Part: 21

Classes, Public and Private access modifiers in C++

In this tutorial, we will discuss classes, public and private access modifiers in C++

Why use classes instead of structures

Classes and structures are somewhat the same but still, they have some differences. For example, we cannot hide data in structures which means that everything is public and can be accessed easily which is a major drawback of the structure because structures cannot be used where data security is a major concern. Another drawback of structures is that we cannot add functions in it.

Classes in C++

Classes are user-defined data-types and are a template for creating objects. Classes consist of variables and functions which are also called class members.

Public Access Modifier in C++

All the variables and functions declared under public access modifier will be available for everyone. They can be accessed both inside and outside the class. Dot (.) operator is used in the program to access public data members directly.

Private Access Modifier in C++

All the variables and functions declared under a private access modifier can only be used inside the class. They are not permissible to be used by any object or function outside the class.

An example program to demonstrate classes, public and private access modifiers are shown in Code Snippet 1.

```
class Employee
{
    private:
        int a, b, c;
    public:
        int d, e;
        void setData(int a1, int b1, int c1); // Declaration
        void getData(){
            cout<<"The value of a is "<<a<<endl;
            cout<<"The value of b is "<<b<<endl;
```

```
            cout<<"The value of c is "<<c<<endl;
            cout<<"The value of d is "<<d<<endl;
            cout<<"The value of e is "<<e<<endl;
        }
};

void Employee :: setData(int a1, int b1, int c1){
    a = a1;
    b = b1;
    c = c1;
}
```

Code Snippet 1: Class Program

As shown in Code Snippet 1, 1st we created an **"employee"** class, 2nd three integer variables **"int a"**, **"int b"**, and **"int c"** were declared under the private access modifier, 3rd two integer variables **"int d"** and **"int e"** was declared under the public access modifiers, 4th **"setData"** function was declared, 5th **"getData"** function was defined and values of all the variables are printed. 6th **"setData"** function was defined outside the **"employee"** class by using a scope resolution operator; **"setData"** function is used to assign values to the private member of the class. An example to create the object of the class and use its class members is shown in Code Snippet 2.

```
int main(){
    Employee harry;
    harry.d = 34;
    harry.e = 89;
    harry.setData(1,2,4);
    harry.getData();
    return 0;
}
```

Code Snippet 2: Creating Object Example

As shown in Code Snippet 2, 1st we created an object **"harry"** of the class **"employee"**; 2nd we assigned values to **"int d"** and **"int e"** which are public class members. If we try to assign values to the private class member's compiler will throw an error. 3rd we passed the values to the function **"setData"** and at the end, we called **"getData"** function which will print the values of all the variables. The output for the following program is shown in figure 1.

```
The value of a is 1
The value of b is 2
The value of c is 4
The value of d is 34
The value of e is 89
```

Figure 1: Class Program Output

As shown in figure 1, all the values of our data members are printed.

Code as described/written in the video

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

class Employee
{
    private:
        int a, b, c;
    public:
        int d, e;
        void setData(int a1, int b1, int c1); // Declaration
        void getData(){
            cout<<"The value of a is "<<a<<endl;
            cout<<"The value of b is "<<b<<endl;
            cout<<"The value of c is "<<c<<endl;
            cout<<"The value of d is "<<d<<endl;
            cout<<"The value of e is "<<e<<endl;
        }
};

void Employee :: setData(int a1, int b1, int c1){
    a = a1;
    b = b1;
    c = c1;
}

int main(){
    Employee harry;
    // harry.a = 134; -->This will throw error as a is private
    harry.d = 34;
    harry.e = 89;
    harry.setData(1,2,4);
    harry.getData();
    return 0;
}
```

Part: 22

OOPs Recap & Nesting of Member Functions in C++

In this tutorial, we will discuss the nesting of a member function in C++

Object-Oriented programming Recap

- **Stroustrup** initially named C++ language as C with classes because C++ language was almost the same as C language but they added a new concept of classes in it.
- Classes are the extension of structures in C language.
- Structures had limitations such as; members are public and no methods.
- Classes have some additional futures than structures such as; classes that can have methods and properties.
- Classes have a feature to make class members as public and private.
- In C++ objects can be declared along with class deceleration as shown in Code Snippet 1.

```
class Employee{
            // Class definition
} harry, rohan, lovish;
```

Code Snippet 1: Declaring Objects with Class Declaration

Nesting of Member Functions

If one-member function is called inside the other member function of the same class it is called nesting of a member function. A program to demonstrate the nesting of a member function is shown below.

```
class binary
{
private:
    string s;
    void chk_bin(void);

public:
    void read(void);
    void ones_compliment(void);
    void display(void);
};
```

Code Snippet 2: Binary Class

As shown in Code Snippet 2, we created a binary class that has, **"s"** string variable and **"chk_bin"** void function as private class members; and **"read"** void function, **"ones_compliment"** void function, and **"display"** void function as public class members. The definitions of these functions are shown below.

```
void binary::read(void)
{
    cout << "Enter a binary number" << endl;
    cin >> s;
}
```

Code Snippet 3: Read Function

As shown in Code Snippet 3, we have created a "read" function. This function will take input from the user at runtime.

```
void binary::chk_bin(void)
{
    for (int i = 0; i < s.length(); i++)
    {
        if (s.at(i) != '0' && s.at(i) != '1')
        {
            cout << "Incorrect binary format" << endl;
            exit(0);
        }
    }
}
```

Code Snippet 4: Check Binary Function

As shown in Code Snippet 4 we have created a **"chk_bin"** function. This **"for"** loop in the function will run till the length of the string and **"if"** condition in the body of the loop will check the whole string that if there are any values in the string other than **"1"** and **"0"**. If there are values other than **"1"** and **"0"** this function will output **"Incorrect binary format"**.

