STRUCTURED PROGRAMMING

PROBLEM SOLVING & PROGRAM DESIGN

Two phases involved in the design of any program:

(i) Problem Solving Phase

(ii) Implementation Phase

PROBLEM SOLVING & PROGRAM DESIGN

In the problem-solving phase the following steps are carried out:

Define the problem

Outline the solution

Develop the outline into an algorithm

Test the algorithm for correctness

The implementation phase comprises the following steps:

Code the algorithm using a specific programming language

Run the program on the computer

Document and maintain the program

STRUCTURED PROGRAMMING CONCEPT

Structured programming techniques assist the programmer in writing effective error free programs.

The elements of structured of programming include:

- Top-down development
- Modular design.

STRUCTURED PROGRAMMING CONCEPT

The Structure Theorem:

It is possible to write any computer program by using only three (3) basic control structures, namely:

Sequence

– one command is executed after previous one.

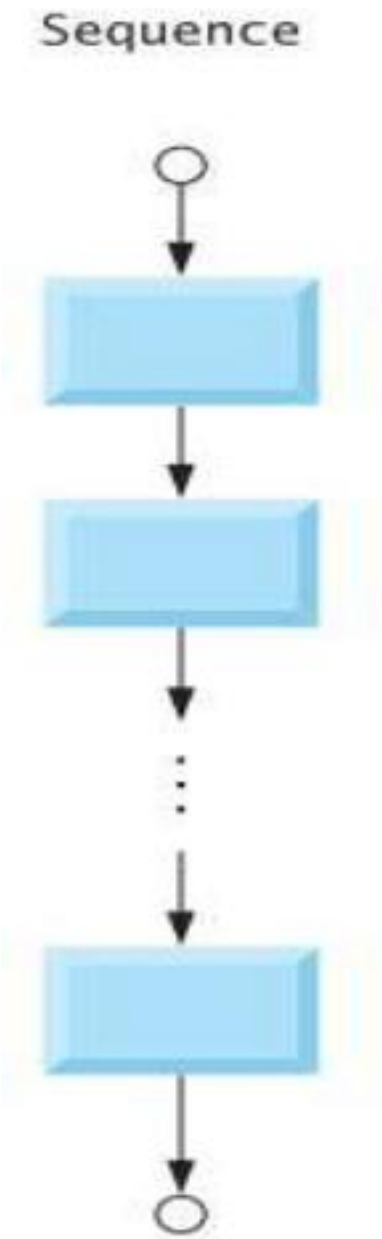
Decision (Selection (if-then-else)) –

– statement(s) is (are) executed if certain condition gives TRUE or FALSE value.

