Programming and Problem Solving (CoE132)

Course Description:

The course Indicate some reasons for studying programming fundamentals, covers the basics of programming and the "C++" programming language, including syntax, fundamental data structures. algorithms and basic problem-solving, control structures, string manipulation and list processing. concepts of executive programs.

Course Topics:

An introduction to programming fundamentals: topics, variables. data types, and operations. Programming paradigms (Functional, Procedural, Object-oriented, and Event-Driven).

Problem-solving Algorithms: Problem-solving strategies and process, Implementation strategies for algorithms, Debugging strategies, the concept and properties of algorithms, structured decomposition

Programming in C++: Basic syntax and semantics, Variables. types, expressions, assignment, Mathematical functions, logical and bitwise and arithmetic operations, Simple I/O, Functions and parameter passing, procedural programming, Encapsulation and information-hiding Separation of behavior and implementation.

Control structures: Conditional and iterative control structures, loops, sequencing, selection, and iteration functions.

Basic Data Structures: Primitive types, Arrays, Strings and string processing, Records, stack, and heap allocation.

Structure programming: static and dynamic structure programming.

Recursion: Recursive mathematical functions, Divide-and-conquer strategies, Recursive backtracking, Implementation of recursion in C++.

References:

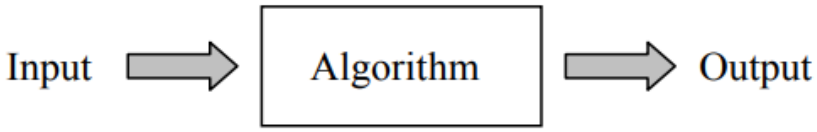
- **C++ How to Program 8th Edition**
  by Paul Deitel , Harvey Deitel ,
  Publisher Prentice Hall

- **C++: The Complete Reference 4th Edition**
  by Herbert Schildt
  Publisher Mcgraw- Hill Education

Problem- Solving Algorithms

Introduction

An algorithm is a computational procedure consisting of a set of instructions, that takes some value or set of values, as *input*, and produces some value or set of values, as *output*.



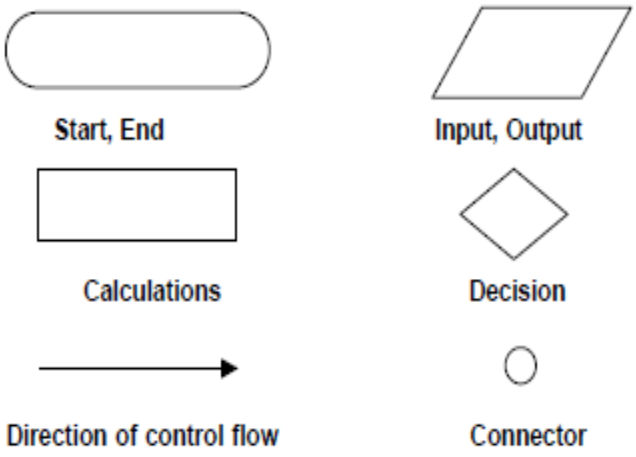
Characteristics Of An Algorithm

What makes an algorithm an algorithm? There are four essential properties of an algorithm.

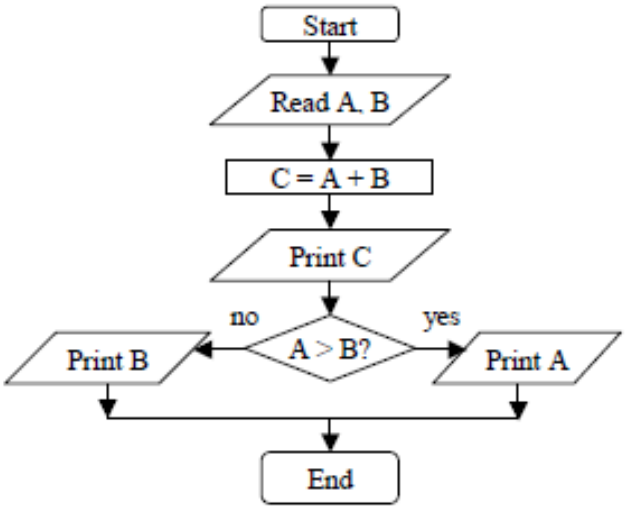
1. **Each step of an algorithm must be exact**. An algorithm must be exactly and clearly described.
2. **An algorithm must terminate.** The purpose of an algorithm is to solve a problem. If the program does not stop when executed, we will not be able to get any result from it. Therefore, an algorithm must contain a finite number of steps in its execution.
3. **An algorithm must be effective.** An algorithm must provide the correct answer to the problem.
4. **An algorithm must be general.** This means that it must solve every instance of the problem. For example, a program that computes the area of a rectangle should work on all possible dimensions of the rectangle, within the limits of the programming language and the machine.

