1.Explain the various Symbols Used in a Flowchart with a complete flowchart to find the sum of two numbers
2.a) Differentiate primary memory and secondary memory

Primary memory	**Secondary memory**
The primary memory of a computer is the main memory that is utilized to store data temporarily.	Secondary memory defines to additional storage devices that are utilized to store data permanently.
Primary memory is temporary.	Secondary memory is permanent.
Primary memory is faster than secondary memory because it is directly accessible to the CPU.	Secondary memory is non-volatile, which means it retains data even when the power is off.
Primary memory is directly accessible by Processor/CPU.	Secondary memory is not directly accessible by the CPU.

Nature of Parts of Primary memory varies, RAM- volatile in nature. ROM- Non-volatile.	It’s always Non-volatile in nature.
Primary memory is volatile, which means it is wiped out when the computer is turned off.	Since it is non-volatile, data can be retained in case of a power failure.
Primary memory devices are more expensive than secondary storage devices.	Secondary memory devices are less expensive when compared to primary memory devices.
The memory devices used for primary memory are semiconductor memories.	The secondary memory devices are magnetic and optical memories.
It can hold data/information currently being used by the processing unit.	It can hold data/information that are not currently being used by the processing unit.

The capacity of primary memory is usually within the range of 16 to 32 GB.	It stores a considerable amount of data and information. The capacity of secondary memory ranges from 200 GB to some terabytes.
Primary memory is also known as Main memory or Internal memory.	Secondary memory is also known as External memory or Auxiliary memory.
It can be accessed by a data bus.	It can be accessed using I/O channels.
Examples: RAM, ROM, Cache memory, PROM, EPROM, Registers, etc.	Examples: Hard Disk, Floppy Disk, Magnetic Tapes, etc.

b)Advantages of Algorithms:
1. It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
2. An algorithm uses a definite procedure.
3. It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
4. Every step in an algorithm has its own logical sequence so it is easy to debug.
5. By using algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for programmer to convert it into an actual program.
Certainly! Let's break down algorithms, flowcharts, and pseudocode, highlighting their characteristics, suitable examples, and when to use each.

3.Compare and contrast algorithms, flowcharts, and pseudocode with suitable examples. When should each be used?
1. Algorithms

Definition: An algorithm is a step-by-step procedure or formula for solving a problem. It provides a clear set of instructions that can be implemented in a programming language.

Example:
To find the maximum of three numbers (a, b, c):
1. Start
2. If a > b and a > c, then max = a
3. Else if b > a and b > c, then max = b
4. Else max = c
5. End

When to Use:
- Use algorithms when you need a clear and detailed understanding of the steps to solve a problem.
- Ideal for complex problems that require rigorous logic.

2. Flowcharts

Definition: A flowchart is a visual representation of an algorithm. It uses shapes and arrows to show the flow of control and the steps involved in the process.

Example:
```plaintext
  [Start]
     |
  [Input a, b, c]
     |
   +---+
   |  |
 [a>b?]---Yes---> [a>c?]---Yes---> [max=a]
   |  |          |  |
   No  |         No  |
   |  |          |  |
 [b>c?]---Yes---> [max=c]
   |  |
  No  |
   |   |
 [max=b]
   |
  [End]
```

When to Use:
- Use flowcharts for visual representation of algorithms, making it easier to understand complex processes.
- Ideal for presentations or documentation where clarity is essential.

3. Pseudocode

Definition: Pseudocode is a high-level description of an algorithm that uses a mix of natural language and programming constructs. It's not bound to any specific syntax, making it easier to read and understand.

Example:
```plaintext
BEGIN
  INPUT a, b, c
  IF a > b AND a > c THEN
    max = a
  ELSE IF b > a AND b > c THEN
    max = b
  ELSE
    max = c
  END IF
  OUTPUT max
END
```

When to Use:
- Use pseudocode when you want to express an algorithm without worrying about syntax errors.
- Good for early stages of development and when communicating with non-programmers.

Summary

- **Algorithms:** Provide a precise sequence of steps for problem-solving. Use them for detailed planning.

- **Flowcharts:** Offer a visual representation of an algorithm. Use them for clarity and communication.
- **Pseudocode:** Blend of natural language and code structure. Use it for conceptualizing algorithms without strict syntax.

Each method has its strengths and is suitable for different contexts within programming and algorithm design. Depending on your audience and purpose, you might choose one over the others.

4)a)Write an algorithm to find the area and circumference of a circle
b)Write a Pseudocode to find the area of a triangle

5)Describe the complete compilation process of a C program, detailing in each phase.

In C programming, errors can generally be classified into two main types: syntax errors and semantic errors. Understanding the differences between them is crucial for debugging and writing effective code.

1. Syntax Errors

Definition: Syntax errors occur when the code violates the grammatical rules of the C programming language. These errors prevent the compiler from interpreting the code correctly.

Examples:

- **Missing Semicolon:**

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

- Incorrect Use of Braces:

```
int main() {
    if (1 == 1) {
        printf("True");
}  // Missing closing brace for main function
```

Detection by Compiler:

- The compiler detects syntax errors during the parsing stage. When it encounters a statement that doesn't conform to the language's grammar rules, it generates an error message.
- Example error message: "error: expected ';' before 'return'."

