

FUNDAMENTALS OF
PROGRAMMING
LANGUAGE 'C'

DR. PRANAV PATEL

red'shine
PUBLICATION
INDIA

CONTENTS

SR.NO.	CHAPTER NAME	PAGE NO.
1	INTRODUCTION TO PROGRAMMING	1
2	C TOKENS	15
3	OPERATORS	20
4	DECISION MAKING AND BRANCHING	25
5	DECISION MAKING AND LOOPING	38
6	ARRAYS	43
7	CHARACTER ARRAYS AND STRING	48

PRACTICAL LIST

NO.	PRACTICAL	PAGE NO.
1	Write a C program to display "hello computer" on the screen.	53
2	Write a C program to print roll no, name and address.	54
3	Write a C program to find the area of circle using the formula $\text{Area} = \pi * r * r$.	55
4	Write a C program to find the area of rectangle, cube and triangle.(Formula are: Rectangle= $l * b * h$, triangle = $(I * b) * 0.5$, cube = $L * L * L$	56
5	Write a C program to find the area and volume of sphere. Formulas are $\text{Area} = 4 * \pi * R * R$ Volume = $\frac{4}{3} * \pi * R * R * R$.	57
6	Write a C program to evaluate simple interest $I = \frac{P * R * N}{100}$.	58
7	Write a C program to enter a distance into K.M and convert it in to meter, feet, inches and Centimeter	59
8	Write a C program to interchange two numbers.	60
9	Write a C program to convert Fahrenheit into centigrade	61
10	Write a C program for summation, subtraction, multiplication, division of two number using Arithmetic operator	62
11	Write a C program to enter days and convert into years, month and reminder days.	63
12	Write a C program to find out the largest value from given three numbers using conditional Operator	64
13	Write a C program to find the maximum number from given three numbers.	65
14	Write a C program to find that the enter number is Negative, or Positive or Zero.	67
15	Write a C program to Checked whether entered char is capital, small, digit or any special Character	68
16	Write a C program to read number 1 to 7 and print relatively day Sunday to Saturday.	69
17	Write a C program to find out the max. and min. number from given 10 numbers.	70

NO.	PRACTICAL	PAGE NO.
18	Write a C program to find the sum of digit of accepted number.	72
19	Write a C program to find the sum of first 100 odd numbers. And even numbers.	73
20	Write a C program to display first 25 Fibonacci nos.	74
21	Write a C program to check the accepted number is prime number or not.	75
22	Write a C program to display first' 100 prime numbers.	76
23	Write a C program to find factorial of accepted numbers.	78
24	Write a C program to print accepted no and its reverse number.	79
25	Write a C program to find whether the accepted number is palindrome or not.	80
26	Write a C program to convert decimal numbers into equivalent binary number.	81
27	Write a C program to convert decimal numbers into equivalent to octal number.	82
28	Write a C program to convert decimal numbers into equivalent hexadecimal numbar.	83
29	Write a C program to display first 5 Armstrong number.	85
30	Write a C program to arrange the accepted numbers in ascending order and descending order.	86
31	Write a C program to find whether the accepted string is palindrome or not.	88
32	Write a C program to convert given line into upper case or lower case.	89
33	Write a C program to count no of word, character, line and space from given text.	91
34	Write a C program to sort given string in ascending order.	93
35	<p>Write a C program to prepare pay slip using following data.</p> <p>Da = 10% of basic, Hra = 7.50% of basic, Ma = 300, Pf = 12.50% of basic, Gross = basic + Da + Hra + Ma, Nt = Gross – Pf.</p>	94

NO.	PRACTICAL	PAGE NO.
36	Write a C program to read marks and your program will display grade. Marks Grade 100 – 80 Dist 60 – 79 First 50 – 59 Second 35 – 49 Pass 0 – 34 Fail	95
37	Write a C program to find $1+1/2+1/3+1/4+....+1/n$.	97
38	Write a C program to display following output on the screen. 1 12 123 1234	98
39	Write a C program to display following output on the screen. 1 22 333 4444	99
40	Write a C program to display following output on the screen. 0 1 1 2 0 1 0 1 0 1 1 0 1 0 1	100
41	Write a C program to display following output on the screen. 2 3 2 3 3 3 4 4 4 4 5 5 5 5 5	101
42	Write a C program to display following output on the screen. 1 2 3 4 5 6 7 8 9 10	102

NO.	PRACTICAL	PAGE NO.
43	Write a C program to display following output on the screen <pre> * * * * * * * * * * * * * * *</pre>	103
44	Write a C program to display following output on the screen. <pre> * * * * * * * * * * * * * * *</pre>	104
45	Write a C program to display following output on the screen. <pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</pre>	105
46	Write a C program to display following output on the screen <pre> C CP CPR CPRO CPROGRAMMING CPRO CPR CP C</pre>	106
47	Write a C program to find maximum & minimum value from the given array.	108
48	Write a C program to find next minimum from the given array.	109

NO.	PRACTICAL	PAGE NO.
49	Write a c program to input N and find out the sum, average, max, min, total even no and total odd no. [with out use of array]	111
50	Write a c program to input N no and find out the sum, average, max, min, total even no and total odd no. [using array]	113
51	Write a c program to display the two matrix on screen and perform the addition of two matrix and print on screen.	115
52	Write a c program to display the two matrix on screen and perform the multiplication of two matrix and print on screen.	118

CHAPTER 1

INTRODUCTION TO PROGRAMMING

Algorithm:

An Algorithm is a step by step problem solving procedure that can be carried out by a computer.

Properties:

- It should be simple.
- It should be clear with no ambiguity.
- It should lead to a unique solution of the problem.
- It should involve a finite number of steps to arrive at solutions.
- It should have the capability to handle some unexpected situations which may arise during the solution of a problem.
- E.g. Division by Zero.

Advantages of algorithm:

- It is a step by step solution to a give problem which is very easy to.
- Definite procedure which can be exacted within a set period of.
- It is easy to first develop an algorithm, then convert it into a flow char and then into a computer program.
- It is easy to debug as every step has got its own logical sequence.
- It is independent of programming language.

Disadvantages of algorithms:

- It is time consuming and cumbersome as an algorithm is developed first which is converted into a flow chart and then into a computer program.







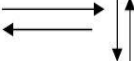
Concept of flow chart:

- A flow chart is a diagrammatic or pictorial representation of the algorithm.

- A flow chart is a graphical aid for defining algorithm inputs, process and outputs it represent flow of data through the program.

Flowcharting Symbols: -

- There are 6 basic symbols commonly used in flowchart.
- Terminal, Process, input/output, Decision, Connector and Predefined Process.
- This is not a complete list of all the possible flowcharting symbols; it is the ones used most often in the structure of Assembly language programming.

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program.
	Flow Lines	Shows direction of flow.

General Rules for flowcharting:

- All boxes of the flowchart are connected with Arrows. (Not lines).
- Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all

flowchart symbols is on the bottom except for the Decision symbol.

- The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
- Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.
- Connectors are used to connect breaks in the flowchart. Examples are:
 - From one page to another page.
 - From the bottom of the page to the top of the same page.
 - An upward flow of more than 3 symbols
- Subroutines and Interrupt programs have their own and independent flowcharts.
- All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.
- All flowcharts end with a terminal or a contentious loop.

Advantages of using flow chart:

- **Communication:** - Flow charts are a good visual aid for communication the logic of a system to all concern.
- **Quicker grasp of relationships:** - Before any problem could be solved, the relationship that exist among problem elements must be identified.
- **Effective analysis:** - The flow chart becomes blue print of program that can be broken down into detailed parts for study.

Limitations:

- **Complex Logic:** - When the program logic is complex the flow charts quickly become complex and lacks the clarity of decision table.
- **Alteration and modifications:** - If alterations are required the flow chart may require re drawing completely.

