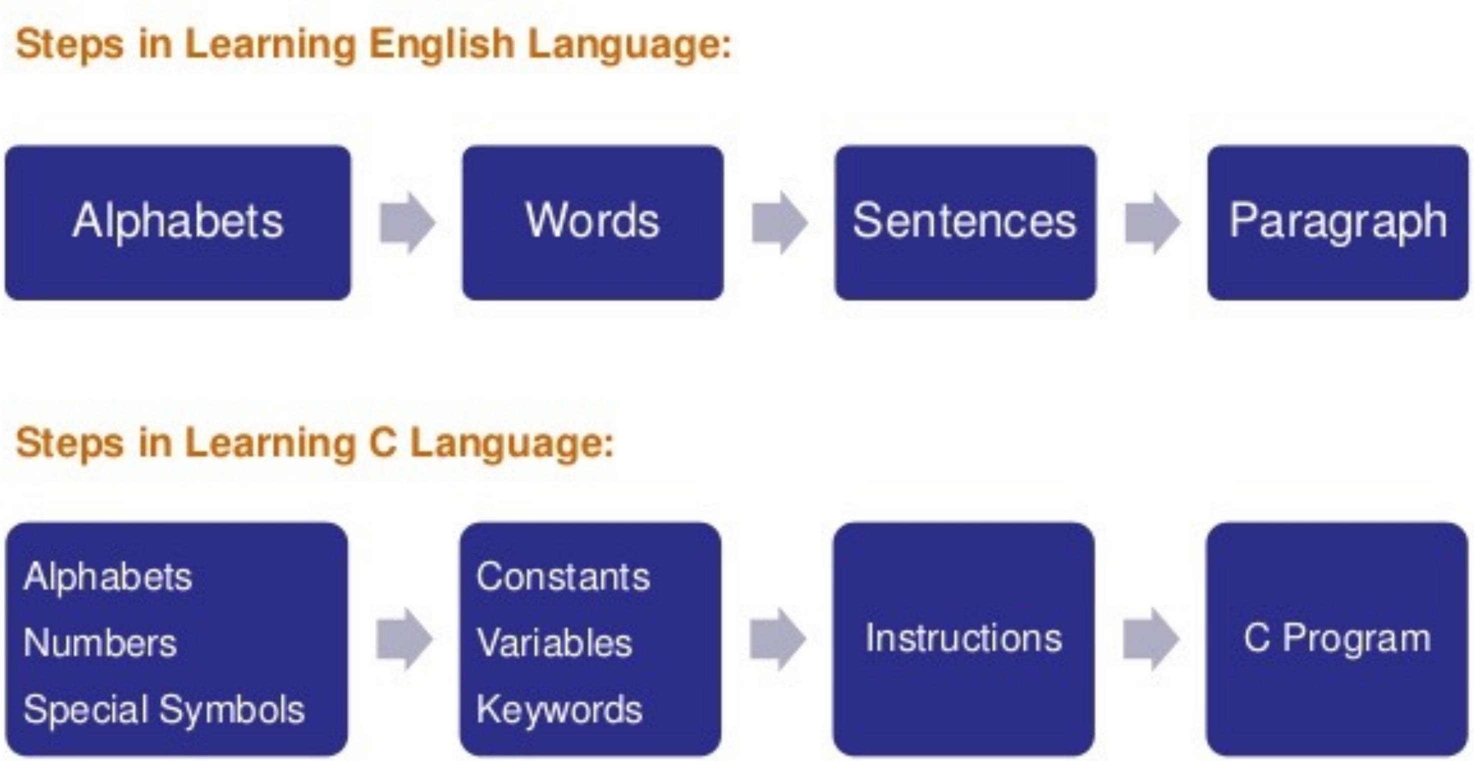
Unit 1 : Introduction to C Programming

Er. Nipun Thapa

1.1. Introduction and History

- **C** is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie.
- In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc
- It was initially designed for programming UNIX operating system. Now the software tool as well as the C compiler is written in C. Major parts of popular operating systems like Windows, UNIX, Linux is still written in C.
- This is because even today when it comes to performance (speed of execution) nothing beats C.
- Moreover, if one is to extend the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C.
- C seems so popular is because it is **reliable**, **simple** and **easy** to use. Often heard today is – "C has been already superceded by languages like C++, C# and Java.

1.1. Introduction to Programming Language



Steps in Learning English Language:

Alphabets → Words → Sentences → Paragraph

Steps in Learning C Language:

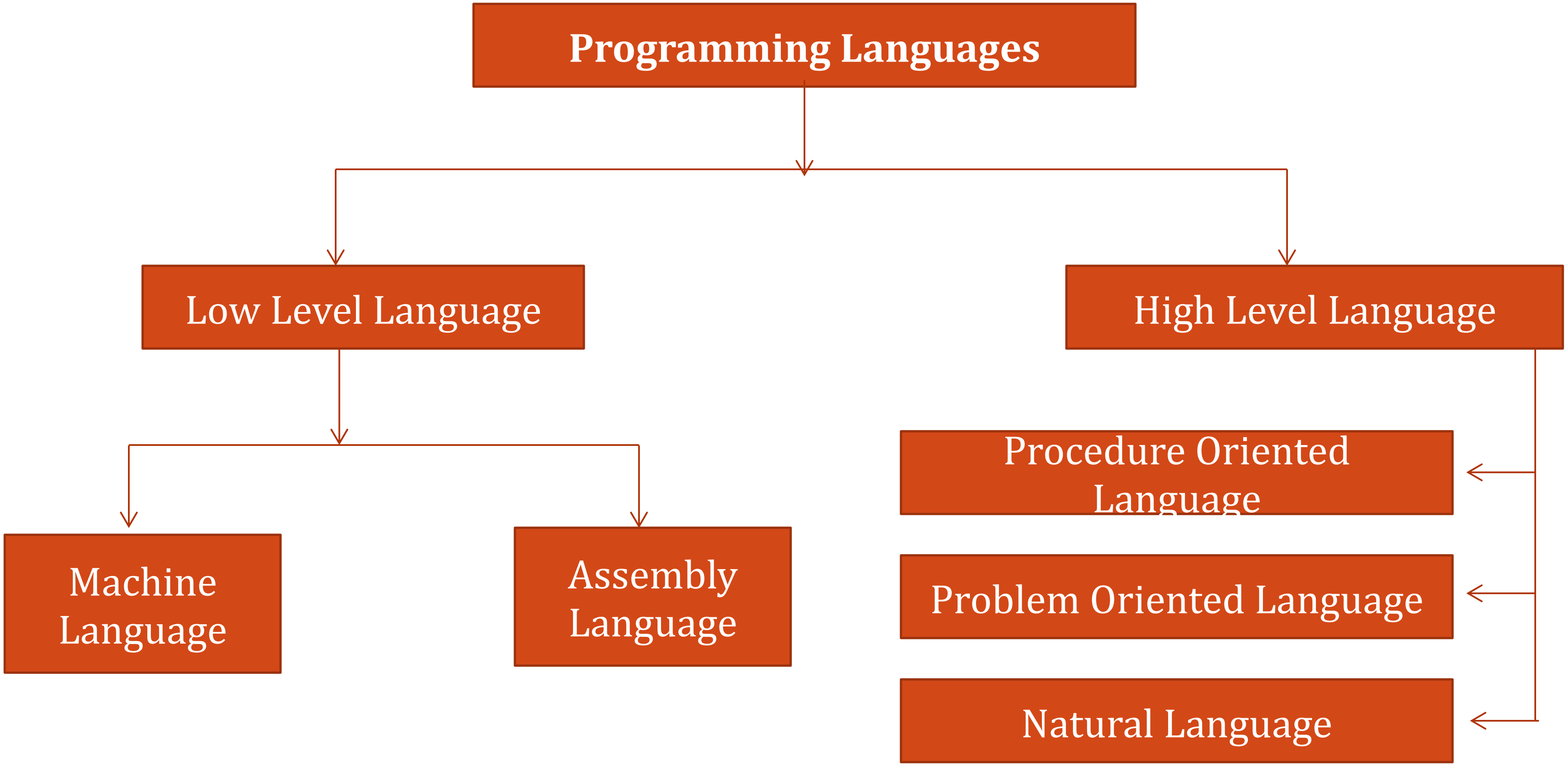
Alphabets, Numbers, Special Symbols → Constants, Variables, Keywords → Instructions → C Program

Fig. 1.1. Steps in C program

1.1. Introduction to Programming Language

- Learning C is similar and easier.
- Instead of straight-away learning how to write programs, we must first know what **alphabets**, **numbers** and **special symbols** are used in C, then how using them **constants**, **variables** and **keywords** are constructed, and finally how are these combined to form an **instruction**.
- A group of instructions would be combined later on to form a **program**.
- So a computer ***program*** is just a collection of the instructions necessary to solve a specific problem.
- The basic operations of a computer system form what is known as the computer's ***instruction set.*** And the approach or method that is used to solve the problem is known as an ***algorithm***.

