Chapter – 1

Constants , Variables & Keywords

Variables :-

A variable is a container which stores a value. In kitchen , we have containers storing Rice , Dal , Sugar , etc. Similar to those variables in C language. Stores a Constant / Value. A variable is an entity whose value can be changed. Variable is the name given to a memory location.

Example :-

```
int a ;  // Variable declaration

a = 23 ; // Variable Initialization

int a = 23; // Variable declaration and initialization together
```

Tip :-

We must create meaningful variables in our programs. This enhances the readability of our programs.

Types of Variables :-

1. Integer variable int a = 23;
2. Real / Decimal / Float / Double variable
   float b = 2.3;
3. Character variable char p = ‘R’ ;

Rules For Naming Variables in C :-

1. First character must be an alphabet or underscore (_).
2. No commas and spaces allowed.
3. No special symbols other than underscore (_) allowed.
4. Variable names are case sensitive.
5. It can contain alphabets and digits.

Constants :-

An entity whose value doesn’t change is called a Constant.

Types of Constants :-

Primarily , there are 3 types of Constants :-

1. Integer Constant :- 6 , 23 , -33 , -42 , etc.
2. Real Constant :- 123.6 , 51.9 , -33.3 , etc.
3. Character Constant :- ‘a’ , ‘R’ , ‘P’ , ‘$’ , ‘@’ , etc.

Note :- A character constant must be enclosed with single quotes.

Basic Building Blocks of a C Program :-

1. Keywords

These are reserved words , whose meaning is already known by compiler. These words cannot be used as variable names. There are 32 Keywords available in C language.

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

2. Identifiers :- Name of variables , functions or constants.

3. Stringliterals :- Example :- Hello World

4. Symbols :- Example :- %d , &a

5. Constant :- 0 (Zero)

Basic Structure of a C Program :-

All C Programs have to follow a basic structure. A C program starts with a ' main ' function and executes instructions present inside it. Each instruction is terminated with a semicolon (;). Instructions are case sensitive. Instructions are executed in the same order in which they are written. Each C program has a preprocessor command at the beginning of program.

Comments :-

Comments are used to clarify something about the program in plain language. It is a way for us to add notes to our program. Comments in C

program are not executed and ignored by compiler.

Types of Comments :-

There are 2 types of comments in C :-

1. Single Line Comment :-
   // This is a single line comment.
2. Multi Line Comment :-
   /* This is a
   multi line comment.*/

Compilation and Execution :-

A Compiler is a computer program which converts a programming language into a machine language so that it can be easily understood by the computer.

A C program is written in plain text. This plain text is combination of instructions in a particular sequence. The compiler performs some basic checks and finally converts the program into an executable file that is .exe file.

Format Specifiers :-

Format Specifier is a way to tell the compiler that what type of data is in a variable during taking input and displaying output to the user. There are so many types of format specifiers in C language. Here is the table which describes each format specifier perfectly and briefly. Before that some important notes and tips related to format specifiers :-

These are the basic format specifiers. We can add some other parts with the format specifiers. These are like below :–

- A minus symbol (-) sign tells left alignment
- A number after % specifies the minimum field width. If string is less than the width, it will be filled with spaces
- A period (.) is used to separate field width and precision.

Format Specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)
%i	Unsigned integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long

%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int / Pointer
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

Library Functions :-

C language has a lot of valuable library functions. We have to focus on some basic library functions which we are going to use often in our C programs. We will learn the library functions along with our new concepts. Right now , we will learn only one library function. This library function prints anything written inside it on output screen. That is :-

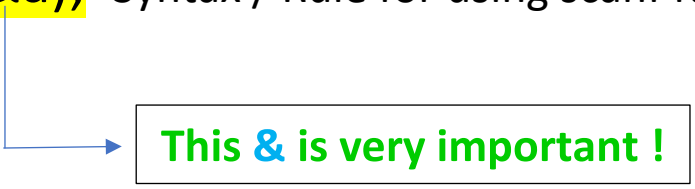
printf(“Prachi”); Syntax / Rule for using printf function

Receiving Input from User :-

Here , we are going to learn a new library function in C which helps us to take input from the user. In order to take input from the user and assign it to a variable , we use the following function :-

```
scanf(" %d " , &a);
```

Syntax / Rule for using scanf function



' %d ' is a format specifier as we already studied. It depends on which type of variable is ' a '. Format specifier and variable name may change according to situations but other syntax will be same in any of the situations.

& Is the “ address of ” operator and it means that the supplied value should be copied to the address which is indicated by variable ‘ a ’ .

Data Types :-

1. Basic :- int , char , float , double
2. Derived :- array , pointer , structure , union
3. Enumeration :- enum
4. Void :- void

Try it Yourself :-

Write a program to add 2 numbers.

Chapter – 1 Practice Set

1. Write a C program to calculate area of a rectangle.

A. Using hard coded inputs.

B. Using inputs given by user.

2. Write a C program to calculate the area of a circle and modify the same program to calculate the volume of a cylinder given its radius and height.

3. Write a C program to convert centigrade into Fahrenheit.

4. Write a C program to calculate Simple Interest for a set of values representing Principal , Number of Years and Rate of Interest.

5. Write a C program to calculate perimeter of a square.

A. Using hard coded inputs.

B. Using inputs given by user.

6. Write a C program to calculate square of any number entered by the user.

Chapter – 2
Instructions & Operators

Operators :-

An operator is a symbol used to perform operations in given programming language. Operators are one of the most important topic in C Programming.

Types of Operators in C :-

1. Arithmetic Operators

Operator	Use
+	Addition :- Adds the numbers
-	Subtraction :- Subtracts the numbers
*	Multiplication :- Multiply the numbers
/	Division :- Divide the numbers
%	Modulus :- Gives the remainder of any division , cannot be applied on float and sign is same as numerator.

2. Relational Operators

Operator	Use
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

3. Logical Operators

Operator	Use
&&	Both operands are non – zero then condition is true
||	One operand is non zero then condition is true
!	Reverses the correct State

4. Bitwise Operators

Operator	Use
&	Both operands are non – zero then condition is true
!	One operand is non - zero then condition is true
^	One operand is non – zero and one operand are zero then condition is true
~	Binary One's Complement
<<	Binary Left Shift
>>	Binary Right Shift

Binary One's Complement , Binary Left Shift and Binary Right Shift are not so important in C language. They come under Bornean Algebra. So , if you know first 3 operators , it is enough.

