Module - I

Introduction to programming, Classification of computer languages, Language translators (Assembler, Compiler, Interpreter), Linker, Characteristics of a good programming language, Factors for selecting a language, Subprogram, Purpose of program planning, Algorithm, Flowchart, Pseudocode, Control structures (sequence, selection, Iteration), Testing and debugging.

What is a Programming Languages

- A programming language is a set of rules that provides a way of telling a computer what operations to perform.

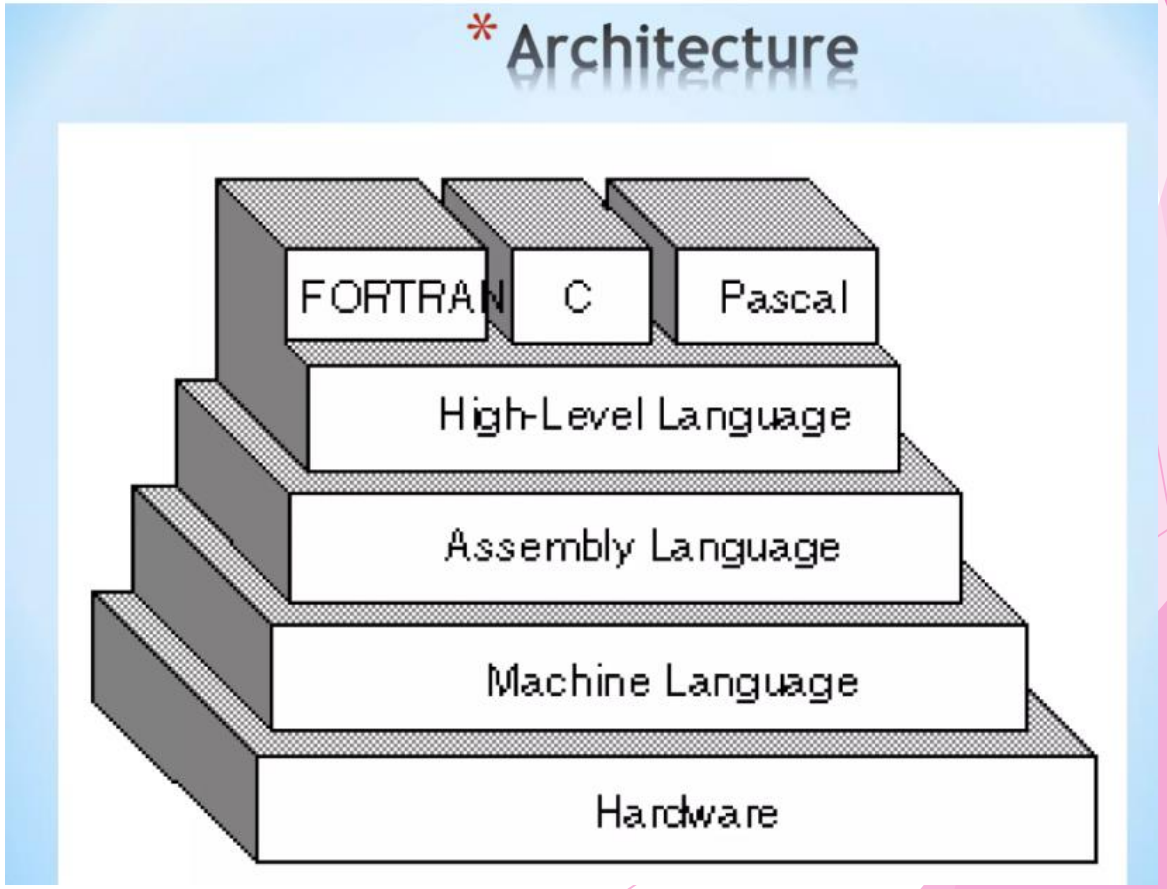
A programming language is a tool for developing executable models for a class of problem domains.

- A programming language also has words, symbols and rules of grammar.
- The grammatical rules are called syntax.
- Each programming language has a different set of syntax rules.

Classification of computer languages

Computer languages are classified into three categories

1. Machine Language
2. Assembly Language
3. High-Level Language



Machine language

Machine code or machine language is a system of instructions and data executed directly by a computer's CPU. The lowest-level programming language that only be understood by computers.

Computer language that is directly executable by a computer without the need for translation by a compiler or an assembler.

- A machine language instruction normally has a two part format.
- The first part is operation code that tells the computer what function to perform.
- The second part is operand that tells where to find or store the data on which the computer has to perform the function.

OPCODE	OPERAND

All computers use binary digits (0s and 1s) for performing internal operations. Hence most computer's machine language instructions consist of binary numbers.

Machine Language

The native language of the computer,

The set of symbolic instructions in binary that is used to represent operations and data in a machine called machine code

Machine Language: "0110101100101000"

machine language is a collection of binary digits or bits that the computer reads and interprets.

Machine language is the only language a computer understands. It is almost impossible for humans to use because they consist entirely of numbers.

Machine level language

- Machine level language is a set of instruction or codes which are directly understand by computer with out help of translator. It is combination of 0 and 1.

