



UNIVERSITY  
OF WOLLONGONG  
AUSTRALIA

csci251  
Advanced Programming  
OOP in C++



# Object Oriented Programming in C++

## Outline

### **Friend Functions**

- 3. What is friend function?**
- 4. Practice 1**

### **this Pointer**

- 5. What is this Pointer?**
- 6. Practice 2**

### **Overloading Operator**

- 7. What is Operator Overloaded**
- 8. Practice 3**

### **Static and Const Member**

- 9. What is static/const member?**

### **Pointer to a Class**

# 1. Friend Function



# What is Friend Function?

- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class.
- Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

# What is Friend Function?

- A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.
- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend.

# How to use Friend Function.

- Syntax is:

```
friend return_type function_name(Class & objClass) {...}
```

- Declare inside the class then define friend function outside the class

# Code - Demo

```
The number is: 8  
-----  
The number is: 14
```

```
#include <iostream>  
using namespace std;  
class Number{  
    private:  
        int num;  
    public:  
        Number(){num=8;}  
        void printNum(){  
            cout<<"The number is: "<<num<<endl;  
        }  
        friend void addNum(Number &n); //Pass by ?  
};  
void addNum(Number &n){  
    n.num = n.num + 6;  
}  
int main(){  
    Number n1;  
    n1.printNum();  
    cout<<"-----"<<endl;  
    // Call friend function  
    addNum(n1);  
    n1.printNum();  
    return 0;  
}
```

# Practice 1: friend Function

- Do in Code:Block
- Copy the code from lecture
- Define the friend function minusNum() that decrease the num by 2
  - $n.num = n.num - 2$
- In main() function:
  - After calling friend addNum(n1) and printing the num
  - Call the friend function minusNum(n1)
  - Access to the method printNum()
- Explain why we have the new number

# Code - Demo

```
#include <iostream>
#include "Library.h"
using namespace std;

Money::Money(string currency, double number, double rate):country(currency), amount(number), exchangeRate(rate){}

City::City(string cityName, string countryName, double NorthSouth, double EastWest):name(cityName), country(countryName), latitude(NorthSouth), longitude(EastWest){}

bool compare(Money& m, City& c)
{
    return (m.country == c.country);
}
```

```
#include <iostream>
class City;
class Money
{
    friend bool compare(Money&, City&);

private:
    string country;
    double amount;
    double exchangeRate;

public:
    Money(string, double, double);
};

class City
{
    friend bool compare(Money&, City&);

private:
    string name, country;
    double latitude, longitude;

public:
    City(string, string, double, double);
};
```

# Code - Demo

```
#include <iostream>
#include "Library.h"
using namespace std;

int main()
{
    Money nzMoney("NZ", 1000, 0.70);

    City C1("Sydney", "Australia", -33.53, 151.10);
    City C2("Auckland", "NZ", -43.33, 172.47);

    if (compare(nzMoney, C1))
        cout << "Different associated countries." << endl;
    else
        cout << "They are associated with the same country." << endl;

    if (compare(nzMoney, C2))
        cout << "Different associated countries." << endl;
    else
        cout << "They are associated with the same country." << endl;
}
```



2. “this” Pointer

# What is “this”?

**Concept:** Object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member functions.

**Important note:** Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

# How to use “this” Pointer

- Syntax is:

```
this -> class_component
```

# How to use “this” Pointer

## Demo:

- Create a class Square
  - Attribute: the length of square
  - Method: area of square

## Do:

Create a function that compares the area between two square using this Pointer

# Code - Demo

We can compare by the length of the square. But for the demo, we add function area

Constructor created.  
Constructor created.  
Two are NOT equal

```
#include <iostream>
using namespace std;
class Square {
private:
    int length;      // Length of a square
public:
// Constructor non-default
Square(int l) {
    cout <<"Constructor created." << endl;
    length = l;
}
int area() {
    return length * length;
}
bool compare(Square sq) {
    return this->area() == sq.area();
}
int main() {
    Square Square1(3);    // Create Square1
    Square Square2(4);    // Create Square2
    if(Square1.compare(Square2)) {
        cout << "Two are equal" << endl;
    } else {
        cout << "Two are NOT equal" << endl;
    }
    return 0;
}
```

# Practice 2: this Pointer

- Do in Code:Block
- Copy the code from lecture
- Create a function to add the areas of two square
  - Int addArea(Square sq)
- In main() function:
  - Create two objects from class Square with constructor
  - Print to screen the total area of two squares



# 3. Overloading Operator

# What is “Operator Overloading”?

**Concept:** Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined.

Like any other function, an overloaded operator has a return type and a parameter list.

