INDEX

EXPERIMENT NO: 01

Title: Algorithm and Flowchart.
Aim: To write algorithm and draw a flow chart for given problem.

Theory:

Algorithm:
In programming, algorithm is the set of well-defined instruction in sequence to solve a program. An algorithm should always have a clear stopping point.
Qualities of a good algorithm

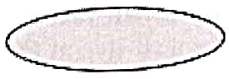
1. Inputs and outputs should be defined precisely.
2. Each steps in algorithm should be clear and unambiguous.
3. Algorithm should be most effective among many different ways to solve a problem.
4. An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages.

Example:
Write an algorithm to add two numbers entered by user.
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
0sum←num1+num2
Step 5: Display sum
Step 6: Stop

Flowchart:

Flowchart is graphical representation of algorithm or problem.

There are different symbols are used for representation
1. Oval - Begin/End



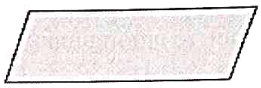
The first figure is oval. Also could be met as an "ellipse","circle", but it has the same meaning. This is the first and the last symbol in every flow chart. I like to use ellipse for

"begin" and "end". When I divide an algorithm in several parts I use small circles for the start/end of each part.

Rectangle - Assignment statements

You will use a rectangle to perform an action (also called "statement"). More precisely, we use them for assignment statements - when you change a value of a variable.

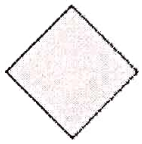
3. Parallelogram - Input / Output



The parallelogram flow chart symbol serves for input/output(I/O) to/from the program.

One note: output means that the program gives an output, for example – display a message on the screen. Input is an input to the program. It could be when the user enters a value for a variable.

4. Rhombus - making a decision

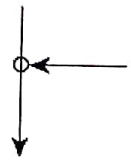


Rhombus: This is what we use when our program has to make a decision. This is the only block that has more than one exit arrow. The rhombus symbol has one(or several) entrance point and exactly two outputs.

Inside it stays a valid conditional expression. The expression could be evaluated as "true" or "false". If the result is true, the next step of the algorithm is the element, pointed by the "True" (Yes) arrow. If the condition is false we continue with the "false"(No) arrow. Note that both cases must be handled – you can't leave one of them handing.

5. Arrows and junctions

Arrows show the order of the blocks in the chart flow. There is no convention for the direction of the arrows. We choose which direction is better for our graphics.



Sample Questions:

1. What is Flowchart?
2. Write an Algorithm to perform arithmetic Operations?
3. What is the relation between a flowchart and an algorithm?
4. What is a symbol used connect two symbols of flowchart?
5. Write Algorithm and Draw Flowchart for Following Problem:
    - Find the sum of 5 numbers
    - Print Hello World 10 times
    - Draw a flowchart to log in to Facebook account

EXPERIMENT NO: 02

Title: Program to print "hello world". (To display a message on Screeen)
Aim: Implement a simple program to print "Hello world " on screen using printf() function.

Theory:

The objective of the first laboratory experiment is to make students familiar with the different Operating Systems on which they can able to run a C Program. Also students will be able to use different IDEs on Windows Platform as well as on Linux Platform and Compilers such as gcc, g++, cc etc for program processing.
Student should execute a program which will display the message "Basic Computer Programming" on Screen. Student should compile, and execute the program on either Linux or Windows Operating System. Students should also use different Escape Sequence characters such as "\n" to print a message on newline, "\t" for tab between to words, "\b" for backspace (to delete a character which is before the \b) etc. in the same program.

Program:

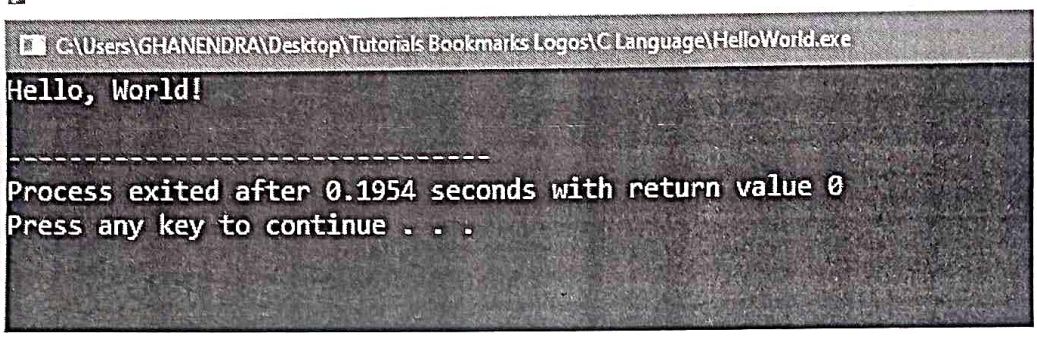
```
#include<stdio.h>
void main()
{
printf("Basic Computer Programming");
}
```

To write a program: use vi or gedit editor and save the program with .c extension.
To Compile: On Linux, g++ Program Name.c and on Turbo C/C++ IDE alt+F9.
To Run: ./a.out on Turbo C/C++ IDE ctrl+F9.

Program Output: Basic Computer Programming

HelloWorld.c

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");

    return 0;
}
```



Sample Programs:

1. C program to check whether the given number is positive or negative
2. Reverse an input number using recursion
3. Program to find greatest of three numbers
4. C Program to print Fibonacci series in a given range
5. C Program to find factorial of a given number
6. Find Prime numbers in a given range

EXPERIMENT NO: 03

Title: Standard input output functions (printf() and scanf())
Aim: Implement program which take input from user and display the value of variables on console.

Theory:

The objective of the second laboratory experiment is to make students familiar with how to use *declaration of variable* in C program, different *primitive data types* available in C and different placeholders i.e. *format specifier* for the primitive data types. Also get to know the purpose of in built function i.e. *scanf()* in C Programming Language and what is the reason to use *address of operator (&)* in *scanf()*. Student should execute a program which will take the input value through keyboard and same value display on User's Screen. Student should use different data types in same program to take input through keyboard. Student also execute a program that will use ASCII code of different characters (display ASCII code of a character on Screen). Student should also execute a program that demonstrate printing the number right justified within columns, rounds to two or three or zero digit places, display a number in exponential notation.

Program:

```
#include<stdio.h>

void main()

{

int a, d =1130; //variable declaration and initialization

float b;

char c;

printf("\n Enter a Number:");

scanf("%d", &a); // %d format string is used in case of integer

printf("\n Entered number = %d and assign value to d = %d", a, d);

printf("\n 4 digit integer right justified to 6 column: %6d\n", d);
```

```
printf("\n Enter a number: ");
scanf("%f", &b); // %f format string is used in case of floats
printf("\n Value of b = %f", b);
printf("\n Floating point number rounded to 2 digits: %.2f\n", b);
printf("\n Enter a character: ");
scanf("%c", &c); // %c format string is used in case of character
printf("\n You entered a character = %c", c);
printf("\n ASCII value of a character %c is %d.", c, c);
}
```

Program Output:

Enter a Number: 5

Entered number = 5 and assign value to d = 1130

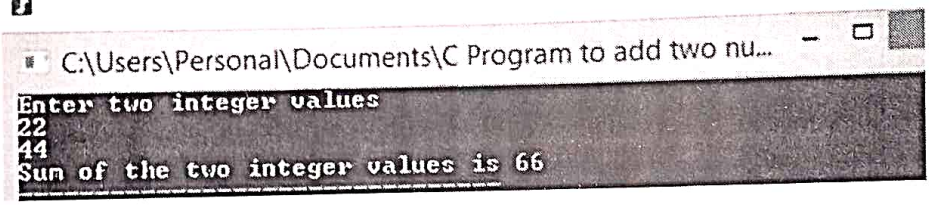
4 digit integer right justified to 6 column: 1130

Enter a number: 4.952

Value of b = 4.952000

```
#include<stdio.h>

int main()
{
 int a = 24, b = 4;
 int addition, subtraction, multiplication, division, modulus;

 addition = a + b;
```

```
#include <stdio.h>
int main()
{
   int number1, number2, sum;
   printf("Enter two integer values \n");
   scanf("%d %d",&number1,&number2);
   sum = number1+number2;
   printf("Sum of the two integer values is %d", sum);
   return 0;
}
```



Sample Questions:

1. What are the commonly used input output functions in C?
2. Which function takes input from user in C?
3. How to take input and output of advanced type in C?
4. Predict the output of following program?

```
#include "stdio.h"
int main()
{
 char arr[100];
 printf("%d", scanf("%s", arr));
 /* Suppose that input value given
 for above scanf is "GeeksQuiz" */
 return 1;
 }
```

5. Write a C program that accepts some integers from the user and find the highest value and the input position.

EXPERIMENT NO: 04

Title: Use of operators and expressions.
Aim: To perform programs by using c operators and expression.

Theory:

OPERATORS IN C:-

An operator is a symbol which tells the computer to perform certain mathematical or logical manipulations. The operators are used in mathematical and logical expressions. An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators

- ❑ Arithmetic Operators
- ❑ Relational Operators
- ❑ Logical Operators
- ❑ Bitwise Operators
- ❑ Assignment Operators
- ❑ Misc Operators

Arithmetic Operators:-

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then

Operator	Description	Example
+	Adds two operands.	A + B = 30
–	Subtracts second operand from the first.	A – B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- =

Relational Operators:-

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A! = B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false	

Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. The truth tables for &, |, and ^ is as follows:-

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assignment Operators

The following table lists the assignment operators supported by the C language:-

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

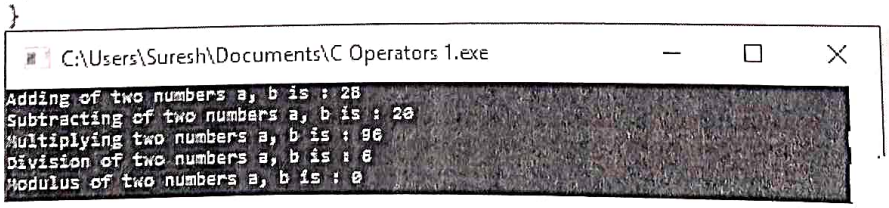
Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

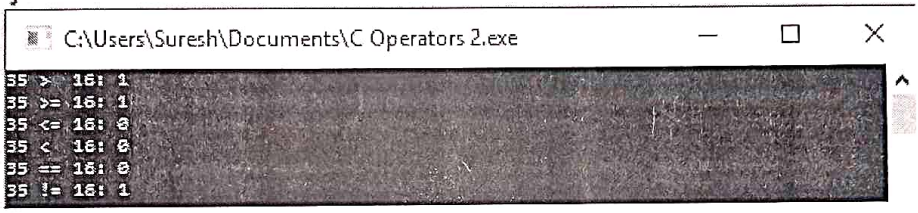
Arithmetic Operators Example

```
#include<stdio.h>

int main()
{
 int a = 24, b = 4;
 int addition, subtraction, multiplication, division, modulus;

 addition = a + b;
 subtraction = a - b;
 multiplication = a * b;
 division = a / b;
 modulus = a % b;

 printf("Adding of two numbers a, b is : %d\n", addition);
 printf("Subtracting of two numbers a, b is : %d\n", subtraction);
 printf("Multiplying two numbers a, b is : %d\n", multiplication);
 printf("Division of two numbers a, b is : %d\n", division);
 printf("Modulus of two numbers a, b is : %d\n", modulus);
}
```



Relational Example

```
#include <stdio.h>

main()
{
  int a = 35;
  int b = 16;

  printf("%d >  %d: %d \n", a, b, a > b);
  printf("%d >= %d: %d \n", a, b, a >= b);
  printf("%d <= %d: %d \n", a, b, a <= b);
  printf("%d <  %d: %d \n", a, b, a < b);
  printf("%d == %d: %d \n", a, b, a == b);
  printf("%d != %d: %d \n", a, b, a != b);
}
```



Sample Questions:

1. Who is the father of C language?
2. Which operator takes only integer operands?
3. C programs are converted into machine languge with the help of
   - An editor
   - A compiler
   - An operating system
   - None
4. How many keywords are recognized by standard ANSI C?
5. Why variable names beginning with underscore is not encouraged?
6. In C programming language, which of the following types of operators will have highest precedence
7. If i,j,k are integer variables with values 1,2 3 respectively, then what is the value of the expression !((j+k) > (i+5)?
8. Arrange these operators ., | |, <, =, if arranged in ascending order of precedence?
9. The format identifier "%i" is used for which data type?

- **LAB ASSIGNMENT:**

1.Find the output of the following code?