Loops(Repetition (iteration, looping, Do While)) -

– statement(s) is (are) executed repeatedly until

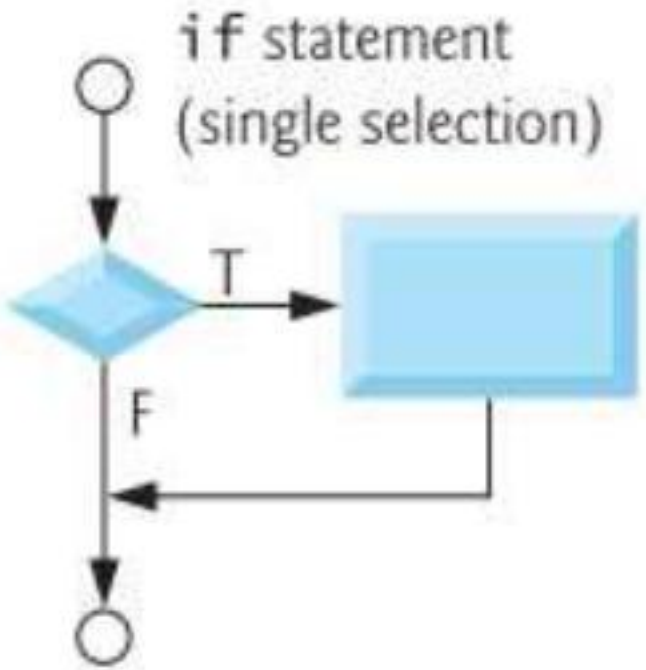
Sequences



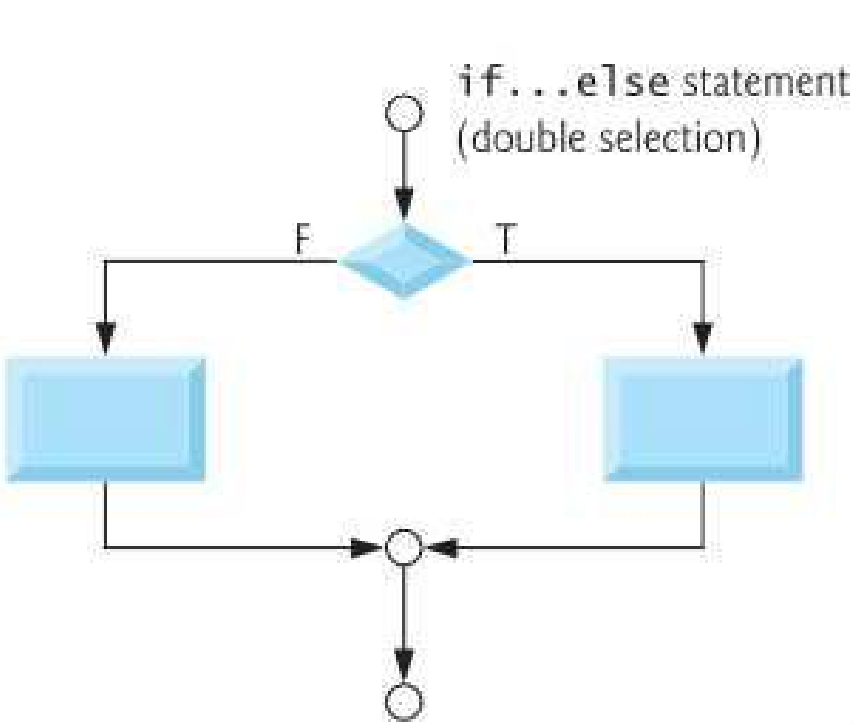
Decisions (selections)

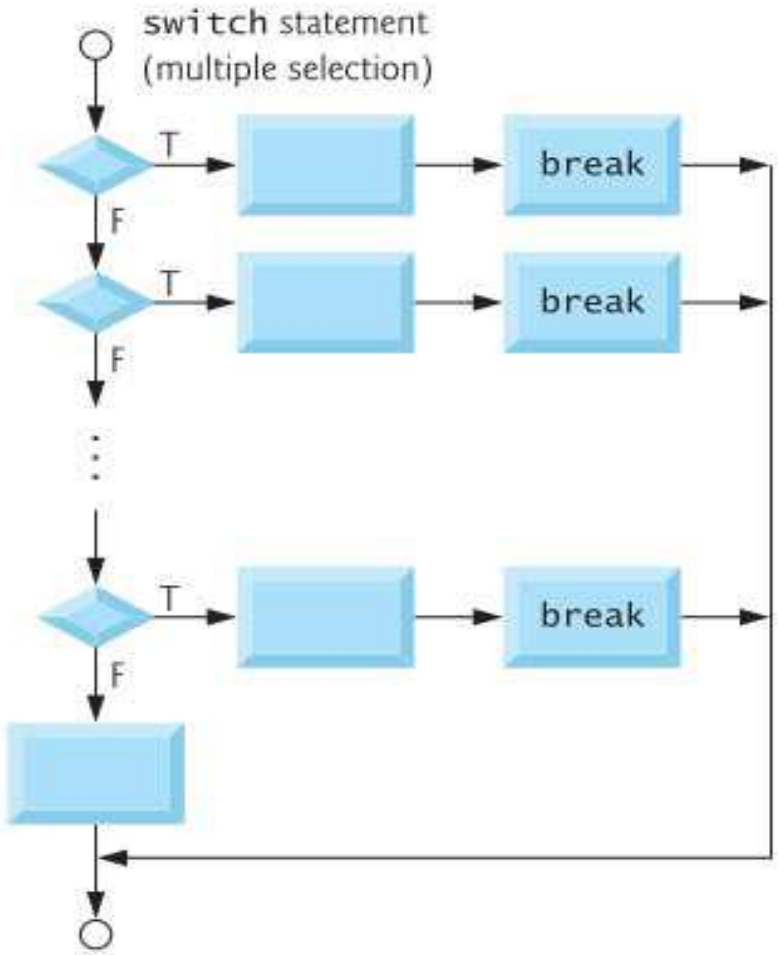
Three selection structure in C programming:

