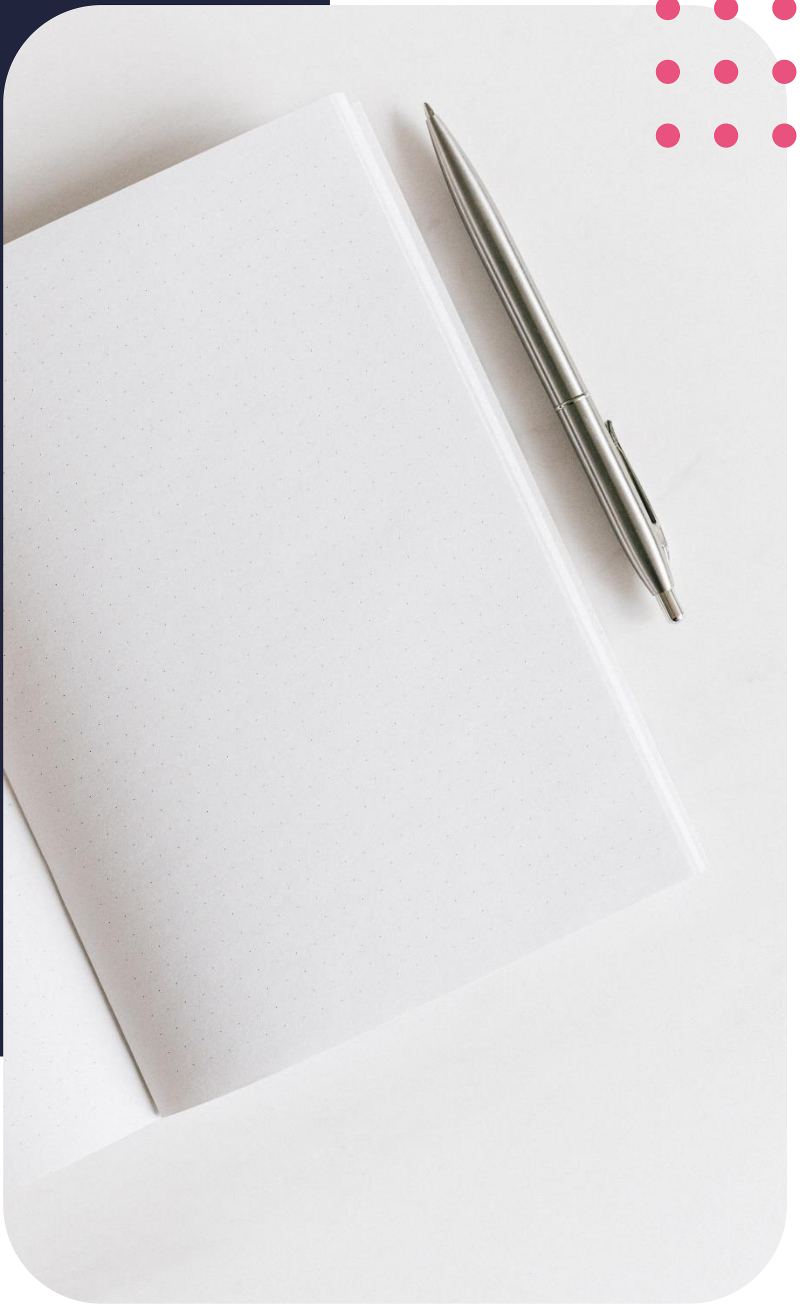
Diploma in Computer Science

C Syntax

Contents



Lesson outcomes

By the end of the lesson, you will be able to:

- Explore the syntax of the C programming language
- Discuss the uses of various features of the language
- Identify the keywords in C
- Explore the correct way of including comments in your code

Welcome to our fourth lesson in our second module of computer science. In this lesson, we will explore the cornerstones of programming, syntax. These are the rules that govern how you write your code and interact with the hardware. Just like on the road, at the airport, at the gym and even at home, there are rules that govern how we carry ourselves. You can't wear shoes on your ears, for example, neither can you overtake in front of oncoming traffic. Some rules, when not adhered to, can cause things not to transpire as expected, or even create a dangerous situation. Similarly, in c, if you do not follow the syntax and semantics, most of the time, you program will just flat out refuse to compile, or if you get away with compiling it, you will get unexpected results. In this lesson we will go through the rules that will help you get along nicely with C.

