Lecture 7
OOP

Khola Naseem
khola.naseem@uet.edu.pk

Excercise:

- In abc Software Company you are working as a programmer. You have to Create a class that implements the working of bank account. The class should perform the following tasks:
  - Store the balance.
  - Store the # of transactions performed by the user on the account.
  - Deposits money to the account.
  - Withdrawal money from the account.
  - Interest should be Calculate for the period.
  - Show the current account balance at time
  - Show the current number of transactions at any time.

Const member functions in C++:

- Constant member functions are those functions which are denied permission to change the values of the data members of their class.
- To make a member function constant, the keyword "const" is appended to the function prototype and also to the function definition header.
- the objects of a class can also be declared as **const**. an object declared as const cannot be modified and hence, can invoke only const member functions as these functions ensure not to modify the object.
- Syntax:
  - Return type FunctionName() const; // if definition is outside the class
    - Return type <class_name> : : FunctionName() const { body }
  - Return type FunctionName() const {body } //definition inside the class

Const member functions in C++:

- When a function is declared as const, it can be called on any type of object, const object as well as non-const objects.
- When a function is declared as non const, it can be called on only non-const objects.
- Why?
-

```
class Box{
    private:
        float area;

    public:
        void setarea(float a)
        {
            area=a;
        }
        float getarea()
        {
            area++;
            return area;

        }

};

int main(int argc, char** argv) {
    Box b1;
    b1.setarea(4);
    cout<<"area is "<<b1.getarea();
}
```

Output:

```
area is 5
--------------------------------
```

Credit: Khola Naseem

Const member functions in C++:

- When a function is declared as const, it can be called on any type of object, const object as well as non-const objects.
- When a function is declared as non const, it can be called on only non-const objects.
- Why?

```
class Box{
    private:
        float area;

    public:
        Box(float a)
        {
            area=a;
        }
        float getarea() const
        {
            area++;
            return area;

        }

};

int main(int argc, char** argv) {
    Box b1(5);
    cout<<"area is "<<b1.getarea();
}
```

Error:

E:\UET\Spring 23\OOP\Class\Constant member functi...	In member function 'float Box::getarea() const':
E:\UET\Spring 23\OOP\Class\Constant member function...	[Error] increment of member 'Box::area' in read-only object
E:\UET\Spring 23\OOP\Class\Makefile.win	recipe for target '"Constant member function.o"' failed

// to resolve the error, comment this line

Const member functions in C++:

- Define this function outside the class

Const member functions in C++:

- Constant data member:

```
class Box{
    private:
        const float area=5;

    public:
    /*  Box(float a)
        {
            area=a;
        }

        void setarea(int n)
        {
            area=n;


        }*/
        float getarea()
        {
            return area;
        }

};

int main(int argc, char** argv) {
    Box b1;
    //b1.setarea(3.3)
    cout<<"area is "<<b1.getarea();
}
```

Output:

```
area is 5
--------------------------------
```

Constant Objects C++:

- Constant Objects:

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

class Box{
    private:
        float area;

    public:
        Box(float b)
        {
            area=b;
        }
        void setarea(float a)
        {
            if(a>3)
            {
                area=a;
            }
        }
        float getarea() const
        {

            return area;

        }

};

int main(int argc, char** argv) {
    const Box b1(3);
//  b1.setarea(4);                  //Error
    cout<<"area is "<<b1.getarea();
}
```

Constant Objects C++:

- Const correctness:
  - Constant Objects can only access constant member function
- Output and Why?

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

class Box{
    private:
        float area;
    public:
        Box(float b)
        {
            area=b;
        }
        void setarea(float a)
        {
            if(a>3)
            {
                area=a;
            }
        }
        float getarea() const
        {
            setarea(5);
            return area;

        }
};

int main(int argc, char** argv) {
    const Box b1(3);
    //b1.setarea(4);                    //Error
    cout<<"area is "<<b1.getarea();
}
```

Credit: Khola Naseem

Constant Objects C++:

- Const correctness:
  - Constant Objects can only access constant member function
  - cannot call any non-const member functions from within a const member function

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

class Box{
    private:
        float area;
    public:
        Box(float b)
        {
            area=b;
        }
        void setarea(float a)
        {
            if(a>3)
            {
                area=a;
            }
        }
        float getarea() const
        {
            setarea(5);
            return area;

        }
};

int main(int argc, char** argv) {
    const Box b1(3);
    //b1.setarea(4);                    //Error
    cout<<"area is "<<b1.getarea();
}
```

Credit: Khola Naseem

Object of a class inside another class:

- Example:

```
class Salary {
private:
    int id;
    float Mon_Salary[12];
    float avg;
public:
    Salary()
    {
        id = 0;
        for(int i=0;i<12;i++)
        {
            Mon_Salary[i] = 0.0;
        }
        avg=0.0;
    }
    void readSalary()
    {

        cout << "Enter id: ";
        cin >> id;
        for(int i=0;i<12;i++)
        {
        cout << "\n Enter salary of month "<<i+1<<" :";
        cin >> Mon_Salary[i];
        avg=avg+Mon_Salary[i];
        }
        avg=avg/12;
    }
    void printSalary()
    {

        cout << "id: " << id << endl;
        cout<<"Average salary :"<<avg;
}
};
```

```
class Employee {
private:
    Salary obj;
    char name[30];
public:
    void readEmployee()
    {
        cout << "Enter name: ";
        cin.getline(name, 30);

        obj.readSalary();
    }
    void printEmployee(void)
   {
        cout << "Name: " << name << endl;
        obj.printSalary();
    }
};
int main()
{
    Employee e1;
    e1.readEmployee();
    e1.printEmployee();
    return 0;
}
```

Credit: Khola Naseem

Arrow operator (->):

- Arrow operator (->) in C++ also known as Class Member Access Operator
- It is used to access the public members of a class with the help of a pointer variable
- There is a .(dot) operator in C++ that is also used to access the members of a class.
- But .(dot operator) accesses the member or variable directly means without using the pointers whereas instead of accessing members directly the arrow operator(->) in C++ uses a pointer to access them.
- So the advantage of the -> operator is that it can deal with both pointer and non-pointer access but the dot(.) operator can only be used to access the members of a class
- Arrow (->) operator in C++ represents the same meaning which is done by the (*) asterisk operator and dot(.) operator. Arrow operator can be written in a different way which is using the combination of two operators *.. For example, (*s).element is the same as s->element.

Credit: Khola Naseem

Arrow operator (->):

- Arrow operator (->) in C++ also known as Class Member Access Operator