Example-1: Draw the flowchart for count of factorial values.

$$1! = 1$$

$$2! = 1 * 2 = 2$$

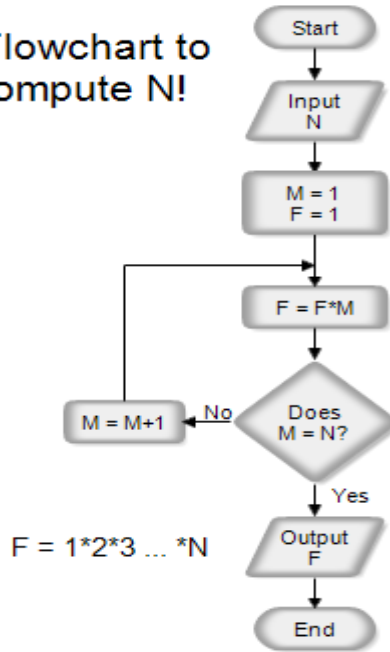
$$3! = 1 * 2 * 3 = 6$$

$$4! = 1*2*3*4 = 24$$

...

$$N! = 1*2*3*...*N$$

A Flowchart to Compute N!



Example-2: How do you draw a flowchart to calculate the first N Fibonacci numbers?

Definition: The next number is always the sum of the previous two.

Syntax: $\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$.

Example:

$$\text{Fibonacci}(2) = 0 + 1 = 1$$

$$\text{Fibonacci}(3) = 1 + 1 = 2$$

$$\text{Fibonacci}(4) = 1 + 2 = 3$$

$$\text{Fibonacci}(5) = 2 + 3 = 5$$

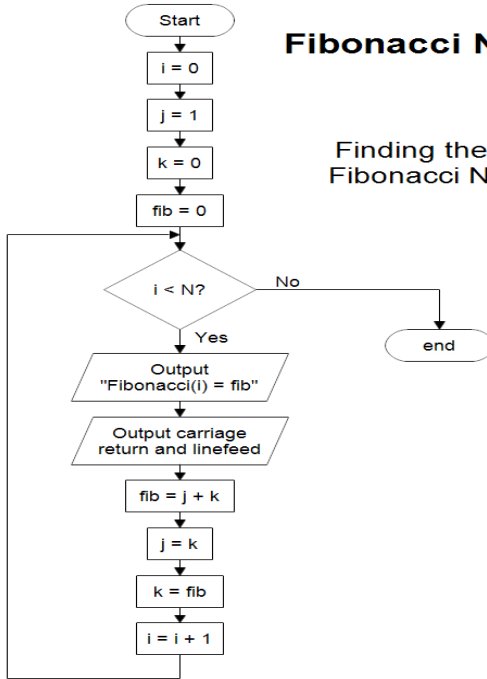
$$\text{Fibonacci}(6) = 3 + 5 = 8$$

$$\text{Fibonacci}(7) = 5 + 8 = 13$$

$$\text{Fibonacci}(8) = 8 + 13 = 21$$

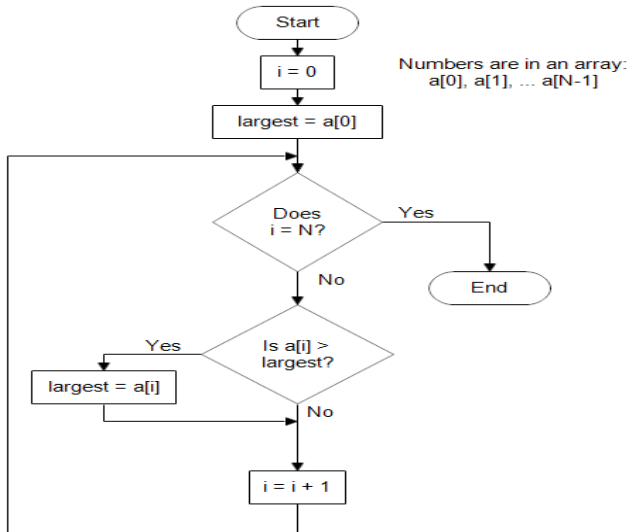
Fibonacci Numbers

Finding the First N
Fibonacci Numbers



Example-3: Finding the Largest Number in an Unsorted List of Numbers.

Finding the Largest Number in a List of Numbers

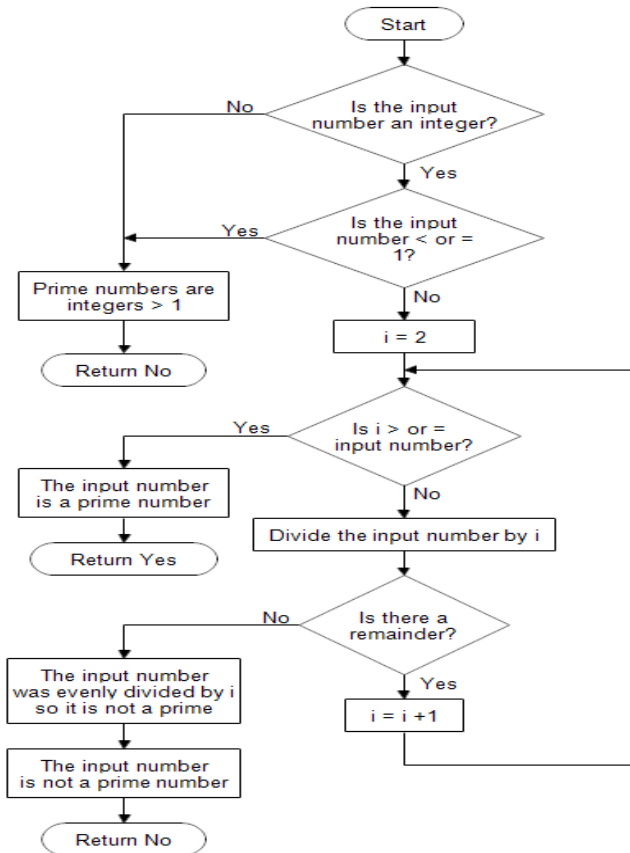


Example-4: A Flow Chart for Finding Prime Numbers.

Description: Prime numbers are positive integers that can only be divided evenly by 1 or themselves. By definition, negative integers, 0, and 1 are not considered prime numbers. The list of the first few prime numbers looks like:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

For example, 5 is a prime number because you can divide 5 by 1 evenly and divide 5 by 5 without a remainder, but if you divide 5 by any other integer, you get a remainder.

Function: IsThisNumberPrime

Machine language:

1. The set of instruction codes, whether in binary or in decimal notation, which can be directly understood by the computer program without the help of a translating program, is called a machine code or machine language.
2. A computer understands information composed of only zeros and ones. This means that a computer uses binary digits for its operations.
3. The computer's instructions are therefore coded and stored in the memory in the form of 0s and 1s is called machine language program.
4. Machine code is the fundamental language of a computer and it is normally written as string of binary 0s and 1s. However it is not necessary to coded in strings of binary digits. It can also be written using decimal digits.

Advantage:

- Programs written in machine language can be executed very fast by the computer. This is mainly because machine instructions directly understood by the CPU, no translation of program required.

Disadvantage:

1. **Machine dependent:** Because the internal design of computers is different from one another and need different electrical signals to operate. So, programmer will be required to learn a new machine code and would have to write all existing programs again, in case computer system is changed.
2. **Difficult to program:** Computer easily uses machine language; it is very difficult to write a program in this language. It is necessary for the programmer either to memorize dozens of code numbers for the commands in the machine's instruction set or constantly refer to reference card. A machine language programmer must also be an expert who knows about the hardware structure of the computer.
3. **Error Prone:** For writing a program in machine language programmer not only has to remember the operation codes but also has to keep a track of the storage locations of data and instructions. It therefore becomes very difficult to him to

concentrate fully on the logic of the problem. This frequently causes errors in programming.

