Session - 1

Organization : TeachToTech
Instructor : Ayush Raj
Duration : 1.5 hr

(knight @leetcode, 4★@codechef, Specialist @codeforces)

Welcome to the World of C Programming

What You'll Master Today

This session covers the essential building blocks that form the foundation of C programming. You'll gain hands-on experience writing actual code and understand how your programs come to life.

Today's Agenda

- Introduction to C and its importance
- Structure and anatomy of C programs
- Your first "Hello World" program
- Program vs. Process concepts
- The compilation pipeline
- Variables, keywords, and data types

Why Learn C? The Foundation of Modern Computing

The Foundation

Modern languages like Python, Java, and C++ are all built on top of C. Understanding C means understanding how these languages work under the hood.

Blazing Speed

C is extremely fast and efficient, making it the go-to choice for gaming engines, operating systems, and high-performance applications.

System Control

C provides direct access to computer hardware—memory, CPU, and more. This low-level control is essential for system programming.

The Structure of a C Program

Every C program follows a specific structure—think of it as a recipe that must be followed exactly. Miss a part, and your code won't run.

01

#include <stdio.h>

The Toolbox: This header file contains pre-built tools like printf() so you don't have to write everything from scratch.

02

int main()

The Start Button: This is where your program begins. The computer looks for this function to start executing your code.

03

{ } Curly Braces

The Boundaries: All your program logic must live inside these braces. They define the scope of your main function.

04

return 0;

The Exit Status: This tells the operating system that your program finished successfully without errors.

Your First Program: Hello World

Let's Write Code!

Type this code into your compiler exactly as shown. Pay attention to every character.

```c
#include <stdio.h>

int main() {
    printf("Hello, World!");
    return 0;
}
```

Don't forget the semicolon (;) at the end. it tells the compiler where each statement ends.

Program vs. Process: A Critical Distinction

Understanding this difference is very essential.

Program: The Passive Entity

- A file stored on your hard disk
- Contains instructions but isn't running
- Think of it as a **recipe book** sitting on a shelf
- Static and unchanging until executed

Process: The Active Entity

- The program currently running in RAM
- Actively executing instructions
- Think of it as **cooking the meal** from that recipe
- Dynamic and consuming system resources

The Compilation Process: From Code to Execution

Computers only understand binary (0s and 1s). The compiler translates your C code into machine language through four distinct stages.

- ## Writing the Source Code

  We write the program in **C language. T**his code is written by **humans**, not understood by the machine.

- ## Preprocessor

  Cleans up the code by removing comments and expanding macros. Prepares your code for translation.

- ## Compiler

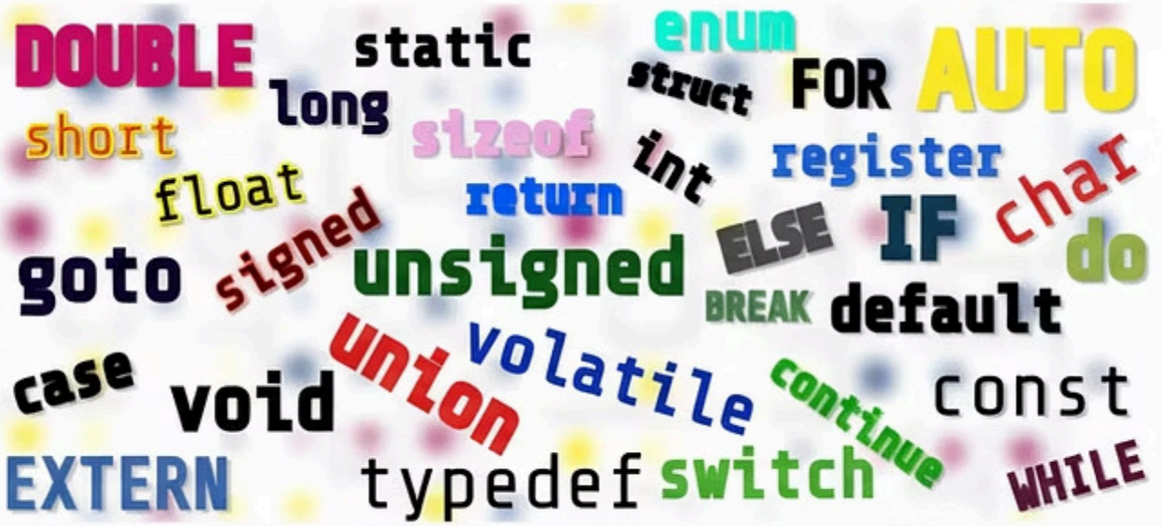
  Translates your C code into Assembly Language—a low-level, human-readable form.

- ## Assembler

  Converts Assembly into Machine Code—the binary language computers understand.

- ## Linker

  Connects your code with library files to create the final executable (.exe) program.



Keywords vs. Variables: Understanding the Rules

Keywords

Reserved words owned by the C language. You **cannot** use them as variable names because they have special meanings.

- `int` - integer data type
- `return` - exit function
- `float` - decimal data type
- `if` - conditional statement

Variables

Named containers you create to store data. You choose the names following C's naming rules.

- `age` - stores a person's age
- `marks` - stores test scores
- `salary` - stores income amount
- `grade` - stores letter grade

Keywords :

List Of Keywords In C & Their Purpose

break	case	char	const	auto	short	struct	switch
double	int	else	enum	float	continue	sizeof	default
extern	for	do	goto	if	typedef	union	void
static	signed	long	register	return	unsigned	volatile	while

Data Types: Choosing the Right Container

Data types tell the compiler what kind of information you're storing and how much memory to allocate. Think of them as different-sized boxes for different items.

Data Type	Description	Format	Examples
int	Whole numbers (integers)	%d	10, -5, 0
float	Decimal numbers	%f	3.14, 98.6
char	Single character	%c	'A', 'z', '$'
double	Large/precise decimals	%lf	3.14159265

c

```c
#include <stdio.h>

int main() {
    int age = 20;
    char grade = 'A';
    float height = 5.9;

    printf("I am %d years old.\n", age);
    printf("My Grade is %c.\n", grade);
    printf("My Height is %.1f.\n", height);

    return 0;
}
```

Output

css

```
I am 20 years old.
My Grade is A.
My Height is 5.9.
```

codechef problem QR

Problem 1

Practice Printing in C

Problem 2

Hello World!! in C

Problem 3

Hello World! Question in C

Problem 4

Valid Code Block Usage in C

Your Homework Challenge

1

Set Up Your Environment

Install a code editor on your personal laptop. Recommended options: VS Code,.

2

Homework 1

Create a C program that prints two things: the first letter of your name and your year of birth using appropriate data types.

3

Homework 2

Create a C program that prints your name, age, class, cgpa, year, branch using appropriate data types.