Tokens

We'll kick off with a flashback. Remember when we talked about language construction in module one? We discussed that when we want to construct a sentence, we use words, and when we want to create words, we use letters. A token is the smallest individual element in a programming language; the building blocks for creating a program. If you remember, again from module one, the compiler breaks down code into the smallest possible units during the compilation process. In module 1 we also looked at abstract machines as a basis for the design of programming language, this lesson is specifically an application of an abstract machine; legal programs are a subset of the possible strings that can be formed from the alphabet of the language. Also, tokens correspond to a set of strings. We will look at how each set is defined shortly. Tokens can be divided into 6 basic categories, namely keywords, identifiers, constants, strings, special symbols, and operators.

Tokens

comprises:

Letters:	
	Uppercase characters (a-z)
	Lowercase characters (a-z)

Numbers:	
	Digits from 0 to 9
	White space
	New line
	Carriage return
	Horizontal tab
Special characters:	
,	(comma)
{	(opening curly bracket)
.	(period)
}	(closing curly bracket)
;	(semi-colon)
[	(left bracket)
:	(colon)
]	(right bracket)
?	(question mark)
(	(opening left parenthesis)
'	(Apostrophe)
)	(closing right parenthesis)
"	(double quotation mark)
&	(ampersand)
!	(exclamation mark)
^	(caret)
|	(vertical bar)
+	(addition)
/	(forward slash)
-	(subtraction)
\	(backward slash)

*	(multiplication)
~	(tilde)
/	(division)
_	(underscore)
>	(greater than or closing angle bracket)
$	(dollar sign)
<	(less than or opening angle bracket)
%	(percentage sign)
#	(hash sign)

Keywords are words that have a specific, fixed meaning in a programming language, implying that it cannot under any circumstances be used to refer to anything else within the programming language. There are 32 keywords in c, all of which are written in lower case.

Auto	Declares automatic variables. Variables declared within function bodies are automatic by default. They are recreated each time a function is executed.
Break	Terminates the innermost loop immediately when it's encountered. It's also used to terminate the switch statement.
Case	Used when a block of statements has to be executed among many blocks. This is used in conjunction with switch and default
Char	Declares a character variable
Const	Used to declare constants
Continue	Generally used with for, while and do-while loops, when compiler encounters this statement it performs the next iteration of the loop, skipping rest of the statements of current iteration
Default	Use used when a block of statements has to be executed among many blocks. This is used in conjunction with case and switch. The statement under default is executed if all other conditions are not met
Do	Used in the while loop for conditional execution
Double	Used for declaring floating type variables.
Else	Used in the if statement for conditional execution
Enum	Used to declare enumerated variables
Extern	Declares that a variable or a function has external linkage outside of the file it is declared.
Short	Used in conjunction with int for small integers from -32768 to 32767
For	Used for looping code
Goto	Used to transfer control of the program to the specified label.
If	Used for conditional execution
Int	Used to declare integer type variables
Long	Used in conjunction with int for long integers, from -2147483648 to 214743648

Register	Creates register variables which are much faster than normal variables
Return	Terminates the function and returns the value.
Float	Used for declaring floating type variables.
Signed	Used in conjunction with int for normal integers, from -32768 to 32767
Sizeof	Evaluates the size of data in a variable or a constant
Static	Creates a static variable. The value of the static variables persists until the end of the program
Struct	Used for declaring a structure. A structure can hold variables of different types under a single name.
Switch	Used when a block of statements has to be executed among many blocks. This is used in conjunction with case and default
Typedef	Used to explicitly associate a type with an identifier.
Union	Used for grouping different types of variables under a single name.
Unsigned	Used in conjunction with int for positive integers, from 0 to 65535
Void	Means no value is passed or evaluated
Volatile	Used for creating volatile objects. A volatile object can be modified in an unspecified way by the hardware.
While	Used to execute a block of code repeatedly

This table shows the keywords that are available in c. As well as what they are used for. We will encounter these keywords in every program we write. Any program you write will have at least one keyword.

