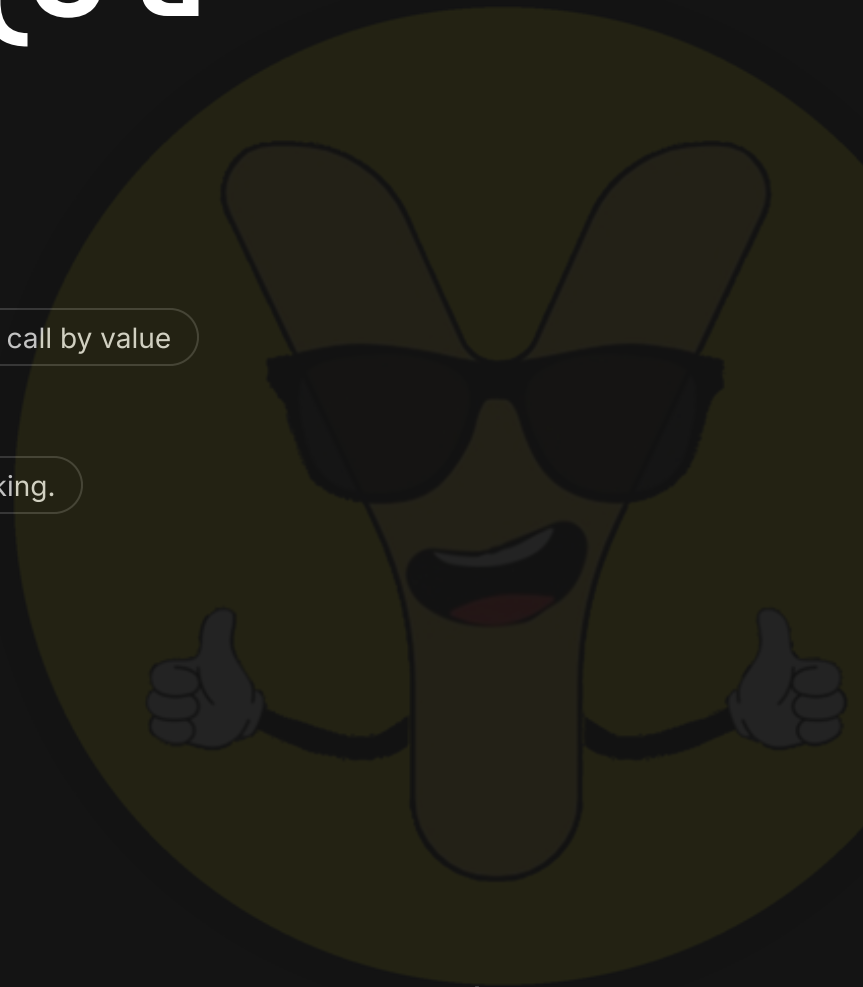
AKTU · 1ST YEAR · SEMESTER 1 · ACS101

Programming Languages (C & Python)

Unit 2 — Decision Making and Control Flow (C)

- Control Statements
- Loops — for, while, do-while; break and continue
- Functions — definition, call, return; call by value
- Recursion — factorial and Fibonacci (simple examples only)
- Real connect: Control flow is the brain of every program — from traffic signal logic to drone autopilot decision-making.



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Real Connect: The Brain of Every Program

Control flow dictates the execution path of software, mirroring real-world decision-making systems.

- Traffic signal logic: Checks timer values and sensor inputs to switch red, yellow, and green states sequentially.
- Drone autopilot: Continuously runs loops to read altitude, adjust motor speeds, and avoid obstacles dynamically.
- Without control flow, programs would execute blindly line by line without adapting to input data or conditions.
- It transforms static code into responsive, intelligent software capable of handling complex operational logic.

Control Statements in C

CONTROL STATEMENT

Programming constructs that alter the sequential execution of statements based on specific logical conditions.

- Decision making statements: if, if-else, nested if-else, and switch-case.
- Selection statements evaluate boolean expressions to branch execution into different logic blocks.
- Essential for building dynamic applications that require branching paths based on runtime user input.

Conditional Branching: if, if-else, and switch

C provides multiple constructs to handle conditional execution depending on the complexity of the choices.