- If
- If – else
- Switch



Decisions (selections)

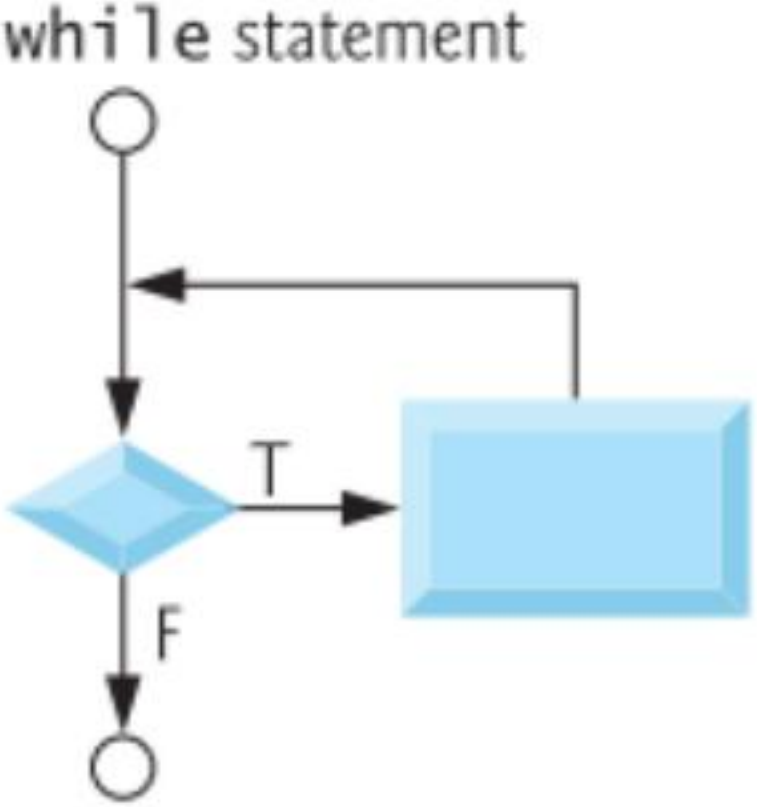




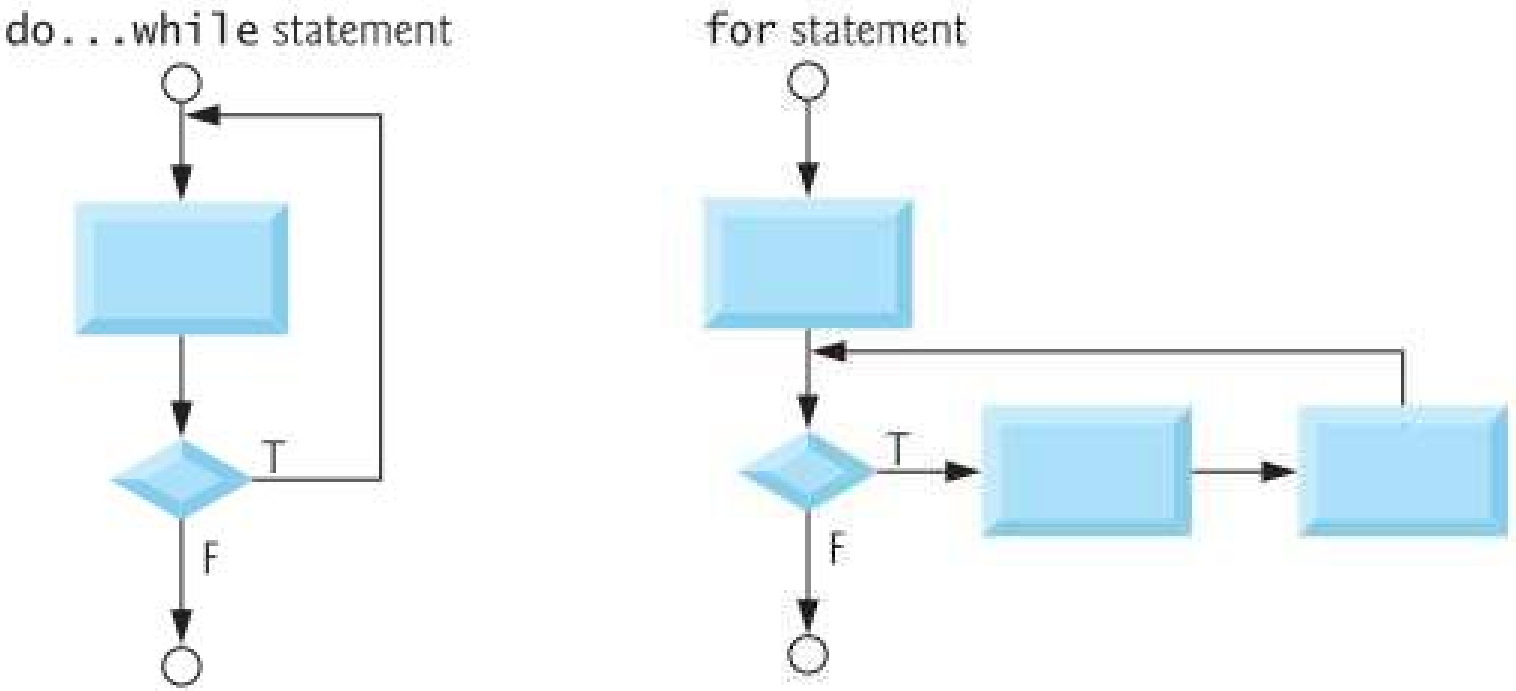
Loops (repetition)