5.Assignment Operators

Operator	Use
=	Assign Value from Right to left
+=	Add Right to Left and assign result to Left
-=	Subtract Right to Left and assign result to Left
*=	Multiply Right to Left and assign result to Left
/=	Divide Left to Right and assign result to Left

6. Miscellaneous Operators

Operator	Use
sizeof()	Return size of variable
&	Return address of variable
*	Pointer to a variable
?:	Short hand ‘if - else’

Operator Precedence :-

Associativity :- In programming languages , the associativity of an operator is a property that determines how operators of same precedence are grouped in the absence of parentheses.

Category	Operators	Associativity
Postfix	() → _ ++ --	Left to Right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to Right
Additive	+ -	Left to Right
Shift	<< >>	Left to Right
Relational	< <= > >=	Left to Right
Equality	== !=	Left to Right
Bitwise AND	&	Left to Right
Bitwise XOR	^	Left to Right
Bitwise OR	|	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= > >= < <= &= ^= |=	Right to Left
Comma	,	Left to Right

Note :-

1. No operator is assumed to be present.

int R = pd → Invalid
int R = p*d → Valid

2. There is no operator to perform exponentiation in C. However , we can use pow (x , y) from <math.h> .

Type Conversion :-

An Arithmetic operation between :-

Int and Float	results	Float
Int and Int	results	Int
Float and Float	results	Float

Now we will learn types of instructions :-

Types of Instructions :-

1. Type declaration Instruction

```
int a ;
float b;
char R;
int a = 4;
float b = 2.3;
char R = 'P' ;
```

2. Arithmetic Instruction :-

```
int d = ( 3*2 ) + 1
```

We use arithmetic operators for writing arithmetic instructions.

3. Control Instruction :-
   A. Sequence Control Instruction
   B. Decision Control Instruction
   C. Loop Control Instruction
   D. Case Control Instruction

Chapter – 2 Practice Set

1. Which of the following is Invalid ?

A. int a , b = a;
B. int v = 3^3;
C. char dt = " 21 December 2020 "

2. What data type will 3.0/8-2 return ?

3. Write a C program to check whether the number is divisible by 97 or not.

4. Explain step by step evaluation of 3*x/y-z+k where x = 2 , y = 3 , z = 3 , k = 1.

5. 3.0 + 1 =
   A. Integer
   B. Float
   C. Character

Chapter – 3
Conditional Instructions

Conditional instructions are based on conditions. We give condition to the computer that if this condition is true then execute this code and so on.

Types of Conditional Instructions :-

1. Decision-making Instructions
2. Switch Case Instruction

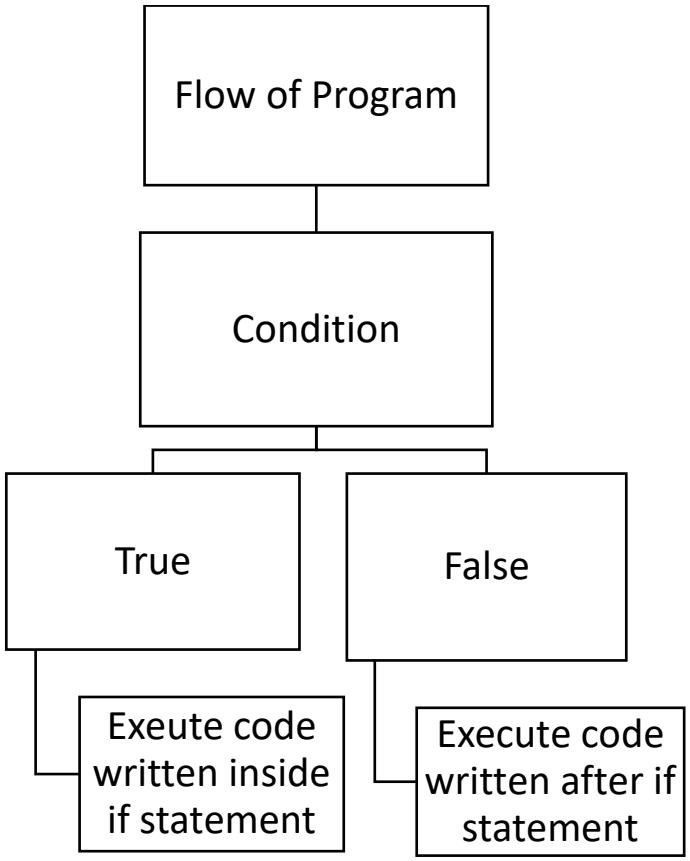
Types of Decision-making Instructions :-

