Computer Programming (a)

E1123

Lecture 1

Introduction to C/C++

INSTRUCTOR

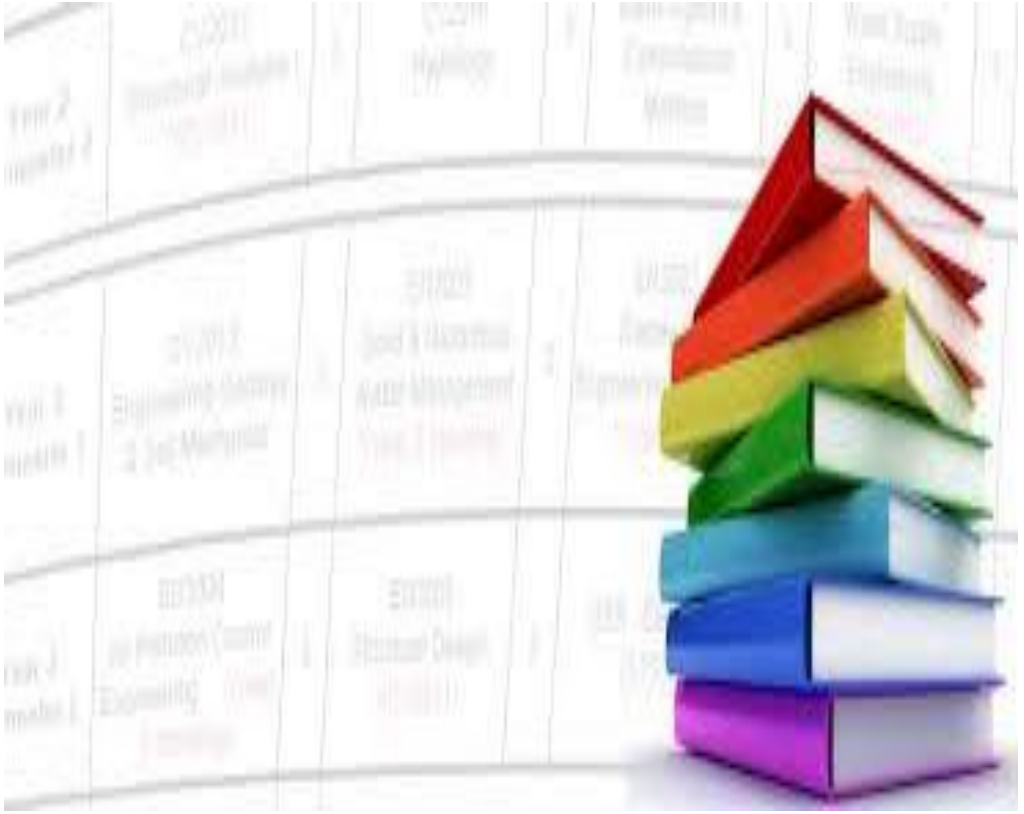
DR / AYMAN SOLIMAN

Contents

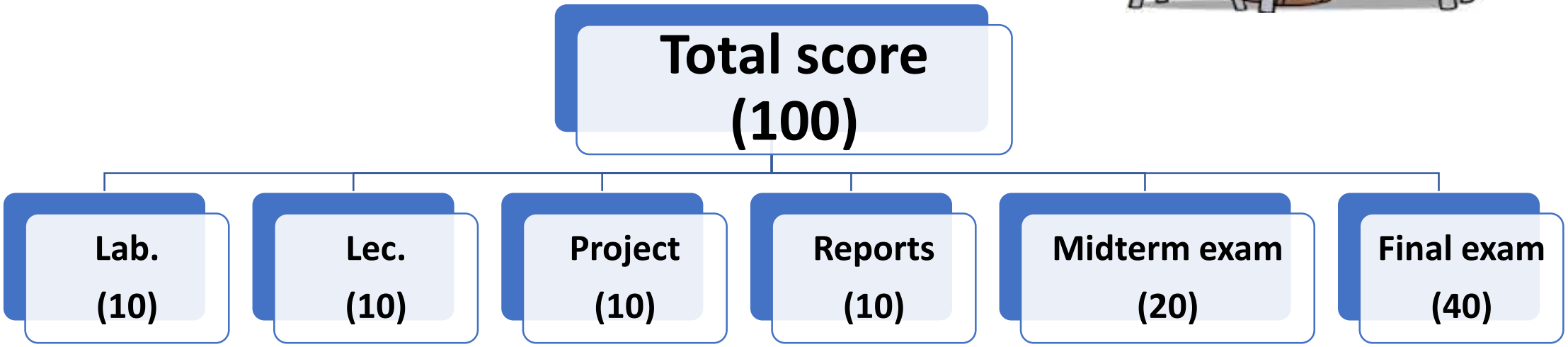
1) Course Contents.
2) Grading System & distribution.
3) Course Information.
4) Course Policy.
5) Objectives.
6) Introduction.
7) Flow Chart

1) Course Contents.

- Introduction to C/C++.
- C++ Basics.
- Data types and representation.
- Operators & bitwise operation.
- Conditional tools.
- Functions.
- Arrays.
- Pointers.
- Searching and sorting algorithms



2) Grading System & distribution.



3) Course Information.

Lectures: Tuesday, (10:40 - 11:25 AM)

Office Hours: Tuesday, Thursday.

Prerequisite: E1021 - E1022

References:

- C++ Programming: From Problem Analysis to Program Design, Fifth Edition D.S. Malik
- Object-Oriented Programming Using C++, Fourth Edition Joyce Farrell
- www.learncpp.com

Instructor:

Dr. Ayman Soliman

Ayman.mohamed01@bhit.bu.edu.eg

TAs:

Eng. Nada Elmeligy
Eng. Enas Mohamed
Eng. Rehab Ibrahim
Eng. Mohamed Mostafa
Eng. Ahmed Ragab
Eng. Mohamed Abd Satar
Eng. Mahmoud Osama
Eng. Khalid Diaa

4) Course Policy.

- Any forms of **cheating or plagiarism** will result in a **Zero grade** for the required task, report or exam (No discussion nor excuses).

- Students are expected to **respect** Instructors, TAs, and their colleagues.
- Be **on time** and cell phones should be silent or off during the lecture.
- Your grades is based on **merit only** nothing else.

5) Objectives

- Analyze a problem and construct a solution using C++ programming language.
- Explain how an existing C++ program works, discovering errors and fix them.
- Critique a C++ program and describe ways to improve it.
- Follow up intermediate and advanced level of C++ programming language.

6) Introduction

- Before C++, there was C

- The C language was developed in 1972 by Dennis Ritchie at Bell Telephone laboratories, primarily as a systems programming language (a language to write operating systems with).