Effect on Program Execution:

- Syntax errors prevent the program from compiling successfully. If there are any syntax errors, the executable code is never generated, and the program cannot run.

2. Semantic Errors

Definition: Semantic errors occur when the code is syntactically correct but doesn't make logical sense. These errors often arise from incorrect logic, misuse of operators, or wrong assumptions about data types.

Examples:

- **Incorrect Variable Usage:**

```
int main() {
    int a = 10, b = 0;
    int result = a / b;  // Division by zero
    printf("%d", result);
    return 0;
}
```

- **Using an Uninitialized Variable:**

```
int main() {
    int x;  // x is uninitialized
    printf("%d", x);  // Using x before it is given a value
    return 0;
}
```

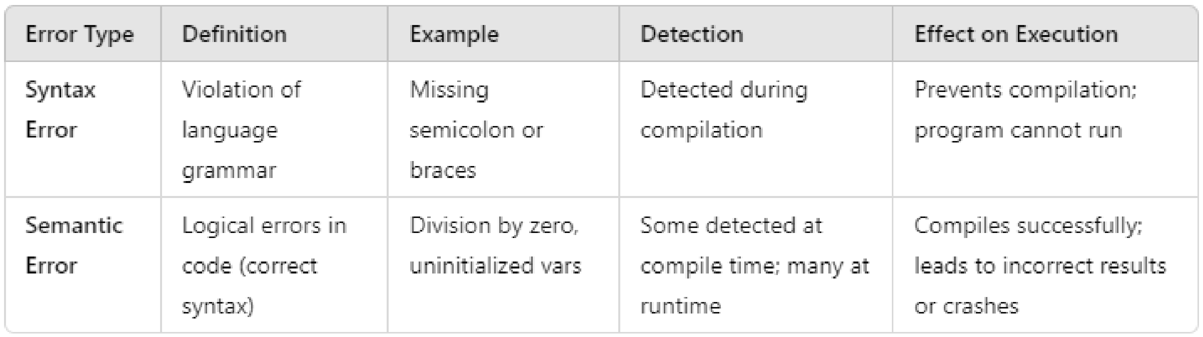
Detection by Compiler:

- Compilers can sometimes detect certain semantic errors, such as type mismatches (e.g., trying to assign a string to an integer variable). However, many semantic errors can only be detected during program execution (e.g., runtime errors like division by zero).
- The compiler may not generate an error message for a semantic error; instead, the error might lead to incorrect output or a runtime error.

Effect on Program Execution:

- Semantic errors can lead to incorrect results or unexpected behavior during program execution. The program may compile successfully but produce unintended results or crash at runtime (e.g., segmentation faults).

Summary



Error Type	Definition	Example	Detection	Effect on Execution
Syntax Error	Violation of language grammar	Missing semicolon or braces	Detected during compilation	Prevents compilation; program cannot run
Semantic Error	Logical errors in code (correct syntax)	Division by zero, uninitialized vars	Some detected at compile time; many at runtime	Compiles successfully; leads to incorrect results or crashes

7.Explain in detail about Developing Phases in C program.

8.Explain the character set used in C, and describe its importance in programming

9.Explain the different primitive data types in C with their size, default values, and usage.

10.Explain type conversion in C with examples. Discuss the difference between implicit and explicit conversion, and the potential issues with each.

1. Implicit Conversion

Definition: Implicit conversion, also known as automatic type conversion or coercion, occurs when the compiler automatically converts one data type to another without explicit instruction from the programmer. This typically happens when mixed-type expressions are evaluated.

Eg:

```
#include <stdio.h>


int main() {

    int a = 5;

    float b = 3.2;

    float result = a + b;  // 'a' is implicitly converted to float

    printf("Result: %f\n", result);  // Output: Result: 8.200000

    return 0;

}
```

In this example, the integer a is automatically converted to a float when added to b, ensuring that both operands are of the same type for the addition.

Potential Issues:

- **Precision Loss:** When converting from a larger data type to a smaller one (e.g., from double to float), precision can be lost.
- **Unexpected Results:** Implicit conversions can sometimes lead to results that are not intuitive, especially in mixed-type operations.

2. Explicit Conversion

Definition: Explicit conversion, also known as type casting, occurs when the programmer manually specifies the type conversion using casting operators. This provides more control over how conversions are handled.

Eg:

```
#include <stdio.h>


int main() {

    double a = 5.9;

    int b = (int)a;  // Explicitly converting double to int

    printf("Converted value: %d\n", b);  // Output: Converted value: 5

    return 0;

}
```

In this example, the double variable a is explicitly cast to an int, resulting in the loss of the decimal portion.

Potential Issues:

- **Data Loss:** When converting from a floating-point type to an integer type, the fractional part is discarded, which can lead to unexpected results.
- **Undefined Behavior:** Incorrect use of type casting (e.g., casting incompatible types) can lead to undefined behavior, especially with pointers.

Summary of Differences:

Feature	Implicit Conversion	Explicit Conversion
Initiation	Done automatically by the compiler	Done manually by the programmer
Syntax	No special syntax needed	Requires casting syntax (e.g., `(type)`)
Control	Less control over the conversion process	More control; the programmer specifies how to convert
Precision Loss	May happen in certain situations	Often results in data loss (e.g., float to int)
Safety	Generally safer, but can lead to surprises	Riskier if done incorrectly