B.Sc. (I.T.) / M.Sc. (I.T.) 1s Semester
Course: 104: Fundamentals of Programming Using C-1
Unit 3: Introduction to C language • 3.1 Overview of C • 3.2 Constants, Variables and Data types • 3.3 Operators and expressions • 3.4 Simple Assignment statement • 3.5 Basic Input/Output Statements • 3.6 Decision Making Statements • 3.7 Looping • 3.8 Nested Control Structures

3.1 Overview of C

What is C?

C is a **general-purpose, structured, and procedural programming language** developed by **Dennis Ritchie at Bell Labs in 1972**.

It is widely used because of its:

- **Portability** (write once → compile anywhere)
- **Speed** (close to hardware, minimal overhead)
- **Flexibility** (systems programming to application development)
- **Rich set of operators**
- **Support for low-level programming concepts**

Key Features

1. **Simple & Efficient** — Minimal keywords, fast execution.
2. **Structured Programming** — Programs are divided into functions.
3. **Portability** — C code runs on different machines with minor/no change.
4. **Rich Library** — Standard functions for I/O, math, string operations.
5. **Extensible** — Programmers can define their own functions.
6. **Memory Control** — Supports pointers for direct memory management.

Applications of C

- Operating systems (UNIX, Linux components)
- Compilers & interpreters
- Embedded systems
- System-level utilities
- IoT & microcontrollers
- Game engines
- Database systems

Why Should Students Learn C First?

- Teaches **logic building**
- Strengthens **memory concepts, loops, and conditions**
- Helps in understanding **data structures & algorithms**
- Builds base for advanced languages: C++, Java, Python

3.2 Constants, Variables and Data Types

Constants

Values that never change during program execution.

Types of Constants

1. Integer constants → 10, 0, -45
2. Real constants → 3.14, -0.5
3. Character constants → 'A', '1', '+'
4. String constants → "Hello", "C Language"
5. Symbolic constants → Defined using #define
   Example: #define PI 3.14

Variables

A variable is a named memory location used to store data.

Rules for Naming Variables

- Only letters, digits, and underscore
- Cannot start with a digit
- Case-sensitive
- No keywords allowed (int, printf, etc.)

Examples:

total, marks1, student_name, _count

Data Types

Primary Data Types

Data Type	Size	Meaning
Int	2/4 bytes	Integer values
Float	4 bytes	Decimal values (single precision)
Double	8 bytes	Decimal values (double precision)
Char	1 byte	Single character

Derived Data Types

- Arrays
- Pointers
- Functions
- Structures
- Unions

Type Modifiers

- short
- long

- signed
- unsigned

Example:
unsigned int age;

3.3 Operators and Expressions

Operators are symbols that perform operations on variables/values.

Types of Operators
1. Arithmetic Operators

+ - * / %

2. Relational Operators

> < >= <= == !=

3. Logical Operators

&& || !

4. Assignment Operators

= += -= *= /= %=

5. Increment/Decrement

++ --

6. Conditional (Ternary)

condition ? expression1 : expression2;

7. Bitwise Operators

& | ^ << >> ~

8. Special Operators

sizeof()

, (comma)

& (address of)

*** (pointer dereference)**

Expressions
Combination of variables, constants, and operators.
Example:
total = marks1 + marks2 * 2;
Execution follows operator precedence & associativity rules.

3.4 Simple Assignment Statement

Used to assign a value.
Syntax:
variable = expression;
Examples
a = 10;
b = a + 3;
c = b * 2 - a;
Assignment occurs right → left.

3.5 Basic Input/Output Statements

C uses stdio.h library.
Input: scanf()
scanf("format", &variable);
Examples:
scanf("%d", &age);
scanf("%f", &salary);
scanf("%s", name);

Output: printf()
printf("Name: %s", name);
printf("Total = %d", total);

3.6 Decision Making Statements

Used to execute code based on conditions.

1. Simple if

```
if (marks >= 40)
  printf("Pass");
```

2. if-else

```
if (age >= 18)
  printf("Eligible");
else
  printf("Not Eligible");
```

3. Nested if

```
if (a > b) {
  if (a > c)
    printf("A is greatest");
}
```

4. Ladder if-else

```
if (marks >= 75)
  printf("Distinction");
else if (marks >= 60)
  printf("First Class");
else if (marks >= 40)
  printf("Pass");
else
  printf("Fail");
```

5. switch-case

Used for menu-driven programs.

```
switch(choice) {
  case 1: printf("Add"); break;
  case 2: printf("Subtract"); break;
  default: printf("Invalid");
}
```

3.7 Looping (Iterative Control)

Loops repeat a block of code.

1. while loop

Entry-controlled loop.

```
while (i <= 10) {
  printf("%d", i);
  i++;
}
```

2. do-while loop
Exit-controlled loop.

```
do {
  printf("%d", i);
  i++;
} while (i <= 10);
```

3. for loop
Most commonly used.

```
for (i = 1; i <= 10; i++)
  printf("%d", i);
```

4. break and continue

- break → exit loop
- continue → skip current iteration

3.8 Nested Control Structures

Using one control structure inside another.

Examples
Nested Loop

```
for (i = 1; i <= 5; i++) {
  for (j = 1; j <= i; j++) {
    printf("* ");
  }
  printf("\n");
}
```

Nested Decisions

```
if (x > 0) {
   if (y > 0)
      printf("Quadrant 1");
}
```

Loop inside if

```
if (n > 0) {
   for (i = 1; i <= n; i++)
      printf("%d", i);
}
```

Nested structures are essential for patterns, matrix operations, ATM simulation, etc.