```
void main()
{
int i=10;
i = !i > 14;
printf("i=%d",i);
}
```

2.Find the output of the following code?

```
void main()
{
int i=0, j=1, k=2,m; m = i++ || j++ || k++;
printf(%3d%3d%3d%3d", m, i, j, k);
}
```

3.Find the output of the following code?

```
void main()
{
int main=3;

printf("%d", main);
}
```

4.Find the output of the following code?

```
void main()
{
int k;
k = 'a' > 60;
printf("%d", k);
}
```

5.Find the output of the following code?

```
void main()
{
printf("iare\rcollege\n");
}
```

6.Find the output of the following code?

```
void main()
{
int const k = 5; k++;
printf("k is %d", k);
}
```

7.Find the output of the following code?

```
int x;
void main()
{
```

```
if (x) else
}
 printf("hi"); printf("how are u");
```

8.Find the output of the following code?

```
 void main()
{
int x = 0; if (x == 0)
printf("hi"); else
printf("how are u"); printf("hello");
}
```

compile error

9.Find the output of the following code?

```
void main()
{
int x = 5;
if (x < 1); printf("Hello");
}
```

hello

10.Find the output of the following code?

```
 void main()
{
int ch;
printf("enter a value btw 1 to 2:");

scanf("%d", &ch);
switch (ch)
{
case 1: printf("1\n"); default:printf("2\n");
}
}
```

1 = 1 | 2
other than
↓
2

11.Find the output of the following code?

```
void main()
{
char ch; int i;
ch = 'G';
i = ch-'A'; printf("%d", i);
}
```

6

12.Find the output of the following code?

```
void main()
{
int i=065, j=65; printf("%4d%4d",i, j);
}
```

Error printing terminal

13.Find the output of the following code?

```
void main()
{
unsigned int a = 10; a = ~a; printf("%d\n", a);
```

}

14.Find the output of the following code?

```
 void main()
{
int x = 97;
int y = sizeof(x++); printf("x is %d",x);
}
```

15.Find the output of the following code?

```
void main()
{
int x = 4, y, z; y = --x;
z = x--;
printf("%3d%3d%3d", x, y, z);
}
```

2 3 3

EXPERIMENT NO: 05

Title: Decision Making statements(if---else, nested if else, else if ladder)
Aim: To implement program using decision making statements.

Theory:

Decision Control statements:

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

There are many ways to use if statement in C language:

- If statement
- If-else statement
- If else-if ladder
- Nested if

1. If Statement

The single if statement in C language is used to execute the code if condition is true. The syntax of if statement is given below:

if(**expression**)

{

//code to be executed

}

2.If-else Statement

The if-else statement in C language is used to execute the code if condition is true or false. The syntax of if-else statement is given below:

if(**expression**)

{

```
//code to be executed if condition is true
}
else
 {
//code to be executed if condition is false
}
```

3.If else-if ladder Statement

The if else-if statement is used to execute one code from multiple conditions. The syntax of if else-if statement is given below:

```
if(condition1)
{
//code to be executed if condition1 is true
}
else
{
 if (condition2)
 {
 //code to be executed if condition2 is true
}
else
{
 if(condition3)
{
```

```
//code to be executed if condition3 is true
}
...
else
{
//code to be executed if all the conditions are false
}
}
}
```

4. Nested if Statement

```
if (condition1)
 {
 //code to be executed if condition1 is true
 if (condition2)
  {
  //code to be executed if condition2 is true
  }
  else
  {
   //code to be executed if all the conditions are false
  }
  }
```

```
/* If Statement in C language Example */
#include <stdio.h>
int main()
{
    int number;

    printf("Enter any integer Value\n");
    scanf("%d",&number);

    if( number >= 1 )
    {
        printf("You Have Entered Positive Integer\n");
    }
    return 0;
}
```

```
Enter any integer Value
 22
You Have Entered Positive Integer
```

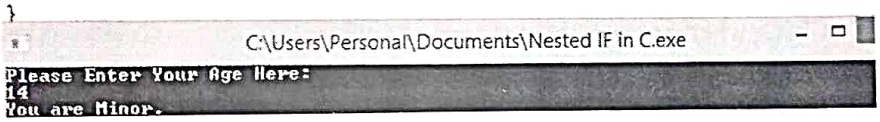
If else statement

```
#include <stdio.h>
int main()
{
 int marks;
 printf("Enter you subject Marks:\n");
 scanf("%d",&marks);
 if(marks >= 50)
  {
   printf("Congratulations\n"); //s1
   printf("You cleared the subject"); //s2
  }
 else
  {
   printf("You Failed\n");//s3
   printf("Better Luck Next Time");//s4
  }
```

```
C:\Users\Personal\Documents\IF ELSE Statement ...
Enter you subject Marks:
60
Congratulations
You cleared the subject
```

Nested If

```
#include<stdio.h>
int main()
{
 int age;
 printf("Please Enter Your Age Here:\n");
 scanf("%d",&age);
 if ( age < 18 )
  {
    printf("You are Minor.\n");
    printf("Not Eligible to Work");
  }
 else
  {
   if (age >= 18 && age <= 60 )
    {
     printf("You are Eligible to Work \n");
     printf("Please fill in your details and apply\n");
    }
   else
    {
     printf("You are too old to work as per the governament rules\n");
     printf("Please Collect your pension! \n");
    }
  }
 return 0;
}
```



Sample Questions:

1. What are the two types of flow control used?
2. What do you mean by looping?
3. What is the difference between while and do while loop?
4. Which loop is known as entry controlled loop?
5. Which loop is known as exit control loop?
6. What are the two commands to control the loop?
7. In which loop the test condition is tested at the end of the loop?
8. What is a prime number?
9. Which decision making statement is used to check the determinant value of a quadratic equation?
10. Using which operator we can write multiple conditions in an if statement?

Sample Programs:

1. Find largest number between two numbers using if----else
2. Find number is even or odd using if else.
3. Find greater number between three numbers using nested if else.

4. Program For else if ladder.

- **LAB ASSIGNMENT:**

1. Any character is entered through the keyboard, write a program to determine whether the character entered is a capital letter, a small case letter, a digit, or a special symbol.
2. Write a C program to display the following in triangular form

