09/08/2020

1. Overview Of C

5m Importance of C : X.

*. The increasing popularity of C is probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used to write any complex program. The C compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages. In fact, many of the C compilers available in the market are written in C.

*. Program written in C are efficient and fast. This is due to its variety of data types and powerful operators. It is many times faster than BASIC. For example, a program to increment a variable from 0 to 15000 takes about one second in C while it takes more than 50 seconds in an interappert interpreter BASIC.

*. There are only 32 keywords and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs.

*. C is highly portable. This means that C Programs writes for one computer can be run on another with little or no modification. Portability is important if we plan to use a new computer with a different operating system.

2. Format of Simple C program: (2m)

```
main ()          ──────► Function name
{                ──────► Start of program
  .....
  .....          ──────► Program statements
  .....
}                ──────► End of program.
```

Sample program 1 : Printing a message.

```
main()
{
/*.....Printing begins.....*/
    printf ("I see, I remember")
/*.....Printing ends.....*/
}
```

Introduction to C:

*. C is a general-purpose high level language that was originally developed by. Dennis Ritchie for the UNIX operating system. at Bell laboratories.

*. It was first implemented on the Digital Equippment corporation PDP-11 computer in 1972.

*. The UNIX operating system and virtually all unix applications are written in the C language

*. C has now become a widely used professional language for various reasons.

* Easy to learn.
* structured language.
* It produces efficient programs.
* It can handle low-level activities.
* It can be compiled on a variety of computers.

Some Facts:

* C is a successor of B language which was introduced around 1970s.
* Language was formalized in 1988 by the American National Standard Institute (ANSI).
* By 1973 UNIX OS almost totally written in C.
* Today C is the most widely used System Programming Language.
* Most of the State of the art Software have been implemented using C.

Influenced by

- Algol 60 (1960)
- CPL (Cambridge, 1963)
- BCPL (Martin Richard, 1967)
- B (Ken Thompson, 1970).

Structure of C program : (10m)

*. C program can be viewed as a group of building blocks called functions.

*. A function is a subroutine that may include one or more statements designed to perform a specific task.

*. A C program may contains one or more sections.

- Documentation section
- Link section
- Definition section.
- Global Declaration section
- main () Function section

```
{
    Declaration part
    Executable part
}
```

- Subprogram section

```
Function 1
Function 2      } (user-defined functions).
   :
Function n
```

1) Documentation Section:

*. The documentation section is the part of the Program where the programmer gives the details associated with the program.

*. He usually gives the name of the program, author, 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.

ex:

```
/**
    File name : Helloworld.c
    Author : Manthan Naik
    Date : 09/08/2019
    Description : a program to display hello world
*/
```

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.

ex:

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

3) Definition Section:

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

ex:

```
#define PI = 3.14
```

Macros are symbolic constants

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.

ex:

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

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. { }.

6.) Sub Program Section:

*. The C program here will the subprogram section contains all the user defined functions that are used to perform a specific task.

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

Programming Style:

*. Unlike some other programming languages (COBOL., FORTRAN., etc....).

*. C is a free-from language.

*. Although several alternative styles are possible, we should select one style and use it with total consistency.

*. C Program statements are written in lowercase letters.

*. Uppercase letters are used only for

symbolic constants.

*. Braces group program statements together and mark the beginning and the end of functions.

*. A proper identation of braces and statements would make a program easier to read and debug.

```
a = b;
x = y+1;
z = a+x;
```

can be written in one line as,

```
a=b; x=y+1; z=a+x;
```

*. The generous use of comments inside a Program cannot be overemphasized.

*. Judiciously inserted comments not only increase the readability but also help to understand the program logic.

*. This is very important for debugging and testing the program.

Executing A 'C' Program:

*. Executing a program written in C involves a series of steps;

1.) Creating the program.

2.) Compiling the program.

3.) Linking the program with functions that are needed from the Library.

4.) Executing the program.

1.) creating the program:

* Once we load the UNIX operating system into the memory, the computer is ready to receive the program.

* The file name can consists of letters, digits and special characters, followed by a dot and a letter c.

Ex: hello.c ; program.c ; ebg1.c.

* The file is created with the help of a text editor, either ed or vi.

* The command for calling the editor and creating the file is.

ed filename.

* Any corrections in the program are done under the editor.

* The program that is entered into the file is known as source program.

2.) Compiling and Linking:

Compiling.

* Let us assume that the program has been created in a file named ebg1.c.

