



Presented By :Mostafa Saqly

C Introduction





Content

- ◇ **Software Development Life Cycle(SDLC)**
- ◇ **Interpreter Vs Compiler**
- ◇ C Datatypes
- ◇ C Memory & Variables & Constant
- ◇ C Keywords & Identifier
- ◇ C Inputs/Outputs
- ◇ C operators





What is the Software Development Life Cycle?

Software Development Life Cycle is the application of standard business practices to building software applications. It's typically divided into six to eight steps : Planning, Requirements, Design, Build, Document, Test, Deploy, Maintain.

Some project managers will combine, split, or omit steps, depending on the project's scope. These are the core components recommended for all software development projects..

SDLC is a way to measure and improve the development process. It allows a fine-grain analysis of each step of the process. This, in turn, helps companies maximize efficiency at each stage. As computing power increases, it places a higher demand on software and developers. Companies must reduce costs, deliver software faster, and meet or exceed their customers' needs. SDLC helps achieve these goals by identifying inefficiencies and higher costs and fixing them to run smoothly.



How the Software Development Life Cycle Works?

The Software Development Life Cycle simply outlines each task required to put together a software application. This helps to reduce waste and increase the efficiency of the development process. Monitoring also ensures the project stays on track, and continues to be a feasible investment for the company.



The Seven Phases of the SDLC

Seven Phases of Software Development Life Cycle

Planning



Design & Prototyping



Testing



Operations & Maintenance



Define Requirements

Software Development

Deployment



1

Planning

In the Planning phase, project leaders evaluate the terms of the project. This includes calculating labor and material costs, creating a timetable with target goals, and creating the project's teams and leadership structure.

Planning should clearly define the scope and purpose of the application

A decorative pattern of hexagons in various shades of blue and teal. Some hexagons contain icons: a lightbulb, a thumbs up, a smartphone, a magnifying glass, a gear, and a speech bubble. A large teal hexagon in the center-left contains the number '2'.

2

Define Requirements

Defining requirements is considered part of planning to determine what the application is supposed to do and its requirements.

Requirements also include defining the resources needed to build the project



3

Design and Prototyping

The Design phase models the way a software application will work some aspects of the design include : **Architecture , User Interface , Platforms , Programming , Communications , Security .**

Prototyping can be a part of the Design phase. A prototype is like one of the early versions of software in the Iterative software development model. It demonstrates a basic idea of how the application looks and works.



4

Software development

This is the actual writing of the program. A small project might be written by a single developer, while a large project might be broken up and worked by several teams.

They also help ensure compatibility between different team projects and to make sure target goals are being met.

A decorative pattern of hexagons in various shades of blue and teal on the left side of the slide. Some hexagons contain icons: a lightbulb, a thumbs up, a smartphone, a magnifying glass, a gear, and a speech bubble.

5

Testing

It's critical to test an application before making it available to users.

Different parts of the application should also be tested to work seamlessly together—performance test, to reduce any hangs or lags in processing.



6

Deployment

In the deployment phase, the application is made available to users. Many companies prefer to automate the deployment phase. This can be as simple as a payment portal and download link on the company website. It could also be downloading an application on a smartphone.



7

Operations and Maintenance


At this point, the development cycle is almost finished. The application is done and being used in the field. The Operation and Maintenance phase is still important, though. In this phase, users discover bugs that weren't found during testing. These errors need to be resolved, which can spawn new development cycles.



Compiler vs Interpreter

Interpreter Vs Compiler

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
Interpreters usually take less amount of time to analyze the source code. However, the overall execution time is comparatively slower than compilers.	Compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Programming languages like JavaScript, Python, Ruby use interpreters.	Programming languages like C, C++, Java use compilers.





C Data Types

- ◇ In C programming, data types are declarations for variables. This determines the type and size of data associated with variables.



C Data Types

Type	Size (bytes)	Format Specifier
<code>int</code>	at least 2, usually 4	<code>%d</code> , <code>%i</code>
<code>char</code>	1	<code>%c</code>
<code>float</code>	4	<code>%f</code>
<code>double</code>	8	<code>%lf</code>
<code>short int</code>	2 usually	<code>%hd</code>
<code>unsigned int</code>	at least 2, usually 4	<code>%u</code>
<code>long int</code>	at least 4, usually 8	<code>%ld</code> , <code>%li</code>
<code>long long int</code>	at least 8	<code>%lld</code> , <code>%lli</code>
<code>unsigned long int</code>	at least 4	<code>%lu</code>
<code>unsigned long long int</code>	at least 8	<code>%llu</code>
<code>signed char</code>	1	<code>%c</code>
<code>unsigned char</code>	1	<code>%c</code>
<code>long double</code>	at least 10, usually 12 or 16	<code>%Lf</code>



C Data Types

◆ **Int :**

Integers are whole numbers that can have both zero, positive and negative values but no decimal values. For example, 0, -5, 10 , The size of int is usually 4 bytes (32 bits). And, it can take 2^{32} distinct states from -2147483648 to 2147483647

◆ **float and double :**

The size of float (single precision float data type) is 4 bytes. And the size of double (double precision float data type) is 8 bytes.

