GROOT ACADEMY

C language

* Course Overview :

- No. of modules to cover = 12
- Brief details of modules.
  - → Quick Introduction
  - → Datatypes, variables and control statment
  - → Loops & if-else, switch cases
  - → Arrays in C & its types
  - → Strings in C & its types
  - → Functions in C
  - → Pointers in C
  - → Structures & union
  - → Dynamic memory allocation in C
  - → Linked lists
  - → file Management in C
  - → Preprocessor

+91-8233266276
info@grootacademy.com
122/166, 2nd Floor, vijay path, Mansarovar, 302020
www.Grootacademy.com

* Module One

◎ What is a a programming language?

⇒ A programming language is a formal set of rules and instructions used to communicate with a computer and create software application. It provides a way to write algorithms that a computer can understand and execute.

⇒ Programming languages can be categorized into:

a) Low level languages – close to machine code, eg. Assembly, Machine lang.
b) High level languages – easier to read & write. ex- C, C++, python, java.
c) Scripting languages – used for automation, eg. javascript, python, bash.

NOTE: → each language has its syntax (rules) and designed for specific types of tasks, like web development, system programming or data analysis.

⊙ History and features of C?

⇒ History:

- developed by 'Dennis Ritchie' at Bell labs in 1972.
- it was created for system programming, especially for writing UNIX operating system.
- later on, the language was influenced by "B" and "BCPL" lang.

- led to the development of C++, java and other modern languages
- Standardized as ANSI C (C89/C90) and later as C99, C11, C18.

⇒ Features of C:

a) Procedural language - follows a structured approach using functions.

b) fast & efficient - close to hardware, minimal runtime overhead

c) low level manipulation - supports pointers, memory management & bitwise operations.

d) Portable - code can run on different systems with minimal modif.

e) Modular Programming - supports function based modular code

f) Rich library - standard library functions for I/O, memory and string operations.

Q why C is important?

⇒ it is the Foundation of Modern programming.

↳ learning C helps us to understand how computer works at low level.

↳ C++, java, python, JS have roots in C

⇒ System programming and Operating Systems:

↳ C was used to build UNIX and modern OS like- linux, windows & MacOS are written in C

↳ it provides direct access to hardware through pointers & memory allocation

embedded systems & Hardware Programming:

↳ C is widely used in microcontrollers, IOT & robotics

↳ device drivers are written in C.

Performance & efficiency:

↳ C is a compiled language. Making it faster than interpreted languages like python.

↳ it has low memory footprint & direct hardware access.

⇒ Portability & cross-platform development:
↳ can be used on multiple platforms
↳ for ex- desktops, super computers, mobile devices.

⇒ Memory Management:
↳ many languages has automatic garbage collection.
C allows manual memory management through malloc() and free().

⇒ used in game development & graphics:
↳ game engines like "unreal engine" use C/C++ for core functionalities
↳ graphics libraries like "vulkan, OpenGL, rely on C for performance.

⇒ Database & compiler development:
↳ popular databases like MySQL, PostgreSQL are written in C.

GROOT ACADEMY

⊙ Procedural language :

⇒ A procedural Programming language is a type of programming paradigm that follows a step by step approach using procedures (functions) to structure the code.

⇒ Key characteristics are -

a) follows a top-down Approach.

↳ programs are structured into functions (procedures) that performs specific tasks.

↳ execution starts from the main() function and proceeds step by step.

↳ to avoid repetitive logic, reusable functions are used -

↳ ex- without functions :

```
#include <stdio.h>

int main () {
    int a = 10; int b = 20;
    int sum1 = a + b;
    printf ("sum : %d \n", sum1);

    int x = 5; int y = 10;
    int sum2 = x + y;
    printf ("sum : %d \n", sum2);

return 0;
}
```

with functions :

```
#include <stdio.h>

int add(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int result1 = add(10, 5);
    int result2 = add(5, 15);
```

→ Printf("sum : %d \n", result

+91-8233266276
info@grootacademy.com
122/166, 2nd Floor, vijay path, Mansarovar, 302020
www.Grootacademy.com

b) The program is built around functions that manipulate global or local variables.