A. ‘ if ’ statement
B. ‘ If else ’ statement
C. ‘ If-else-if ’ ladder
D. ‘ Nested if ’ statement

1.‘ if ’ statement

Syntax :- **if (condition) {true}**

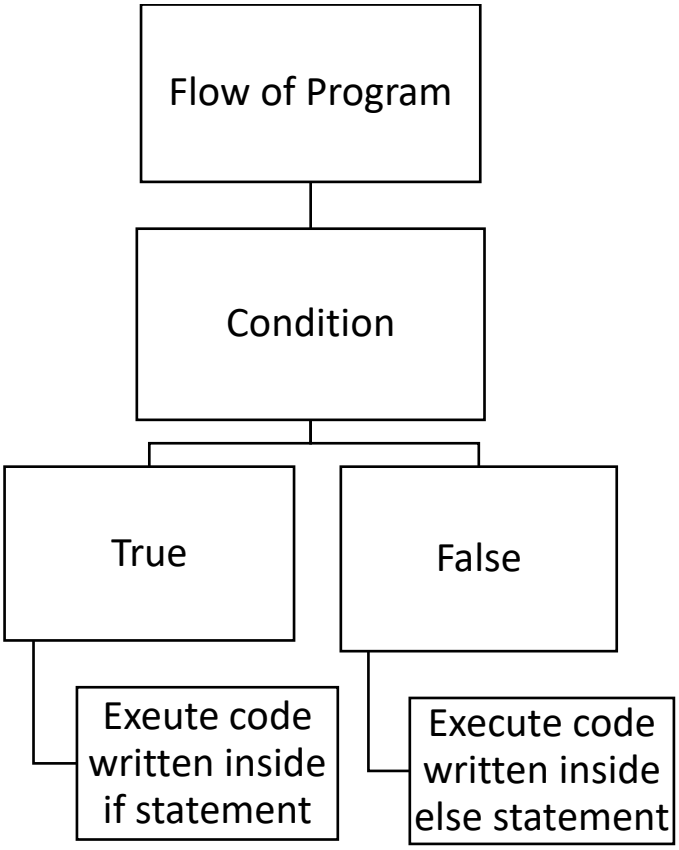
Flowchart :-



2.‘ If else ’ statement

Syntax :- **if (condition) {true} else {if false}**

Flowchart :-

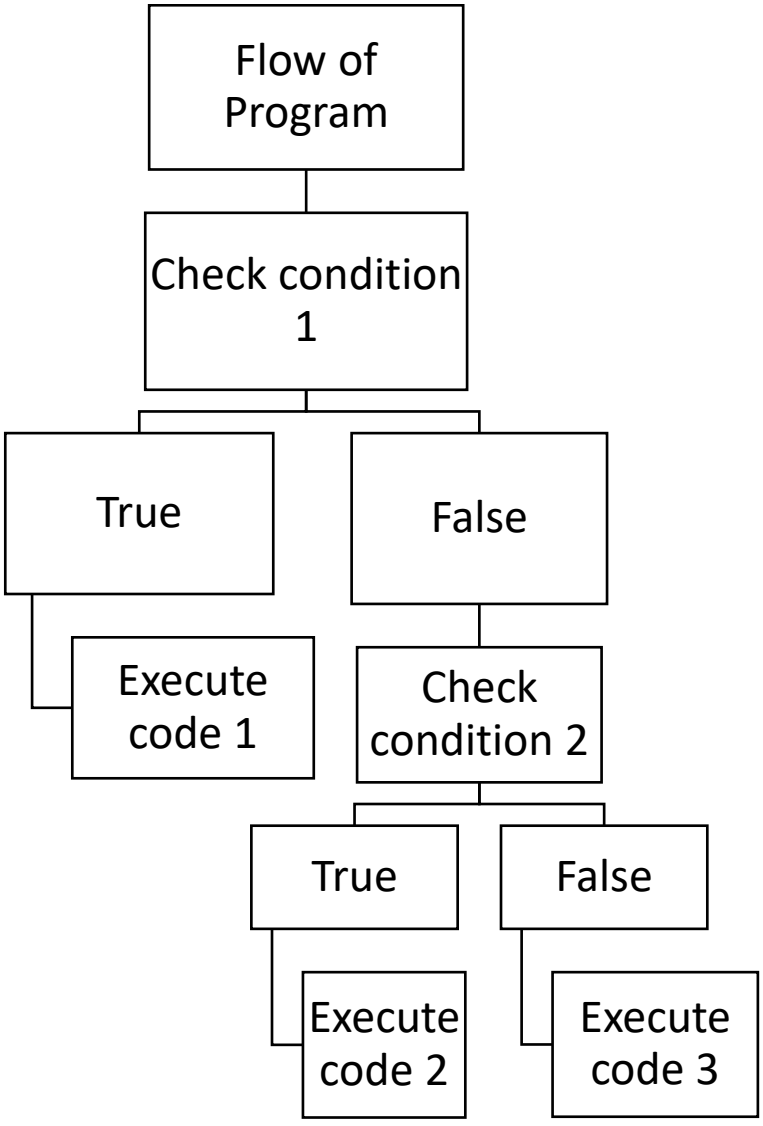


3.‘ If-else-if ’ ladder

Syntax :-

if (condition 1) {code 1} else if (condition 2) {code 2}.....else {code 3}

Flowchart :-



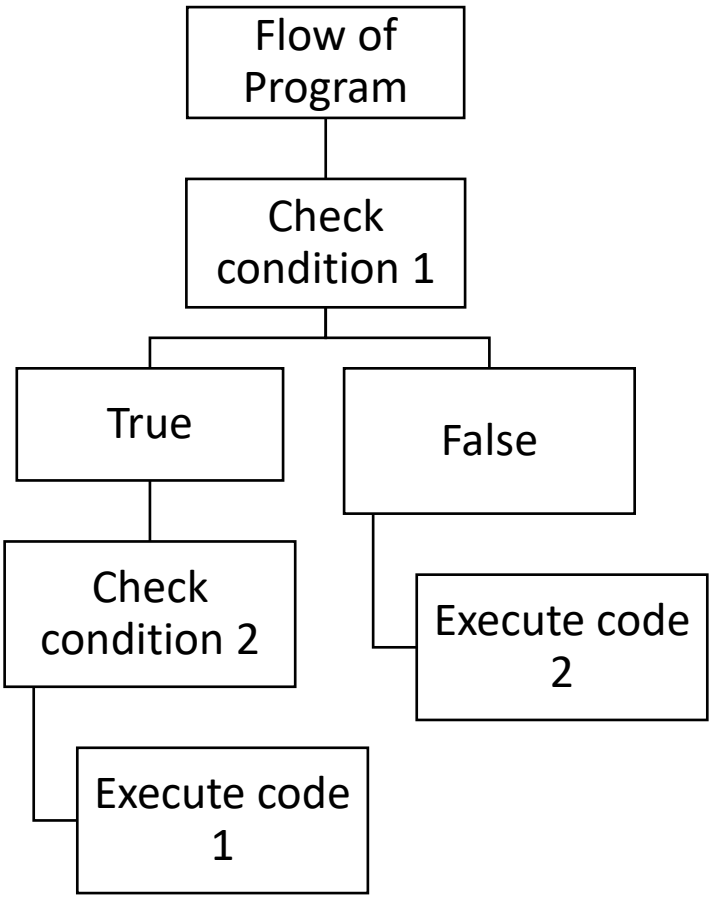
4.‘ Nested if ’ statement

Syntax :-

if (condition 1) { if (condition 2)

{code 1} else {code 2} }

Flowchart :-



Conditional Operators :-

A short hand ‘ if – else ’ can be written using the conditional or ternary operators.

Syntax :-

condition ? code if true : code if false

Switch Case Instruction :-

Switch case is used when we have to make a choice between number of alternatives for a given variable.

Syntax :-

switch (integer or character expression)

{ case c1:

code;

break;

case c2:

code;

break;

default :

code;

break; }

The value of integer or character expression is matched against any of the case ; all the code written after it are executed.

Rules for switch statement usage :-

1. Switch expression and value must be int or char.
2. Case must come inside switch.
3. break is not a must.
4. A switch can occur within another but in practice this is rarely done.
5. We can write switch case statements in any order of our choice. (Not necessarily ascending or descending)

Try it yourself :-

Write a C program to find grade of a student based on marks given below:-

90 to 100 → A

80 to 89 → B

70 to 79 → C

60 to 69 → D

50 to 59 → E

Less than 50 → Fail

Chapter – 3 Practice Set

1. What will be the output of this program ?

```
int age = 10;
if (age = 11) {
printf("I am 11");
}
else {
printf("I am not 11");
}
```

2. Write a C program to find out whether a student is pass of fail ; if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as input from user.

