F.Y.B.Sc.IT-SEM 1

PROGRAMMING PRINCIPLES WITH C

(USIT101)

BY,
NIKITA MADWAL


C

Programming Principles With C

- Reference Books:

Books and References:					
Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Programming Language	Brian W. Kernighan and Denis M. Ritchie.	PHI	2nd	1988
2.	Mastering C	K R Venugopal	Tata McGraw-Hill	6th	2007
3.	Programming with C	Byron Gottfried	Tata McGRAW-Hill	2nd	1996
4.	Let us C	Yashwant P. Kanetkar	BPB publication		
5.	Programming in ANSI C	E.Balagurusamy	Tata McGraw-Hill	7th	1982

UNIT 1

CHAPTER 1
INTRODUCTION

Introduction To Computers

- A **computer** is an electronic device that manipulates information, or data.
- It is an device that takes input in digitized format,process it according to predefined algorithms and produces desired output in the form of information or signal
- It has the ability to **store**, **retrieve**, and **process** data.

Computer System

- **Computer system** is an **interconnection** of ***hardware,software,user and data***
- Where it **allows** the *user* to **enter data** through **input devices**,**process** it *using* predefined algorithms in **CPU**,stores data into memory and **prints** desired **output** on **output devices**

Hardware

- Hardware is any part of your computer that has a **physical structure**, such as the keyboard or mouse. It also includes all of the computer's internal parts
- Example:
- Input device: Keyboard,Mouse,Scanner etc.
- Processing Unit: CPU,Motherboard,RAM etc.
- Storage Unit: RAM,Hard Disk,Pen Drives etc.
- Output device: Monitor,Printer,etc.

Software

- Software is any **set of instructions (**Set of programs**)** that tells the hardware **what to do** and **how to do it**
- It is a collection of programs that makes a computer system functional
- System Software has divided into 2 parts

I. **System Software**
II. **Application Software**

System Software

- Software acts as the interface between **the computer hardware** and the **application software**.
- It is a set of programs that control and manage the resources and operations of computer hardware.
- It also enables application programs to run correctly

Types of System Software

Some of the different types of system software include:

- **Operating system**: It enables interaction between hardware, system programs, and applications (**e.g.** Windows, Mac OS)
- **Device driver**: It controls a device and helps it to perform its functions. It facilitates device communication with the operating system and other programs (**e.g.** PC Speaker, Hard Drives, Webcam)
- **Translator**: It converts high-level human-readable languages to low-level machine codes (**e.g.** Programming Language Translator)

Application Software

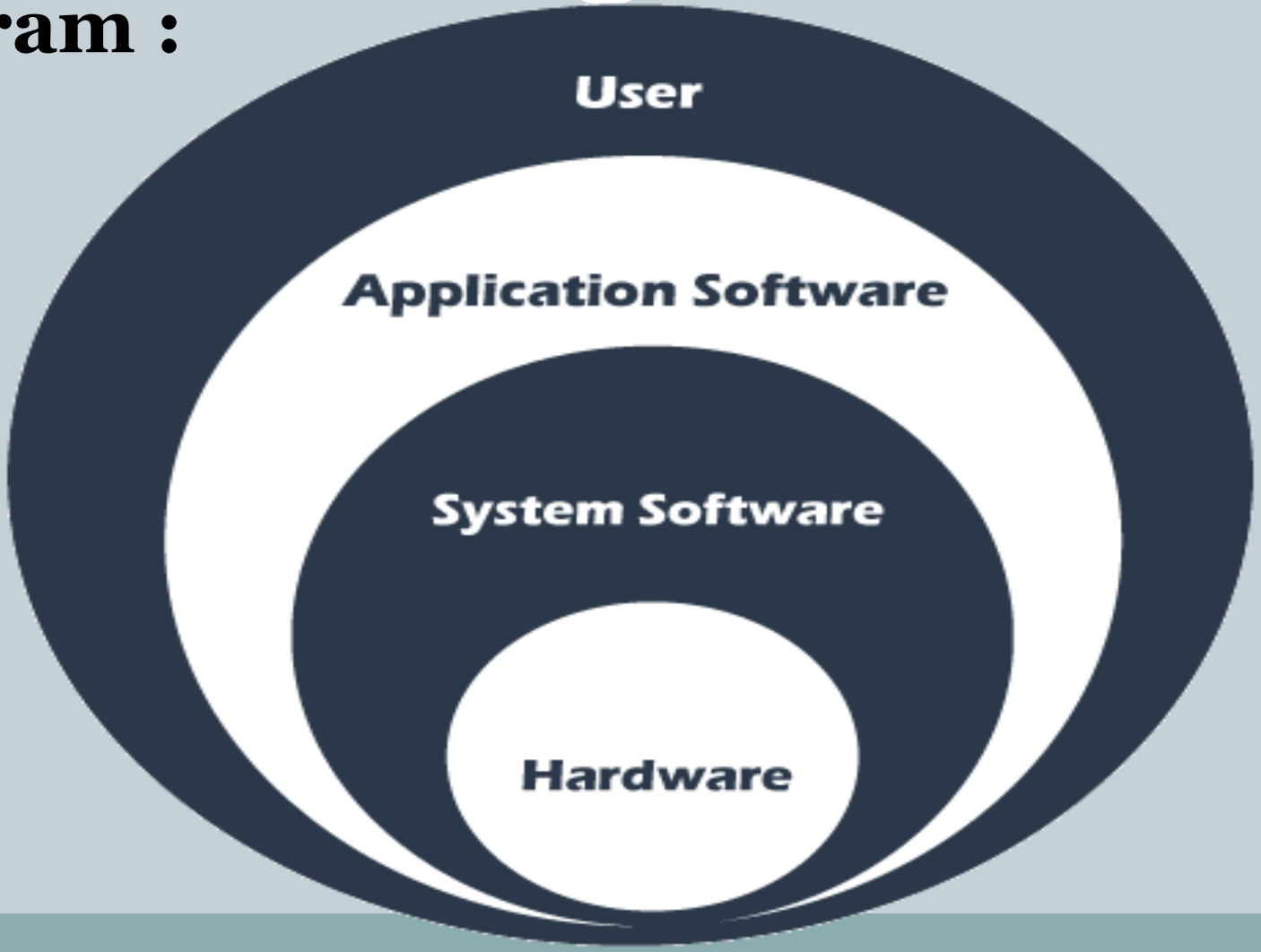
- Application software is a program that performs a specific task for the end-user.
- It runs on the platform **provided** *by* **system software**.
- It acts as a platform between the **system software** and the **end-user**.
- While the application software is not essential for running a computer, it makes the computer more useful.
- It is a specific-purpose software.

Types of Application Software

Some of the different types of application software are:

- **Word-processing software**: It enables us to create, modify, view, store, and print documents (**e.g.** Microsoft Word)
- **Spreadsheet software**: Represents data in tabular form and allows easy calculations using formulas and functions (**e.g.** Microsoft Excel)
- **Graphics software**: Facilitates easy editing of visual data (**e.g.** Microsoft Paint, Photoshop)
- **Database software**: Helps us effectively organize, manage, and access data (**e.g.** Microsoft Access)
- **Web browsers**: It enables easy surfing of the internet (**e.g.** Google Chrome, Firefox, Internet Explorer)

Diagram :



Programming Language

- **What is programming language?**
- To communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called **Programming language**.
- A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**.
- It is a set of instructions written in any specific language (C, C++, Java, Python ,etc.) to perform a specific task.
- A programming language is mainly used to ***develop*** **desktop applications, websites, and mobile applications**.

Types of Programming Language

1. Low-Level Language

- Low-level language is **machine-dependent**
- The language that directly corresponds to a specific machine
- it is represented in 0 or 1 forms, which are the machine instructions
- The processor runs low- level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.

There are two types of low-level languages

- **Machine language**
- **Assembly Language**

- **Machine language**
- Machine language is the lowest level of programming language and was the first type of programming language to be developed
- The most basic of this is machine language is a collection of very detailed, cryptic instructions that control the computer's internal circuitry.
- A language that is **directly *interpreted*** into the **hardware**
- We do the coding in 0's and 1's i.e. binary code

E.g.