Using Flow Chart To Symbolize Algorithm

Since the flows of computational paths are represented as a picture, it is called a *flow chart*. Let us start with an example. Suppose we have to find the sum and also the maximum of two numbers. First the two numbers have to be received and kept in two places, under two names. Then the sum of them is to be found and printed. Then depending on which one is bigger, a number is to be printed. In the flow chart, each shape has a particular meaning.



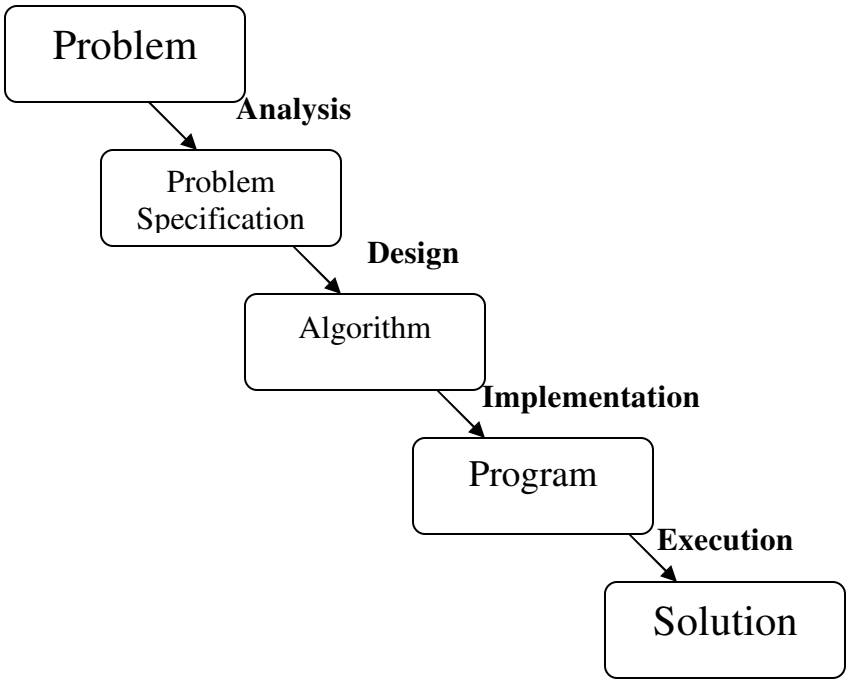
In the following flow chart, the first block represents the start point of the flow chart, the second block is to read the values of variables A and B. The rectangle used to calculate addition, subtracting, multiplying……. Here A and B are added and the sum is saved in C.
Printing process is an output, so it putted in parallelogram shape. Then the condition is placed to test if A>B. The right option of the condition is always (yes) and the left is (No). End block used to exit the flow chart.



Implementation of strategies for algorithms

Programs are created faster and quickly. To create efficient and effective program, the following procedure has to be followed. They are

- **Understand the Problem:** Collect the problem to generate a program.
- **Analyze the Problem:** Analyze the various steps of solving a problem.
- **Design the Problem:** Design the problem by creating flow chart and writing algorithm.
- **Code the Program:** Create coding by using proper programming language.
- **Test and Debug the Program:** Use proper testing methods and check the program and debug the errors.
- **Complete the Documentation:** Create documentation for all the details about the program. For instance, the analysis data, design data, source code, testing details etc., has to be maintained.
- **Maintain the Program:** Create the steps for modifying program to remove previously detected errors etc.



Debugging Strategies

A *debugger* allows the programmer, to interact and inspect the running program, making it possible to trace the flow of execution and track down the problems.

For example if we forgot to put a semicolon(;) after return statement in the following programming section:

```
1  int main()
2   {
3  return 0
4  }
```

Your compiler should generate an error something like…

```
\Ali.cpp(4) : error C2143: syntax error : missing ';' before '}'
```