3. Write a C program to calculate income tax by taking income as input as per slabs mentioned below :-

   Below 2.5 lakh → No tax

   2.5 lakh to 5 lakh → 5%

   5 lakhs to 10 lakhs → 20%

   Above 10 lakhs → 50%

4. Write a C program to find greatest of four numbers entered by user.

5. Write a C program to create a simple calculator using switch case statement.

6. Write a C program to check whether a number is even or odd. Take number as input from user.

7. Write a C program to check whether a triangle is Equilateral , Isosceles or Scalene.

8. Write a C program to check whether the two integers entered by user are equal or not.

9. Write a C program to determine whether a character entered by user is lowercase or not.

10. Write a C program to check whether the triangle is possible or not with the help of sides. Take sides as input from user.

Chapter – 4

Loop Control Instructions

Sometimes , we want our program to execute set of instructions repeatedly.
Loops are used for doing this. It helps us in executing the same code repeatedly without writing it multiple times.

Types of Loops :-

1. While Loop

Syntax :- while (condition) {code}

Example :-

```
int d = 1;

while (d<11) {

printf (“%d\n” , d); d++;  }
```

Note :- While loop first checks the condition then executes the code.

Try It Yourself :-

Write a C program to print natural numbers from 10 to 20 when initial loop counter is initialized to 0. The loop counter need not to be int , it can be float as well !

2. Do While Loop

Syntax :- do {code} while (condition)

Example :-

```
int d = 1;
do {
printf (“%d\n” , d);
d++;
} while (d<11);
```

Note :- Do while loop first executes the code then checks the condition.

Try It Yourself :-

Write a C program to print first ‘n’ natural numbers using do while loop. Take ‘n’ as input from the user.

3.For Loop

Syntax :- for (initialize; test; increment or decrement) {code}

Example :-

```
for(i=1;i<3;i++) {
printf("%d\n", i);
}
```

Try It Yourself :-

Write a C program to print first 'n' natural numbers in reverse order.

Increment & Decrement Operators :-

d++ → 'd' incremented / increased by 1.

d-- → 'd' decremented / decreased by 1.

Break Statement :-

The break statement is used to exit the loop irrespective of whether the condition is true or false. Whenever a "break" is encountered inside the loop , the controls are sent outside the loop.

Continue Statement :-

The continue statement in c is used to immediately move to next iteration of the loop. The control is taken to next iteration , thus skipping everything below continue statement inside the loop for that iteration.

Chapter – 4 Practice Set

1. Write a program to print the multiplication table of a given number n.

2. Write a program to print a multiplication table of 10 in reversed order.

3. A do-while loop is executed:

   - At least once
   - At least twice
   - At most once

4. What can be done using one type of loop can also be done using the other two types of loops – True or False.

5. Write a program to sum the first ten natural numbers using a while loop.

6. Write a program to implement program 5 using for and do-while loop.

7. Write a program to calculate the sum of the numbers occurring in the multiplication table of 8. (Consider 8x1 to 8x10)

8. Write a program to calculate the square of a given number using for loop.

9. Repeat 8 using a while loop.

10. Repeat 7 using do while loop.

Project – 1 Number Guessing Game

Problem :-

Write a program that generates a random number and asks the player to guess it. If the player's guess is higher than the actual number than the program displays "Lower Number Please !". If the player's guess is lower than the actual number than the program displays "Higher Number Please !". When player guesses the correct number , the program displays the number of attempts.

Hint :-

Use loops and a random number generator.

Chapter – 5
Function & Recursion

Sometimes , our program gets bigger in size and it is not possible for a programmer to track which piece of code is doing what. So, to avoid this kind of issues we use Function.

Function :-

A function is a block of code which performs a particular task. A function can be reused by the programmer in a given program any number of times. We have to take three steps to work with the function and the three ways are :-

- ***Function Prototype***
- ***Function Call***
- ***Function Definition***

Let us look at each of them in detail.

Function Prototype :-

Function Prototype is a way to tell the compiler about the function we are going to define in the program. We must add a datatype while declaring the function. Function declaration must be done before int main function execution starts.

Function Call :-

Function Call is a way to tell the compiler to execute the function body (Function Definition) at the time the call is made.

Function Definition :-

Function Definition contains the exact set of instructions which are executed during the function call. When a function is called from main() ; the main function falls asleep and gets temporarily suspended. During this time , the control goes to the function being called. When the function body is done , the main resumes. The Function Definition must be done after the return ; is called.

Syntax :-

```
#include <stdio.h>

void display ();          ---> Function Prototype

int main ()
{
     display();           ---> Function Call
      return 0;
}
void display ()           ---> Function Definition
{
    printf("Prachi");
}
```

Whenever display(); function is called , it will print Prachi on the screen !

Try It Yourself :-

Write a program with 3 functions :- GoodMorning , GoodAfternoon , GoodEvening which prints Good Morning , Good Afternoon , Good Evening respectively. main() should call them in sequence.

C program can have more than one function and every function gets called directly or indirectly from main().

Types of Functions :-

1. Library Functions :- Commonly required functions grouped together in a library file on disk.

2. User Defined Functions :- These are the functions declared and defined by the user.

Why to use Functions ?

1. To avoid returning the same logic again and again.
2. To keep track of what we are doing in a program.
3. To test and check logic independently.

Passing Values to Functions :-

We can pass values to a function and can get a value in return from a function.

Parameters :-

Parameters are the values or variable placeholders in function definition.

Arguments :-

Arguments are the actual values passed to the function to make a call.

Important points :-

1. A function can return only 1 value at a time.
2. If the passed variable is changed inside the function call doesn't change the value in the calling function.

   Example :-

```
#include <stdio.h>
int change (int a);
int main () {
      int b = 22;
      change (b);
      printf("%d" , b);
      return 0;
}
int change (int a) {
     a = 77;
return 0;
}
```

change is a function which changes a to 77. This program will print 22. This happens because a copy of b is passed to the change function.

Try It Yourself :-

Use the library functions to calculate the area of a square with side a.

Recursion :-

A function defined in C can call itself. This is called Recursion. A function calling itself is also known as recursive function.

A very good example of recursion is factorial.

Since we can write factorial of a number in terms of itself , we can program it using Recursion.

Recursion is sometimes the most direct way to code an algorithm. The condition which doesn't call the function any further in a recursive

function is called as the base condition. Sometimes , due to a mistake made by the programmer , a recursive function can keep running without returning anything resulting in a memory error.

Program to find Factorial of a number using Recursion in C :-

```
#include<stdio.h>
int factorial(int number);
int main() {
   int number;
   printf("Enter a number :-\n");
   scanf("%d" , &number);
   printf("Factorial of %d is %d", number , factorial(number));
   return 0;
}
int factorial(int number) {
   if (number>=1)
      return number*factorial(number-1);
   else
      return 1;
}
```

Here , factorial calls itself till the number reaches 1. And when a function call itself , it is called Recursion.

