

NAME: SHALMON N. ANANDAS

CLASS: M.Sc. PART – I

COURSE: BIOINFORMATICS

ACADEMIC YEAR: 2021-2022

ROLL NO: 91

PAPER CODE: GNKPSBI104 (PAPER 4)

COURSE TITLE: FUNDAMENTALS OF BIOLOGY

GURU NANAK KHALSA COLLEGE
MATUNGA, MUMBAI – 400019

DEPARTMENT OF BIOINFORMATICS

CERTIFICATE

This is to certify that Mr. **Shalmon N Anandas** of M.Sc. Part I Bioinformatics has satisfactorily completed the practical semester I course prescribed by the university of Mumbai during the academic year 2021-2022

TEACHER IN CHARGE

HEAD OF DEPARTMENT

INDEX

SR. NO	WEBLEM	PAGE NO	DATE	SIGNATURE
1	Basic I/O programs	4	6/12/21	
2	Conditional Statement and loop	8	6/12/21	
3	Arrays	15	6/12/21	
4	Functions and structure	21	6/12/21	
5	String manipulation	23	6/12/21	
6	Class and objects, encapsulation [private, public and protected]	26	6/12/21	
7	Inheritance	28	6/12/21	
8	Polymorphism, Virtual Function, Friend Function	32	6/12/21	
9	Constructors and Destructor	35	6/12/21	

PRACTICAL 1 – Basic I/O Programs

AIM:

To write basic I/O Programs

Theory:

Introduction: C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's. C++ is a superset of C. Most of what we already know about C applies to C++ also. Therefore, almost all C programs are also C++ programs. However, there are a few minor differences that will prevent a C program to run under C++ compiler.

Advantages: Portability, Mid-level programming language, Object-Oriented, Multi-paradigm programming language, Memory management.

Disadvantages: Pointers, No garbage collection, unsafe, Complex, No custom operators, No built-in threads, lack of algebraic data types.

Applications: C++ is widely used in operating systems, browsers, libraries, banking applications, Cloud systems, many databases and compilers for other languages also use C++ as backend, Embedded systems.

Header file: Header files contain definitions of Functions and variables which is imported or used into any C++ program.

Functions: A function in C++ is a group of statements that together perform a specific task. Every C/C++ program has at least one function that the name is main. The main function is called by the operating system by which our code is executed. We can make n number of function in a single program but we can make only one main function in a single program. Every program has only one main function.

Datatypes:

- **Built-in:**

- **Integral Type**

- Int
 - Char

- **Floating type**

- Float
 - Double

- **User-Defined:**

- Structure
 - Union
 - Class
 - Enumeration

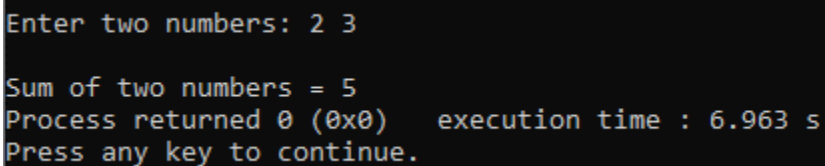
- **Derived type:**
 - Array
 - Function
 - Pointer
 - Reference

Q1: Write a program in C++ to print the sum of two numbers entered by user.

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;
    cout << "\nEnter two numbers: ";
    cin >> a >> b;
    c = a + b;
    cout << "\nSum of two numbers = " << c;
    return 0;
}
```

OUTPUT:



```
Enter two numbers: 2 3

Sum of two numbers = 5
Process returned 0 (0x0)   execution time : 6.963 s
Press any key to continue.
```

Fig1. Output for C++ program to print the sum of two numbers entered by user.

Q2. Write a program in C++ to find size of fundamental data types

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "\nSize of int is " << sizeof(int);
    cout << "\nSize of float is " << sizeof(float);
    cout << "\nSize of char is " << sizeof(char);
    cout << "\nSize of double is " << sizeof(double);
}
```

OUTPUT:

```
Size of int is 4
Size of int is 4
Size of int is 1
Size of int is 8
Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

Fig2. Output for C++ program to find size of fundamental data types.

Q3. Write C++ program to swap two numbers entered by user.

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, temp;
    cout << "\nEnter value of a and b: ";
    cin >> a >> b;
    cout << "\nBefore swap a = " << a << " and b = " << b;
    temp = a;
    a = b;
    b = temp;
    cout << "\nAfter swap a = " << a << " and b = " << b;
}
```

OUTPUT:

```
Enter value of a and b: 10 20

Before swap a = 10 and b = 20
After swap a = 20 and b = 10
Process returned 0 (0x0)   execution time : 3.677 s
Press any key to continue.
```

Fig3. Output for C++ program to swap two numbers entered by user

Q4. Write C++ program to calculate the volume of a cube

CODE:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    int a, b;
    cout << "\nEnter side of a cube: ";
    cin >> a;
    b = pow(a, 3);
    cout << "\nVolume of cube = " << b;
}
```

OUTPUT:

