

Lecture 6

OOP

Khola Naseem
khola.naseem@uet.edu.pk

Getter and setter functions

➤ **Getter Function:**

- Inhibiting all external access to the values of private member variables of a class is rather extreme
- Find out the value of the member variable
- But expose the member variables by using the public keyword either.
- You can provide access to the values of private member variables by adding member functions to return their values.
- The values of the member variables are fully accessible, but they can't be changed from outside the class
- Functions that retrieve the values of member variables are referred to as **accessor functions**.

Getter and setter functions

➤ Getter Function:

➤ Example

```
#include <iostream>
using namespace std;
/* run this program using the console pauser or add your own */
class Box{
private:
    float length;
    float breadth;
public:
    Box()
    {
        length=2.2;
        breadth=2.4;
    }
    float getlength()
    {
        return length;
    }
    float getbreadth()
    {
        return breadth;
    }
    double Area()
    {
        return length*breadth;
    }
};
int main(int argc, char** argv) {
    Box b1;

    cout<<"Area of b1 is "<<b1.Area()<<endl;
    cout<<"Get value of the length  "<<b1.getlength()<<endl;
    cout<<"Get value of the length  "<<b1.getbreadth()<<endl;
}
```

Output:

```
Area of b1 is 5.28
Get length value  2.2
Get breadth value 2.4
```

Getter and setter functions

➤ **Setter Function:**

- Modify value of private data member using the member function
- Member functions that allow member variables to be modified are sometimes called **mutators**.
- allow member variables to be changed from outside the class
- Has opportunity to apply checks on the values.

Getter and setter functions

- **Setter Function:**
- Modify value of private data member using the member function

Output:

```
class Box{
private:
    float length;
    float breadth;
public:
    Box()
    {
        length=2.2;
        breadth=2.4;
    }
    float getlength()
    {
        return length;
    }
    float getbreadth()
    {
        return breadth;
    }
    void setlength(float a)
    {
        if (a>2.3)
        {
            length=a;
        }
    }
    void setbreadth(float b)
    {
        if (b>0)
        {
            breadth=b;
        }
    }
    double Area()
    {
        return length*breadth;
    }
};

int main(int argc, char** argv) {
    Box b1;
    float new_len=3.0, new_breadth=-9.0;
    cout<<"Area of b1 is "<<b1.Area()<<endl;
    cout<<"Get value of the length "<<b1.getlength()<<endl;
    cout<<"Get value of the length "<<b1.getbreadth()<<endl;
    b1.setlength(new_len);
    b1.setbreadth(new_breadth);
    cout<<"return the value of the length after setter function call "<<b1.getlength()<<endl;
    cout<<"return the value of the breadth after setter function call "<<b1.getbreadth()<<endl;
}
```

```
Area of b1 is 5.28
Get value of the length  2.2
Get value of the length  2.4
return the value of the length after setter function call 3
return the value of the breadth after setter function call 2.4
```

Access and utility functions

- Access functions can read or display data. Another common use for access functions is to test the truth or falsity of conditions—such functions are often called predicate functions. E.g isEmpty
- A utility function (also called a helper function) is not part of a class's public interface; rather, it's a private member function that supports the operation of the class's other member functions.
- Utility functions are not intended to be used by clients of a class (but can be used by friends of a class)
- Example:

```
class Studentmarks{  
    public:  
        static const int sub=5;  
        Studentmarks();    //initialize marks of all subjects to 0  
        void getMarks();    //get marks from user  
        void setMarks(int, float); //set marks  
        void printTotalMarks(); // print marks  
    private:  
        double totalMarks();    // calculate the total marks  
        double marks[sub];  
};
```

Separate declaration and definition of member functions

- The separate declaration

- `ReturnType functionName(parameterList);`

- Separate definition of member functions

- `ReturnType ClassName::functionName(ParameterList)`

- The two colons are called the **scope resolution operator**. When `Box::` appears before the name of a function in a function header, it identifies the function as a member of the Box class.