- Simple if and if-else
  - Executes a block if the condition evaluates to true (non-zero).
  - The else block provides an alternative path when the condition is false.
- Nested if-else
  - Placing an if-else construct inside another if or else block.
  - Used for multi-tiered decision making where conditions depend on prior outcomes.
- Switch-case statement
  - Multi-way branch statement replacing long if-else-if ladders.
  - Matches a variable against constant integral or character labels.

Control Statements Example

```c
#include <stdio.h>
int main() {
    int marks = 75;
    if (marks ⩾ 80) {
        printf("Grade A");
    } else if (marks ⩾ 60) {
        printf("Grade B");
    } else {
        printf("Grade C");
    }
    return 0;
}
```



- Demonstrates sequential conditional evaluation using if-else-if ladders.
- Only one block executes based on the first condition that evaluates to true.

Control Flow Architecture



Execution flow in conditional branching constructs.

Loops in C

ITERATION STATEMENT (LOOP)

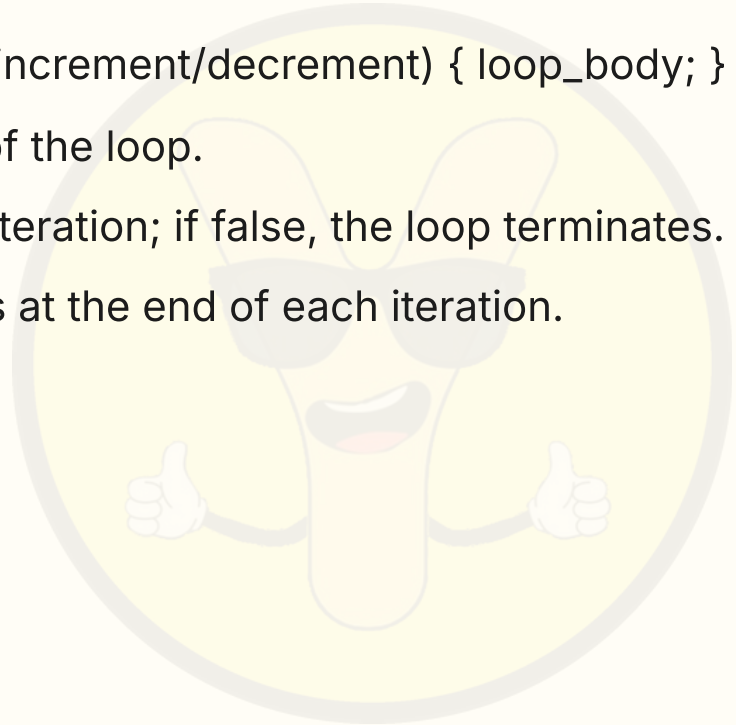
A programming structure that repeats a block of instructions automatically until a termination condition is met.

- Eliminates code redundancy by automating repetitive tasks.
- Consists of initialization, condition checking, loop body execution, and updating the loop variable.
- C provides three primary loop constructs: for, while, and do-while.

The for Loop

Designed for counter-controlled iteration where the exact number of repetitions is known beforehand.

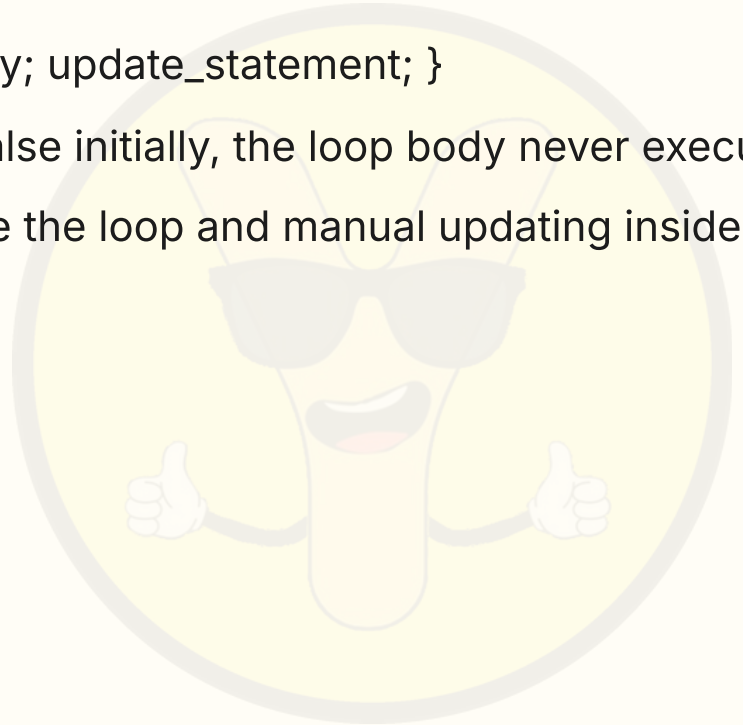
- Syntax: for(initialization; condition; increment/decrement) { loop_body; }
- Initialization runs once at the start of the loop.
- Condition is checked before every iteration; if false, the loop terminates.
- Increment or update statement runs at the end of each iteration.