Below is an example of machine language (binary) for the text "Hello World."

```
01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010
01101100 01100100
```

- **Advantages**
- Machine language makes fast and efficient use of the computer.
- It requires no translator to translate the code. It is directly understood by the computer (As computers can understand only machine instructions, which are in binary digits)
- **Disadvantages**
- Machine language is very cumbersome(difficult to handle) to work
- All operation codes have to be remembered
- All memory addresses have to be remembered.
- It is hard to find errors in a program written in the machine language.
- A machine-level language is not portable (carried out *or* movable *or* transportable) as each computer has its machine instructions, so if we write a program in one computer will no longer be valid in another computer.

Assembly Language

- Assembly language was developed to overcome some of the many inconveniences of machine language
- Assembly language in which operation codes and operands(operand is the part of a computer instruction that specifies data that is to be operating on or manipulated) are given in the form of alphanumeric symbols instead of 0's and 1's
- These alphanumeric symbols are known as **mnemonic codes** (such as MPY for multiply and ADD for addition etc)
- Because of this feature, assembly language is also known as *"Symbolic Programming Language"*
- It uses an **assembler** to convert the *assembly language* to *machine language*.

E.g.

Assembly program to add two numbers		
Label	Instruction	Comment
	ORG 0	/Origin of program is location 0
	LDA A	/Load operand from location A
	ADD B	/Add operand from location B
	STA C	/Store sum in location C
	HLT	/Halt computer
A,	DEC 83	/Decimal operand
B,	DEC -23	/Decimal operand
C,	DEC 0	/Sum stored in location C
	END	/End of symbolic program

- **Advantages**
  - Assembly language is easier to understand and use as compared to machine language.
  - It is easy to locate and correct errors. It is easily modified
- **Disadvantages**
  - Mnemonic codes should remember
  - It has to translate into machine code

2. High-Level Language

- High-level computer languages use formats that are similar to English
- The purpose of developing high-level languages was to enable people to write programs easily
- High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes
- The high-level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer

- A program that is written in a high-level language must, however, be translated into machine language before it can be executed. This is known as **compilation or interpretation**
- The original *high-level program* is called the **source program**, and the resulting *machine-language program* is called the **object program**
- High-level programming language includes **C, C++, C#, Java, JavaScript, PHP, Python,etc.**

High level language

Machine code

Easy for programmer to understand

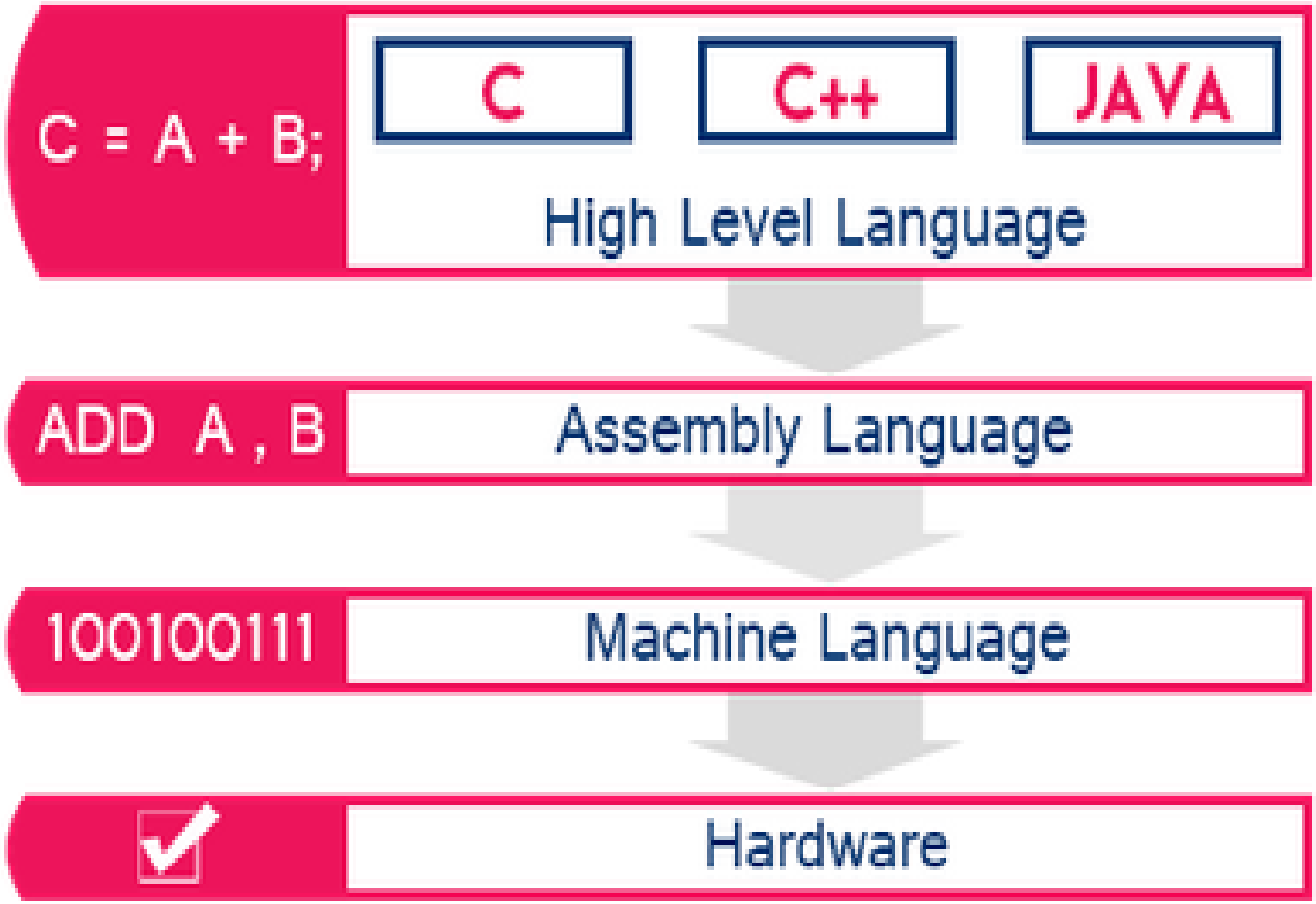
Contains English words

Translator program

The computer's own language

Binary numbers
All 1s and 0s

- **Advantages**
  - High-level languages are user-friendly
  - Similar to english language
- **Disadvantages**
  - A high-level language has to be translated into the machine language by a translator, which takes up time

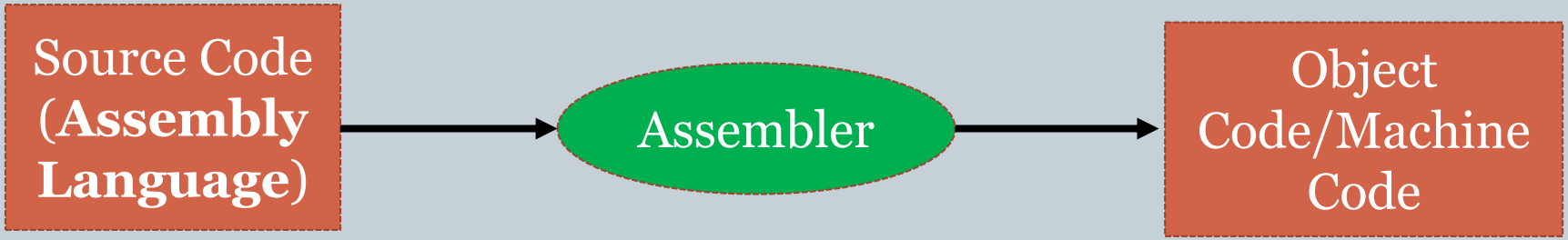

C = A + B;
C
C++
JAVA
High Level Language
ADD A , B
Assembly Language
100100111
Machine Language
Hardware

- **Computer Language Translator**
- A **translator** is a program that converts a computer program from one language to another.
- It takes a program **written** in **source code** and **converts** it ***into*** **machine code**.
- There are 3 different types of translators as follows:

1. Assembler
2. Compiler
3. Interpreter

Assembler

- An **assembler** is a translator used to translate ***assembly language*** to ***machine language***



Compiler

- A **compiler** is a translator used to convert ***high-level*** programming language to ***low-level*** programming language
- A Compiler translates the **whole program at once**
- It translate the entire program into machine language before executing any of the instructions

- **Advantages**
- Fast in execution
- The object program can be used whenever required without the need to of recompilation
- **Disadvantage**
- Debugging (removing bug or error) a program is much harder. Therefore not so good at finding errors.
- When an error is found, the whole program has to be re-compiled.