- Ritchie's primary goals were to produce a minimalistic language that was easy to compile, allowed efficient access to memory, produced efficient code, and was self-contained (not reliant on other programs).

- For a high-level language, it was designed to give the programmer a lot of control, while still encouraging platform (hardware and operating system) independence (that is, the code didn't have to be rewritten for each platform).

6) Introduction (cont.)

- C ended up being so efficient and flexible that in 1973, Ritchie and Ken Thompson rewrote most of the Unix operating system using C.

- Many previous operating systems had been written in assembly. Unlike assembly, which produces programs that can only run on specific CPUs, C has excellent portability, allowing Unix to be easily recompiled on many different types of computers and speeding its adoption.

- C and Unix had their fortunes tied together, and C's popularity was in part tied to the success of Unix as an operating system.

6) Introduction (cont.)

C++

- C++ (pronounced see plus plus) was developed by Bjarne Stroustrup at Bell Labs as an extension to C, starting in 1979.

- C++ adds many new features to the C language and is perhaps best thought of as a superset of C, though this is not strictly true (as C99 introduced a few features that do not exist in C++).

- C++'s claim to fame results primarily from the fact that it is an object-oriented language. As for what an object is and how it differs from traditional programming methods, well, we'll cover that in second term.

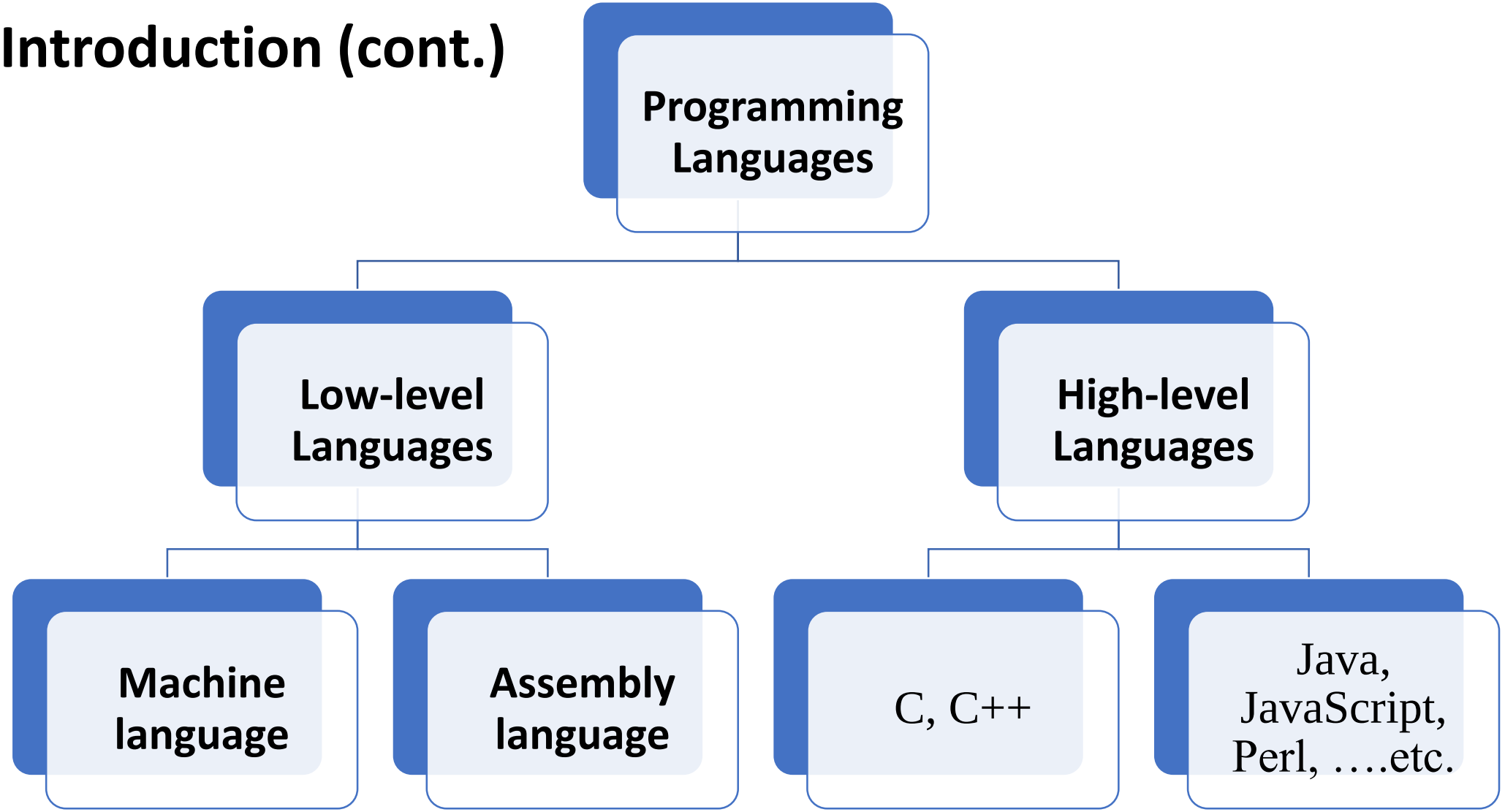
6) Introduction (cont.)

- C++ was standardized in 1998 by the ISO committee (this means the ISO committee ratified a document describing the C++ language, to help ensure all compilers adhered to the same set of standards). A minor update was released in 2003 (called C++03).

- Three major updates to the C++ language (C++11, C++14, and C++17, ratified in 2011, 2014, and 2017 accordingly).

- Each new formal release of the language is called a language standard (or language specification). Standards are named after the year they are released in. For example, there is no C++15, because there was no new standard in 2015.

6) Introduction (cont.)

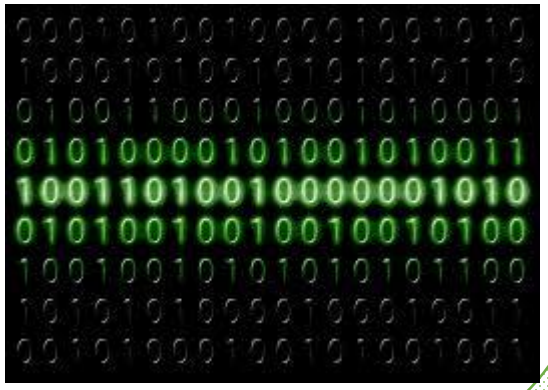


Machine language

Computers understand a very limited set of instructions and must be told what to do exactly. A program (also could be called an application or software) is a set of instructions that tells the computer what to do. The physical computer machinery that executes the instructions is the hardware.