Chapter – 5 Practice Set

1. Write a C program using functions to find average of three numbers.
2. Write a C program to using functions to convert Celsius to Fahrenheit.
3. Write a C program using functions to calculate force of attraction on a body of mass m exerted by earth. (G = 9.8m/s^2)
4. Write a C program to using functions to print the following pattern :-

```
*

***

******
```

Chapter – 6

Pointers

A pointer is a variable which stores the address of another variable.

i = 6;

address → 87994

j = 87994;

address → 87998

‘ j ’ is a pointer because it is storing the address of ‘ i ’ in it.

The ‘ address of ’ operator (&) is used to obtain the address of given variable. If you refer the examples above :-

&i → 87994

&j → 87998

Format specifier for printing pointer address is ***"%u"***.

The **value at address** operator (*) is used to obtain the value of present at a given memory address.

*(&i) = 6

*(&j) = 87994

How to declare a pointer ?

int *j = &i ; → Declare pointer j and initialize it to address of i

Pointer can be type of int , char and float as well.

Although it is a good practice to use meaningful variable names , we should be careful while reading and working on programs of fellow programmers.

A Program to demonstrate pointers :-

```
#include <stdio.h>

int main ()
{
        int Prachi = 6 ;
        int *Pointer = &Prachi ;

        printf("Value of Prachi is %d\n" , Prachi);
        printf("Value of Prachi is %d\n" ,*Pointer);
        printf("Address of Prachi is %u\n" , &Prachi);
        printf("Address of Prachi is %u\n" , Pointer);
        printf("Address of Pointer is %u\n" , &Pointer);
        printf("Address of Prachi is %d\n" , *(&Pointer));
        return 0;
}
```

Output :-

Value of Prachi is 6

Value of Prachi is 6

Address of Prachi is 6422300

Address of Prachi is 6422300

Address of Pointer is 6422296

Address of Prachi is 6422300

Pointer To A Pointer :-

Just like j is pointing to I or starting the address of i , we can have another variable k which can further store the address of j. What will be type of k ?

i = 6;

address → 87994

j = 87994;

address → 87998

k = 87998;

address → 88004

int **k;

k = &j;

We can go further as many times we need.

Types of function calls :-

Based on the way we pass arguments to the function; function calls are of two types.

1. Call by value → sending the values of arguments.
2. Call by reference → sending the address of arguments

Call by value :-

Here the values of the arguments are passed to the function. Consider this example:

int c = sum (3 , 4); // Assume x = 3 , y = 4

If sum is defined as sum(int a, int b), the values 3 and 4 are copied to a and b. Now even if we change a and b, nothing happens to the variables x and y.

This is **call by value.**

In C, we usually make a call by value.

Call by Reference:

Here the address of the variable is passed to the function as arguments.

Now since the addresses are passed to the function, the function can now modify the value of a variable in calling function using * and & operators. Example:

```
void swap (int *x , int *y)
{
   int temp;
   temp = *x;
   *x = *y;
    *y = temp;
}
```

This function is capable of swapping the values passed to it. If a=3 and b=4 before a call to swap(a , b), a=4 and b=3 after calling swap.

We declared the variables as pointers so we passed arguments as address so this is call by reference.

```
int main ()
{
int a = 3;
int b = 4;
swap(&a , &b)
return 0;
}
```

Chapter – 6 Practice Set

1. Write a program to print the address of a variable. Use this address to get the value of this variable.

2. Write a program having a variable i. Print the address of i. Pass this variable to a function and print its address. Are these addresses same? Why?

3. Write a program to change the value of a variable to ten times its current value. Write a function and pass the value by reference.

4. Write a program using a function that calculates the sum and average of two numbers. Use pointers and print the values of sum and average in main().

5. Write a program to print the value of a variable i by using the "pointer to pointer" type of variable.

6. Try problem 3 using call by value and verify that it doesn’t change the value of the said variable.

Chapter – 7

Arrays

An array is a collection of similar elements. One variable --> Capable of storing multiple values.

Syntax :-

The syntax of declaring an array looks like this :-

```
int marks [3] ;      // Integer Array

char name [4] ;      // Character Array

float percentile [6] ;      // Float Array
```

The values can now be assigned to marks array like this :-

```
marks [0] = 23;

marks [1] = 100;

marks[2] = 93;
```

Note: It is very important to note that the array index starts with 0.

Accessing Elements :-

Elements of an array can be accessed using :

```
scanf("%d" , &marks[0]);
printf("%d" , marks[0]);
```

Try It Yourself :-

Write a program to accept marks of five students in an array and print them to the screen.

Initialization of an array :-

```
int array [3] = {9,8,8}
```
→ Arrays can be initialized while declaration

```
float marks[] = {33,40}
```

Arrays In Memory :-

Consider This Array :-

int array [3] = {1,2,3} → 1 integer = 4 bytes

This will reserve 4x3=12 bytes in memory. 4 bytes for each integer.

1	2	3

62302 62306 62310 {Array in Memory}

Pointer Arithmetic :-

A pointer can be incremented to point to the next memory location of that type :-

Consider this example →

```
int i = 32;
int *a = &i;    =--> a =87994 (address = 87994)
a++;            =--> now a = 87998

char a = 'A';
char *b = &a;   =--> b = 87994
b++;            =--> now b = 87995

float i = 1.7;
float*a = &i;   =--> Address of i or a = 87994
a++;            =--> Now a = 87998
```

Following operations can be performed on pointers :-

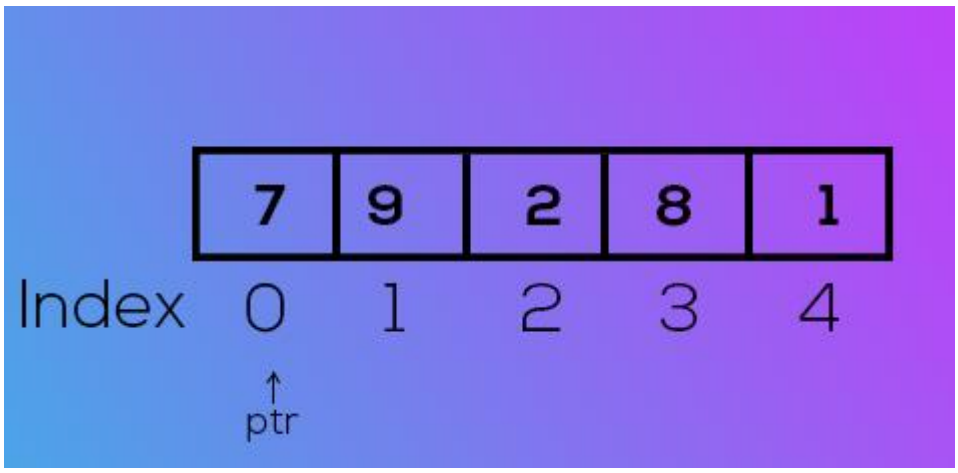
1. Addition of a number to a pointer.
2. Subtraction of a number from a pointer
3. Subtraction of one pointer from another
4. Comparison of two pointer variables

Try It Yourself :-

Try these operations on another variable by creating pointers in a separate program. Demonstrate all four operations.

