my_char++;
    // printf("my_char is %c", my_char);

    return 0;
}
```

Repl 2.2: https://replit.com/@michaelTUDublin/22-Increment-and-Decrement#main.c

Next: Have a look at these two programs:

```
/*
Increment and decrement operator
*/

#include <stdio.h>

int main(void)
{
    int var1;
    int var2;

    var1 = 1;
    var2 = 2;

    printf("Initial values\n");
    printf("var1 is %d, var2 is %d\n", var1, var2);

    //Increment var1
    var1++;

    //Decrement var2
    var2--;

    printf("Final values\n");
    printf("var1 is %d, var2 is %d\n", var1, var2);
```

```
    return 0;
}
```

Repl 2.3: https://replit.com/@michaelTUDublin/23-Increment-and-Decrement#main.c

Now, let's modify the above code so that it uses look at it uses **PRE**-Increment and **POST**-Increment

```
/*
PRE and POST increment and decrement
*/

#include <stdio.h>

int main(void)
{
    int var1;
    int var2;
    int var3;
    int var4;

    var1 = var2 = 1;

    // This is POST-Increment
    var3 = var1++;

    //This is PRE-Increment
    var4 = ++var2;

    printf("var1 is %d, var2 is %d\n", var1, var2);
    printf("var3 is %d, var4 is %d\n", var3, var4);

    return 0;
}
```

Repl 2.4: https://replit.com/@michaelTUDublin/24-Pre-and-Post#main.c

Be very careful if you use PRE or POST increment/decrement and know the difference.

Continued below …

Operator Precedence

Look at this:

```
int num1 = 0;
num1 = 4 + 6 / 2;
printf("num1 contains %d", num1);
```

Question: What is the content of num1? Is it 5 or 7?

Answer: The correct answer is 7. Why? Because division (/) has a higher precedence than addition (+)

Most programming languages, including C, have a mathematical operator precedence. This is shown in the following Table

Operator	Precedence	Meaning	Order
Braces (and)	Highest	Braces place highest precedence	

-	Higher	Unary minus	e.g.,
			`int a = 10;`
			`int b = -a * 2;`
			`// b = -10`

* / %	High	Multiply Divide Modulus	Left to Right
+ -	Low	Add Subtract	Left to Right

Any mathematical expression that uses operators with the same precedence, then C will execute those operators from Left to Right

e.g.,

```
int num1 = 0;


num1 = 4 * 6 / 2;
printf("num1 contains %d", num1);
```

Above, the answer is 12.

However, you can override the above code and force the division to execute first by using braces.

e.g.,

```
int num1 = 0;


num1 = 4 * 6 / 2;
printf("num1 contains %d", num1);
```

Type conversion and Casting

It is important that you do not use mixed expressions in an arithmetic operation.

Let's have a look at the following code example:

```
/*
Program to demonstrate mixed data type expressions
*/
#include <stdio.h>

int main()
{
    int var1 = 13;
    float var2 = 2.5;
    float var3;
    int var4;
    float var5;

    // Mixed data type expression
    var3 = var1 / var2;
    printf("var3 contains %f\n", var3);
```

```
    // Mixed data type expression
    var4 = var1 / var2;
    printf("var4 contains %d\n", var4);

    // casting - temporarily changing var1 to be a float
    var5 = (float)var1 / 4;
    printf("var5 contains %f", var5);

    return 0;
}
```

Repl 2.5: https://replit.com/@michaelTUDublin/25-Casting#main.c

Programming pitfalls

```
int main()
{
    int a, b;

    a = 2;
  b = 4;

  /*
    Redundant code - the OS will ignore the sum of these two numbers
and does nothing.
    */
  a + b;
}
```

Also, look at this:

```
/*
```

```
In this example, it is vital that you always initialise a variable
BEFORE you use it. If not, random data will be inside the variable
and may cause an incorrect value following the operation
*/
int main()
{
    int counter = 0;

    counter++;
    printf("counter is %d", counter);

}
```