Machine language (Instruction set)

- Only language computer directly understands as It is incapable of speaking C++.
- Instruction is composed of several binary digits, which can only be a 0 or a 1.
- Here is an example of machine language instruction: 10110000 01100001
- instruction tells the computer to do a very specific job.

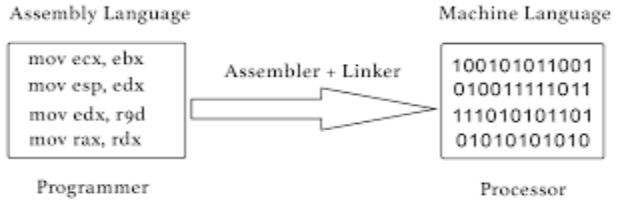


Defects

- Different types of CPUs will typically have different instruction sets.
- Very difficult and time-consuming thing to do (Cumbersome for humans).

Assembly language

- Instruction is identified by a short name and variables can be identified by names rather than bits and numbers.
- Clearer to humans and much easier to read and write than machine language.
- Assembly languages tend to be very fast, and it is still used today when speed is critical.
- Here is an example : mov b1, 031h



Defects

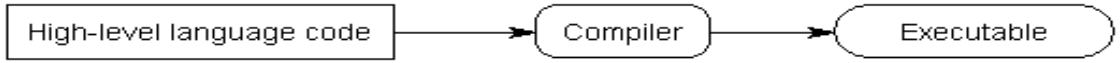
- Assembly language must be translated into machine language by using an assembler.
- The reason assembly language is so fast is because it is tailored to a particular CPU (assembly programs written for one CPU will not run on another CPU).
- It still requires a lot of instructions to do even simple tasks and are not very human readable.

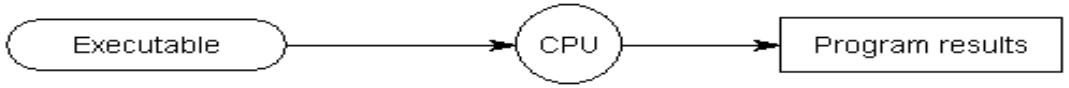
High-level Languages

- C, C++, Java, JavaScript, Perl, ….etc. are all high-level languages.
- Like everyday English, use common mathematical notations
- Single statements accomplish substantial tasks y=32*x+3, In assembly language, this would take 5 or 6 different instructions.
- Allow the programmer to write programs without having to be as concerned about what kind of computer the program is being run on.
- Programs must be translated into a form that the CPU can understand.
- This can be done by two ways: compiling and interpreting.

6) Introduction (cont.)

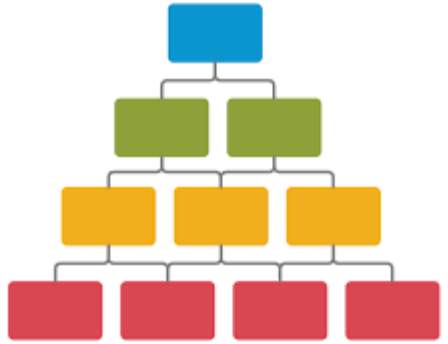
A compiler is a program that reads code and produces a stand-alone executable program that the CPU can understand directly.





Algorithms are sequence of steps for solving problems and there are some ways that can be used for representing an algorithm such as:

- Hierarchy Chart (Structure Chart).
- Pseudocode.
- Flowchart

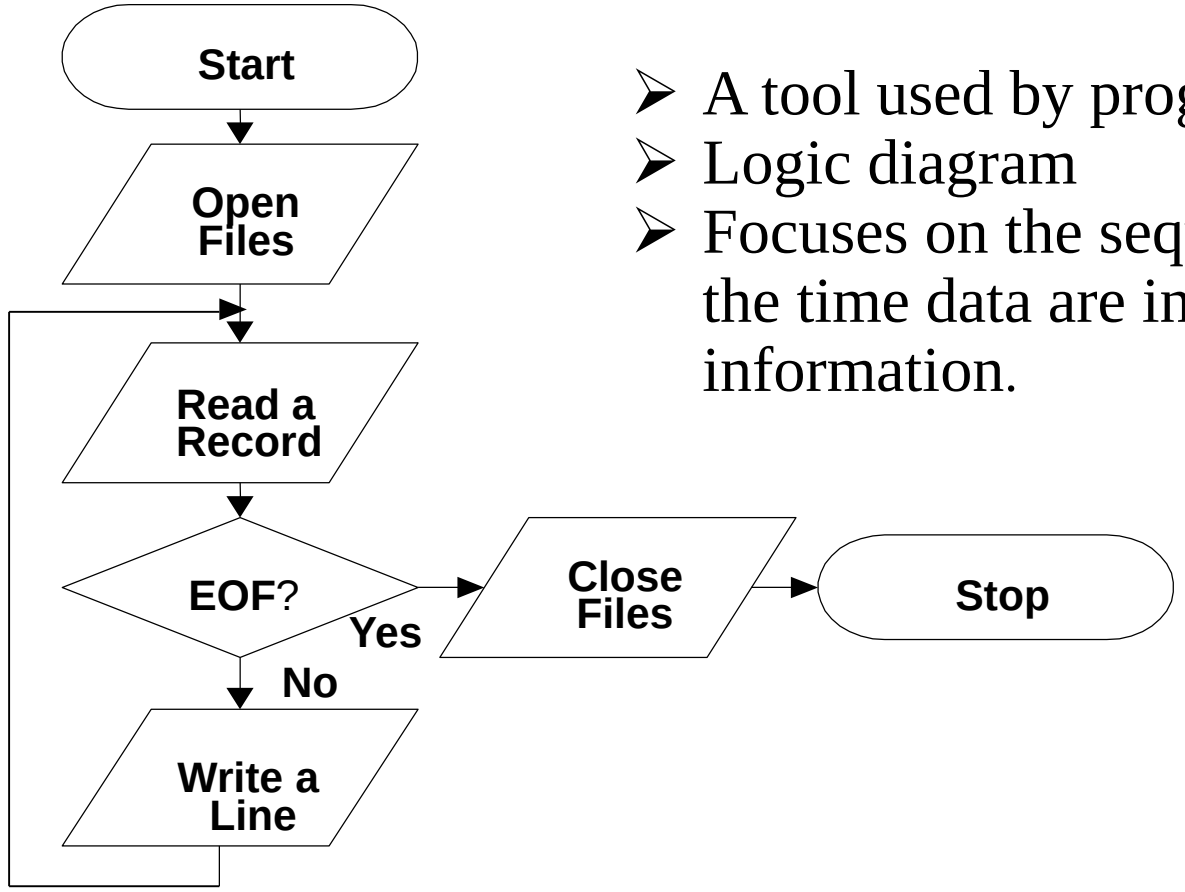


```
1. START
2. READ R
3. SET PI=3.142
4. CALCULATE AREA OF CIRCLE
   4.1.    FORMULA:
           Area= PI * R * R
5. DISPLAY Area
6. END
```