4. **Difficult to modify:** It is very difficult to correct or modify machine language programs. Checking machine instructions to locate errors is about as tedious as writing them initially. In short writing a program in machine language is so difficult and time consuming that it is rarely used now a days.

Assembly language:

A low level programming language in which mnemonics are used to code operations and alphanumeric symbols are used for addresses is called assembly language.

A mnemonic meaning memory aid is a name or symbol used for some code or function.

The software which converts the assembly language program to machine language is called assembler.

Advantage:

The advantage of assembly language over the high level language is that the computation time of an assembly language is less. An assembly language program runs faster to produce the desired result.

Disadvantage:

1. Programming is difficult and time consuming.
2. Assembly language is machine dependent. The programmer must have detailed knowledge of registers and instruction set of computer, connections.
3. Program written in assembly language for one computer cannot be used in any other computer.

High level language:

- A programmer can formulate problems more efficiently in a high level language. Besides, he need not have a precise knowledge of the architecture of the computer he is using.
- The instructions written in a high level language are called statements. The statements resemble move closely English and mathematics as compared to mnemonics in assembly languages.

- Examples of high level language are BASIC, PASCAL, COBOL, ALGOL etc.

Advantages of high level language:

1. **Machine independent:** High level languages are machine independent. That means a computer change doesn't require rewriting a program.
2. **Portability:** High level language programs are independent of computer architecture. The same program will run on any other computer which has a compiler for that language.

Problem Oriented Language

- A problem oriented language is a computer language designed to handle a particular class of problems. For example COBOL was designed for business applications; FORTRAN was designed for scientific and mathematical problems.

Interpreter vs. Compiler:-

1. An interpreter is program which translates one statement of program at a time. On the other hand a compiler goes through the entire high- level language program once then translates the entire program in to machine codes.
2. A compiler is 5 to 15 times faster than an interpreter.
3. An interpreter is small program as compared to a compiler. It occupies less memory space, so it can be used in small system which has limited memory space.
4. The object code produced by the compile is permanently saved for future reference. On the other hand the object code of the statements produced by an interpreter is not saved.

Why C is called robust language?

C is called robust language because its rich set of built in functions and operators can be used to write any complex program.

Why C is called portable language?

C program written for one computer can be run on another computer with or without modification, so c is called portable language.

Why C is called structured programming language?

We can write a C program in terms of function modules or blocks, so C is called structure programming language.

Why C is called free form language?

C is called free form language because it is not necessary to write each new statement in new line. We can write a multiple statement in single line.

What is Variable?

It is a symbolic name for a memory location in which the value is stored.

What is program?

A program is a finite set of instructions given to a computer for performing a predefined task.

What is programming?

The process of writing step by step instructions using a chosen language is known as programming.

What is translator? What is translation?

A software program which translates instructions written in any programming language to the language understandable by computer is called translator. The process of converting one language to another language is known as translation.

What is compiler?

Compiler is one system software which converts a whole program from high level language to machine level language and then executes a program.

What is interpreter?

An interpreter converts a high level language to machine level language line by line.

Source code / Program:-

The program that we write in any computer language using an editor is known as source code / program.

Object code/ Program:-

A machine language program, translated from a source program by translator is known as object code/program.

Executable code / Program:-

A program generated from the object code after linking with the library functions with the use of linker is called an executable code / program.

History of C: -

- The root of all modern language is ALGOL. ALGOL is introduced by international group in 1960. ALGOL was the first computer language to use a block structure.
- In 1967 Martin Richard developed a language called BCPL.
- In 1970 Ken Thompson created a language using BCPL and called it simply B.
- C was developed from ALGOL, BCPL and B by Dennis Ritchie at Bell Laboratory in 1972.
- C uses many concepts from these languages.
- An also use the many concept from these language and added the concept of data type and other powerful features.
- Then after in 1978 Kernighan and Ritchie developed a new language K&R C.
- In 1989 the ANSI-C was recognize by ANSI committee.
- The in the year 1990 the ISO committee standardise the language is known as ANSI/ISO C.

Year	Language Name	Developed By
1960	ALGOL	International C
1967	BCPL	Martin Richard
1970	B	Ken Thompson
1972	Traditional C	Dennis Ritchie
1978	K & R C	Kernighan and Ritchie
1989	ANSI C	ANSI Committee
1990	ANSI/ISO C	ISO Committee



Important of C:

- Programs written in C are efficient and fast.
- C is efficient because C has much different type of data type and powerful operators.
- C is many times faster than other language.
- There are only 32 keyword and its strength lies in its built in functions.
- C is highly portable. This means that C programs written for one computer can be run on another with little or no modification.
- C language is block structure programming language. So user can create a block or function module.
- Another important feature of C is its ability to extend itself.
- We can add our own functions to C library.
- **C provides a lot of inbuilt functions** that make the development fast.

Basic Structure of C Programs:

- **Documentation Section:** - The documentation section consists of a set of comment line. In this section we can provide the name of program, date of program, author and other general information regarding to program.
- **Link Section:** - The link section provides instruction to the compiler to link function from the header file.
- **Definition Section:** - The definition section defines the symbolic constant.
- **Global Declaration Section:** - There are some variables that are used in more than one function. That kind of variable are called global variables. The Global variables are declared in global declaration section. That is outside of all the function. This section is also known as **user-defined** section.
- **Main Function:** - Every C program must have one main() function. This section is consisting of two parts. Declaration part and Executable part. In declaration part we must declared all variable which are used in a program. There is at least one executable statement in executable part. These two parts must write between the opening and closing brace.
- **Subprogram Section:** - The subprogram section contains all the user-defined function. The user define function are generally placed immediately after the main function.

Documentation Section
Link Section
Definition Section
Global Declaration Section
main() function section <pre> { Declaration Part Executable Part } </pre>
Subprogram Section (User Define Function)

Example of Basic C structure Program:

```

/**           //Documentation
 * file: age.c
 * author: Ami Patel
 * description: program to find our age.
 */

#include <stdio.h>    //Link

#define BORN 2000    //Definition

int age(int current); //Global Declaration

int main(void)       //Main() Function
{
    int current = 2021;
    printf("Age: %d", age(current));
    return 0;
}

int age(int current) { //Subprograms
    return current - BORN;
}

```

Output:

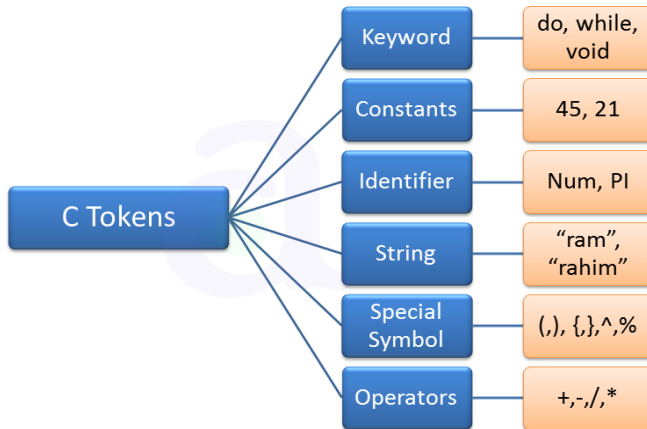
Age: 21

Just Remember:

- Every C program must have a main() function.
- Use of more than one main() is illegal.
- Execution of the program is start from main().
- The execution of function starts from opening brace and ends at closing brace.
- C programs are written in lowercase. Uppercase may be used for identifier, variable name or output string.
- Every C program statement in a C language must end with a semicolon (;).
- All variables must be declared their types before they are used in program.
- We must include header files using #include directive.
- STDIO.H is short name of **Standard Input & Output** Header file while the CONIO.H stands for **Console Input & Output**. Header file.
- STDIO.H includes basic functions like printf() (to print on screen), scanf() (used to take input from user) etc while CONIO.H includes functions like clrscr() (Clears the Screen), getch() (it holds the screen until and input from keyboard is received).
- **file should always be saved with .c extension**

CHAPTER 2

C TOKENS



- In a C program the smallest individual units are known as a C Tokens.
 - C has six types of tokens as shows as under.
1. **Keywords.** Ex.- float, int, for, while, if, else, switch, struct, union, break, etc..
 2. **Constants.** Ex.- -15.98, 30, 100, etc..
 3. **Identifiers.** Ex.- main, amount, etc..
 4. **String.** Ex.- "ABC", "pqr", "Year", "Amount", etc..
 5. **Special Symbols.** Ex.- [], { }, (), etc..
 6. **Operators.** Ex.- +, *, -, /, <, >, =, etc..