```
*
*   *
*   *   *
*   *   *   *
```

3. Write a program to display the triangular number where a triangular number is nothing but summation of 1 to given no N.
4. Read a 3 digit number N from keyboard and find individual digits in units place (U), ten's place (T) and hundred's place (H).check U 3 + T2 + H = N (given no)
5. Check whether a year is leap year or not?
6. Display the largest among three numbers using if else statement?
7. Write a C program to calculate the sum of natural numbers 1+2+3+4+ +n
8. Write a C program to reverse a given number?
9. Write a C program to display the series 1 3 5 7 9 11.....
10. Write a C program to display all the ASCII values and their equivalent characters using while loop?

EXPERIMENT NO: 06

Title: Decision making using switch case
Aim: Implement program using Switch Case

Theory:

The switch statement in C language is used to execute the code from multiple conditions. It is like if else-if ladder statement. The syntax of switch statement in c language is given below:

switch(expression)

{

casevalue1:

//code to be executed;

break; //optional

case value2:

//code to be executed;

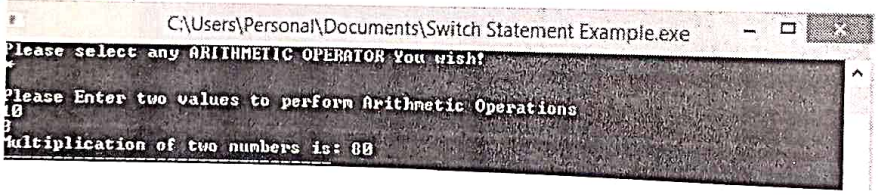
break; //optional

default:

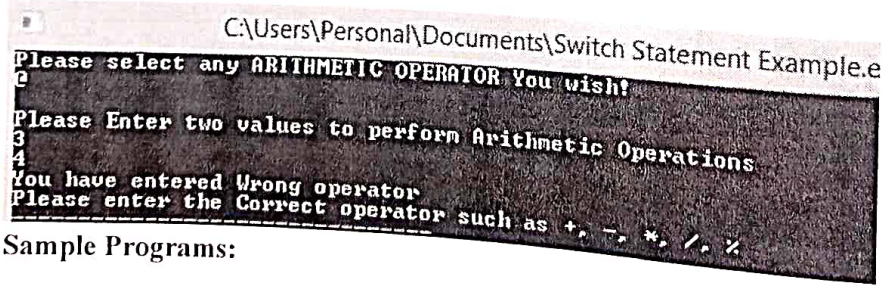
code to be executed if all cases are not matched;

}

```
#include<stdio.h>
int main()
{
 char opertor;
 int number1,number2;
 printf("Please select any ARITHMETIC OPERATOR You wish!\n");
 scanf("%c",&opertor);
 printf("\nPlease Enter two values to perform Arithmetic Operations\n");
 scanf("%d %d",&number1,&number2);
 switch(opertor)
  {
   case '+':
     printf("Addition of two numbers is: %d", number1+number2);
     break;
   case '-':
     printf("Subtraction of two numbers is: %d", number1-number2);
     break;
   case '*':
     printf("Multiplication of two numbers is: %d", number1*number2);
     break;
   case '/':
     printf("Division of two numbers is: %d", number1/number2);
     break;
   case '%':
     printf("Module of two numbers is: %d",number1%number2);
     break;
   default:
     printf("You have entered Wrong operator\n");
     printf("Please enter the Correct operator such as +, -, *, /, %%");
     break;
```



OUTPUT 2: Let us enter the wrong operator to check the default value



Sample Programs:

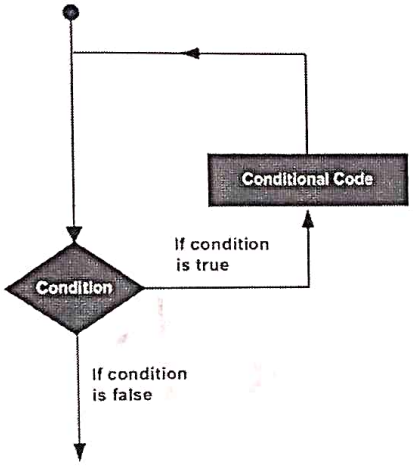
1. Calculate different arithmetic operations using switch case
2. Program for calculating areas of different geometrical objects using switch case.
3. Write menu driven program for options 1) Number is even or odd 2) find largest number

EXPERIMENT NO: 07

Title: Loop Control Statements(for loop)
Aim: To implement programs using loop control statement (for loop)

Theory:

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages –



For loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

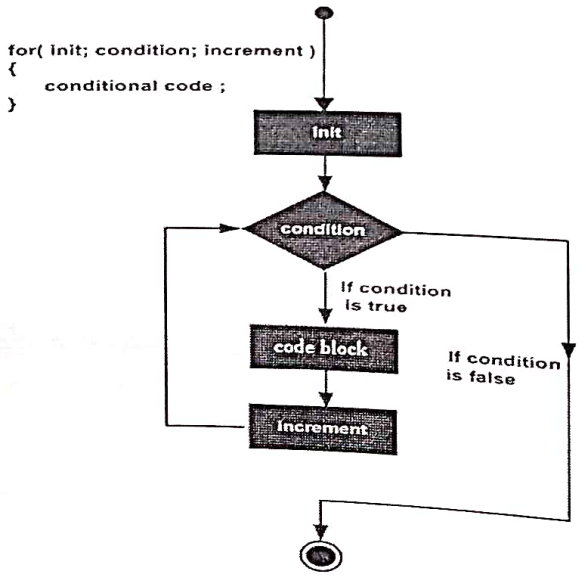
The syntax of a **for** loop in C programming language is –

```
for ( init; condition; increment ) {
   statement(s);
}
```

Here is the flow of control in a 'for' loop –

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Flow Diagram



CProgramming 〉 CProgramming 〉 c main.c 〉 No Selection