Interpreter

- An interpreter translates **line by line** , executes the instruction and then **repeats** the **procedure** for the ***remaining instructions***
- Interpreters on the other hand, proceed through a program by translating and then executing single instructions or small groups of instructions
- An instruction converted by the interpreter is executed and then the next instruction is taken up for processing.

- **Advantages**
- Good at locating errors in programs
- Debugging is easier since the interpreter stops when it encounters an error
- If an error is deducted there is no need to retranslate the whole program
- **Disadvantages**
- Rather slow
- No object code is produced, so a translation has to be done every time the program is running

- The difference between a compiler and an interpreter is that the compiler converts the entire program at once whereas the interpreter does so in parts.

Algorithms

- Program design has two phases

A. ***Problem Solving Phase*** : Creates an algorithm to solve the problem
B. ***Implementation Phase*** : Translates an algorithm in programming language (Completely coding part)

- **What is Algorithms**?

- In programming language an algorithm is a procedure or step-by-step instruction for solving a problem.
- Algorithm is an ordered sequence of clear and well defined, finite set of computational instruction that accomplishes (perform) particular task and halts in finite time.

- It is a set of instructions that must be carried out in a specific order to produce the desired result.
- An algorithm designed as language-independent, i.e. they are just plain instructions that can be implemented in any language
- They form the foundation of writing a program.
- It always a good practice to write down the algorithm before writing a program

Characteristics of an Algorithm

- **Input** - An algorithm should have 0 (zero) or more well-defined inputs
- **Output** - An algorithm should have 1 or more well-defined outputs on the basis of input
- **Finiteness** - Algorithms must terminate after a finite number of steps (i.e. in countable number of steps)
- **Definiteness** - Each step in algorithm should be defined or stated clearly
- **Effectiveness** - Each step of algorithm must be easily convertible into program statement

- **Write an Algorithm to add two numbers**

step 1: Start

step 2: Read x , y

step 3: Add z =x + y

step 4: Display sum

step 5: Stop

- **Write an algorithm to find the largest among two different numbers entered by the user.**

step 1: Start

step 2: Read x , y

step 3: if x>y then

a) Display x is greatest number

else

b) Display y is greatest number

end if

step 4: Stop

1. Write an algorithm to take 5 subjects marks from user and display the total and percentage
2. Write an algorithm to find the largest among three different numbers entered by the user.
3. Write an Algorithm for finding the average of three numbers is as follows
4. Write an Algorithm to find Area of Square
5. Write an algorithm to take radius from user and display area of circle
6. Write an algorithm to read marks of four subjects and print grade of the student according to total marks obtained

Total Marks	Grade
Above 800	A
601-800	B
401-600	C
201-400	D

- **Write an algorithm to take 5 subjects marks from user and display the total and percentage**

step 1: Start

step 2: Read sub1,sub2,sub3,sub4,sub5

step 3: total=sub1+sub2+sub3+sub4+sub5

step 4: per=total/5

step 5: Display total and per

step 6: stop

- **Write an algorithm to find the largest among three different numbers entered by the user.**

step 1: Start

step 2: Read a , b , c

step 3: if a>b and a>c then

a) Display a is largest number

else if b>a and b>c then

b) Display b is largest number

else

c) Display c is largest number

end if

step 4: Stop

- **Write an Algorithm for finding the average of three numbers is as follows**

step 1: Start

step 2: Read a , b , c

step 3: sum = a + b + c

step 4: average = sum/3

step 5: Display average

step 6: Stop

- **Write an Algorithm to find Area of Square**

step 1: Start

step 2: Read L

step 3: Area = L * L

step 4: Display Area

step 5: Stop

- **Write an algorithm to take radius from user and display area of circle**

step 1: Start

step 2: Read r

step 3: area=3.14*r*r

step 4: Display area

step 5: Stop

- **Write an algorithm to read marks of four subjects and print grade of the student according to total marks obtained**

Total Marks	Grade
Above 800	A
601-800	B
401-600	C
201-400	D

step 1: Start

step 2: Read a , b , c , d

step 3: total=a + b + c + d

step 4: if total>800 then

a) Display Grade A

Else if total>600 then

b) Display Grade B

Else if total > 400 then

c) Display Grade C

Else if total > 200 then

d)Display Grade D

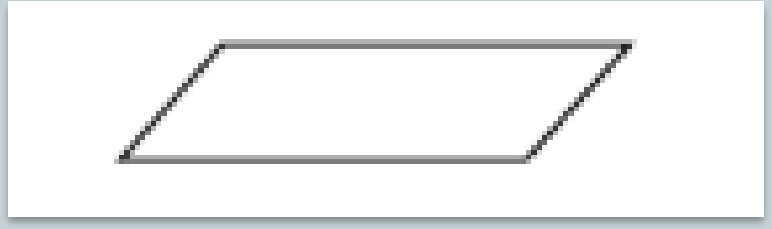
End if

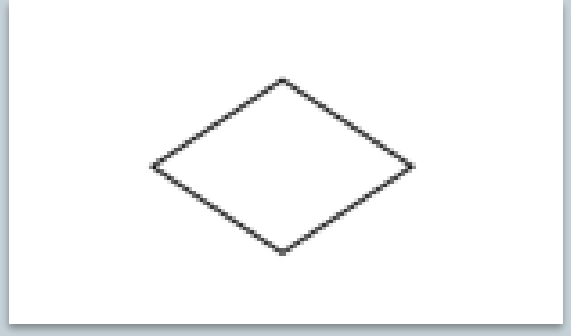
step 5: stop

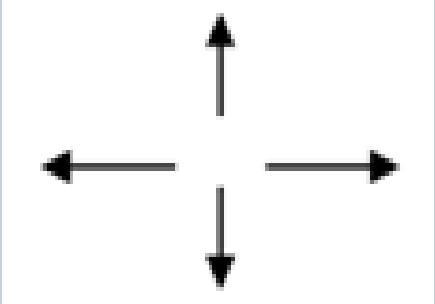
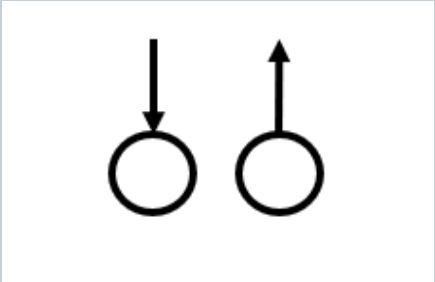
Flowchart

- It is a **graphical tool** for **representing** the defined **solution of a problem**.
- A flowchart is a **pictorial representation** of an **algorithm**.
- It shows the logic of the algorithm and the flow of control
- The flowchart uses different symbols to represent specific actions and arrows to indicate the flow of control
- Flowcharts are language independent and constitute (i.e. create) a very general way of representing an algorithm

Different Symbols Used In Flowcharts

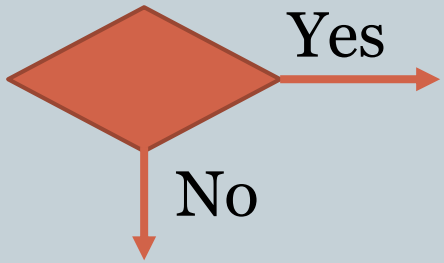
Symbol	Description
	**Terminal (Oval)**: The symbol is used to represent start and end of flowchart
	**Input / Output (Parallelogram)**: The symbol is used to for input and output operation

Symbol	Description
	➢ **Processing (Rectangle) :** The symbol is used for arithmetic operation and data-manipulation
	➢ **Decision (Diamond) :** The symbol is used to represent operation in which there are two/three alternatives, true and false etc.