1.2. Types of Programming Language


Programming Languages

Low Level Language

High Level Language

Machine Language

Assembly Language

Procedure Oriented Language

Problem Oriented Language

Natural Language

1.2. Types of Programming Language

So for as programming language concern these are of two types.

- **Low level language**
- **High level language**

1.2. Types of Programming Language

1. Low level language:

- Low level languages are **machine level** and **assembly level language**.
- In machine level language computer only understand digital numbers i.e. in the form of 0 and 1. So, instruction given to the computer is in the form binary digit, which is difficult to implement instruction in binary code.
- This type of program is not portable, difficult to maintain and also error prone. The **assembly language** is on other hand modified version of machine level language.
- Where instructions are given in English like word as ADD, SUM, MOV etc. It is easy to write and understand but not understand by the machine. So the translator used here is assembler to translate into machine level.

1.2. Types of Programming Language

2. High level language:

- These languages are machine independent, means it is portable. The language in this category is Pascal, Cobol, Fortran etc.
- High level languages are understood by the machine. So it need to translate by the translator into machine level.
- A translator is software which is used to translate high level language as well as low level language in to machine level language.

1.3. Language Processor

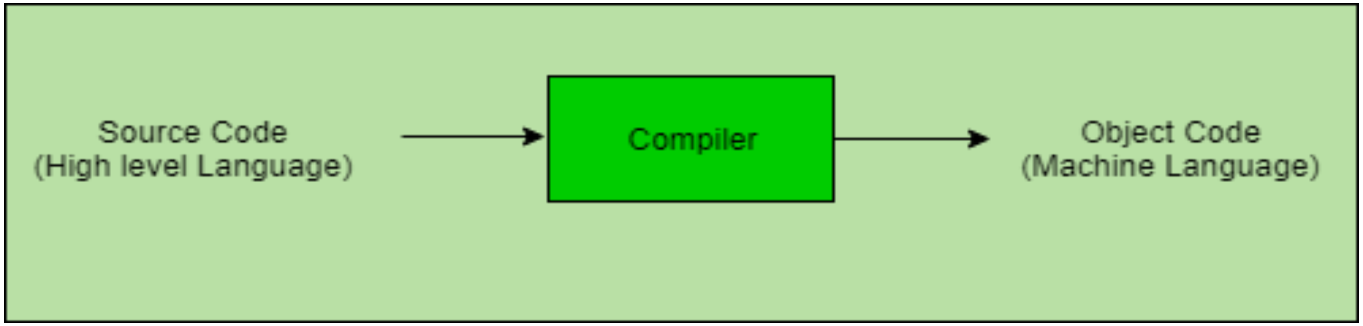
Three types of translator are there:

- **Compiler**
- **Interpreter**
- **Assembler**

1.3. Language Processor

1. Compiler

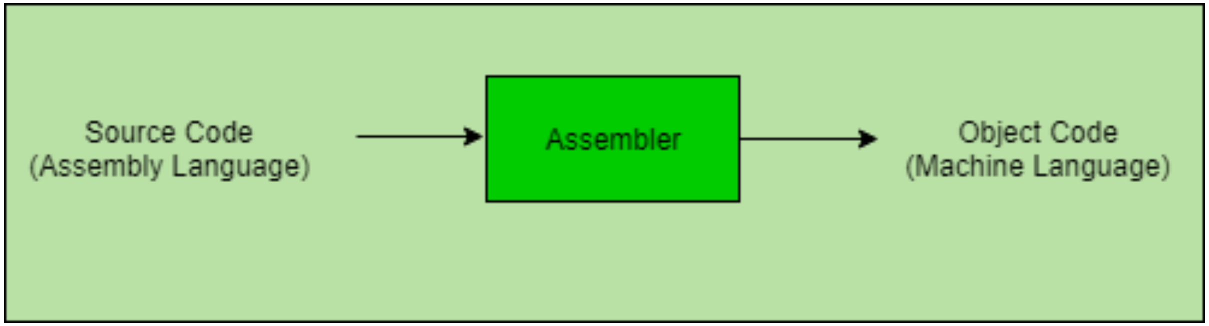
- The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler.
- **Example:** C, C++, C#, JavaIn a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully recompile the source code again.



1.3. Language Processor

2. Assembler

- The Assembler is used to translate the program written in Assembly language into machine code.
- The source program is a input of assembler that contains assembly language instructions.
- The output generated by assembler is the object code or machine code understandable by the computer.



1.3. Language Processor

3. Interpreter

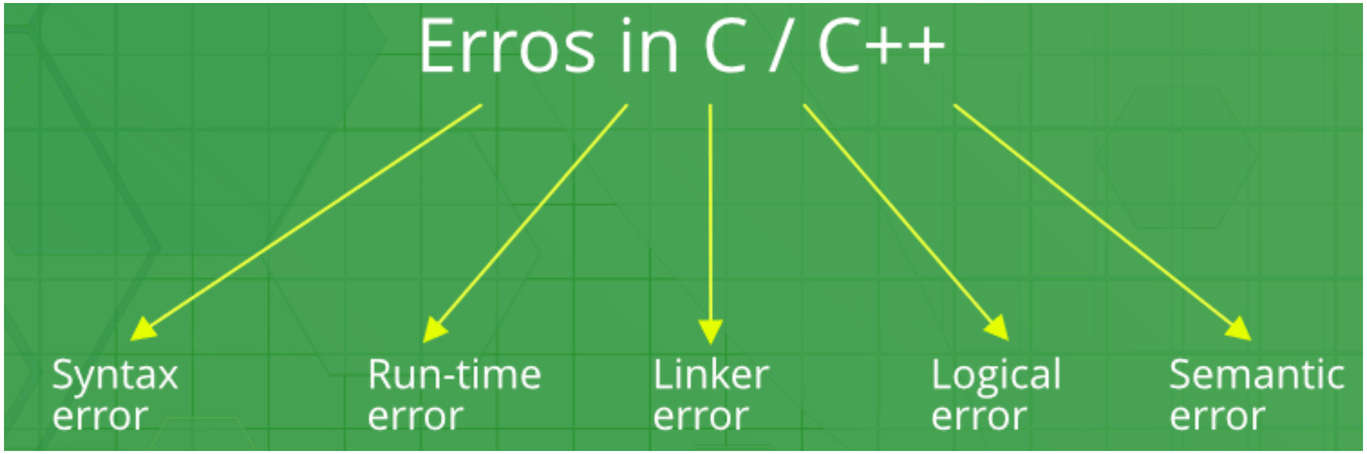
- The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter.
- If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message.
- The interpreter moves on to the next line for execution only after removal of the error.
- An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. **Example:** Perl, Python and Matlab.

1.3. Language Processor

COMPILER	INTERPRETER
1. A compiler is a program which coverts the entire source code of a programming language into executable machine code for a CPU.	1. interpreter takes a source program and runs it line by line, translating each line as it comes to it.
2. Compiler takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	2. Interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
3. Compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present any where in the program.	3. Its Debugging is easier as it continues translating the program until the error is met
4. Generates intermediate object code.	4. No intermediate object code is generated.
5. Examples: C, C++, Java	5. Examples: Python, Perl

1.4. Program Errors

- Error is an illegal operation performed by the user which results in abnormal working of the program.
- Programming errors often remain undetected until the program is compiled or executed.
- Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.
- The most common errors can be broadly classified as follows.