The while Loop

An entry-controlled loop used when the number of iterations is not known in advance.

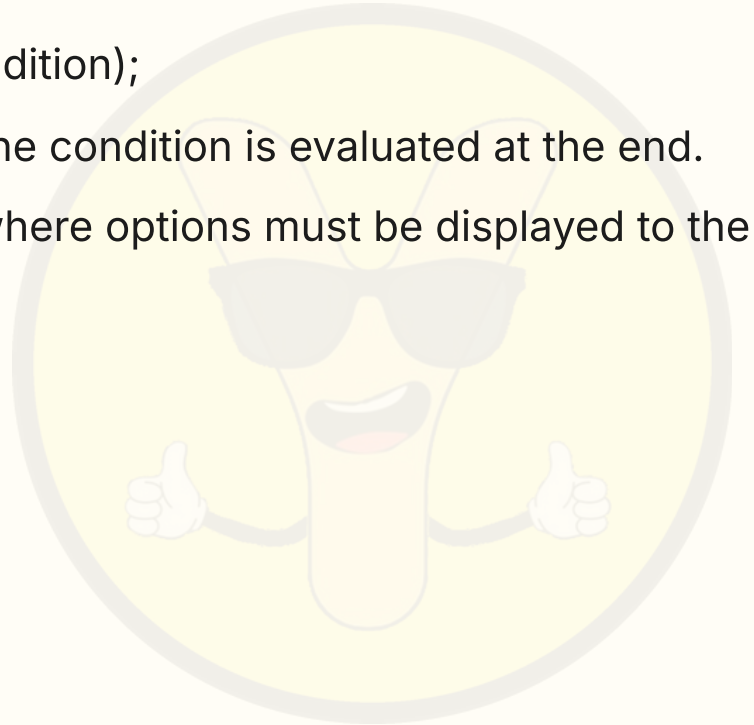
- Syntax: while(condition) { loop_body; update_statement; }
- Condition is evaluated first. If it is false initially, the loop body never executes.
- Requires manual initialization before the loop and manual updating inside the loop body to avoid infinite execution.



The do-while Loop

An exit-controlled loop that guarantees at least one execution of the loop body.

- Syntax: do { loop_body; } while(condition);
- The body executes first, and then the condition is evaluated at the end.
- Useful for menu-driven programs where options must be displayed to the user at least once before checking input.



