CSC 202.2: Computer Programming and Applications NOT FOR SALE

INTRODUCTION TO COMPUTER PROGRAMMING LANGUAGE

Definition of programming language.

Language is a mode of communication that is used to share ideas, opinions, with each other. As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language 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) to perform a specific task. It is mainly used to develop desktop application, websites, and mobile application.

Computer Program: Is a sequence of instructions written using a computer programming language to perform specific tasks. Everything a computer does is done by using a computer program.

There are mainly two categories of computer programming language:

(a) Low Level Language and (b) High Level Language

- **Low Level Language:** These are programming language mainly developed to provides little or no abstraction of programming concepts and it is very close to writing actual machine instructions.

Two types of Low-Level Language

- **Machine Language:** It is written as a program understood basically for the functioning of the computer hardware, it is usually coded in binary digit.
- **Assembly Language:** It is written for a specific type of processor like microprocessor to guide. It to execute instructions e.g. written for operating system, it consumes less memory but very slow execution.

- **High Level Language:** Is a programming language with strong abstractionthat enables development of a program in a much more user-friendly programming context. e.g., C, FORTRAN, Pascal, JAVA, C#, c++ etc. They are called High Level Languages because they are closer to human languages than machine language.

Difference between Machine, Assembly, High Level Languages

Feature	Machine	Assembly	High Level
Form	0‘s and 1‘s	Mnemonic codes	Normal English
Machine Dependent	Dependent	Dependent	Independent
Translator	Not Needed	Needed(Assembler)	Needed(Compiler)
Execution Time	Less	Less	High
Nature	Difficult	Difficult	Easy
Memory Space	Less	Less	More

Qualities of Good Programming

(i) **Efficiency:** It must be fast & should take less memory.
(ii) **Reliability:** It must produce expected outcome or result without failure.
(iii) **Accuracy:** It must be correct following the instructions.
(iv) **Portability:** Must be machine independent & can be transferred
(v) **Usability:** It must be useful for its purpose.
(vi) **User-friendly:** It must be easy to use.
(vii) **Maintainability:** It must be easy to be modified & maintained.

Steps in Writing Program

(I) Identify and understand the problem
(II) Development of Algorithm
(III) Develop code (coding).
(IV) Compiling debugging and test-running the program.
(V) Documentation.

Debugging Bug: Bugs are errors found in the program and debugging is the process of correcting the errors. There are three types of Bugs (Errors)

(i) **Syntax Errors:** It is error as a result from violation of the syntax or order of the programming language
(ii) **Logical Errors:** Error due to semantic rule, making the program to produce unexpected result. Logical errors are also called semantic error.
(iii) **Run Time Error:** Such error causes a program to end or terminate abruptly, due to wrong or illegal operation or data entry.

PERFORMING DEBUGGING.

1. Realizing existence of bugs.
2. Separating the source of bug
3. Identification of cause of bug
4. Selecting a fix for the bug
5. Implementing a fix for the bug and test.

Language Translators

These are the programs which are used for converting the programs in one language into machine language instructions, so that they can be executed by the computer.

1) Compiler: It is a program which is used to convert the high-level language programs into machine language

2) Assembler: It is a program which is used to convert the assembly level language programs into machine language

3) Interpreter: It is a program, it takes one statement of a high-level language program, translates it into machine language instruction and then immediately executes the resulting machine language instruction and so on.

COMPILER AND INTERPRETER

The program written in high level language is known as source program and the corresponding machine level language program is called an object program. Both compiler and interpreter perform the same task but there working is different. Compiler read the program at-a-time and searches the error and lists them. If the program is error free then it is converted into object program. When program size is large then compiler is preferred. Whereas interpreter read only one line of the source code and convert it to object code. If it checks error, statement by statement and hence of take more time.

Comparison between a Compiler and Interpreter

COMPILER	INTERPRETER
A Compiler is used to compile an entire program and an executable program is generated through the object program	An interpreter is used to translate each line of the program code immediately as it is entered
The executable program is stored in a disk for future use or to run it in another computer	The executable program is generated in RAM and the interpreter is required for each run of the program
The compiled programs run faster	The Interpreted programs run slower
Most of the Languages use compiler	A very few languages use interpreters.
An example of program that make use of Compiler are C, C++, Java, C#, COBOL, e.t.c	An example of program that make use of Interpreter are Javascript, Python, PHP, e.t.c

Problem Solving Techniques

Problem solving is the process by which the unfamiliar situation is resolved and that is the core essence of computer Science. Solving a problem requires a sequential process of analyzing information related to a given situation (problem).

Steps in solving a problem

a) Identification and definition of a problem
b) Analyze the problem
c) Generate potential solution
d) Select the best solution
e) Evaluate and implement the solution

The following summarizes the necessary inputs needed for each of the steps.

(a) Define the Problem (System Analyst)

- Language to be deploy
- Type of report
- Type of data
- Stages of processing
- Type of hardware

(b) Analyze the problem (the use of Algorithm and Flowcharts). To analyze the problem at hand, here it is possible to formulate a mathematical model using algorithm and flowcharts. In brief;

Algorithm: this involves the step -by-step procedure of solving a problem. For example, how to prepare a cup of tea. The following steps explain the procedure in preparing a cup of tea

Step 1: wash the kettle

Step 2: add clean water into it

Step 3: apply heat to the kettle.

Step 4: check whether it is boiling

Step 5: if yes go to step 6 else go to step 3.

Step 6: add ingredients

Step 7: Stop.

(c) Generate Potential solution

Input: Pick list of possible solutions and device a decision-making criterion.

Output: Take a decision of what solution you will implement

Flowchart: A step-by-step representation of all that is expected in pictorial form. Its involves documentation in symbolic form(diagrams)/ the processing steps (algorithms) involves in the program. In essence, it is a diagrammatic representation of an algorithm.

(d) Analyze the solution (choosing the suitable language and coding). Choosing a suitable programming language in the execution of the solution rely on the prescribed outputs.

(e) Select the best solution (Optional use of computer time, effort and resources)

2.0 ALGORITHM

An algorithm is a finite/specific set of clearly defined procedures of solving a given task or problem.

2.1 PROPERTIES/CHARACTERISTICS OF ALGORITHMS

An algorithm must have the following properties/characteristics.

a) It must be specific or definite.
b) Its Correctness.
c) It Starts and ends appropriately.
d) Steps must follow proceeding step that must be well defined.

e) It must have specific or finite number of steps.

f) No ambiguity/complications.

2.2 ALGORITHM COMPONENTS

a) **Sequence:** steps of the statements are written in order to solve the given problem

b) **Selection:** The instructions are written such that decision /choice making is involved

c) **Iteration:** Simply means repetition of instruction to execute. That is going all over a set of instructions from start to end.