```
Enter side of a cube: 5  
Volume of cube = 125  
Process returned 0 (0x0)   execution time : 1.620 s  
Press any key to continue.
```

Fig4. Output for C++ program to calculate the volume of a cube

PRACTICAL 2: Conditional and loops

AIM:

To write programs using conditionals and loops

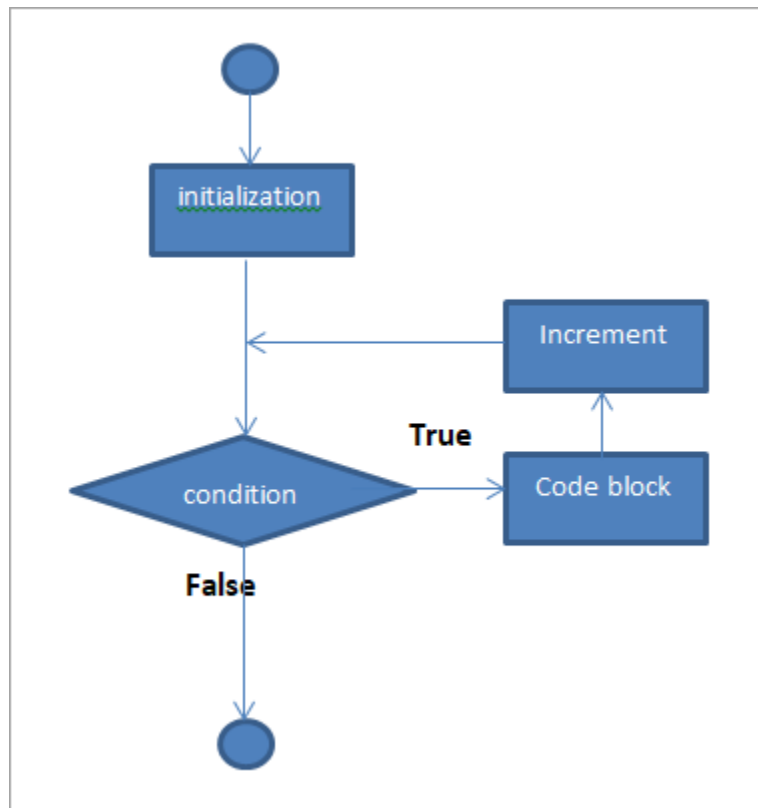
Theory:

Loops:

- **Syntax:**

```
for (initial value; test; increment)
{
    action1;
}
action2;
```

- **Diagram:**

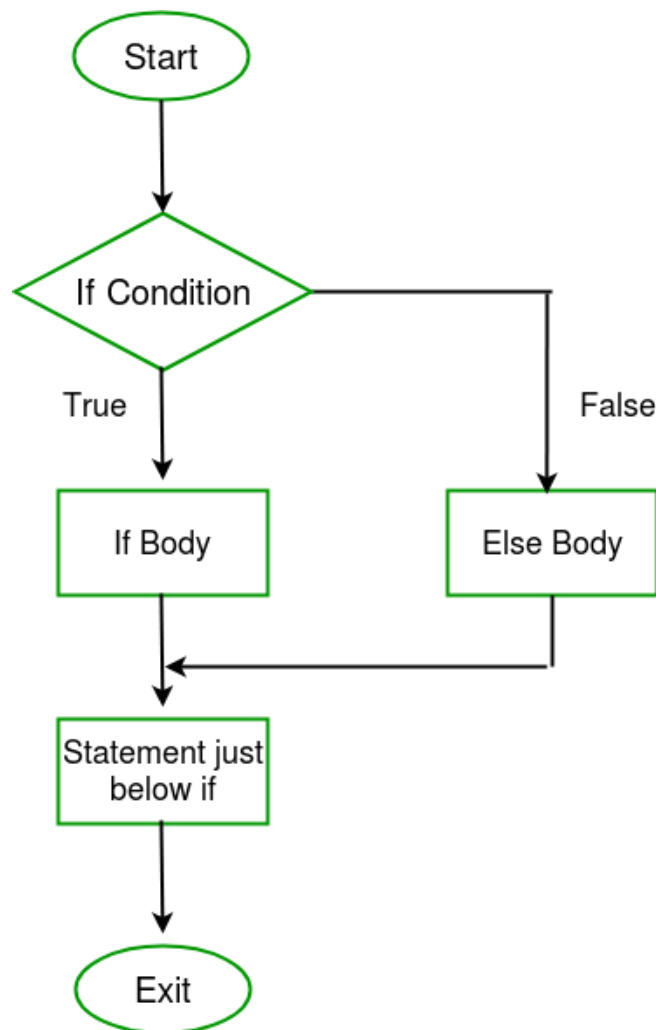


Conditionals:

- **Syntax:**

