OBJECT ORIENTED PROGRAMING

OOP

PRACTICALS

NAME: SYEDA FAIZA ASLAM

SECTION: B

ROLL NO: CT-25064

LAB NO: 04

COURSE CODE: CT-260

COURSE TEACHER: Mr. AHMED ZAKI

Exercise

1. Create a class ‘Employee’ having two data members ‘EmployeeName’(char*) and ‘EmployeeId’ (int). Keep both data members private. Create three initialized objects ‘Employee1’, ‘Employee2’ and ‘Employee3’ of type ‘Employee’ in such a way that the employee name for each employee can be changed when required but the employee Id for each employee must be initialized only once and should remain same always. Use member initializer list, accessors/getters and mutators/setters for appropriate data members. The result must be displayed by calling the accessors.

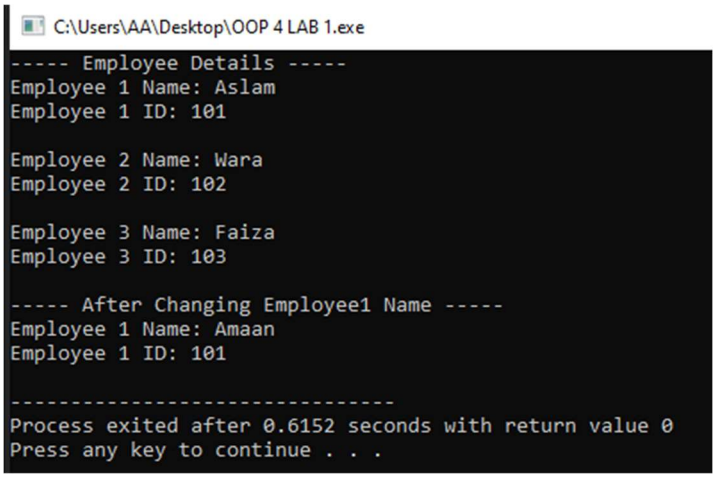
SOURCE CODE:

```
#include <iostream>
#include <cstring>
using namespace std;

class Employee
{
private:
   char *EmployeeName;
   const int EmployeeId;

public:
   // Constructor using member initializer list
   Employee(const char* name, int id) : EmployeeId(id)
   {
      EmployeeName = new char[strlen(name) + 1];
      strcpy(EmployeeName, name);
   }

   // Setter (only for name because ID is const)
```

```
    void setEmployeeName(const char* name)
    {
        delete[] EmployeeName;
        EmployeeName = new char[strlen(name) + 1];
        strcpy(EmployeeName, name);
    }

    // Getter for name
    char* getEmployeeName() const
    {
        return EmployeeName;
    }

    // Getter for ID
    int getEmployeeId() const
    {
        return EmployeeId;
    }

    // Destructor
    ~Employee()
    {
        delete[] EmployeeName;
    }
};

int main()
{
    Employee Employee1("Aslam", 101);
    Employee Employee2("Wara", 102);
    Employee Employee3("Faiza", 103);

    cout << "----- Employee Details -----" << endl;
```

```
    cout << "Employee 1 Name: " << Employee1.getEmployeeName() << endl;
    cout << "Employee 1 ID: " << Employee1.getEmployeeId() << endl;

    cout << "\nEmployee 2 Name: " << Employee2.getEmployeeName() << endl;
    cout << "Employee 2 ID: " << Employee2.getEmployeeId() << endl;

    cout << "\nEmployee 3 Name: " << Employee3.getEmployeeName() << endl;
    cout << "Employee 3 ID: " << Employee3.getEmployeeId() << endl;

    // Changing name (Allowed)
    Employee1.setEmployeeName("Amaan");

    cout << "\n----- After Changing Employee1 Name -----" << endl;
    cout << "Employee 1 Name: " << Employee1.getEmployeeName() << endl;
    cout << "Employee 1 ID: " << Employee1.getEmployeeId() << endl;

    return 0;
}
```

OUTPUT:



2. Create a class called DynamicArray. The class should contain a pointer to the array, and size of the array as data members. Create the parameterized constructor which take size of the array as input and initializes all the values with 0. Create the member function “add” which takes value as parameter and adds it to the end of the array. Create the member function size() which returns the size of the array.