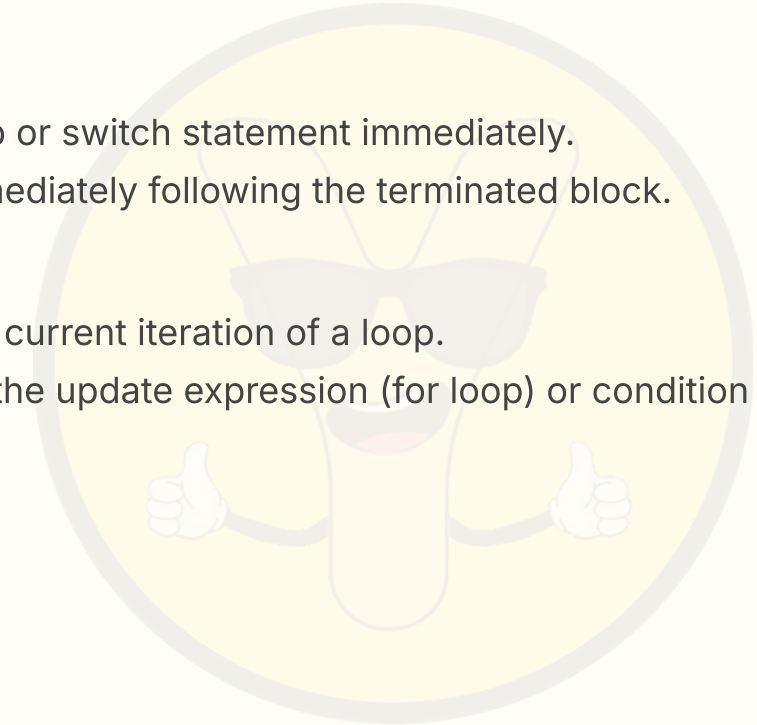
Comparison of Loops in C

Feature	for loop	while loop	do-while loop
Control Type	Entry-controlled	Entry-controlled	Exit-controlled
Minimum Executions	0	0	1
Use Case	Known iterations	Unknown iterations	Menu-driven tasks
Syntax Compactness	High (single line header)	Medium	Medium

Jump Statements: break and continue

Statements used to unconditionally transfer program control within loops and switch blocks.

- break statement
  - Terminates the nearest enclosing loop or switch statement immediately.
  - Control resumes at the statement immediately following the terminated block.
- continue statement
  - Skips the remaining statements in the current iteration of a loop.
  - Forces the control to jump directly to the update expression (for loop) or condition check (while/do-while loop).



Functions in C

FUNCTION

A self-contained block of code that performs a specific, well-defined task and can be called multiple times.

- Promotes code reusability, modularity, and easier debugging.
- Divided into declaration (prototype), definition, and function call.
- Library functions (like printf, sqrt) are pre-written; user-defined functions are created by programmers.

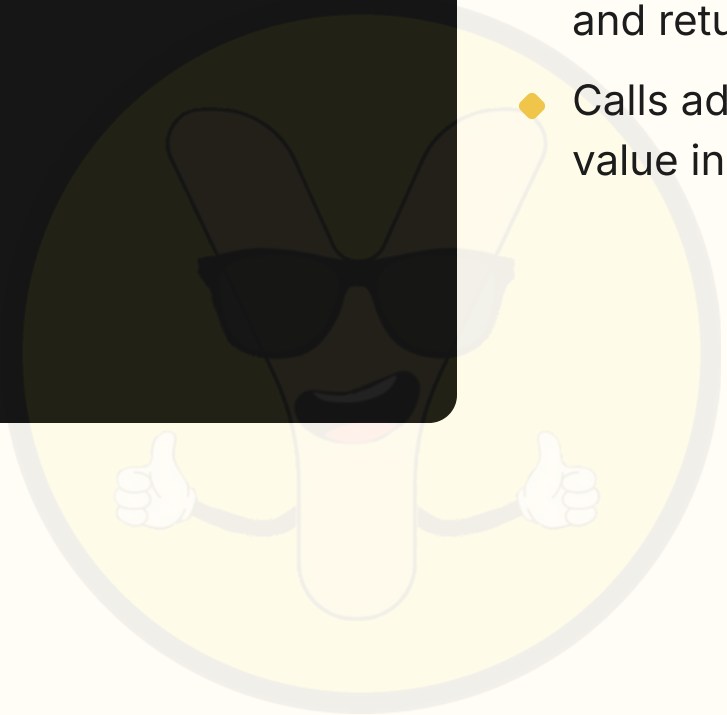
Function Anatomy: Definition, Call, Return

Understanding the lifecycle and syntax of user-defined functions in C.

- Definition
  - Contains the actual body of the function along with return type and parameters.
  - Syntax: return_type function_name(parameters) { statements; }
- Call
  - Invoking the function by passing arguments from another function (e.g., main).
  - Transfers execution control to the function definition.
- Return
  - Sends the computed result back to the caller using the return keyword.
  - Terminates the function execution immediately.

