**Ques 1. What is computational thinking? Describe various steps involve in it?**

**Ans.** Computational Thinking is a **problem-solving process** that involves breaking down complex problems into manageable steps and using logical reasoning.

Computational thinking helps in understanding and **developing efficient solutions** for problems by following systematic methods.

1. **Decomposition:** Decomposition is the process of breaking down a complex problem or system into smaller, more manageable parts. By focusing on individual components, it becomes easier to understand and solve.

**Example:** Imagine you are asked to organize a school event. Instead of handling everything at once, break it down:

* **Venue Selection**
* **Invitations**
* **Food Arrangements**
* **Decoration**
* **Entertainment**

1. **Pattern Recognition:** Pattern recognition involves identifying similarities or patterns within a problem. Allows for understanding of repetitive elements, which can be used for optimization.

**Example:**

* If you're solving puzzles, like a Rubik's cube, you may notice patterns in the colors and their positions. Once you've identified a pattern, you can apply a set of moves that often work to solve the puzzle.
* In coding, **loops** are used because repetitive tasks can be automated by recognizing that the same block of code needs to run multiple times.

1. **Abstraction:** Abstraction is the process of focusing only on the relevant details and ignoring unnecessary information.

**Example:**

* When you're driving, you don’t need to think about how the engine works. You only care about **steering, speed control, and direction**—the relevant information needed to get you to your destination.

1. **Algorithm Design:** **Algorithm Design** is the process of defining a clear, step-by-step procedure or set of instructions to solve a specific problem. The goal of an algorithm is to take inputs (data) and transform them into the desired outputs (solutions) in an efficient and organized way.

Algorithms can be applied to a wide range of problems, from simple tasks like adding two numbers, to complex tasks like sorting a list or even processing huge amounts of data.

**Real-life example:** In coding, a simple algorithm to sort a list of numbers from lowest to highest could be:

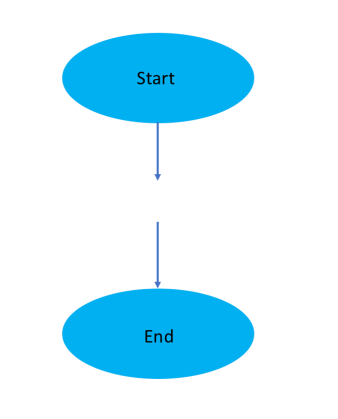
1. Compare the first two numbers.
2. Swap them if they are in the wrong order.
3. Move to the next number and repeat the process.
4. Continue until the list is sorted.

**Ques 2. Describe the key difference between modular programming language and object-oriented programming language?**

**Ans.**

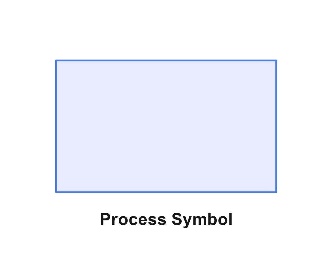
|  |  |  |
| --- | --- | --- |
| **S.No.** | **Modular Programming Language** | **Object-Oriented Programming Language** |
| 1. | Program is divided into **modules** (functions or procedures). | Program is divided into **objects** (real-world entities). |
| 2. | Focus is on **functions** and procedures. | Focus is on **data** and objects. |
| 3. | Data is **shared** among functions. | Data is **encapsulated** inside objects and accessed through methods. |
| 4. | Does not support features like inheritance and polymorphism. | Supports **inheritance**, **polymorphism**, and **encapsulation**. |
| 5. | Example languages: C, Modula-2 | Example languages: Java, C++, Python |
| 6. | Less reusable and harder to maintain for large projects. | More reusable and easier to maintain through objects and classes. |
| 7. | No concept of classes and objects. | Uses **classes** and **objects** as the main building blocks. |

**Ques 3. What are the various flowchart symbols? Explain with diagram? Draw flowchart to calculate and print the sum of First N prime numbers?**

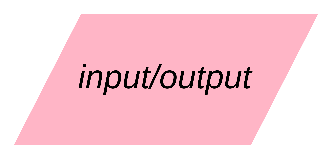
**Ans.** A **flowchart** is a diagram that shows the **step-by-step flow of a process** or **algorithm** using different **symbols**.  
Each symbol represents a specific type of action or step.