Low level language	High level language
Low level language are the language which are machine dependent	High level language are the language which are machine independent
Low level language can only run on single or specific type of computers	High level language can run on multiple type of computer and operating system
Each instruction in this type of language equates to single machine instruction	In high level language, each instruction equates to several machine instruction
Machine language and assembly language are the low level language.	Pascal, C++ and java are some example of high level language.
In low level language, there is lack of structured ability.	In high level language, structured ability is provided.
Low level languages are in binary form which is understandable to the computer.	High level language is written in English which is not understandable to computer.
No compiler, interpreter are required by the low level language as it is in binary form.	Compiler and interpreter are required by high level language for converting programs into binary form'
Low level language is difficult to learn.	It is easy and quick to learn.

7) Flow Chart

What is a program flowchart?



- A tool used by programmers to develop program logic
- Logic diagram
- Focuses on the sequence of data transformations from the time data are input until they are output as information.

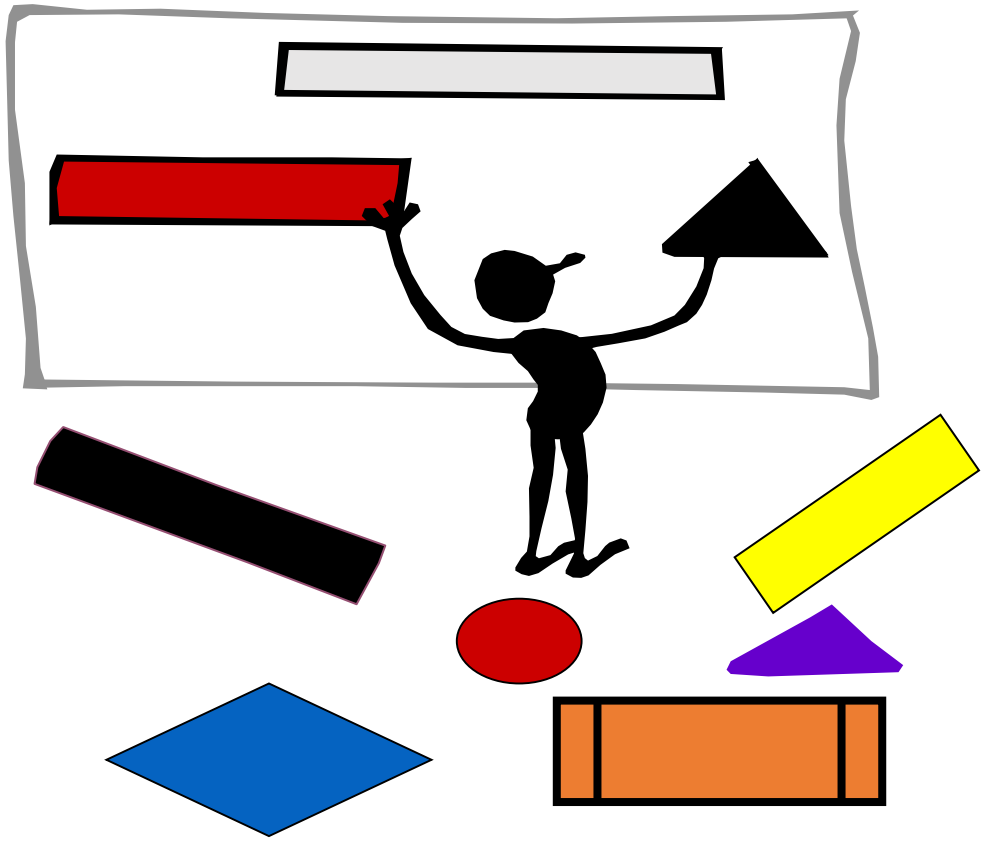
7) Flow Chart (cont.)

Why do programmers use flowcharts to develop program logic?

- Flowcharts are a good visual reference of what the program will do
- Serve as program documentation and as a communications tool
- Allow the programmer to test alternative solutions

7) Flow Chart (cont.)

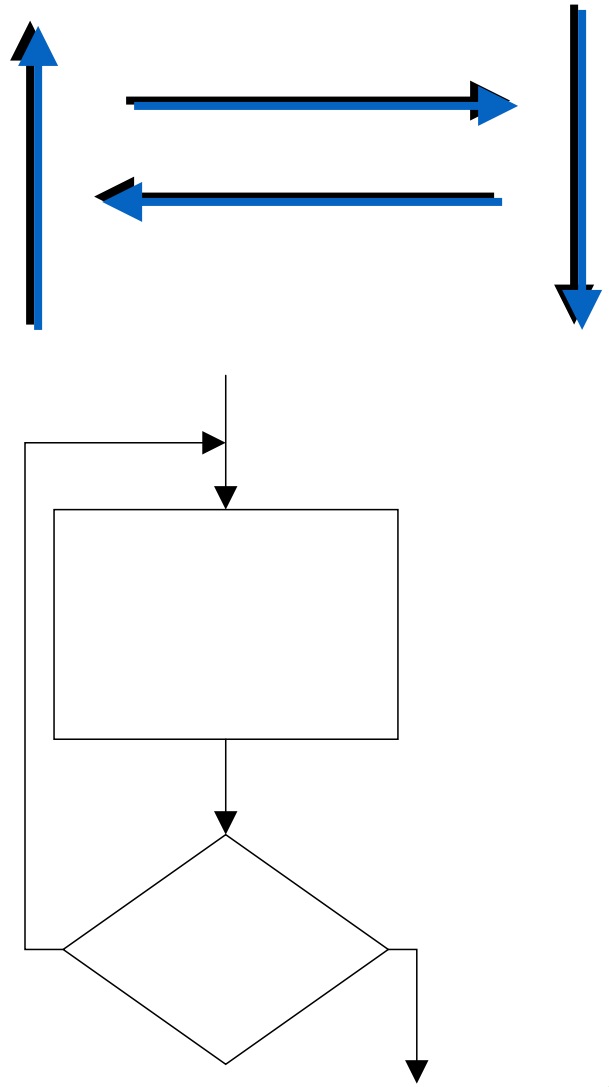
What are the flowchart symbols?



7) Flow Chart (cont.)