Three repetition structure in C programming:

- While
- Do – while
- For

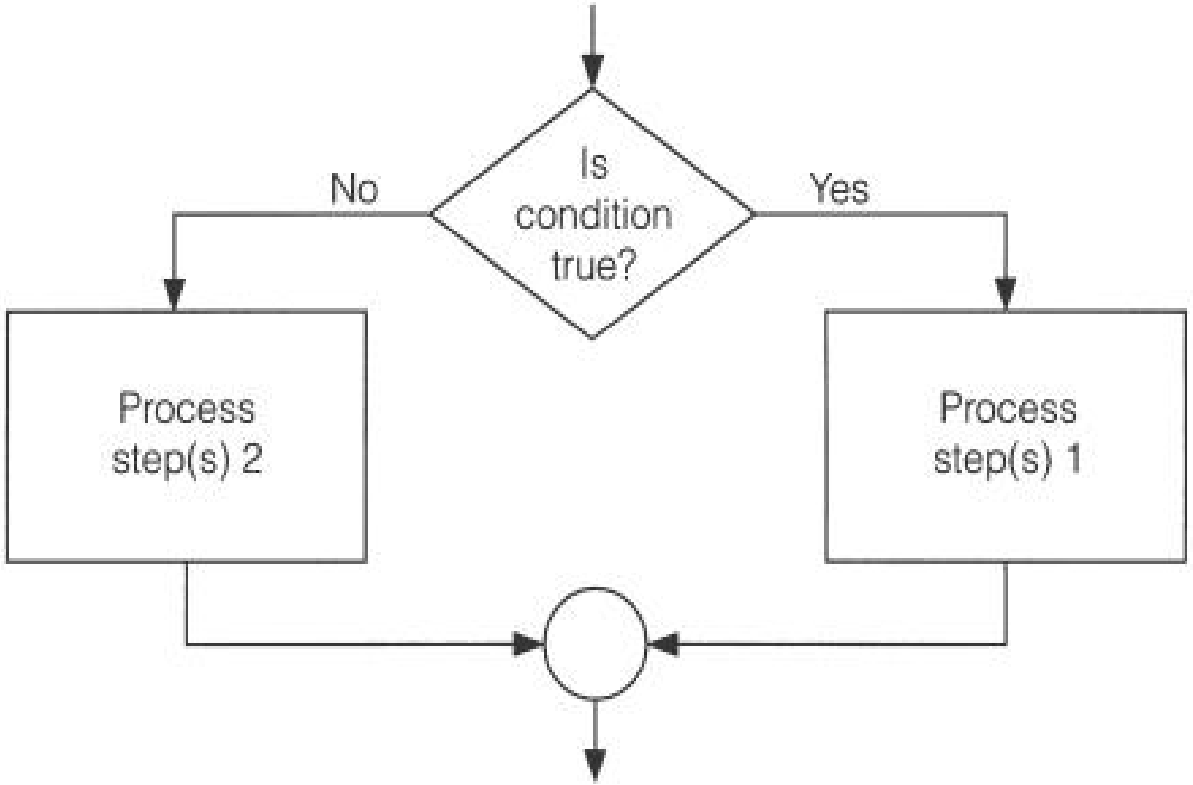


Loops (repetition)



