FAST

National University of Computer and Emerging Sciences Peshawar

OOP Lab # 10

C++ (Classes and Objects)

Computer Instructor: Muhammad Abdullah Orakzai

DEPARTMENT OF COMPUTER SCIENCE

Programming

الذی علم بالقلم۔ علم الانسان ما لم یعلم۔

Contents

C++ this Pointer

In C++ programming, **this** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

- It can be used **to pass current object as a parameter to another method.**
- It can be used **to refer current class instance variable.**
- It can be used **to declare indexers.**

C++ this Pointer Example

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member (also instance variable)

       string name; //data member(also instance variable)
       float salary;

       Employee(int id, string name, float salary)
        {
            this->id = id;
            this->name = name;
            this->salary = salary;
        }
```

C++ this Pointer Example…

```
void display()
        {
            cout<<id<<"  "<<name<<"  "<<salary<<endl;

        }
};   // class body ends
```

C++ this Pointer Example…

```
int main(void) {
Employee e1 =Employee(101, “Ali", 890000); //creating an object of Employee
Employee e2=Employee(102, “Sania", 59000); //creating an object of Employee

e1.display();
e2.display();
    return 0;
}
```

Output:

101 Ali 890000
102 Sania 59000

C++ static Keyword

- In C++, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members.
- In C++, static can be field (variable), method, constructor, class, properties, operator and event.

Advantage of C++ static keyword

Memory efficient: Now we don't need to create instance for accessing the static members, so it saves memory.

Moreover, it belongs to the type, so it will not get memory each time when instance is created.

C++ Static Field

- A field which is declared as static is called static field.
- Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory.
- It is shared to all the objects.
- It is used to refer the common property of all objects such as rateOfInterest in case of Account, companyName in case of Employee etc.

- Let's see the simple example of static field in C++.

C++ static field example

```
#include <iostream>
using namespace std;
class Account {
   public:
       int accno;      //data member (also instance variable)
       string name;    //data member(also instance variable)
       static float rateOfInterest;
       Account(int accno, string name)
        {
            this->accno = accno;
            this->name = name;
        }
       void display()
        {
            cout<<accno<< " "<<name<< " "<<rateOfInterest<<endl;
        }
};
```

C++ static field example…

```
// Initialize static member of class Account
float Account::rateOfInterest=6.5;

int main(void) {
    Account a1 =Account(201, "Kashif"); //creating an object of Employee

    Account a2= Account(202, "Amir"); //creating an object of Employee

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

    return 0;
}
```

Output:
201 Kashif 6.5
202 Amir 6.5

Uses of static class data

Why would you want to use static member data ?

An example, suppose an object needed to know how many other objects of its class were in the program. In road-racing game**, for example**, a race car might want to know how many other cars are still in the race. In this case a static variable count could be included as a member of the class. All the objects would have access to this variable. It would be the same variable. It would be the same variable for all of them; they would all see the same count.

C++ static field example: Counting Objects

```
#include <iostream>
using namespace std;
class Count
{
private:
static int count; //only one data item for all objects
//note: “declaration” only!
public:
Count() //increments count when object created
{
    count++;
    //cout<<count;

}
int getcount() //returns count
{ return count; }
};
```

C++ static field example: Counting Objects…

