Syntax

Every spoken language has a general set of rules for how words and sentences should be structured. These rules are collectively known as the language syntax. In computer programming, syntax serves the same purpose, defining how declarations, functions, commands, and other statements should be arranged.

Many computer programming languages share similar syntax rules, while others have a unique syntax design. For example, C and Java use a similar syntax, while Perl has many characteristics that are not seen in either the C or Java languages.

A program's source code must have correct syntax in order to compile correctly and be made into a program. In fact, it must have perfect syntax, or the program will fail to compile and produce a "syntax error." A syntax error can be as simple as a missing parenthesis or a forgotten semicolon at the end of a statement. Even these small errors will keep the source code from compiling.

Fortunately, most integrated development environments (IDEs) include a parser, which detects syntax errors within the source code. Modern parsers can even highlight syntax errors before a program is compiled, making it easy for the programmer to locate and fix them.

NOTE: Syntax errors are also called compile-time errors, since they can prevent a program from compiliing. Errors that occur in a program after it has been compiled are called runtime errors, since they occur when the program is running.

How to speak computing syntax

Strange punctuation is a "feature" of most computer commands. Occasionally, in order to describe a command with this punctuation, I'll have to pronounce it out loud. However, you may never have heard it pronounced this way before, so here's a quick guide.

!	"bang", "exclamation point"
@	"at", and rarely, "strudel"
#	"crunch", "hash", "pound", and rarely, "octothorpe"
^	"circumflex", "hat", "chapeau"
&	"ampersand", "and"
*	"splat", "star", "asterisk", ("times" for multiplication)
_	"underscore"
-	"hyphen", "dash", "minus sign"
.	"dot", "period"
,	"comma"
:	"colon"
;	"semi-colon"
/	"slash" or “forward slash”
\	"backslash"
~	"twiddle", also "squiggle", or more correctly, "tilde"
'	"tick", "quote", "apostrophe" (pronounced apostrophY)
"	"double-quote"
`	"backtick", "backquote"
<	"less-than", "left angle bracket"
>	"greater-than", "right angle bracket"

These pronunciations are all actually fairly common among the geek clique (or the nerd herd, if you prefer).

Other common vocabulary: Balises = tags or code

Computer Programming Basic Syntax

http://www.tutorialspoint.com/computer_programming/computer_programming_syntax.htm

Let's start with little coding, which will really make you computer programmer. I'm going to write a single-line computer program to write **Hello, World!** on your screen. Let's see how it can be written using different programming languages:

Hello World Program in C

Try to click **Try It** option to see the output of the following program. This Try it option is available at the top right-corner of the following code box. Try to change the content inside printf instead of **Hello World!** and then check its result. It just prints whatever you keep inside two double quotes.

```
#include <stdio.h>

main()
{
   /* printf() function to write Hello, World! */
   printf( "Hello, World!" );
}
```

This little Hello World program will help us in understanding various basic concepts related to C Programming.

Program Entry Point

For now just forget about **#include <stdio.h>** statement, but keep a note that you have to put this statement at the top of a C program.

So, every C program starts with main, which is called main function and then it is followed by a left curly brace. Rest of the program instruction is written in between and finally a right curly brace ends the program.

The coding part inside these two curly braces is called program body. The left curly brace can be in the same line as main{ or in the next line like it has been mentioned in the above program.

Functions

Functions are small units of programs and they are used to carry out a specific task. For example, above program makes use of two functions ***a* main** and ***b* printf**. Here, function main provides the entry point for the program execution and another function printf is being used to print an information on computer screen.

You can write your own functions which we will see in separate chapter, but C programming itself provides various built-in functions like main, printf, etc., which we can use in our programs based on our need.

Few programming languages use word **sub-routine** instead of function but their functionality is more or less same.

Comments

A C program can have statements enclosed inside **/*.....*/**. Such statements are called comments and these comments are used to make program user friendly and easy to understand the program. Good thing about comments is that they are completely ignored by compilers and interpreters. So you can whatever language you want to write your comments.

Whitespaces

When we write a program using any programming language, we use various printable characters to prepare programming statements. These printable characters are **a, b, c,......z, A, B, C,.....Z, 1, 2, 3,...... 0, !, @, #, $, %, ^, &, *, (,), -, _, +, =, \, |, {, }, [,], :, ;, <, >, ?, /, \, ~. `. ", '**. Hope I'm not missing any printable characters from your keyboard.