Pseudocode and Flowchart for a Decision Structure

```
If condition is true Then
      Process step(s) 1
   Else
      Process step(s) 2
End If
```



Example

- Write an algorithm to determine a student's average grade and indicate whether he is successful or not.
- The average grade is calculated as the average of mid-term and final marks.
- Student will be successful if his average grade is grater or equals to 60.

Pseudocode

- Start
- Use variables mid term , final and average
- Input mid term and final
- Calculate the average by summing mid term and final and dividing by 2
- if average is below 60

  Print "FAIL"

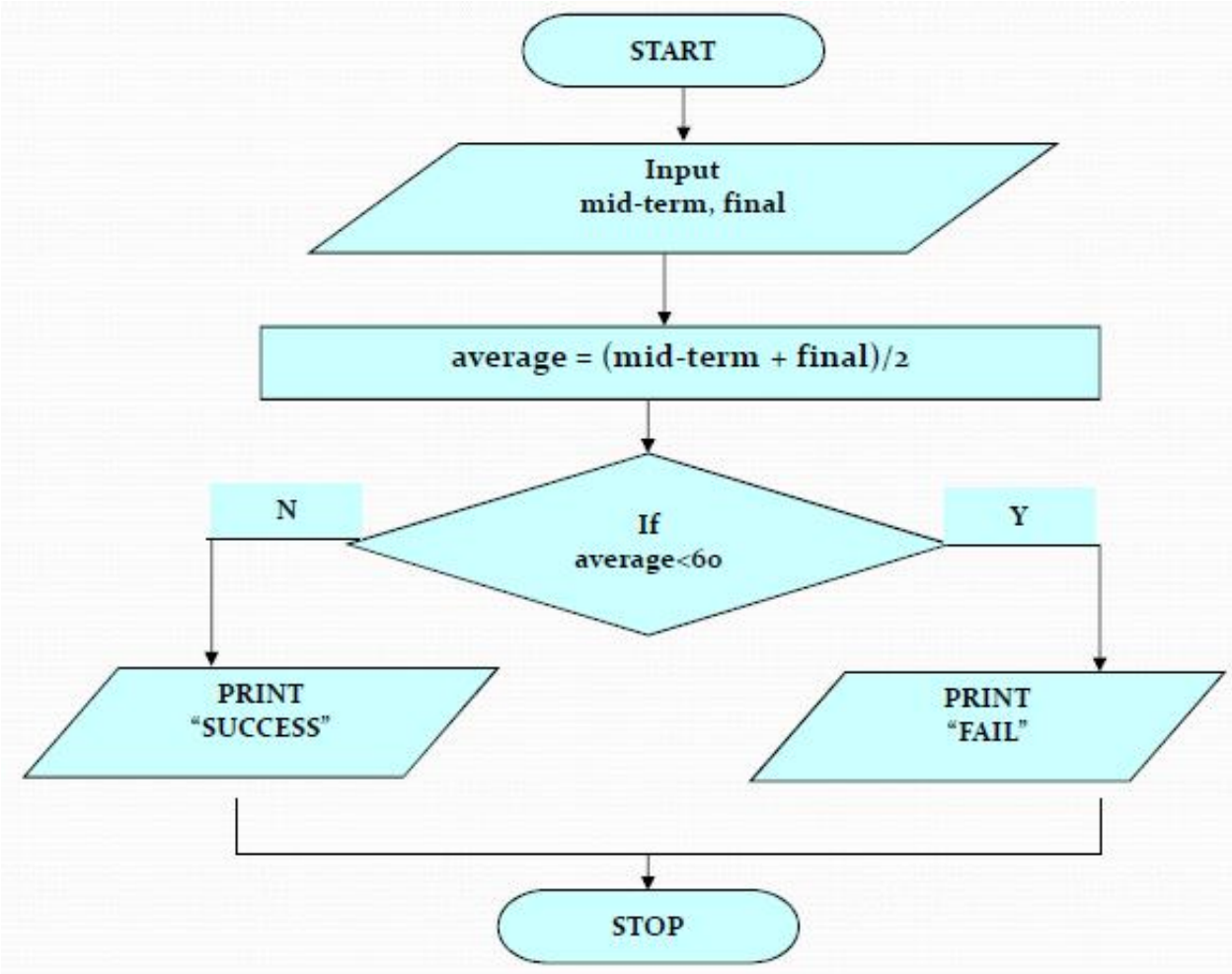
  else

  Print "SUCCESS"
- Stop

Detailed Algorithm

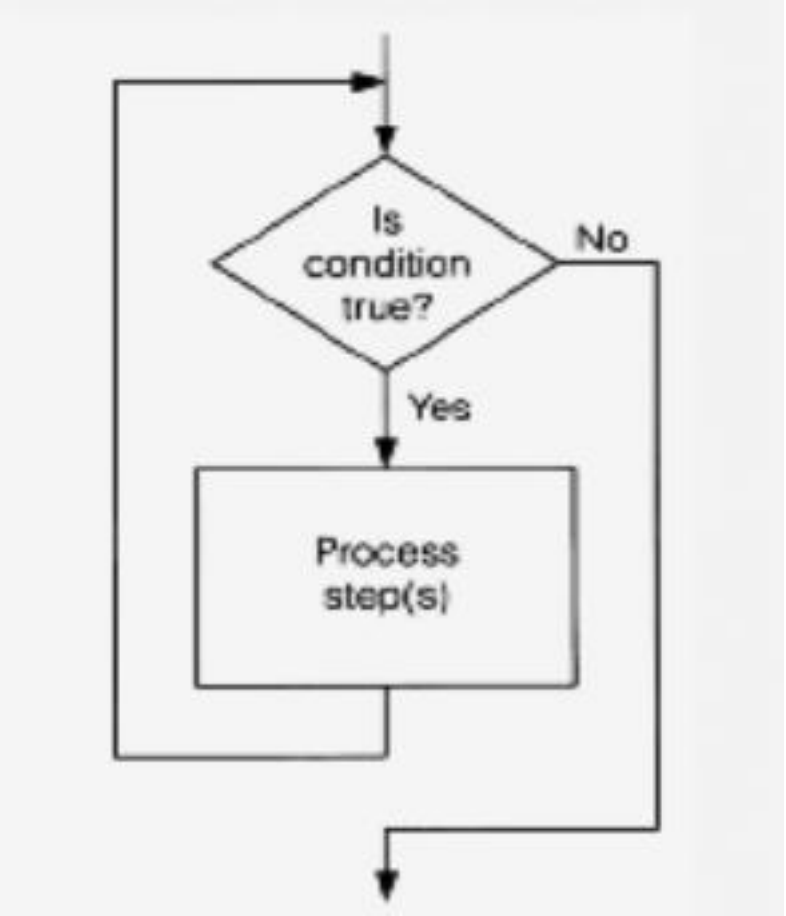
- 1. Step: Input mid-term and final
- 2. Step: average = (mid-term + final)/2
- 3. Step: if (average < 60) then
  Print "FAIL"
  else
  Print "SUCCESS"
  endif

Flowchart


START
Input
mid-term, final
average = (mid-term + final)/2
N
If
average<60
Y
PRINT
"SUCCESS"
PRINT
"FAIL"
STOP

Pseudocode and Flowchart for a Loop

```
Do While condition is true
    Process step(s)
Loop
```