Keywords:

- Every C word is classified as a keyword.
- All keyword have fixed meanings.
- These meaning cannot be changed.
- All keywords must be written in lowercase. Since C is case-sensitive.
- There are a total of 32 keywords supported by the C language.

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

Example Keywords: -

```
int num = 10; //int keyword
```

```
char firm[10] = "Man"; //char keyword
```

These two lines can be modified as: (without knowing the data-type)

```
auto num = 10; //auto keyword is used to deduce the data type of a variable
```

```
auto firm = "Man";
```

Identifiers:

- Identifier refers to the name of variables, functions and array name, structure name.
- These names are defined by user, so they are known as user-define names.
- Identifier consists of letters and digits.
- Both uppercase and lowercase are permitted.

Rules of Identifiers:

1. First character must be an alphabet or underscore.

2. Must consist of only letters, digits and underscore.
3. Only 31 characters are significant.
4. We can not use keyword as identifiers.
5. White space are not allowed in identifiers.

Some Valid Identifiers:

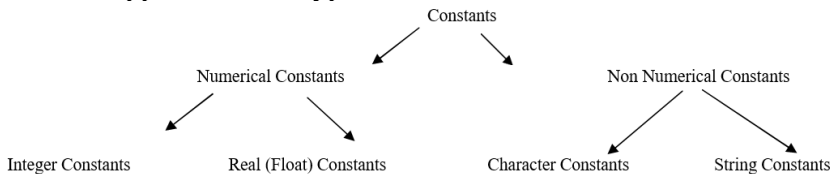
scaler, _scaler, scaler123, scaler_123, count1_, Double

Some invalid Identifiers:

100scaler *//started with a numerical digit*
 _hello,morning *//can't use comma operator*
 int *//keyword*
 float *//keyword*
 Idar(100) *//circular brackets can't be used*

Constants:

- Constants refer as fixed value in C that does not change during the execution of a program.
- C supports several types of constants.



- Integer Constants refer to a sequence of digit like 1056.
- Real (Float) constants refer a fractional (point) part like 17.76. Such numbers are called real or floating point constants.
- A single character constant contains only a single character. They are enclosed within single quotemark. Ex. – ‘g’, ‘K’, ‘l’ etc...
- A string constant is sequence of character enclosed within double quote mark. Ex. – “Well Done”, “Hello Word”, etc...

Variables:

- A variable is a data name that may be used to store a data value.

- Variables may take different values at different time during execution.
- A variable name can be chosen by the programmer.
- Ex.- Average, height, Total, Counter_1, class_strength etc....

Syntax: -

data_type variable_name = value; // defining single variable
or
data_type variable_name1, variable_name2; // defining multiple variable

- **data_type:** Type of data that a variable can store.
- **variable_name:** Name of the variable given by the user.
- **value:** value assigned to the variable by the user.

Rules For Variable:

1. They must begin with letter.
 2. Variable of 31 characters are valid.
 3. Uppercase and lowercase are significant.
 4. It should not be a keyword.
 5. White space is not allowed.
- Some valid example of variable.
 Vanita, Value, T_raise, Idar, x1, ph_value, mark, sum2, sum_5 etc...

Data Types: -

- C language is very rich in data types.
- The varieties of data types are available in ANSI – C.
- C supports three type of data types,
 1. Primary (Fundamental or Basic) data type.
 2. Derived data type.
 3. User-define data type.
- All C compiler support five type of fundamental data types.
- Integer (int), character (char), floating point (float), double floating point (double) (long double) and null (void).
- Integers are whole numbers with the range of values and define by keyword int.
- Floating point numbers are defining in C by the keyword float, double and long double.

- A single character can be defined as a char data type.
- The void type has no values. This is usually used to specify the type of function.
- The range of data types is given bellow.

Data Type	Size (in Byte)	Range
char	1 byte	-127 to 127
unsigned char	1 byte	0 to 255
int	2 byte	-32,768 to 32,767
unsigned int	2 byte	0 to 65535
float	4 byte	3.4E -38 to 3.4E +38
double	8 byte	1.7E -308 to 1.7E +308
long double	10 byte	3.4E -4932 to 1.1E +4932
void	Null	Null

CHAPTER 3

OPERATORS

- C supports several kinds of operators.
- An operator is a symbol that tell to computer that to perform mathematical or logical operation.
- An operator is used in program to perform the operation on variable or data values.
- C operator can be classified as under.
 1. Arithmetic operator
 2. Relational operator
 3. Logical operator
 4. Assignment operator
 5. Increment and Decrement operator
 6. Conditional operator
 7. Special operator
 8. Bitwise operator

Arithmetic operator:

- C provides all the basic arithmetic operators.
- The operators +, -, *, /, % all work the same way as they do in other language.
- These can operate on any basic data type.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Division

- For example, here **a** and **b** are variable and known as a operand.
- $a + b$: variable a is added to b.
- $a - b$: variable a is subtract from b.
- $a * b$: variable a is multiply by b.
- a / b : variable a is divided by b and integer division truncate the fractional part.
- $a \% b$: variable a is divided by b and produce the remainder of an integer division.

Note: - Modulo division (%) operator can not be used with floating point data.

Relational operator:

- C provides all the basic relational operators.
- These operators are used to compare two more values with each other to get relation.
- These kinds of comparisons can be done with the help of relational operators.
- The value of expression is either one or zero.
- The value is one if the relational expression is true.
- The value is zero if the relational expression is false.
- C supports six relational operators.
- These operators and meaning are shown as under.

Relational operator	Meaning
<	Is less then
<=	Is less then or equal to
>	Is greater then
>=	Is greater then or equal to
==	Is equal to
!=	Is not equal to

- For example here a and b are integer type of variables. $a < b$
 $a <= b$ $b >= a$
 $b == a$ $a != b$
- Among the six relational operators, each one is complement of another operator.

>	is complement of	<=
<	is complement of	>=
==	is complement of	!=
- We can simplify an expression the not and the less then operators using complement as shown as:

!(x < y)	x >= y
!(x > y)	x <= y
!(x != y)	x == y
!(x <= y)	x > y
!(x < y)	x >= y

Logical operator: -

- C has following logical operators.

Logical operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

- The logical operator `&&` and `||` are used when we want to test more than one condition and take decision. An example is : `a>b && x == 10`.
- This kind of expression, which is combination of two or more relational expression, is known as logical expression or compound expression.
- Like the simple relational expression, logical expressions also provide a value one for true expression and zero for false expression.
- Some examples of logical expression.
 1. `if (age > 55 && salary < 1000)`
 2. `if (number < 0 || number >= 1000)`

Assignment operator:

- Assignment operators are used to assign the result of an expression.
- Also C has short hand assignment operators.
- Syntax is: `variable op = expression`, where `op` is a operators.
- The assignment statement

`V op = exp;` Is

equal to

`V = v op (exp);`

Ex:-

`X += y+1;`

Is equal to

`X = x+(y+1);`

- Some of common shorthand assignment operators are shown as under.