This table is quite a mouthful, so were going to condense it into a list, sorted by usage.

Basic usage of these keywords

Auto Declares automatic variables. Variables declared within function bodies are automatic by default. They are recreated each time a function is executed.

Break Terminates the innermost loop immediately when it's encountered. It's also used to terminate the switch statement.

Case Used when a block of statements has to be executed among many blocks. This is used in conjunction with switch and default

Char Declares a character variable

Const Used to declare constants

Continue Generally used with for, while and do-while loops, when compiler encounters this statement it performs the next iteration of the loop, skipping rest of the statements of current iteration

Default Use used when a block of statements has to be executed among many blocks. This is used in conjunction with case and switch. The statement under default is executed if all other conditions are not met

Do Used in the while loop for conditional execution

Double	Used for declaring floating type variables.
Else	Used in the if statement for conditional execution
Enum	Used to declare enumerated variables
Extern	Declares that a variable or a function has external linkage outside of the file it is declared.
Short	Used in conjunction with int for small integers from -32768 to 32767
For	Used for looping code
Goto	Used to transfer control of the program to the specified label.
If	Used for conditional execution
Int	Used to declare integer type variables
Long	Used in conjunction with int for long integers, from -2147483648 to 214743648
Register	Creates register variables which are much faster than normal variables
Return	Terminates the function and returns the value.
Float	Used for declaring floating type variables.
Signed	Used in conjunction with int for normal integers, from -32768 to 32767
Sizeof	Evaluates the size of data in a variable or a constant
Static	Creates a static variable. The value of the static variables persists until the end of the program
Struct	Used for declaring a structure. A structure can hold variables of different types under a single name.
Switch	Used when a block of statements has to be executed among many blocks. This is used in conjunction with case and default
Typedef	Used to explicitly associate a type with an identifier.
Union	Used for grouping different types of variables under a single name.
Unsigned	Used in conjunction with int for positive integers, from 0 to 65535
Void	Means no value is passed or evaluated
Volatile	Used for creating volatile objects. A volatile object can be modified in an unspecified way by the hardware.
While	Used to execute a block of code repeatedly

This table shows the keywords that are available in C as well as what they are used for. We will encounter these keywords in every program we write. Any program you write will have at least one keyword.

This table is quite a mouthful, so were going to condense it into a list, sorted by usage.

Basic usage of these keywords – (heading for new slide)

If, else, switch, case, default, for, while, do, break are all used for conditional execution in some form for decision control programming structures.

Int, float, char, double, long – these are the data types and used during variable declaration.

Void – a return type

Goto – used for redirecting the flow of execution.

Auto, signed, const, extern, register, unsigned – used to define a variable.

Return –used for returning a value.

Continue – used to move to the next iteration

Enum – set of constants.

Sizeof – it is used to evaluate the size of a variable or constant

Struct, typedef -used to define data structures

Union – it is a collection of variables, which shares the same memory location and memory storage.

Volatile- used for creating volatile objects

We mentioned that c is case sensitive, and this means you can actually get away with using a reserved word in capital letters, but this is not good programming practice and will just hurt the readability of your program. Imagine someone reading your code and see a variable declared as int case. It will work, yes, but that raises questions and stirs confusion.

Identifiers, variables, and constants

Identifiers are the names assigned to variables, functions, arrays, structures and pretty much everything else that you give your own name in a c program. In short, these are user defined. But hold your horses, you don't just name things willy-nilly! There are a few rules here too, though they still leave you with a decent amount of liberty as to what identifiers you can pick.