* Now the program is ready for compilation. The compilation command to achieve this task under UNIX is,

cc ebg1.c.

* The source program instructions are now

translated into a form that is suitable for execution by the computer.

*. If the everything is alright, the compilation proceeds silently and the translated program is stored on another file with the name ebg 1.0.

*. This program is known as Object code.

Linking.

*. Linking is the process of putting together other program files and functions that are required by the program

Example:

*. If the program is using exp() function, then the object code of this function should be brought from the **math library** of the system and linked to the main program.

*. Under UNIX, the linking is automatically done, when the cc command is used.

*. If mistakes in the **syntax** and **semantics** of the language are discovered, they are listed and compilation ends there.

*. The errors should be corrected by editor and compilation done again.

*. The compiled and linked program is called the **executable object code** and stored in another file named **a.out**.

`cc filename -lm`

is the command under UNIPLUS SYSTEM V operating system.

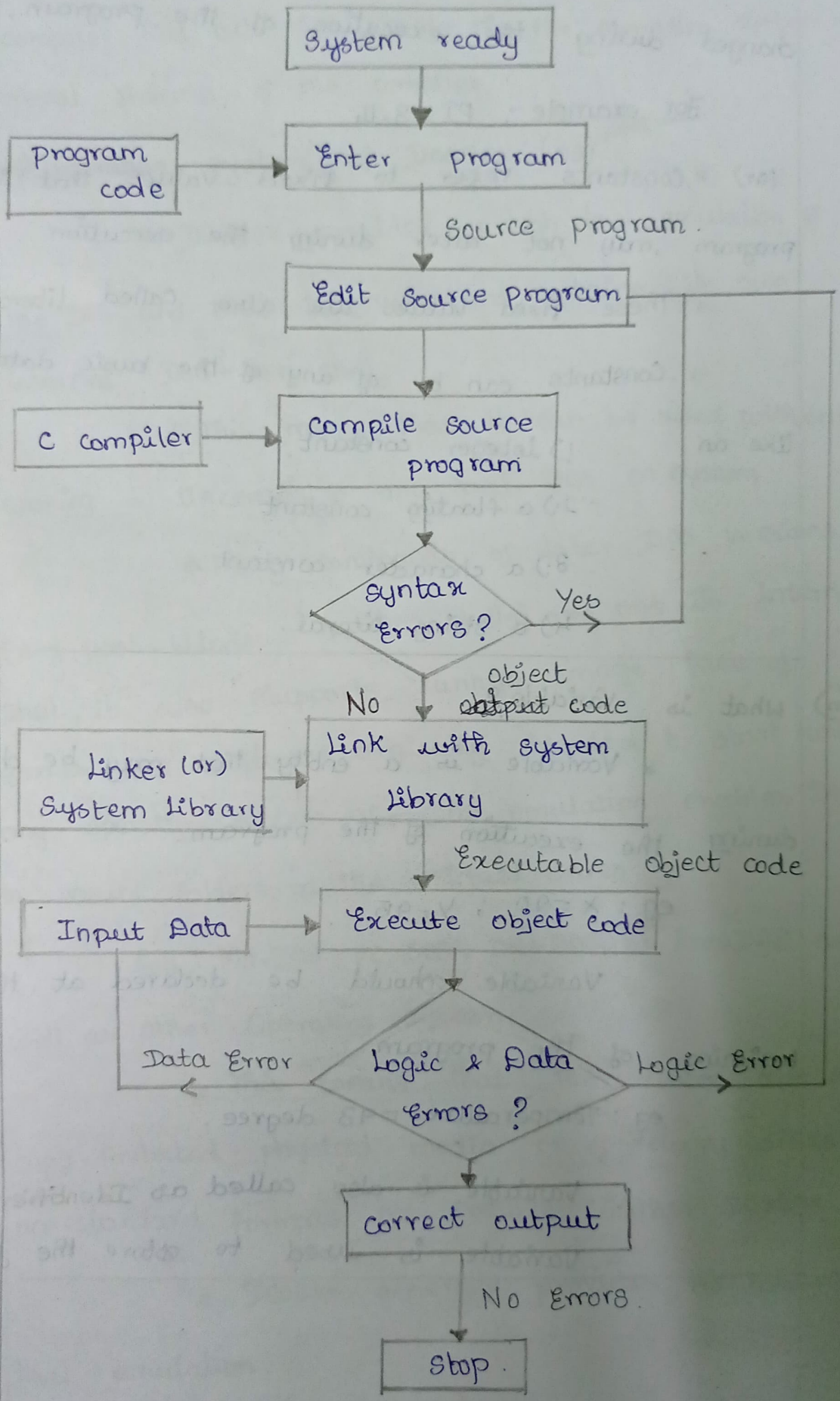
4) Executing the program:

* Execution is a simple task. The command
* a.out would load the executable object code into the computer memory and execute the instructions.
* If something is wrong with the program logic or data.
* Then it would be necessary to correct the source program or the data.
* In case the source program is modified, the entire process of compiling, linking and executing the program should be repeated.

(2m) What is a computer program?

* A computer program is a collection of instructions that can be executed by a computer to perform a specific task.
* A program is a set of commands.
* A program is a step by step instructions called as codes.
* By study a C program we can create a own program.
* Program is used to work a task.
* We have to write a computer program in a proper syntax.

10m Process of compiling and executing a C Program.



(2m) What is constant?

*. A constant is a value that cannot be changed during the execution of the program.

For example ; PI = 3.14.

(or) *. Constants refer to fixed values that the program may not alter during the execution.

*. These fixed values are also called literals.

*. Constants can be of any of the basic data types like an

1.) integer constant.

2.) a floating constant.

3.) a character constant.

4.) a string literal.

(2m) What is Variable?

*. Variable is a entity that may be changed during the execution of the program.

eg: X = 90 ; Y = 95.

*. Variable should be declared at the beginning of the program.

eg: Temperature = 93 degree.

*. Variable is also called as Identifier.

*. Variable is used to store the data.

Constants, Variables and Data types.

Introduction:

* A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as **information**.

* The task of processing of data is accomplished by executing a sequence of precise instructions called a **Program**.

* These instructions are formed using certain symbols and words according to some rigid rules known as **syntax rules (or grammar)**.

* C has its own vocabulary and grammar.

* Every program instruction must confirm precisely to the syntax rules of the prog language.

Character Set:

* The characters that can be used to form words, numbers and expressions depend upon the computer on which the program is run.

* A subset of characters is available that can be used on most personal, micro, mini and mainframe computers.

The characters in C are:

1.) Letters.
2.) Digits.
3.) special characters
4.) White spaces.

* The compiler ignores white spaces unless they are a part of a string constant, white spaces may be used to separate words, but are prohibited between the characters of keywords and identifiers.

Letters:

Uppercase ⇒ A .... Z ; Lowercase ⇒ a .... z.

Digits:

All decimal digits 0 .... 9.

Special characters:

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

] → right bracket	{ → left brace	} → Right brace
# → number sign.		

White spaces:

→ Blank space
→ Horizontol tab
→ carriage return
→ New line
→ Form feed.

Trigraph characters:

*. Many non-English keyboards do not support all the characters mentioned in ANSI C introduces the concept of "Trigraph" sequences to provide a way to enter certain characters that are not available on some keyboards.

*. Each trigraph sequence consists of three characters (two question marks followed by another character).

eg: if a keyboard does not support square brackets, we can still use them in a program using the trigraphs ?? (and ??).

Trigraph sequence	Translation
??=	# number sign
?? (	[left bracket
??)	] right bracket
?? <	{ left brace
?? >	} right brace
?? !	| vertical bar
?? /	\ backslash
?? '	^ caret
?? -	~ tilde

C Tokens:

*. In a passage of text, individual words and punctuation marks are called **tokens**.

Keywords → float, while

Identifiers → main, amount

Constants → -15.5, 100

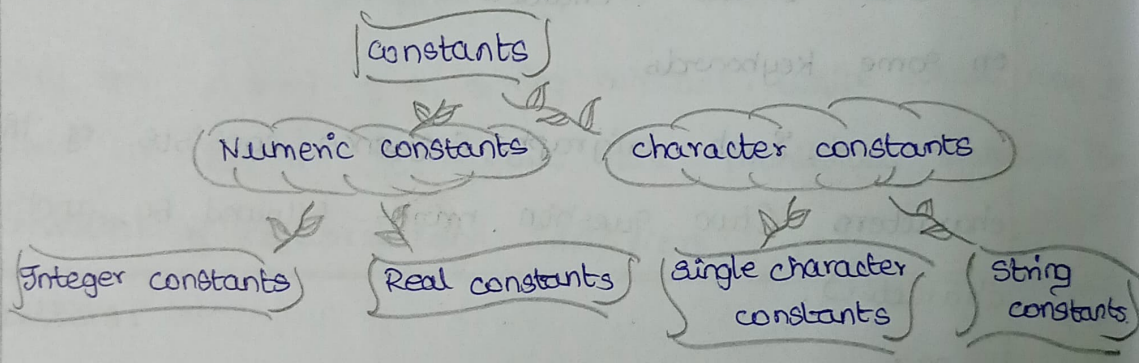
Strings → *ABC*, *year*.

Operations → +, −, *, /

Special Symbols → [], {}.

Constants:

*. Constants in c refer to fixed values that do not change during the execution of a program.



Integer constants:

*. An integer constant refers to a sequence of digits. There are three types namely decimal integer, octal integer or hexadecimal integer.

⇒ Decimal integers consists of a set of digits, 0 through 9, preceded by an optional − or + sign.

*. Embedded spaces, commas, and non-digit characters are not permitted between digits.

eg: 15 750 20,000 $1000 are illegal numbers.

⇒ Octal integer consists of any combination of digits from the set 0 through 7, with a leading 0.

eg: 037 0 0435 0551.

=) Hexadecimal constants include alphabets (A through F) or (a through f). (A through F) represents the numbers (10 through 15).

eg: 0x2 0x9F 0xbcd 0x.

=) We rarely use octal and hexadecimal numbers in programming.

*. The largest integer value that can be stored is machine-dependent. It is 32767 on 16-bit machines and 2,147,483,647 on 32-bit machines.

Real constants:

*. Integer numbers are inadequate to represent quantities that vary continously, such as distances, heights, temperatures, prices and so on.

*. These contains fractional parts like 17.5548, such numbers are called real constants.

eg: 0.0083 -0.75 435.36 +247.0

*. A real number may also be expressed in decimal notation. For example, the value 215.65 may be written as 2.15565e2 in exponential notation, e2 means multiply by 10^2.

Manbissa e exponent:

*. The manbissa is either a real number expressed in decimal notation or an integer.

*. The exponent is an integer with an optional plus or minus sign and can be written in lowercase or uppercase.

eg: 0.65e4 12e-2 1.5e+5.

* Embedded white space is not allowed.

* Exponential notation is useful for representing numbers that are either very large or very small in magnitude.

Single character constant :

* A single character constant contains a single character enclosed within a pair of single quote marks.

eg: '5', 'X', ' '

* The character constant '5' is not the same as the number 5. The last constant is a blank space.

* character constants have integer values known as ASCII values.

eg: printf("%d", 'a');

String Constants :

* A string constant is a sequence of characters enclosed in double quotes.

* The characters may be letters, numbers, special characters and blank space.

eg:

"Hello!" "1987" "WELL DONE"

* A character constant is not equivalent to the single character string constant.

* A single character string constant does not have an equivalent integer value while a character constant has an integer value.

Backslash character constant :

* C supports some backslash character.
* Constants they are used in output functions.

Variables :

* A variable is a data name that may used to store a data value.
* A variable may take different values at different times during execution.
* A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. Some names are;
* Average ; height ; Total ; counter_1 , class_strength
* Variable name may consists of letters, digits, and the underscore (_), character.

Conditions :

1.) They must begin with a letter some systems permit underscore as the first character.

2.) ANSI standard recognizes a length of 31 characters. (In C99, at least 63 characters are significant)

3.) Uppercase and lowercase are significant. That is, variable Total is not the same as total or TOTAL.

4.) It should not be a keyword.

5.) White spaces is not allowed.

eg: John , x1 , T_raise ; Invalid => 123 (area)

%. 25th

Declaration of Variables:

*. After designing suitable variables names, we must declare them to the compiler. Declaration does two things:

1.) It tells the compiler what the variable name is.

2.) It specifies what type of data the variable will hold.

Declaration of variables must be done at the beginning of a statement block before they are used in a program.

1.) primary type declaration:

*. A variable can be used to store a value of any data type. That is, the name has nothing to do with its type. The syntax for declaring a variable is as follows:

```
data-type v1, v2, ...vn;
```

*. $v_1, v_2, \ldots v_n$ are the names of variables.

*. Variables are separated by commas. A declaration statement must end with a semicolon.

=> int count; => int number, total; => double ratio;

*. int and double are the keywords to represent integer type and real type data values respectively.

2.) User-defined type declaration:

*. C supports a feature known as "type definition" that allows users to define an identifier that would represent an existing data type.

*. The user-defined data type identifier can later be used to declare variables. It takes the general form:

```
typedef type identifier;
```

*. Where type refers to an existing data type and "identifier" refers to the "new" name given to the data type.

* The existing data type may belong to any class of type, including the user-defined ones.

* Remember that the new type is "new" only in name, but not the data type.

"typedef" cannot create a new type.

```
typedef int units;
typedef float marks;
```

* Here units symbolizes int and marks symbolizes float

* The main advantage of "typedef" is that we can create meaningful data type names for increasing the readability of the program.

* The definition and declaration of enumerated variables can be combined in one statement. Ex.

```
enum day {Monday, ... Sunday} week_st, week_end;
```

* Another user-defined data type is enumerated data type provided by ANSI standard.

```
enum identifier {value1, value2, ... valuen};
```

* The "identifier" is a user-defined enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces (known as enumeration constants).

* After this definition, we can declare variables to be of this "new" type

```
enum identifier V1, V2, ..., Vn;
```

* The enumerated variables $V_1, V_2, \ldots, V_n$ can only have one of the values value1, value2, ... valuen.

eg:

```
enum day {Monday, Tuesday, ... Sunday};
enum day week_st, week_end;
```

Assigning Values to Variables:

* Variables are created for use in program statements such as,

```
Value = amount + inrate * amount ;
while (year <= period)
{
.....
.....
year = year + 1 ;
}.
```

* The numerical value is stored in the variable inrate is multiplied by the values stored in amount and the product is added to amount.

* The result is stored in the variable value.

* This process is possible only if the variables amount and inrate have already been given values.

* This variable value is called the target variable.

* Similarly, the variable year and the symbolic constant PERIOD in the while statement must be assigned values before this statement encountered.

Assignment statement:

* values can be assigned to variables using the assignment operator as follows:-

```
Variable_name = constant ;
```

* As assignment statement implies that the value of the variable on the left of the "equal sign" is set equal to the value of the quantity on the right.

```
year = year + 1 ;
```

means that the "new value" of year is equal to the "old value" of year plus 1.

variable at the time the variable is declared. This takes the following form:

data-type variable_name = constant ;

*. The process of giving initial values to variables is called initilization.

*. C permits the initialization of more than one variables in one statement using multiple assignment operators.

*. Some compilers permit the use of the "prompt message" as a part of the control string in scanf, like.

scanf ("Enter a number %d", &number);

Reading data from keyboard:

*. Another way of giving values to variables is to input data through keyboard using the scanf function.

*. It is a general input function available in C and is very similar concept of the printf function.

*. It works much like an INPUT statement in BASIC. The general format of scanf is as follows:

scanf ("control string", &variable1, &variable2, ...);

*. The control string contains the format of data being received.

*. The ampersand symbol & before each variable name is an operator that specifies the variable name's address.

*. The use of scanf provides an interactive feature and makes the program 'user friendly'. The value is assigned to the variable number.

Symbolic Constants:

*. We often use certain unique constants in the program.

*. These constants may appear repeatedly in a number of places in the program.

Eg: 3.142, representing the mathematical constant "Pi".

*. We face two problems in the subsequent use of such programs;

1.) problem in modification of the program and

2.) Problem in understanding the program.

Modifiability:

*. We may like to change the value of "Pi" from 3.142 to 3.14159 to improve the accuracy of calculations or the number to 50 to 100 to process the test results of another class.

*. In both the cases, we will have to search throughout the program and explicitly change the value of the constant wherever it has been used.

*. If any value is left unchanged, the program may produce disastrous outputs.

Understandability:

*. When a numeric value appears in a program, its use is not always clear, especially when the same value means different things in different places.

*. Assignment of such constants to a symbolic name frees us from these problems.

Eg: The name STRENGTH for no. of students. and PASS_MARK for pass marks.

#define symbolic-name value of constant.

Eg:

#define STRENGTH 100

#define PI 3.14159.

*. Symbolic names are sometimes called constant identifiers.

*. Since the symbolic names are constants, they do not appear in declarations.

*. The following rules apply to #define statement which define a symbolic constant:

1.) Symbolic names have the same form as variable names.

2.) No blank space between the pound sign '#' and the word define is permitted.

3.) '#' must be the first character in the line.

4.) A blank space is required between #define and symbolic name and between the symbolic name and the constant.

5.) #define statements must not end with a semicolon.

6.) symbolic names are NOT declared for data types. Its data type depends on the type of constant.

*. #define statement is a preprocessor compiler directive and is much more powerful that what has been mentioned here.