Accessing Arrays Using Pointers :-

Consider this array :-



If ptr points to index 0, ptr++ will point to index 1 & so on...

This way we can have an integer pointer pointing to the first element of the array like this :-

int * ptr = & arr[0]; --> or simply arr

ptr++;

*ptr --> will have 9 as its value

Passing Arrays To Functions :-

Arrays can be passed to the functions like this :-

printArray(arr,n); --> function call

void printArray(int *i , int n); --> function prototype

or

void printArray(int i[] ,int n);

Multidimensional Arrays :-

An array can be of 2 dimension / 3 dimension / n dimensions.

A 2-dimensional array can be defined as :

int arr [3][2] ={

```
{1,4}
{7,9}
{11;22}     };
```

We can access the elements of this array as arr [0] [0], arr [0] [1] & so on...

At arr [0] [0] value would be 1 and at arr [0] [1] value would be 4.

2-D arrays in Memory :-

A 2-d array like a 1-d array is stored in contiguous memory blocks like this:

Try It Yourself :-

Create a 2-d array by taking input from the user. Write a display function to print the content of this 2-d array on the screen.

Chapter – 7 Practice Set

1. Create an array of 10 numbers. Verify using pointer arithmetic that (ptr+2) points to the third element where ptr is a pointer pointing to the first element of the array.
2. If S[3] is a 1-D array of integers, then *(S+3) refers to the third element:

- True
- False
- Depends

3. Write a program to create an array of 10 integers and store a multiplication table of 5 in it.
4. Repeat problem 3 for a general input provided by the user using scanf().

5. Write a program containing a function that reverses the array passed to it.
6. Write a program containing functions that counts the number of positive integers in an array.
7. Create an array of size 3x10 containing multiplication tables of the numbers 2,7 and 9, respectively.
8. Repeat problem 7 for a custom input given by the user.
9. Create a three-dimensional array and print the address of its elements in increasing order.

Chapter – 8

Strings

A string is a 1-d character array terminated by a null('\0') --> {this is null character}

The null character is used to denote string termination, characters are stored in contiguous memory locations.

Initializing Strings :-

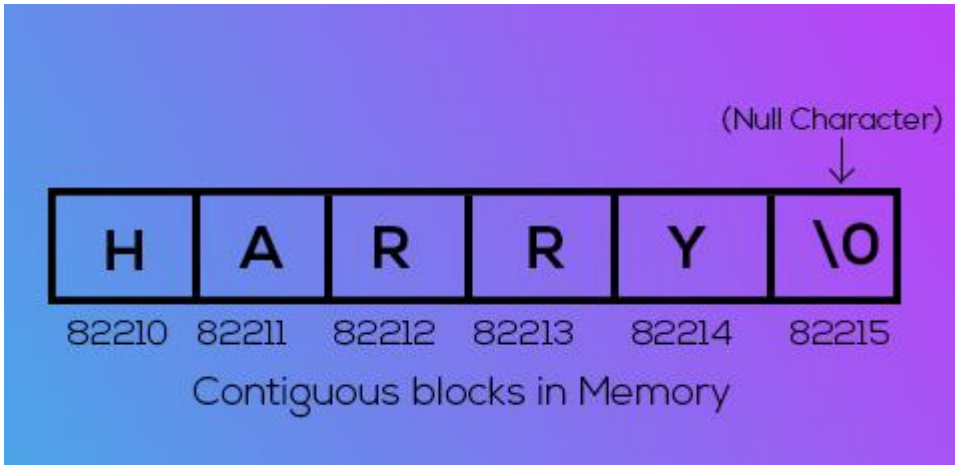
Since string is an array of characters, it can be initialized as follows:

char s[]={'H','A','R','R','Y','\0'}

There is another shortcut for initializing strings in C language:

char s[]="HARRY"; --> In this case C adds a null character automatically. Strings in memory

A string is sorted just like an array in the memory as shown below



Try It Yourself :-

Create a string using " " and print its content using a loop.

Printing Strings :-

A string can be printed character by character using printf and %c.

But there is another convenient way to print strings in C.

char st[] = ”HARRY”;

printf(“%s”, st); --> prints the entire string

Taking string input from the user :-

We can use %s with scanf to take string input from the user :

char st[50];

scanf(“%s”, &st);

scanf automatically adds the null character when the enter key is pressed.

Note:

1. The string should be short enough to fit into the array.
2. scanf cannot be used to input multi-word strings with spaces.

gets() and puts() :-

gets() is a function that can be used to receive a multi-word string.

char st[30];

gets(st); --> the entered string is stored in st!

Multiple gets() calls will be needed for multiple strings.

Likewise, puts can be used to output a string.

puts(st); -->Prints the string and places the cursor on the next line

Declaring a string using pointers :-

We can declare strings using pointers :-

char *ptr= ”Harry”;

This tells the compilers to store the string in the memory and the assigned address is stored in a char pointer.

Note:

1. Once a string is defined using char st[] = “harry”, it cannot be initialized to something else.
2. A string defined using pointers can be reinitialized. --> ptr = “Rohan”;

Standard library functions for Strings :-

C provides a set of standard library functions for strings manipulation.

Some of the most commonly used string functions are:

strlen() - This function is used to count the number of characters in the string excluding the null ('\0') character.

Example :-

int length=strlen(st);

These functions are declared under <string.h> header file.

strcpy() - This function is used to copy the content of second string into first string passed to it.

char source[ ]= ”Harry”;

char target[30];

strcpy(target , source); --> target now contains “Harry”

Target string should have enough capacity to store the source string.

strcat() - This function is used to concatenate two strings

char s1[11]= ”Hello”;

char s2[ ]= ”Harry”;

strcat(s1,s2); --> s1 now contains “Hello Harry” <No space in between>

strcmp() - This function is used to compare two strings. It returns: 0 if strings are equal

Negative value if first strings mismatching character's ASCII value is not greater than second string's corresponding mismatching character. It returns positive values otherwise.

strcmp(“For”, “Joke”); --> positive value

strcmp(“Joke”, “For”); --> Negative value

Chapter – 8 Practice Set

1. Which of the following is used to appropriately read a multi-word string ?

- Gets()
- Puts()
- Printf()
- Scanf()

2. Write a program to take a string as an input from the user using %c and %s. Confirm that the strings are equal.

3. Write your own version of strlen function from <string.h>
4. Write a function slice() to slice a string. It should change the original string such that it is now the sliced strings. Take m and n as the start and ending position for slice.
5. Write your own version of strcpy function from <string.h>
6. Write a program to encrypt a string by adding 1 to the ASCII value of its characters.
7. Write a program to decrypt the string encrypted using the encrypt function in problem 6.
8. Write a program to count the occurrence of a given character in a string.
9. Write a program to check whether a given character is present in a string or not.