```c
/* For Loop in C Programming Example */
#include <stdio.h>

int main()
{
    int i, number, total= 0;

    printf("\n Please Enter any integer\n");
    scanf("%d", &number);

    for(i=1; i<= number; i++)
    {
        total = total + i;
    }

    printf("\nSum of n natural numbers is: %d\n", total);
    return 0;
}
```

```
 Please Enter any integer
20

Sum of n natural numbers is: 210
```

Sample Programs:

1. Print first n consecutive numbers in sequential and reverse order.
2. Print first 10 even and odd numbers.
3. Calculate factorial of given number using for loop.
4. Check number is perfect or not using for loop.
5. Print Fibonacci series up to n number.

EXPERIMENT NO: 08

Title: Loop control statements (while loop, do while loop).

Aim: To implement programs using loop control statements (while loop, do while loop).

Theory:

1. While loop

The most basic loop in C is the while loop.A while statement is like a repeating if statement. Like an If statement, if the test condition is true: the statments get executed. The difference is that after the statements have been executed, the test condition is checked again. If it is still true the statements get executed again.This cycle repeats until the test condition evaluates to false.

Basic syntax of while loop is as follows:

```
while ( expression )
{
  Single statement
  or
  Block of statements;
}
```

2. do...while loop

do ... while is just like a while loop except that the test condition is checked at the end of the loop rather than the start. This has the effect that the content of the loop are always executed at least once.

Basic syntax of do...while loop is as follows:

```
do
{
  Single statement
  or
  Block of statements;
}while(expression);
```

3. break and continue statements

C provides two commands to control how we loop:

- break -- exit form loop or switch.
- continue -- skip 1 iteration of loop.

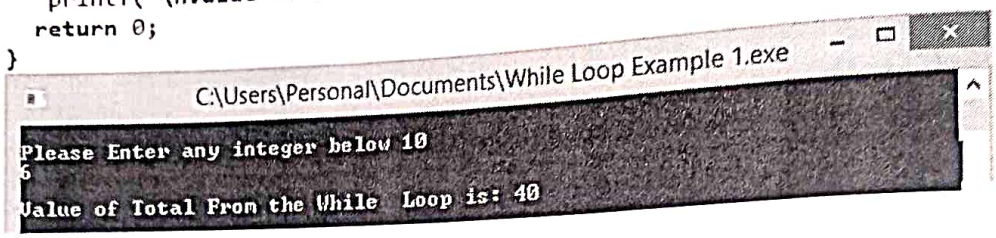
You already have seen example of using break statement. Here is an example showing usage of **continue** statement.

```
#include

main()
{
int i;
int j = 10;

   for( i = 0; i <= j; i ++ )
   {
     if( i == 5 )
     {
       continue;
     }
printf("Hello %d\n", i );
   }
}
```

This will produce following output:

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 6
Hello 7
Hello 8
Hello 9
Hello 10
```

```
#include <stdio.h>
int main()
{
  int number, total=0;
  printf("\nPlease Enter any integer below 10\n");
  scanf("%d", &number);
  while (number <= 10)
   {
     total = total+number;
     number++;
   }
   printf("\nValue of Total From the While  Loop is: %d\n", total);
  return 0;
}
```



Sample Programs:

1. Program to print square of given number.
2. Program to print square and factorial of given number.
3. Program to check number is prime or not.
4. Program to generate prime number up to n.
5. Program to check number is Armstrong or not.

Sample Questions:

1. What is a null statement?
2. How would you decide the use of one of the three loops in C for a given problem?
3. Give a typical example where we find the application of goto necessary?
4. How can we use for loops when the number of iterations are not known?
5. Compare in terms of their functions, between while and do...while loops?
6. In an exit-controlled loop, if the body is executed n times, then the test condition is evaluated how many times?
7. Do you think in a for loop expression, the starting value of the control variable must be less than its ending value?
8. Do you think the use of continue statement is considered as unstructured programming?
9. Write a for statement to print each of the sequences of integers 1, 2, 4, 8, 16, 32?

10. Change the following for loop to while loop for(m = 1;m < 10; m = m + 1) printf(m)

- **LAB ASSIGNMENT:**

1. Write a C program to display the friendly numbers between 1 to N?
2. Write a C program to display the series 1 + 2/2! + 3/3! + 4/4! +.....
3. Write a C program to check whether the given number is palindrome or not?
4. Write a C program to display the right most digit of any integer?
5. Write a C program to read a set of single digits n and convert them into a single decimal number. E.g read a given set of number { 7, 4, 9, 3} and return 7493.
6. Write a C program to generate all combination of numbers using 1, 2, 3?
7. Write a C program to print Pascal triangle?
8. Write a C program to print Floyd's triangle?
9. Write a C program to find LCM and GCD of two given numbers?
10. Write a C program to print the diamond pattern?

EXPERIMENT NO: 09

Title: Functions and program structures.

Aim: To implement program using functions in C.

Theory:

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, **streat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

Defining a Function

The general form of a function definition in C programming language is as follows –

```
return_typefunction_name( parameter list ){
  body of the function
}
```

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function –

- **Return Type** – A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.

- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

- **Function Body** – The function body contains a collection of statements that define what the function does.

Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts –

return_typefunction_name(parameter list);

For the above defined function max(), the function declaration is as follows –

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

int max(int, int);

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

Function Arguments

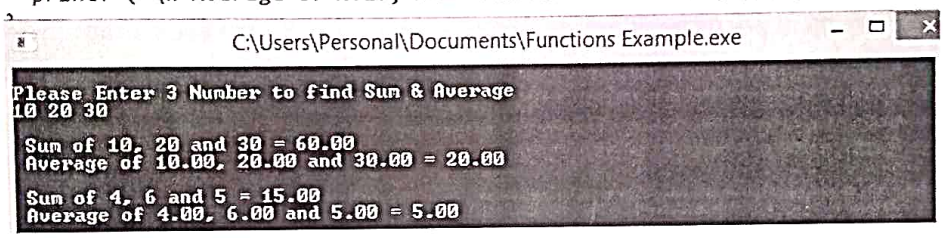
If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function –