2.3 ALGORITHM DESIGN

a) **Pseudo-code:** Written as sentence which is usually not understood by a compiler but for human. Not a computer Programming language, only to guide a program to write computer readable language.

b) **Flowchart:** As the name implies is a step-by-step flow of instructions in a chart to solve a given task. It can also be defined as the diagrammatic representation of an algorithm.

Example 1: Write an algorithm to find the area of a circle

$$Area = \pi r^2$$

Step 1: Let pie = 3.142

Step 2: Reading / input value for radius (Read r)

Step 3: Compile/Calculate Area = pie × r × r

Step 4: Display Area

Step 5: Stop/End

Example 2: Find the larger number of Two given numbers with an algorithm solution

Step 1: Read A, B

Step 2: Is A > B? (If yes, go to **step 3**, otherwise/ else go to **step 4**

Step 3: Print A

Step 4: Print B

Step 5: Exit.

Example 3: Develop an algorithm to generate all integers from 1 to 100 and then compute their sum.

Step 1: [Initialize **counter** and **sum**] set count = 1, set sum = 0

Step 2: Repeat **Step 3** to **Step 5** While count < = 100

Step 3: Set sum = sum + count

Step 4: [Display count] print: count

Step 5: [Incremental count] set count = count + 1

[End of Step 2 (while) loop]

Step 6: Print sum [Display value of sum]

Step 7: Stop

3.0 FLOWCHART

Flowchart can be described as, some form of diagrams which is used to represent an algorithm. It shows the logical flow of a program, from the input to the output (Result). In essence, it is a diagrammatic representation of an algorithm.

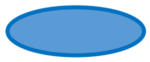
3.1 GUIDES FOR FLOWCHART DESIGN

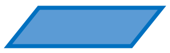
a) Must use the standard flowchart symbols
b) No crossing of flow arrows or lines in the flowchart
c) Must not appear crowded & clumsy
d) Flow direction must be consistent.

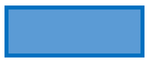
3.2 TYPES OF FLOWCHART

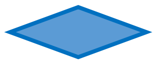
a) Sequential flowchart
b) Branched / Selection flowchart
c) Loop / Repetition flowchart

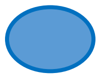
3.3 FLOWCHART SYMBOLS

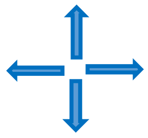
a)  Start / Stop: beginning & end

b)  Input / Output symbol.

c)  Processing / Computation/ Operation/ Event

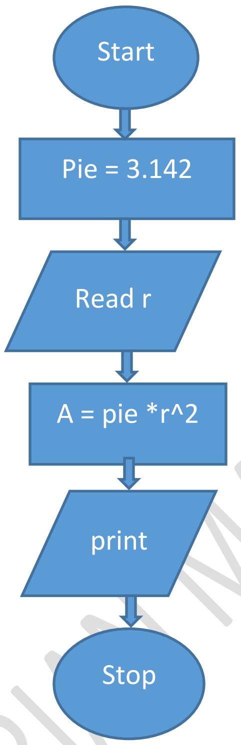
d)  Decision or Choice

e)  Connection: Continuation / Connection of parts of the chart

f)  Showing direction of flow

Example 1: Flow chart to calculate are of a circle $A = \pi r^2$

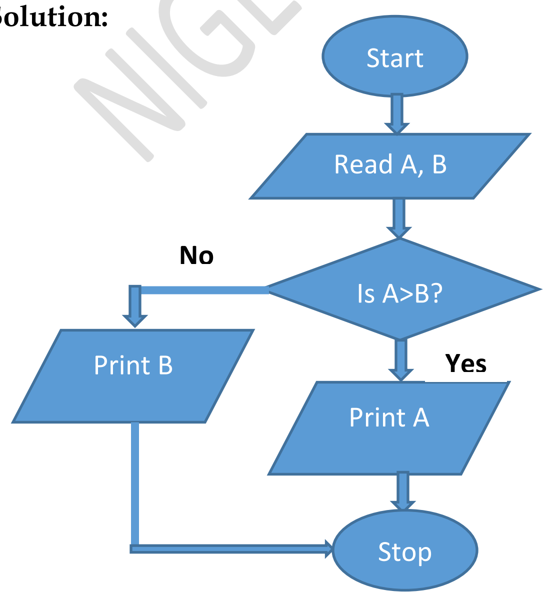
Solution:



Sequential Flowchart

Example 2: Find the larger of two numbers

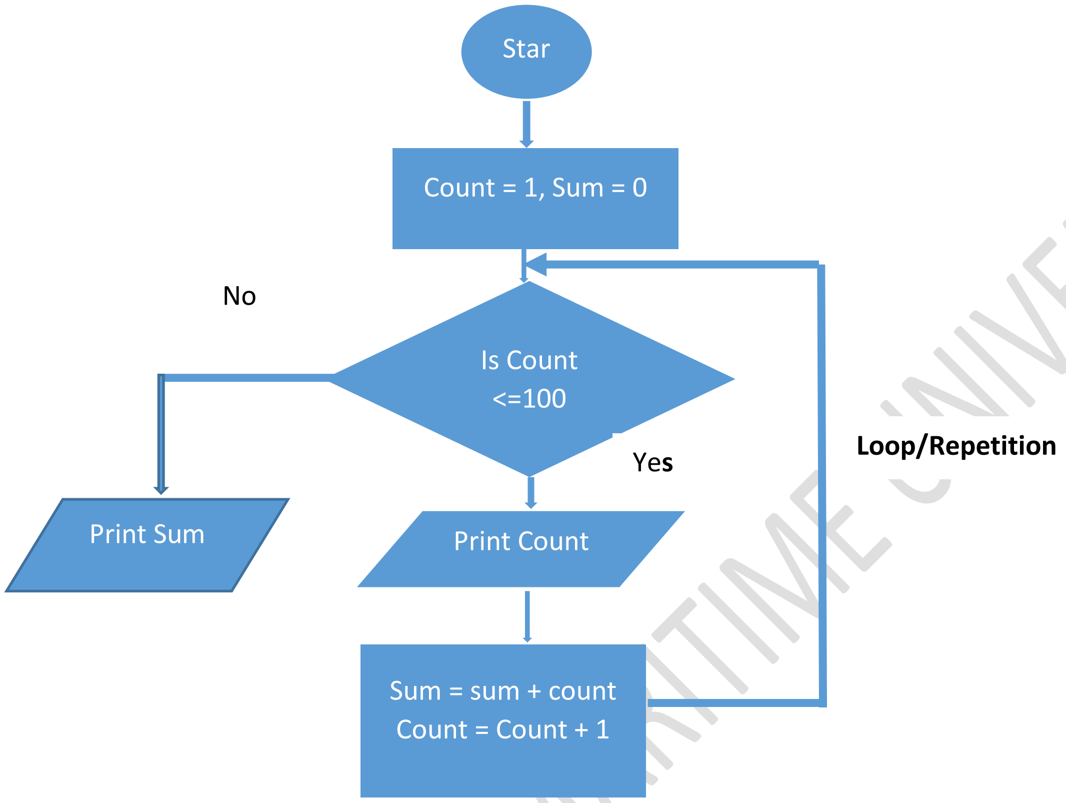
Solution:



Branched Flowchart

Fig1.2: Branched Flowchart

Example 3: Develop a flowchart to generate all positive integers from 1 to 100 and then compute their sum.