Advantages

- It is Witten in machine code(0 and 1) so no need to translate.
- It is faster then other language.

Disadvantage

- It is difficulty to understand and develop the program by using this language.
- The knowledge of computer architecture is required.
- Debugging is difficult.

Assemble language

- Assemble language is also known as low level language. Which uses the mnemonics code as a instruction. The language which uses the mnemonics codes and symbol to develop any program is called Assemble language. This language uses some mnemonics codes they are ADD for addition, SUB for subtraction and MUL for multiplication , LDA for load accumulator etc.

Advantage

- .Less time to consumed respect to machine code.
- Coding is faster then machine code because mnemonics codes are use.
- Debugging is easier then machine code.

Disadvantage

- Machine oriented language.
- The knowledge of computer hardware is required.
- This language is not understand by hardware so we need to translator like assembler

High Level Languages

- High Level Language: High Level Language: are problem oriented language. Most programs are written in high level language which are quite similar to English languages that's why they are easier then machine level language. This programming language is used to develop different softwares. For example C,C++,JAVE etc.

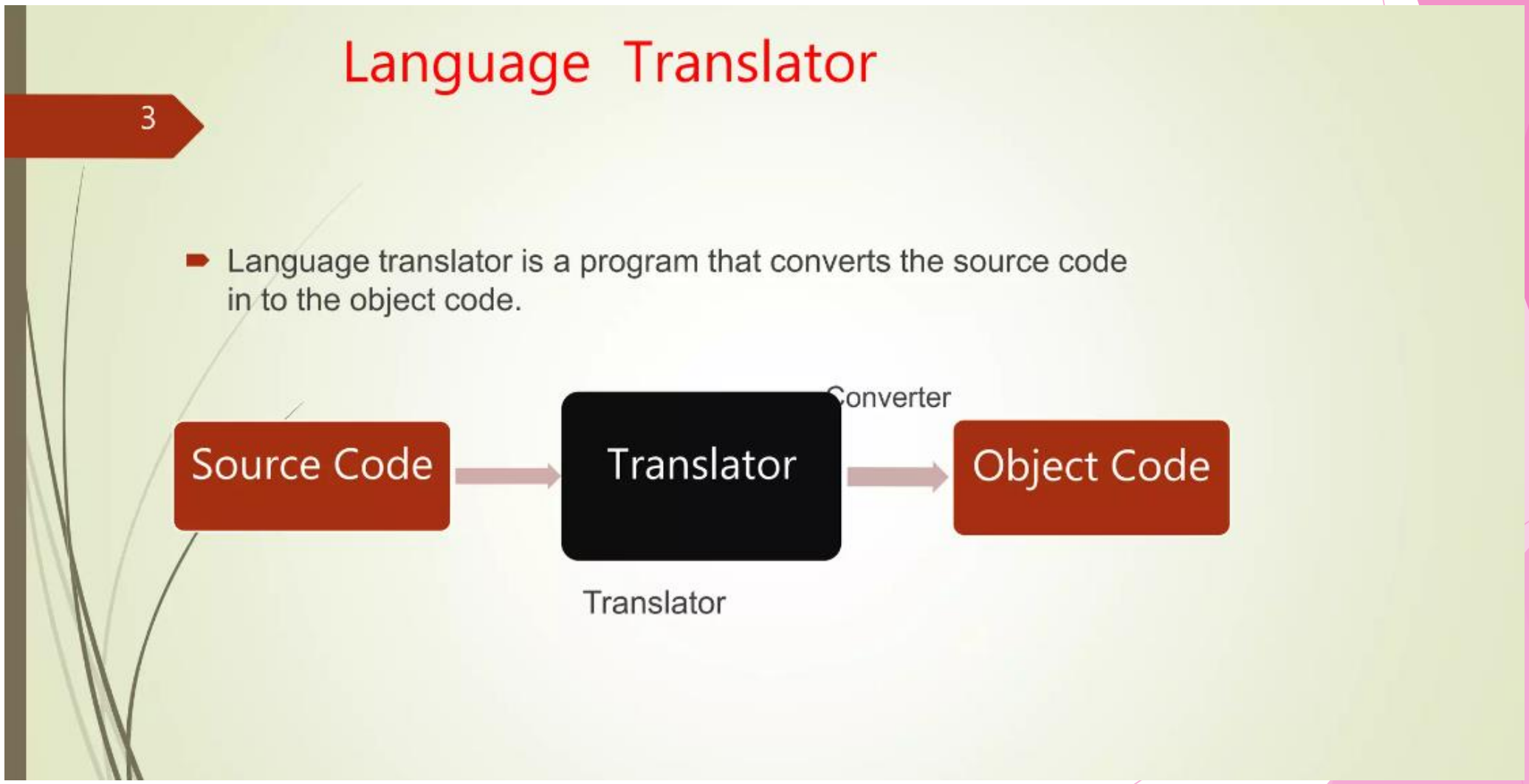
Advantages

- Simple English is used for programming coding.
- Machine independent.
- The knowledge of computer architecture is not required.
- It requires less time for program coding.
- Program can be debugging easily because the code is written in simple English language .