S.N.	Call Type & Description
1	**Call by value** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
2	**Call by reference** This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

```
#include<stdio.h>
// Function Declaration
void Average ( float, float, float );
int main( )
{
  float a, b, c;
  int x = 4, y= 6, z =5;
  printf ("\nPlease Enter 3 Number to find Sum & Average \n");
  scanf ( "%f %f %f", &a, &b, &c ) ;
  Average (a, b, c);
  Average (x, y, z);
}
void Average ( float x, float y, float z)
{
   float Sum, Average;
   Sum = x + y + z;
   Average = Sum/3;
   printf ("\n Sum of %.f, %.f and %.f = %.2f",x,y,z, Sum );
   printf ("\n Average of %.2f, %.2f and %.2f = %.2f\n",x,y,z, Average);
```



Sample Programs:

1. Write a function which receive float and int from main(), find the product of these two and returns the product which is printed through main().
2. Write a function that receives 5 integers and of these numbers. Call this function from main() and print results in main()
3. Write a function which receives marks received by student in three subjects and returns average and percentage of these marks. Call the function from main() and returns the result in main().
4. Write a program to calculate area of rectangle by using call by value and call by reference.
5. Program to find GCD of two entered numbers.

Sample Questions:

1. What do you mean by formal and actual parameters in a function?
2. What is upward and downward communication in functions?
3. Can you pass an entire structure into a function?
4. Can you pass an array to a function?
5. What do you mean by function prototype and why it is used?
6. Can you write a function without any parameters?
7. Can you return multiple values from a function using return statement?
8. Is it mandatory to write a return statement in a function?
9. What are the various ways of writing return statement in a function?
10. Is it possible convert all iterative programs into recursion?
11. What is recursion?
12. What is the necessity of creating a user defined function in C?
13. What are the types of functions according to return values and number of arguments?
14. Can you pass a parameter to a function? If yes, then what are the parameter passing techniques?
15. Which data structure is used in recursion?
16. Give any two differences between recursion and iteration?
17. What do mean by base / stopping criteria in recursion? What is its importance?
18. Differentiate between standard functions & user-defined functions?
19. How does the function definition differ from function declaration?
20. In C, if you pass an array as an argument to a function, what actually gets passed?

EXPERIMENT NO: 10

Title: Structures in C
Aim: To study and implement concepts of structure in C.

Theory:

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct[structure tag]{

  member definition;
  member definition;
...
  member definition;
}[one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the

structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure –

```
structBooks{
char  title[50];
char  author[50];
char  subject[100];
intbook_id;
} book;
```

Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program –

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

structBooks{
char  title[50];
char  author[50];
char  subject[100];
intbook_id;
};
int main(){
structBooksBook1;/* Declare Book1 of type Book */
structBooksBook2;/* Declare Book2 of type Book */

/* book 1 specification */
```

```
strcpy( Book1.title, "C Programming");
strcpy( Book1.author, "Nuha Ali");
strcpy( Book1.subject, "C Programming Tutorial");
Book1.book_id = 6495407;
/* book 2 specification */
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Zara Ali");
strcpy( Book2.subject, "Telecom Billing Tutorial");
Book2.book_id = 6495700;
/* print Book1 info */
printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author);
printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);
/* print Book2 info */
printf( "Book 2 title : %s\n", Book2.title);
printf( "Book 2 author : %s\n", Book2.author);
printf( "Book 2 subject : %s\n", Book2.subject);
printf( "Book 2 book_id : %d\n", Book2.book_id);
return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
```

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book_id : 6495700

Structures as Function Arguments

You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

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

structBooks{
char  title[50];
char  author[50];
char  subject[100];
intbook_id;
};
/* function declaration */
voidprintBook(structBooks book );

int main(){

structBooksBook1;/* Declare Book1 of type Book */
structBooksBook2;/* Declare Book2 of type Book */

/* book 1 specification */
strcpy(Book1.title,"C Programming");
strcpy(Book1.author,"Nuha Ali");
strcpy(Book1.subject,"C Programming Tutorial");
Book1.book_id =6495407;
```

```
/* book 2 specification */
strcpy(Book2.title,"Telecom Billing");
strcpy(Book2.author,"Zara Ali");
strcpy(Book2.subject,"Telecom Billing Tutorial");
Book2.book_id =6495700;
/* print Book1 info */
printBook(Book1);
/* Print Book2 info */
printBook(Book2);

return0;
}

voidprintBook(structBooks book ){

printf("Book title : %s\n",book.title);
printf("Book author : %s\n",book.author);
printf("Book subject : %s\n",book.subject);
printf("Book book_id : %d\n",book.book_id);
}
```

When the above code is compiled and executed, it produces the following result –

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
```

Book author : Zara Ali

Book subject : Telecom Billing Tutorial

Book book_id : 6495700

Pointers to Structures

You can define pointers to structures in the same way as you define pointer to any other variable –

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the '&'; operator before the structure's name as follows –

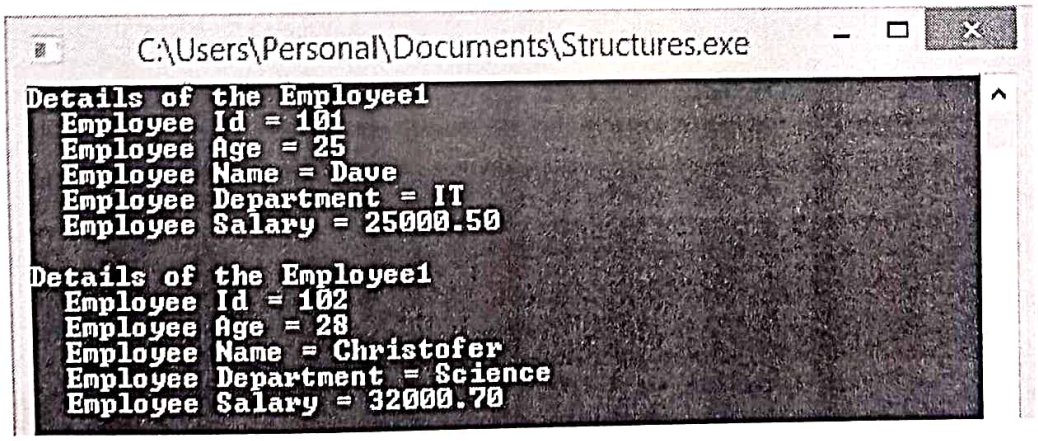
```
struct_pointer=&Book1;
```

To access the members of a structure using a pointer to that structure, you must use the → operator as follows –