2.0 WHAT IS C PROGRAMMING LANGUAGE?

C language is one of the procedural languages which facilitates a very efficient approach to the development and implementation of computer programs. In procedural, the program code is written as sequence of instructions. Here, user specify "what to do" and also "how to do" (step by step procedure). The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce "an unambiguous and machine independent definition of the language C "while still retaining its spirit.

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to a different hardware architecture.

WHY STILL STUDY C?

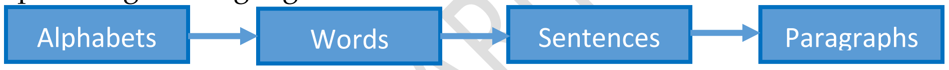
I. It is necessary foundation to C++, C# and JAVA, which are Object-Oriented Programming (OOP), as the language elements of C important programming skills.
II. Even with the advanced framework of C++, C# & JAVA, they are based on C
III. Learning C makes it easier to learn Object-Oriented Programming like C++, C#, JAVA, Python, MATLAB etc.
IV. C gives understanding of Objects, classes, inheritance, polymorphism, references, templates, exception handling etc. All of which are in object programming (OOP)
V. Many modules or parts of operating systems such as; windows, UNIX, Linux are still written in C due to its speed and general performance. In integration of devices with operating systems, these device driver programs are usually written in C.
VI. As C gives numerous language elements, human- computer interaction is possible without much effect on performance, hence is more preferred.
VII. Due to the constraints (Limitations) of memory space and time in mobile devices, microwaves ovens, ATMs, washing machines etc. The microprocessor, operating system and the program embedded in these machines are mostly programd in "C".

5.3 BEGINNING C

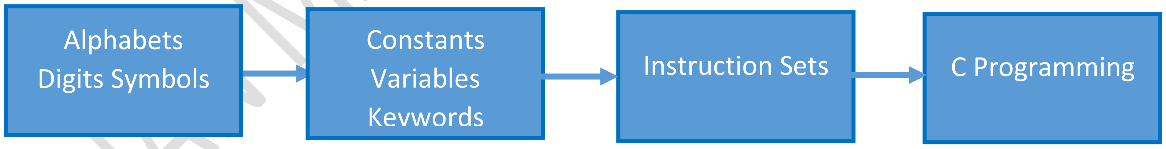
To interact with computers, one must talk in a language these computers understand. For two persons to communicate they must speak in a language they understand.

Example: Comparing C and English language.

1. Steps in English language.



2. Steps in C



Features of C Programming Language

Character set: refers to the composite number of different characters that are being used and supported by a computer software and hardware. E.g %, #, @, !, &, }, { $,), (, e.t.c.

Data types: In programming, data types is a classification that specifies the type of value a variable has and what mathematical, relational, or logical operations can be applied to it. E.g integer (int), floating point (float), Boolean (bool), e.t.c.

Naming convention: In computer programming, a naming convention is a set rules for choosing the character sequence to be used for identifiers which denotes variables, types, functions and others entities in

source code. For instance, in C programming, an identifier can be composed of letters such as uppercase, lowercase, underscore, digit but the starting letter should be either an alphabet or an underscore and not digit.
Program structure: Is the overall form of a program with particular emphasis on the individual component of the program and the interrelationship between these components.

BASIC PROGRAM STRUCTURE OF C LANGUAGE

The program written in C language follows this basic structure (basic building block). The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed. Without a proper structure, it becomes difficult to analyse the problem and the solution.

1. Documentation section
2. Preprocessor/Linking section
3. Definition section
4. Global declaration section
5. Main function section

{
Declaration section
Executable section
}

6. Sub program or function section

1. DOCUMENTATION SECTION: The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the program would like to use later. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with multiline comment /* and */. Single line comments are represented with double forward slash (//). Whatever is written between these two are called comments. Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

2. PREPROCESSOR/LINKING SECTION: This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.
All preprocessor command starts with (#). E.g of preprocessor command are #define, #undef, #ifdef, #ifndef, #endif e.t.c. for instance, #include <stdio.h> which is the standard input-output header file. There are various header files available for different purposes. Another example is # include <math.h> which is used for mathematical function in a program

3. DEFINITION SECTION : It is used to declare some constants and assign them some value.
e.g. #define MAX 25
Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

4. GLOBAL DECLARATION SECTION : Here the variables which are used through out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)
e.g. int i; (before main())

5. MAIN FUNCTION SECTION : It tells the compiler where to start the execution from. It is the first function to be executed by the computer. It is necessary for a code to include the main(). The parenthesis () is used for passing parameters (if any) to a fuction. The void main() specifies that the program will not any value. The int main() specifies that the program can return integer type data.

```
main()
{
point from execution starts
}
```

main function has two sections

a. Declaration section: In this the variables and their data types are declared.
b. Executable section: This has the part of program which actually performs the task we need.

6. SUB PROGRAM OR FUNCTION SECTION: This has all the sub programs or the functions which our program needs.

SIMPLE 'C' PROGRAM:

```
/* simple program in c */
#include<stdio.h>
main()
{
printf("welcome to C programming");
return 0;
}
```

Let us take a look at the various parts of the above program-

- The first line (/* simple program in c */) which is comment line will be ignored by the compiler as it is use for documentation and explaining program.
- The next line (#include<stdio.h>) is a preprocessor command, which tell a C compiler to include stdio.h file before going to actual compilation.
- The next line main() is the main function where the program execution begins.
- The next line printf(…) is another function available in C which causes the message "Welcome to C programming" to be displayed on the screen.
- The next line is return 0; terminates the main() function and returns the value 0. Note that without this the above lines of code will run without displaying error message
- Note that the braces {and} delineate/describe the extent or size of the function block (lines of code).

C-TOKENS

Tokens are individual words and punctuations marks in English language sentence. The smallest individual units are known as C tokens. It can also be defined as a single element of a programming language.

Operator: Operator is a symbol the tell the compiler or interpreter to perform specific mathematical, relational, or logical operation on data and produce result.