Symbol	Description
	➢ **Flow Lines (Arrow):** The symbol indicates flow of operation , the exact sequence in which the instruction are executed
	➢ **Connectors :** 1. Used to represent link between parts of flowchart, if flowchart is large and unfit in single page 2. When flowcharts become very complex and spread across more than one page , then connectors are used to connect two flowcharts with each other.

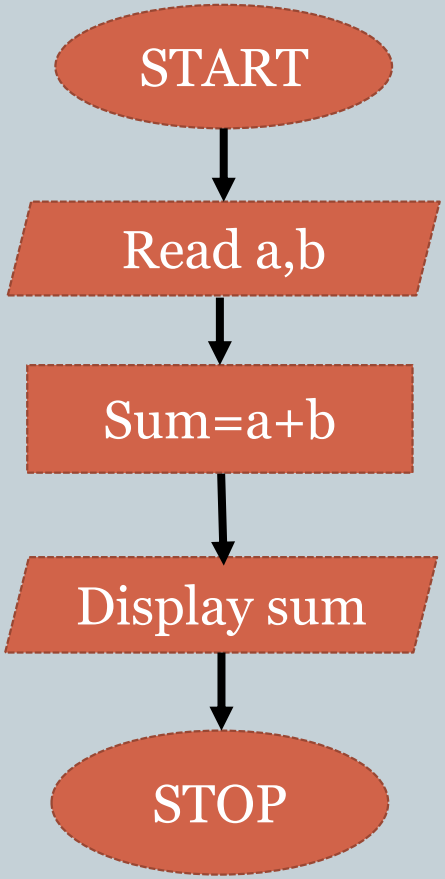
Rules For Drawing A Flowchart

- It should contain only one start and one end symbol
- All symbols in the flowchart must be connected with an arrow line.
- The branches of decision box must be labelled