First character: the first character of the identifier should be either a letter of the alphabet or an underscore. It cannot be a special character or a digit.

No special characters: when it comes to identifiers, c really hates special characters. C does not support the use of special characters in identifiers. A comma or punctuation mark, for example, can't be used.

No keywords: this should come as no surprise, we discussed earlier that keywords are (only) used to identify special instructions.

No white space: white spaces include blank spaces, newline, carriage return, and horizontal tab, all of which can't be used.

Word limit: identifiers are limited to 31 characters.

Case sensitive: uppercase and lowercase characters are treated differently. Computer, for example, would be treated differently from computer.

Variables in c are basically the same as variables in mathematics; a symbol which functions as a placeholder for varying expression or quantities. The only difference is that variables in c, rather, in any programming language, variables are not arbitrary, they represent an actual block of memory, and that block of memory is reserved for the variable when the computer encounters it. Variables

Hold the information that you will be processing, and you can update them as the program that runs. To save memory, there are commands to destroy variables when they are no longer in use. We have a dedicated lesson on variables, and we are going to learn all about them in lesson 6

Constants

Constants are also derived from mathematics, and they work in the exact same wat as they do in mathematics -they do not change during execution, well in fact, they do not change at all, once defined in a program, they run as is and are not modified. When you define a constant, like a variable, you assign a block of memory to that constant, and give it a value. In lesson 6 we are going to look at constants in more depth and see exactly how they work and why you need them. A very popular and fairly obvious example of a constant is pi, the number 22/7. When performing calculations, you have no reason for changing the value of pi, you just use it as it is, so you declare it as a constant.

Punctuators

Punctuators are symbols that have syntactic and semantic meaning to the compiler but do not, themselves, specify an operation that yields a value. Some punctuators, either alone or in combination, can also be c operators or be significant to the pre-processor.

Any of the following characters are considered punctuators:

Punctuator	**Usage**	**Example**
< >	Header name	#include <stdio.h>
[ ]	Array delimiter	Char a[4];
{ }	Initializer list, function body, or compound statement delimiter	Char a[4] = {'s', 'h', 'a', 'w' };
( )	Function parameter list delimiter; also used in expression grouping	Int a (x,y)
*	Pointer declaration	Int *a;
,	Argument list separator	Char a[4] = { 's', 'h', 'a', 'w'};
:	Statement label	Labela: if (x == 0) x += 1;
=	Declaration initializer	Char a[4] = { "shaw" };
;	Statement end	X += 1;

...	Variable- length argument list	Int a (int y, ...)
#	Preprocessor directive	#include "shapes.h"
' '	Character constant	Char a = 'a';
" "	String literal or header name	Char a[] = "shaw";

As illustrated in this table, most(identifiers) of these work in pairs. You will find that if you leave out one of the two, your ide will scream before you even get to compiling your code! This is because of the fact that most of these

Punctuators are used to "open" and "close" sections of your code, essentially directing the compiler to treat that code in a certain way. The compiler needs to know when to start, for example, treating a stream of characters as a string, and when to stop, so you can never get away with leaving out a punctuator. This is, of course, true for all aspects of your code, as we have been discussing in our lessons; every single line of code that you write should mean exactly one thing.

The most used punctuator is probably the semicolon. There is no way you can write a program without using the semicolon, because it signifies the end of a statement. The semicolon is normally referred to as a terminator. When debugging your code, one of the first things that you can check for is whether you have terminated all your statements. The weird thing is, your program might actually compile, but then go on to proceed to produce amazingly incorrect results.

Operators

Operators are symbols that

tell the computer to perform a specific arithmetic or logical operation. C contains arithmetic, relational, logical, bitwise and assignment operators. We are going to cover operators in more detail in lesson 8, which is entirely dedicated to that. Since we have a lesson reserved entirely for this, we are just going to glide over and see what it's about.

Arithmetic operators do exactly what you are used to them doing in mathematics as illustrated in this table.

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

--	Decrements operator decreases integer value by one

Relational operators

Relational operators are used to check certain attributes of the operands in question.