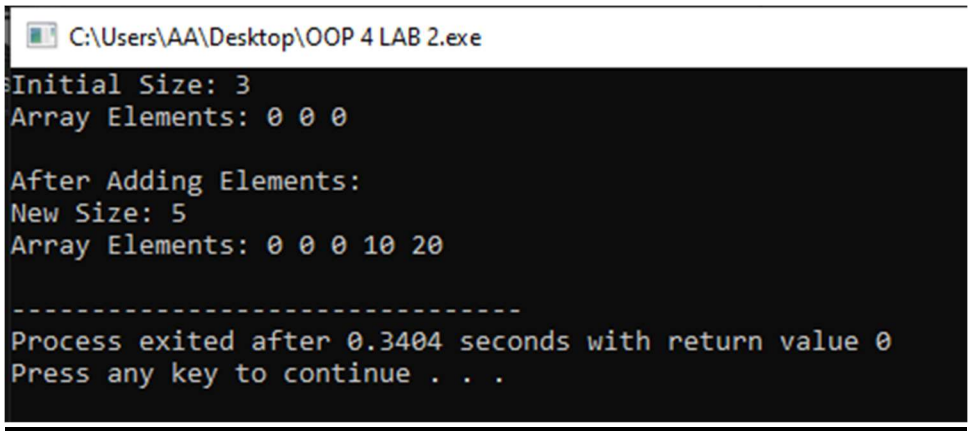
SOURCE CODE:

```
#include <iostream>
using namespace std;

class DynamicArray
{
private:
   int *arr;
   int arrSize;

public:
   // Parameterized Constructor
   DynamicArray(int size)
   {
      arrSize = size;
      arr = new int[arrSize];

      for(int i = 0; i < arrSize; i++)
      {
         arr[i] = 0;
      }
   }

   // Add function (adds value at the end)
   void add(int value)
```

```
{
   int *newArr = new int[arrSize + 1];

   for(int i = 0; i < arrSize; i++)
   {
      newArr[i] = arr[i];
   }

   newArr[arrSize] = value;

   delete[] arr;
   arr = newArr;
   arrSize++;
}

// Size function
int size()
{
   return arrSize;
}

// Display function (extra, for checking)
void display()
{
   cout << "Array Elements: ";
   for(int i = 0; i < arrSize; i++)
   {
      cout << arr[i] << " ";
   }
   cout << endl;
}

// Destructor
~DynamicArray()
{
```

```
        delete[] arr;
    }
};

int main()
{
    DynamicArray d1(3);

    cout << "Initial Size: " << d1.size() << endl;
    d1.display();

    d1.add(10);
    d1.add(20);

    cout << "\nAfter Adding Elements:" << endl;
    cout << "New Size: " << d1.size() << endl;
    d1.display();

    return 0;
}
```

OUTPUT:



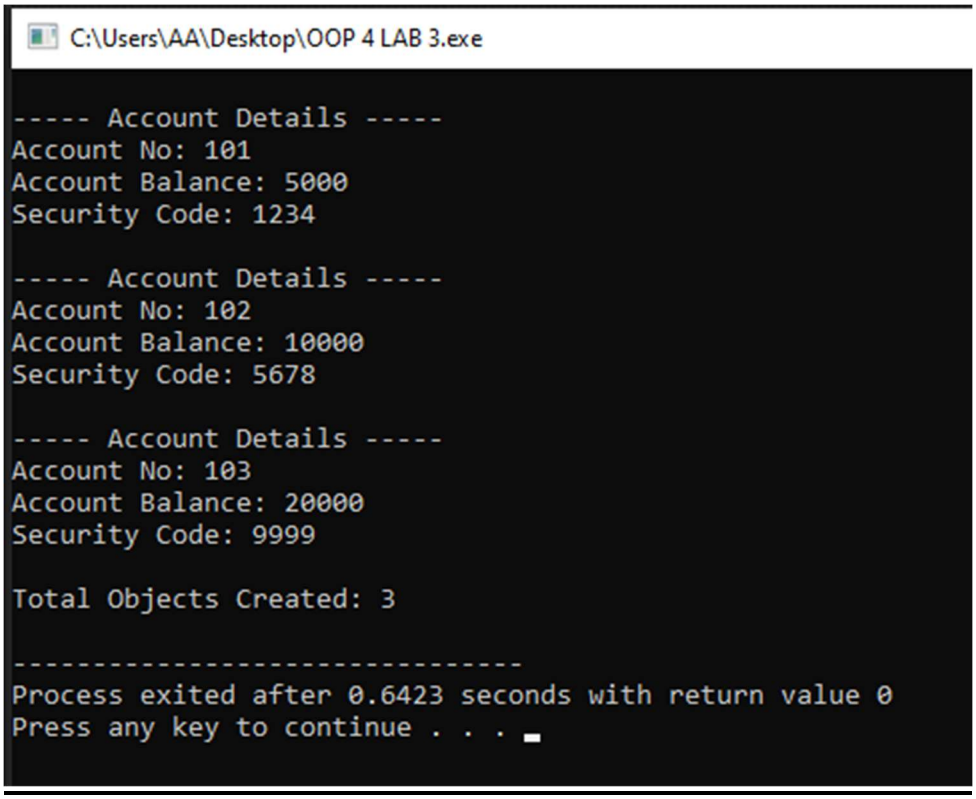
3. Write a program in which a class named Account has private member variables named account_no ,account_bal ,security_code. Use a public function to initialize the variables and print all data. Keep track of number of objects using the keyword static.

SOURCECODE:

```
#include <iostream>
using namespace std;

class Account
{
private:
   int account_no;
   float account_bal;
   int security_code;

   static int count;   // static member

public:
   Account()
   {
      count++;   // increase count whenever object is created
   }

   void setData(int accNo, float bal, int code)
   {
      account_no = accNo;
      account_bal = bal;
      security_code = code;
   }

   void display()
   {
```

```
        cout << "\n----- Account Details -----" << endl;
        cout << "Account No: " << account_no << endl;
        cout << "Account Balance: " << account_bal << endl;
        cout << "Security Code: " << security_code << endl;
    }

    static int getCount()
    {
        return count;
    }
};

// Initialize static member outside the class
int Account::count = 0;

int main()
{
    Account a1, a2, a3;

    a1.setData(101, 5000, 1234);
    a2.setData(102, 10000, 5678);
    a3.setData(103, 20000, 9999);

    a1.display();
    a2.display();
    a3.display();

    cout << "\nTotal Objects Created: " << Account::getCount() << endl;

    return 0;
}
```

OUTPUT:


C:\Users\AA\Desktop\OOP 4 LAB 3.exe
----- Account Details -----
Account No: 101
Account Balance: 5000
Security Code: 1234
----- Account Details -----
Account No: 102
Account Balance: 10000
Security Code: 5678
----- Account Details -----
Account No: 103
Account Balance: 20000
Security Code: 9999
Total Objects Created: 3

Process exited after 0.6423 seconds with return value 0
Press any key to continue . . .

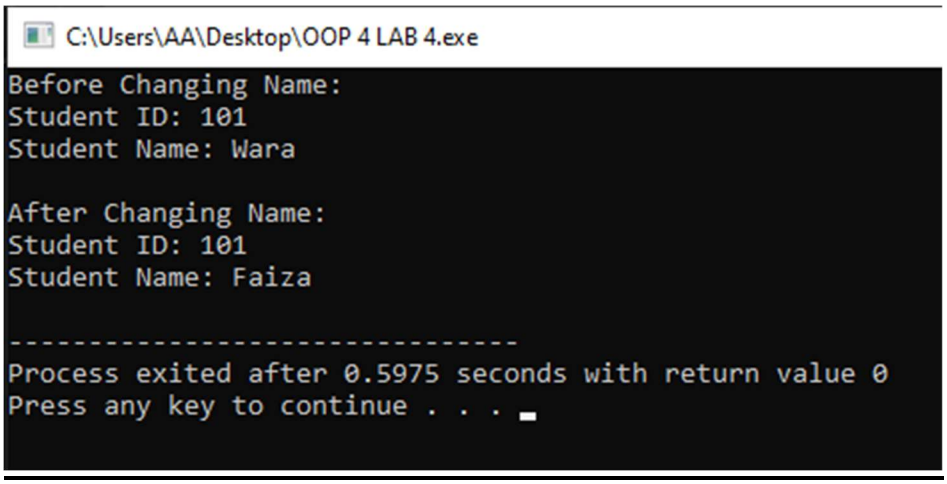
4. Write a program of your own in which you demonstrate the concept of constant keyword.