Chapter – 9

Structures

Arrays and Strings --> Similar data (int, float, char)

Structures can hold --> dissimilar data

Syntax for creating Structures :-

A C Structure can be created as follows:

We can use this user-defined data type as follows:

```
struct employee{
int code;        --> this declares a new user-defined datatype
float salary;
char name[10];
};               →semicolon is important
```

We can use this user-defined data type as follows :-

So, a structure in c is a collection of variables of different types under a single name.

Try It Yourself :-

Write a program to store the details of 3 employees from user-defined data. Use the structure declared above.

Why to use Structures?

We can create the data types in the employee structure separately but when the number of properties in a structure increase, it becomes difficult for us to create data variables without structures. In a nutshell:

1. Structures keep the data organized.
2. Structures make data management easy for the programmer.

Array of Structures :-

Just like an array of integers, an array of floats, and an array of characters, we can create an array of structures :-

struct employee Facebook[100]; -->an array of structures

We can access the data using :-

Facebook[0].code=100;

Facebook[1].code=101;

.....and so on.

Initializing structures :-

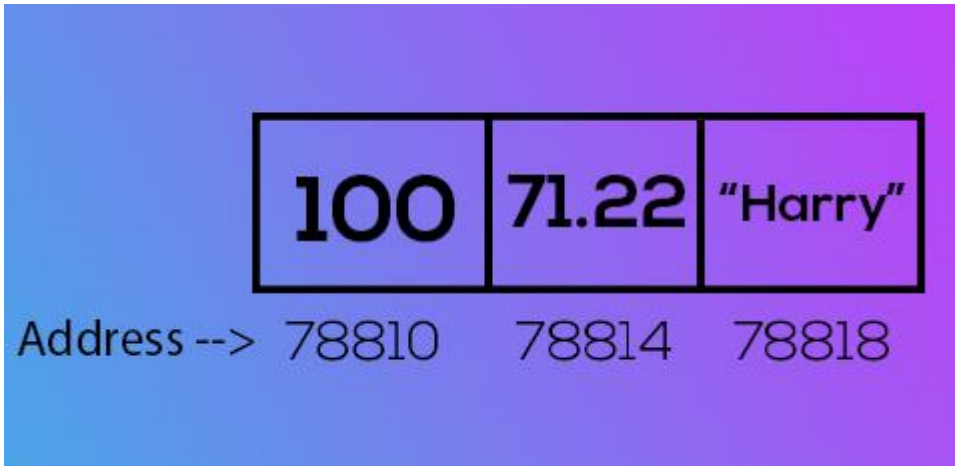
Structures can also be initialized as follows:

struct employee harry={100,71.22,”Harry”};

struct employee shubh={0}; // All the elements set to 0

Structures in memory :-

Structures are stored in contiguous memory locations for the structures e1 of type struct employee, memory layout looks like this:



In an array of structures, these employee instances are stored adjacent to each other.

Pointer to structures :-

A pointer to the structure can be created as follows:

Now we can print structure elements using:

struct employee *ptr;

ptr=&e1;

Arrow operator :-

Instead of writing *(ptr).code, we can use an arrow operator to access structure properties as follows :-

*(ptr).code or ptr -> code

Here **->** is known as an arrow operator.

Passing Structure to a function :-

A structure can be passed to a function just like any other data type.

void show(struct employee e); -->Function prototype

Try It Yourself :-

Complete this show function to display the content of the employee.

Typedef keyword :-

We can use the typedef keyword to create an alias name for data types in c.

typedef is more commonly used with structures :-

```
struct complex{
float real;    // struct complex c1,c2; for defining complex numbers
float img;
};
typedef struct complex{
float real;
float img;    // ComplexNo c1,c2; for defining complex numbers
}
ComplexNo;
```

Chapter – 9

Practice Set

1. Create a two-dimensional vector using structures in C.

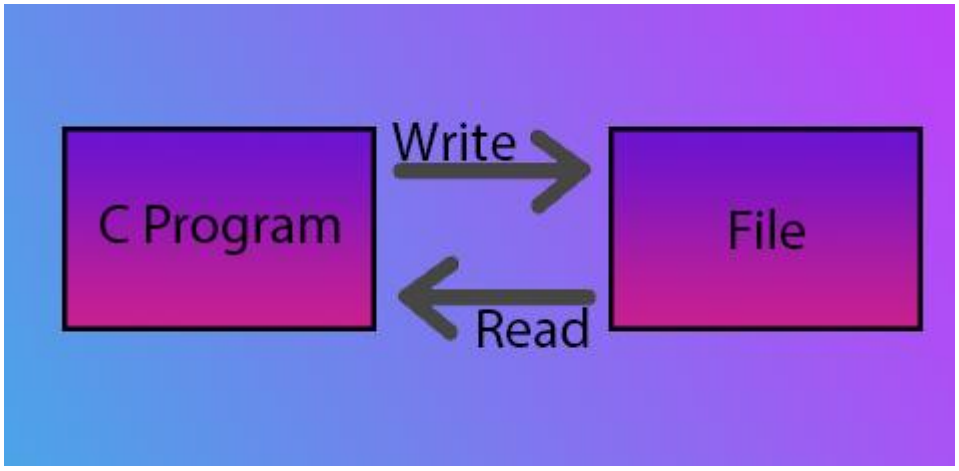
2. Write a function SumVector which returns the sum of two vectors passed to it. The vectors must be two-dimensional.
3. Twenty integers are to be stored in memory. What will you prefer- Array or Structure?
4. Write a program to illustrate the use of an arrow operator -> in C.
5. Write a program with a structure representing a Complex number.
6. Create an array of 5 complex numbers created in problem 5 and display them with the help of a display function. The values must be taken as an input from the user.
7. Write problem 5's structure using typedef keyword.
8. Create a structure representing a bank account of a customer. What fields did you use and why?
9. Write a structure capable of storing date. Write a function to compare those dates.
10. Solve problem 9 for time using typedef keyword.

Chapter – 10

File I/O

The random-access memory is volatile and its content is lost once the program terminates. In order to persist the data forever, we use files.

A file data stored in a storage device. A C program can talk to the file by reading content from it and writing content to it.



File pointer :-

The “File” is a structure that needs to be created for opening the file. A file pointer is a pointer to this structure of the file.

File pointer is needed for communication between the file and the program.