Flowline Symbol

- Used to connect symbols and indicates the flow of logic (sequence of operations)
- Normal flow is top to bottom and left to right
- Arrowheads must be used when the flow is other than the normal flow



7) Flow Chart (cont.)

Terminal Symbol

- Used to represent the beginning (start) or the end (End) of a program or routine

START

STOP

HEADINGS

RETURN

7) Flow Chart (cont.)

Input/output (I/O) Symbol

- Used for input and output operations, such as reading and displaying. The data to be read or displayed are described inside.

7) Flow Chart (cont.)

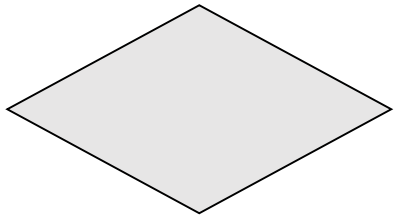
Processing Symbol

- Used for arithmetic and data manipulation operations

MOVE SCORE1 TO ACCUMULATOR

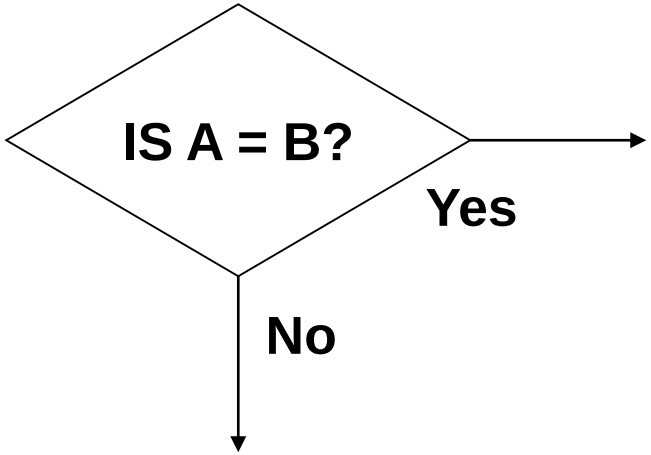
AVERAGE = (SCORE1 + SCORE2) / 2

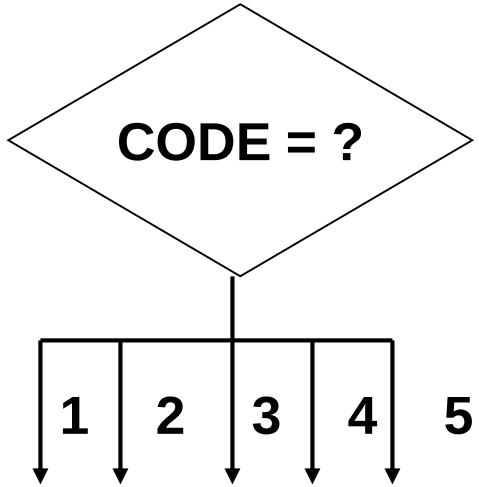
7) Flow Chart (cont.)



Decision Symbol

- Used for any logic or comparison operation. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two or more exit paths.

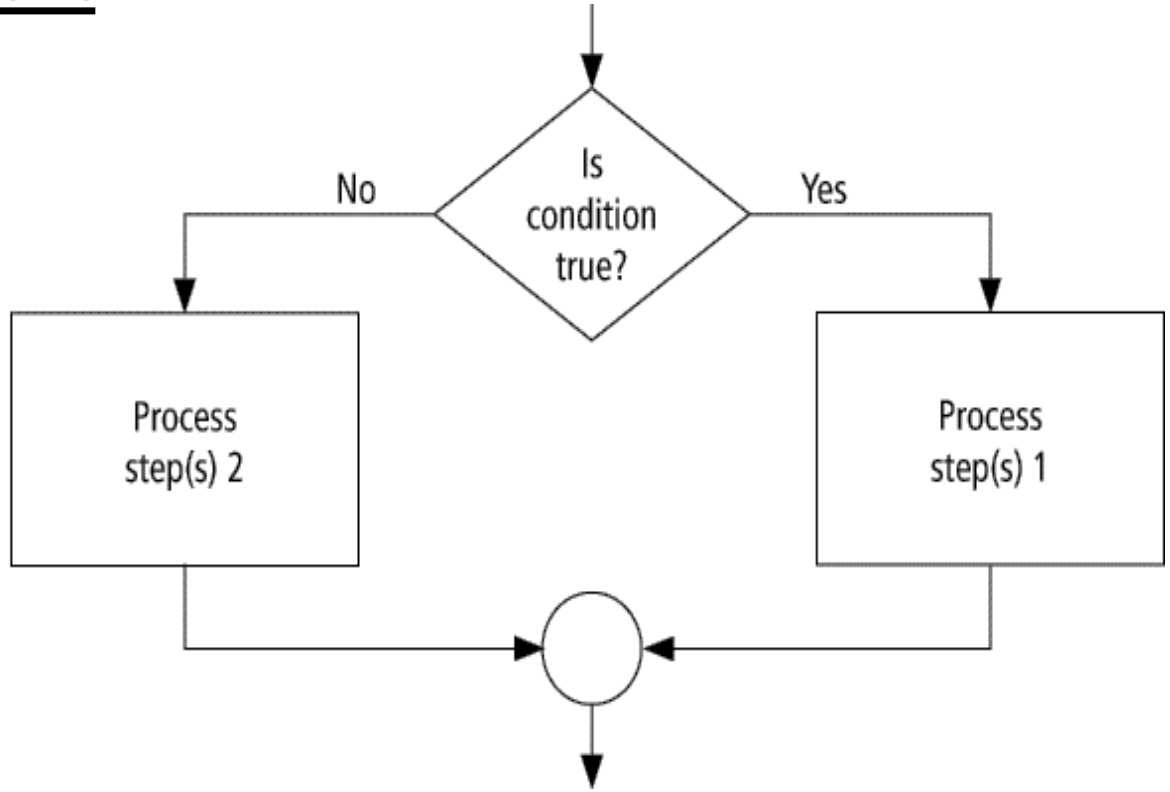




7) Flow Chart (cont.)