```
struct_pointer->title;
```

```
/* Example for Structures in C Programming */
#include <stdio.h>
#include <string.h>

struct Employee
{
  int Empolyee_ID;
  int age;
  char Name[50];
  char Department[20];
  float Salary;
};

int main()
{
  struct Employee emp1 = { 101, 25, "Dave", "IT", 25000.50 };
```

```
struct Employee emp2;

  emp2.Empolyee_ID = 102;
  emp2.age = 28;
  strcpy(emp2.Name, "Christofer");
  strcpy(emp2.Department, "Science");
  emp2.Salary = 32000.70;

 printf(" Details of the Employee1 \n " );
 printf(" Employee Id = %d \n ", emp1.Empolyee_ID );
 printf(" Employee Age = %d \n ", emp1.age );
 printf(" Employee Name = %s \n ", emp1.Name );
 printf(" Employee Department = %s \n ", emp1.Department );
 printf(" Employee Salary = %.2f \n\n ", emp1.Salary );

 printf(" Details of the Employee1 \n " );
 printf(" Employee Id = %d \n ", emp2.Empolyee_ID );
 printf(" Employee Age = %d \n ", emp2.age );
 printf(" Employee Name = %s \n ", emp2.Name );
 printf(" Employee Department = %s \n ", emp2.Department );
 printf(" Employee Salary = %.2f \n ", emp2.Salary );

 return 0;
}
```



Sample Questions:

1. What is a structure ?
2. Differentiate between array and structure ?
3. How do you access the member of a structure ?
4. What is the difference between structure and union?
5. How to access structure?
6. What is the role of structure?

- **LAB ASSIGNMENT:**

1. Write a C Program to Store Information (name, roll and marks) of a Student Using Structure.
2. Write a C Program to Add Two Distances (in inch-feet) System Using Structures.
3. Write a C Program to Calculate Difference between Two Time Period using structure.
4. Read 100 students details from the input using structure and sort the records based student name.
5. Read 100 students marks and find total and percentage of each student and display the details.

EXPERIMENT NO: 11

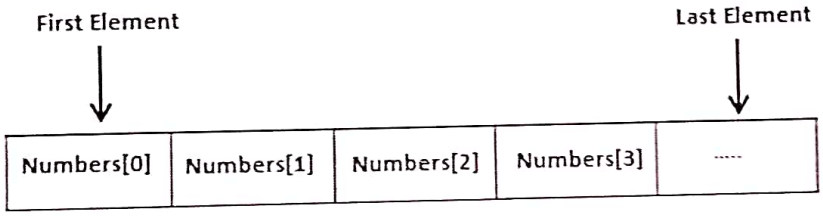
Title: Arrays in C (one dimensional array & Multidimensional array)
Aim: To study and implement problems on array using multidimensional array.

Theory:

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ... numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

type arrayName [arraySize];

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement –

double balance[10];

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows –

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

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

```
balance[4] = 50.0;
```

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above –

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

```
double salary = balance[9];
```

The above statement will take the 10th element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration –

type name[size1][size2]...[sizeN];

For example, the following declaration creates a three dimensional integer array –

intthreedim[5][10][4];

Two-dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows –

type arrayName [x][y];

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows –

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4]={
{0,1,2,3},/*  initializers for row indexed by 0 */
{4,5,6,7},/*  initializers for row indexed by 1 */
{8,9,10,11}/*  initializers for row indexed by 2 */
```

};

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example –

```
intval = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array. You can verify it in the above figure. Let us check the following program where we have used a nested loop to handle a two-dimensional array –

```
#include<stdio.h>
int main ()
{
/* an array with 5 rows and 2 columns*/
int a[5][2]={{0,0},{1,2},{2,4},{3,6},{4,8}};
int i, j;
/* output each array element's value */
for( i =0; i <5; i++){
for( j =0; j <2; j++){
printf("a[%d][%d] = %d\n",i,j, a[i][j]);
}
}
return0;
}
```

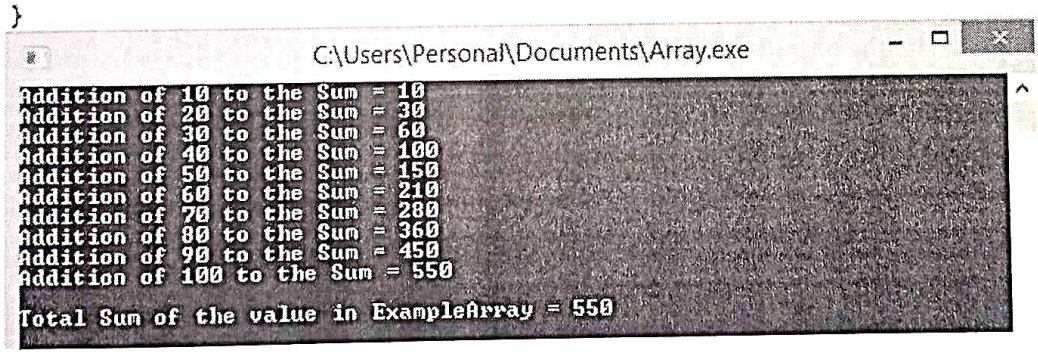
When the above code is compiled and executed, it produces the following result –

```
a[0][0]: 0
```

a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

```
#include<stdio.h>
int main()
{
 int i, Sum =0;
 int ExampleArray[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
 for (i=0; i<10; i++)
 {
  Sum = Sum + ExampleArray[i];
  printf("Addition of %d to the Sum = %d \n", ExampleArray[i], Sum);
 }
 printf("\nTotal Sum of the value in ExampleArray = %d \n", Sum);
 return 0;
}
```



Sample Programs:

1. Program to find maximum and minimum number from n entered number
2. Program to print alternate numbers from n entered numbers.
3. Program to search an element in an array using linear and binary search.
4. Program to print entered number in ascending order using bubble sort and selection sort.
5. Program to print addition, subtraction, and multiplication of matrices.

Sample Questions:

1. **What will happen if in a C program you assign a value to an array element whose subscript exceeds the size of array?**
2. **What does the following declaration mean? int (*ptr)[10];**
3. **What is the output of the following program? int a[6] = {2, 7, 3, 1, 5, 9}; printf("%d",a[5] + 6);**
4. **What will be the output of the following code ?**