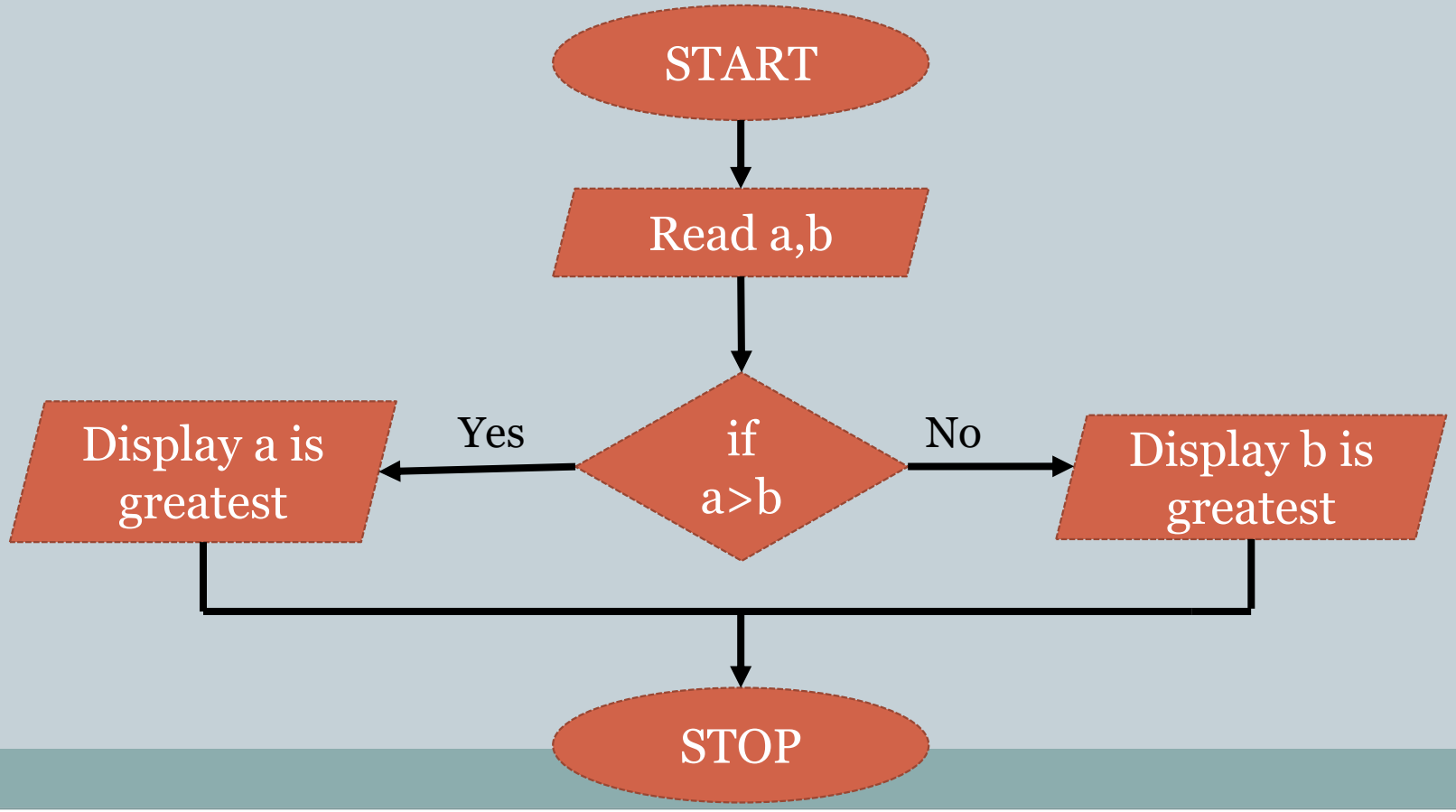


- Only standard symbols should be used

- **Create a flowchart to add two numbers entered by the user**

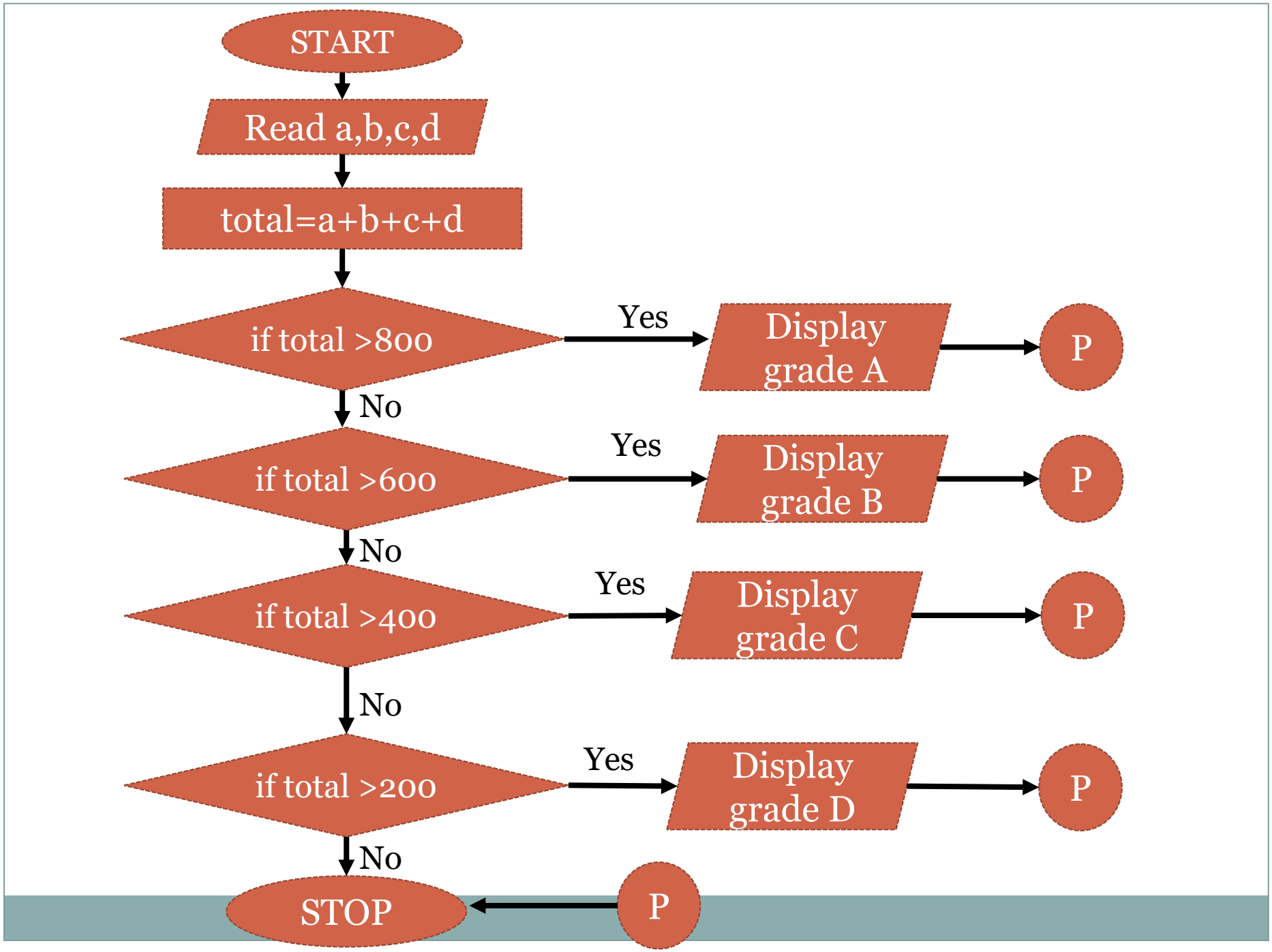


- **Create a flowchart to find largest among two different numbers entered by the user**

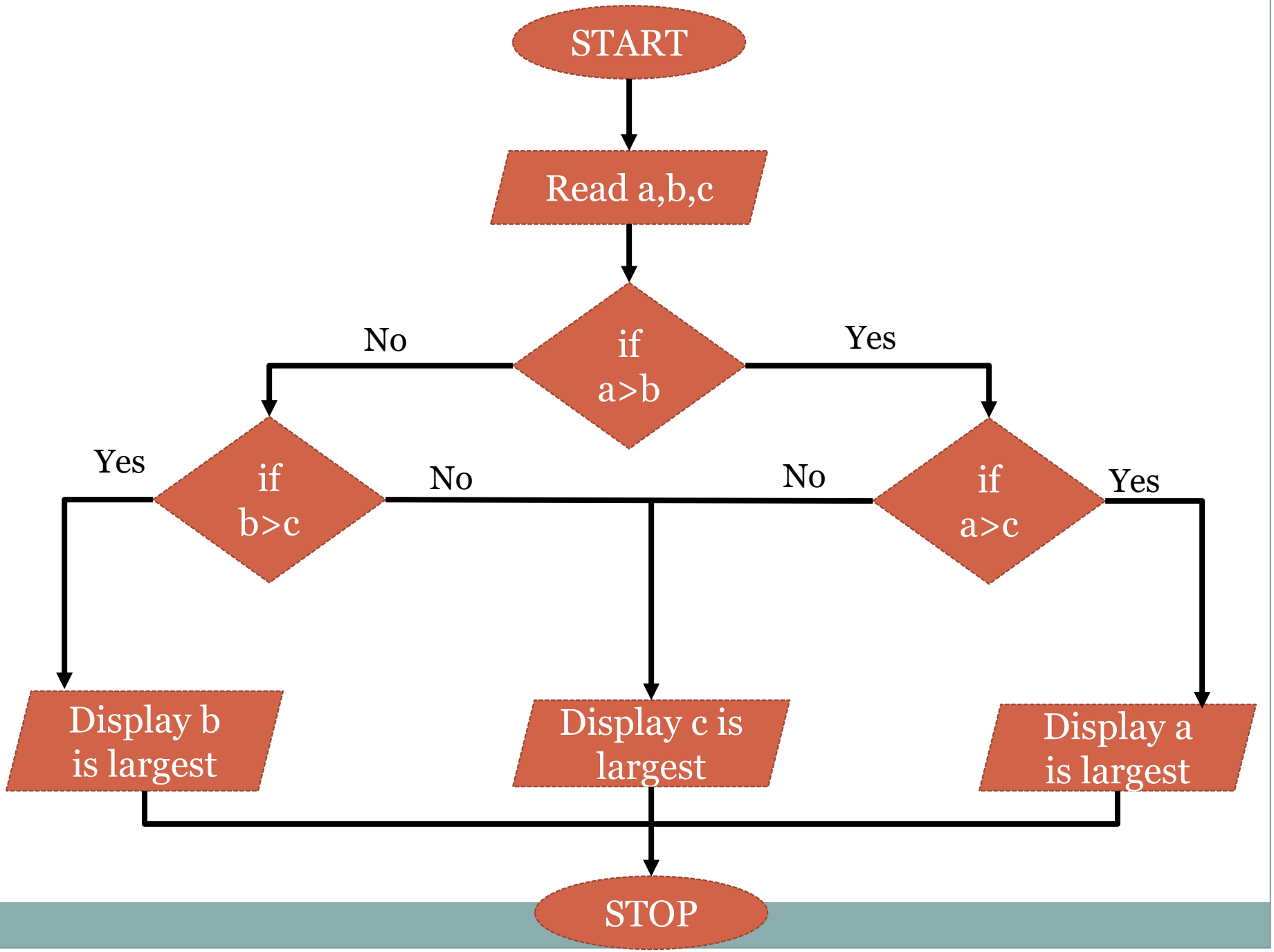


- **Create a flowchart to read marks of four subjects and print grade of the student according to total marks obtained**

Total Marks	Grade
Above 800	A
601-800	B
401-600	C
201-400	D


START
Read a,b,c,d
total=a+b+c+d
if total >800
Yes
Display grade A
P
No
if total >600
Yes
Display grade B
P
No
if total >400
Yes
Display grade C
P
No
if total >200
Yes
Display grade D
P
No
STOP
P

- **Create a flowchart to find the largest among three different numbers entered by the user.**


START
Read a,b,c
if
a>b
No
Yes
if
b>c
Yes
No
No
if
a>c
Yes
Display b
is largest
Display c is
largest
Display a
is largest
STOP

Advantages of flowchart:

- Flowchart is an excellent way of communicating the logic of a program.
- Easy and efficient to analyze problem using flowchart.
- During program development cycle, the flowchart plays the role of a blueprint, which makes program development process easier.
- The flowchart makes program or system maintenance easier.
- It is easy to convert the flowchart into any programming language code.

Disadvantage of flowchart

- Not suitable for large problems
- Modification in logic is not easy
- Time consuming

Pseudocode

- The pseudocode in C is an informal way of writing a program for better human understanding.
- It is written in simple English, making the complex program easier to understand.
- It doesn't follow the programming language's syntax; it is thus written in pseudocode so that any programmers or non-programmers can easily understand it.

- **Create a pseudocode to add 2 numbers together and then display the result**

1. Start program
2. Enter two numbers a,b
3. Add the numbers together
4. Display sum
5. End program

Introduction to C

- The C programming language is one of the most popular and widely used programming languages.
- It is a general-purpose, structured programming language or procedure oriented language.
- Its instructions consist of terms that resemble algebraic expressions, augmented(denoted) by certain English keywords such as if, else, for, do and while
- C can be written in simple English language so that it is very easy to understand and developed by programmer
- C includes extensive library functions which enhance the basic instruction.
- It is faster and efficient

History

- **C** is a computer high-level programming language developed in 1972 by **Dennis M. Ritchie** at AT & T Bell Laboratories to develop the UNIX Operating System
- C was invented to write an operating system called UNIX and 1972 **UNIX OS** almost totally written in C.
- C is also called **mother Language** of all programming Language.
- All other programming languages were derived directly or indirectly from C programming concepts.

Application of C

- C is used in writing embedded systems software's like coffee machines, microwaves, climate control systems etc.
- It has been used in various gaming applications and graphics.
- It is used in firmware for various electronics, industrial and communications products which use micro-controllers.
- C is used to implement different operating system operations
- UNIX Kernel is completely developed in C Language

Structure of C program

- Below are few commands and syntax used in C programming to write a simple C program
- **void main():**
  - This is the main function from where execution of any C program begins.
  - The main() function is the entry point of every program in c language
  - The two curly brackets { } shows the start and finish of the function.
  - Every **statement** within a **function ends** with a **semicolon (;)**

- Comments:
- There are 2 ways to add comment.