1.4. Program Errors

1. Syntax errors:

- Errors that occur when you **violate the rules** of writing C/C++ syntax are known as syntax errors. This compiler error indicates something that must be fixed before the code can be compiled. All these errors are detected by compiler and thus are known as compile-time errors. Most frequent syntax errors are:
  - Missing Parenthesis (})
  - Printing the value of variable without declaring it
  - Missing semicolon like this:// C program to illustrate

```
// syntax error
#include<stdio.h>
int main()
{
   int x = 10;
   int y = 15;
   printf("%d", (x, y)) // semicolon missed
    retutn 0;
}  Error:
```

1.4. Program Errors

2. Run-time Errors :

- Errors which occur during program execution(run-time) after successful compilation are called run-time errors. One of the most common run-time error is division by zero also known as Division error. These types of error are hard to find as the compiler doesn’t point to the line at which the error occurs. For more understanding run the example given below.

```
#include<stdio.h>
int main()
{
    int n = 9, div = 0;
     // wrong logic
    // number is divided by 0,
    // so this program abnormally terminates
    div = n/0;
     printf("resut = %d", div);
     return 0;
}
```

1.4. Program Errors

3. Logical Errors :

- On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appears to be error free are called logical errors. These are one of the most common errors done by beginners of programming. These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

```
// C program to illustrate
// logical error
int main()
{
   int i = 0;
     // logical error : a semicolon after
loop
   for(i = 0; i < 3; i++);
   {
      printf("loop ");
      continue;
   }
   getchar();
   return 0;
}
```

1.5. Features of good Program

- Every computer requires appropriate instruction set (programs) to perform the required task.
- The quality of the processing depends upon the given instructions.
- If the instructions are improper or incorrect, then it is obvious that the result will be superfluous.
- Therefore, proper and correct instructions should be provided to the computer so that it can provide the desired output.
- Hence, a program should be developed in such a way that it ensures proper functionality of the computer. In addition, a program should be written in such a manner that it is easier to understand the underlying logic.

1.5. Features of good Program

- A good computer program should have following characteristics:
  1. **Portability**
  2. **Readability**
  3. **Efficiency**
  4. **Structural**
  5. **Flexibility**
  6. **Generality**
  7. **Documentation**

1.5. Features of good Program

1. Portability:

- Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes.
- Due to rapid development in the hardware and the software, nowadays platform change is a common phenomenon.
- Hence, if a program is developed for a particular platform, then the life span of the program is severely affected.

1.5. Features of good Program

2. Readability:

- The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort.
- If a program is written structurally, it helps the programmers to understand their own program in a better way.
- Even if some computational efficiency needs to be sacrificed for better readability, it is advisable to use a more user-friendly approach, unless the processing of an application is of utmost importance.

1.5. Features of good Program

3. Efficiency:

- Every program requires certain processing time and memory to process the instructions and data.
- As the processing power and memory are the most precious resources of a computer, a program should be laid out in such a manner that it utilizes the least amount of memory and processing time.

1.5. Features of good Program

4. Structural:

- To develop a program, the task must be broken down into a number of subtasks.
- These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask.
- If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier.

1.5. Features of good Program

5. Flexibility:

- A program should be flexible enough to handle most of the changes without having to rewrite the entire program.
- Most of the programs are developed for a certain period and they require modifications from time to time.
- For example, in case of payroll management, as the time progresses, some employees may leave the company while some others may join.
- Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.

1.5. Features of good Program

6. Generality:

- Apart from flexibility, the program should also be general. Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain.
- For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.

1.5. Features of good Program

7. Documentation:

- Documentation is one of the most important components of an application development.
- Even if a program is developed following the best programming practices, it will be rendered useless if the end user is not able to fully utilize the functionality of the application.
- A well-documented application is also useful for other programmers because even in the absence of the author, they can understand it.

1.6. Introduction to Program Technique

- C is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system.
- C is the most widely used computer language.
- It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

1.6. Top down and Bottom up Approach

Top down Approach

- The basic task of a top-down approach is to divide the problem into tasks and then divide tasks into smaller sub-tasks and so on.
- Each part of it then refined into more details, defining it in yet more details until the entire specification is detailed enough to validate the model.
- It break the problem into parts, Then break the parts into parts soon and now each of part will be easy to do.

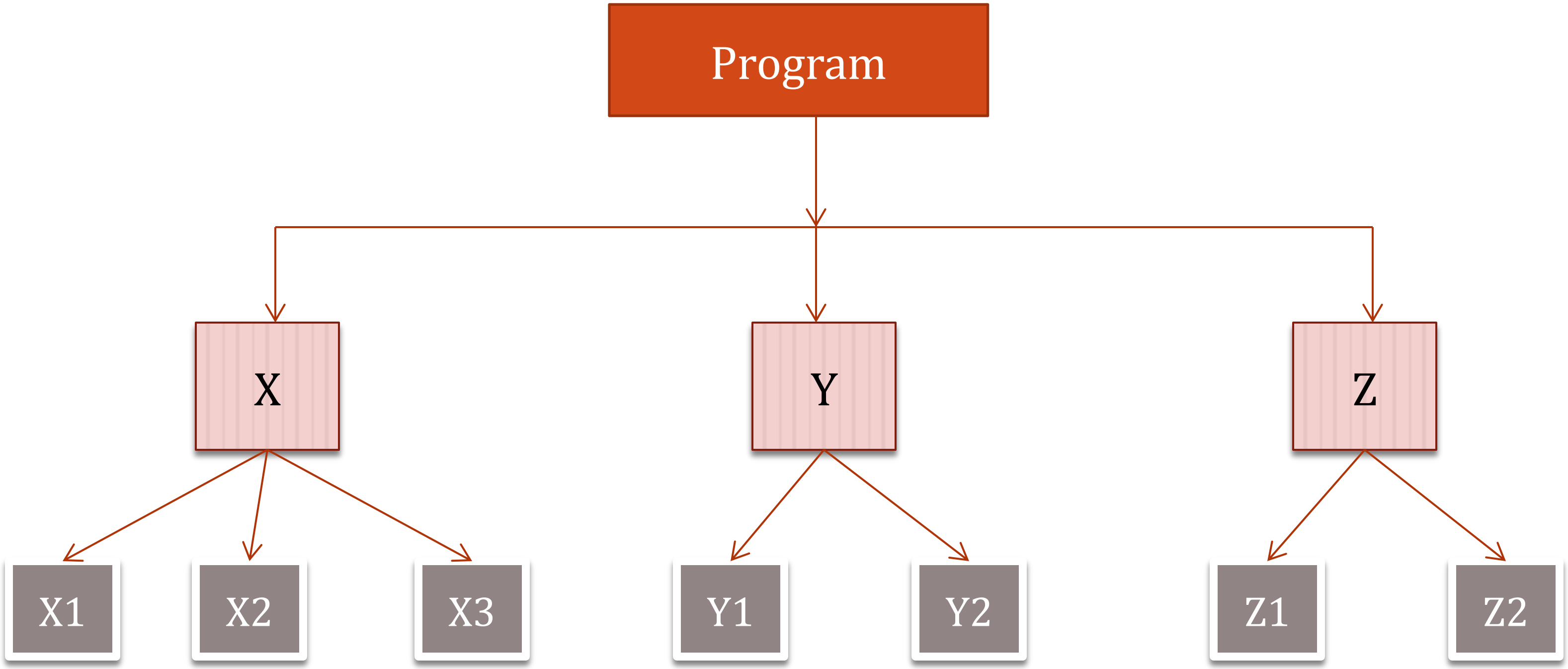
1.6. Top down and Bottom up Approach

Top down Approach

- C programming language supports this approach for developing projects.
- It is always good idea that decomposing solution into modules in a hierarchal manner.
- In this approach, first we develop the main module and then the next level modules are developed.
- This procedure is continued until all the modules are developed.

1.6. Top down and Bottom up Approach

Top down Approach



1.6. Top down and Bottom up Approach

Top down Approach

Advantages:

- Breaking problems into parts help us to identify what needs to be done.
- At each step of refinement new parts will become less complex and therefore easier to solve.
- Parts of solution may turn out to be reusable.
- Breaking problems into parts allows more than one person to solve the problem.

