**Lab 02 – *Classes and Data Abstraction***

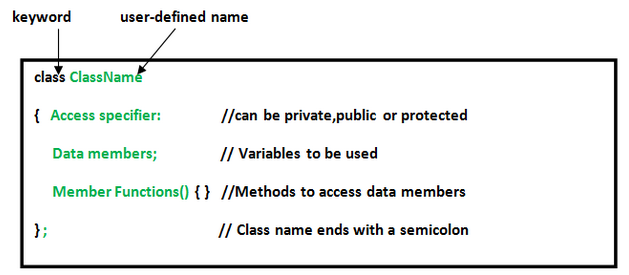
1. **Objectives:**

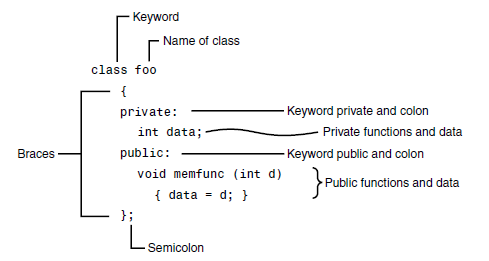
Objective of this lab is to understand the importance of classes and construction of objects using classes.

1. **Introduction:**

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object. A class serves as a plan, or blueprint. It specifies what data and what functions will be included in objects of that class. An object is often called an “instance” of a class **Syntax:**

Classes are generally declared using the keyword class, with the following format:





**3.2 Data Abstraction:**

Abstraction is the process of recognizing and focusing on important characteristics of a situation or object and leaving/filtering out the un-wanted characteristics of that situation or object. For example a person will be viewed differently by a doctor and an employer.

* A doctor sees the person as patient. Thus he is interested in name, height, weight, age, blood group, previous or existing diseases etc of a person
* An employer sees a person as an employee. Therefore employer is interested in name, age, health, degree of study, work experience etc of a person.
  1. **Member Functions and Variables:**

Member variables represent the characteristics of the object and member functions represent the behavior of the object. For example length & width are the member variables of class Rectangle and set\_values(int,int), area() are the member functions.

* 1. **Constructors:**

It is a special function that is automatically executed when an object of that class is created. It has no return type and has the same name as that of the class. It is normally defined in classes to initialize data members.

**Syntax:**

class\_name( )

{

// Constructor body

}

* 1. **Destructors:**

It is a special function that is automatically executed when an object of that class is destroyed. It has no return type and has the same name as that of the class preceded by tild (~) character. Unlike constructors, destructors cannot take arguments.

**Syntax:**

~ class\_name ()

{

// Destructor body

}

1. **In-Lab Tasks:**
   1. **Task#01**

Write a class that displays a simple message “I am object no. \_\_”, on the screen whenever an object of that class is created.

* **Code:**

#include<iostream>

using namespace std;

class Alpha

{

public:

static int Count;

Alpha()

{

Count++;

cout<<"\n\n\t I'm The Object Numeber :"<<Count<<endl;

}

};

int Alpha::Count=0;

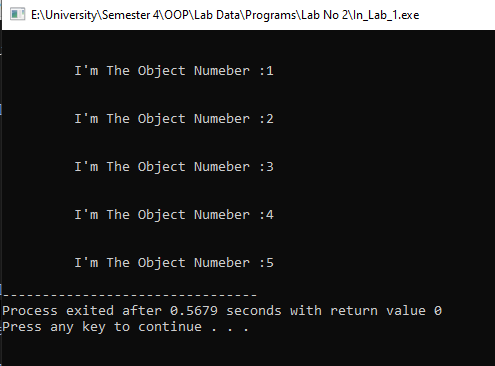
int main()

{

Alpha C1,C2,C3,C4,C5;

}

* **Output:**



* 1. **Task#02**

Write a program to calculate the number of objects created and destroyed for the counter class.