1. single line comment → //
2. Multi-line comment → /* */

- In programming, comments are hints that a programmer can add to make their code easier to read and understand.
- Comments are completely ignored by C compilers.

- **#include<stdio.h>**
- This is a header file
- This includes the standard input output library functions.
- The **printf() and scanf()** function is defined in stdio.h

i. **printf() -** The printf() function is **used to print data** on the console.The text to be printed is enclosed in double quotes(“ “)

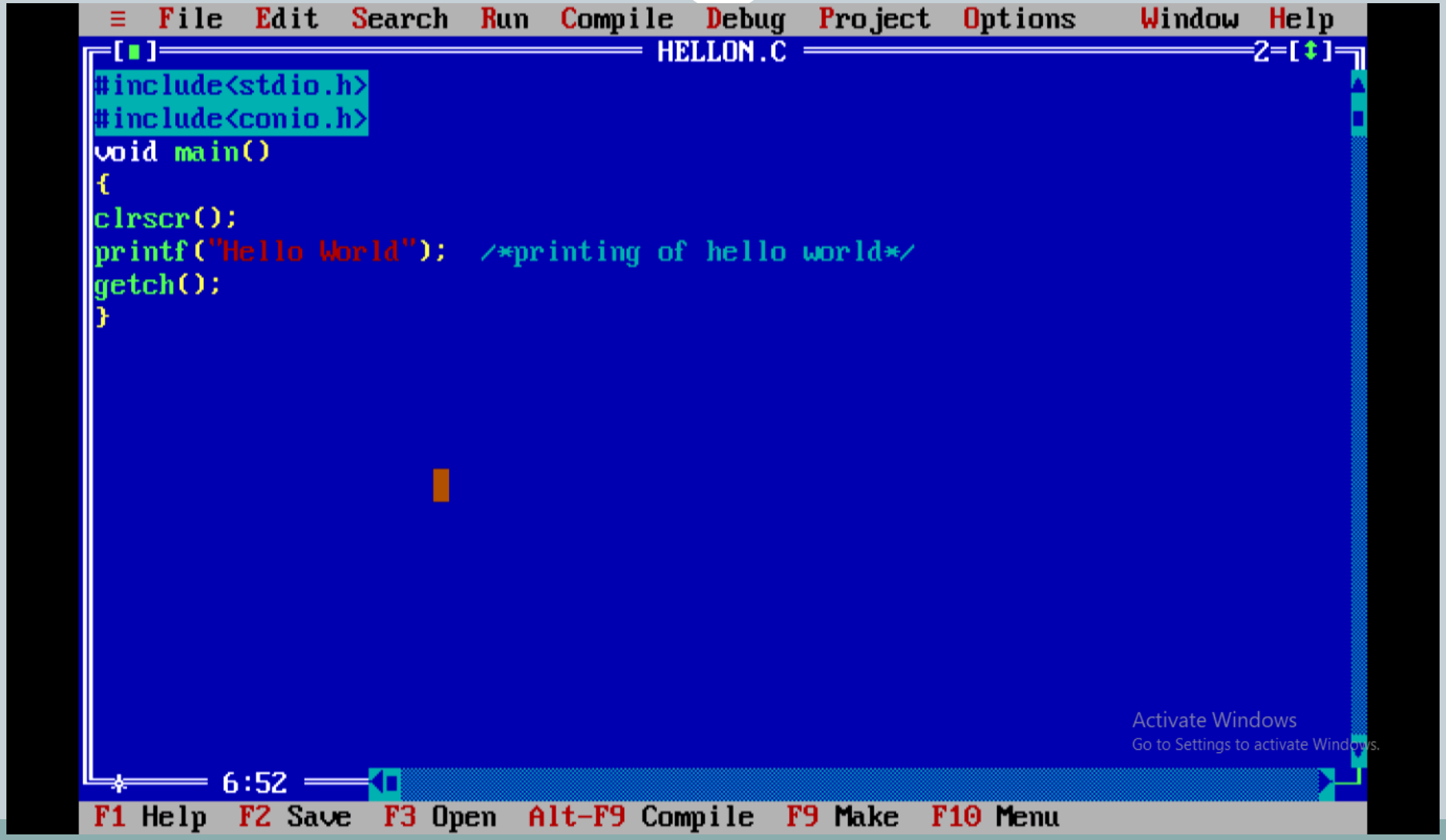
ii. **scanf() –** The scanf() is one of the commonly used function to take input from the user.

- **#include<conio.h>**
- This is a header file
- It includes the console input output library functions. The **getch() and clrscr()** function is defined in conio.h file.

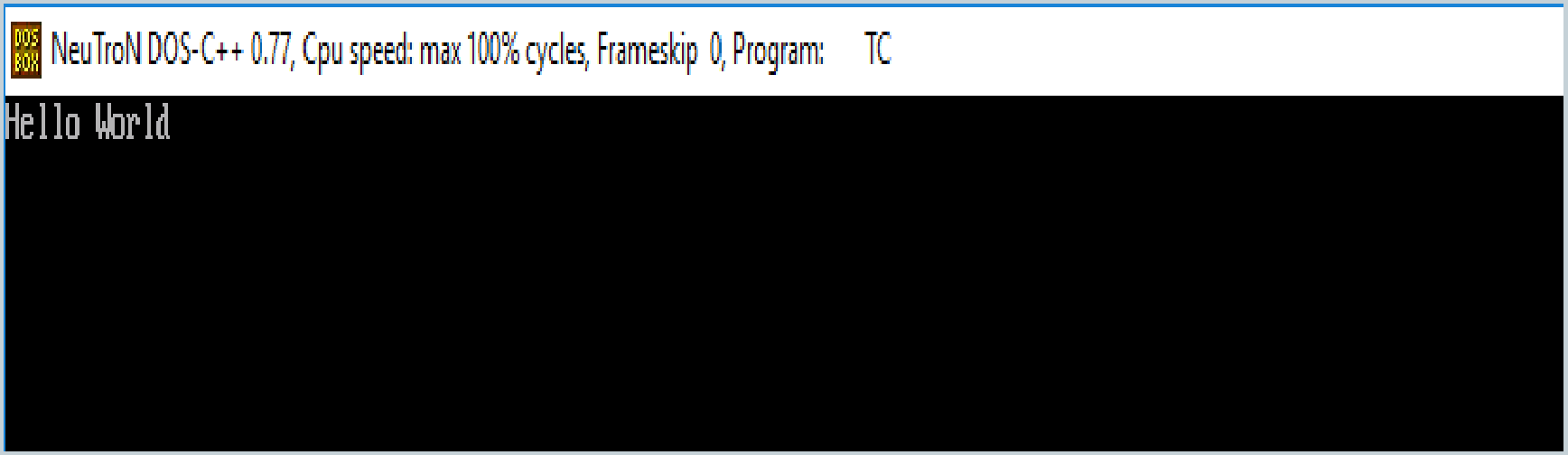
i. **clrscr() -** Clears the screen (not used in Dev C++)

ii. **getch() -** The getch() function asks for a single character. Until you press any key, it blocks the screen

Using Turbo C



- Output


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Hello World

Using Dev C++

hello.c

```c
#include<stdio.h>
#include<conio.h>
void main()
{
    printf("Hello, World!"); //printing hello world
    getch();
}
```

Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Admin\Desktop\backup 29.7.22\f\PATKAR_BSCIT\Education\F.Y.B.Sc.IT\2022_202
- Output Size: 128.1015625 KiB
- Compilation Time: 0.33s
```

Sel: 0 | Lines: 7 | Length: 123 | Insert | Done parsing in 0.015 seconds

- Output

C:\Users\Admin\Desktop\backup 29.7.22\f\PATKAR_BSCIT\Education\F.Y.B.Sc.IT\2022_2023\PPC\pr...