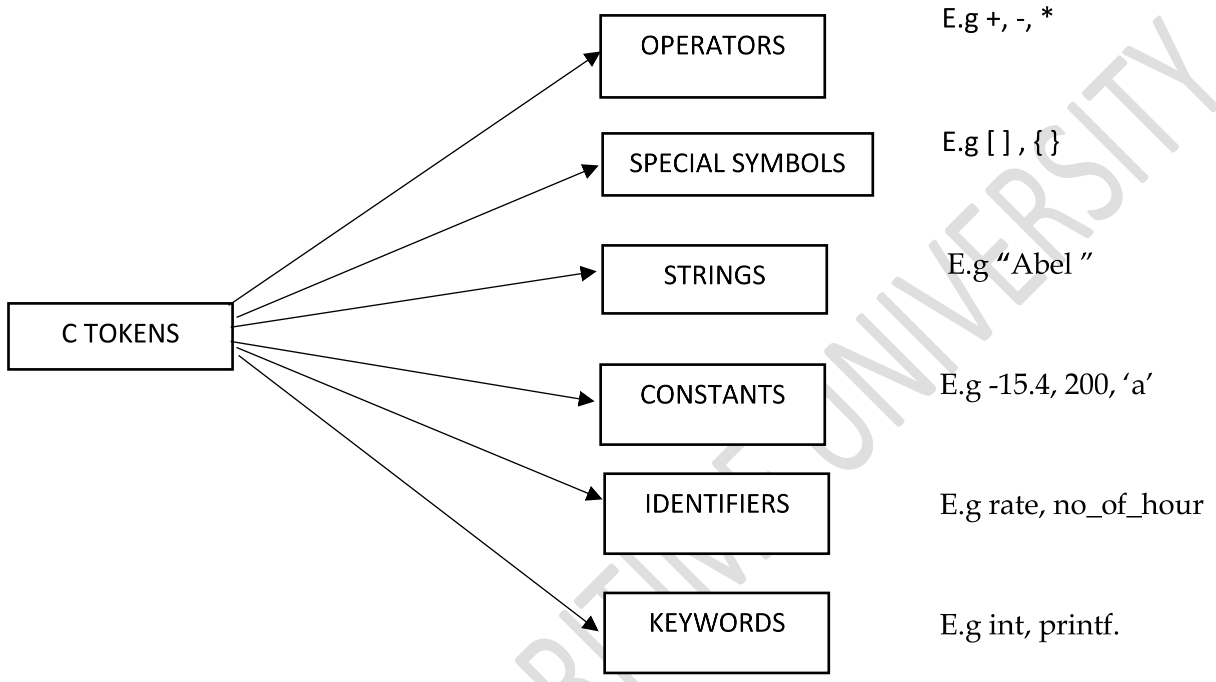
Special symbols: Are symbol character that is not considered a number or letter. E.g @, #, $, %, &, e.t.c.

String: A string is a traditionally a sequence of character which is really an "array of character". E.g "Abel", "1234", e.t.c.

Constants: Is a value that should not be altered by the programming during normal execution. e.g pie=3.142, e.t.c.

Identifiers: Are names you supply for variable, types, functions, and labels in your program. E.g int totalscore, double totalbalance, e.t.c. int and double are keywords while totalscore and totalbalance are identifiers.

Keywords: Is a word that is reserved by a program because the word has a special meaning. In C programming language, we have 32 keywords in C programming language. E.g double, int, char, e.t.c.



A C program can be divided into these tokens. A C program contains minimum **3 tokens** no matter what the size of the program is.

SET OF CHARACTERS IN C

A character in Computer programming stands for any alphabet digit or symbol which is used to represent information. For **example,**

Table 2

1.Alphabets	A, a, B, b, C, c,……………X, x, Y, y, Z, z
2.Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
3.Symbols	~,’’, !, @, #, %, ^, &, *, (),-,+, =,|,\, /, [ ], { }, < >, ?, etc.

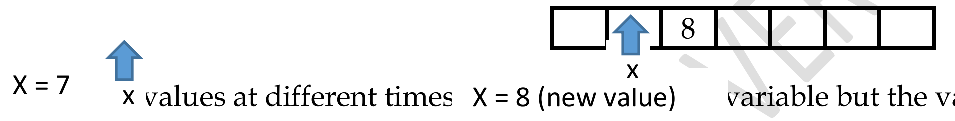
5.5 CONSTANT, VARIABLES AND KEYWORDS.

- A proper combination of the numbers, alphabet and special symbols will form constants, variables and keywords.
- Constant and Variables:
  A given constant is an entity or something which does not change (it is fixed) during the execution of a program while a variable is an entity that changes during the execution of a program.

Values are stored in computer memory which has millions of cells (memory locations) similar to human memory. These memory locations are assigned names as the values stored in a location can change, such names are called variable names. A memory location can store only one value at a time.

Example:

Fig1.6



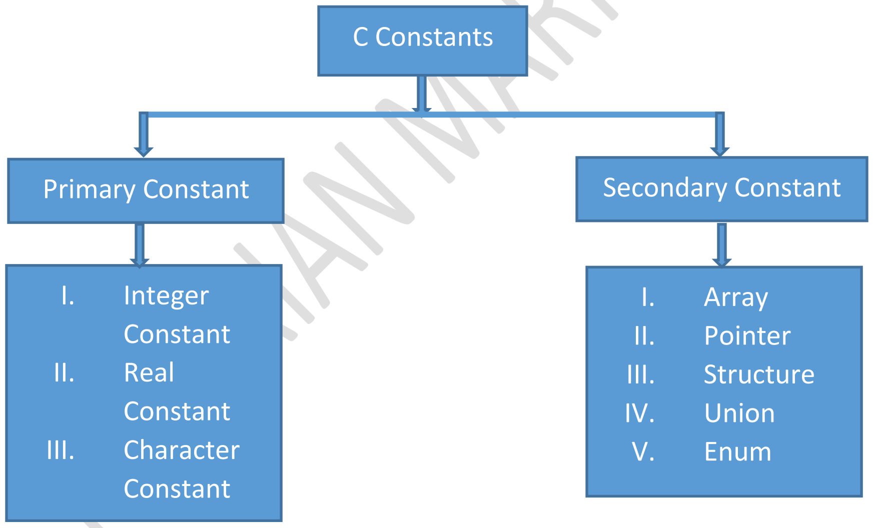
The location X stor values at different times variable but the values 7 & 8 are constants as they do not change.

5.6 TYPES OF CONSTANTS IN C

Constants in C, are of two categories;

i. Primary constants
ii. Secondary constants

They two constants further divided into sub-categories.



5.6.1 What are primary constants?

- It is made-up of Integer, real and character constants.
- Constants are constructed and in the process of construction, the following rules are to be obey.