Statement with simple assignment operator	Statement with shorthand operator
<code>a=a+1</code>	<code>a+=1</code>
<code>a=a-1</code>	<code>a-=1</code>
<code>a=a*(n+1)</code>	<code>a*=(n+1)</code>
<code>a=a/(n+1)</code>	<code>a/=(n+1)</code>
<code>a=a%b</code>	<code>a%=b</code>

- The use of shorthand assignment operator has three advantages.

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is easier to read.
3. The statement is more efficient.

Increment and Decrement operator:

- C has two very useful operators. These two operators not found in other language.
- These operators are Increment (++) and Decrement (--) operators.
- The operator ++ adds 1 to operand.
- The operator -- subtracts 1 from operand.
- For example:

M++;	++M;
M--;	--M;
- We use the increment and decrement operator in for loop, while loop and do while loop.
- There are two types of increment and decrement operators. Prefix and postfix operators.
- A prefix operator first adds 1 to the operand and then result is assigned to the variable.
- A postfix operator first assigns the value to the variable and then increments the operand.

Conditional (Ternary) operator:

- Conditional operator is also known as a ternary operator.
- A conditional operator pair “ ? : “ is available in C.
- A conditional operator is used to create conditional expression.
- Syntax is :

exp1 ?	exp2 :	exp3;
--------	--------	-------
- In this syntax exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of expression.
- If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.
- Note that only one of the expressions either exp2 or exp3 is evaluated.
- For example:

```
int a=10;int
b=15;int x;
x = (a<b) ? a : b ;
```

In this example, x will be assigning the value of b.

Precedence of arithmetic operators:

- An arithmetic expression without parentheses will be evaluated from left to right using the rules of precedence of operator.
- There are two levels of arithmetic operators in C.
- High priority : * / %
- Low priority : + -
- Consider the following statement that has been used.

$$X = a - b/3 + c*2 - 1$$

When a=9, b=12, and c=3 then the statement become

$$X = 9 - 12/3 + 3*2 - 1$$

First pass:

$$\text{Step-1: } x = 9 - 4 + 3*2 - 1 \quad \text{Step-2: } x = 9 - 4 + 6 - 1$$

Second Pass:

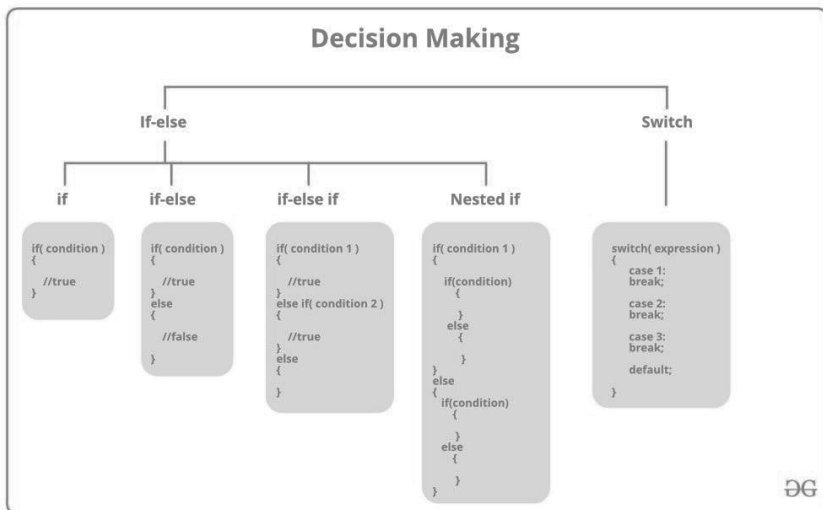
$$\text{Step-3: } x = 5 + 6 - 1$$

$$\text{Step-4: } x = 11 - 1 \quad \text{Step-5: } x = 10$$

CHAPTER 4

DECISION MAKING AND BRANCHING

- C language has some decision-making capabilities by supporting of the following statements:
 1. Simple if statement.
 2. if.....else statement.
 3. Nested if.....else statement.
 4. else if ladder.
 5. Switch statement.
 6. goto statement.

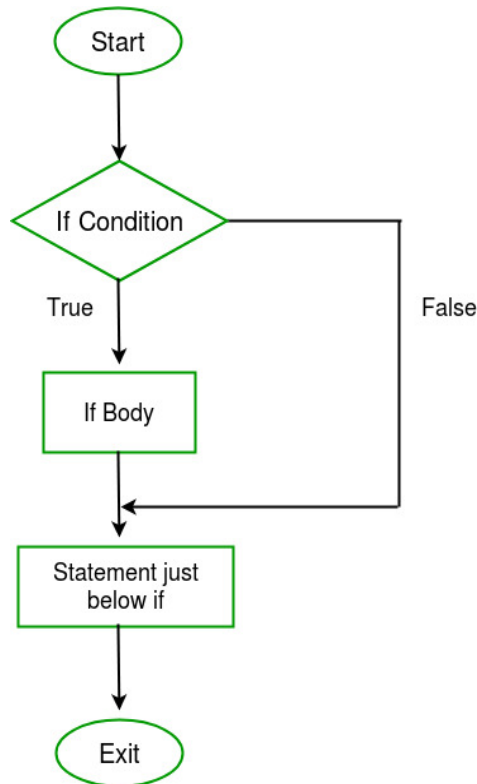


Simple IF statement: -

- The if statement is powerful decision-making statement.
- The if statement is used to control the flow of execution of statement.
- It is two-way decision statements and is used to conjunction with expression.
- General form of if statement is:

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```


- Here, the condition after evaluation will be either true or false.
- C if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not.
- If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

Flowchart: -

Example:

```
// C program to illustrate If statement
#include <stdio.h>

int main()
{
    int i = 10;

    if (i > 15) {
        printf("10 is greater than 15");
    }

    printf("I am Not in if");
}
```

Output:

I am Not in if

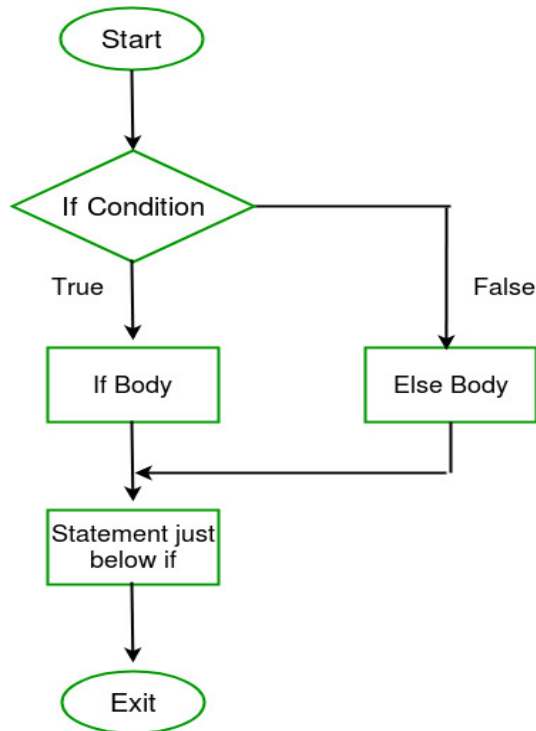
- As the condition present in the if statement is false. So, the block below the if statement is not executed.

if.....else statement:

- The if.....else statement is an extension of simple if statement.
- The general form of if.....else statement is as under.

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

- If the condition is true, then the true block statement is executed.
- If the condition is false, then the false block statement is executed.
- In either case, either true block or false block will be executed, not both.
- In both cases, the control is transferred subsequently to statement-x.

Flowchart: -**Example:**

```
// C program to illustrate If statement
#include <stdio.h>

int main()
{
    int i = 20;

    if (i < 15) {
        printf("i is smaller than 15");
    }
    else {
        printf("i is greater than 15");
    }
    return 0;
}
```

Output:

i is greater than 15

- The block of code following the *else* statement is executed as the condition present in the *if* statement is false.