```
if (expression is true)
{
    action1;
}
else
{
    action2;
}
action3;
```

- **Diagram:**

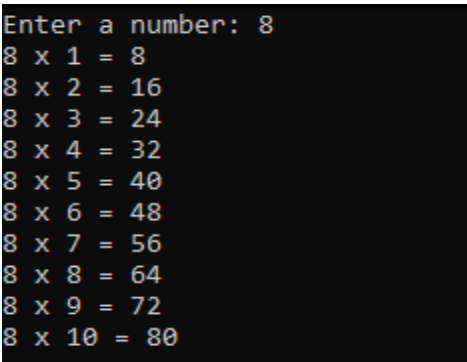


Q1. Write a program in C++ that takes a number as input and prints its multiplication table up to 10.

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int num, mul;
    cout << "Enter a number: ";
    cin >> num;
    for (int i=1; i<=10; i++)
    {
        mul = num * i;
        cout << num << " x " << i << " = " << mul << endl;
    }
}
```

OUTPUT:



The screenshot shows the output of the C++ program. It starts with the prompt "Enter a number: 8". Below this, it displays the multiplication table for the number 8, from 8 x 1 to 8 x 10. The output is as follows:

8	x	1	=	8
8	x	2	=	16
8	x	3	=	24
8	x	4	=	32
8	x	5	=	40
8	x	6	=	48
8	x	7	=	56
8	x	8	=	64
8	x	9	=	72
8	x	10	=	80

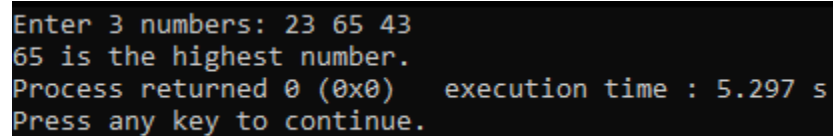
Fig1. Output for C++ program to print its multiplication table up to 10

Q2. Write a C++ program which prints three highest numbers from entered 3 numbers

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c, highest;
    cout << "Enter 3 numbers: ";
    cin >> a >> b >> c;
    if (a > b && b > c)
    {
        cout << a << " is the highest number.";
    }
    else if (b > c)
    {
        cout << b << " is the highest number.";
    }
    else
    {
        cout << c << " is the highest number.";
    }
}
```

OUTPUT:



```
Enter 3 numbers: 23 65 43
65 is the highest number.
Process returned 0 (0x0)   execution time : 5.297 s
Press any key to continue.
```

Fig2. Output for C++ program which prints highest number out of 2 given numbers.

Q3. Write a C++ program to compute the sum of even numbers

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int num, sum = 0;
    cout << "Enter a number: ";
    cin >> num;
    cout << "Numbers to be added: ";
    for (int i=1; i<=num; i++)
    {
        if (i % 2 == 0)
        {
            cout << i << " ";
            sum = sum + i;
        }
    }
    cout << "\nTotal of all numbers is: " << sum;
}
```

OUTPUT:

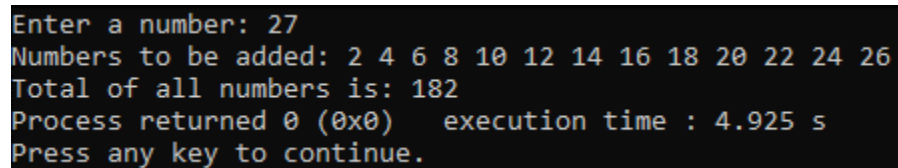
A screenshot of a terminal window showing the output of the C++ program. The text is as follows:
Enter a number: 27
Numbers to be added: 2 4 6 8 10 12 14 16 18 20 22 24 26
Total of all numbers is: 182
Process returned 0 (0x0) execution time : 4.925 s
Press any key to continue.

Fig3. Output of C++ program to compute the sum of even numbers

Q4. Write a C++ program to accept side of a triangle and display equilateral, isosceles and scalene.

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;
    cout << "Enter sides of a triangle: ";
    cin >> a >> b >> c;
    if (a == b && b == c)
    {
        cout << "It is an Equilateral triangle";
    }
    else if (a == b || b == c || a == c)
    {
        cout << "It is an isosceles triangle";
    }
    else
    {
        cout << "It is a scalar triangle";
    }
}
```

OUTPUT:

```
Enter sides of a triangle: 5 5 8
It is an isoceles triangle
Process returned 0 (0x0)   execution time : 5.822 s
Press any key to continue.
```

Fig4. Output with the program identifying Isosceles triangle

```
Enter sides of a triangle: 5 5 5
It is an Equilateral triangle
Process returned 0 (0x0)   execution time : 1.930 s
Press any key to continue.
```

Fig5. Output of the program identifying equilateral triangle