5.6.1.0 Integer Constant

I. An integer constant must not contain a decimal point.

II. An integer constant can be positive or negative.
III. An integer constant must be at least one digit.
IV. An integer constant is assumed to be positive if no negative sign proceeds it,
V. It is not allow having Commas and blank spaces within an integer constant.
VI. The range of integer constants is -32768 to 32767. (although the range depends on the compiler 16-bit or 32 bit)

Examples: 642, + 782, -421, -7000 etc.

There are three types of integer constants:

Decimal Integer Constants Integer constants consisting of a set of digits, 0 through 9, preceded by an optional – or + sign. Example of valid decimal integer constants 341, -341, 0, 8972

Octal Integer Constants Integer constants consisting of sequence of digits from the set 0 through 7 starting with 0 is said to be octal integer constants. Example of valid octal integer constants 010, 0424, 0, 0540

Hexadecimal Integer Constants
Hexadecimal integer constants are integer constants having sequence of digits preceded by 0x or 0X. They may also include alphabets from A to F representing numbers 10 to 15. Example of valid hexadecimal integer constants 0xD, 0X8d, 0X, 0xbD It should be noted that, octal and hexadecimal integer constants are rarely used in programming.

5.6.1.1 Real Constants

Real constants are floating-point constants, which are normally in two forms: The fractional form and the exponential form. These may be represented in one of the two forms called *fractional form* or the *exponent form* and may also have either + or – sign preceding it.

5.6.1.2 The Rules for Real Constants

1. A real constant must have a decimal point.
2. It must have at least one digit.
3. It must be either positive or negative.
4. It is by default with positive sign.

Examples: 426.0, -27.36, -58.4331, +357.41

Note:

- In real constant, the exponential form is normally used when the value of the constant is too small or extremely large.
- There are two parts in real constant, (a) The mantissa and (b) The exponential part 'e'

Example: $+5.2e-3$ $+5.2e^{-3}$ $\underbrace{+5.2}_{Mantissa}\ \underbrace{e^{-3}}_{Exponential}$

$8.4e^{7}$

$-3.e^{+5}$

$-3.7e^{-8}$

5.6.1.3 Rules for Character Constants

I. The maximum length of a character constant we can have is 1.
II. The character constant is single alphabet, a single digit or special symbol within a single quote e.g. 'B', '1', '5', '='.

Certain character constants can be represented by escape sequences like '\n'. These sequences look like two characters but represent only one.
The following are the some of the examples of escape sequences:

Escape sequence	Description
\a	Alert
\b	Backspace
\f	Form feed
\n	New Line
\r	Carriage return
\t	Horizontal Tab
\v	Vertical Tab

STRING CONSTANTS

String constants are sequence of characters enclosed within double quotes. For example, "hello"

"abc"

"hello911".

Every sting constant is automatically terminated with a special character called the **null character** which represents the end of the string. For example, "hello" will represent "hello" in the memory.

Thus, the size of the string is the total number of characters plus one for the null character.

Special Symbols

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose. [] () {} , ; : * … = #

Braces{}: These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.

Parentheses(): These special symbols are used to indicate function calls and function parameters.

Brackets[]: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

5.7 C Variables

A variable is an entity that may continue to vary in a program during the execution of a program. It is also referred to as names given to a computer memory location to store values in a computer program. Variable names are names assigned to memory locations in the computer memory.

- An integer variable stores only an integer constant, a Real Variable can store only a real constant as well as character variable stores only a character constant.

5.7.1 The Rule for Variable Names

(i) A given variable name should be any combination of alphabets, digits or underscores 1 to 31 in size.
(ii) Commas or blank spaces are permitted within a variable name.
(iii) The first character in a variable name must be an alphabet or an underscore.
(iv) Special symbols are not to be used in variable name apart from underscore symbol (e.g. Shop_gal)

Example: si_int
m_hra
pop_e_89

These set of rules are the same for all types of Primary and Secondary Variable names. In C language, the variable name MUST be declared first. The compiler distinguished between variable names, since variable name type **must** be declared. The variable name type declaration is made from the start of the program.
Note: The maximum length of a variable name is 31 characters.

Example:
```
int  si, m _ hra;
Float   cassil;
Char    spat;
```

Variable initialization

When we assign any initial value to variable during the declaration, is called initialization of variables. Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows –

type variable_name = value;

Some examples are –

```
int d = 3, f = 5;
Or
int a;
a=20;
```

5.7.2 Keywords in C

- keywords (reserved words) are words that have a predefined meaning in 'C' language.
- Keywords are words that cannot be used as variable.
- The words cannot be substituted for variable names, these keywords are also called "**Reserved Words"**
- There are only 32 keywords made known in C but it can be more depending on the type of compiler.

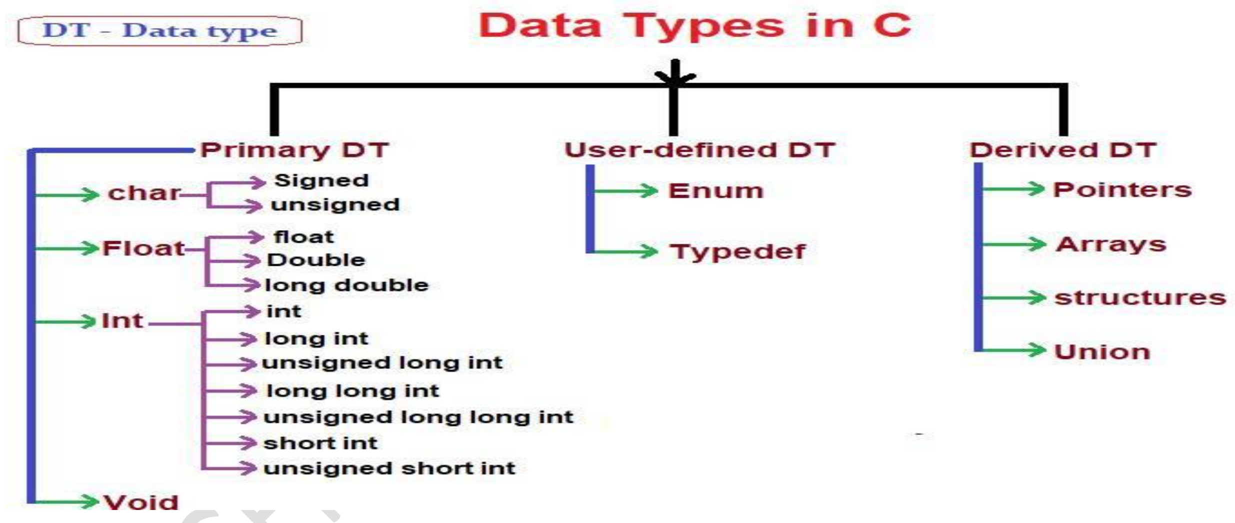
Example:

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

IDENTIFIERS

Names of the variables and other program elements such as functions, array, etc are known as identifiers.
There are few rules that govern the way variable are named(identifiers).
1. Identifiers can be named from the combination of A-Z, a-z, 0-9, _(Underscore).
2. The first alphabet of the identifier should be either an alphabet or an underscore. digit are not allowed.
3. It should not be a keyword.
Eg: name,ptr,sum
After naming a variable we need to declare it to compiler of what data type it is .
The format of declaring a variable is:
Data-type id1, id2,.....idn;
where data type could be float, int, char or any of the data types.
id1, id2, id3 are the names of variable we use. In case of single variable no commas are required.
eg float a, b, c;
int e, f, grand total;
char present_or_absent;

Data types in C



Data types in C refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows –

- **Basic Types:** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.