Operator	**Usage**
==	Used to check if two operands are equal
!=	Used to check if two operands are not equal.
>	Used to check if the operand on the left is greater than the operand on the right
<	Used to check if the operand on the left is smaller than the operand on the right
>=	used to check if the operand on the left is greater than or equal to the operand on the right
<=	Used to check if the operand on the left is smaller than or equal to the operand on the right

Logical operators

3 logical operators are supported in C.

Operator	**Usage**
&&	Logical AND
||	Logical OR
!	Logical NOT

Bitwise operators

Bitwise operators perform data manipulation operations at bit level. These operators also perform shifting of bits from right to left.

Operator	**Usage**
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Assignment Operators

Assignment operators are used to assign values to variables

Operator	**Description**
=	Whatever is on the right side is copied into the operand on the left
+=	Whatever is on the right side is added into the operand on the left and the result is stored in the operand on the left
-=	Whatever is on the right side is subtracted the operand on the left and the result is stored in the operand on the left
*=	Whatever is on the right side is multiplied by the operand on the left and the result is stored in the operand on the left
/=	Whatever is on the right side is divided by the operand on the left and the result is stored in the operand on the left
%=	Calculates the modulus using two operands and assigns the result to left operand

Other operators

Operator	**Description**
sizeof	Returns the size of a variable
&	Returns the address of a variable
*	Pointer to a variable

Syntax categorisation

The C syntax can be categorised into distinct control types, such as functions and control statements.

Functions

We can’t talk about code without talking about functions, right? Functions are the containers in which most of the functionality of our code will be contained. This includes the main function, from which we control the program.

A function is a set of statements that take inputs, do some specific computation and produce output. The idea here is to put together a task that is used multiple times in a program, give it a name and when you want to use it, call it by name. this saves you the trouble of writing the same code again and again. A typical function comprises a return type, a function name, arguments and code.

More specifically, the syntax is

```
return_type function_name([ arg1_type arg1_name, ... ]) { code }
```

Note here, that we have used underscores in return type and the function name. this is because spaces are not allowed here. If we look at how we declared the main function in our hello world program, we will see that it follows these exact guidelines.

```
int main(void){}
```

Here, int is the return type. Remember we said if the function executes successfully, the return value is 0, which is what int denotes. Main is the name of the main function, and since our function is not necessarily returning any value from calculation, the argument is void. This is enclosed in curved brackets. The code in the function is enclosed in curly brackets. For user defined functions, we need to terminate the function, meaning the general form will look like this

```
return_type function_name([ arg1_type arg1_name, ... ]) { code };
```

This is needed so that the computer knows that the block of code has come to an end.

Flow control

You can choose to

execute a block of code only when certain conditions are met, or only while a certain condition is true. these are called control structures. There’s the famous if structure, for structure, the while structure and the case structure. The if structure (heading)allows you to execute code based on a set of conditions. The basic syntax is

```
If(condition){
code}
else{
code};
```

You can nest as many of these as you want, but it would just be neater to use the case statement

The case statement (heading) is a bit like the if statement, except it has a lot more conditions. Its general syntax looks like this:

```
switch(expression) {

   case constant-expression  :
      statement(s);
      break;

   case constant-expression  :
      statement(s);
      break;
     default :
   statement(s);
}
```

Under default we type in the code that is executed if none of the conditions are met.

The while structure (heading) is used to repeat a block of code until a certain condition is met, and the syntax looks like this

```
while(condition) {
   statement(s);
}
```

As long as the condition in brackets is true, the code within the while statement will be executed over and over

The for statement (heading) works in pretty much the same way as the while statement, except that it starts with an initial condition, and increments or decrements until it reaches a final condition. At every step, the condition is evaluated. The syntax for the for statement looks like this

```
for ( initialCondition; FinalCondition; increment ) {
   statement(s);
}
```

Data types

Since we have covered some of the available data types, we will condense it in this table.

