Problem Solving With C

SESSION 1: INTRODUCTION TO C & BASIC DATA TYPES

Chapter 1: What is Programming?

Understanding Programming in Simple Terms

Programming is like giving instructions to a computer to perform tasks. Just like you would give someone directions to reach a destination, you give a computer step-by-step instructions to solve problems.

It is a broader activity that includes not only coding but also **problem-solving, planning, designing algorithms, and testing**.

Real-Life Example: Think of a recipe for making tea: 1. Boil water 2. Add tea leaves 3. Add sugar 4. Add milk 5. Strain and serve

Similarly, a program is a recipe for the computer to follow!

Coding

Writing actual lines of code (instructions) in a specific programming language (C, C++, Java, Python, etc.).

Program vs Process

In Computer Science, there are two fundamental terms in operating system: Program and Process.

Program is a set of instructions written to perform a task, stored in memory.

A process is the active execution of a program, using system resources like CPU and memory.

In other words, a program is static, a process is dynamic, representing the program in action.

What is C Language?

C is a programming language created in 1972 by Dennis Ritchie. It's like a language we use to talk to computers. Just as we use English to communicate with people, we use C to communicate with computers.

Why Learn C? - Foundation for other languages (C++, Java, Python concepts) - Fast and efficient - Used in operating systems, games, embedded systems - Helps understand how computers work

Chapter 2: Features and Structure of a C Program

Basic Structure of Every C Program

```
#include <stdio.h>    // Header file - like a dictionary for the computer

int main()           // Starting point of program
{
    // Your code goes here
    printf("Hello World!");
    return 0;        // Program ended successfully
}
```

Understanding Each Part:

1. Header Files (#include <stdio.h>)

What it is: Think of header files as toolboxes. When you write `#include <stdio.h>`, you're telling the computer "I need the input/output toolbox."

Common Question: "Why do we need stdio.h?" **Answer:** stdio.h contains definitions for input/output functions like printf() and scanf(). Without it, the computer won't understand these commands.

2. main() Function

What it is: Every C program MUST have a main() function. It's the entry point - where the computer starts reading your program.

Common Question: "Why int before main?" **Answer:** 'int' means the function returns an integer value. main() returns 0 to indicate successful execution.

3. Curly Braces

These mark the beginning and end of a block of code. Everything between { } belongs together.

4. Statements

Each instruction ends with a semicolon (;). It's like a full stop in English.

Complete First Program Example:

```
#include <stdio.h>

int main()
{
    printf("Welcome to C Programming!\n");
    printf("This is my first program.\n");
    return 0;
}
```

Output:

```
Welcome to C Programming!
This is my first program.
```

Chapter 3: Data Types - The Building Blocks

What are Data Types?

Data types tell the computer what kind of information you want to store. It's like choosing the right container: - Use a bottle for water - Use a box for books - Use a wallet for money

Similarly, we use different data types for different kinds of data.

Primary Data Types in C

1. int (Integer)

Purpose: Store whole numbers (no decimals) **Size:** 4 bytes (32 bits) **Range:** -2,147,483,648 to 2,147,483,647

```c
int age = 20;
int temperature = -5;
int score = 100;
```

Common Question: "What happens if I store 3.14 in int?" **Answer:** It will store only 3, the decimal part is lost.

2. float (Floating Point)

Purpose: Store decimal numbers **Size:** 4 bytes **Precision:** 6-7 decimal digits

```c
float price = 99.99;
float pi = 3.14159;
float weight = 65.5;
```

3. double

Purpose: Store decimal numbers with more precision **Size:** 8 bytes **Precision:** 15-16 decimal digits

```c
double precise_value = 3.141592653589793;
double scientific = 1.23e-10;  // Scientific notation
```

Common Question: "When to use float vs double?" **Answer:** Use float for normal calculations (saves memory). Use double when you need high precision (scientific calculations).

4. char (Character)

Purpose: Store single characters **Size:** 1 byte

```
char grade = 'A';
char symbol = '@';
char digit = '5';  // Note: '5' is character, not number
```

Important: Characters are enclosed in single quotes ’ ’

Visual Memory Representation

```
Memory Layout Example:
```

int	float	double	char
4 bytes	4 bytes	8 bytes	1 byte
age	price	pi	grade
20	99.99	3.14159	'A'

Chapter 4: Variables - Storage Containers

What is a Variable?