A file pointer can be created as follows:

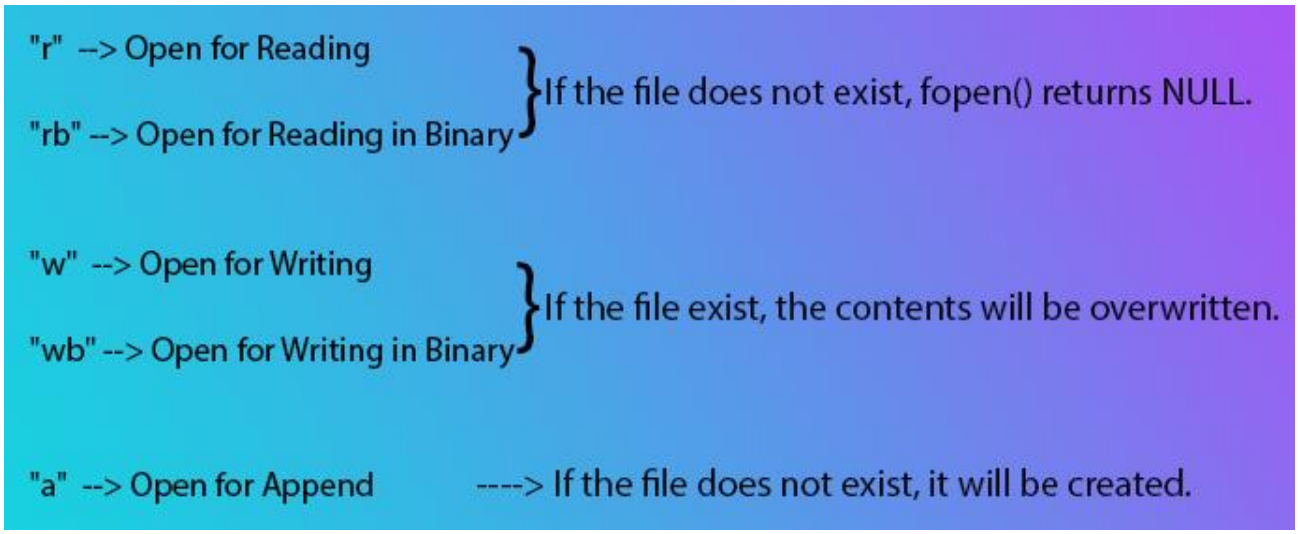
FILE *ptr;

ptr=fopen(“filename.ext”,”mode”);

File opening modes in C :-

C offers the programmers to select a mode for opening a file.

Following modes are primarily used in C File I/O :-



Types of Files :-

There are two types of files:

1. Text files(.txt , .c)
2. Binary files(.jpg , .dat)

Reading a file :-

A file can be opened for reading as follows :-

```
FILE *ptr;

ptr=fopen("Harry.txt","r");

int num;
```

Let us assume that "Harry.txt" contains an integer

We can read that integer using:

```
fscanf(ptr,"%d",&num);
```

→ fscanf is file counterpart of scanf

This will read an integer from the file in the num variable.

Try It Yourself :-

Modify the program above to check whether the file exists or not before opening the file.

Closing the file :-

It is very important to close file after read or write. This is achieved using fclose as follows:

fclose(ptr);

This will tell the compiler that we are done working with this file and the associated resources could be freed.

Writing to a file :-

We can write to a file in a very similar manner as we read the file :-

```
FILE *fptr;

fptr=fopen(“Harry.txt”,”w”);

int num=432;

fprintf(fptr,”%d”,num);
```

fclose(fptr);

fgetc and fputc are used to read and write a character from/to a file :-

fgetc(ptr); => Used to read a character from file

fputc('c',ptr); => Used to write character 'c' to the file

EOF: End of File :-

fgetc returns EOF when all the characters from a file have read. So , we can write a check like below to detect the end of file :-

```
while(1){

ch=fgetc(ptr);      // When all the content of a file has been read, break the loop

if(ch==EOF){

break;

}
```

```
//code
}
```

Chapter – 10

Practice Set

1. Write a program to read three integers from a file.
2. Write a program to generate a multiplication table of a given number in text format. Make sure that the file is readable and well-formatted.
3. Write a program to read a text file character by character and write its content twice in a separate file.
4. Take name and salary of two employees as input from the user and write them to a text file in the following format:

   name1, 3300

   name2, 7700

5. Write a program to modify a file containing an integer to double its value.

   If old value = 2, then new file value = 4

Project – 1 Snake Water Gun Game

Problem :-

Snake, Water, Gun or **Rock, Paper, Scissors** is a game most of us have played during school time. Write a C program capable of playing this game with you. Your program should be able to print the result after you choose Snake/Water or Gun.

Chapter – 11

Dynamic Memory Allocation

C is a language with some fixed rules of programming. For example: Changing the size of an array is not allowed.

Dynamic Memory Allocation :-

Dynamic memory allocation is a way to allocate memory to a data structure during the runtime we can use DMA function available in C to allocate and free memory during runtime.

Function for DMA in C :-

Following functions are available in C to perform dynamic memory allocation:

1.malloc()
2.calloc()
3.free()
4.realloc()

malloc() function :-

Malloc stands for memory allocation. It takes number of bytes to be allocated as an input and returns a pointer of type void.

Syntax:

The expression returns a NULL pointer if the memory cannot be allocated.

Try It Yourself :-

Write a program to create a dynamic array of 5 floats using malloc().

calloc() function :-

calloc stands for continuous allocation.

It initializes each memory block with a default value of 0.

Syntax:

ptr = (float*) calloc(30*sizeof(int)) //Allocates Contiguous space in memory for 30 blocks

If the space is not sufficient, memory allocation fails and a NULL pointer is returned.

Try It Yourself :-

Write a program to create an array of size n using calloc() where n is an integer entered by the user.

free() function :-

We can use free() function to allocate the memory.

The memory allocated using calloc/malloc is not deallocated automatically.

Syntax:

free(ptr); --> Memory of ptr is released

Try It Yourself :-

Write a program to demonstrate the usage of free() with malloc().

realloc() function :-

Sometimes the dynamically allocated memory is insufficient or more than required.

realloc is used to allocate memory of new size using the previous pointer and size.

Syntax:

ptr = realloc(ptr,newSize);

ptr = realloc(ptr, 3* sizeof(int)) //ptr now points to this new block of memory, which is capable of storing 3 integers

Chapter – 11

Practice Set

1. Create an array dynamically capable of storing 5 integers. Now use realloc so that it can now store 10 integers.
2. Create an array of the multiplication table of 7 up to 10 (7x10=70). Use realloc to make it store 15 numbers(from 7x1 to 7x15).