Separate declaration and definition of member functions

```
class Box{
private:
    float length;
    float breadth;
public:
    Box();
    Box(float a,float b);
    double Area();
};

Box::Box()
{
    length=2.0;
    breadth=2.0;
}

Box::Box(float a,float b)
{
    length=a;
    breadth=b;
}

double Box::Area()
{
    return length*breadth;
}

int main(int argc, char** argv) {
    Box b1;
    Box b2(3.1,8.1);
    cout<<"Area of b1 is "<<b1.Area()<<endl;
    cout<<"Area of b2 is "<<b2.Area();
}
```

Output:

```
Area of b1 is 4
Area of b2 is 25.11
-----
```


Passing an Object as argument

- Objects are pass to a function in a similar manner as regular arguments.
- Example:
 - Copy constructor
 - Example2:

```
class Employee {  
private:  
    double Salary;  
public:  
    Employee(double s) {  
        Salary = s;  
    }  
    void calculateAverageSalary(Employee e2) {  
        double average = (Salary + e2.Salary) / 2;  
        cout << "Average Marks = " << average << endl;  
    }  
};  
int main()  
{  
    Employee E1(35666);  
    Employee E2(2366.88);  
  
    E1.calculateAverageSalary(E2);  
}
```

Return an Object from a function

➤ Example:

```
class Employee {
private:
    double Salary;
public:
    Employee()
    {
        Salary=0.0;
    }
    Employee(double s) {
        Salary = s;
    }
    Employee calculateSum(Employee e2) {
        Employee e3;
        e3 = (Salary + e2.Salary);
        return e3;
    }
    void print()
    {
        cout<<"Salary is "<<Salary;
    }
};

int main()
{
    Employee E1(35666);
    Employee E2(2366.88);
    Employee E3;
    E3=E1.calculateSum(E2);
    E3.print();
}
```

Cascaded calls to functions

- when multiple functions called using a single object name in a single statement, it is known as cascaded function call
- Example:
 - `ob.FUN1().FUN2().FUN3();`

this pointer:

➤ Example:

```
class Demo {
    int a;
public:

    void FUN1()
    {
        cout << "\nFUN1 CALLED" << endl;
    }

    void FUN2()
    {
        cout << "\nFUN2 CALLED" << endl;
    }

    void FUN3()
    {
        cout << "\nFUN3 CALLED" << endl;
    }
};

int main()
{
    Demo ob;
    ob.FUN1().FUN2().FUN3();
    return 0;
}
```

➤ Error

Return an Object from a function

➤ Example:

```
class Employee {
private:
    double Salary;
public:
    Employee()
    {
        Salary=0.0;
    }
    Employee(double s) {
        Salary = s;
    }
    Employee calculateSum(Employee e2) {
        Employee e3;
        e3 = (Salary + e2.Salary);
        return e3;
    }
    void print()
    {
        cout<<"Salary is "<<Salary;
    }
};

int main()
{
    Employee E1(35666);
    Employee E2(2366.88);

    E1.calculateSum(E2).print();
}
```

```
Salary is 38032.9
-----
Process exited after 0.3059 seconds with return value 0
Press any key to continue . . .
```

Return an Object from a function

➤ Example:

- Write a class family class having firstName and secondName as data members. Write getter and setter function and a function called newName() in which one of the objects is passed as parameter. Concatenate the first name of first and second object and pass it into the first name of third object and Concatenate the Second name of first and second object and pass it into the second name of third object. Return the third object from the newName() function. And display() the First and second name of the third object.

this pointer:

- How class objects interact with the functions and data members of a class.
 - Each object gets its own copy of the data member.
 - All-access the same function definition as present in the code segment.
- Meaning each object gets its own copy of data members and all objects share a single copy of member functions.
- there must be a way for the function to refer to the members of the particular object for which it has been called

```
void Box:: calculate(){  
    cout<<"anwser is ="<<length*breadth*height<<endl;  
}
```

- Box b1(1.2,2.3.4.5),b2(1.1,1.2.1.3);
- b1.calculate()
- b2.calculate()
- The compiler takes care of adding the this pointer name to the member names in the function. In other words, the compiler implements the function as follows:

Separate declaration and definition of member functions

1. The class should have following **four private data members**.

1. An **integer** named **id** that holds the **item's item number**.
2. A **string** named **name** that holds the **item's name**.
3. An **integer** named **quantity** for holding the **quantity** of the items on hand.
4. A **float** named **cost** for holding the wholesale **per-unit cost** of the item.

Value should only be assigned to data member **id**, **quantity** and **cost** if they are positive, **zero** otherwise.