SOURCECODE:

```
#include <iostream>
#include <string>
using namespace std;

class Student
{
private:
   const int id;
   string name;

public:
   // Constructor using initializer list (for const variable)
   Student(int i, string n) : id(i)
   {
      name = n;
   }

   void setName(string n)
   {
      name = n;
   }

   void display() const
   {
      cout << "Student ID: " << id << endl;
      cout << "Student Name: " << name << endl;
   }
};
```

```
int main()
{
   Student s1(101, "Ali");

   cout << "Before Changing Name:\n";
   s1.display();

   s1.setName("Hassan");

   cout << "\nAfter Changing Name:\n";
   s1.display();

   // s1.id = 200;   // Error because id is const

   return 0;
}
```

OUTPUT:



5. “Hotel Mercato” requires a system module that will help the hotel to calculate the rent of the customers. You are required to develop one module of the system according to the following requirements:

The hotel wants such a system that should have the feature to change the implementation independently of the interface. This will help when dealing with changing requirements.

The hotel charges each customer 1000.85/- per day. This amount is being decided by the hotel committee and cannot be changed fulfilling certain complex formalities.

The module then analyses he number of days. If the customer has stayed for more than a week in the hotel, he gets discount on the rent. Otherwise, he is being charged normally.

The discounted rent is being calculated after subtracting one day from the total number of days.

In the end, the module displays the following details:

o Customer name

o Days

o Rent

Note that, the function used for displaying purpose must not have the ability to modify any data member.

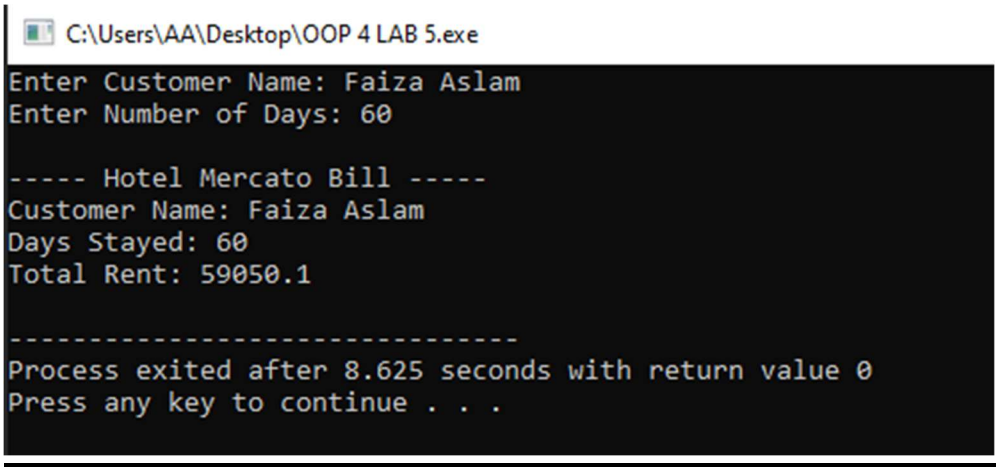
SOURCECODE:

```
#include <iostream>
#include <string>
using namespace std;

// Interface (Abstract Class)
class HotelInterface
```

```
{
public:
   virtual void calculateRent() = 0;
   virtual void display() const = 0;
};

// Implementation Class
class HotelMercato : public HotelInterface
{
private:
   string customerName;
   int days;
   float rent;

   const float rentPerDay;   // fixed rent decided by committee

public:
   HotelMercato(string name, int d) : rentPerDay(1000.85)
   {
      customerName = name;
      days = d;
      rent = 0;
   }

   void calculateRent()
   {
      if(days > 7)
      {
         rent = (days - 1) * rentPerDay;  // discount rule
      }
      else
      {
         rent = days * rentPerDay;
      }
   }
```

```
    void display() const
    {
        cout << "\n----- Hotel Mercato Bill -----" << endl;
        cout << "Customer Name: " << customerName << endl;
        cout << "Days Stayed: " << days << endl;
        cout << "Total Rent: " << rent << endl;
    }
};

int main()
{
    string name;
    int days;

    cout << "Enter Customer Name: ";
    getline(cin, name);

    cout << "Enter Number of Days: ";
    cin >> days;

    HotelMercato h(name, days);

    h.calculateRent();
    h.display();

    return 0;
}
```