Apart from these characters, there are some characters which we use very frequently but they are invisible in your program and these characters are spaces, tabs \t, new lines\n. These characters are called **whitespaces**.

These three important whitespace characters are common in all the programming languages and they remain invisible in your text document having your program:

Whitespace	Explanation	Representation
New Line	This will be used to create a new line.	\n
Tab	This will be used to create a tab.	\t
Space	This will be used to create a space.	empty space

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it. Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. So you can write **printf"*Hello,World*!";** as follows. Here all the created spaces around "Hello, World!" are useless and the compiler will ignore them at the time of compilation.

```
#include <stdio.h>

main()
{

   /* printf() function to write Hello, World! */

   printf(    "Hello, World!"      );

}
```

Assuming, I make all these whitespace characters visible, then your above program will look like something below and you will not be able to compile it:

```
#include <stdio.h>\n
\n
main()\n
{
\n
\t/* printf() function to write Hello, World! */
\n
\tprintf(\t"Hello, World!"\t);\n
\n
}\n
```

Semicolons

Every individual statement in C Program, must be ended with a semicolon **;** For example, if you want to write "Hello, World!" twice, then it will be written as follows:

```
#include <stdio.h>

main()
{
   /* printf() function to write Hello, World! */
   printf( "Hello, World!\n" );
   printf( "Hello, World!" );
}
```

This program will produce the following result:

```
Hello, World!
Hello, World!
```

Here, I'm using new line character **\n** in first printf function to create a new line. Let us see what happens if I do not use this new line character:

```
#include <stdio.h>

main()
{
   /* printf() function to write Hello, World! */
   printf( "Hello, World!" );
   printf( "Hello, World!" );
}
```

This program will produce following result:

```
Hello, World! Hello, World!
```

I'm skipping explanation about identifiers and keywords and will take them in next few chapters.

Program Explanation

Let's try to understand how the above C program to print "Hello, World!" works. First of above program is converted into a binary format using C compiler. So let's put this code in test.c file and compile it as follows:

```
$gcc test.c -o demo
```

If there is any grammatical error *Syntaxerrorsincomputerterminologies*, then we fix it before converting it into binary format. If everything goes fine then it produces binary file called **demo**. Finally we execute produced binary demo as follows:

```
$./demo
```

Which produces following result:

```
Hello, World!
```

Here, when we execute binary **a.out** file, what computer does is, it enters inside the program starting from main and encounters a printf statements. Keep a note about line inside /*....*/ is a comment so it is filtered at the time of compilation. So printf function instructs computer to print the given line at the computer screen. Finally it encounters a right curly brace which indicates the end of main function and exit of the program.

Syntax Error

If you do not follow rules defined by the programing language then at the time of compilation you will get syntax error and program will not be compiled. From syntax point of view, even a single dot or comma or single semicolon matters and you should take care of such small syntax as well. Following is Hello, World! program but here I'm not using semicolon, let's try to compile following program:

```
#include <stdio.h>

main()
{
   printf("Hello, World!")

}
```

This program will produce following result:

```
main.c: In function 'main':
main.c:7:1: error: expected ';' before '}' token
 }
 ^
```

So bottom-line is that if you are not following proper syntax defined by the programming language in your program then you will get similar type of syntax errors and before trying next compilation you will need to fix them and then proceed.

Hello World Program in Java

Following is the equivalent program written in Java. This program will also produce same result **Hello, World!**.

```
public class HelloWorld
{
   public static void main(String []args)
   {
      /* println() function to write Hello, World! */
      System.out.println("Hello, World!");
   }
}
```

Hello World Program in Python

Following is the equivalent program written in Python. This program will also produce same result **Hello, World!**.

```
#  print function to write Hello, World! */
print "Hello, World!"
```

Hope you noted that for C and Java examples, first we are compiling programs and then executing produced binaries but in Python program we are directly executing it. As I explained in previous chapter, Python is an interpreted language and it does not need intermediate step called compilation.

Python programming languages does not require a semicolon ; to terminate a statement like we used in C and Java, rather a new line always means termination of the statement.

Conclusion

Not sure if you understood what I taught you above in this chapter, but if you did not understand then I will suggest to go through it once again and make sure you understood all the above concepts. But if you understood these concepts, then you are almost done and let's proceed to the next chapter, which you are going to enjoy a lot.