# How to use “Operator Overloading”

- Syntax is:

```
Class operatorsymbol "+" (const Class & obj) { }
```

# How to use “Operator Overloading”

## Demo:

- Create a class Distance
  - Attribute: the length
  - Method: setter and getter

## Do:

Create a Operator+ overloading function to add two objects. In the main, create three objects and call operator overloading

# Code - Demo

```
Distance 1 is: 3
Distance 2 is: 4
-----
The total of 2 distance is: 7
```

```
#include <iostream>
using namespace std;
class Distance {
    private: int length;      // Length of a square
    public:
        void setLength(int l){length = l;}
        int getLength()const {return length;}
        Distance operator+(const Distance d) {
            Distance dTmp; // Create of temporary Distance
            dTmp.length = this->length + d.length;
            return dTmp;
        }
};
int main() {
    Distance d1;      // Create Distance 1
    Distance d2;      // Create Distance 2
    Distance d3;
    d1.setLength(3);
    d2.setLength(4);
    cout << "Distance 1 is: "<<d1.getLength()<<endl;
    cout << "Distance 2 is: "<<d2.getLength()<<endl;
    // Add two objects
    d3 = d1 + d2;
    cout <<"-----"<<endl;
    cout << "The total of 2 distance is: "<<d3.getLength()<<endl;
    return 0;
}
```

# Practice 3: operator Overloading

- Do in Code:Block
- Copy the code from lecture
- Create an operator- to minus two Distance object
  - Distance operator- (const Distance d)
- In main() function:
  - Create three objects from class Distance
  - Implement over the object to compute the deduction of two distance objects

# List of operators which can be overloaded

+	-	*	/	%	<sup>^</sup>
&		~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	
+ =	- =	/ =	% =	<sup>^</sup> =	& =
=	* =	<< =	>> =	[]	( )
->	->*	new	new []	delete	delete []

## 4. Static Members

# Static Local Variables

Concept: A static local variable is declared within a function, it is initialized only once and retains its value between function calls.

```
#include <iostream>

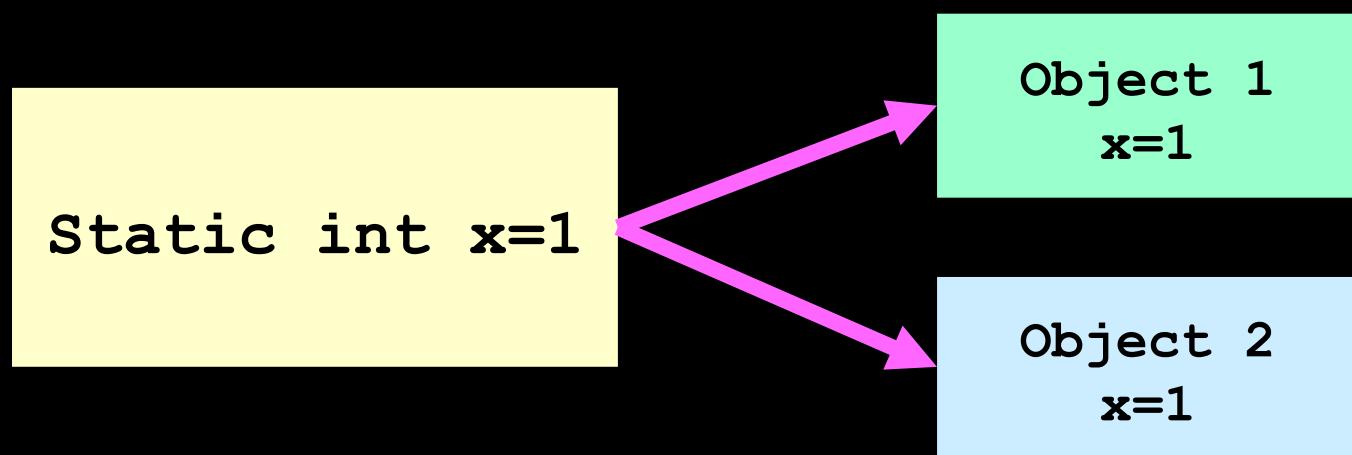
void counter() {
    static int count = 0; // Static local variable
    count++;
    std::cout << "Count: " << count << std::endl;
}

int main() {
    counter(); // Output: Count: 1
    counter(); // Output: Count: 2
    counter(); // Output: Count: 3
    return 0;
}
```