OUTPUT:


C:\Users\AA\Desktop\OOP 4 LAB 5.exe
Enter Customer Name: Faiza Aslam
Enter Number of Days: 60
----- Hotel Mercato Bill -----
Customer Name: Faiza Aslam
Days Stayed: 60
Total Rent: 59050.1

Process exited after 8.625 seconds with return value 0
Press any key to continue . . .

SOLVED EXAMPLE:

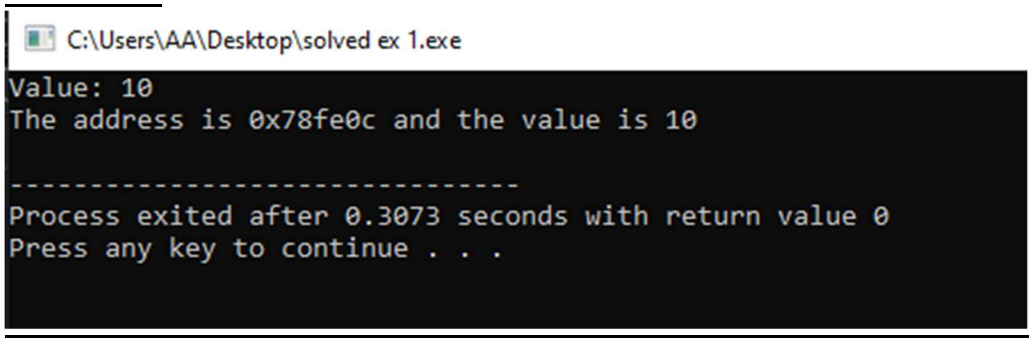
1) this Pointer Example (Using (*this))

SOURCECODE:

```
#include<iostream>
using namespace std;

class example
{
private:
   int x;

public:
   void set(int x)
   {
      (*this).x = x;
   }

   int get()
   {
      return x;
   }

   void printAddressAndValue()
   {
      cout << "The address is " << this
           << " and the value is " << (*this).x << endl;
   }
};

int main()
{
   example e1;
   e1.set(10);
```

```
    cout << "Value: " << e1.get() << endl;
    e1.printAddressAndValue();

    return 0;
}
```

OUTPUT:



---

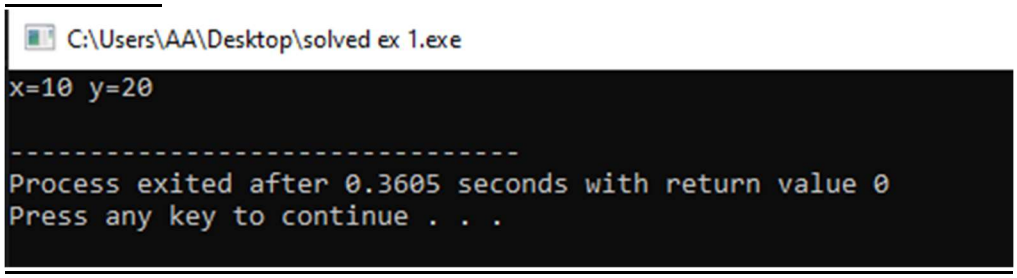
2) this Pointer Example (Using this->)

SOURCECODE:

```
#include<iostream>
using namespace std;

class Test
{
private:
    int x;
    int y;

public:
    Test(int x = 0, int y = 0)
    {
        this->x = x;
        this->y = y;
    }

    void print()
    {
```

```
        cout << "x=" << x << " y=" << y << endl;
    }
};

int main()
{
    Test obj1(10, 20);
    obj1.print();

    return 0;
}
```

OUTPUT:



---

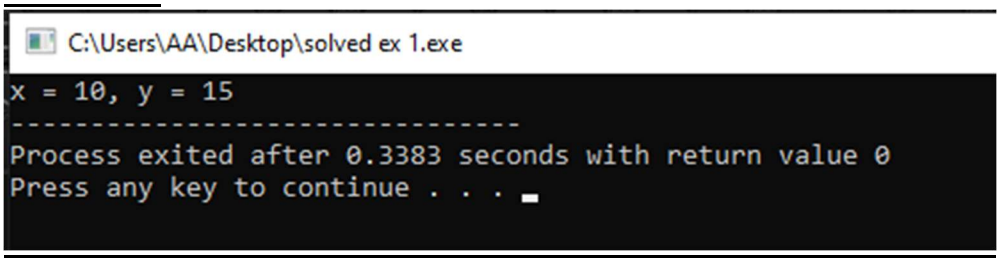
3) Member Initialization List Example

SOURCECODE:

```
#include<iostream>
using namespace std;

class Point
{
private:
    int x;
    int y;

public:
    Point(int i = 0, int j = 0) : x(i), y(j)
    { }

    int getX() const
```

```
    {
        return x;
    }

    int getY() const
    {
        return y;
    }
};

int main()
{
    Point t1(10, 15);

    cout << "x = " << t1.getX() << ", ";
    cout << "y = " << t1.getY();

    return 0;
}
```

OUTPUT:



---

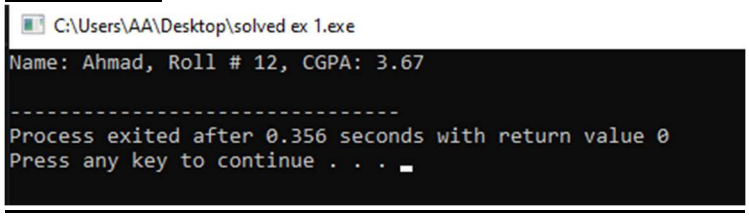
4) Constant Data Members Example

SOURCECODE:

```
#include <iostream>
#include <string>
using namespace std;

class Students
```

```
{
private:
   string name;
   const int rollno;
   float cgpa;

public:
   Students(int rno) : rollno(rno)
   { }

   void set(string sname, float cg)
   {
      name = sname;
      cgpa = cg;
   }

   void print()
   {
      cout << "Name: " << name
         << ", Roll # " << rollno
         << ", CGPA: " << cgpa << endl;
   }
};

int main()
{
   Students s(12);
   s.set("Ahmad", 3.67);
   s.print();

   return 0;
}
```

OUTPUT:



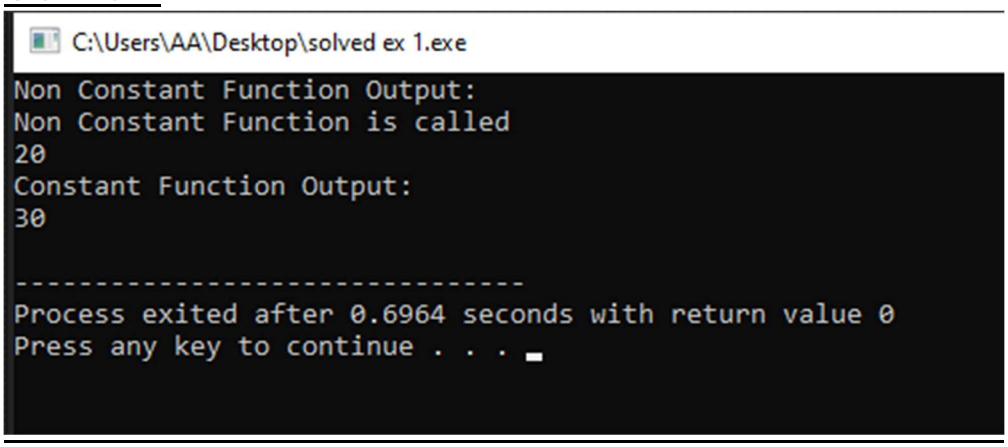
5) Constant Member Function Example

SOURCECODE:

```
#include<iostream>
using namespace std;

class test
{
private:
   int a;

public:
   int nonconstFunction(int a)
   {
      cout << "Non Constant Function is called" << endl;
      a = a + 10;
      return a;
   }

   int constFunction(int a) const
   {
      return a;
   }
};

int main()
{
   test t;

   cout << "Non Constant Function Output: " << endl;
   cout << t.nonconstFunction(10) << endl;

   cout << "Constant Function Output: " << endl;
   cout << t.constFunction(30) << endl;
```

```
    return 0;
}
```