- **Enumerated types:** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
- **The type void:** The type specifier *void* indicates that no value is available.
- **Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Basic Types

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **int:** As the name suggests, an int variable is used to store an integer.
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision.
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision.

Derived types

Array: Same as any other language, Array in C stores multiple values of the same data type. That means we can have an array of integers, chars, floats, doubles, etc

int numbers[] = ;

double marks[7];

float interest[5] = ;

Pointers: Pointers are considered by many to be complex in C, but that is not the case. Simply put, a pointer is just a variable that stores the address of another variable. A pointer can store the address of variables of any data types. This allows for dynamic memory allocation in C. Pointers also help in passing variables by reference.

Structs: A struct is a composite structure that can contain variables of different data types. For example, all the student data that we declared earlier in basic data types can be put under one structure. Instead of having the information scattered, when we give it a structure, it is easier to store information about more students.

Union: With a union, you can store different data types in the same memory location. The union can have many members, but only one member can have a value at one time. Union, is thus, a special kind of data type in C.

DataType	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	16		%Lf

EXPRESSION AND OPERATOR

4.1.3 OPERATORS AND EXPRESSIONS IN PROGRAMMING.

Table 1

OPERATOR	DEFINITION	ALGEBRAIC EXPRESSION	PROGRAMMING EXPRESSION
+	Addition	a + b	a + b
-	Subtraction	a - b	a - b
*	Multiplication	a * b	a * b
/	Division	a/b	a / b
\	Integer Division	a/b	a \ b
^	Exponential	A^b	A ^ b
=	Assignment	A= b + c	A= b + c
%	Modular Division (a Modulus b)	A % b	5 % 2 = 1 10 % 3= 1

Expressions

An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:-

int z= x+y // arithmatic expression
a>b //relational
a==b // logical
func(a, b) // function call

Therefore a and b are called operands

Expressions consisting entirely of constant values are called *constant expressions*. So, the expression
121 + 17 - 110
is a constant expression because each of the terms of the expression is a constant value. But if i were declared to be an integer variable, the expression
180 + 2 – j
would not represent a constant expression.

Operator

This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operation.
Several operators are there those are, arithmetic operator, assignment, increment , decrement, logical, conditional, comma, size of , bitwise and others.

1. Arithmatic Operator

This operator used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator. Where Unary arithmetic operator required only one operand such as +,-, ++, --,!, tiled. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction), *(multiplication), /(division), %(modulus). But modulus cannot applied with floating point operand as well as there are no exponent operator in c. Unary (+) and Unary (-) is different from addition and subtraction. When both the operand are integer then it is called integer arithmetic and the result is always integer. When both the operand are floating point then it is called floating arithmetic and when operand is of integer and floating point then it is called mix type or mixed mode arithmetic . And the result is in float type.

2.Assignment Operator

A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement and assignment expression. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression. When variable on the left hand side is occur on the right hand side then we can avoid by writing the compound statement. For example,
int x= y;
int Sum=x+y+z;

3.Increment and Decrement

The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one. Similarly decrement operator decrease the value of the variable by one. And these operators can only use with the variable, but cann't use with expression and constant as ++6 or ++(x+y+z). It again categories into prefix post fix. In the prefix the value of the variable is incremented 1st, then the new value is used, where as in postfix the operator is written after the operand (such as m++,m--).
EXAMPLE
let y=12;
z= ++y;
y= y+1;
z= y;
Similarly in the postfix increment and decrement operator is used in the operation. And then increment and decrement is perform.
EXAMPLE
let x= 5;
y= x++;
y=x;
x= x+1;

4.Relational Operator

It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression. Here the value is assign according to true or false value.
a.(a>=b) || (b>20)
b.(b>a) && (e>b)
c. 0(b!=7)

5. Conditional Operator

It sometimes called as ternary operator. Since it required three expressions as operand and it is represented as (? , :).
SYNTAX
exp1 ? exp2 :exp3
Here exp1 is first evaluated. It is true then value return will be exp2 . If false then exp3.

Example

```
void main()
{
int a=10, b=2
int s= (a>b) ? a:b;
printf("value is:%d");
}
```

Output:

```
Value is:10
```

6. Comma Operator

Comma operator is use to permit different expression to be appear in a situation where only one expression would be used. All the expression are separator by comma and are evaluated from left to right.

EXAMPLE

```
int i, j, k, l;
for(i=1,j=2;i<=5;j<=10;i++;j++)
```

Logical or Boolean Operator

Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression. C has three logical operators :

Operator	**Meaning**
&&	AND
||	OR
!	NOT

Where logical NOT is a unary operator and other two are binary operator. Logical AND gives result true if both the conditions are true, otherwise result is false. And logial OR gives result false if both the condition false, otherwise result is true.

6.0 YOUR FIRST C PROGRAM

C consists of functions and variables

A. A function in C is made up of statements that specify the computational operations to be performed while the variables stored assigned values during the execution of the program.

Example 1: Write a program in C to display "Hello World".

```
/* First C Program: Hello World*/
#include <stdio.h>

int main(void)
{
        printf ("Hello World! \n");
}
```

Fig1.8

6.1 Explanation of lines

1) Comments in C starts with /* and are terminated with */.
2) Inclusion of a standard library header-file. Almost all of the functions of C comes from libraries.
3)
4) Every program in C has **main()** as the entity-point function which are in two forms:
   Int main (void)
   Int main (int argc, char * argv[])

(5& 7) the braces {and} delineate/describe the extent or size of the function block (lines of code)

6) The above program has only the statement, where a function calls to the standard library function **printf()**, this prints a character string to standard output(the screen).

Note: the string constant "Hello World! \n", here the \n in the end of the string is called an escape character to start a new line.

End; finally, the statement in C are terminated with a semicolon (;).