Decision flow chart

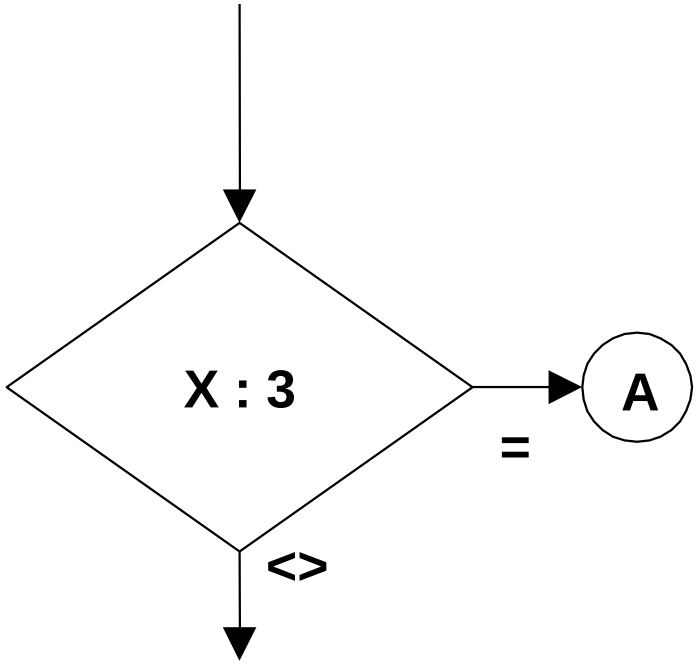
```
If condition is true Then
      Process step(s) 1
  Else
      Process step(s) 2
End If
```



7) Flow Chart (cont.)

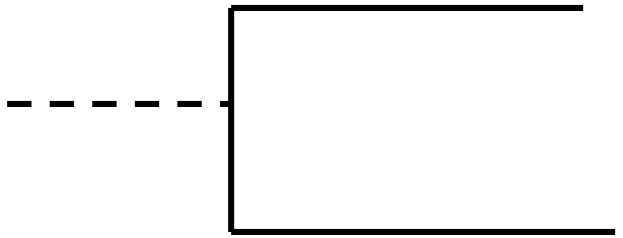
Connector Symbol

- Connects parts of the flowchart when the use of flowlines would be confusing or otherwise undesirable

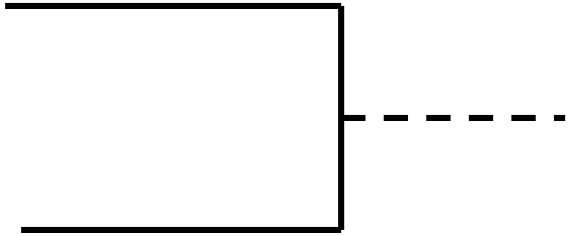


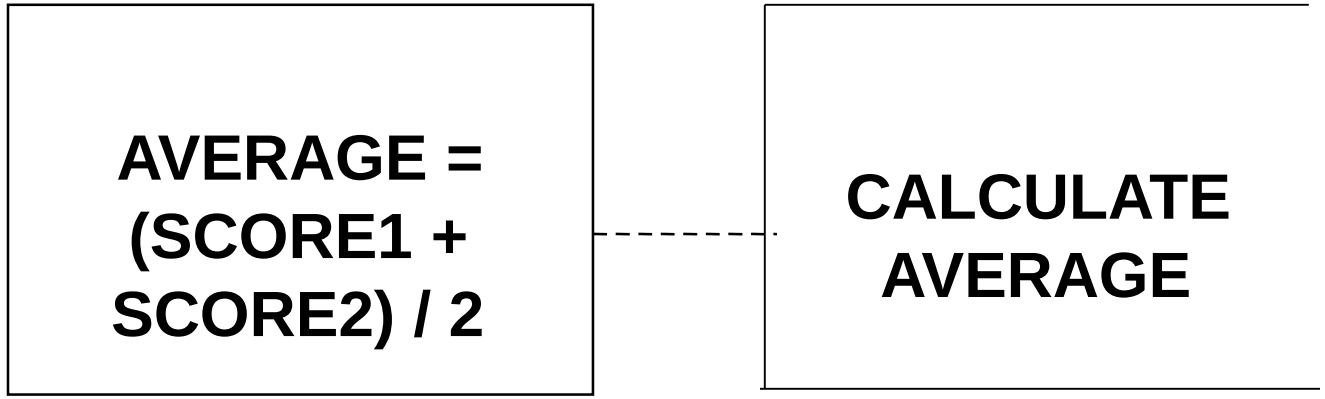
7) Flow Chart (cont.)

Annotation Symbol



- Provide additional information (comments) about another flowchart symbol

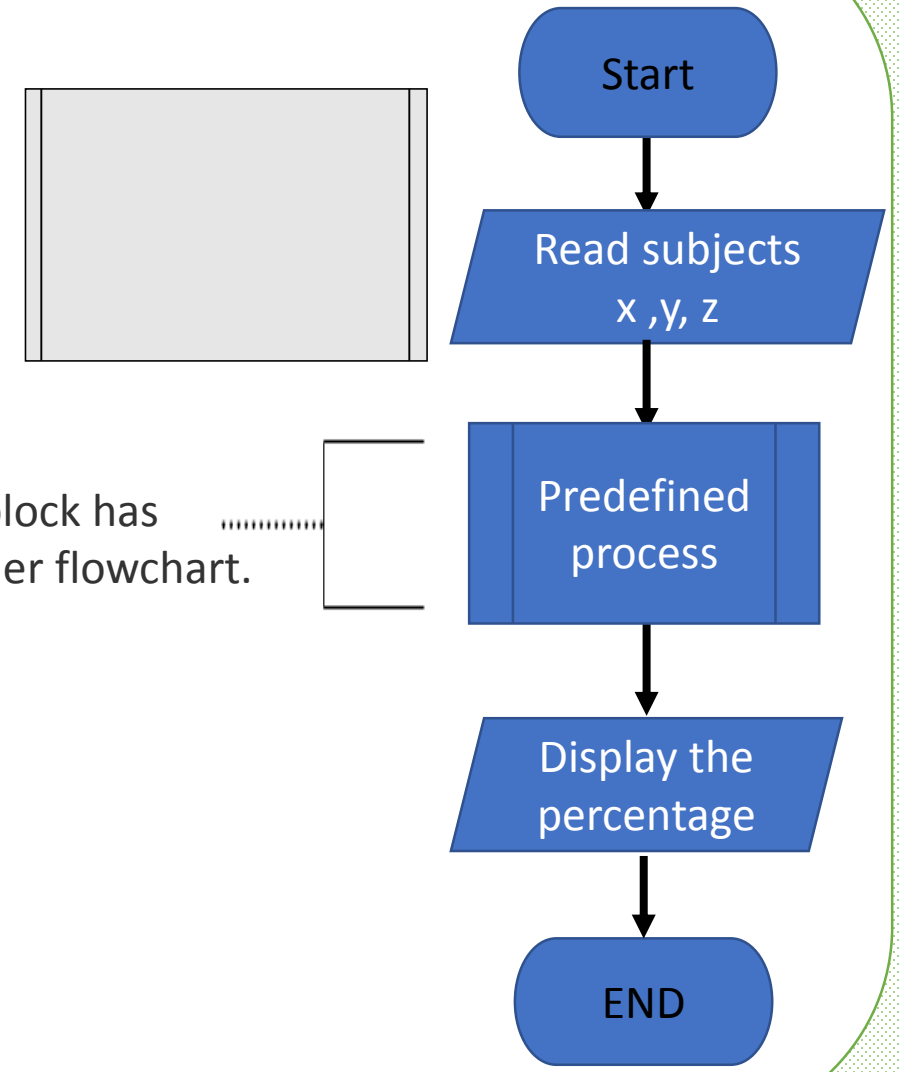




7) Flow Chart (cont.)