Function Example

```c
#include <stdio.h>
int add(int a, int b) {
    return a + b;
}
int main() {
    int result = add(5, 3);
    printf("Sum: %d", result);
    return 0;
}
```



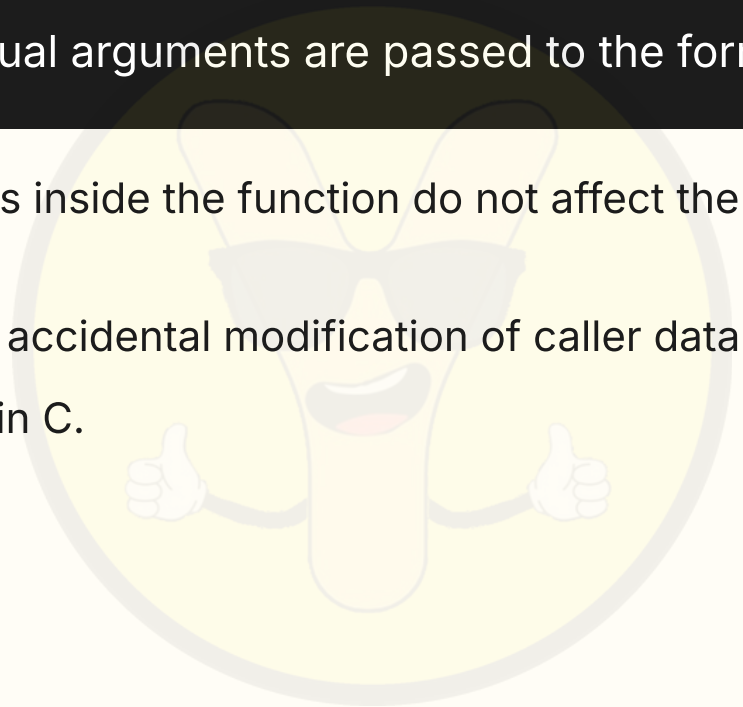
- Defines an add function taking two integers and returning their sum.
- Calls add inside main, capturing the return value in a variable.

Parameter Passing: Call by Value

CALL BY VALUE

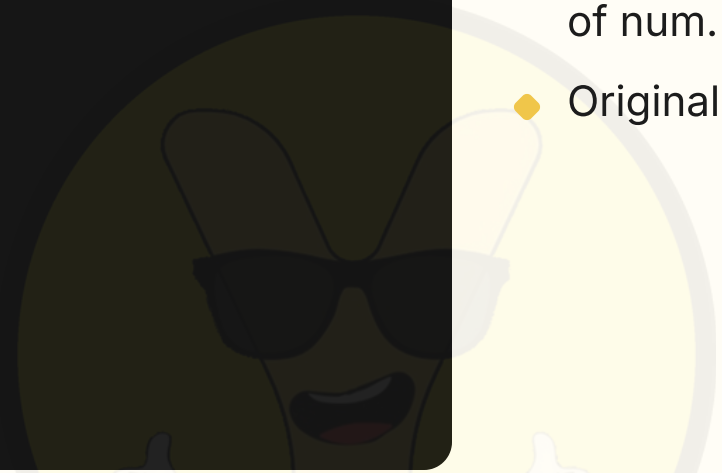
A mechanism where copies of actual arguments are passed to the formal parameters of a function.

- Changes made to formal parameters inside the function do not affect the original variables in the calling function.
- Ensures data security and prevents accidental modification of caller data.
- Default parameter passing method in C.



Call by Value Demonstration

```c
#include <stdio.h>
void modify(int x) {
    x = 100;
}
int main() {
    int num = 10;
    modify(num);
    printf("%d", num);
    return 0;
}
```



- Output is 10, because modify receives a copy of num.
- Original num in main remains unchanged.

Recursion

RECURSIVE FUNCTION

A programming technique where a function calls itself directly or indirectly to solve a smaller instance of the same problem.

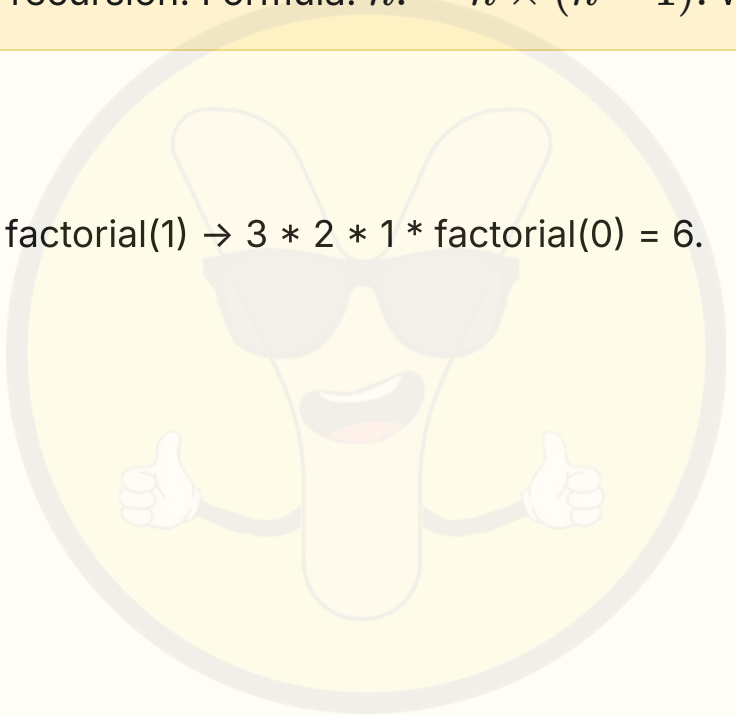
- Must include a base condition to terminate recursive calls and prevent stack overflow.
- Consists of the recursive step (moving towards the base case) and the base case (stopping condition).
- Alternative to iterative loops for problems with self-similar subproblems.