B. The output on the screen for a C program is obtained using a default library function e.g
   - **Printf()**
     %f for printing real values (e.g 2.137)
     %d for printing integer values (e.g -1,2,3…)
     %c for printing character values (e.g A,0,1,2…)

Example 2: write a C program to calculate Simple interest.

```
        /* Calculating Simple interest */
        /* Date: --/03/2020 */
main()
{
        int p,n;
        Float r,si;
        P = 2000;
        N = 4;
        R = 9.2;
/* Formula to calculate Simple Interest */
        Si = p*n*r/100;
        Printf("%f", si);
}
```

6.2 Important points in the program.

❖ Any variables used in the program must be first declared e.g.

```
int p, n;
float r, si;
```

6.3 COMPILING AND EXECUTING IN C

The written C program is typed into an Integrated Development Environment (IDE) which consist of an Editor and a Compiler. Then the program is compiled and run to output the result. In C program **printf**() gives the output values to the screen while **scanf()** receives the values from the keyboard.

There are three (3) basic instructions in C;

1. **Type declaration instruction:** this is used to declare the type of variables used in a C program.
   Example
   ```
   int I;
   int j;

   Float a;

   I = 10;

   J =25;

   A =1.32;
   ```
2. **Arithmetic instruction:** it is used to perform arithmetic operations between variables and constants.
3. **Control instruction:** this is for the control of the sequence of execution of different statements in the C program.

6.4 Arithmetic Instruction examples

In C program arithmetic instruction consists of a variable name on the left hand side (LHS) of the equality sign (=) whereas variable constants on the right hand side (RHS) which are connected by the arithmetic operators eg. +,-,*,/ etc.

Example:

```
int x;
float y;

float z;

x = 3;

y = 1.4;

z = x + y;
```

Here z is variable name on the LHS while x, +, y are all on the RHS.

6.5 INTEGER AND FLOAT CONVERSIONS

There are rules used for the conversion of floating points and integer values in C.

a) An arithmetic operation between integers gives an integer result.
b) A given operation between a real & real always gives real value
c) A given operation between a real value and an integer result in a real value.

Example:

Operation	Result
5/2	2
5.0/2	2.5
5/2.0	2.5
5.0/2.0	2.5

Operation	Result
2/5	0
2.0/5	0.4
2/5.0	0.4
2.0/5.0	0.4

Fig1.11

6.6 TYPE CONVERSION IN ASSIGNMENTS

When an integer is assigned a float value, that value will be demoted to an integer value and stored. Also, when a floating point is assigned an integer value, that integer value is promoted and stored as floating point value.

Example:

```
int     i;
Float   b;
i =     2.7
b =     40
```

Here i (which is integer value is demoted from 2.7 to 2 and stored as integer value. Then b, which is declared as floating point (real) value is promoted from 40 (integer) to 40.000 and stored as floating point value).

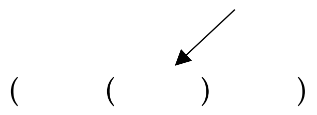
6.7 PRECEDENCE OF OPERATORS

The operators in C have an order of operation in hierarchy (priority).

PRIORITY	OPERATORS	DESCRIPTION
1st	× / %	Multiplication $\longrightarrow$ Division $\longrightarrow$ Modular Division
2nd	+, -	Addition, Subtraction
3rd	=	Assignment

NOTE:

When there are more than one set of brackets, the operators in the inner most bracket should be performed first and the next pair of bracket.



Example: Solve the expression in order of the operators.

$$x = 2 \times {}^{3}/_{4} + {}^{4}/_{4} + 8 - 2 + {}^{5}/_{8}$$

Solution:

$x = 2 \times {}^{3}/_{4} + {}^{4}/_{4} + 8 - 2 + {}^{5}/_{8}$

$x = {}^{6}/_{4} + {}^{4}/_{4} + 8 - 2 + {}^{5}/_{8} \quad \Rightarrow \times$

$x = 1 + {}^{4}/_{4} + 8 - 2 + {}^{5}/_{8} \quad \Rightarrow /$

$x = 1 + 1 + 8 - 2 + {}^{5}/_{8} \quad \Rightarrow /$

$x = 1 + 1 + 8 - 2 + 0 \quad \Rightarrow /$

$x = 2 + 8 - 2 + 0 \quad \Rightarrow +$

$x = 10 - 2 + 0 \quad \Rightarrow +$

$x = 8 + 0 \quad \Rightarrow -$

$x = 8 \quad \Rightarrow +$

Equation 1

Express $\left[\frac{2by}{d+1} - \frac{x}{3(z+y)}\right]$

$= \left(2 * b * {}^{y}/_{d + 1}\right) - {}^{x}/_{3 * (z + y)}$

$Example\colon \frac{1}{2 + 3^2} + \frac{4}{5} \times \frac{6}{7}$ *Express the algebraic equaiton to C form.*

${}^{1}/_{(2 + 3\hat{}2)} + {}^{4}/_{5} \times {}^{6}/_{7}$

$$Example\colon Z = \frac{8.8(a+b)\,{}^{2}/_{c} - 0.5 + {}^{2a}/_{(q+r)}}{(a+b) \times \left({}^{1}/_{m}\right)} \ \{Convert\ the\ equation\ to\ C\}$$

Example Evaluate the expression by showing the priorities in C.

$S = {}^{1}/_{3} \times {}^{a}/_{4} - {}^{6}/_{2} + {}^{2}/_{3} \times {}^{6}/_{g}\,;$

$(take\ a = 4, g = 3\,)$

Equation 2

Example: *Give the possible output of this C programme.*

```
main()

{

        int I =2, j = 3, k, l;

        float a, b;

                  k = i / j * j;
                  L = j / i * i;
                  a = l /j * j;
                  b = j / i * i;
        printf ("%dln%dln%fln%fln", k,l,a,b,);
        return 0;
    }
```

Fig1.12

7.0 DECISION CONTROL STRUCTURE IN C

Most often in C, decisions are made when there are different choices to be made. Hence, decision instructions can be made in C as shown below;

(i) The if statement
(ii) The if-else statement
(iii) The conditional operators.

7.1.1 THE if STATEMENT

The C language employs the keyword if to implement decision control instructions, a common form of if statement is:

```
if (this condition is true)
        execute the statement;
```

the condition which follows the keyword if is always in open and close brackets. If a given condition is true, the statement is executed but if it is not true the condition is not executed.

Relational operators are used to compare two values for equality or if one is greater than the other

EXPRESSION	… is true if
$X == Y$	X is equal to Y
X! = Y	X is not equal to Y
X < Y	X is less than Y
X > Y	X is greater than Y
X < = Y	X is less than or equal to Y
X > = Y	X is greater than or equal to Y

```
Example: /*To demonstrate the if statement*/
main()
{
          int num;
          printf ("enter a number less than 10");
                    Scanf ("%d",&num);
                              If (num <=10)
                    Printf ("what a good learner you are!");
}
```

Fig 1.13

Example: in a sales company a discount of 10% is given, if the quantity of items bought is more than 100. If the quantity and price of each item typed in from the keyboard. Develop a C program to calculate the total expenses.

```
Answer:
/*To calculate total expenses*/
main()
{Int qty, dis = 0;
          Float rate, tot;
          Printf("Enter quantity and rate");
          Scanf("%dIn%fIn", &qty,&rate);
                    If (qty > 100)
                    dis = 10;
                tot = (qty * rate) – (qty * rate * dis/100);
          printf("total expenses = %f\n", tot);
          return 0;

}
```

Sometimes in a program more than one statement is needed to be executed. In such a situation these multiple statements are placed in a pair of brackets.