OUTPUT:



---

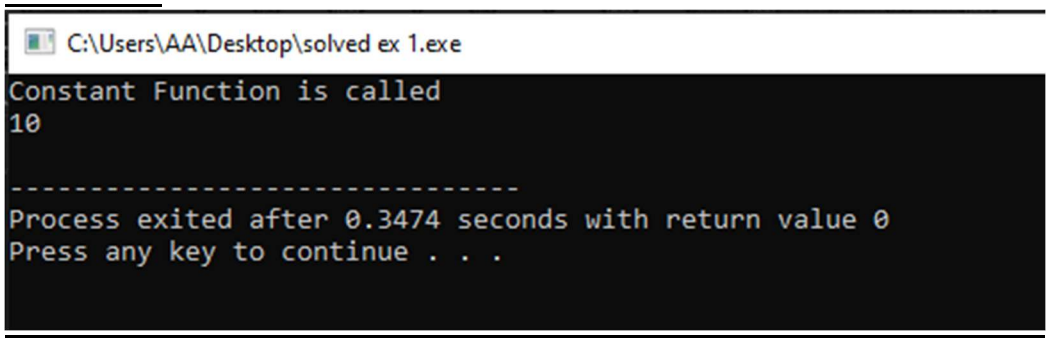
6) Constant Object Example

SOURCECODE:

```
#include<iostream>
using namespace std;

class test
{
public:
    int a;

    test()
    {
        a = 8;
    }

    int nonconstFunction()
    {
        cout << "Non Constant Function is called " << endl;
        return a;
```

```
    }

    int constFunction(int a) const
    {
        // this->a = a + 10;   //  ERROR (cannot modify inside const function)
        return a;
    }
};

int main()
{
    const test t;

    cout << "Constant Function is called" << endl;

    // t.a = 10;   // ERROR (can't modify const object)

    // t.nonconstFunction(); //  ERROR (can't call non-const function)

    cout << t.constFunction(10) << endl;

    return 0;
}
```

OUTPUT:



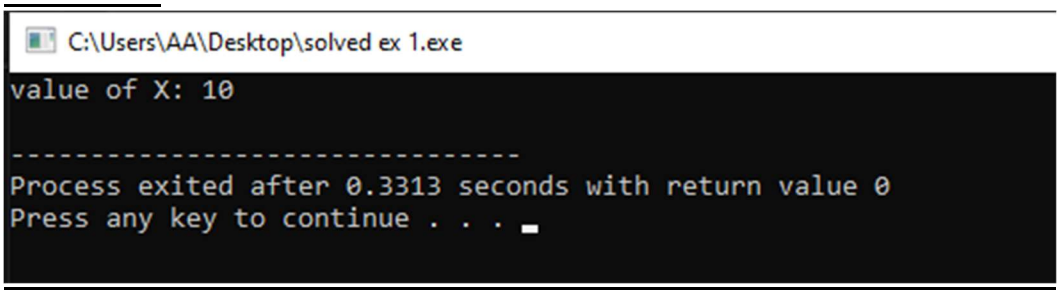
---

7) Static Member Function Example

SOURCECODE:

```
#include <iostream>
using namespace std;

class Demo
{
private:
   static int x;

public:
   static void fun()
   {
      cout << "value of X: " << x << endl;
   }
};

// defining static member
int Demo::x = 10;

int main()
{
   Demo X;
   X.fun();

   return 0;
}
```

OUTPUT:



8) Access Static Data Member Without Static Function

SOURCECODE:

```
#include <iostream>
using namespace std;

class Demo
{
public:
   static int ABC;
};

// defining static member
int Demo::ABC = 10;

int main()
{
   cout << "\nValue of ABC: " << Demo::ABC;

   return 0;
}
```

OUTPUT: