**Experiment No. 2**

**Title:** **Implementation of class with constructors, destructor and getter-setter methods**

**Batch:B2 Roll No.:16010421091 Experiment No.:2**

**Aim**: Define a class named ‘Complex’ which stores the real part ‘a’ and imaginary part ‘b’ of a complex number and perform the following:

1. Define parameterized and copy constructors in the class
2. Define destructor in the class
3. Define getter-setter functions in the class
4. Define following functions:
5. display ( ) – to print complex number in the form of ‘a + bi’
6. increment ( ) – to increment value of real part ‘a’ and imaginary part ‘b’ by 1
7. add ( ) – to add two complex numbers
8. multiply( ) – to multiply two complex numbers

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resources needed: Text Editor, C++ compiler**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

### Theory:

**Classes**

A Class is a plan which describes the object. We call it as a blue print of how the object should be represented. It defines what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. A class is to an object, as a blueprint is to a building.

Syntax:

class class\_name

{

access specifier:

data members

member functions

};

**Objects**

Any real world entity which can have some characteristics or which can perform some work is called as Object. An Object is an instance of the class. Each Object consist of

* Attributes/Characteristics/Properties
* Behaviour/Function/method

Syntax:

class\_name object\_name

The members of a class are accessed using (.) dot operator as

Objectname.member

**Constructor**

It is a member function which initializes a class. A constructor has:

(i) the same name as the class itself

(ii) no return type

A constructor is called automatically whenever a new instance of a class is created. The constructor should be declared in the public section only.

**Constructor Overloading**

More than one constructor can be defined in a class, as long as each has a different list of arguments

Let us consider a class rectangle as shown below:

class rectangle

{

private:

float height;

float width;

int xpos;

int ypos;

public:

void draw();

};

**Default Constructor:** If a constructor is not specified, the compiler generates a default constructor. It expects no parameters and has an empty body.

Default Constructor for the rectangle class:

rectangle()

{

}

**Parameterized Constructor:** It is a constructor that can take parameters. The parameters of a parameterized constructor can be any type except the class name to which it belongs to.

Parameterized Constructor for the rectangle class:

rectangle(float w, float h) //constructor definition

{

height = h;

width = w;

xpos = 0;

ypos = 0;

}

int main()

{

rectangle r1(200,300); //calling the constructor

return 0;

}

Parameterized Constructor with default argument for the rectangle class:

rectangle(float w, float h = 100) //constructor definition

{

height = h;

width = w;

xpos = 0;

ypos = 0;

}

int main()

{

rectangle r1(200); //calling the constructor

return 0;

}

**Copy Constructor:** A Copy Constructor is a special type of constructor which initializes all the data members of the newly created object by copying the contents of an existing object. It accepts a reference to its own class as a parameter.

Copy Constructor for the rectangle class:

rectangle(rectangle &t)//constructor definition

{

height = t.height;

width = t.width;

xpos = t.xpos;

ypos = t.ypos;

}

int main()

{

rectangle r1; //calling the default constructor

rectangle r2(r1); //calling copy constructor

return 0;

}

**Destructor**

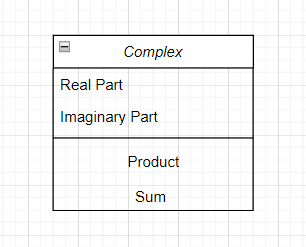
A destructor is a member function having same name as that of its class preceded by ~(tilde) sign and which is used to destroy the objects that have been created by a constructor. It is implicitly invoked when an object’s scope is over.

Example: ~rectangle()

{

}

Class Diagram:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Results: (Program with snapshot of output)**

**#include<iostream>**

**#include<cmath>**

**using namespace std;**

**class Complex**

**{**

**public:**

**int a;**

**int b;**

**Complex(int real,int img)//Parametrized Constructor**

**{**

**a = real;**

**b = img;**

**}**

**~Complex()**

**{**

**cout<<"Finished!!"<<endl;**

**}**

**int getreal()//Getter Function**

**{**

**return a;**

**}**

**int getimg()//Getter Function**

**{**

**return b;**

**}**

**void add(Complex c)**

**{**

**//Adding two Complex numbers**

**int add\_a,add\_b;**

**add\_a = c.a + a;**

**add\_b = c.b + b;**

**cout<<"Addition is: "<<add\_a<<" + "<<add\_b<<"i"<<endl;**

**}**

**void multiply(Complex c)**

**{**

**//Mulitplying two complex numbers**

**int mult\_a,mult\_b;**

**mult\_a = a\*c.a - b\*c.b;**

**mult\_b = a\*c.b + b\*c.a;**

**cout<<"Multiplication is: "<<mult\_a<<" + "<<mult\_b<<"i"<<endl;**

**}**

**};**

**int main()**

**{**

**Complex c1(9,9);**

**Complex c2(6,6);**

**Complex c3 = c2;//Copy Constructor**

**cout<<c1.getreal()<<endl;**

**cout<<c2.getreal()<<endl;**

**cout<<c1.getimg()<<endl;**

**cout<<c2.getimg()<<endl;**

**c1.add(c2);**

**c1.multiply(c2);**

**}**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Test Cases (minimum 5 test cases required):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Sample Input** | **Sample Output** | **Description** | **Test Case Type (general/special)** | **Pass/Fail** |
| **1.** | **Complex c1(3,4);**  **Complex c2(5,8);** | **Addition is: 8 + 12i**  **Multiplication is: -17 + 4** |  | **General** | **Pass** |
| **2.** | **Complex c1(12,5);**  **Complex c2(3,7);** | **Addition is: 15 + 12i**  **Multiplication is: 1 + 99i** |  | **General** | **Pass** |
| **3.** | **Complex c1(1,2);**  **Complex c2(5,6);** | **Addition is: 6 + 8i**  **Multiplication is: -7 + 16i** |  | **General** | **Pass** |
| **4.** | **Complex c1(0,0);**  **Complex c2(2,3);** | **Addition is: 2 + 3i**  **Multiplication is: 0 + 0i** |  | **General** | **Pass** |
| **5.** | **Complex c1(12,5);**  **Complex c2(3,7);** | **Addition is: 15 + 15i**  **Multiplication is: 0 + 108i** |  | **General** | **Pass** |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Questions:**

1. Compare the class in C++ with a structure in C

Structures are value types; classes are reference types. A variable of a structure type contains the structure's data, rather than containing a reference to the data as a class type does.

Structures use stack allocation; classes use heap allocation.

All structure elements are Public by default; class variables and constants are Private by default, while other class members are Public by default. This behavior for class members provides compatibility with the Visual Basic 6.0 system of defaults.

A structure must have at least one nonshared variable or nonshared, noncustom event element; a class can be completely empty.

Structure elements cannot be declared as Protected; class members can.

A structure procedure can handle events only if it is a SharedSub procedure, and only by means of the AddHandler Statement; any class procedure can handle events, using either the Handles keyword or the AddHandler statement. For more information,

Structure variable declarations cannot specify initializers or initial sizes for arrays; class variable declarations can.

2. Define a class named ‘Account’ which stores following data: Name of Account Holder, Account Number and Balance. Define withdraw( ), deposit( ) and viewBalance( ) functions in the class and use them appropriately in main( ) to perform operations with 2 accounts.

#include <iostream>

using namespace std;

class Acc

{

public :

int amtinacc;

int amtdep;

int withdrawal, value;

void accountnumber()

{

float accnum;

cout<<"Enter your account number : ";

cin>>accnum;

cout<<"Enter the amount in your bank acc : ";

cin>>amtinacc;

cout<<amtinacc;

}

void withdraw()

{

cout<<"enter the amount you want to withdraw : ";

cin>>withdrawal;

if (withdrawal>amtinacc)

{

cout<<"INSUFFICIENT BALANCE";

}

else

{

value = amtinacc - withdrawal;

cout<<"Now you acc balance is : "<<value<<endl;

}

}

void Deposit()

{

cout<<"enter the amount to deposit";

cin>>amtdep;

amtinacc = amtinacc + amtdep;

cout<<"Total amt : "<<amtinacc<<endl;

}

void Checkbalance()

{

cout<<"Your account balance is : "<<value<<endl;

}

};

int main()

{

Acc a1;

a1.accountnumber();

a1.Deposit();

a1.withdraw();

a1.Checkbalance();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Outcomes:**

Initializing Class Objects: Constructors, Using Default Arguments With Constructors, Using Destructors, Classes : Const(Constant) Object And Const Member Functions, Object as Member of Classes, Friend Function and Friend Classes, Using This Pointer, Dynamic Memory Allocation with New and Delete, Pointers to objects, Static Class Members, Container Classes And Integrators, Proxy Classes, Function overloading.

--------------------------------------------------------------------

**Conclusion: (Conclusion to be based on the outcomes achieved)**

From this experiment we can conclude that:

1. We have learnt about creating basic classes and objects in OOP.
2. We have learnt about creating default, parametrized and copy constructors.
3. We have learnt about creating destructors and using getter and setter functions.
4. We have applied all the concepts into a program and implemented them thoroughly.

**Grade: AA / AB / BB / BC / CC / CD /DD**

Signature of faculty in-charge with date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**References:**

**Books/ Journals/ Websites:**

1. E Balagurusamy, Object oriented Programming with C++, Tata McGraw-Hill, 8th Edition September 2020
2. Herbert Schildt, C++: The Complete Reference, McGraw Hill Education, 4th edition, July 2017
3. Jeff Langr, Modern C++ Programming with Test-Driven Development : Code Better,Sleep Better, O′Reilly, 1st edition, November 2013
4. <https://docs.microsoft.com/en-us/cpp/cpp/?view=msvc-170>