Subroutine Symbol (Predefined process)

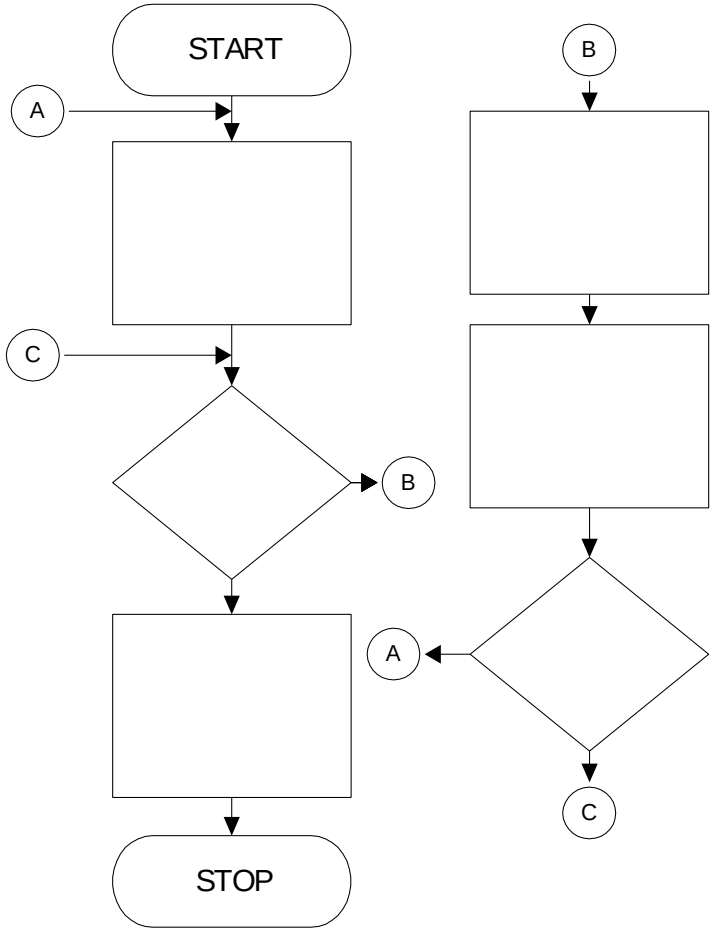
- Used to represent a group of statements that perform one processing task

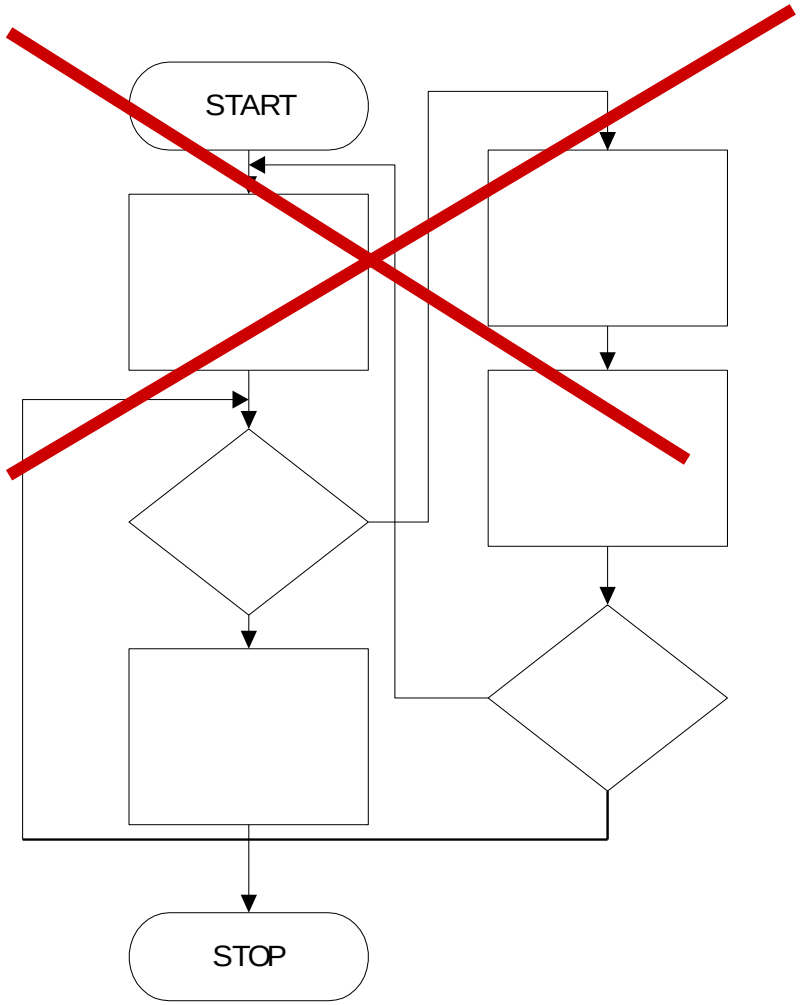


7) Flow Chart (cont.)

More Rules...