For Example: `float normalizationFactor = 22.442e2;`





C Data Types

◇ **char**

Keyword char is used for declaring character type variables. For example,

For Example : `char test = 'h';`


The size of the character variable is 1 byte.

◇ **void**

void is an incomplete type. It means "nothing" or "no type". You can think of void as **absent**.

- ◇ For example, if a function is not returning anything, its return type should be void.

Note that, you cannot create variables of void type.





C Data Types

short and long :


- ◇ If you need to use a large number, you can use a type specifier long. Here's how:
- ◇ `long a; long long b; long double c;` Here variables a and b can store integer values. And, c can store a floating-point number.
- ◇ If you are sure, only a small integer ($[-32,767, +32,767]$ range) will be used, you can use short.
- ◇ `short d;`





C Data Types

signed and unsigned

- ◇ In C, signed and unsigned are type modifiers. You can alter the data storage of a data type by using them. For example,
 - ◇ `unsigned int x; int y;`
 - ◇ Here, the variable `x` can hold only zero and positive values because we have used the unsigned modifier.
 - ◇ Considering the size of `int` is 4 bytes, variable `y` can hold values from -2^{31} to $2^{31}-1$, whereas variable `x` can hold values from 0 to $2^{32}-1$.
- 

Variables



- ◇ In programming, a variable is a container (storage area) to hold data.
- ◇ To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location.
- ◇ For example:


```
int playerScore = 95;
```

Here, playerScore is a variable of int type. Here, the variable is assigned an integer value 95.





Variables

- ◇ **Rules for naming a variable**
 - ◇ A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
 - ◇ The first letter of a variable should be either a letter or an underscore.
 - ◇ There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.
 - ◇ **Note:** You should always try to give meaningful names to variables. For example: firstName is a better variable name than fn.
- 




Variables

Floating-point Literals :

- ◇ A floating-point literal is a numeric literal that has either a fractional form or an exponent form. For example:
- ◇ -2.0 0.0000234 -0.22E-5

Characters :

- ◇ A character literal is created by enclosing a single character inside single quotation marks. For example: 'a', 'm', 'F', '2', '}' etc.
- 



Variables

Escape Sequences

- ◇ Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc.



Variables

Escape Sequences	
Escape Sequences	Character
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	Newline
<code>\r</code>	Return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\\</code>	Backslash
<code>\'</code>	Single quotation mark
<code>\"</code>	Double quotation mark
<code>\?</code>	Question mark
<code>\0</code>	Null character