Nesting of if....else statement: -

- When a series of decision are involved, we may have to use more than one if....else statement innested from as shown below.

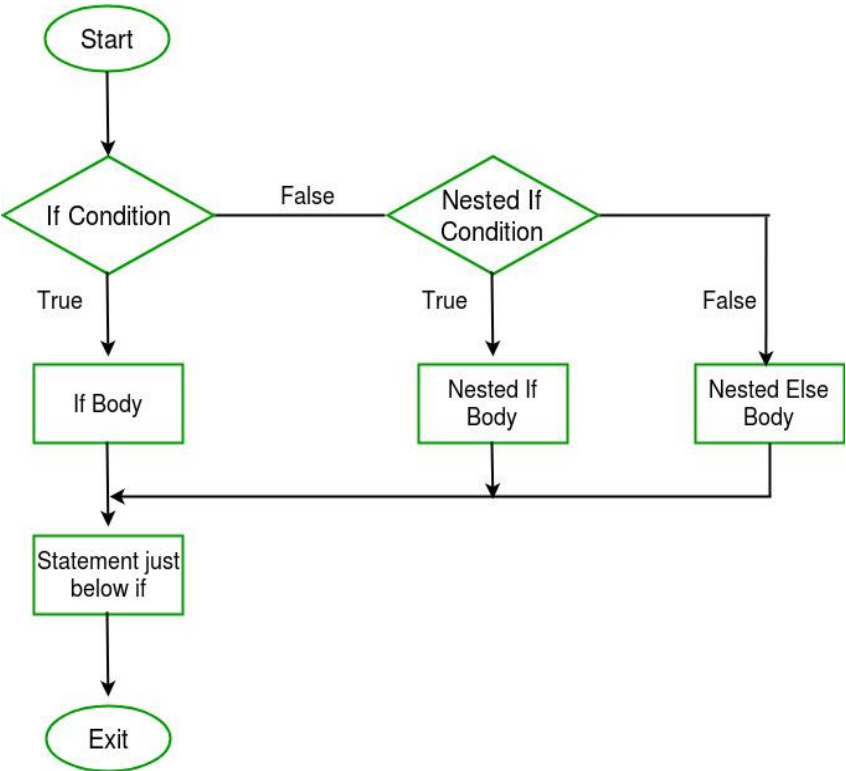
```

if(test condition-1)
{
    if(test condition-2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
statement-x;

```

- If the condition-1 is true then test condition-2 is executed.
- If condition-2 is true then statement-1 is executed.
- If condition-2 is false then statement-2 is executed.
- But if the condition-1 is false then control is transfer to the statement-3. In this case statement-1 and statement-2 is not executed.
- In both case control is transfer to the statement-x.

Flowchart:



Example: -

```
// C program to illustrate nested-if statement
#include <stdio.h>

int main()
{
    int i = 10;

    if (i == 10) {
        // First if statement
        if (i < 15)
            printf("i is smaller than 15\n");

        // Nested - if statement
        // Will only be executed if statement above
        // is true
        if (i < 12)
            printf("i is smaller than 12 too\n");
        else
            printf("i is greater than 15");
    }

    return 0;
}
```

Output:

```
i is smaller than 15
i is smaller than 12 too
```

The else if ladder:

- There is another way of putting ifs together when multi path decisions are involved.
- A multi path decision is a chain of ifs in which the statement associated with each else is an if.
- General form of else if ladder is:

```

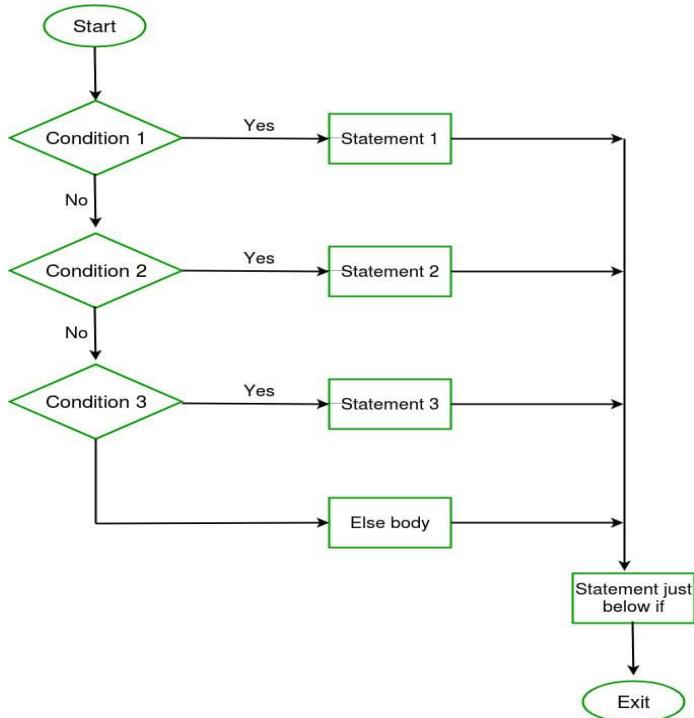
if (condition-1)
    statement-1;else
if (condition-2)
    statement-2;
else if (condition-3)
    statement-3;
else if (condition-n)
    statement-n;else
    default-statement;

statement-x;

```

- This construct is known as the else if ladder.
- The conditions are evaluated from the top (of ladder) downwards.
- As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x.
- When all the conditions become false, then the final else containing the default-statement will be executed.

Flowchart:



Example: -

```
// C program to illustrate else if ladder statement
#include <stdio.h>

int main()
{
    int i = 20;

    if (i == 10)
        printf("i is 10");
    else if (i == 15)
        printf("i is 15");
    else if (i == 20)
        printf("i is 20");
    else
        printf("i is not present");
}
```

Output: -

i is 20

The switch statement: -

- The switch statement is another decision making and branching statement.
- We have seen that when one of many alternatives is to be selected, we can use an if statement to control the selection.
- When the numbers of alternatives are increased at that time we have to use switch statement.
- C has built-in multi way decision statement known as a switch.
- The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statement associated with that case is executed.
- The general form of the switch statement is shown below:


```

switch (expression)
{
    case value-1:
        statement-1;
        break;

    case value-2:
        statement-2;
        break;

    case value-3:
        statement-3;
        break;

    .....
    .....
    default :
        default statement;
        break;
}
statement-x;

```

- The switch expression must be either integer or character type.
- Case labels must be integer or character.
- Case labels must be unique. No two labels can have the same values.
- Case labels must end with semicolon.
- The break statement transfers the control out of the switch statement.
- The break statement is optional. That is, two or more case labels may belong to the same statement.
- The default label is optional. If present, it will be executed when the expression does not find a matching case label.
- There can be at most one default label.
- The default may be placed anywhere but usually placed at the end.
- It is permitted to nest switch statements.

The goto statement: -

- The goto requires a label in order to identify the place where the branch is to be made.

- A label is valid variable name, and must be followed by colon.
- The label is placed immediately before the statement where the control is to be transfer.
- The general form of goto statement is as under:

```
Goto label;
.....
.....
.....
Label:
Statement;
```

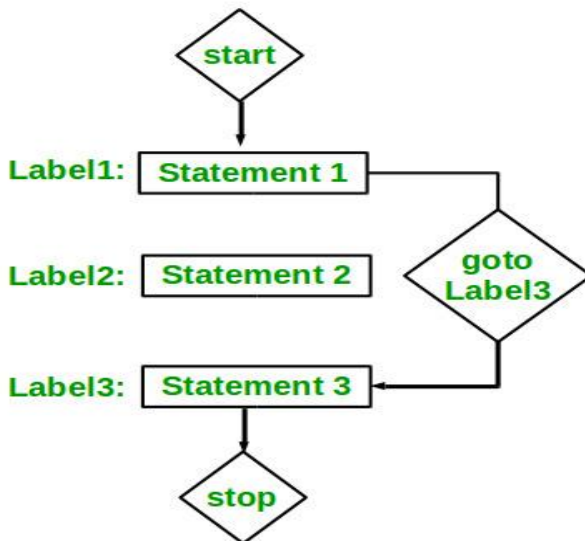
Forward Jump

```
label:
statement;
.....
.....
.....
goto label;
```

Backward Jump

- The label can be anywhere in the program either before or after the goto label.
- If the label: is before the statement goto label; will be known as backward jump.
- If the label: is after the statement goto label; will be known as forward jump.