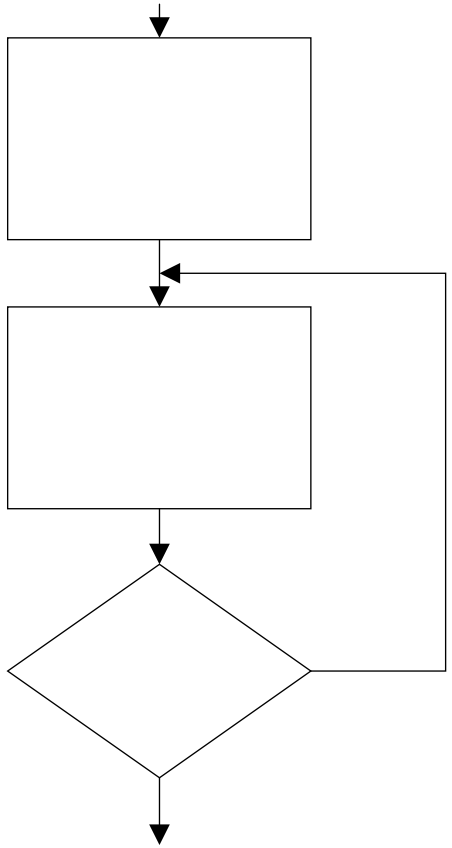
Avoid intersecting flowlines

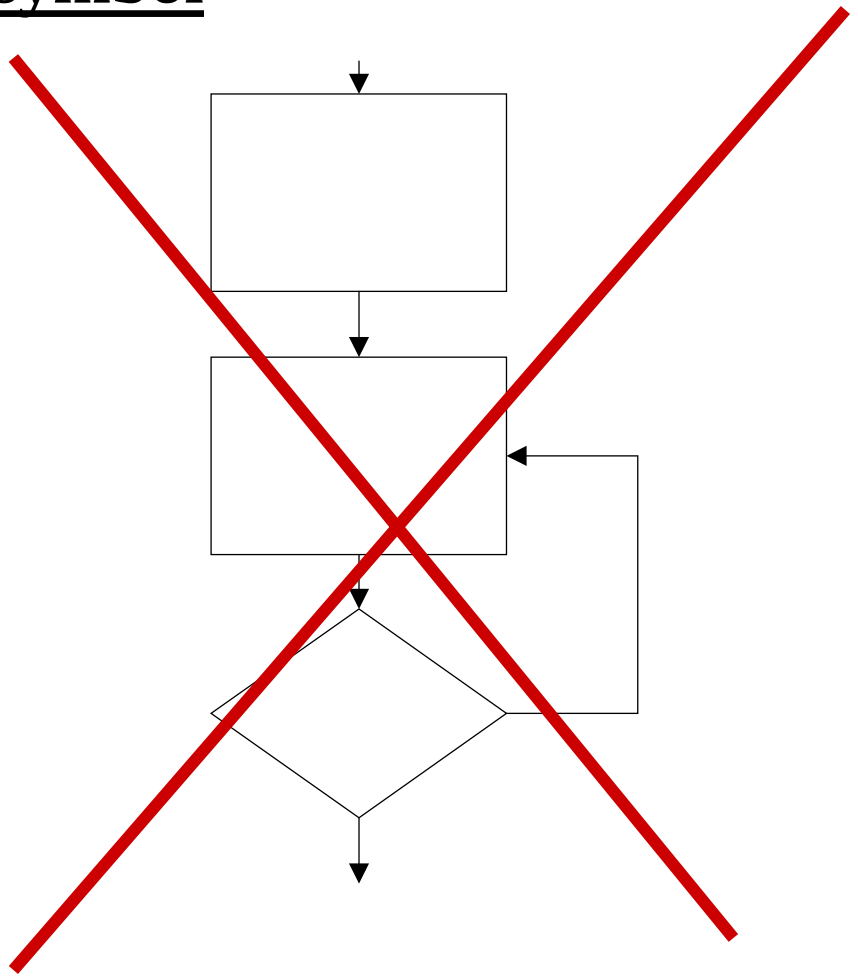




7) Flow Chart (cont.) More Rules...

Avoid multiple flowlines entering a symbol



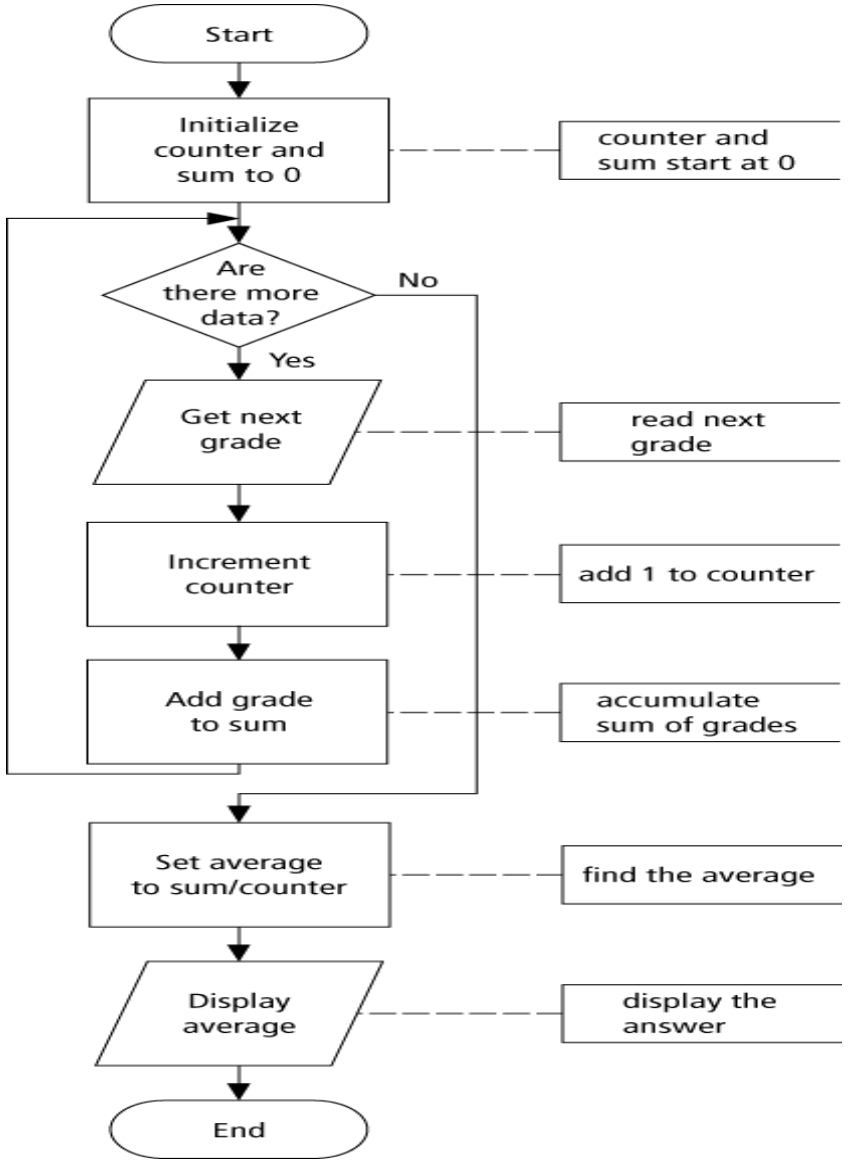


Examples

Class Average Algorithm

- Problem: Calculate and report the grade-point average for a class.
- Discussion: The average grade equals the sum of all grades divided by the number of students.
- Output: Average grade.
- Input: Student grades.
- Processing: Find the sum of the grades; count the number of students; calculate average.

Class Average flowchart



Thank you