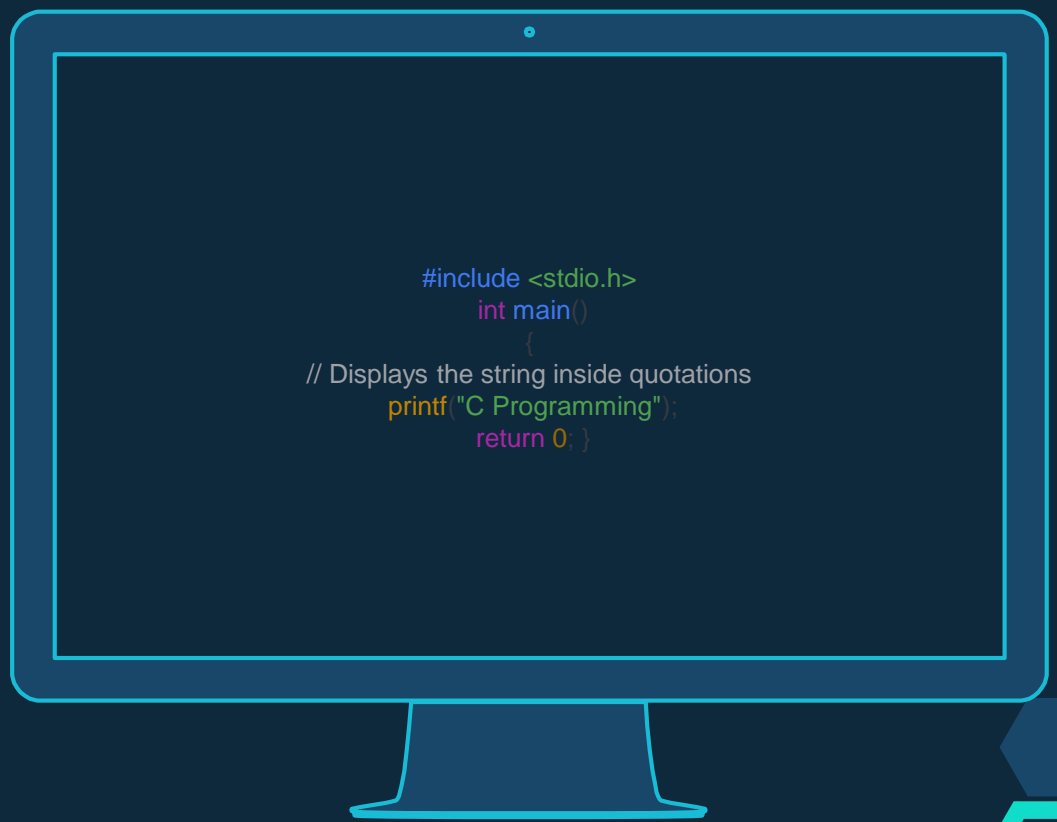


C Output

- ◇ In C programming, printf() is one of the main output function. The function sends formatted output to the screen.

- ◇ **Output**

C Programming



```
#include <stdio.h>
int main()
{
    // Displays the string inside quotations
    printf("C Programming");
    return 0; }
```





C Input Output (I/O)

How does this program work?

- ◇ All valid C programs must contain the `main()` function. The code execution begins from the start of the `main()` function.
- ◇ The `printf()` is a library function to send formatted output to the screen. The function prints the string inside quotations.
- ◇ To use `printf()` in our program, we need to include `stdio.h` header file using the `#include <stdio.h>` statement.
- ◇ The `return 0;` statement inside the `main()` function is the "Exit status" of the program. It's optional.



C Input

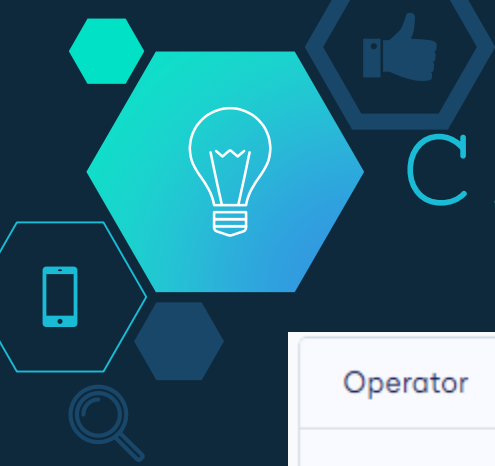
- ◇ In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards.

```
#include <stdio.h>
int main() {
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
    printf("Number = %d", testInteger); return 0; }
```



C Programming Operators

- ◇ An operator is a symbol that operates on a value or a variable.
- ◇ C has a wide range of operators to perform various operations:
 1. **C Arithmetic Operators**
 2. **C Increment and Decrement Operators**
 3. **C Assignment Operators**
 4. **C Relational Operators**
 5. **C Logical Operators**
 6. **C Bitwise Operators**
 7. **Other Operators**



C Arithmetic Operators

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)



C Increment and Decrement Operators

- ◇ C programming has two operators increment `++` and decrement `--` to change the value of an operand (constant or variable) by 1.
- ◇ Increment `++` increases the value by 1 whereas decrement `--` decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.



C Assignment Operators

Operator	Example	Same as
=	<code>a = b</code>	<code>a = b</code>
+=	<code>a += b</code>	<code>a = a+b</code>
-=	<code>a -= b</code>	<code>a = a-b</code>
*=	<code>a *= b</code>	<code>a = a*b</code>
/=	<code>a /= b</code>	<code>a = a/b</code>
%=	<code>a %= b</code>	<code>a = a%b</code>



C Relational Operators

Operator	Meaning of Operator	Example
==	Equal to	<code>5 == 3</code> is evaluated to 0
>	Greater than	<code>5 > 3</code> is evaluated to 1
<	Less than	<code>5 < 3</code> is evaluated to 0
!=	Not equal to	<code>5 != 3</code> is evaluated to 1
>=	Greater than or equal to	<code>5 >= 3</code> is evaluated to 1
<=	Less than or equal to	<code>5 <= 3</code> is evaluated to 0



C Logical Operators

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression <code>((c==5) && (d>5))</code> equals to 0.
	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression <code>((c==5) (d>5))</code> equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression <code>!(c==5)</code> equals to 0.



C Bitwise Operators

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right



Other Operators

Comma Operator

- ◇ Comma operators are used to link related expressions together.
- ◇ For example : `int a, c = 5, d;`

The sizeof operator

- ◇ The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc.).
- ◇ Other operators such as ternary operator `?:`, reference operator `&`, dereference operator `*` and member selection operator `->`



Code Blocks Download Link

<https://sourceforge.net/projects/codeblocks/files/>



Thanks!

Any questions?