Type	Explanation	Minimum size (bits)	Format specifier
char	Smallest addressable unit of the machine that can contain basic character set. It is an integer type.	8	%c
signed char	Of the same size as *char*, but guaranteed to be signed.	8	%c (or %hhi for numerical output)
unsigned char	Of the same size as *char*, but guaranteed to be unsigned.	8	%c (or %hhu for numerical output)
short short int signed short signed short int	*Short* signed integer type.	16	%hi or %hd
unsigned short unsigned short int	*Short* unsigned integer type.	16	%hu
int signed signed int	Basic signed integer type.	16	%i or %d
unsigned unsigned int	Basic unsigned integer type.	16	%u
long long int signed long signed long int	*Long* signed integer type.	32	%li or %ld

unsigned long unsigned long int	*Long* unsigned integer type.	32	%lu
long long long long int signed long long signed long long int	*Long long* signed integer type	64	%lli or %lld
unsigned long long unsigned long long int	*Long long* unsigned integer type.	64	%llu
float	Real floating-point type, usually referred to as a single-precision floating-point type.		Converting from text: %f %F %g %G %e %E %a %A
double	Real floating-point type		%lf %lF %lg %lG %le %lE %la %lA
long double	Real floating-point type, usually mapped to an extended precision floating-point number format.		%Lf %LF %Lg %LG %Le %LE %La %LA

There are other data types such as structures, pointers, unions and arrays, which we shall cover later on in the course.

Extras

In this section, we look at some of the things that are often overlooked but can give you a headache when writing code. Because computer language is so precise and not as flexible as human language, it’s important to know all the fine details.

Comments

As discussed in the previous lesson, comments are necessary to make code readable, and are not needed during the compilation process. Whether you write code with comments or without comments, it will still compile the same way.

Quick recap, when writing multiline comments, you use a forward slash followed by an asterisk /* to open the comment. Here you can write as many paragraphs as you want, till your heart is content, then close off the comment with an asterisk and a forward slash*/

When you did the challenge form last lesson, you may have notices that when you put opening punctuation on the code, everything that comes after that punctuation is immediately identified as a comment. This is one thing you should be wary of when using multiline comments; if you forget to close the comment, all the code below that comes after the opening punctuation will be regarded as one big comment and will not be compiled.

The other type of comment is inline comments. These only apply to the line in which they are written. These are indicated by a double forward slash//. This type of comments do not need to be closed. Once the compiler encounters a double slash in the line in which the comment is written, it disregards everything that comes after it and moves to the next line. It's very rare to encounter errors that involve single line comments, unless maybe you put one slash instead of two, or put the comment in the wrong place, such as inside an expression or at the start of a line in which an expression is contained. That said, it is worth mentioning that inline comments should be placed either in its own line or at the end of a line that contains code.

You also cannot have comments within comments. There's no real reason why you would want to do that, and the text will already be considered a comment anyway.

You can include code in comments. Within comments, you can put just about anything as the rules of C do not apply here. A popular type of comment that you will encounter if you look at other people's code is the TODO comment. This is usually used for code that was either implemented but does not yet work as expected or was left out so that it would be attended to at a later time.

You will find it a lot in open-source code, where programmers indicate to other contributors that the code needs improvements, or it is a work-in-progress. Also, any personal decisions that you make should be reflected in your documentation. This will be beneficial if, for example, someone is assigned to your previous projects while you work on newer projects, there will be no back and forth between you and the assignee as everything will be clear.

Case sensitivity

As mentioned earlier, C is case sensitive. (example)This means that a variable name, such as age, would be a different variable from another declared as Age. It's also worth mentioning that it's not a good practice to name your variables too similarly as this causes a lot of confusion to someone who reads your code, sure enough, it conforms to the syntax and semantics of C programming, but you should always remember the programming anthem at all times; code should be readable. This not only applies to C, but to every language you are going to learn in your life.