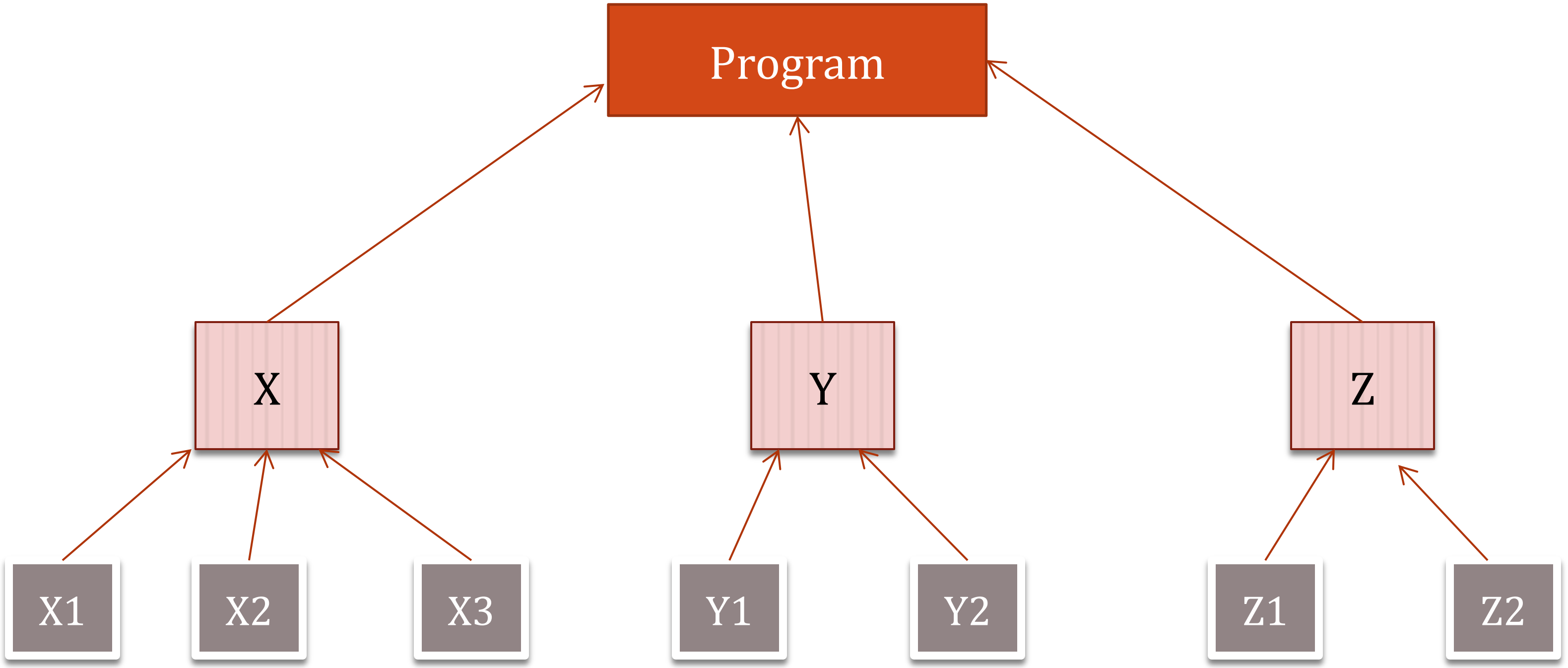
1.6. Top down and Bottom up Approach

Bottom-Up Design Model:

- In this design, individual parts of the system are specified in details.
- The parts are the linked to form larger components, which are in turn linked until a complete system is formed.
- This approach is exactly opposite to the top-down approach.
- In this approach, bottom level modules developed first (Lower level module developed, tested and debugged).
- Then the next module developed, tested and debugged.
- This process is continued until all modules have been completed.
- This approach is good for reusability of code.
- Object oriented language such as C++ or java uses bottom up approach where each object is identified first.

1.6. Top down and Bottom up Approach

Bottom-Up Design Model:



1.6. Top down and Bottom up Approach

Bottom-Up Design Model:

Advantage:

- Make decisions about reusable low level utilities then decide how there will be put together to create high level construct.
- Contrast between Top down design and bottom up design.

1.6. Top down and Bottom up Approach

S.NO.	TOP DOWN APPROACH	BOTTOM UP APPROACH
1.	In this approach We focus on breaking up the problem into smaller parts.	In bottom up approach, we solve smaller problems and integrate it as whole and complete the solution.
2.	Mainly used by structured programming language such as COBOL, Fortan, C etc.	Mainly used by object oriented programming language such as C++, C#, Python.
3.	Each part is programmed separately therefore contain redundancy.	Redundancy is minimized by using data encapsulation and data hiding.
4.	In this the communications is less among modules.	In this module must have communication.
5.	It is used in debugging, module documentation, etc.	It is basically used in testing.
6.	In top down approach, decomposition takes place.	In bottom up approach composition takes place.
7.	In this top function of system might be hard to identify.	In this sometimes we can not build a program from the piece we have started.
8.	In this implementation details may differ.	This is not natural for people to assemble.

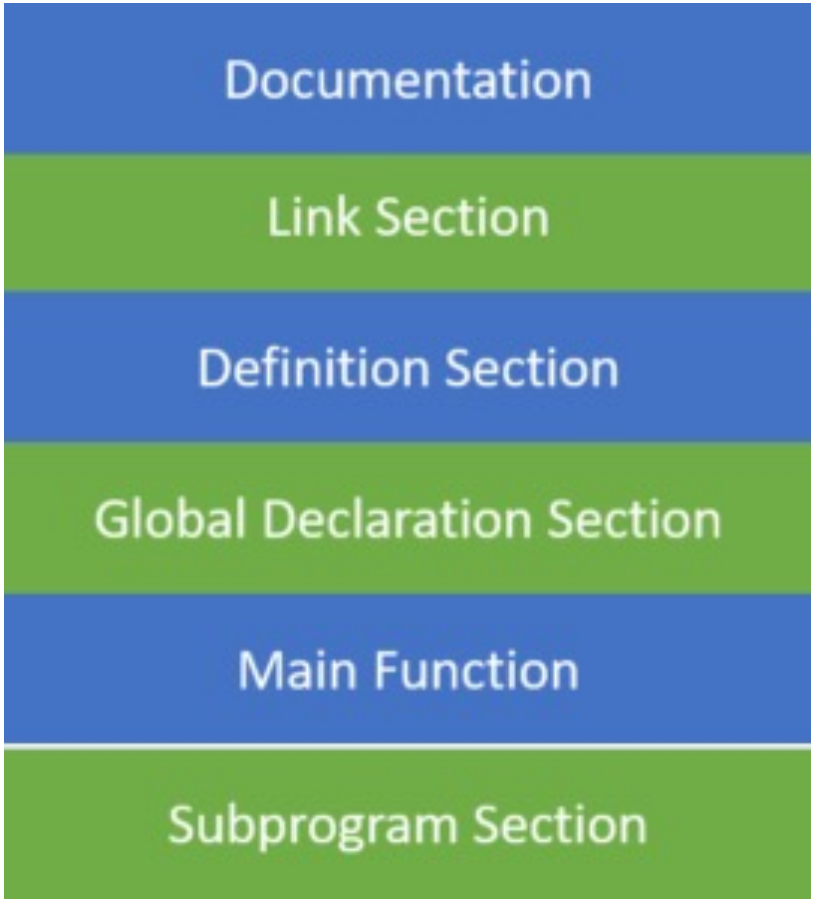
1.7.Strucuted Programming

A C program is divided into different sections. There are six main sections to a basic c program.

The six sections are,

- Documentation
- Link
- Definition
- Global Declarations
- Main functions
- Subprograms

1.7.Structucted Programming



Documentation

Link Section

Definition Section

Global Declaration Section

Main Function

Subprogram Section

Figure: Basic Structure Of C Program

1.7.Strucuted Programming

1. Documentation Section

- The documentation section is the part of the program where the programmer gives the details associated with the program.
- It usually gives the name of the program, the details of the author and other details like the time of coding and description. It gives anyone reading the code the overview of the code.

Example

```
/**
* File Name: Helloworld.c
* Author: Manthan Naik
* date: 09/08/2019
* description: a program to display hello world
*           no input needed
*/
```

Moving on to the next bit of this basic structure of a C program article,

1.7.Strucuted Programming

2. Link Section

- This part of the code is used to declare all the header files that will be used in the program.
- This leads to the compiler being told to link the header files to the system libraries.

Example

```
#include<stdio.h>
```

Moving on to the next bit of this basic structure of a C program article,

1.7.Strucuted Programming

3. Definition Section

- In this section, we define different constants. The keyword define is used in this part.