```
void main()
{
int a[5]={3,5,6,2,7};
int i=3; if(a[i]==a[i+3])
printf("true"); else
printf("false");
}
```

5. **What will be the output of the following code?**

```
void main()
{
int a[5] = {5,7,3,1,2}; a[2]=a[1];
a[1]=a[3];
a[2]=a[2]+a[3];
printf("%d\t%d", a[1],a[2]);
}
```

6. **Distinguish Lvalue and Rvalue of an array element?**
7. **Write the output of the following code?**

```
void main()
{
int a[3][2] = {10. 20, 30, 40, 50, 60};
printf("%d", a[0][4]);
}
```

8. **Which of the following multi-dimensional array declaration is correct for realizing a 2x3 matrix?**
   a. int m[2][3]
   b. int m[3][2]
   c. int m[3],m[2]
9. **Which of the following is the correct syntax for initialization of two dimensional array?**
   a. table[2][3]={0,0,0,1,1,1};
   b. table[2][3]={ {0,0,0} {1,1,1} };
   c. table[2][3]={0,1},{0,1},{0,1};
10. **Write the output of the following code?**

```
void main()
{
int x[10] = {5}; printf("%3d%3d", x[1], x[9]);
```

}

- **LAB ASSIGNMENT**

1. Read a matrix and find the transpose of the matrix?.
2. Formulate a C-program that makes the following tasks:
   i.Check whether matrix is magic square or not ?
   ii.Print square of each element of 2D array matrix
   iii.Add the elements of lower triangular matrix
   iv.Calculate sum of upper triangular matrix elements
   v.Addition of diagonal elements in matrix
   vi.Addition of all elements in matrix
   vii.Find the inverse of 3 x 3 matrix
3. Write a C program to reverse the contents of an array?
4. Given an array of n elements. Read n elements into array. Formulate a C-program for finding smallest element in array using pointers.
5. Write a C program to find the determinant of a matrix.

EXPERIMENT NO: 12

Title: Strings in C
Aim: To study and implement concepts of strings in C.

Theory:

Strings are actually one-dimensional array of characters terminated by a **null**character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

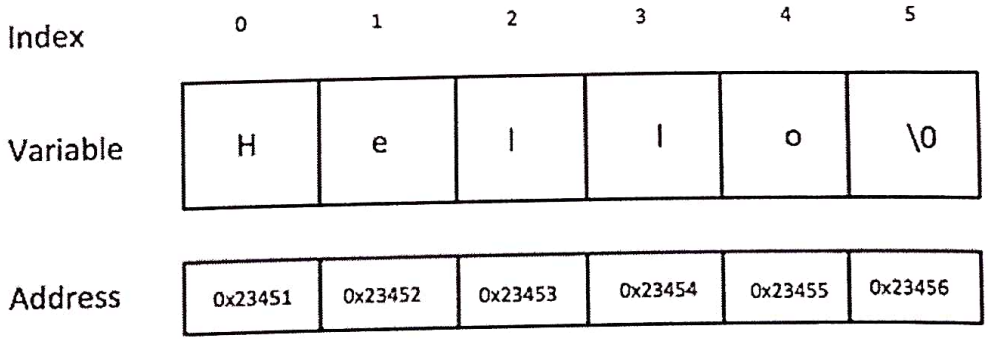
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows –

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++ –



Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string –

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

```
int main (){

char greeting[6]={'H','e','l','l','o','\0'};
printf("Greeting message: %s\n", greeting );
return0;
}
```

When the above code is compiled and executed, it produces the following result –

Greeting message: Hello

C supports a wide range of functions that manipulate null-terminated strings –

S.N.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

5 **strchr(s1, ch);**

Returns a pointer to the first occurrence of character ch in string s1.

6 **strstr(s1, s2);**

Returns a pointer to the first occurrence of string s2 in string s1.

```c
/* C String Example */
#include <stdio.h>

int main(int argc, const char * argv[])
{
    char name[50];
    int i = 0;

    printf("Please enter the Name : ");
    scanf("%s", name);

    while (name[i] != '\0')
    {
        printf("The Character at %d Index Position = %c \n", i, name[i]);
        i++;
    }
    return 0;
}
```

```
Please enter the Name : hello
The Character at 0 Index Position = h
The Character at 1 Index Position = e
The Character at 2 Index Position = l
The Character at 3 Index Position = l
The Character at 4 Index Position = o
```

Sample Programs:

1. Program to find length of string. (with and without using function)
2. Program to demonstrate all string related library functions.
3. Program to concatenate two strings without using library functions.
4. Program to reverse a given string and check the entered string is palindrome or not

Sample Questions:

1. Using which function a string "1234" can be converted to a number?
2. Name the different string handling functions?
3. Which command is used to combine the two strings?
4. Which command is used to copy the strings?
5. What is the difference between string copy strcpy() and memory copy memcpy()?
6. How can you copy just a portion of a string?
7. Using which function you can find a substring from a string?
8. Which function is used to compare two strings?
9. How can you remove trailing spaces from a string?
10. How can you remove leading spaces from a string?

- **LAB ASSIGNMENT:**

1. Code a program to convert the given string into lower case to upper case and vice-versa?
2. Read two strings and store them in variables str1, str2. Formulate a C program for finding whether the both strings are equal or not?
3. Program for finding the length of the given string "COMPUTER PROGRAMMING" with and without using string length function?
4. Read two strings and store them in variables str1, str2. Formulate a C program for copying 2 strings to a new variable str3?
5. Formulate a program for concatenating the following two strings with and without using strcat() function. Str1 is "COMPUTER", str2 is "PROGRAMMING".
6. Formulate a program for reversing the given string 'REVERSE' with and without using stirng reverse function?
7. Formulate a program for determining whether the given string is palindrome or not?
8. Read two strings, str1 as "goodday" and str2 as "noontime". By using these strings generate a new string str3 as "goodnoon". Now compare str3 with str1 to find the greatest?
9. Accept two strings and compare one string with another string without using string handling functions and find two strings are equal or not.
10. Read a string and delete the duplicate character in the string.