Flowchart: -



Example: -

```

/ C program to print numbers
// from 1 to 10 using goto
// statement
#include <stdio.h>

// function to print numbers from 1 to 10
void printNumbers()
{
    int n = 1;
label:
    printf("%d ", n);
    n++;
    if (n <= 10)
        goto label;
}

// Driver program to test above function
int main()
{
    printNumbers();
    return 0;
}

```

Output:

1 2 3 4 5 6 7 8 9 10

The ? : operator: -

- Conditional operator is also known as a ternary operator.
- A conditional operator pair “ ? : “ is available in C.
- A conditional operator is used to create conditional expression.
- Syntax is :

$$\text{exp1 ? exp2 : exp3;}$$
- In this syntax exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and become the value of expression.
- If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.
- Note that only one of the expressions either exp2 or exp3 is evaluated.
- For example:

```
int a=10;int  
b=15;int x;
```

```
x = (a<b) ? a : b ;
```

- In this example, x will be assigning the value of b.

CHAPTER 5

DECISION MAKING AND LOOPING

- Depending on the position of the control statement in the loop, a control structure may be classified either as the entry-control loop or as the exit-control loop.
- In entry control loop, the control conditions are tested before the start of the loop execution.
- If the conditions are not satisfied, then the body of the loop will not be executed.
- In the case of exit control loop, the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time.
- The entry control and exit control loops are also known as pre tested and post tested loops.
- There are three type of loops available in C.
- For loop, while loop and do..while loop.

The while statement: -

- The simplest of all the looping structure in C is the while loop.
- The basic format of the while statement is:

```
while (test condition)
{
    body of the loop;
}
statement-x;
```

- The while is an entry controlled loop statement.
- The test-condition is evaluated and if the condition is true, then the body of the loop is executed.
- After execution of the body, the test-condition is once again evaluated and if it is true, the body is executed once again.
- The process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out of the loop.

- On exit, the program continues with the statements immediately after the body of the loop.
- Let us consider the following example.

```

sum =0;
n=1;
while(n<=10)
{
    sum = sum + n * n ;n = n
    + 1;
}
printf("sum = %d", sum);
.....

```

The do...while statement: -

- On some time it is required to execute the body of the loop before the test condition is checked.
- This kind of situation can be handle with help of the do while statement.
- The do while statement is also known as exit control or post tested loop.
- General form of do while statement is :do

```

{
    body of the loop;
}

while (test condition);
statement-x;

```

- On reaching the do statement, the program proceeds to evaluate the body of the loop first.
- At the end of loop, the test condition in the while statement is executed.
- If the condition is true then control is further transfer to body of the loop.
- But if the condition is false control is transfer to statement-x.
- The do while loop create an exit control loop and therefore body of the loop is always executed atleast once.
- A simple example of do while loop is :

```
do{
    printf("input a number");
    number = number + 1 ;

} while (number>0 && number < 100);
```

The for statement: -

- The for loop is another entry control or pre tested loop.
- The general form of for loop is:

```
for (initialization ; test condition ; increment or decrement)
{
    body of the loop;
}
statement-x;
```

- There are three part in a for loop.
- Initialization, test condition and increment or decrement.
- Initialization done first. Initialization portion is executed only one time.
- Then control is transfer to test condition. If the condition becomes true then control is transfer to body of the loop.
- After the execution of body of the loop control is goes to increment or decrement portion.
- After the execution of increment or decrement part control is goes to test condition.
- If condition becomes false then control is transfer to statement-x.
- Consider the following example:

```
for (x=0; x<=10 ; x++)
{
    printf("%d", x);
}
printf("\n");
```

Nesting of for loop: -

- Nesting of for loops, that is, one for statement within another for statement.
- For example consider the following :

```

for(row = 1; row <= romax; row++)
{
    for (column =1; column <= colmax; ++column)
    {
        y = row * column;
        printf(“%d”, y);
    }
    printf(“\n”);
}

```

- The nesting may continue up to any desired level.

Give difference: -

The for loop	The while loop
The for' statement is usually used when the statements are to be executed repeatedly for a fixed number of times.	When number of iterations cannot be pre – determined, while loop is more suitable.
Loop terminating condition is tested after entering the loop	Loop terminating condition is tested before entering the loop.
Counter variable is initialized inside the loop	Counter variable is initialized before the loop

Give difference between while loop and do....while loop.

The While loop	The do...While loop
The condition is checked before executing the body of the loop	The value of the test condition is checked after executing the body of the loop
This is an entry – controlled loop	This is an exit – controlled loop.
Body of the while loop may not be execute at all	Body of the do...while loop executed at least once even if condition is false very first time.

Give difference between break and continue statement.

Break	Continue
The break statement breaks the loop and stops further iterations.	The continue statement breaks only current iteration and then continues with the next iteration.
The break statement terminates the loop itself	The continue statement continues with next iteration

Jumping in loops: -

- Loops perform the set of operations repeatedly until the control variable fails to satisfy the test condition.
- But sometimes, when executing a loop it becomes desirable to skip a part of loop or to leave the loop as soon as a certain condition is occurs.
- At that time we have to use a break statement. When the break statement is encountered inside the loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

Skipping a part of loop:-

- During the loop operation, it may necessary to skip a part of the body of the loop under certain conditions.
- Like the break statement, C supports another similar statement called continue statement.
- The continue statement tells the compiler that “SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT STATEMENT.”
- We can use the continue statement with for loop, while loop, do while loop, and goto statement.
- For example. :-

```

while(test condition)
{
    .....
    if( condition)
        continue;
    .....
}

```

CHAPTER 6

ARRAYS

- Definition: - “Array is sequence collection of related data item that share same name and same data type, but store a different value.”
- C supports a derived data type known as array that can be used for many applications.
- Three type of array is available in C :
 1. One dimensional array.
 2. Two dimensional arrays.
 3. Multidimensional arrays.

One dimensional array: -

- Definition: - “Array is sequence collection of related data item that share same name and same data type, but store a different value.”
- A list of items can be given one variable name using only one subscript and such a variable is called a one-dimensional array.
- Like any other variable, arrays must be declared before they are used.
- The general form of array declaration is: data-type

variable-name [size];

- The data-type specifies the data type of array.
- Variable is a name of variable which is declared by user.
- Size indicates the maximum number of elements that can be stored inside the array.
- For example, if we want to represent a set of five number, say (35,40,20,57,19), by an array variable **number**, then we may declare the variable **number** as follows:

int number[5];

- The computer provides five storage locations as shown below:

	number[0]
	number[1]
	number[2]
	number[3]
	number[4]

- The values to the array elements can be assigned as follows:

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

- This would cause the array **number** to store the values as shown below: number[0]

35	number[1]
40	number[2]
20	number[3]
57	number[4]
19	

- These elements may be used in programs just like any other C variable.
- For example, the following are valid statements:

```
a = number[0] + 10;
number[4] = number[0] + number [2];number[2]
= x[8] + y [10];
```

- When the compiler sees the character array sting, it terminates it with additional null character.
- The null character of the character is determined by '\0'.
- When declaring character array, we must allow one extra element space for the null terminator.