`\main.cpp(4)` The error is in the file Ali.cpp line 4

`error C2143:` The compiler specific error code

`syntax error :` Messed up some syntax

`missing ';' before '}'` There's a missing semi-colon before a closing bracket

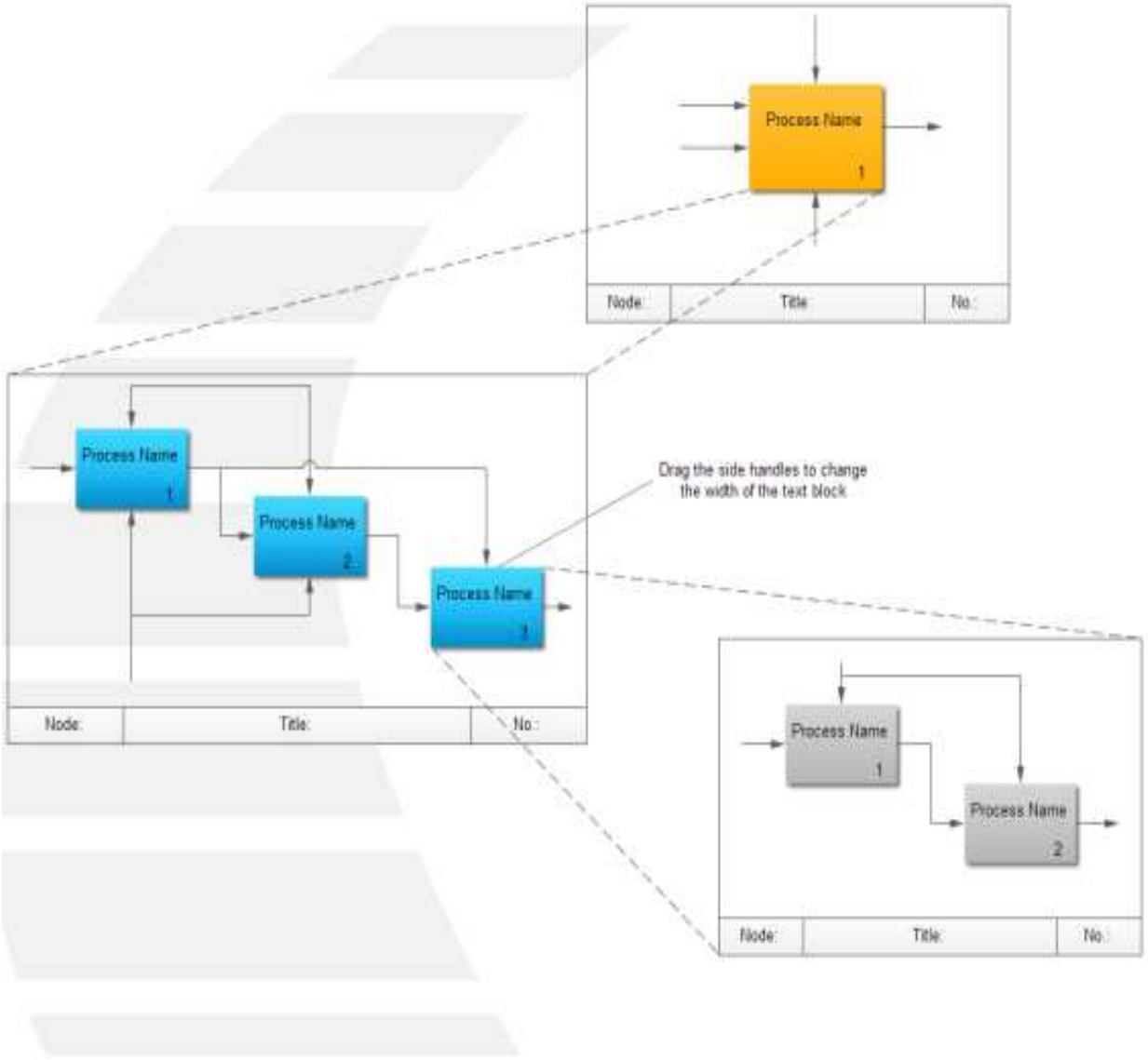
The following general strategies will reduce the time that debugging takes in the process:

- Minimize Problems by Avoiding Copy-Paste Syndrome.
- Make Big Problems Found Late Small Problems Found Early
- Check Your Helper Functions
- Make sure you know what your program is doing

Decomposition

Decomposition is a technique for breaking large complex problems into a series of smaller related problems (sub-problem). There are different types of decomposition defined to solve problems:

- *structured programming* : breaks a process down into well-defined steps.
- *Structured analysis*: breaks down a software system from the system context level to system functions and data entity.
- *Object-oriented decomposition*: breaks a large system down into increasingly smaller classes or objects that are responsible for some part of the problem domain.



The figure above shows a general representation of decomposition concept. The first block is the main process, which contain the three blocks 1,2 and 3. Block 3 contains two other blocks 1 and 2. It is easy to programmer now to solve the problem starting with smaller problems 1 and 2 of block 3 reaching to the main block. This method is called Top-Down planning.