# Static Data vs Function Member

## What is Static Members?

Concept: When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.



# Static Data vs Function Member

## What is Static Members?

- A static member is shared by all objects of the class
- We cannot put in the class definition but it can be initialized outside the class using scope resolution (:) to identify which class it belongs.

# How to use “static data member”

- Inside the class A

```
static type variable;
```

- Outside the class

```
type A::variable=init;
```

# How to use “static data member”

## Demo:

- Create a class A
  - Declare static data member (count) in public modifier
  - Create a default constructor
  - Define static data member outside the class

## Do:

In the main function, generate a number of constructors then count the constructor created by using static data member

# Code - Demo

```
#include <iostream>
using namespace std;
class A {
public:
    static int count;
    A(){
        cout << "Constructor is created."<<endl;
        count++;
    }
};
// Initialising static data
int A::count=0;
int main() {
    A a1, a2;
    cout <<"Number of create constructors: "<<A::count;
    return 0;
}
```

Constructor is created.  
Constructor is created.  
Number of create constructors: 2

# Static Function Member

- 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.

# Const Members

**Const data members:** once initialized, they cannot be changed.

**Const function members:** they cannot modify the state of the object on which it is called.

Syntax of const function members:

```
return_type function_name() const;
```

```
return_type function_name () const { //function body }
```

Examples:

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

class Demo {
public:
    int x;
    void set_data(int a) { x = a; }

    // constant member function
    int get_data() const
    {
        x++; //it does not work
        return x;
    }
};

int main()
{
    Demo d;
    d.set_data(10);
    cout << endl << d.get_data();
    return 0;
}
```

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

class Demo {
public:
    int x;
    void set_data(int a) { x = a; }

    // constant member function
    int get_data(Demo b) const
    {
        //yes it works
        b.x++;
        return b.x;
    }
};

int main()
{
    Demo d;
    Demo b;
    b.set_data(10);
    d.set_data(10);
    cout << endl << d.get_data(b);
    return 0;
}
```

# Code - Demo

```
Init value: 0  
Number of create constructors: 2
```

## Practice 4

```
#include <iostream>  
using namespace std;  
class A {  
    private:  
        int x;  
        static int count;  
    public:  
        A(){  
            count++;  
        }  
        static int getCount(){  
            return count;  
        }  
};  
// Initialising static data  
int A::count=0;  
int main() {  
    cout<<"Init value: "<<A::getCount()<<endl;  
    A a1, a2;  
    cout <<"Number of create constructors: "<<A::getCount()  
;  
    return 0;  
}
```

## 5. Pointer to a Class

# What is Pointer to a Class?

A pointer to a C++ class is done exactly the same way as a pointer to a structure and to access members of a pointer to a class you use the member access **operator -> operator**, just as you do with pointers to structures. Also as with all pointers, you must initialize the pointer before using it.

# Code - Demo

```
Constructor created.  
Constructor created.  
the area of square 1 is: 9  
the area of square 2 is: 16
```

```
#include <iostream>  
using namespace std;  
class Square {  
private:  
    int length;      // Length of a square  
public:  
    // Constructor non-default  
    Square(int l) {  
        cout << "Constructor created." << endl;  
        length = l;  
    }  
    int area() {  
        return length * length;  
    }  
};  
int main() {  
    Square square1(3);      // Declare Square1  
    Square square2(4);      // Declare Square2  
    Square *ptr;  
    ptr = &square1;  
    cout << "the area of square 1 is: " << ptr->area() << endl;  
    ptr = &square2;  
    cout << "the area of square 2 is: " << ptr->area();  
    return 0;  
}
```