```
Hello, World!
--------------------------------
Process exited after 4.733 seconds with return value 13
Press any key to continue . . .
```

desirable program characteristics

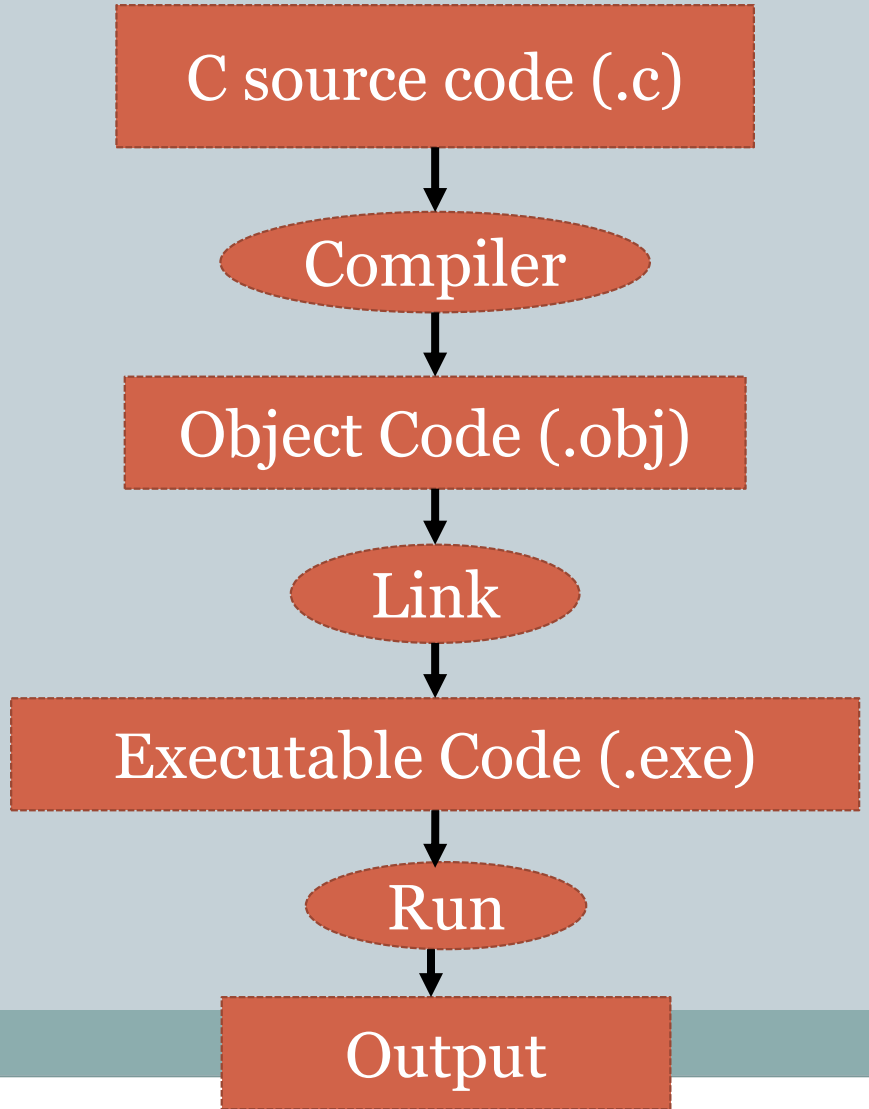
- ***Clarity***: The program should be written clearly. It should be possible for another programmer to follow the program logic without any effort.
- ***Accuracy***: The calculations should be accurate. All other properties will be meaningless if calculations are not carried out accurately.
- ***Simplicity***: Clarity and accuracy are enhanced by keeping the things simple.
- ***Efficiency***: Execution Speed and efficient memory utilization are the important ingredients of efficiency.
- ***Generality***: Placing fixed values into the programs is not good programming practice. The program should be such that values entered by the user should be used in the program.
- ***Modularity:*** Breaking down of programs into small modules(breaking down in functions) in order to increase the clarity and accuracy of the program is known as modularity.

Compilation and Execution of a Program

- In the first step ,the C program must be typed and **save it with a extension** (.c). For example, our program to find sum of two numbers can be saved with a name **(sum.c)**.It means our program source code is stored in the file sum.c
- The next step is to **compile the program** using C compiler. The **compiler *converts*** the program **source code** into equivalent ***machine code***. ***The machine code is called object code***. The file generated with .obj as extension name i.e. **(sum.obj)**

- After machine code generated, the next step is to **include the additional machine code** needed by the program from the C library. **This is called linking**. For example, if we use printf() function in C program, then that function code which is available in the library should be included into that C program.
- Once **linking completed**, the **executable file get created** with .exe extension name **(sum.exe)**
- Finally, **the executable file should be run by the C compiler to obtain the output**.

- Since the **executable file contains machine code *instruction*** which are understandable to the ***processor***, then the processor will be able to run it and display the output.


C source code (.c)
Compiler
Object Code (.obj)
Link
Executable Code (.exe)
Run
Output

- There are some basic elements is used to construct simple C statements
- These elements include the C character set, identifiers and keywords, data types, constants, variables

Character Set

- A character set denotes any alphabet, digit or special symbol used to represent information.
- The **alphabets and digits** are **together** called the ***alphanumeric characters***.
- Alphabets

  A B C D E F . . . . . X Y Z

  a b c d e f . . . . . x y z
- Digits

  0 1 2 3 4 5 6 7 8 9
- Special Character → (Next Slide)

Special Character

, comma	< opening angle bracket	> closing angle bracket
. period	_ underscore	(left parenthesis
; semicolon	$ dollar sign	) right parenthesis
: colon	% percent sign	[left bracket
# number sign	? question mark	] right bracket
' apostrophe	& ampersand	{ left brace
" quotation mark	^ caret	} right brace
! exclamation mark	* asterisk	/ slash
| vertical bar	- minus sign	\ backslash
~ tilde	+ plus sign	

identifiers and keywords

- In a C program, every word is either classified as an identifier or keyword.
- **identifiers :**
- identifiers are names that are given to various program elements, such as variables, functions,structures,arrays,etc.
- identifiers are created to give **unique** name to C entities to identify it during execution of program.

- **Rules for constructing identifiers in c**
- First character should be an alphabet or underscore(_)
- Succeeding characters might be digits or letter.
- Identifiers should not be keywords.
- C is case sensitive (i.e. upper and lower case letters are treated differently).It is a general practice to use lower or mixed case for variable and function name.
- Punctuation and special characters aren't allowed except underscore.
- Must not contains white space
- Identifiers should be unique in a program.

- **Keywords :**
- Keywords are the words whose meaning has already been explained to the C compiler.
- Tis is also called reserved words, that have standard, predefined meanings in C
- These keywords can be used only for their intended purpose
- They cannot be used as a variable name.
- There are only **32 keywords** available in C.

Keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

EXAMPLE 2.1 The following names are valid identifiers.

`x`	`y12`	`sum_1`	`_temperature`
`names`	`area`	`tax_rate`	`TABLE`

The following names are *not* valid identifiers for the reasons stated.

`4th`	The first character must be a letter.
`"x"`	Illegal characters (`"`).
`order-no`	Illegal character (–).
`error flag`	Illegal character (blank space).

- E.g. here x , y, total are the identifiers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y,total;
x=20,y=10;
total=x+y;
printf("total=%d",total);
getch();
}
```

Data Types

- Data types are used to define a variable before to use in a program
- A data type defines a set of values and the operations that can be performed on them
- It always tell which type of data/value you want to store in variable.
- Every identifier that represents a number or a character within a C program must be associated with one of the basic data types before the identifier appears in an executable statement
- C supports several different types of data, each of which may be represented differently within the computer's memory.