Disadvantage

- This language is not easily understand by computer hardware so we need to translate this language in to machine code. By the help of compiler.
- The program execution is slower then machine language.
- The conversion time is slower then Assemble language.

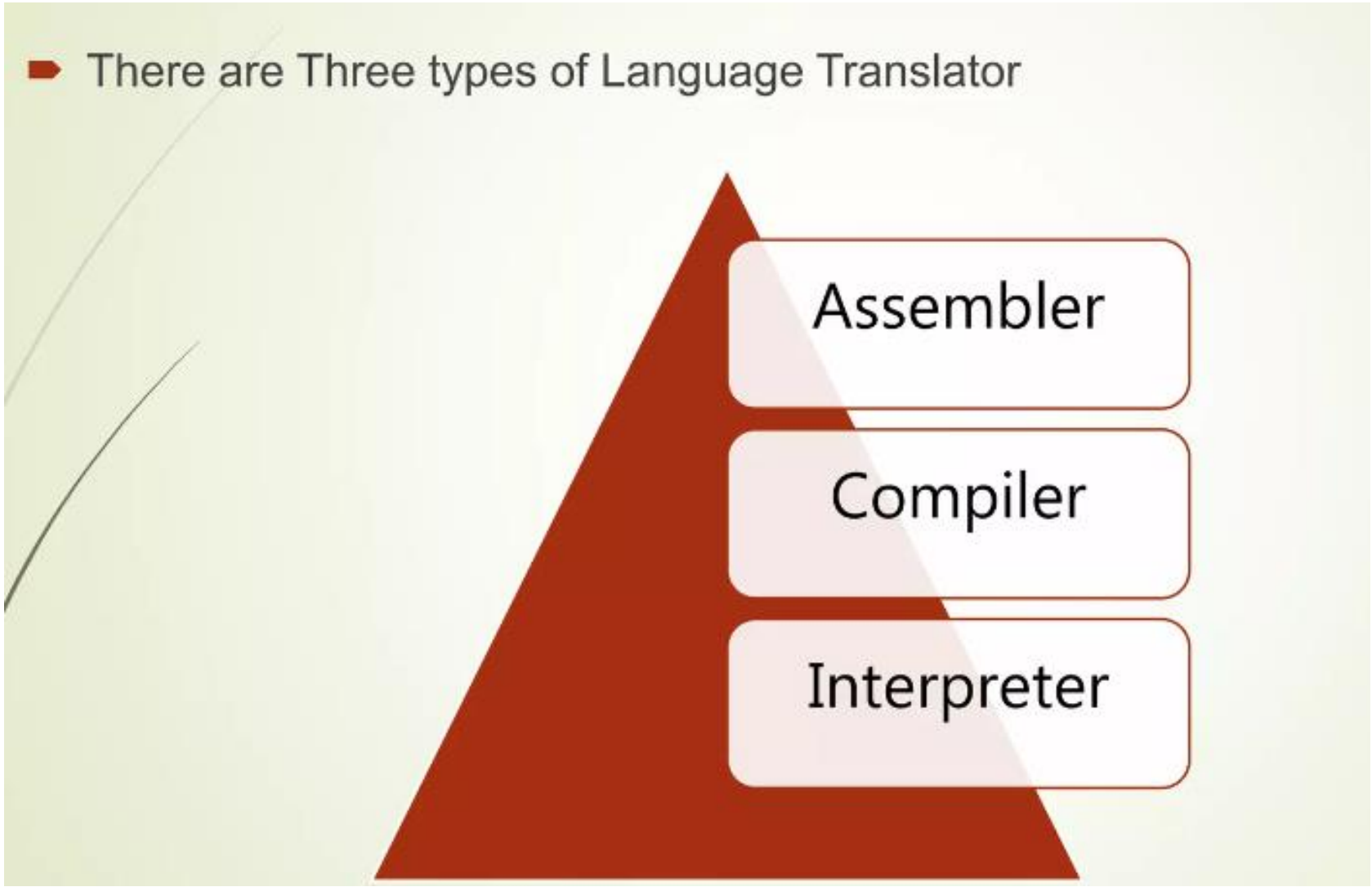
Language translators



Why Language Translators?

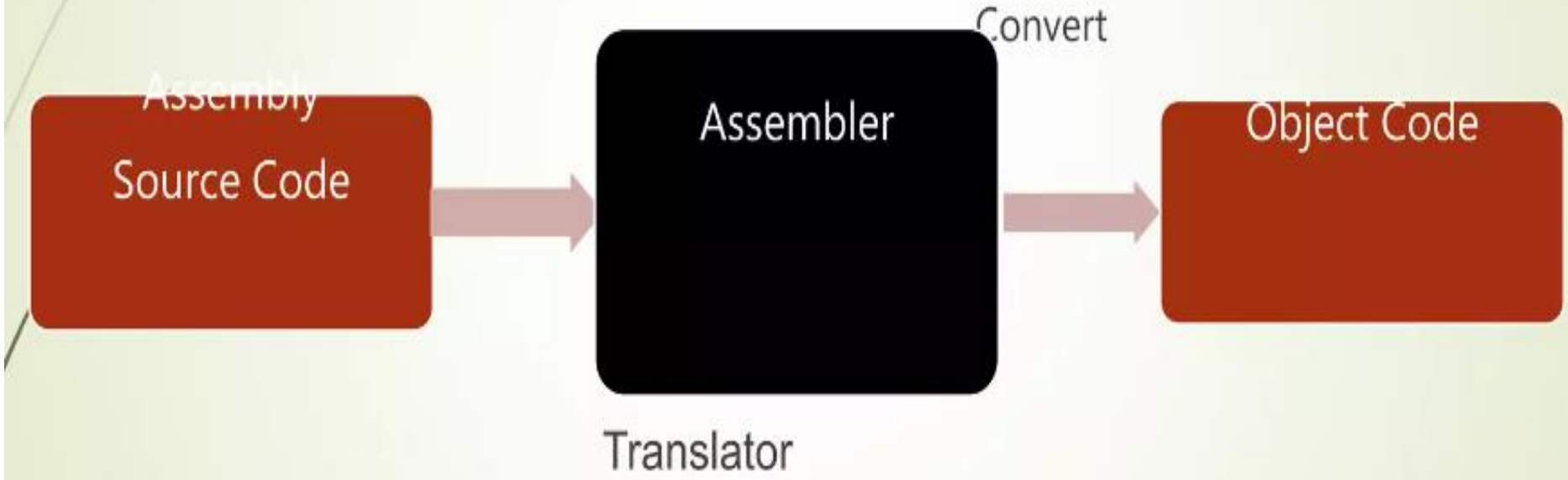
- Computer only understands object code (machine code).
- It does not understand any source code.
- There must be a program that converts source code in to the object code so that the computer can understand it.
- The language translator is one which does this job.
- The programmer writes the source code and then translator converts it in machine readable format (object code).

- There are Three types of Language Translator



Assembler

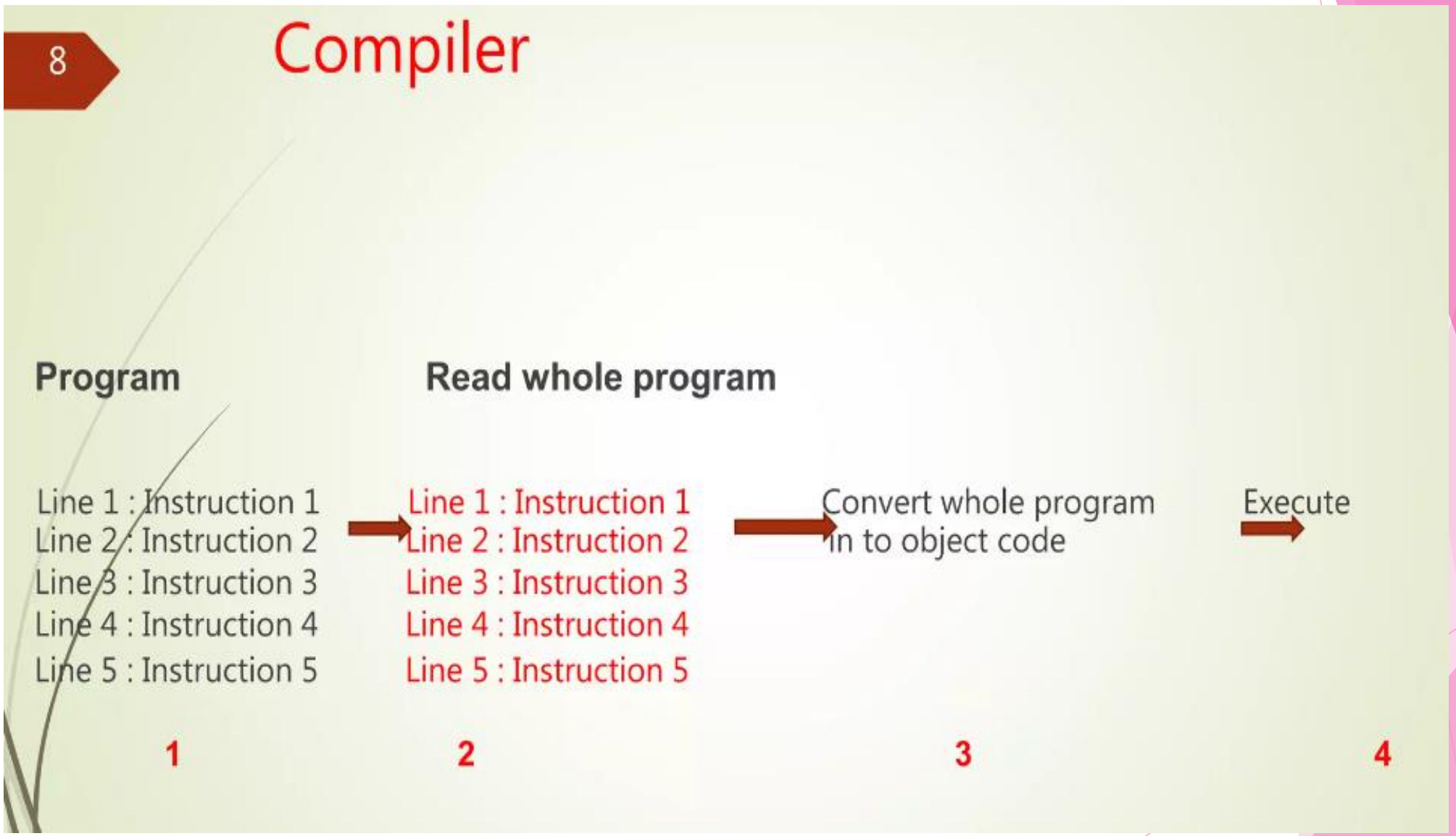
- Assembler is the language translator that converts assembly language code in to the object code (machine code).



Compiler

- Compiler is the language translator that converts high level language code in to the object code (machine code).
- It converts the whole code at a time.

Compiler


Program
Line 1 : Instruction 1
Line 2 : Instruction 2
Line 3 : Instruction 3
Line 4 : Instruction 4
Line 5 : Instruction 5
1
Read whole program
Line 1 : Instruction 1
Line 2 : Instruction 2
Line 3 : Instruction 3
Line 4 : Instruction 4
Line 5 : Instruction 5
2
Convert whole program in to object code
3
Execute
4

Interpreter

- Interpreter is the language translator that converts high level language code in to the object code (machine code).
- It converts the code line by line.

Interpreter

Program

Line 1 : Instruction 1 — **Read Line 1** — **Convert in to Object code** — **Execute** — **2**

Line 2 : Instruction 2 — **Read Line 2** — **Convert in to Object code** — **Execute** — **3**

Line 3 : Instruction 3 — **Read Line 3** — **Convert in to Object code** — **Execute** — **4**

Line 4 : Instruction 4 — **Read Line 4** — **Convert in to Object code** — **Execute** — **5**

Line 5 : Instruction 5 — **Read Line 5** — **Convert in to Object code** — **Execute** — **6**

1

11

Difference between Compiler and Interpreter

Compiler

- It converts whole code at a time.
- It is faster.
- Requires more memory.
- Errors are displayed after entire program is checked.
- Example: C, C++, JAVA.

Interpreter

- It converts the code line by line.
- It is slower.
- Requires less memory.
- Errors are displayed for every instruction interpreted (if any)
- Example: GW BASIC, Ruby, Python

Linker

Introduction

Linker

Linker is a program that takes one or more objects generated by a compiler and combines/assembles them into a single executable program.

Loader

Loader is a program that is responsible for loading programs from executables into memory, preparing them for execution and then executing them.

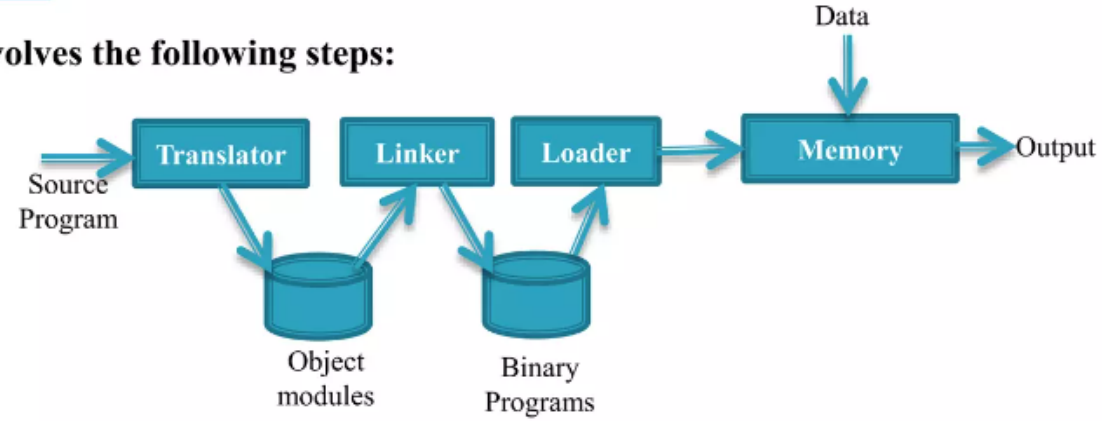
The execution of a program involves the following steps:

Step 1: Translation

Step 2: Linking

Step 3: Relocation

Step 4: Loading



Characteristics of a good programming language

Selection Criteria for a Programming Language

1. **Usability**
   Easy to learn, ease of use for an experienced programmer.
2. **Performance**
   Speed of program execution, speed of compiler execution (a program which translates the program into machine code), stability (lack of defects).
3. **Portability**
   A portable language is one which is implemented in variety of computers (design relatively machine dependent). Well defined language are more portable than others e.g. C, C++.
4. **Extendibility**
   Possibility of developing the language and its implementation, existence function libraries, class libraries, etc.
5. **Continuity**
   Continuity of the manufacturer, language continuity, implementation continuity, existence of international standards for defining the language, conformity of implementation by following standards, existence of other manufacturers for that language.

Factors for selecting a language,

Targeted platform: The most important thing to decide is how and where the program would function. It is well known that not all the languages can run in all types of platforms. When the program is written in the renowned C language, it requires compilers to function on Linux and Windows-based systems.

Efficiency: It is important for the compilers to match with the language you are selecting. It should be efficient so that it helps in making the language to function fast.

Performance and elasticity: When you are choosing a language, you have to remain flexible enough. By being flexible, you can add extra features and programs in it.

Tool support: It is recommended to purchase tool oriented language which provides several elements and also methods to control, work and edit.

Subprograms

In computer science, a **subroutine** or **subprogram** (also called **procedure**, **method**, **function**, or **routine**) is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code.

As the name "subprogram" suggests, a subroutine behaves in much the same way as a computer program that is used as one step in a larger program or another subprogram.

A subroutine is often coded so that it can be started ("*called*") several times and/or from several places during a single execution of the program, including from other subroutines, and then branch back (return) to the next instruction after the "call" once the subroutine's task is done.

There are two distinct categories of subprograms:

1. Procedures
2. Functions.

Function

In a programming language, function is said to be a set of instructions that take some input and execute some tasks. A function can either be predefined or user-defined. In the C program, a function can be called multiple times to provide reusability and modularity. It may or may not return a value.

Procedure

It is an important programming construct for a compiler. The procedure is used for generating good code for procedure calls and returns. It does not deal with an expression. It is defined as the set of commands that are executed in order.

In DBMS, a procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements. In structured query language (or SQL), it does not return a value. In Java, both function and procedure are the same and called sub-routines.

Purpose of program planning

- Planning is the first step in developing and clarifying the concept for a program.
- In addition to writing down the idea for their program, students may also draw and label a diagram of what the program will look like.
- Planning software can be a powerful tool to organize projects and resources into phases and tasks with attainable deadlines – all in one place

Algorithm

An algorithm describes the step by step action to solve a problem.

An algorithm has a well defined sequence of steps, it gives you an output, and it will eventually terminate.

PROPERTIES OF THE ALGORITHM

1) **Finiteness**: - an algorithm terminates after a finite numbers of steps.

2) **Definiteness**: - each step in algorithm is unambiguous. This means that the action specified by the step cannot be interpreted (explain the meaning of) in multiple ways & can be performed without any confusion.

3) **Input**:- an algorithm accepts zero or more inputs

4) **Output**:- it produces at least one output.

5) **Effectiveness**:- it consists of basic instructions that are realizable. This means that the instructions can be performed by using the given inputs in a finite amount of time.

Eg. Develop an Algorithm to find the average of three numbers taken as input from the user.

Algorithm AVERAGE

Step 0 START

Step 1 INPUT first number into variable A

Step 2 INPUT second number into variable B

Step 3 INPUT third number into variable C

Step 4 COMPUTE sum= A+B+C

Step 5 COMPUTE Average = sum/3

Step 6 DISPLAY Average

Step 7 END

Eg. Develop an Algorithm to divide one number by another and find the quotient

Algorithm DIVISION

Step 0 START

Step 1 INPUT first number into variable A

Step 2 INPUT second number into variable B

Step 3 If B!=0 then

Q=A/B

DISPLAY Q

End if

Step 4 END

Questions

1. Develop an algorithm to find the maximum of two input numbers.
2. Write an algorithm to check whether the input number is even or odd.
3. Write an algorithm to check whether the input number is divisible by 5.
4. Write an algorithm to check whether the input number is a multiple of 6.
5. Write an algorithm to find the product of three input numbers.

FLOWCHART

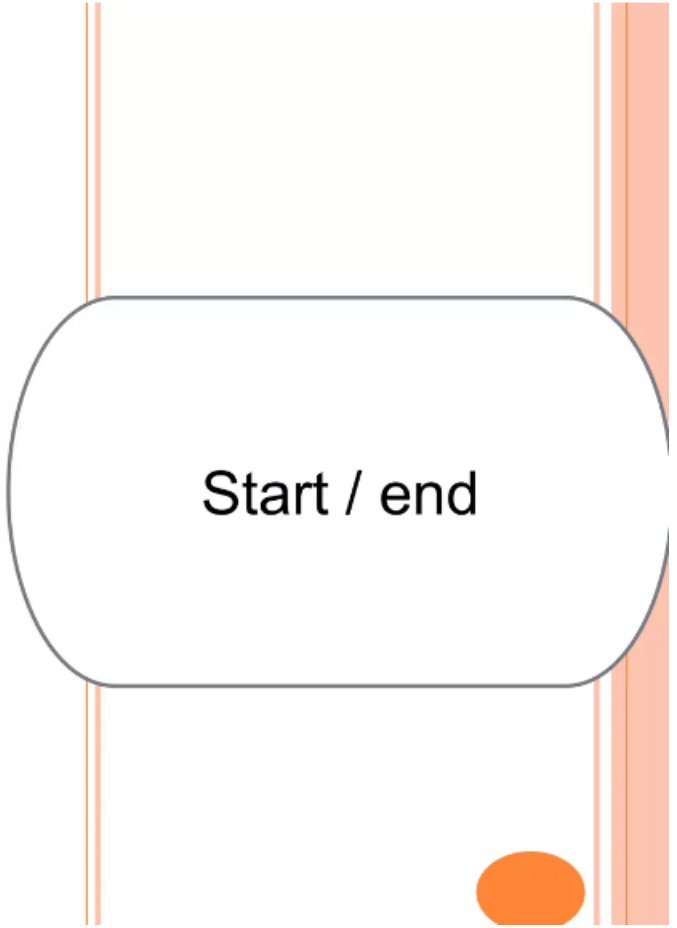
Is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

Flow chart are used in designing and documenting complex processes or programs.

Symbols of Flow chart

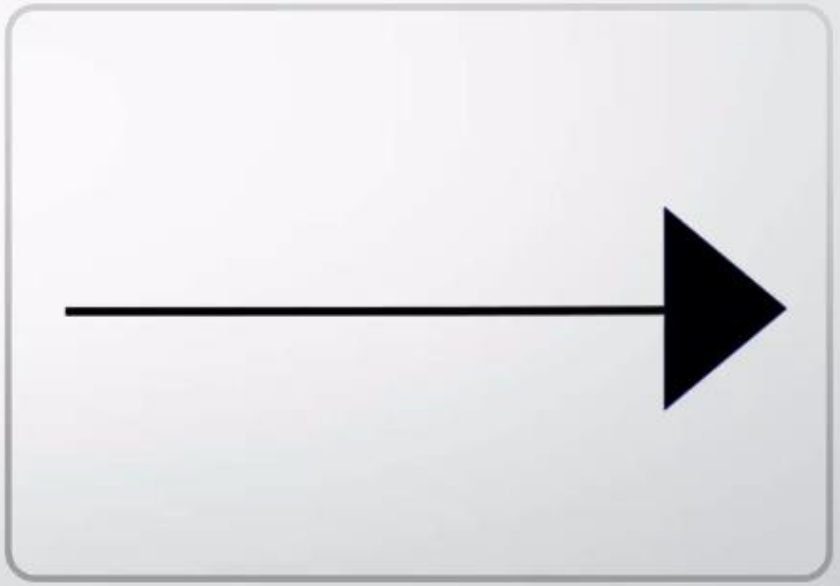
Terminator

Represented as circles , ovals or rounded rectangles, usually containing the word “Start” or “End” , or another phrase signalling the start or end of a process , such as “submit inquiry” or “receive product”.



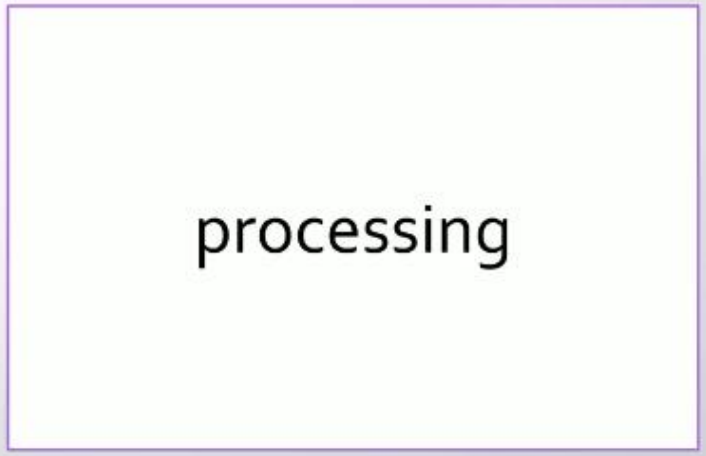
Arrows

- Showing "flow of control".
- An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to .
- The line for the arrow can be solid or dashed.
- The meaning of the arrow with dashed line may differ from one flowchart to another and can be defined in the legend



Processing

- Represented as rectangles.
- Use it to represent an event which is controlled within the process.
- Typically this will be a step or action which is taken.
- In most flowcharts this will be the most frequently used symbol.
- Examples: " Add 1 to X"; "replace identified part"; "save changes" or similar.



INPUT / OUTPUT

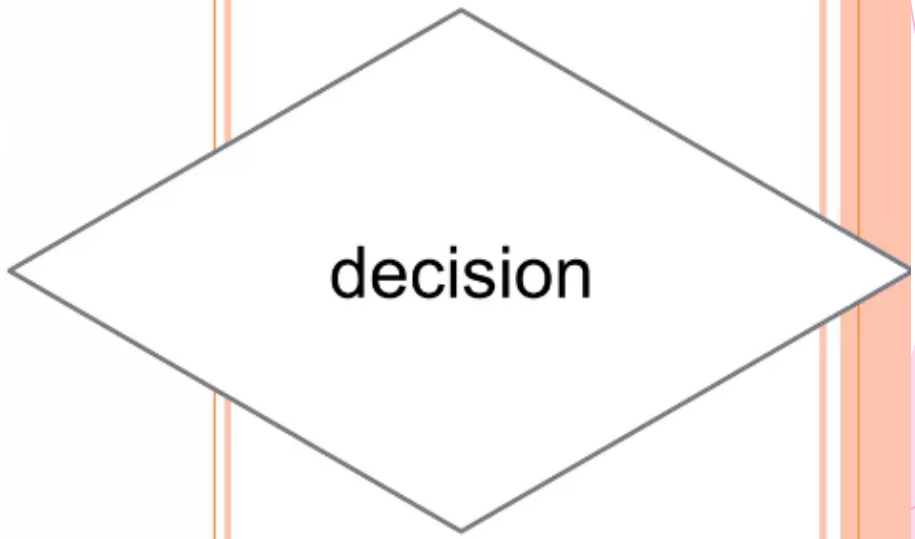
Represented as a parallelogram.

Represents material or information entering or leaving the system, such as customer order (input) or a product (output).

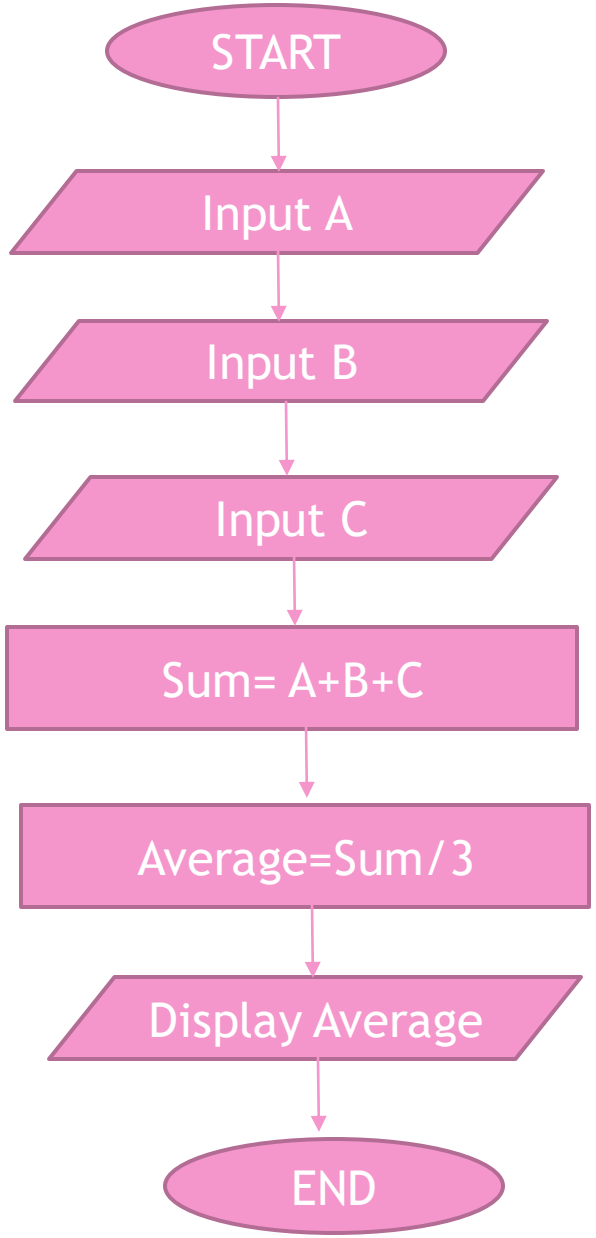
Example: Get X from the user; display X.

Decision

A decision or branching point. Lines representing different decisions emerge from different points of the diamond.



Average of three numbers



Testing and Debugging

Program Errors

- Compiler errors (syntax errors)
- Runtime errors
- Logic errors

Compiler Errors

- ***Syntax error***
  - Error in usage of Code
  - Detected by the compiler
  - A program with compilation errors cannot be run
- ***Syntax warning***
  - Warning message generated by the compiler
  - The program can be run

Compiler Errors

- Very common (but sometimes hard to understand). Examples of syntax errors:
  - Forgetting a semicolon
  - Leaving out a closing bracket }
  - Redeclaring a variable
  - Others?

Compiler Errors

- Hints to help find/fix compiler errors:
  - Compiler errors are cumulative: when you fix one, others may go away
  - Read the error messages issued by the compiler
  - Realize that the error messages from the compiler are always not very helpful
    - The compiler does not know what you intended to do, it merely scans the code

Runtime Errors

- ***Runtime error***: program runs but gets an *exception* error message
  - Program may be terminated
- Runtime errors can be caused by
  - Program bugs
  - Bad or unexpected input
  - Hardware or software problems in the computer system

Logic Errors

- **Logic error**: program runs but results are not correct
- Logic errors can be caused by:
  - incorrect algorithms
- Very common logic errors are:
  - using == instead of the *equals* method
  - infinite loops
  - misunderstanding of operator precedence
  - starting or ending at the wrong index of an array
    - If index is invalid, you would get an exception
  - misplaced parentheses (so code is either inside a block when it shouldn't be, or vice versa)

Thank You