Example 1: To enter the current year and the year a worker has been employed are inputted through the keyboard. A cash bonus of ₦50,000 is given if the worker has worked for more than 4years otherwise nothing is given.

Answer:

```
/* To calculate cash bonus*/
main()
{
          int cashbonus, pyr, yoe, yrs_of_ser;


          Printf("Enter present year and year of employment");
          Scanf("%d%d\n",&pyr,&yoe);


                    Yrs_of_ser = pyr – yoe;
                    If (yrs_of_ser > 4)
                    {
                              Cashbonus = 50,000;
                              Printf("cashbonus = %d", cashbonus);
                    }
}
```

Fig 1.15

Here the two statements are possible to be executed if the conditions are met(satisfied). As the second brackets are concerned the program executes it immediately after the **if statement**.

7.2 THE IF-ELSE STATEMENT

For the **if statement** it executes a statement or group of statements whenever an expression that follows the if happens to be true but will not execute if that statement or condition is false.

For the **if-else** when a statement that follows the if is true and another set of statement if a condition is false the program can still be executed. That is if the 1st **if** condition is not true then the 2nd "**else**" condition is executed.

Example:

```
If (expression of condition)
        Statement
Else
        Statement
```

Example 1: To write an **IF-ELSE** C program showing whether a given number is even or odd.

Answer:

```
main()
{
        int num, res;

        printf("Enter a number:");
        scanf("%d",&num);
                res = num%2;
        if (res == 0)
                printf("Then number is even");
        else
                printf("The number is odd");
}
```

Another example of if else statement

Answer:

```
/* A quick demonstration f if -else */
main()
{
        int i;
        printf("Enter either 1 or 2");
        scanf("%d",&i);
        if (i == 1)
                printf("It is a bright day!");
        else
                printf("It is a wet day!");
}
```

This statement is when an **if-else** statement is written within curly brackets immediately after the first **if-else** statement.
Example: /* To demonstrate nested **if-else** */

```
 main()
{
        int k;
        printf("Enter either 3 or 4");
        scanf("%d",&k);
        if (k == 3)
                printf("Welcome!");
        else
        {
                if(k==4)
                        printf("Goodbye!");
                else
                        printf("See later!");
        }
}
```

Fig1.18

8.0 ARRAY

Arrays in C is a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. A specific element in an array is accessed by an index. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. Arrays in C are indexed starting at 0, as opposed to starting at 1. The first element of the array above is point[0]. The index to the last value in the array is the array size minus one.

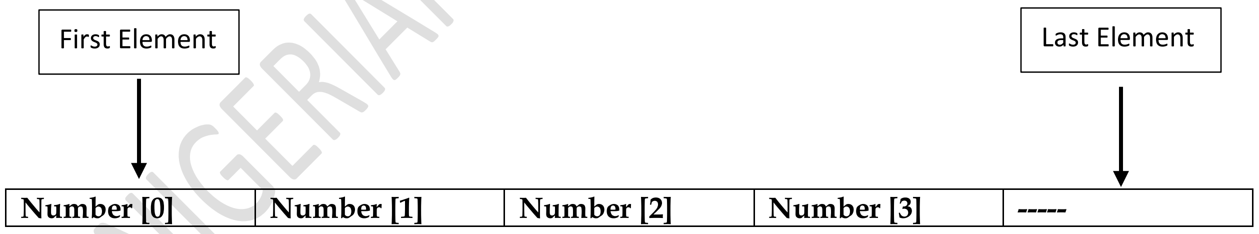


Fig1.19

8.1 Declaring Array

In C programming, it's required the declaration of the array by specifying the type and uses the square bracket to represents the number of elements as follows:

type array**Name**[array**Size**];

This type of array is called a single-dimensional array where the **type** can hold any C data type. The **arraySize** must be an integer constant which is a non-zero.

Example1:

Int student[3];

From the above, it signifies that the name of the array is **student** and the data type is **integer** (int) while the **size** of the array is three (3) {([0][1][2]}.

Example 2:

Double business[10]**;**

8.2 Initializing Arrays

In C programming, an array can be initialized by either one by one or applying single-statement. For instance,

$$\text{double} \quad \text{business}[5] = \{2000.0, 5.0, 7.3, 6.0, 50.0\};$$

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array

$\text{business}[3] = 6.0;$

In the above statement, the assigned value for the 3rd element is 6.0. The arrays start counting its element from index 0 as the first element and this is known as the base index while the last index is known when 1 is subtracted from the total size of the array.

The diagram below shows the pictorial representation of the array as explained in the above example.

	0	**1**	**2**	**3**	**4**
business	2000.0	5.0	7.3	6.0	50.0

8.3 Accessing Array Elements

The elements in an array can be access through the array name using indexing. This is carried out by ordering the index of the element in a square bracket [] following the array name. for clearer understanding the following example makes better.

$$double\ market = business\ [10];$$

In the above example of array, market is assigned with 10th elements. The three major concepts used in array are shown in the program below including declaration, assignment and accessing of arrays.

```
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Fig1.21

The above code when compiled and executed, the result can be as follows:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

*Datatype

***Pointer:**
A pointer in C is a variable whose value is the address of another variable. This shows the direct address of the memory location. A pointer is declared before it is used to store any variable address in a program. The basic form a pointer variable is declare is as follows:

$$type * var - name;$$

From the general form of a pointer, **type** is the pointer's base type, is should be notified that the data type must be the one that acceptable in C and **var-name** is the name of the pointer variable. The asterisk (*) is used to declare a pointer, it is used to designate a variable as a pointer.

$int \ \ * ip; \ \ /* pointer\ to\ an\ integer */$
$int \ \ * dp; \ \ /* pointer\ to\ a\ double */$
$int \ \ * fp; \ \ /* pointer\ to\ a\ \ float */$
$int \ \ * ip; \ \ /* pointer\ to\ a\ character */$

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

```
#include <stdio.h>

int main () {

   int  var = 20;    /* actual variable
declaration */
   int  *ip;         /* pointer variable
declaration */

   ip = &var;  /* store address of var in
pointer variable*/

   printf("Address of var variable: %x\n", &var
);

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n",
ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

After the program had been compiled and executed, the following result will appear

```
Address of var va riable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```