```
//-----------------------------------------------------------
int Count::count = 0; //*definition* of count
////////////////////////////////////////////////////////////
int main()
{
Count c1, c2, c3; //create three objects
cout << "Count is "<<  c1.getcount() << endl; //each object
cout << "Count is " << c2.getcount() << endl; //sees the
cout << "Count is " << c3.getcount() << endl; //same value
return 0;
}
```

C++ static field example: Counting Objects…

The class Count in this example has one data item, count, which is type static int. The constructor for this class causes count to be incremented. In main() we define three objects of class Count. Since the constructor is called three times, count is incremented three times. Another member function, getcount(), returns the value in count. We call this function from all three objects, and—as we expected—each prints the same value. Here's the output:

- count is 3 <-------static data
- count is 3
- count is 3

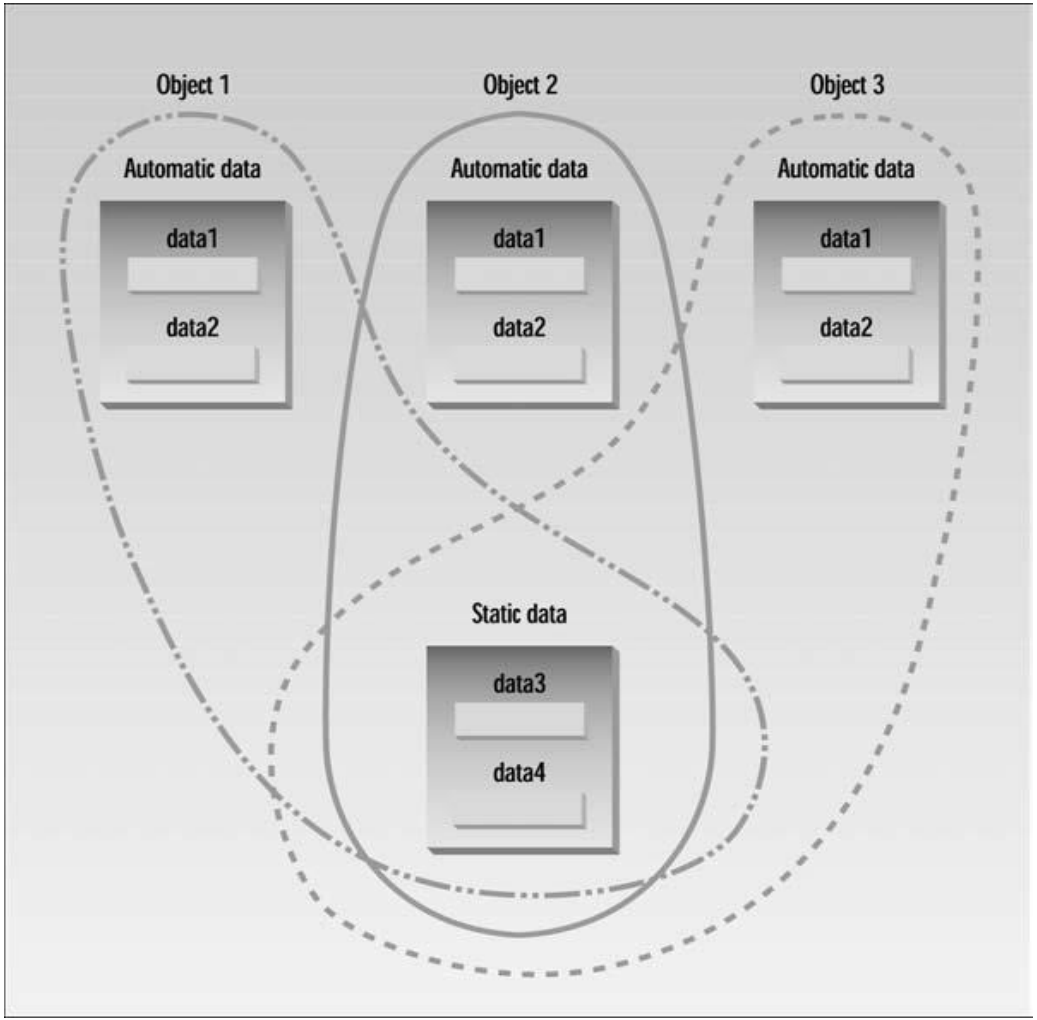
C++ static field example: Counting Objects…

If we had used an ordinary automatic variable—as opposed to a static variable—for count, each constructor would have incremented its own private copy of count once, and the output would have been

- count is 1 <------automatic data
- count is 1
- count is 1

C++ static field example: Counting Objects…

- Static class variables are not used as often as ordinary non-static variables, but they are important in many situations.
- Figure in next slide shows how static variables compare with automatic variables.


Object 1
Automatic data
data1
data2
Object 2
Automatic data
data1
data2
Object 3
Automatic data
data1
data2
Static data
data3
data4

C++ static field example: Counting Objects…