A variable is like a labeled box where you store data. You can: - Put something in the box (assign value) - See what’s in the box (read value) - Change what’s in the box (modify value)

Rules for Naming Variables

MUST Follow:

1. Start with letter or underscore (_)
2. Can contain letters, digits, underscore
3. No spaces allowed
4. Case sensitive (age ≠ Age)
5. Cannot use keywords

Examples:

```
// VALID Variable Names
int studentAge;
float total_price;
char grade1;
double _result;

// INVALID Variable Names
int 2ndPlace;       // Cannot start with digit
float total-price; // Cannot use hyphen
char my grade;      // Cannot have space
int float;          // Cannot use keyword
```

Variable Declaration and Initialization

Method 1: Declare then Initialize

```
int age;          // Declaration
age = 20;         // Initialization
```

Method 2: Declare and Initialize Together

```
int age = 20;     // Declaration + Initialization
```

Multiple Variables

```
int x, y, z;           // Declare multiple
int a = 5, b = 10;     // Declare and initialize multiple
```

Common Mistakes and Solutions

Mistake 1: Using uninitialized variables

```
int x;
printf("%d", x);  // WRONG! x has garbage value
```

Solution: Always initialize variables

```
int x = 0;
printf("%d", x);  // Correct
```

Mistake 2: Type mismatch

```
int age = "twenty";  // WRONG! Cannot store text in int
```

Solution: Use correct data type

```
int age = 20;        // Correct
```

Chapter 5: Hello World Program - Complete Understanding

The Traditional First Program

```
#include <stdio.h>

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

Step-by-Step Execution:

1. **Preprocessor** reads #include and loads stdio.h
2. **Compiler** finds main() function
3. **Executes** printf() statement
4. **Displays** “Hello, World!” on screen

5. **Returns** 0 to operating system

Variations and Learning:

```c
#include <stdio.h>

int main()
{
    // Different ways to print
    printf("Hello, World!\n");           // With new line
    printf("Hello, ");
    printf("World!");                    // Multiple printf

    printf("\nWelcome to C!\n");        // \n creates new line
    printf("Line 1\nLine 2\nLine 3");   // Multiple lines

    return 0;
}
```

Understanding printf() Function

Syntax: `printf("format string", arguments);`

```c
printf("My age is %d years", 20);        // %d for integer
printf("Price is %.2f", 99.99);          // %.2f for float (2 decimals)
printf("Grade: %c", 'A');                // %c for character
```

Escape Sequences - Special Characters

Sequence	Meaning	Example
	New line	printf(“Line1”);
Tab space	printf(“Name”);	
\	Backslash	printf(“C:\folder”);
"	Double quote	printf(“He said "Hello"”);
'	Single quote	printf(“It's working”);

Chapter 6: Program Compilation Process

What Happens When You Run a C Program?

```
Your Code (.c file)
        ↓
[1. Preprocessor] - Handles #include, #define
        ↓
Expanded Code
        ↓
[2. Compiler] - Converts to assembly code
        ↓
Assembly Code
        ↓
```

```
[3. Assembler] - Converts to machine code
        ↓
Object Code (.obj)
        ↓
[4. Linker] - Links with libraries
        ↓
Executable File (.exe)
        ↓
[5. Loader] - Loads into memory
        ↓
Program Runs!
```

Understanding Each Stage:

1. Preprocessing

- Includes header files
- Replaces macros
- Removes comments

2. Compilation

- Checks syntax errors
- Converts to assembly language

3. Assembly

- Converts assembly to machine code (0s and 1s)

4. Linking

- Combines your code with library functions
- Creates final executable

Common Compilation Errors:

Syntax Error:

```
printf("Hello")  // Missing semicolon
```

Fix: Add semicolon: `printf("Hello");`

Missing Header:

```
int main()
{
    printf("Hello");  // Error: printf not defined
}
```

Fix: Add `#include <stdio.h>`

Chapter 7: Activities and Practice Programs

Activity 1: Personal Information Display

```c
#include <stdio.h>

int main()
{
    // Declare variables
    char initial = 'J';
    int age = 20;
    float height = 5.8;

    // Display information
    printf("Personal Information\n");
    printf("===================\n");
    printf("Initial: %c\n", initial);
    printf("Age: %d years\n", age);
    printf("Height: %.1f feet\n", height);

    return 0;
}
```

Chapter 8: Keywords Reference Table

Complete List of 32 Keywords in C

Category	Keywords	Purpose
Data Types	int, float, double, char	Define variable types
	short, long, signed, unsigned	Modify data types
	void	No value/type
Control Flow	if, else	Conditional execution
	switch, case, default	Multiple conditions
	for, while, do	Loops
	break, continue	Loop control
	goto	Jump to label
Storage	auto, register, static, extern	Storage classes
Others	const	Constant values
	sizeof	Size of data type
	typedef	Create type alias
	return	Return from function
	struct, union, enum	User-defined types
	volatile	Special variable

Important Notes:

- Keywords are reserved words - you CANNOT use them as variable names
- All keywords are in lowercase
- They have special meaning in C