↳ on-

```
#include <stdio.h>
int globalvar = 40;

void modifyglobal () {
    globalvar += 50;
    printf("modifyglobal : globalvar = %d\n", globalvar);
}
void localexample() {
    int localvar = 30;
    printf("localVar: localvar = %d\n", localvar);

int main () {
    printf("globalvar is : globalvar = %d\n", globalvar);

    modifyglobal();
    printf("now new globalvar = %d\n", globalvar);

    localexample();
    printf("%d\n", localvar);
    return 0;
}
```

error

c) Sequential Execution Flow
- code is executed in order, it appears until control structures (loops, conditions) change it.

d) Memory management with stack and heap:
- local variables are stored in stack
- dynamically allocated memory is stored in the heap.
- ex-

```
#include <stdio.h>
#include <stdlib.h>
```

→ function for stack

```
void stack() {
    int stackvar = 10;
    printf("stack var is %d \n", stackvar);
}
```

→ function for heap

```
void heap() {
    int *heapvar = (int*) malloc(sizeof(int));
    if (heapvar = NULL) {
        printf("memory allo. failed");
        return;
    }
    *heapvar = 20;
    printf("Heapvar is %d \n", *heapvar);
    free(
```

•) Role of Compiler's in C :

a) Preprocessor : expands (#include <stdio.h>)
b) Compiler : converts code to assembly language
c) Assembler : generates object file (main.o)
d) Linker : links "printf()" and creates final executable

↳ Some popular C compilers are :
a) GCC (GNU compiler collection)
b) clang (fast, used in Mac OS)
c) MSVC (Microsoft visual C++) (windows)

↳ How to compile a program in C :

Step-1 - install GCC
⊙ for windows : MinGW or WSL
⊙ for Linux/MacOS : Open terminal and run :
"sudo apt install gcc
sudo yum install gcc
brew install gcc"

Step-2 - to create a file, and save the file as, - "hello.c"
↳ file : #include <stdio.h>
int main() {
printf("Hello world!\n");
return 0;
}

Step-3 - compile the program
⊙ open a terminal & run →
"gcc hello.c -o hello

[gcc - calls compiler
hello.c - C source file
-o hello - creates an output file named hello]

Step-4 : run → hello.exe (windows)

① Role of Interpreter :

↳ an interpreter is a program that executes code line-by-line, instead of converting into a machine code all at once.

↳ languages that use interpretor are- python, JS, Ruby, PHP.

↳ Compilation vs Interpretation Example :

C (compiled)

```
#include <stdio.h>
int main () {
    printf ("Hello compiler" \n");
    return 0;
}
```

→ Run

```
gcc program.c -o program
./program
```

{translated beford execution}

Python (interpreted)

```
print ("Hello Interpretor");
```

→ Run

```
python program.py
```

{translated dureing execution}

* C does not use an interpretor but relies on a compiler
* Interpretors are useful for dynamic interactive execution.

GROOT ACADEMY

① The structure of a C program

- a C program consists of following sections.
  1.) Preprocessor Derivatives (#include)
  2.) Global declarations (Macros, constants, global variables, function Prototypes)
  3.) main() function (entry point of program)
  4.) User defined functions (optional, for modular programming)

- ex-

```
#include <stdio.h>          // 1. preprocessor directive

#define PI 3.14             // 2. global declaration (optional)

void greet();               // function prototype

int main() {                // entry point main fun.
    printf("Hello world");
    greet();                // function call
    return 0;
}

void greet() {              // user-defined function or function definition
    printf("welcome to C programming");
}
```

⊙ writing the C programs:

⇒ writing a C program with the user input.

```
#include <stdio.h>
int main() {
    int age;                          → variable declaration
    printf("enter your age :");
    scanf("%d", &age);                → taking input from user
    printf("your age is : %d \n", age);   → printing output
    return 0;
}
```

Miscellaneous Topics

⇒ what are the real world use cases of C language?

↳ Operating System Development (windows, macOS etc.)

↳ Embedded systems & IoT (MRI machines, pacemakers)
Smart TVs, washing Machines)

↳ database Mangement systems
(MySQL)
(Orcale database, Postgres SQL)

↳ Networking & telecommunication Systems (Cisco Router, network drivers,
TCP/IP, HTTP, FTP protocol)

↳ Robotics & AI

↳ Cybersecurity & Cryptography