White space

A line containing only whitespace is referred to as a blank line, and a C compiler will ignore it. White space is only really there to make code readable to humans, and the computer doesn't need it. When the lexical analyser goes through your code, it produces a stream of tokens, and white space won't be needed to distinguish between elements anymore. This means that all those spaces that you put between words will be chucked out before the compiler even kicks in!

White space refers to blanks, tabs, newline characters and comments. Remember earlier we said comments have no significance in the compilation process and are just chucked out. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as a keyword, ends and the next element begins. For instance, in the statement:

```
int num1;
```

there must be at least one whitespace character between int and num1 for the compiler to be able to tell which is the keyword and which is the variable.

Now that we have learnt the basics of what code entails in C, let's not look at a few side notes that will make your life easier as a programmer and computer scientist.

Learning how to code is pretty much the same as learning to speak a new language. the movies speak it and interact with text and people who are proficient in the language, the more you become proficient yourself. Likewise, the more you practise writing code and encountering various syntax and semantics, the clearer it becomes to you and the better the code you write.

Writing code is not just about writing a programme that does the job anyhow. there is also an aspect that differentiates everyone else from the top cream of programmers. that is code quality. If you recall from module 1, the sole purpose of computer science is to come up with ways to better utilise the power that computers have. As a computer scientist, your job is to create faster and better instructions for a computer, and this starts by understanding the language that the computer speaks. This is precisely why we spent an entire lesson learning about syntax!

Think of it this way, a lot of people know how to speak English, but writers are a lot better than many other people because they interact with the language intricately and produce colourful text and imagery that can send you imagination on a road trip. if a person who has 0 experience writing novels, tries their hand at writing a novel, you will probably fall asleep on the first page of that novel, as it will be so boring that you feel like pulling your hair out.

This is the same thing with programming. you might write code, and it works, but what does it computer think about it? is it really the best way of doing it? as that old saying goes, there are so many ways of killing a cat, but then, in programming and computer science in general, there are very few ways of effectively and efficiently killing a cat! back to computer science, you will not step they are a number of popular algorithms which have been endorsed as the best way of solving a particular problem. there are even problems that have one known way of solving them! learning the syntax of a programming language is only a tool in computer science that you use as a language to communicate your solutions to a computer. by the end of this course, you have been introduced to all the tools that you need to solve computer science problems. the rest is really up to you to use those tools as effectively as possible.

Conclusion

In this lesson we looked at the things that govern your code, as well as things you can do while writing code. This forms the foundation of your understanding of the programming language and helps you master the building blocks that will form meaningful statements in your code. As we established in module 1, code needs to make sense to both you and the computer as it is our medium of communication with the machines that power our lives. In our next lesson, we will go deeper in exploring the things that C can do as we explore one of the most important components of our programs, that is data types. Thai is all from me, and I hope you have a fun time coding. See you next time!

References

(2020) Csit.kutztown.edu. Available at: https://csit.kutztown.edu/~schwesin/spring20/csc310/lectures/Regular_Languages.pdf

Automata Theory and Languages Introduction to Automata Theory. (n.d.). [online] Available at: https://www.univ-orleans.fr/lifo/Members/Mirian.Halfeld/Cours/TLComp/TLComp-introTL.pdf.

C Programming Tutorial. (n.d.). [online] Available at: https://www.unf.edu/~wkloster/2220/ppts/cprogramming_tutorial.pdf.

Programiz.com. (2020). The Complete List of all 32 C Programming Keywords (With Examples) - Programiz. [online] Available at: https://www.programiz.com/c-programming/list-all-keywords-c-language

Uic.edu. (2020). Programming Concepts Course Notes - Data Types. [online] Available at: https://www.cs.uic.edu/~jbell/CourseNotes/ProgrammingConcepts/DataTypes.html

Utah.edu. (2020). Programming - Topics. [online] Available at: https://www.cs.utah.edu/~germain/PPS/Topics/index.html

Wisc.edu. (2020). Lecture notes - C programming. [online] Available at: http://pages.cs.wisc.edu/~powerjg/cs354-fall15/Notes/C.basics.html