```
Enter sides of a triangle: 5 6 7
It is a scalene traingle
Process returned 0 (0x0)   execution time : 1.368 s
Press any key to continue.
```

Fig6. Output of the program identifying scalene triangle

PRACTICAL 3: Arrays

AIM:

To write C++ programs demonstrating arrays.

Theory:

One-dimensional array: One dimensional array is also known as a list or a linear array. It consists of only one column or one row.

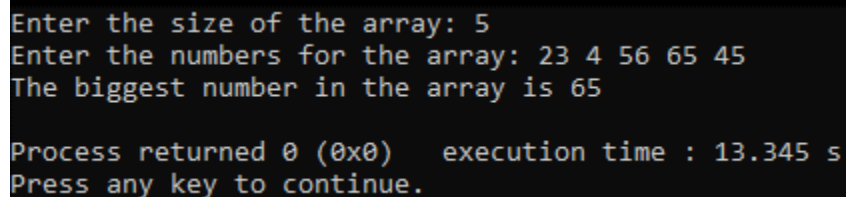
Two-dimensional array: A two-dimensional array consists of columns and rows. Each element of the two-dimensional array is referenced by its index values or subscripts. The index value of a two-dimensional array consists of two subscripts. One subscript represents the row number and the second subscript represents the column number.

Q1. Write a C++ program to find the largest element of a given array of integers

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int n, i, max;
    cout << "Enter the size of the array: ";
    cin >> n;
    int array[n];
    cout << "Enter the numbers for the array: ";
    for (i = 0; i < n; i++)
    {
        cin >> array[i];
    }
    max = array[0];
    for (i = 0; i < n; i++)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }
    cout << "The biggest number in the array is " << max << endl;
}
```

OUTPUT:



```
Enter the size of the array: 5
Enter the numbers for the array: 23 4 56 65 45
The biggest number in the array is 65

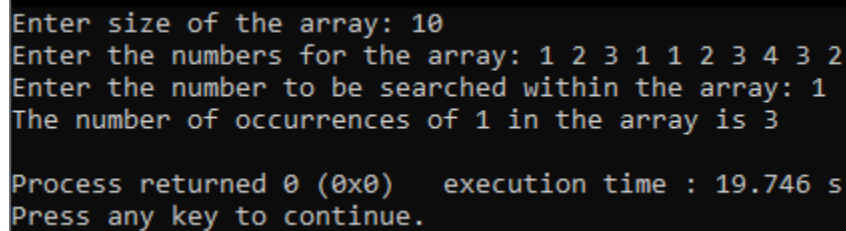
Process returned 0 (0x0)   execution time : 13.345 s
Press any key to continue.
```

Fig1. Output of program showing biggest number out of an array of given numbers

Q2. Write a C++ program to count the number of occurrences of a given number in an array of integers

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int n, i, query, count = 0;
    cout << "Enter size of the array: ";
    cin >> n;
    int array[n];
    cout << "Enter the numbers for the array: ";
    for (i = 0; i < n; i++)
    {
        cin >> array[i];
    }
    cout << "Enter the number to be searched within the array: ";
    cin >> query;
    for (i = 0; i < n; i++)
    {
        if (array[i] == query)
        {
            count++;
        }
    }
    cout << "The number of occurrences of " << query << " in the array is " << count << endl;
}
```

OUTPUT:A screenshot of a terminal window showing the output of the C++ program. The text is as follows:
Enter size of the array: 10
Enter the numbers for the array: 1 2 3 1 1 2 3 4 3 2
Enter the number to be searched within the array: 1
The number of occurrences of 1 in the array is 3

Process returned 0 (0x0) execution time : 19.746 s
Press any key to continue.

Fig2. Output of program showing number of occurrences of a specified number in an array

Q3. Write a C++ program to display even number from an entered array**CODE:**

```
#include <iostream>
using namespace std;
int main()
{
    int n, i;
    cout << "Enter the size of the array: ";
    cin >> n;
    int array[n];
    cout << "Enter numbers in the array: ";
    for (i = 0; i < n; i++)
```

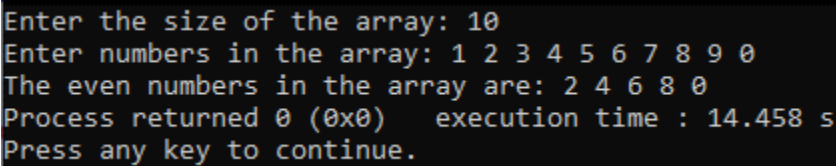


```

{
    cin >> array[i];
}
cout << "The even numbers in the array are: ";
for (i = 0; i < n; i++)
{
    if (array[i] % 2 == 0)
    {
        cout << array[i] << " ";
    }
}
}

```

OUTPUT:



```

Enter the size of the array: 10
Enter numbers in the array: 1 2 3 4 5 6 7 8 9 0
The even numbers in the array are: 2 4 6 8 0
Process returned 0 (0x0)   execution time : 14.458 s
Press any key to continue.

```

Fig3. Output of program showing even numbers out of an array

Q4. Write a C++ program to accept 3x3 matrix and display the transpose of a given matrix

CODE:

```

#include <iostream>
using namespace std;
int main()
{
    int i, j;
    int matrix[3][3];
    cout << "Enter the value for the 3x3 matrix::\n";
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cin >> matrix[i][j];
        }
    }
    cout << "The transpose of the matrix is:\n";
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cout << matrix[j][i] << " ";
        }
        cout << endl;
    }
}

```

OUTPUT:

```
Enter the value for the 3x3 matrix::  
1 3 5  
2 34 65  
21 43 56  
The transpose of the matrix is:  
1 2 21  
3 34 43  
5 65 56  
  
Process returned 0 (0x0)   execution time : 10.882 s  
Press any key to continue.
```

Fig4. Output of transpose of a given matrix

Q5. Write a C++ program to accept two 3x3 matrix and display sum of the two matrices

CODE:

```
#include <iostream>
using namespace std;
int main()
{
    int i, j;
    int matrix1[3][3];
    int matrix2[3][3];
    int sum[3][3];
    cout << "Enter the value for the first 3x3 matrix::\n";
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cin >> matrix1[i][j];
        }
    }
    cout << "Enter the value for the second 3x3 matrix::\n";
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cin >> matrix2[i][j];
        }
    }
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
    cout << "The sum of the two matrices is:\n";
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cout << sum[i][j] << " ";
        }
        cout << endl;
    }
}
```

OUTPUT:

```
Enter the value for the first 3x3 matrix::  
1 2 3  
4 5 6  
7 8 9  
Enter the value for the second 3x3 matrix::  
1 2 3  
4 5 6  
7 8 9  
The sum of the two matrices is:  
2 4 6  
8 10 12  
14 16 18  
  
Process returned 0 (0x0)   execution time : 18.032 s  
Press any key to continue.
```

Fig5. Output of program showing sum of two given matrices

PRACTICAL 4: Functions and Structure

AIM:

To write programs demonstrating functions and structure

THEORY:

Functions: A function is a subprogram or module to which any amount of data can be sent but which returns only one value. A function is used to perform some logically isolated tasks. It makes it easier to write programs and keep track of what they are doing. It avoids rewriting the same series of instructions over and over. These instructions are written once but are used at several places in the main program as and when desired.

Types of user defined function:

1. Function with no argument no return value
2. Function with no argument with return value
3. Function with an argument with no return value
4. Function with an argument with return value

Q1. Write a C++ program to create a function named as area and find area of triangle using no return and no parameter

CODE:

```
#include <iostream>

using namespace std;

//function to calculate area of triangle using no return and no parameters
void area_of_triangle()
{
    float base, height;
    cout << "Enter the base and height of the triangle: ";
    cin >> base >> height;
    cout << "The area of the triangle is: " << (base * height) / 2 << endl;
}

//main function
int main()
{
    area_of_triangle();
}
```

OUTPUT:

```
Enter the base and height of the triangle: 5 7
The area of the triangle is: 17.5

Process returned 0 (0x0)   execution time : 3.797 s
Press any key to continue.
```

Fig1. Output of program showing the are of triangle after taking base and height as input

Q2. Write a C++ program to create function for adding two numbers with return and with parameter

CODE:

```
#include <iostream>
using namespace std;
int add(int a, int b)
{
    return a + b;
}
int main()
{
    int a, b;
    cout << "Enter the two numbers to add: ";
    cin >> a >> b;
    cout << "The sum of the two numbers is: " << add(a, b) << endl;
}
```

OUTPUT:

```
Enter the two numbers to add: 5 10
The sum of the two numbers is: 15

Process returned 0 (0x0)   execution time : 6.929 s
Press any key to continue.
```

Fig2. Output of program showing addition of two numbers

PRACTICAL 5: String manipulation

Aim:

To write C++ program demonstrating string manipulation

THEORY:

String manipulation: C++ stores a string as an array of characters. An array is a group of contiguous memory location that can be stored same type of data. So, the string declaration is same as array declaration.

String functions: String functions present in the string header file are STRCMP() [String compare], STRCPY() [String copy], STRLEN() [String length], STRCAT() [String concatenate]. How these functions work will be demonstrated below in the program.

Q1a. Find length of a string

CODE:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string str;
    cout << "Enter a string: ";
    cin >> str;
    cout << str.length();
    return 0;
}
```

OUTPUT:

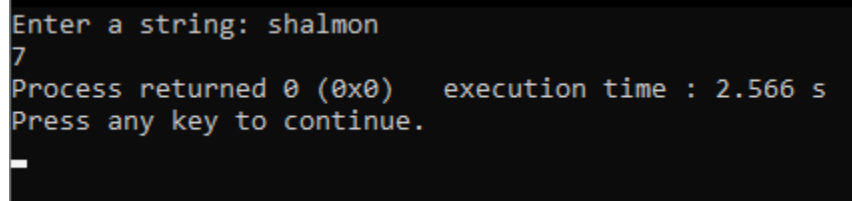
A screenshot of a terminal window with a dark background. It shows the output of the C++ program for Q1a. The first line is 'Enter a string: shalmon'. The second line is '7'. The third line is 'Process returned 0 (0x0) execution time : 2.566 s'. The fourth line is 'Press any key to continue.' followed by a cursor.