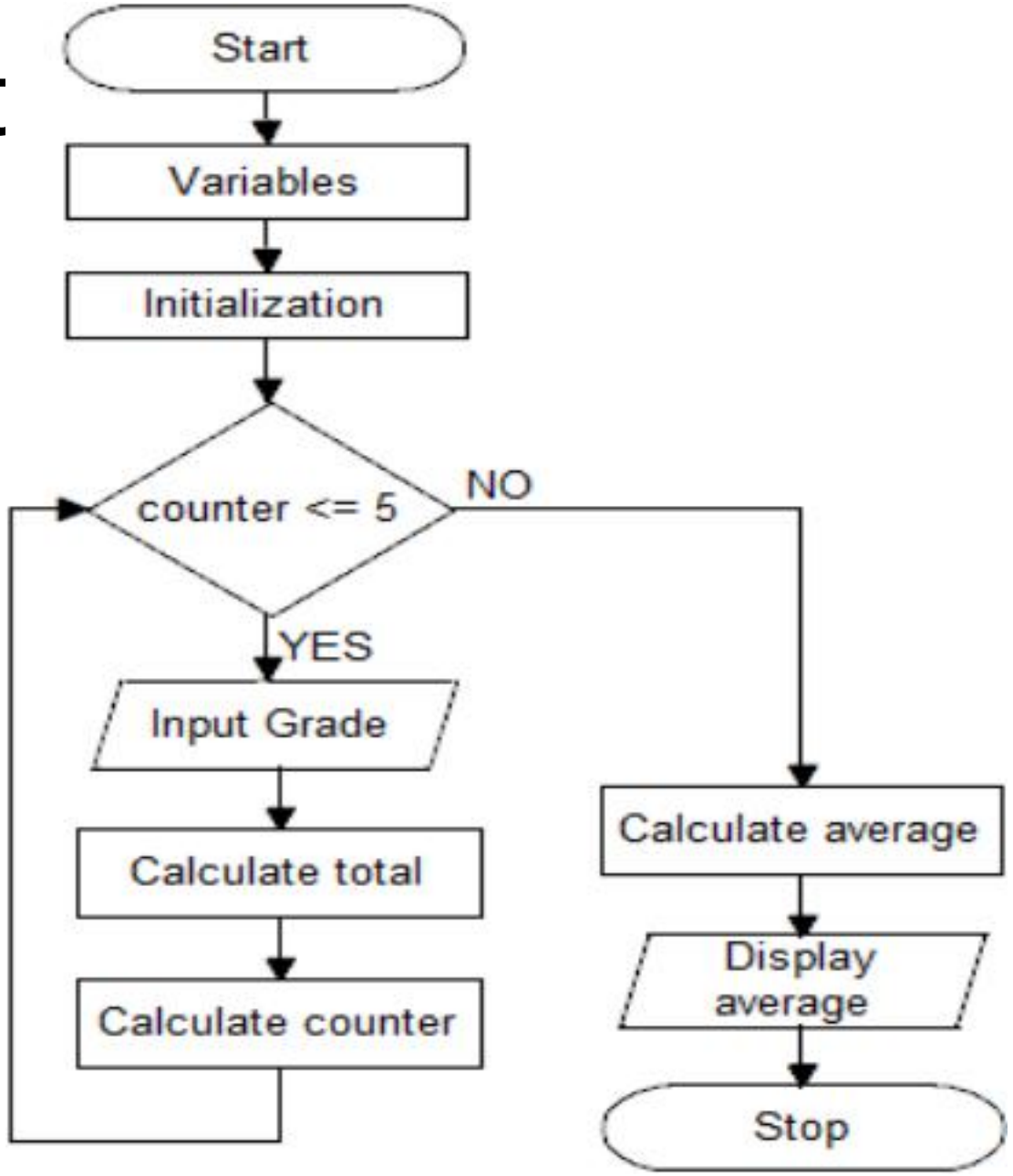
Example

- Write an algorithm which calculates the average exam grade for a class of 5 students.
- What are the program inputs?
  - the exam grades
- Processing:
  - Find the sum of the grades;
  - count the number of students; (counter controlled)
  - calculate average grade = sum of grades / number of students.
- What is the program output?
  - the average exam grade

Pseudocode

- Start
- Use variables total, counter, grade, average
- Initialize total = 0
- Initialize counter = 1
- While (counter <= 5)
- Input grade
- Calculate total = total + grade
- Calculate counter = counter + 1
- End-while
- Calculate average = total / 5
- Display average
- Stop

Flowchart


Start
Variables
Initialization
counter <= 5
NO
YES
Input Grade
Calculate total
Calculate counter
Calculate average
Display average
Stop

- Write an algorithm which calculates the average exam grade for a class of unknown number of students.
- What are the program inputs?
  - the exam grades
- Processing:

  Find the sum of the grades till sentinel value is given; for example -99 to break loop (sentinel controlled)

  calculate average grade = sum of grades / number of students.
- What is the program output?

  the average exam grade

Pseudocode

- Start
- Use variables total, counter, grade, average
- Initialize total = 0
- Initialize counter = 0
- While (grade != -99)
- Input grade
- Calculate total = total + grade
- Calculate counter = counter + 1
- End-while
- Calculate average = total / counter
- Display average
- Stop

Example

- Write an algorithm which calculates the average exam grade for a class of unknown number of students.
- This time, the number of students have been asked at the beginning of the program.
- Use counter controlled structure.