Recursion Example: Factorial

Calculate the factorial of a number using recursion. Formula: $n! = n \times (n-1)!$ with base case $0! = 1$.

1. Check base case: if n == 0, return 1.
2. Recursive step: return n * factorial(n - 1).
3. Trace for n = 3: 3 * factorial(2) → 3 * 2 * factorial(1) → 3 * 2 * 1 * factorial(0) = 6.

Factorial of 3 is 6.

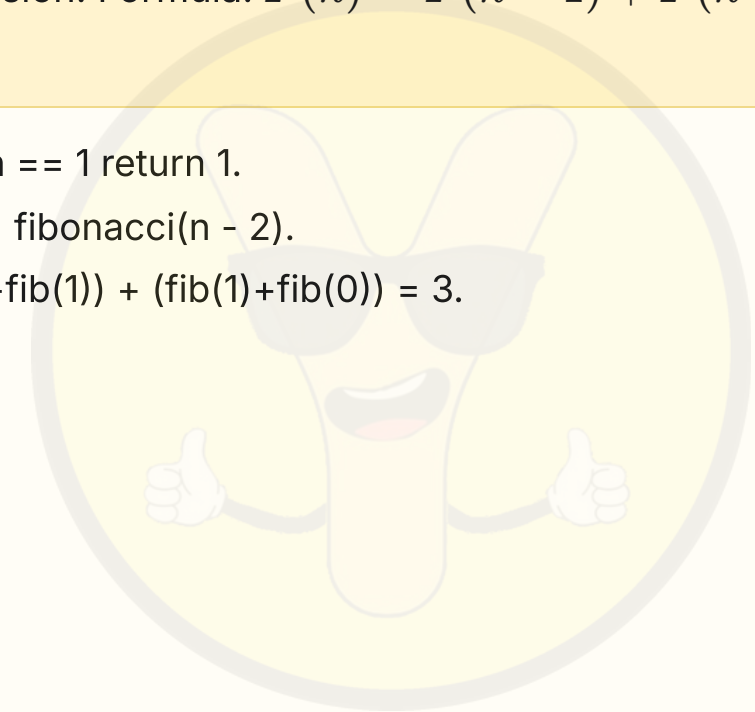


Recursion Example: Fibonacci Sequence

Find the nth Fibonacci term using recursion. Formula: $F(n) = F(n-1) + F(n-2)$ with base cases $F(0) = 0, F(1) = 1$.

1. Check base case: if n == 0 return 0, if n == 1 return 1.
2. Recursive step: return fibonacci(n - 1) + fibonacci(n - 2).
3. Trace for n = 4: fib(3) + fib(2) = (fib(2)+fib(1)) + (fib(1)+fib(0)) = 3.

The 4th Fibonacci term is 3.



Exam Corner

- **Q1** Differentiate between entry-controlled and exit-controlled loops with examples of each in C.
- **Q2** Explain the working of switch-case statement with a suitable syntax and code example.
- **Q3** What is the difference between break and continue statements? Explain with code.
- **Q4** Explain call by value mechanism in C functions with a suitable program.
- **Q5** What is recursion? Write a recursive function to calculate the factorial of a given number.
- **Q6** Compare while and do-while loops. Give scenarios where do-while is preferred.
- **Q7** Write a C program to check whether a given number is prime or not using loops.

Unit 2 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 3 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