**1. Start / End Symbol (Oval)**

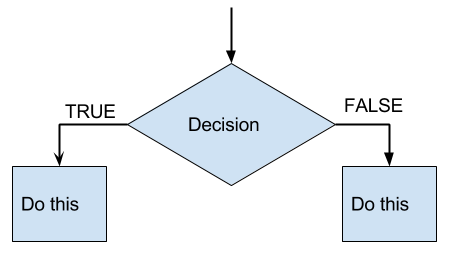
* This symbol is used to indicate **where the flowchart begins and ends**.
* The word **“Start”** is written inside the oval at the beginning and **“End”** at the termination.
* Every flowchart must have one **start** and at least one **end**.
* **Example:** Start of a program or when the output is displayed and the program finishes.

**2. Process Symbol (Rectangle)**

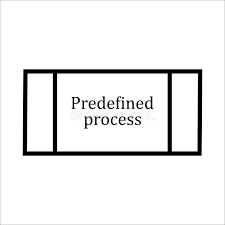
* This symbol shows a **process**, **instruction**, or **operation**.
* It is used to represent steps like calculations, data processing, or assignments.
* **Example:** SUM = A + B or any arithmetic operation.

**3. Input / Output Symbol (Parallelogram)**

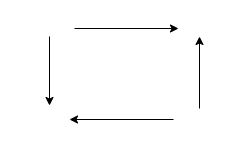
* This symbol is used to represent **input** or **output** operations.
* **Input** means taking data from the user, and **output** means displaying data to the user.
* **Example:** “Enter two numbers” or “Display the result”.

**4. Decision Symbol (Diamond)**

* It represents a **decision-making step** in the flowchart.
* This symbol is always followed by **two or more branches** depending on the answer (e.g., YES/NO or TRUE/FALSE).
* **Example:** “Is A > B?” If yes, follow one path; if no, follow another.

**5. Predefined Process Symbol (Rectangle with Double Lines)**

* This symbol indicates a **sub-process** or **subroutine** that is defined elsewhere.
* It helps to avoid repeating the same set of steps multiple times.
* **Example:** calling a function like Sort() or Calculate().



**7. Flow Lines (Arrows)**

* Arrows are used to **show the direction of flow** from one step to another.
* They connect all the symbols together and guide the order in which steps are executed.
* **Example:** From “Start” → “Input” → “Process” → “Output” → “End”.

**Draw flowchart to calculate and print the sum of First N prime numbers?**

**Ques 4. What do you mean by the scope of the variable?**

**Ans.** In **C**, the **scope of a variable** means **the part of the program where the variable can be accessed or used**. Scope tells you where in the program a variable is “visible” or “alive.”

There are mainly **two types of variable scope** in C—

a) **Local Scope:**

* A variable declared inside a function or a block (like { }) has local scope.
* It can only be used inside that function or block.
* It cannot be accessed outside it.
* It is created when the block starts and destroyed when the block ends.
* **Example:**

#include <stdio.h>

int main() {

int x = 10; // local variable

printf("%d", x);

return 0;

}

**b) Global Scope:**

* A variable declared **outside all functions** has **global scope**.
* It can be **used in any function** of the program.
* It exists **for the entire program**.
* **Example:**

#include <stdio.h>

int x = 10; // global variable

int main() {

printf("%d", x); // accessible here

return 0;

}

void display() {

printf("%d", x); // accessible here too

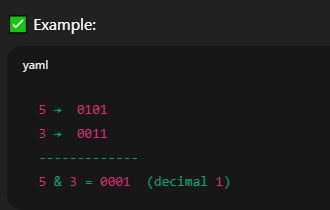
}

**Ques 5. What are the bitwise operators and how bitwise operators being different from other operators? Explain with a proper example to demonstrate the working of all the bitwise operators?**

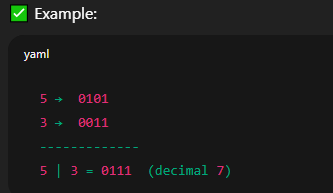
**Ans.**

* Bitwise operators work **directly on binary representations** of integers.
* They can only be applied to **integer types** (int, short, long, char), not floating-point types.
* Bitwise operations are **very fast** because they are performed at the hardware level.
* These operators are often used in **system programming**, **embedded systems**, **networking**, **encryption**, and **bit masking**.
* The **result of a bitwise operation** is always an integer.
* Bitwise operators do **not perform logical comparison**; they only check or change bits.
* Here are the **main bitwise operators**:

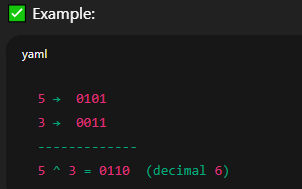
1. **Bitwise AND (&):**

* Compares each bit of two numbers.
* The result bit is **1 only if both bits are 1**, otherwise 0.