Initialisation of one dimensional array: -

- Initialisation means to store the values to an array element.
- An array can be initialized at two ways.

1. At compile time.
2. At run time.

Compile time initialisation: -

- We can initialise the elements of arrays in the same way as the simple variables when they are declared.
- The general form of initialisation of arrays is:

data-type array-name[size] = {list of values};

- The values in the list are separated by commas. For example
`int num[3] = { 10, 23, 43};`
- If the number of elements are greater than number of assigned value at that time only that elements will be initialised. The remaining elements will be initialised with zero.

For example `int num[5] = {2, 4, 5};`

2	num[0]
4	num[1]
5	num[2]
0	num[3]
0	num[4]

- In some case the size of array will be omitted.
- In such case, the compiler allocates enough space for all initialized elements. For example `int num[] = {10, 49, 54, 43};`
- In this statement the size of array will be four.
- Character array may be initialized in same way.
 For example `char name[] = {'d', 'o', 'c', 't', 'o', 'r', '\0'};`
`char city[] = "idar";`
- Last character can be initialised with NULL character '\0' .

Run time initialisation: -

- An array can be initialised at run time.
- Run time initialisation is used with large array.
- Consider the following example.

```

int i;
int num[50];
for( i = 0; i<=49; i++)
{
    scanf("%d", & num[i]);
}

```

- The 50 elements are initialised with values which are provided by user from keyboard at runtime.

Two dimensional arrays: -

- C allows us to define a tabular form by using two dimensional arrays.
- Two dimensional arrays can be declared as under. data_type array_name [row_size] [column_size];
- This is use the one pair of parentheses with commas to separate array size.
- C places each size in its own set of bracket.
- Each dimension of array is indexed from zero to its maximum size minus one.
- First index select the row and the second index selects the column within that row.

Initialisation of two dimension arrays: -

- Two dimension arrays can be initialised by two ways.
 1. At compile time.
 2. At run time.

Compile time initialisation:-

- The general form of initialisation of arrays is:

data-type array-name[row_size] [column_size] = {list of values};

- The values in the list are separated by commas and enclosed with braces.
- For example `int num [2][3] = {3, 5, 6, 10, 23, 43};`
- In this array first row is initialised with value 3, 5 and 6. And another row is stored with values with 10, 23 and 43.
- An above example also can be initialised as under.


```
int num [2] [3] = { { 3, 5, 6}, {10, 23, 43} };
```

- We can also initialize two dimensional array in the form of matrix as shown below:

```
int num [2] [3] = {
    {3, 5, 6},
    {10, 23, 43}
};
```
- We not need to specify the size of the first dimension. For example:

```
int num [ ] [3] = {
    {1, 2, 3},
    {4}
};
```
- An above example the size of array is declared automatically. And also the value 4 is initialised on first element and remaining elements are initialised with zero.

Run time initialisation: -

- An array can be initialised at run time.
- Run time initialisation is used with large array.
- Consider the following example.

```
int i, j;
int num [10] [10]; for( i = 0; i<=9; i++)
{
    for( j = 0; j<=9; j++)
    {
        scanf("%d", & num[i] [j]);
    }
}
```

- The 100 elements are initialised with values which are provided by user from keyboard at runtime.

Just remember: -

- When we declare an array we need to specify three things, data_type, array_name and size of anarray.
- Always remember that starting number of an array element is zero and last element number is always size minus one.
- Do not forget to initialize the elements. Otherwise compiler stores the “garbage” value.
- When initializing character array at that time we must provide enough space for null character ‘\0’.

CHAPTER 7

CHARACTER ARRAYS AND STRING

- A string is collection of characters.
- A string is treated like single data item.
- Group of character is defined under double quotation.
- For example: “well done” or “This is the example of ANSI-C”.
- Character strings are often used to build meaningful and readable programs.
- The common operations performed on character string include:
 1. Reading and writing string.
 2. Add string together.
 3. Copying one string another.
 4. Comparing string for equality.
 5. To get the size of string.
 6. Reverse the string.

Declaring and initializing string variable: -

- C does not support string as a data type.
- It allows us to represent string as character array.
- The general form of declaration of a string variable: `char string_name [size]`;
- The size determines the number of character in the string.
- Example : `char city [10]`;
`char name [20]`;
- When compiler assigns a character string to character array, it automatically assigns the null character ‘\0’ at the end of the string.
- Therefore, the size should be equal to the maximum number of character in the string plus one.
- It is possible that the size will be determines automatically, based on number of character in the array.
For example: - `char city[] = {‘a’, ‘h’, ‘m’, ‘e’, ‘d’, ‘a’, ‘b’, ‘a’, ‘d’, ‘\0’}`; `char city[] = {“ahmedabad”}`;

that is looks like

a	h	m	e	d	a	b	a	d	\0
---	---	---	---	---	---	---	---	---	----

String handling functions: -

- C supports a large number of string handling functions.
- Before using the string handling functions we must use the header file **#include <string.h>**.
- Following are the string handling functions.

String Handling Functions	Action
strcat()	Concatenates two strings.
Strcmp()	Compare two strings.
strcpy()	Copies one string over another.
strlen()	Finds the length of a string.
strrev()	Provide the reverse string.

strcat() function:

- The strcat() function joins two string together..

Syntax:

```
char *strcat(char *str1, char *str2)
```

- The dest_string and source_string are character arrays.
- When the function strcat() is executed, source_string is added to dest_string.
- It does so by removing the null character at the end of the dest_string and placing source_string from there.
- The string source_string remains unchanged.

Example:

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

int main() {

    char string1[10] = "Hello";
    char string2[10] = "World";
    strcat(string1, string2);
    printf("Output string after concatenation: %s", string1);

}
```


Output:

Output string after concatenation: HelloWorld

strcmp() function: -

- The strcmp() function is use to compare the two string.

Syntax:

```
int strcmp(const char *str1, const char *str2, size_t n)
```

- If both strings are same then function returns zero value, otherwise function returns nonzero value.
- The dest_string and source_string are character arrays.
- When the function strcmp() is executed, source_string is compare with dest_string.

Example:-

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

int main() {

    char s1[20] = "ScalerAcademy"; // string1
    char s2[20] = "ScalerAcademy.COM"; // string2
    // comparing both the strings
    if (strcmp(s1, s2) == 0) {
        printf("string 1 and string 2 are equal");
    } else {
        printf("string 1 and 2 are different");
    }
}
```

Output:

string 1 and 2 are different

strcpy() function: -

- The strcpy() copy the one string array to another string array.

Syntax: -

```
char *strcpy( char *str1, char *str2)
```

- This function assigns the contents of source_string to dest_string.
- Both are the character array variable or a string constant.

Example:

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

int main() {

    char s1[35] = "string 1"; // string1
    char s2[35] = "I'll be copied to string 1."; // string2
    strcpy(s1, s2); // copying string2 to string1
    printf("String s1 is: %s", s1); // printing string1

}
```

Output:

String s1 is: I'll be copied to string 1.

strlen() function: -

- This function counts and returns the number of characters in a string.
- It takes following form:
n = strlen(string);
- Where n is an integer variable.
- This receives the value of the length of the string.
- The counting ends at the first null character.

Example:-

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

int main() {
```

```

char string1[20] = "ScalerAcademy";
printf("Length of string string1: %ld", strlen(string1));
return 0;
}

```

Output:-

Length of string string1: 13

strrev() function: -

- This function provides the reverse string of original string variable.
- It takes following form:
 strrev(string);
- This kind of function is used to find out palindrome string.

Just remember:

- Character must enclose with single quotes and string must enclose with double quotes.
- Allocate sufficient space in character array with null character (\0).
- Avoid processing single character as string.
- String cannot be used with operators. Always use the string function.
- Do not forget to append the null character to string array.
- The header file <string.h> is required when using input output functions.
- The header file <string.h> is required when using string handling function.