Example

```
#define PI=3.14
```

Moving on to the next bit of this basic structure of a C program article,

1.7.Strucuted Programming

4. Global Declaration Section

- This part of the code is the part where the global variables are declared.
- All the global variable used are declared in this part.
- The user-defined functions are also declared in this part of the code.

Example

```
float area(float r);
int a=7;
```

Moving on to the next bit of this basic structure of a C program article,

1.7.Strucuted Programming

5. Main Function Section

- Every C-programs needs to have the main function. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket. Both the declaration and execution part are inside the curly braces.

Example

```
int main(void)
{
    int a=10;
    printf(" %d", a);
    return 0;
}
```

Moving on to the next bit of this basic structure of a C program article,

1.7.Strucuted Programming

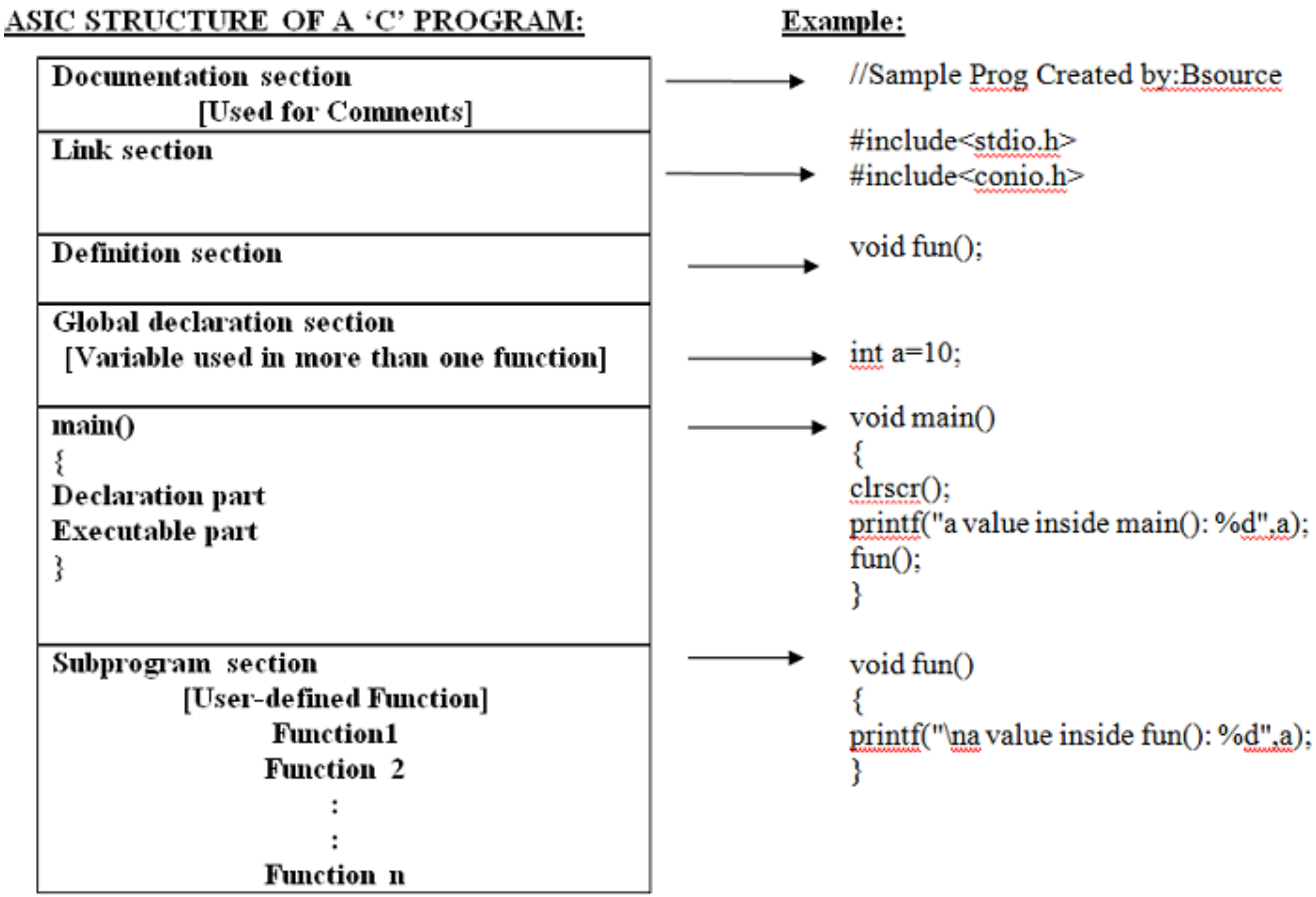
6. Sub Program Section

- All the user-defined functions are defined in this section of the program.

Example

```
int add(int a, int b)
{
    return a+b;
}
```

1.7.Strucuted Programming



ASIC STRUCTURE OF A 'C' PROGRAM:

Documentation section **[Used for Comments]**
Link section
Definition section
Global declaration section **[Variable used in more than one function]**

main()
{
Declaration part
Executable part
}

Subprogram section **[User-defined Function]** **Function1** **Function 2** **:** **:** **Function n**

Example:

```
//Sample Prog Created by:Bsource

#include<stdio.h>
#include<conio.h>

void fun();

int a=10;

void main()
{
clrscr();
printf("a value inside main(): %d",a);
fun();
}

void fun()
{
printf("\na value inside fun(): %d",a);
}
```

1.7.Strucuted Programming

Features of structured programming

- The structured program consists of well structured and separated modules.
- But the entry and exit in a Structured program is a single-time event.
- It means that the program uses single-entry and single-exit elements.
- Therefore a structured program is well maintained, neat and clean program.
- This is the reason why the Structured Programming Approach is well accepted in the programming world.

1.7.Strucuted Programming

Advantages of Structured Programming Approach:

- Easier to read and understand
- User Friendly
- Easier to Maintain
- Mainly problem based instead of being machine based
- Development is easier as it requires less effort and time
- Easier to Debug
- Machine-Independent, mostly.

1.7.Strucuted Programming

Disadvantages of Structured Programming Approach:

- Since it is Machine-Independent, So it takes time to convert into machine code.
- The converted machine code is not the same as for assembly language.
- The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
- Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

1.8. Program Design

The next stage is the program design. The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.

- Algorithm
- Flowchart

1.8. Program Design

1. **Definition of Algorithm**

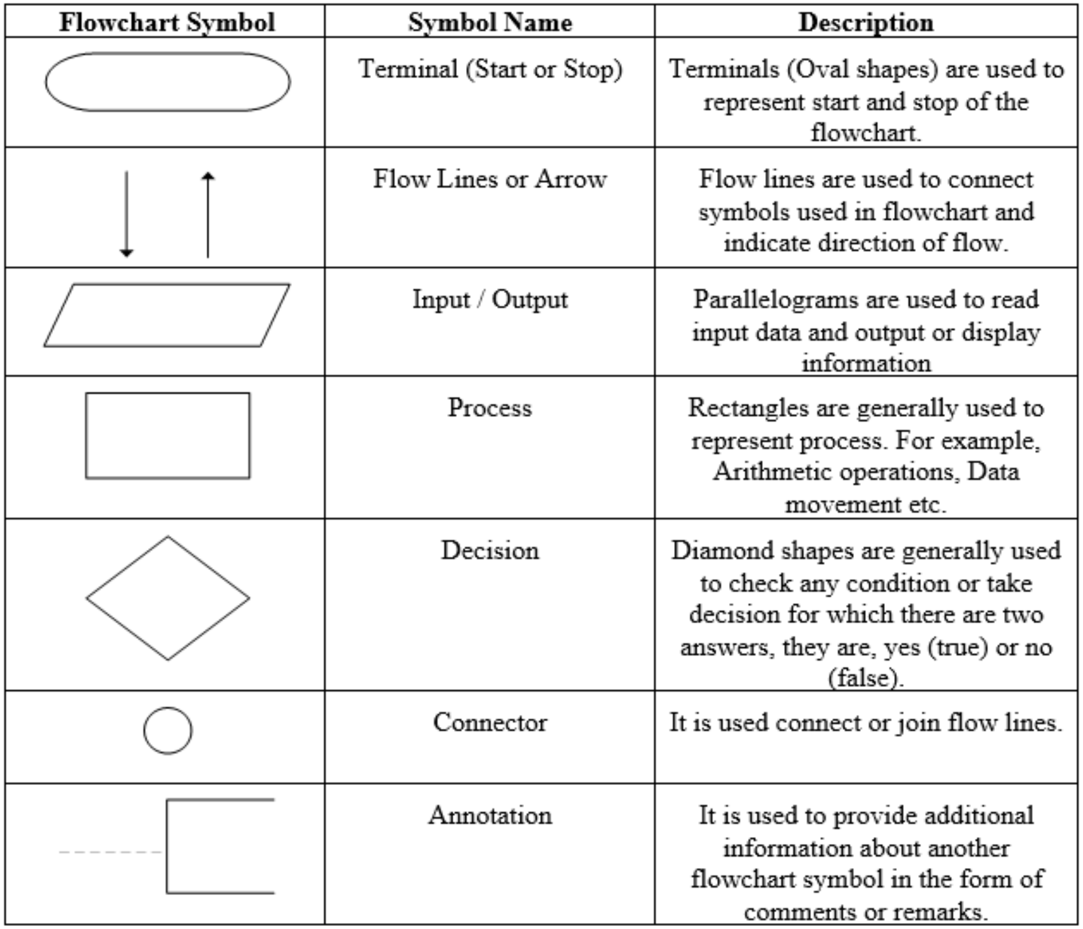
- To write a logical step-by-step method to solve the problem is called the algorithm; in other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step in the process. An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.