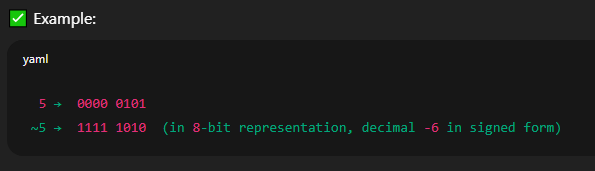
1. **Bitwise OR (|):**

* Compares each bit of two numbers.
* The result bit is **1 if at least one bit is 1**, otherwise 0

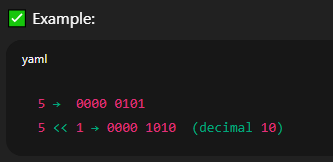
1. **Bitwise XOR (^)** **(Exclusive OR):**

* Compares each bit of two numbers.
* The result bit is **1 if the bits are different**, and 0 if they are the same.

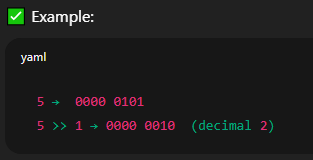
1. **Bitwise NOT (~)** **(One’s Complement):**

* Inverts all the bits of the number.
* 1 becomes 0 and 0 becomes 1.

1. **Left Shift (<<):**

* Shifts all bits to the **left** by a specified number of positions.
* Each left shift multiplies the number by 2.

1. **Right Shift (>>):**

* Shifts all bits to the **right** by a specified number of positions.
* Each right shift divides the number by 2.
* **how bitwise operators being different from other operators?**
* Bitwise operators are **different from other operators** in C because they work **directly on the binary bits** of numbers, not on their mathematical values like arithmetic or logical operators do.
* Bitwise operators operate **bit by bit** (0s and 1s) of integer data.

Other operators (like +, -, \*, /) work on the **entire value** of the variable.

* Bitwise operators work only on **integer data types** (int, char, long, etc.).

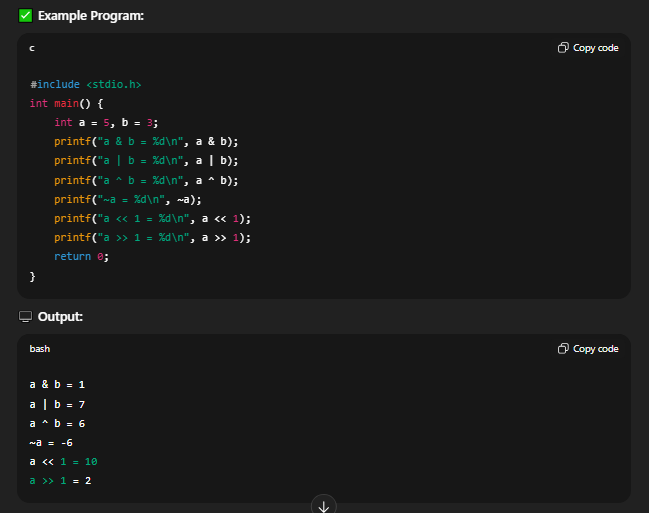
Other operators can work on integers, floats, doubles, etc.

* Bitwise operations are **very fast** because they are done at the **processor level**.

Arithmetic operations may take more time comparatively.

* Bitwise operators don’t check if a number is “true” or “false”; they just manipulate **each bit**.