Fig1. Output of program showing length of the given string

Q1b. Copy a string into another

CODE:

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
    char str[100];
    char str1[100];
    cout << "Enter a string: ";
    cin >> str;
    strcpy (str1, str);
    cout << "The copied string is: " << str1 << endl;
}
```

OUTPUT:

```
Enter a string: shalmon
The copied string is: shalmon

Process returned 0 (0x0)   execution time : 1.065 s
Press any key to continue.
```

Fig2. Output of program showing copied string

Q1c. Combine two strings

CODE:

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
    char str[100];
    char str1[100];
    cout<<"Enter the first string: ";
    cin.getline(str,100);
    cout<<"Enter the second string: ";
    cin.getline(str1,100);
    strcat(str,str1);
    cout<<"The combined string is: "<<str;
}
```

OUTPUT:

```
Enter the first string: shalmon
Enter the second string: anandas
The combined string is: shalmonanandas
Process returned 0 (0x0)   execution time : 4.121 s
Press any key to continue.
```

Fig3. Output of program showing two given strings combined

Q1d. Compare whether entered two strings are equal

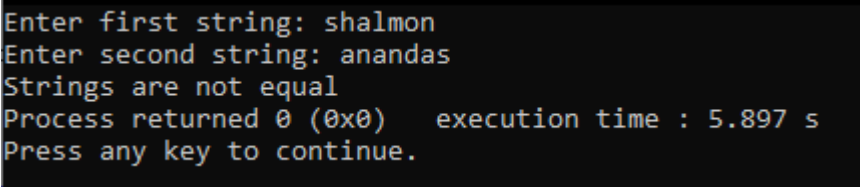
CODE:

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
    char str1[100], str2[100];
    int result;
    cout << "Enter first string: ";
    cin.getline(str1, 100);
    cout << "Enter second string: ";
    cin.getline(str2, 100);
    result = strcmp(str1, str2);
    switch(result)
```



```
{  
    case 0:  
        cout << "Strings are equal";  
        break;  
    case 1:  
        cout << "Strings are not equal";  
        break;  
}
```

OUTPUT:

A screenshot of a terminal window with a black background and white text. The text shows the program's execution: it prompts for the first string, then the second string, compares them, and outputs the result. It also shows the process return code and execution time.

```
Enter first string: shalmon  
Enter second string: anandas  
Strings are not equal  
Process returned 0 (0x0)   execution time : 5.897 s  
Press any key to continue.
```

Fig4. Output of program comparing if the two strings are equal or not

PRACTICAL-6: CLASS AND OBJECTS, ENCAPSULATIONS (private, public and protected)

AIM:

To understand the class and objects, encapsulation (private, public and protected).

THEORY:

Class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

Objects is a real world entity, for example, chair, car, pen, mobile, laptop etc. Object is an instance of a class. All the members of the class can be accessed through object.

Encapsulation refers to the bundling of data, along with the methods that operate on that data, into a single unit.

Access modifiers are used to implement an important aspect of Object-Oriented Programming known as Data Hiding.

There are 3 types of access modifiers available in C++:

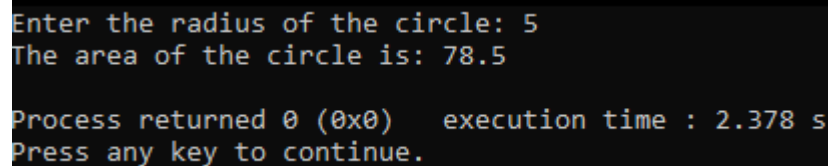
1. Public [everyone can access outside the class]
2. Private [no one can access outside the class]
3. Protected [only child can access outside the class]

Q.1) Write a program to create a class area to calculate the area of a circle using private variables and public functions.

CODE:

```
#include <iostream>
using namespace std;
class circle
{
    private:
        float radius;
    public:
        void set_radius(float r)
        {
            radius = r;
        }
        float get_radius()
        {
            return radius;
        }
        float area()
        {
            return 3.14 * radius * radius;
        }
};
int main()
{
    circle c;
    float r;
    cout << "Enter the radius of the circle: ";
    cin >> r;
    c.set_radius(r);
    cout << "The area of the circle is: " << c.area() << endl;
}
```

OUTPUT:

A screenshot of a terminal window showing the output of the C++ program. The text is as follows:
Enter the radius of the circle: 5
The area of the circle is: 78.5

Process returned 0 (0x0) execution time : 2.378 s
Press any key to continue.

Fig1. Output of program showing area of cricle

PRACTICAL 7: INHERITANCE

AIM:

To understand the inheritance.

THEORY:

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

TYPES OF INHERITANCE:

Single Inheritance: When a Derived Class to inherit properties and behavior from a single Base Class, it is called as single inheritance.

Multi-level Inheritance: A derived class is created from another derived class is called Multi Level Inheritance.

Hierarchical Inheritance: More than one derived classes are created from a single base class, is called Hierarchical Inheritance.

Multipath Inheritance: Multiple inheritance is a method of inheritance in which one derived class can inherit properties of base class in different paths. This inheritance is not supported in .NET Languages such as C#.

Multiple Inheritance: Multiple inheritances allows programmers to create classes that combine aspects of multiple classes and their corresponding hierarchies.

Q.1) Write a program to perform a single inheritance to calculate area of circle.

CODE:

```
#include <iostream>
using namespace std;
class Radius
{
    public:
        int radius;
        void getRadius()
        {
            cout<<"Enter the radius of circle: ";
            cin>>radius;
        }
};
class Area : public Radius
{
    public:
        void getArea()
        {
            cout<<"Area of circle is: "<<3.14*radius*radius<<endl;
        }
};
int main()
{
    Area a;
    a.getRadius();
```

```

a.getArea();
return 0;
}

```

OUTPUT:

```

Enter the radius of circle: 7
Area of circle is: 153.86

Process returned 0 (0x0)   execution time : 1.155 s
Press any key to continue.

```

Fig1. Output of program showing area of circle demonstrating single inheritance

Q.2) Write a program to perform multilevel inheritance student details, course details, marks.

CODE:-

```

#include <iostream>
using namespace std;
class Student
{
    private:
        char name[50];
        int id;
    public:
        int student()
        {
            cout<<"Enter student name: ";
            cin.getline(name,50);
            cout<<"Enter student id: ";
            cin>>id;
        }
        void display_student()
        {
            cout<<"\n\nStudent name: "<<name<<endl;
            cout<<"Student id: "<<id;
        }
};
class Marks : public Student
{
    private:
        int marks[5];
        int total;
        float avg;

    public:
        int student_marks()
        {
            for(int i=0; i<5; i++)
            {

```

```

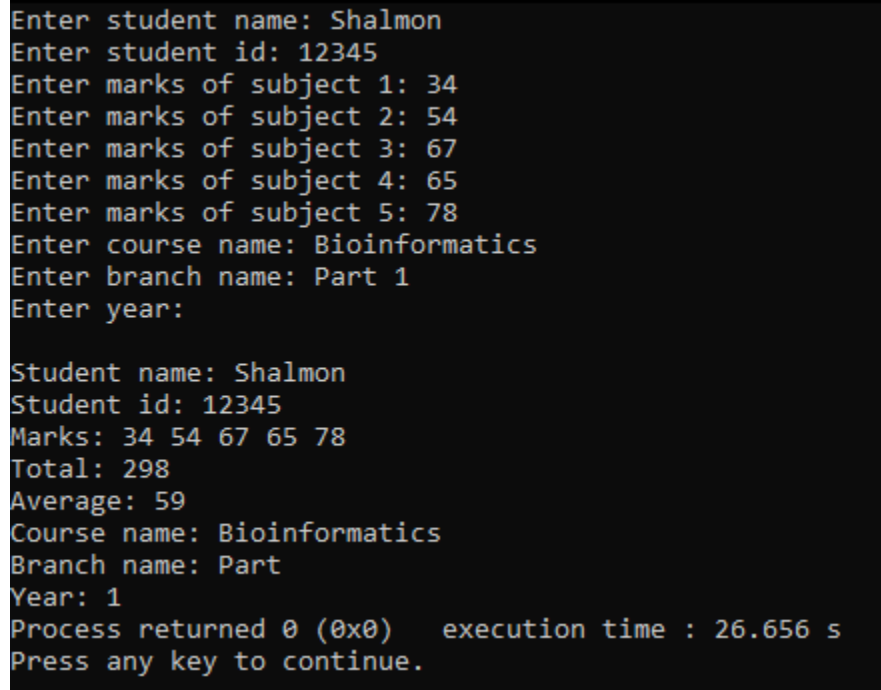
        cout<<"Enter marks of subject "<<i+1<<": ";
        cin>>marks[i];
    }
    total=0;
    for(int i=0; i<5; i++)
    {
        total=total+marks[i];
    }
    avg=total/5;
}
void display_student_marks()
{
    cout<<"\nMarks: ";
    for(int i=0; i<5; i++)
    {
        cout<<marks[i]<<" ";
    }
    cout<<"\nTotal: "<<total;
    cout<<"\nAverage: "<<avg;
}
};
class Course : public Marks
{
private:
    char course[50];
    char branch[50];
    int year;

public:
    int student_course()
    {
        cout<<"Enter course name: ";
        cin>>course;
        cout<<"Enter branch name: ";
        cin>>branch;
        cout<<"Enter year: ";
        cin>>year;
        return 0;
    }
    void display_student_course()
    {
        cout<<"\nCourse name: "<<course;
        cout<<"\nBranch name: "<<branch;
        cout<<"\nYear: "<<year;
    }
};
int main()
{

```