```
pointerVariableName->classMemberName
```

- Example:

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

Arrow operator (->):

- Second Example:

```
class classExample
{
private:
    int x;
public:
void setX(int a);
void print() const;

};

void classExample::setX(int a)
{
    x = a;
}
void classExample::print() const
{
   cout << "x = " << x << endl;
}

int main()
{
   classExample *cExpPtr;
   classExample cExpObject;
   cExpPtr = &cExpObject;
   cExpPtr->setX(5);
   cExpPtr->print();
   return 0;
}
```

Credit: Khola Naseem

Array of objects:

Example:

```
class Box{
    private:
        float length;

    public:
    Box()
    {
        length=2.2;

    }
    float getlength()
    {
        return length;
    }

    void setlength(float a)
    {
        if (a>5.3)
        {
            length=a;
        }
    }


};
int main(int argc, char** argv) {
    Box b[4];

    float new_len=0.0;
    for(int i=0;i<4;i++){
        cout<<"enter value new length : ";
        cin>>new_len;
        b[i].setlength(new_len);
    }
    for(int i=0;i<4;i++){
        cout<<"Get value of the length of "<<i<<" object: "<<b[i].getlength()<<endl;
    }
}
```

Credit: Khola Naseem

Array of objects:

- Example:

```
class Box
{
    private:
    double m_length ;
    double m_width ;
    double m_height;
    public:
    Box(double length, double width, double height);
    Box(double side); // Constructor for a cube
    Box(); // Default constructor
    Box(const Box& box); // Copy constructor
    double volume() const
    { return m_length * m_width * m_height;

    }
};
    Box::Box(double length, double width, double height) : m_length(length), m_width(width), m_height(height)
    {
        cout << "Box constructor 1 called." << std::endl;
    }
    Box::Box(double side)  // Constructor for a cube
    {

        cout << "Box constructor 2 called." << std::endl;
        Box(side, side, side);
    }
    Box::Box() // Default constructor

    {
         m_length=1.0;
         m_width =1.0;
         m_height=1.0;
        cout << "Default Box constructor called." << std::endl;
    }
    Box::Box(const Box& box) : m_length{box.m_length}, m_width{box.m_width}, m_height{box.m_height}
    {
        cout << "Box copy constructor called." << std::endl;
    }
```

Credit: Khola Naseem

Array of objects:

- Example:

```
int main(int argc, char** argv) {
    const Box box1 (2.0, 3.0, 4.0); // An arbitrary box
    Box box2 (5.0); // A box that is a cube
    cout << "box1 volume = " << box1.volume() <<endl;
    cout << "box2 volume = " << box2.volume() << endl;
    Box box3 (box2);
    cout << "box3 volume = " << box3.volume() <<endl; // Volume = 125
    cout << std::endl;
    Box boxes[6]={box1, box2, box3, Box (2.0)};
}
```

- Output:

```
Box constructor 1 called.
Box constructor 1 called.
Box constructor 2 called.
box1 volume = 24
box2 volume = 125
Box copy constructor called.
box3 volume = 125

Box copy constructor called.
Box copy constructor called.
Box copy constructor called.
Box constructor 1 called.
Box constructor 2 called.
Default Box constructor called.
Default Box constructor called.
```

Credit: Khola Naseem

Dynamic memory allocation:

- Example:

```
class Box{
    private:
        float length;
        float *breadth;
    public:
    Box(float a,float b)
    {
        length=a;
        breadth=new float;
        *breadth=b;
        cout<<"address of b1 breadth"<<breadth<<endl;
    }
    Box(Box &obj1)
    {
        length=obj1.length;
        breadth=new float;
        *breadth=*(obj1.breadth);
        cout<<"address of b2 breadth"<<breadth<<endl;
    }
    double Area()
    {
        return length*(*breadth);
    }

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

Credit: Khola Naseem

Dynamic memory allocation:

- Company Example:
- Create a class to store to store the information related to XYZ Company with the following data member
  - name
  - No_of_departments
  - *profit //to store the profit for five different years //dynamic memory allocation(array)
- Use an n-argument constructor to create an object of this class with some standard attribute values, for example,
  - Name = “XYZ”
  - No_of_departments = 5
  - float *profit=new float[5] //take values from user and store in the array
- Also create two class objects and implement the following line with the implementation all required functions
  - Company com2=com1
- Call destructor to free dynamically allocated memory

```
Name XYZ
depart 5
profit of the 1  year
34.555
profit of the 2  year
455.555
profit of the 3  year
7888.56
profit of the 4  year
8900.55
profit of the 5  year
3456.55

Info of company 2
Name XYZ
depart 5
profit of the 1  year
34.555
profit of the 2  year
455.555
profit of the 3  year
7888.56
profit of the 4  year
8900.55
profit of the 5  year
3456.55
```

Dynamic memory allocation:

- **Example:**
- Company Example

```
~company()
{
    delete[] profit;
}
```

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