2. **Definition of Flowchart**

- A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of using a flowchart is to analyze different methods. Several standard symbols are applied in a flowchart:

1.8. Program Design

Definition of Flowchart



Flowchart Symbol	Symbol Name	Description
⬭	Terminal (Start or Stop)	Terminals (Oval shapes) are used to represent start and stop of the flowchart.
↓ ↑	Flow Lines or Arrow	Flow lines are used to connect symbols used in flowchart and indicate direction of flow.
▱	Input / Output	Parallelograms are used to read input data and output or display information
▭	Process	Rectangles are generally used to represent process. For example, Arithmetic operations, Data movement etc.
◇	Decision	Diamond shapes are generally used to check any condition or take decision for which there are two answers, they are, yes (true) or no (false).
○	Connector	It is used connect or join flow lines.
- - - [	Annotation	It is used to provide additional information about another flowchart symbol in the form of comments or remarks.

Start and stop

Flow or connection

Input or output

process

condition

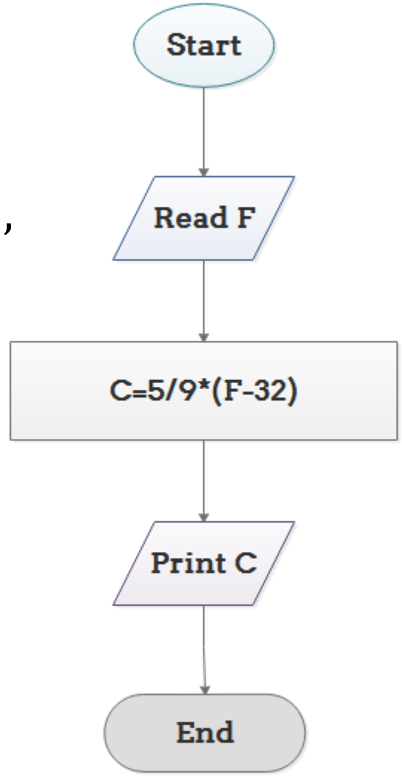
1.8. Program Design

Example 1: Convert Temperature from Fahrenheit (^{0}F) to Celsius (^{0}C)

Flowchart:

Algorithm:

- Step 1: Start
- Step 2: Read temperature in Fahrenheit,
- Step 3: Calculate temperature with formula C=5/9*(F-32),
- Step 4: Print C
- Step 5 : End



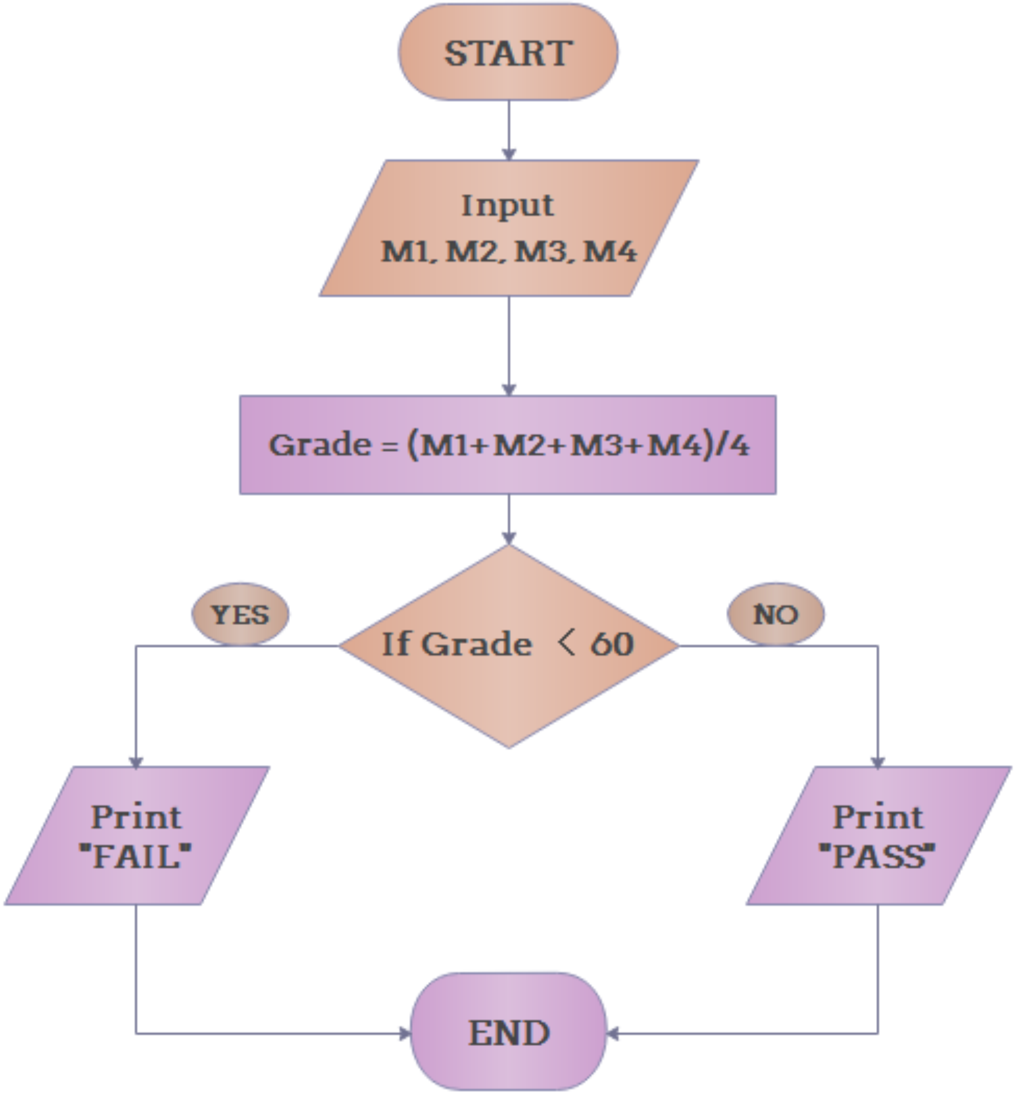
1.8. Program Design

Example 2: Determine Whether A Student Passed the Exam or Not:

Algorithm:

- Step 1: Start
- Step 2: Input grades of 4 courses M1, M2, M3 and M4,
- Step 3: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"
- Step 4: If the average grade is less than 60, print "FAIL", else print "PASS".
- Step 5: End

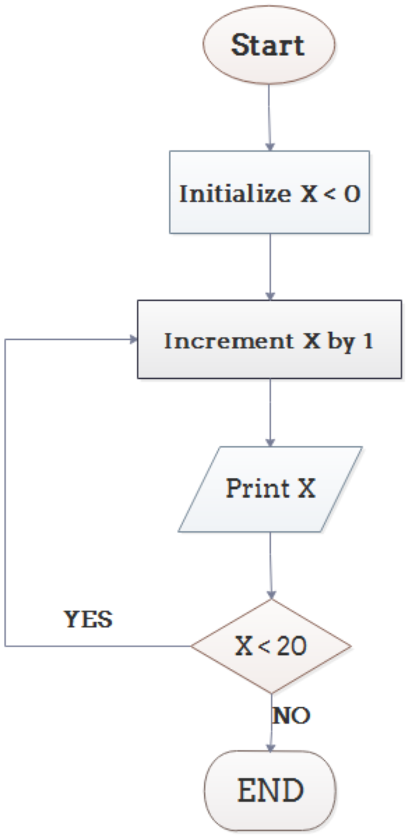
Con….
Flowchart:



1.8. Program Design

Example 3: Print 1 to 20

Flowchart:



Start

Initialize X < 0

Increment X by 1

Print X

X < 20

YES

NO

END

Algorithm:

- Step 1: Start
- Step 2: Initialize X as 0,
- Step 3: Increment X by 1,
- Step 4: Print X,
- Step 5: If X is less than 20 then go back to step 2.
- Step 6: End

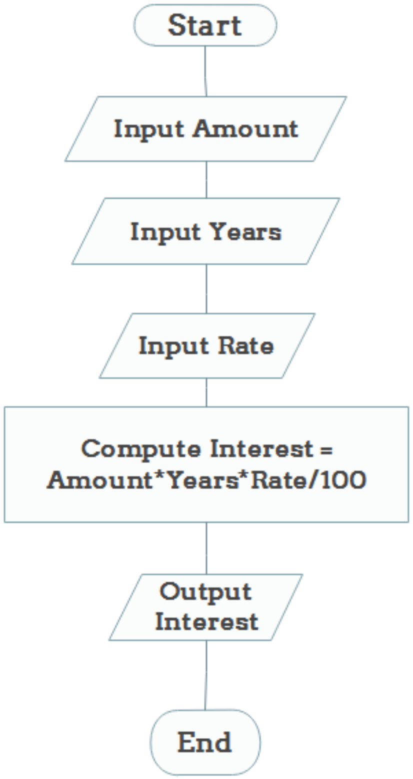
1.8. Program Design

Example 4: Calculate the Interest of a Bank Deposit

Algorithm:

- Start 1: Start
- Step 2: Read amount,
- Step 3: Read years,
- Step 4: Read rate,
- Step 5: Calculate the interest with the formula "Interest=Amount*Years*Rate/100
- Step 6: Print interest,
- Step 7: End

Flowchart:



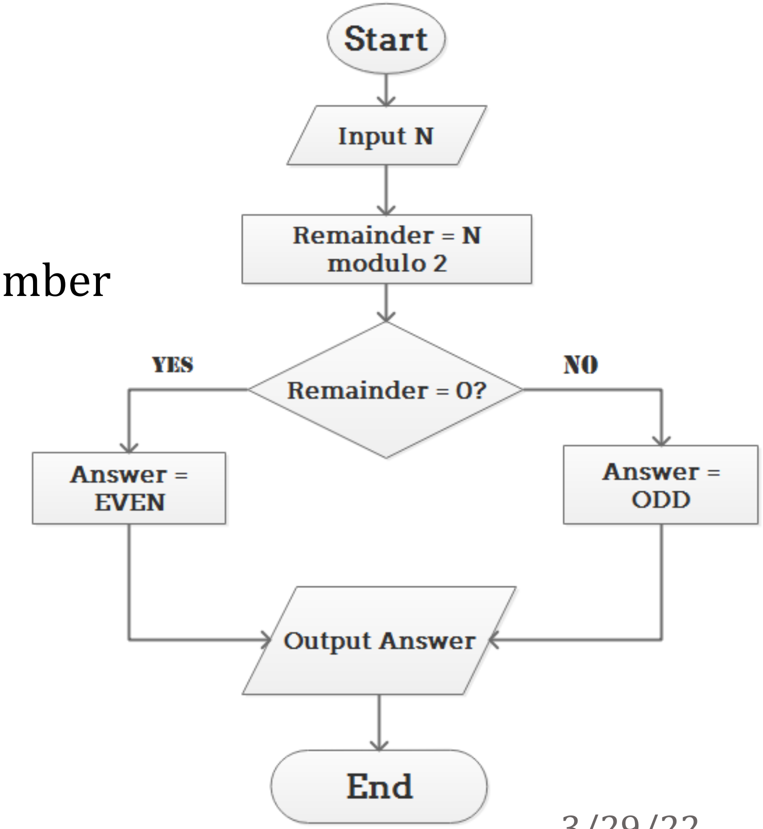
1.8. Program Design

Example 5: Determine and Output Whether Number N is Even or Odd

Algorithm:

- Step 1: Start
- Step 2: Read number N,
- Step 3: Set remainder as N modulo 2,
- Step 4: If the remainder is equal to 0 then number N is even, else number N is odd,
- Step 5: Print output.
- Step 6: End

Flowchart:



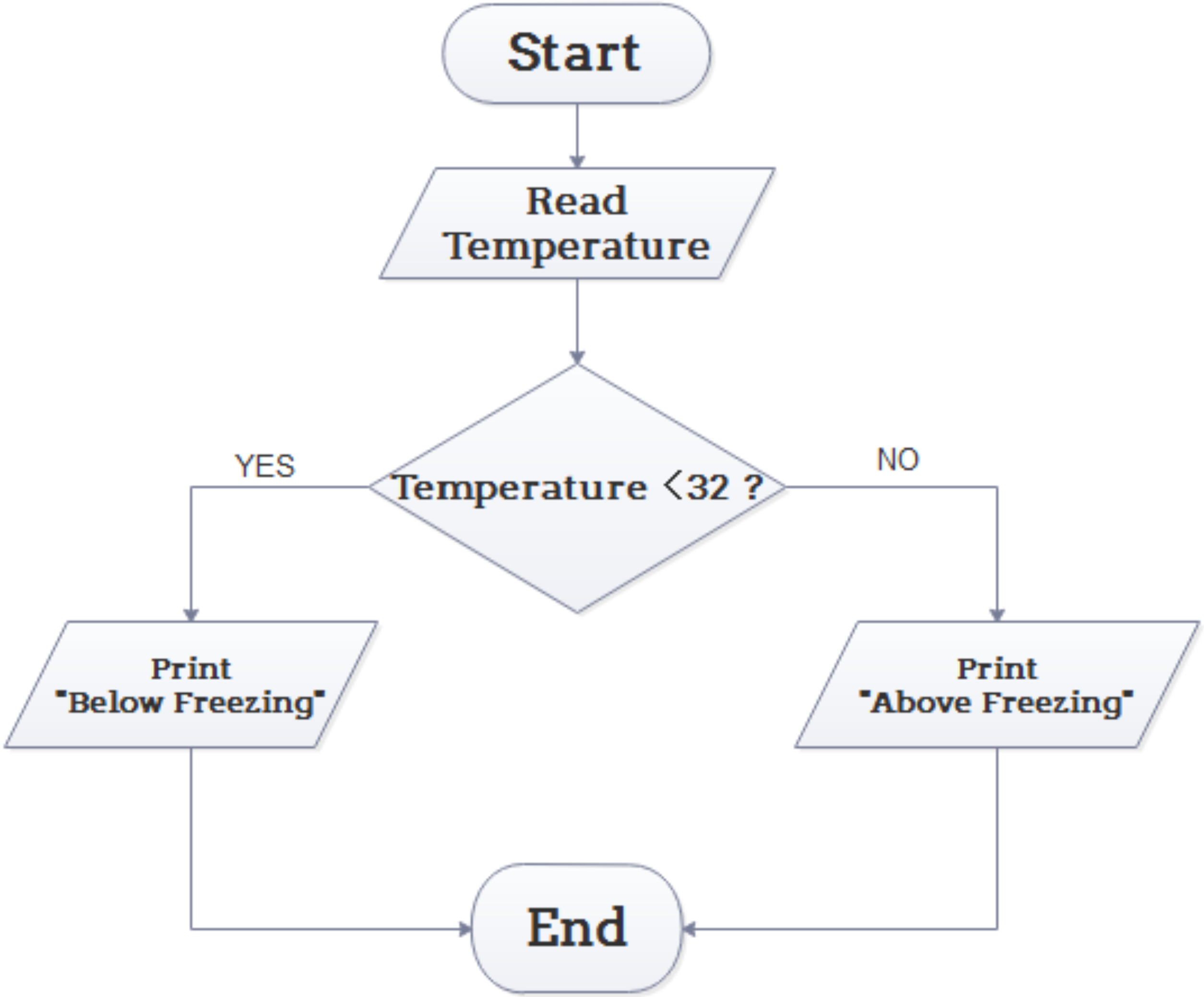
1.8. Program Design

Example 6: Determine Whether a Temperature is Below or Above the Freezing Point

Algorithm:

- Step 1:Start
- Step 2: Input temperature,
- Step 3: If it is less than 32, then print "below freezing point", otherwise print "above freezing point".
- Step 4: End

Flowchart:


Start
Read Temperature
Temperature <32 ?
YES
NO
Print "Below Freezing"
Print "Above Freezing"
End

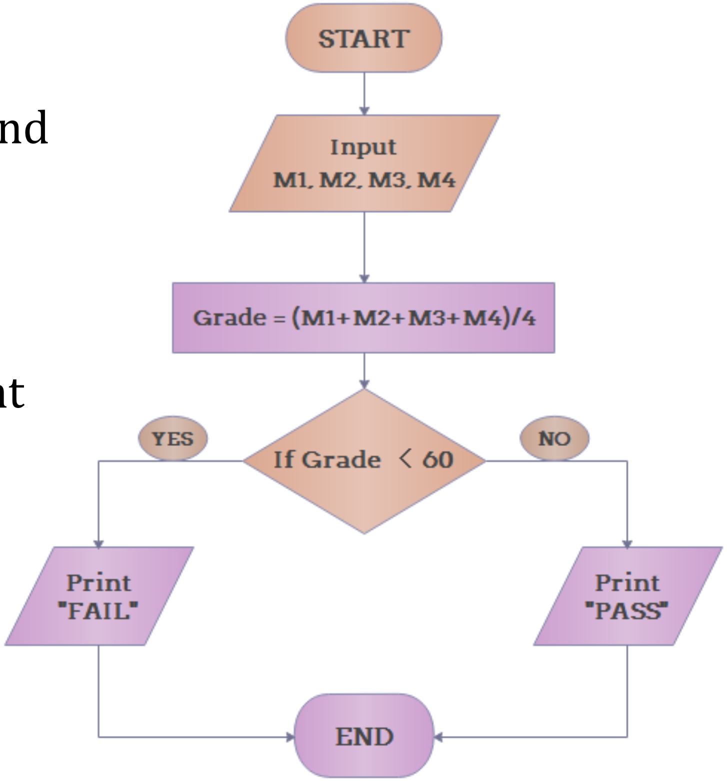
1.8. Program Design

Example 7: Determine Whether A Student Passed the Exam or Not:

Algorithm:

- Step 1:Start
- Step 2: Input grades of 4 courses M1, M2, M3 and M4,
- Step 3: Calculate the average grade with the formula "Grade=(M1+M2+M3+M4)/4"
- Step 4: If the average grade is less than 60, print "FAIL", else print "PASS".
- Step 5: End

Flowchart:



1.8. Program Design(Pseudo Code)

Pseudo Code in C

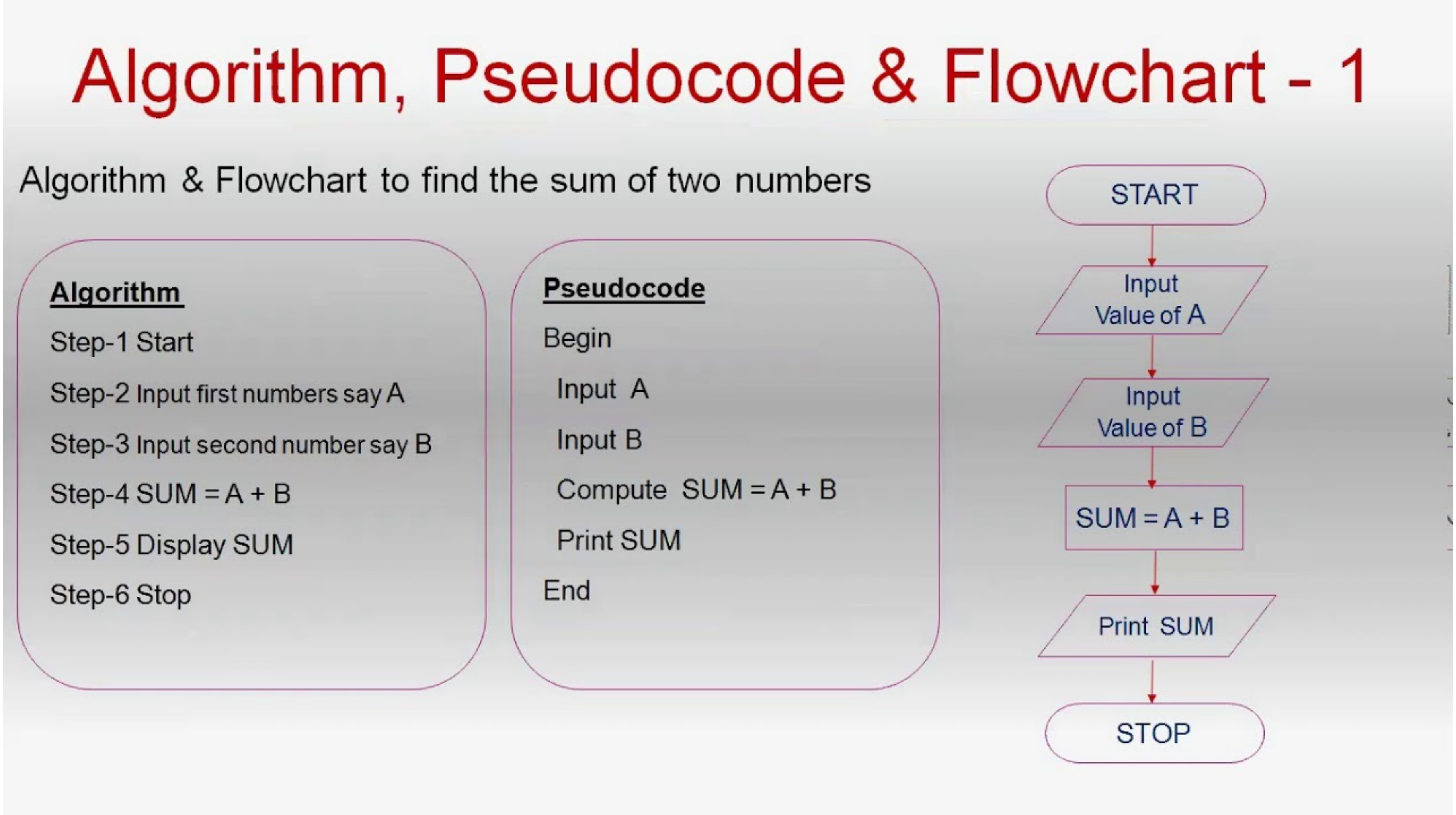
- Pseudo code in C is a simple way to write programming code in English.
- Pseudo-code is informal writing style for program algorithm independent from programming languages to show the basic concept behind the code.
- Pseudocode is not an actual programming language. So it cannot be compiled and not be converted into an executable program.
- It uses short or simple English language syntax to write code for programs before it is converted into a specific programming language.

1.8. Program Design(Pseudo Code)

Pseudocode is also known as Program Design Language (PDL) or Structured has the following characteristics:

- A free syntax of natural language that describes a processing feature.
- A subprogram definition and calling techniques.
- Fixed syntax of keywords that provide for all structured constructs, data declarations and modularity characteristics.
- A data declaration facility.

Pseudocode is a set of sequential written human language instructions, usually numbered, that is used to describe the actions a program will take when it is coded in a programming language.



Algorithm, Pseudocode & Flowchart - 1

Algorithm & Flowchart to find the sum of two numbers

Algorithm

Step-1 Start

Step-2 Input first numbers say A

Step-3 Input second number say B

Step-4 SUM = A + B

Step-5 Display SUM

Step-6 Stop

Pseudocode

```
Begin
  Input A
  Input B
  Compute SUM = A + B
  Print SUM
End
```

1.9. Debugging

- At this stage the errors in the programs are detected and corrected.
- This stage of program development is an important process. Debugging is also known as program validation.
- Some common errors which might occur in the programs include:
  - Un initialization of variables.
  - Reversing of order of operands.
  - Confusion of numbers and characters.
  - Inverting of conditions eg jumping on zero instead of on not zero.

Finished

Unit 1