* **Code:**

#include<iostream>

using namespace std;

class Alpha

{

public:

static int Count;

static int Destruc;

Alpha()

{

Count++;

}

~Alpha()

{

Destruc++;

if(Destruc==Count)

{

cout<<"\n\n\t Total Objects Created:"<<Count<<endl;

cout<<"\n\n\t Total Objects Destroyed:"<<Destruc<<endl;

}

}

};

int Alpha::Count=0;

int Alpha::Destruc=0;

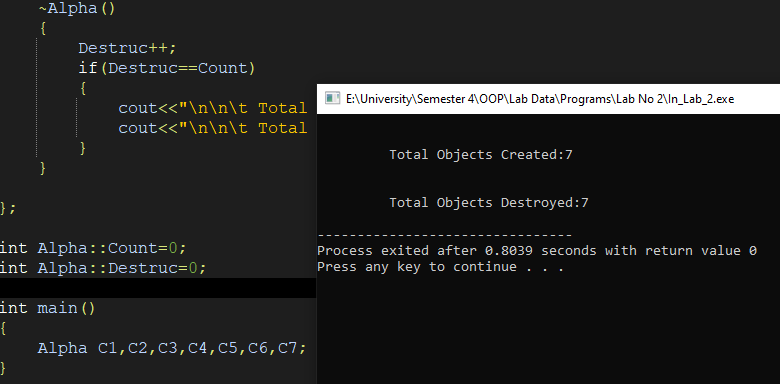
int main()

{

Alpha C1,C2,C3,C4,C5,C6,C7;

}

* **Output:**



* 1. **Task#03**

Create a class that imitates part of the functionality of the basic data type ‘int’, call the class **Int**. The only data in this class is an integer variable. Include member functions to initialize an **Int** to 0, initialize it to an ‘int’ value, to display it, and to add two **Int** values. Write a program that exercises this class by creating one uninitialized and two initialized **Int** values, adding the two initialized **Int** values and placing the response in uninitialized value and then displaying the result.

* **Code:**

#include<iostream>

using namespace std;

class INT

{

private:

int Num;

public:

void Single\_Zero()

{

Num=0;

}

void Single\_Intger(int A)

{

Num=A;

}

void Display()

{

cout<<"\n\n\t Value Stored In A: "<<Num;

}

void Sum(INT O4,INT O5)

{

Num=O4.Num+O5.Num;

}

};

int main()

{

INT O1,O2,O3;

O2.Single\_Intger(15);

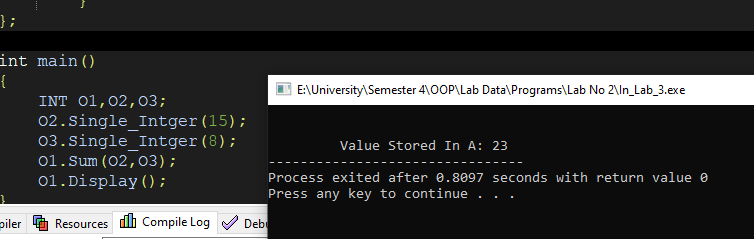
O3.Single\_Intger(8);

O1.Sum(O2,O3);

O1.Display();

}

* **Output:**



1. **Post-Lab Tasks:**
   1. **Task#01**

Create a class named time, the data members are hours, minutes and seconds. Write a function to read the data members supplied by the user, write a function to display the data members in standard (24) hour and also in (12) hour format.

* **Code:**

#include <iostream>

using namespace std;

class Time

{

public:

int Hours;

int Mins;

int Secs;

Time()

{

cout<<"\n\n\t Please Enter The Hours: "; cin>>Hours;

cout<<"\n\t Please Enter The Minutes: "; cin>>Mins;

cout<<"\n\t Please Enter The Seconds: "; cin>>Secs;

}

void Display\_24(int Hou, char Meri)

{

if(Meri=='P'|| Meri=='p' )

{

Hou=Hou+12;

}

else

{

if(Hou==12)

{

Hou=0;

}

}

cout<<"\n\n\t Time In Standered 24 Hour Format is :\t "<<Hou<<" Hour : "<<Mins<<" Miuntes : "<<Secs<<" Seconds";

}

void Display\_12(char Meri)

{

cout<<"\n\n\t Time In Standered 12 Hour Format is :\t "<<Hours<<" Hour : "<<Mins<<" Miuntes : "<<Secs<<" Seconds "<<Meri<<"M";

}

};

int main()

{

Time Alpha;

char Meridiem;

int Format;

int H;

H=Alpha.Hours;