2. Provide the implementation of **mutators** for all the data members (id, name, quantity and cost) of the class.
3. Provide the implementation of **accessors** for all the data members (id, name, quantity and cost) of the class.
4. Provide the implementation of following **constructors** and a **destructor**
 1. The constructor should accept the **item's item number**, **name**, **quantity** and **cost** as arguments. These values should be assigned to the object's appropriate member variables.
 2. The constructor should accept the **item's item number**, **name** and **quantity** as arguments. These values should be assigned to the object's appropriate member variables. The **cost** should be assigned the default value.
 3. The constructor should accept the **item's item number**, **name** and **cost** as arguments. These values should be assigned to the object's appropriate member variables. The **quantity** should be assigned the default value.
 4. A **copy constructor** to initialize an item's object with already existing object.
 5. A **destructor** that do nothing except displaying a simple message "Destructor executed..." on the screen.
5. Provide the implementation of following member functions
 1. **setItem** method accepts **item's item number**, **name**, **quantity** and **cost** as arguments and assigns them to the appropriate member variables.
 2. **getItem** method to **initialize the data** of an item **taken** from the user.
 3. **putItem** method to display the information of a particular **item**.
 4. **getTotalCost** method should provide the facility to **calculate and return the total cost** of an item only if the quantity is greater than or equal to 1, return 0 otherwise.
 5. **isEqual** method should provide the facility to **compare two objects** (left hand side and right hand side) and return **true** if they are having same state, **false** otherwise.

this pointer:

- The compiler takes care of adding the this pointer name to the member names in the function. In other words, the compiler implements the function as follows:

```
double Box::volume()  
{  
    return this->m_length * this->m_width * this->m_height;  
}
```

- You could write the function explicitly using the pointer this if you wanted, but it isn't necessary. However, there are situations where you do need to use this explicitly, such as when you need to return the address of the current object.
- **Static member functions** do not contain a this pointer.

this pointer:

- When local variable's name is same as member's name

```
class Demo {  
    int a;  
public:  
    Demo(int a)  
    {  
        a=a;  
    }  
    void print()  
    {  
        cout<<"value of a "<<a;  
    }  
};  
  
int main()  
{  
    Demo ob(4);  
    ob.print();  
    return 0;  
}
```

value of a 0

this pointer:

➤ this:

```
class Demo {  
    int a;  
public:  
    Demo(int a)  
    {  
        this->a=a;  
    }  
    void print()  
    {  
        cout<<"value of a "<<a;  
    }  
};  
  
int main()  
{  
    Demo ob(4);  
    ob.print();  
    return 0;  
}
```

```
value of a 4  
-----
```

this pointer:

➤ Second solution:

```
class Demo {
    int a;
public:
    Demo(int a)
    {
        this->a=a;
    }

    //Second solution
    /* Demo(int a): a(a){
    }
    */
    void print()
    {
        cout<<"value of a "<<a;
    }
};

int main()
{
    Demo ob(4);
    ob.print();
    return 0;
}
```

value of a 4

this pointer:

- To return reference to the calling object
 - When a reference to a local object is returned, the returned reference can be used to chain function calls on a single object.
- Example:

```
Box& setlength(float length){  
    length=a;  
    return *this;  
}
```

this pointer:

➤ Cascaded calls:

```
class Box {
    float length;
    float breadth;
    float height;
public:
    Box();
    Box& setlength(float length);
    Box& setbreadth(float breadth);
    Box& setheight(float height);
    void calculate();
};

Box::Box()
{
    length=0.0;
    breadth=0.0;
    height=0.0;
}

Box& Box::setlength(float a)
{
    if(a>0.0)
    {
        length=a;
    }

    return *this;
}

Box& Box::setbreadth(float b)
{
    if(b>0.0)
    {
        breadth=b;
    }

    return *this;
}

Box& Box::setheight(float c){
    if(c>0.0)
    {
        height =c;
    }

    return *this;
}

void Box:: calculate(){
    cout<<"anwser is ="<<length*breadth*height<<endl;
}
```

```
int main()
{
    Box b;
    b.setlength(2.3).setheight(4.1).setbreadth(2.1).calculate();
    return 0;
}
```

anwser is =19.803

cout<<"The vlaue is";

Static member:

- Static member variables of a class are used to provide class-wide storage of data that is independent of any particular object of the class type but is accessible by any of them.
- Declare a member variable of a class as **static**,
- only one instance of each static member variable exists, regardless of how many class objects have been defined. It is initialized before any object of this class is created, even before the main starts.
- Static members can be initialized **outside the class** using the scope resolution operator **::**
- Declare: inside the class

```
class X
{
public:
    static int i;
};
int X::i = 0; // definition outside class declaration
```

Static member:

➤ Declare:

```
class Box {  
    float length;  
    float breadth;  
    float height;  
  
public:  
    Box();  
    Box& setlength(float length);  
    Box& setbreadth(float breadth);  
    Box& setheight(float height);  
    void calculate();  
    static int obj_count;  
};
```

➤ Initialize: Error to access if not initialized

```
void Box:: calculate(){  
    cout<<"answer is "<<length*breadth*height<<endl;  
}  
int Box:: obj_count=0;  
int main()  
{  
    Box b,b2;  
    b.setlength(2.3).setheight(4.1).setbreadth(2.1).calculate();  
    cout<<"Box objects: "<<Box::obj_count;  
    return 0;  
}
```


Static member:

➤ Declare:

```
class Box {
    float length;
    float breadth;
    float height;

public:
    Box();
    Box& setlength(float length);
    Box& setbreadth(float breadth);
    Box& setheight(float height);
    void calculate();
    static int obj_count;
};

Box::Box()
{
    length=0.0;
    breadth=0.0;
    height=0.0;
    obj_count++;
}

Box& Box::setlength(float a)
{
    if(a>0.0)
    {
        length=a;
    }

    return *this;
}

Box& Box::setbreadth(float b)
{
    if(b>0.0)
    {
        breadth=b;
    }

    return *this;
}
```

```
Box& Box::setbreadth(float b)
{
    if(b>0.0)
    {
        breadth=b;
    }

    return *this;
}

Box& Box::setheight(float c){
    if(c>0.0)
    {
        height =c;
    }

    return *this;
}

void Box:: calculate(){
    cout<<"anwser is ="<<length*breadth*height<<endl;
}

int Box:: obj_count=0;

int main()
{
    Box b,b2;
    b.setlength(2.3).setheight(4.1).setbreadth(2.1).calculate();
    cout<<"Box objects: "<<Box::obj_count;
    return 0;
}
```

Output:

```
anwser is =19.803
Box objects: 2
```

Static Function Members:

➤ Static Function Members

- By declaring a function member as static, you make it independent of any particular object of the class.
- A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- Static member functions have a class scope and they do not have access to the **this** pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Static Function Members:

➤ Static Function Members

- By declaring a function member as static, you make it independent of any particular object of the class.
- A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- Static member functions have a class scope and they do not have access to the **this** pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

```
using namespace std;
class Box {
    float length;
    float breadth;
    float height;
    static int obj_count;
public:
    Box();
    Box& setlength(float length);
    Box& setbreadth(float breadth);
    Box& setheight(float height);
    void calculate();
    static int getCount();
```

Static Function Members:

➤ Static Function Members

➤ Declaration:

```
using namespace std;
class Box {
    float length;
    float breadth;
    float height;
    static int obj_count;
public:
    Box();
    Box& setlength(float length);
    Box& setbreadth(float breadth);
    Box& setheight(float height);
    void calculate();
    static int getCount();
```

Definition:

```
int Box::getCount() {
    return obj_count;
}

Box& Box::setlength(float a)
{
    if(a>0.0)
    {
        length=a;
    }

    return *this;
}

Box& Box::setbreadth(float b)
{
    if(b>0.0)
    {
        breadth=b;
    }

    return *this;
}

Box& Box::setheight(float c){
    if(c>0.0)
    {
        height =c;
    }

    return *this;
}
```

➤ Calling:

```
int main()
{
    Box b,b2;
    b.setlength(2.3).setheight(4.1).setbreadth(2.1).calculate();
    cout<<"Box objects: "<<Box::getCount() << endl;
    return 0;
}
```

Static Function Members:

➤ **Static Function Members**

➤ Why there is no this pointer in Static Function Members

- First, because static member functions are not attached to an object, they have no this pointer! This makes sense when you think about it -- the this pointer always points to the object that the member function is working on. Static member functions do not work on an object, so the this pointer is not needed.
- Second, static member functions can directly access other static members (variables or functions), but not non-static members. This is because non-static members must belong to a class object, and static member functions have no class object to work with

Reference material

➤ **For Practice Questions, refer to these books**

- C++ Programming From Problem Analysis To Program Design, 5th Edition, D.S.Malik. Chapter 12.
- C++ How to Program, Deitel & Deitel, 5th Edition, Prentice Hall.
- Object Oriented Programming in C++ by Robert Lafore.
- Object Oriented Software Construction, Bertrand Meyer's
- Object-Oriented Analysis and Design with applications, Grady Booch et al, 3Rd Edition, Pearson, 2007
- Web