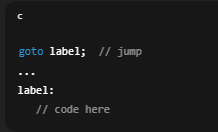
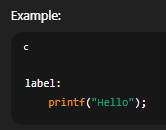
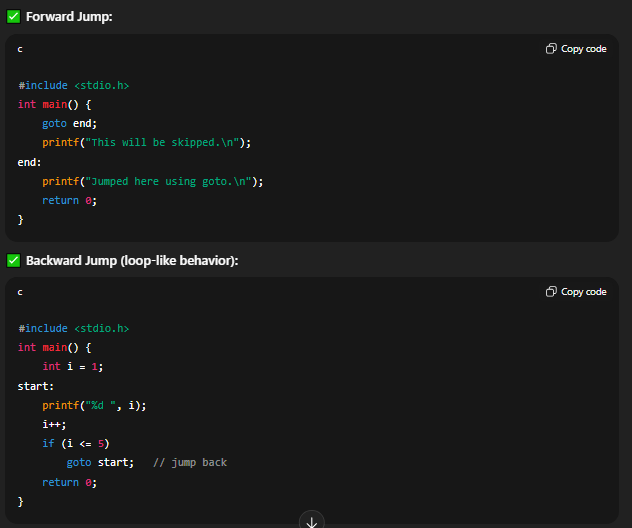
Logical operators (&&, ||, !) work with **boolean (true/false)** values.

**Example:**

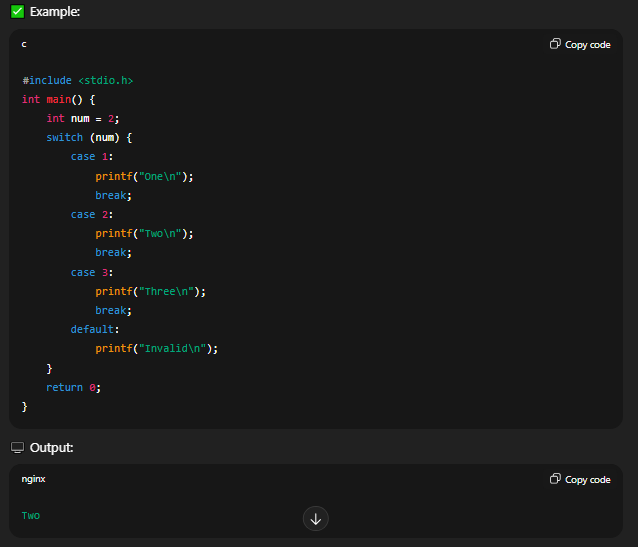
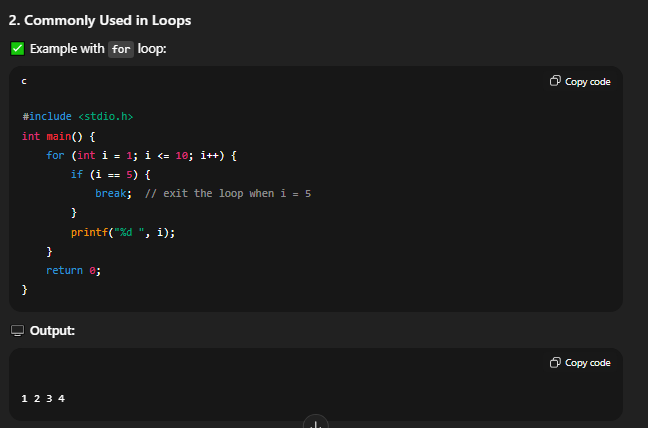
**Ques 6. Explain goto, break and continue statements in C? WAP to print factorial of a number using goto statement?**

**Ans. goto**, **break**, and **continue** are **control statements** used to **change the normal flow** of a program.

**A) goto Statement:** It transfers the control of the program **directly** to a **labeled statement** inside the same function.

* **Unconditional Jump:**
* Unlike loops or conditions, goto **does not check any condition**.
* It jumps directly to the label as soon as it is executed.
* **Label Declaration:**
* The label is **an identifier followed by a colon (:)**.
* The label can be written **before or after** the goto statement.
* Labels must be **unique** inside a function.
* Label name can be anything.
* **Forward and Backward Jump:**
* goto can jump **forward** (down the code) or **backward** (up the code).
* **goto Works Only Within the Same Function:**
* You cannot jump to a label **outside the current function**.
* Labels are **local to the function** in which they are defined.

**B) Break Statement:** The **break statement** is used to **immediately terminate** the **nearest enclosing loop** (for, while, do-while) or a **switch statement**, and **transfer program control to the statement next to it**.



**C) continue Statement:** The **continue statement** is used to **skip the remaining statements** in the **current iteration** of a loop and **jump directly to the next iteration** of that loop.

* continue can only be used inside loops (for, while, do-while).
* Unlike break, it **cannot be used in switch statements** directly.
* It can be Used in Nested Loops.