Data Type	*Description*	*Typical Memory Requirements*
`int`	integer quantity	2 bytes or one word (varies from one compiler to another)
`char`	single character	1 byte
`float`	floating-point number (i.e., a number containing a decimal point and/or an exponent)	1 word (4 bytes)
`double`	double-precision floating-point number (i.e., more significant figures, and an exponent which may be larger in magnitude)	2 words (8 bytes)

Type	Size (bytes)	Format Specifier
`int`	at least 2, usually 4	`%d`, `%i`
`char`	1	`%c`
`float`	4	`%f`
`double`	8	`%lf`
`short int`	2 usually	`%hd`
`unsigned int`	at least 2, usually 4	`%u`
`long int`	at least 4, usually 8	`%ld`, `%li`
`long long int`	at least 8	`%lld`, `%lli`
`unsigned long int`	at least 4	`%lu`
`unsigned long long int`	at least 8	`%llu`

`signed char`	1	`%c`
`unsigned char`	1	`%c`
`long double`	at least 10, usually 12 or 16	`%Lf`

- Declaration of data types with variable

```
int     age;
char    letter;
float   height, width;
```

- Assigning the values to variables according to data type

```
int  age=12;
char  letter= 'h';
float  height=12.5,width=8.9;
```

- E.g. here **int** is a **data type**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y,total;
x=20,y=10;
total=x+y;
printf("total=%d",total);
getch();
}
```

Constants

- There are four basic types of constants in C.

1. Integer constants
2. floating-point constants
3. character constants
4. string constants

- Integer and floating-point constants represent numbers. They are often referred to collectively as **numeric-type** constants

- Rules apply to ***all*** **numeric-type constants**.

1. Commas and blank spaces cannot be included within the constant.
2. The constant can be preceded by a minus (-) sign if desired. (Actually the minus sign is an operator that changes the sign of a positive constant, though it can be thought of as a part of the constant itself.)
3. The value of a constant cannot exceed specified minimum and maximum bounds. For each type of constant, these bounds will vary from one C compiler to another.

Integer Constants

- An integer constant is an integer-valued number
- Thus it consists of a sequence of digits
- Integer constants can be written in three different number systems: decimal (base 10), octal (base 8) and hexadecimal (base 16)

Rules for Constructing Integer Constants

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can be either positive or negative.
- If no sign precedes an integer constant it is assumed to be positive.
- No commas or blanks are allowed within an integer constant.
- The allowable range for integer constants is -32768 to 32767.

- ***Decimal integer*** : A decimal integer constant can consist of any combination of digits taken from the set 0 through 9
- If the constant contains two or more digits, the first digit must be something other than 0

EXAMPLE 2.4 Several valid decimal integer constants are shown below.

0	1	743	5280	32767	9999

The following decimal integer constants are written incorrectly for the reasons stated.

12,245	illegal character (,).
36.0	illegal character (.).
10 20 30	illegal character (blank space).
123-45-6789	illegal character (-).
0900	the first digit cannot be a zero.

- ***Octal integer***: An octal integer constant can consist of any combination of digits taken from the set 0 through 7
- The first digit must be 0,in order to identify the constant as an octal number.

EXAMPLE 2.5 Several valid octal integer constants are shown below.

0	01	0743	077777

The following octal integer constants are written incorrectly for the reasons stated.

743	Does not begin with 0.
05280	Illegal digit (8).
0777.777	Illegal character (.).

- ***Hexadecimal integer*** : A hexadecimal integer constant must begin with either 0x or 0X
- It can then be followed by any combination of digits taken from the sets 0 through 9 and a through f

EXAMPLE 2.6 Several valid hexadecimal integer constants are shown below.

```
0x        0X1        0X7FFF        0xabcd
```

The following hexadecimal integer constants are written incorrectly for the reasons stated.

`0X12.34`	Illegal character (.).
`0BE38`	Does not begin with 0x or 0X.
`0x.4bff`	Illegal character (.).
`0XDEFG`	Illegal character (G).

Floating point constant

- A floating-point constant is a number that contains either a decimal point or an exponent (or both)

Rules for constructing floating point constant

- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- Default sign is positive.
- No commas or blanks are allowed within a real constant.

EXAMPLE 2.8 Several valid floating-point constants are shown below.

0.	1.	0.2	827.602
50000.	0.000743	12.3	315.0066
2E-8	0.006e-3	1.6667E+8	.12121212e12

The following are *not* valid floating-point constants for the reasons stated.

1	Either a decimal point or an exponent must be present.
1,000.0	Illegal character (,).
2E+10.2	The exponent must be an integer quantity (it cannot contain a decimal point).
3E 10	Illegal character (blank space) in the exponent.

Character Constants

- A character constant is a single character, enclosed in apostrophes (i.e., single quotation marks)
- The maximum length of a character constant can be 1 character.

EXAMPLE 2.10 Several character constants are shown below.

'A' 'x' '3' '?' ' '

Character Escape Sequence

- There are some characters which have special meaning in C language.
- They should be preceded by **backslash symbol** to make use of special function of them

Backslash character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\”	Double quote
\’	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark

String Constants

- A string constant consists of any number of consecutive characters (including none), enclosed in (double) quotation marks

EXAMPLE 2.14 Several string constants are shown below.

```
"green"        "Washington, D.C. 20005"        "270-32-3456"
"$19.95"       "THE CORRECT ANSWER IS:"        "2*(I+3)/J"
"      "       "Line 1\nLine 2\nLine 3"        ""
```

Variables

- A **variable is an identifier** that is used to represent some specified type of information within a designated portion of the program
- In its simplest form, a variable is an identifier that is used **to represent a single data item**; i.e., a numerical quantity or a character constant
- Variable might be belonging to any of the data type like int, float, char,etc.

- ***Rules for naming C variable:***
- Variable name must begin with letter or underscore.
- Variables are case sensitive
- No special symbols are allowed other than underscore.
- Variable name should not be a keyword
- sum, height, _value, math_book are some examples for variable name

- ***Declaring & initializing C variable***:
- Variables should be declared in the C program before to use.
- Variable initialization means assigning a value to the variable.
- It tells the compiler what the variable is
- It Specifies what type of data the variable hold

S. No	Type	Syntax	Example
1	Variable declaration	data_type variable_name;	int x, y, z; char flat, ch;
2	Variable initialization	data_type variable_name =value;	int x = 50, y = 30; char flag = ‘x’, ch=’l’;

- E.g. here **x,y,total** is a **variable**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y,total;
x=20,y=10;
total=x+y;
printf("total=%d",total);
getch();
}
```

typedef

- It is known as "**type definition**" that allows users to define an identifier that would represent an existing data type
- The typedef statement is **used** to give **new names** to ***existing types***
- **Syntax**

typedef type identifier;

Here, **type** refers to an **existing data type** and ***identifier*** refers to ***new name*** given to the data type

- **Example**

typedef int unit;

typedef float mark;

Here, **unit** **<u>symbolizes</u>** **<u>int</u>** and ***mark*** ***<u>symbolizes</u>*** ***<u>float</u>***.

They can be later used to declare variables as follows:

unit b1,b2;

mark m1,m2;

b1,b2 are **<u>declared</u>** as **<u>int</u>** variable and ***m1,m2*** are ***<u>declared</u>*** as ***<u>floating point</u>*** variables.

typedef.c

```c
#include<stdio.h>
void main()
{
typedef int unit;
unit b1,b2;
b1=10;
b2=20;
printf("---typedef---");
printf("\nValue of b1 is :%d",b1);
printf("\nValue of b2 is :%d",b2);
getch();
}
```

Output :

C:\Users\Admin\Desktop\Cpractical\t...

```
---typedef---
Value of b1 is :10
Value of b2 is :20
```

- The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program
- **NOTE** : typedef cannot create a new type

typecasting

- Converting one datatype into another is known as **type casting** or **type-conversion**
- There are two types of type conversion:

1. **Implicit Type Conversion**
2. **Explicit Type Conversion**

- Type casting helps a programmer convert any given data type to any other data type in a program.

- **Implicit Type Conversion**
- It is also known as ‘automatic type conversion’.
- Done by the compiler on its own, without any external trigger from the user (without any intervention of user).

implicittypecast.c

```c
#include<stdio.h>
void main()
{
    int a;
    float b;
    a=7.8;
    b=5;
    printf("----Implicit Conversion----");
    printf("\n value of a is %d",a);
    printf("\n Value of b is %f",b);
    getch();
}
```

Output :

C:\Users\Admin\Desktop\Cpractical\i...

```
----Implicit Conversion----
 value of a is 7
 Value of b is 5.000000
```

- **Explicit Type Conversion**
- This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.
- **Syntax**

(type) expression

explicittypecast.c

```c
#include<stdio.h>
void main()
{
    double sum,x = 1.2;
    sum = (int)x + 1;  // Explicit conversion from double to int
    printf("----Explicit Conversion----\n");
    printf("sum = %lf",sum);
    getch();
}
```

Output :

C:\Users\Admin\Desktop\Cpr...

```
----Explicit Conversion----
sum = 2.000000
```