```
#include <iostream>
using namespace std;
class Account {
   public:
       int accno; //data member (also instance variable)
       string name;
       static int count;
       Account(int accno, string name)
        {
            this->accno = accno;
            this->name = name;
            count++;
        }
       void display()
        {
            cout<<accno<<" "<<name<<endl;
        }
};
```

C++ static field example: Counting Objects…

```
int Account::count=0;
int main(void) {

    Account a1 =Account(201, "Ali"); //creating an object of Account
    Account a2=Account(202, "Saad");
    Account a3=Account(203, "Sharjeel");

    a1.display();
    a2.display();
    a3.display();
    cout<<"Total Objects are: "<<Account::count;
    return 0;
}
```

Output:
201 Ali
202 Saad
203 Sharjeel
Total Objects are: 3

C++ Enumeration

- Enum in C++ is a data type that contains fixed set of constants.
- It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The C++ enum constants are static and final implicitly.
- C++ Enums can be thought of as classes that have fixed set of constants.

Points to remember for C++ Enum

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

example of enum data type used in C++ program

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

enum week { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };

int main()
{
    week day;
    day = Friday;
    cout << "Day: " << day+1<<endl;
    return 0;
}
```

Output:
Day: 5

example of enum data type used in C++ program...

```
#include <iostream>
using namespace std;
//specify enum type
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
```

example of enum data type used in C++ program…

```
int main()
{
days_of_week day1, day2; //define variables of type days_of_week

day1 = Mon; //give values to variables
day2 = Thu;
int diff = day2 - day1; //can do integer arithmetic
cout << "Days between = " << diff << endl;
if(day1 < day2) //can do comparisons
{
cout << "day1 comes before day2\n";
}
return 0;
}
```

Output

Days between = 3
day1 comes before day2

Friend Functions

Private members are accessed only within the class they are declared. Friend function is used to access the private and protected members of different classes. **It works as bridge between classes.**

- Friend function must be declared with **friend** keyword.
- Friend function must be declared in all the classes from which we need to access private or protected members.
- Friend function will be defined outside the class without specifying the class name.
- Friend function will be invoked like normal function, without any object.

Friend Functions…

- As we have seen in the previous sections, private and protected data or function members are normally only accessible by the code which is part of same class. However, situations may arise in which it is desirable to allow the explicit access to private members of class to other functions.
- If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend.

Declaration of friend function in C++

```
class class_name
{
   friend data_type function_name(argument/s);          // syntax of friend function.
};
```

In the above declaration, the friend function is preceded by the keyword friend. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword **friend or scope resolution operator**.

Example - 01 of friend function

```
#include <iostream>
using namespace std;
class Rectangle
{
    private :
        int length;
        int width;
    public:
        void setData(int len, int wid)
        {
            length = len;
            width = wid;
        }
```

Example - 01 of friend function…

```
        int getArea()
        {
            return length * width ;
        }

        friend double getCost(Rectangle);//friend of class Rectangle

};  // end of class body

//friend function getCost can access private member of class
double getCost (Rectangle rect)
{
    double cost;
    cost = rect.length * rect.width * 5;
    return cost;
}
```

Example - 01 of friend function…

```
int main ()
{
    Rectangle floor;
    floor.setData(20,3);
    cout<<"Area is:"<<floor.getArea()<<endl;
    cout << "Expense " << getCost(floor) << endl;
    return 0;
}
```

Output:
Area is:60
Expense 300

The getCost function is a friend of Rectangle. From within that function we have been able to access the members length and width, which are private members.

Example - 01 of friend function…

The getCost function is a friend of Rectangle. From within that function we have been able to access the members length and width, which are private members.

Example - 02 of friend function…

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

class RectangleTwo;

class RectangleOne
{
    int L,B; // private data members
    public:
    RectangleOne(int l,int b)
    {
        L = l;
        B = b;
    }
    //friend function
    friend void Sum(RectangleOne, RectangleTwo);
};  // end of RectangleOne class
```

Example - 02 of friend function…

```
class RectangleTwo
{
    int L,B; // private data members
    public:
    RectangleTwo(int l,int b)
    {
        L = l;
        B = b;
    }

    //friend function
    friend void Sum(RectangleOne, RectangleTwo);
};  // end of RectangleTwo
```

Example - 02 of friend function…

```
//friend function definiton
void Sum(RectangleOne R1,RectangleTwo R2)
{
    cout<<"\n\t\tLength\tBreadth";
    cout<<"\n Rectangle 1  :   "<<R1.L<<"\t  "<<R1.B;
    cout<<"\n Rectangle 2  :   "<<R2.L<<"\t  "<<R2.B;
    cout<<"\n ------------------------------";
    cout<<"\n\tSum   :   "<<R1.L+R2.L<<"\t  "<<R1.B+R2.B;
    cout<<"\n ------------------------------";
}
```

Example - 02 of friend function...