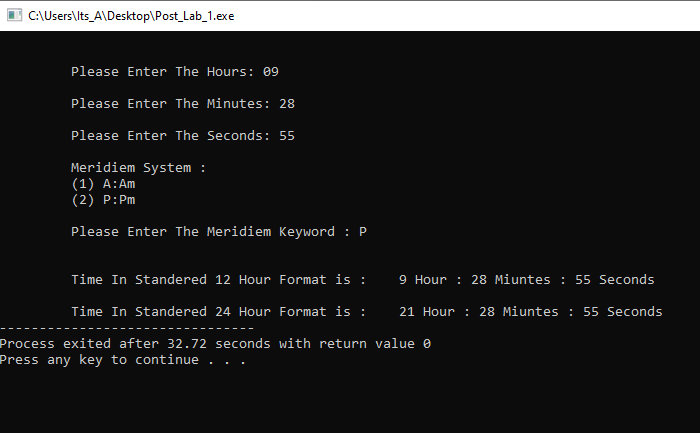
cout<<"\n\t Meridiem System : \n\t (1) A:Am \n\t (2) P:Pm \n\n\t Please Enter The Meridiem Keyword : "; cin>>Meridiem;

Alpha.Display\_12(Meridiem);

Alpha.Display\_24(H,Meridiem) ;

}

* **Output:**



* 1. **Task#03**

Write a class marks with three data members to store three marks. Write three member functions, set\_marks() to input marks, sum() to calculate and return the sum and avg() to calculate and return average marks. Write a program that exercises this class by creating its objects and displaying results.

* **Code:**

#include <iostream>

using namespace std;

class Student

{

public:

int Stu\_No\_1;

int Stu\_No\_2;

int Stu\_No\_3;

int Sum;

int Summer();

int Average(int);

void Set\_Mark();

};

void Student::Set\_Mark()

{

cout<<"\n\n\t Please Enter The Marks of Student No 1 : "; cin>>Stu\_No\_1;

cout<<"\n\t Please Enter The Marks of Student No 2 : "; cin>>Stu\_No\_2;

cout<<"\n\t Please Enter The Marks of Student No 3 : "; cin>>Stu\_No\_3;

}

int Student::Summer()

{

Sum=Stu\_No\_1+Stu\_No\_2+Stu\_No\_3;

return(Sum);

}

int Student::Average(int Su)

{

return(Su/3);

}

int main()

{

Student Alpha;

int Sum;

int Avg;

// Setting Marks Of Student

Alpha.Set\_Mark();

// Creating Sum Of All Student Numbers

Sum=Alpha.Summer();

// Creating Average Of The Students Marks

Avg=Alpha.Average(Sum);

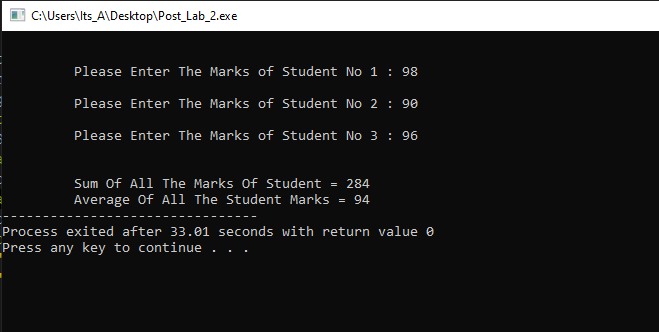
// Displaying Results

cout<<"\n\n\t Sum Of All The Marks Of Student = "<<Sum;

cout<<"\n\t Average Of All The Student Marks = "<<Avg;

}

* **Output:**



1. **Conclusion:**

**Classes** support a powerful programming model by encapsulating related functionality into objects. The **benefit** of organized code is especially important for maintenance, where changes or enhancements can be limited to the objects that are affected by the change. **Classes** enhance code reuse.

**Advantages of Constructors:**

* Automatic initialization of objects at the time of their declaration.
* Multiple ways to initialize objects according to the number of arguments passes while declaration.
* The objects of child class can be initialized by the **constructors** of base class.

**Advantages of Destructor:**

It gives the final chance to clean up the resources that are not in use to release the **memory** occupied by unused objects like deleting dynamic objects, close of the system handles, used files.