Pseudocode

- Start
- Use variables total, counter, grade, average, number_of_students
- Initialize total = 0 , number_of_students = 0 , counter = 1
- Display "write number of students"
- Input number_of_students
- While (counter <= number_of_students)
- Input grade
- Calculate total = total + grade
- Calculate counter = counter + 1
- End-while
- Calculate average = total / number_of_students
- Display average
- Stop

Flowchart

- Draw a flowchart for the above example

STRUCTURED PROGRAMMING CONCEPT

There is no standard definition of structured programs available but it is often thought to be programming without the use of a goto statement.

Structured programming is:

- Concerned with improving the programming process through better organization of program and better programming notation to facilitate correct and clear description of data and control structure.
- Concerned with improved programming languages and organized programming techniques which should be understandable and therefore, more easily modifi-able and suitable for documentation.

STRUCTURED PROGRAMMING CONCEPT

There is no standard definition of structured programs available but it is often thought to be programming without the use of a goto statement.

Structured programming is:

- More economical to run because good organization and notation make it easier for an optimizing compiler to understand the program logic.
- More correct and therefore more easily debugged, because general correctness theorems dealing with structures can be applied to proving the correctness of programs.

STRUCTURED PROGRAMMING CONCEPT

Structured programming can be defined as a

- top–down analysis for program solving
- modularization for program structure and organization
- structured code for individual modules

Top-Down Analysis

A program is a collection of instructions in a particular language that is prepared to solve a specific problem.

Top-down analysis reduces the complexity of the process of problem solving. It provides a strategy that has to be followed for solving all problems.

There are two essential ideas in top-down analysis:

- subdivision of a problem – breaking a big problem into two or more smaller problems. To solve the big problem, first these smaller problems have to be solved.
- hierarchy of tasks -Top-down analysis does not simply divide a problem into two or more smaller problems. This process continues downwards, creating a hierarchy of tasks, from one level to

Top-Down Analysis

The four basic steps to top-down analysis are as follows:

Step 1:

Define the complete scope of the problem to determine the basic requirement for its solution. Three factors must be considered in the definition of a programming problem.

Input: What data is required to be processed by the program?

Process: What must be done with the input data? What type of processing is required?

Output: What information should the program produce? In what form should it be presented?

Top-Down Analysis

Step 2:

Based on the definition of the problem, divide the problem into two or more separate parts.

Step 3:

Carefully define the scope of each of these separate tasks and subdivide them further, if necessary, into two or more smaller tasks.

Step 4:

Repeat step 3. Every step at the lowest level describes a simple task, which cannot be broken further.

Modular Programming

- Modular programming is a program that is divided into logically independent smaller sections, which can be written separately.
- These sections, being separate and independent units, are called modules.

Modular Programming

- A module consists of a series of program instructions or statements in some programming language.
- A module is clearly terminated by some special markers required by the syntax of the language. For example, a BASIC language subroutine is terminated by the return statement.
- A module as a whole has a unique name.
- A module has only one entry point to which control is transferred from the outside and only one exit point from which control is returned to the calling module.

Modular Programming

Some advantages of modular programming.

- Complex programs may be divided into simpler and more manageable elements.
- Simultaneous coding of different modules by several programmers is possible.
- A library of modules may be created, and these modules may be used in other programs as and when needed.
- The location of program errors may be traced to a particular module; thus, debugging and maintenance may be simplified.

Structured Code

- Structured programming is a method of coding, i.e., writing a program that produces a well-organized module.
- A high-level language supports several control statements, also called structured control statements or structured code, to produce a well-organized structured module.
- These control statements represent conditional and repetitive type of executions.
- Each programming language has different syntax for these statements.

The Process of Programming

- The job of a programmer is not just writing program instructions.
- The programmer does several other additional jobs to create a working program.
- Some logical and sequential job steps which the programmer has to follow to make the program operational are as follows:
  1. Understand the problem to be solved
  2. Think and design the solution logic
  3. Write the program in the chosen programming language
  4. Translate the program to machine code
  5. Test the program with sample data
  6. Put the program into operation