```
int main()
{
    RectangleOne Rec1(5,3);
    RectangleTwo Rec2(2,6);

    Sum(Rec1,Rec2);
    return 0;

}  // end of main() function
```

Output:

```
Length  Breadth
Rectangle 1 :  5     3
Rectangle 2 :  2     6
------------------------------
        Sum  :  7     9
------------------------------
```

Example - 02 of friend function…

- In the above example, we have created two classes RectangleOne and RectangleTwo with a common friend function Sum().
- Sum() method is receiving two objects, one of RectangleOne class and second of RectangleTwo class. This method is displaying the values of private data members of both the classes and also calculating the sum of both the rectangle classes.

C++ Friend Classes

- Like friend function, a class can also be a friend of another class. A friend class can access all the private and protected members of other class.
- In order to access the private and protected members of a class into friend class we must pass on object of a class to the member functions of friend class.
- One class member function can access the private and protected members of other class. We do it by declaring a class as friend of other class.

Example - 01 of C++ friend class

```
#include <iostream>
using namespace std;
class CostCalculator;
class Rectangle
{
    private :
        int length;
        int width;
    public:
        void setData(int len, int wid)
        {
            length = len;
            width = wid;
        }
        int getArea()
        {
            return length * width ;
        }
        friend class CostCalculator; //friend of class Rectangle
};
```

Example - 01 of C++ friend class...

```
//friend class costCalculator can access private member of class Rectangle

class CostCalculator
{
    public :
        double getCost (Rectangle rect)
        {
            double cost;
            cost = rect.length * rect.width * 5;
            return cost;
        }
};
```

Example - 01 of C++ friend class...

```
int main ()
{
    Rectangle floor;
    floor.setData(20,3);
    CostCalculator calc;
    cout<<"Area is:"<<floor.getArea()<<endl;
    cout << "Expense " << calc.getCost(floor) << endl;
    return 0;
}

//Note : An empty declaration of class costCalculator at top is necessary.
```

Output:

Area is:60
Expense 300

Example - 02 of C++ friend class

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

class Rectangle
{
    int L,B;

    public:
    Rectangle()
    {
        L=10;
        B=20;
    }

    friend class Square;          //Statement 1
};  // end of Rectangle class
```

Example - 02 of C++ friend class...

```
class Square
{
    int S;

    public:
    Square()
    {
        S=5;
    }
    void Display(Rectangle Rect)
    {
        cout<<"\n\n\tLength : "<<Rect.L;
        cout<<"\n\n\tBreadth : "<<Rect.B;
        cout<<"\n\n\tSide : "<<S;
    }
}; // end of Square class
```

Example - 02 of C++ friend class...

```
int main()
{
    Rectangle R;
    Square S;

    S.Display(R);          //Statement 2
    return 0;
}
```

Output:

Length : 10

Breadth : 20

Side : 5

Example - 02 of C++ friend class...

- In the above example, we have created two classes Rectangle and Square. Using statement 1 we have made Square class, a friend class of Rectangle class. In order to access the private and protected members of Rectangle class into Square class we must explicitly pass an object of Rectangle class to the member functions of Square class as shown in statement 2.
- This is similar to passing an object as function argument but the difference is, an object (R) we are passing as argument is of different class (Rectangle) and the calling object is of different class (Square).

bb

c

References

- https://beginnersbook.com/2017/08/cpp-data-types/
- http://www.cplusplus.com/doc/tutorial/basic_io/
- https://www.w3schools.com/cpp/default.asp
- https://www.javatpoint.com/cpp-tutorial
- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp
- https://www.programiz.com/
- https://ecomputernotes.com/cpp/

THANK YOU