```
Course c;  
c.student();  
c.student_marks();  
c.student_course();  
c.display_student();  
c.display_student_marks();  
c.display_student_course();  
}
```

OUTPUT:



```
Enter student name: Shalmon  
Enter student id: 12345  
Enter marks of subject 1: 34  
Enter marks of subject 2: 54  
Enter marks of subject 3: 67  
Enter marks of subject 4: 65  
Enter marks of subject 5: 78  
Enter course name: Bioinformatics  
Enter branch name: Part 1  
Enter year:  
  
Student name: Shalmon  
Student id: 12345  
Marks: 34 54 67 65 78  
Total: 298  
Average: 59  
Course name: Bioinformatics  
Branch name: Part  
Year: 1  
Process returned 0 (0x0)   execution time : 26.656 s  
Press any key to continue.
```

Fig2. Output of program showing student information demonstrating multilevel inheritance

PRACTICAL 8: Polymorphism, Virtual Function, Friend Function

AIM:

To understand the Polymorphism, Virtual Function, Friend Function.

THEORY:

The word **polymorphism** means having many forms. The ability of a message to be displayed in more than one form.

A **virtual function** is a member function that you expect to be redefined in derived classes.

Friend function is given the same access as methods to private and protected data.

TYPES OF POLYMORPHISM:

Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.

Runtime polymorphism: This type of polymorphism is achieved by function overriding.

Q.1) Write a program to perform addition of two and three numbers using method overloading.

CODE:

```
#include <iostream>
using namespace std;
class Addition
{
    public:
        //function to add two numbers
        int add(int a, int b)
        {
            return a+b;
        }
        //function to add three numbers
        int add(int a, int b, int c)
        {
            return a+b+c;
        }
};
int main()
{
    Addition obj;
    int a, b, c;
    cout<<"Enter two numbers: ";
    cin>>a>>b;
    cout<<"Sum of two numbers is: "<<obj.add(a,b)<<endl;
    cout<<"Enter three numbers: ";
    cin>>a>>b>>c;
    cout<<"Sum of three numbers is: "<<obj.add(a,b,c)<<endl;
}
```

OUTPUT:


```
Enter two numbers: 5 6
Sum of two numbers is: 11
Enter three numbers: 5 6 7
Sum of three numbers is: 18

Process returned 0 (0x0)   execution time : 7.178 s
Press any key to continue.
```

Fig1. Output showing addition of two and three numbers using method overloading

Q.2) Write a program to demonstrate method overriding using virtual function.

CODE:

```
#include <iostream>
using namespace std;
class Base
{
    public:
        virtual void print()
        {
            cout << "print base class on compile" << endl;
        }
        void display()
        {
            cout << "display base class on compile" << endl;
        }
};
class Derived : public Base
{
    public:
        void print()
        {
            cout << "print derived class on runtime" << endl;
        }
        void display()
        {
            cout << "display derived class on runtime" << endl;
        }
};
int main()
{
    Base* bptr;
    Derived d;
    bptr = &d;

    bptr->display();
    bptr->print();
}
```

OUTPUT:

```
display base class on compile  
print derived class on runtime  
  
Process returned 0 (0x0)   execution time : 0.044 s  
Press any key to continue.
```

Fig2. Output of program showing method overriding using virtual function

PRACTICAL 9: CONSTRUCTOR AND DESTRUCTOR

AIM:

To understand constructor and destructor.

THEORY:

A **constructor** is a special type of member function of a class which initializes objects of a class.

Syntax:

```
class Wall { public: // create a constructor Wall() { // code } };
```

DESTRUCTOR:

Destructors are members functions in a class that delete an object.

Syntax:

```
~constructor-name();
```

TYPES OF CONSTRUCTORS:

Default constructor is the constructor which doesn't take any argument. It has no parameter.

Syntax:

```
class _name(parameter1, parameter2, ...)  
{  
    // constructor Definition  
}
```

Parameterized Constructors : These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

Syntax: name_of_class

Copy Constructors: These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object. We will study copy constructors in detail later.

Syntax: Class_name a,b; b = a;

Q.1) Write a program to demonstrate constructor and destructor

CODE:

```
#include <iostream>  
using namespace std;  
class Simple  
{  
    public:  
        Simple()  
        {  
            cout << "Constructor called" << endl;  
        }  
        ~Simple()  
        {  
            cout << "Destructor called" << endl;  
        }  
};  
int main()  
{  
    Simple s;
```

}

OUTPUT:

```
Constructor called  
Destructor called  
  
Process returned 0 (0x0)   execution time : 0.040 s  
Press any key to continue.
```

Fig1. Output of program demonstrating constructors and destructors