```
void binary::ones_compliment(void)
{
    chk_bin();
    for (int i = 0; i < s.length(); i++)
    {
        if (s.at(i) == '0')
        {
            s.at(i) = '1';
```

```
            }
            else
            {
                s.at(i) = '0';
            }
        }
    }
```

Code Snippet 5: One's Compliment

As shown in Code Snippet 5, in the body of the **"ones_compliment"** function; the **"chk_bin"** function is called, and as we have discussed above that if one-member function is called inside the other member function of the same class it is called **nesting of a member function**. The **"for"** loop inside the **"ones_compliment"** functions runs till the length of the string and the **"if"** condition inside the loop replaces the number **"0"** with **"1"** and **"1"** with **"0"**.

```
void binary::display(void)
{
    cout<<"Displaying your binary number"<<endl;
    for (int i = 0; i < s.length(); i++)
    {
        cout << s.at(i);
    }
    cout<<endl;
}
```

Code Snippet 6: Display Function

As shown in Code Snippet 6, the "for" loop inside display function runs till the length of the string and prints each value of the sting.

```
int main()
{
    binary b;
    b.read();
    // b.chk_bin();
    b.display();
    b.ones_compliment();
    b.display();

    return 0;
}
```

Code Snippet 7: Main Function

As shown in Code Snippet 7, we created an object **"b"** of the binary data type, and the functions **"read"**, **"display"**, **"ones_compliment"**, and **"display"** are called. The main thing to note here is that the function **"chk_bin"** is the private access modifier of the class so we cannot access it directly by using the object, it can be only accessed inside the class or by the member function of the class.

Code as described/written in the video

```
// OOPs - Classes and objects

// C++ --> initially called --> C with classes by stroustroup
// class --> extension of structures (in C)
// structures had limitations
//      - members are public
//      - No methods
// classes --> structures + more
// classes --> can have methods and properties
// classes --> can make few members as private & few as public
// structures in C++ are typedefed
// you can declare objects along with the class declarion like
this:
/* class Employee{
            // Class definition
        } harry, rohan, lovish; */
// harry.salary = 8 makes no sense if salary is private

// Nesting of member functions

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

class binary
{
private:
    string s;
    void chk_bin(void);

public:
    void read(void);
    void ones_compliment(void);
    void display(void);
};
```

```
void binary::read(void)
{
    cout << "Enter a binary number" << endl;
    cin >> s;
}
void binary::chk_bin(void)
{
    for (int i = 0; i < s.length(); i++)
    {
        if (s.at(i) != '0' && s.at(i) != '1')
        {
            cout << "Incorrect binary format" << endl;
            exit(0);
        }
    }
}
void binary::ones_compliment(void)
{
    chk_bin();
    for (int i = 0; i < s.length(); i++){
        if (s.at(i) == '0')
        {
            s.at(i) = '1';
        }
       else
        {
            s.at(i) = '0';
        }
    }
}
void binary::display(void){
    cout<<"Displaying your binary number"<<endl;
    for (int i = 0; i < s.length(); i++)
    {
        cout << s.at(i);
    }
    cout<<endl;
}
int main(){
    binary b;
    b.read();
    // b.chk_bin();
    b.display();
    b.ones_compliment();
    b.display();

    return 0;
}
```

Part: 23

C++ Objects Memory Allocation & using Arrays in Classes

In this tutorial, we will discuss objects memory allocation and using arrays in C++

Objects Memory Allocation in C++

The way memory is allocated to variables and functions of the class is different even though they both are from the same class.

The memory is only allocated to the variables of the class when the object is created. The memory is not allocated to the variables when the class is declared. At the same time, single variables can have different values for different objects, so every object has an individual copy of all the variables of the class. But the memory is allocated to the function only once when the class is declared. So, the objects don't have individual copies of functions only one copy is shared among each object.

Arrays in Classes

Arrays are used to store multiple values of the same type. An array is very helpful when multiple variables are required, instead of making multiple variables one array can be used which can store multiple values. Array stores data in sequential order. An example program to demonstrate the use of arrays in classes is shown below.

```
class Shop
{
    int itemId[100];
    int itemPrice[100];
    int counter;

public:
    void initCounter(void) { counter = 0; }
    void setPrice(void);
    void displayPrice(void);
};
```

Code Snippet 1: Shop Class

As shown in Code Snippet 1, we created a shop class which has, **"itemId[100]"** and **"itemPrice"** as integer array variable and **"counter"** variable as private class members; and **"initCounter"** void function, **"setPrice"** void function, and

"displayPrice" void function as public class members. The definitions of these functions are shown below.

```
void Shop ::setPrice(void)
{
    cout << "Enter Id of your item no " << counter + 1 << endl;
    cin >> itemId[counter];
    cout << "Enter Price of your item" << endl;
    cin >> itemPrice[counter];
    counter++;
}
```

Code Snippet 2: Set Price Function

As shown in Code Snippet 2, we have created a **"setPrice"** function. This function will take input for **"itemId"** and **"ItemPrice"** from the user at runtime. The value of the counter will be incremented by one every time this function will run.

```
void Shop ::displayPrice(void)
{
    for (int i = 0; i < counter; i++)
    {
        cout << "The Price of item with Id " << itemId[i] << " is
" << itemPrice[i] << endl;
    }
}
```

Code Snippet 3: Display Price Function

As shown in Code Snippet 3, the **"for"** loop inside the **"displayPrice"** function runs till the length of the counter and prints values of the array **"itemId"** and **"ItemPrice"**.

```
int main()
{
    Shop dukaan;
    dukaan.initCounter();
    dukaan.setPrice();
    dukaan.setPrice();
    dukaan.setPrice();
    dukaan.displayPrice();
    return 0;
}
```

Code Snippet 4: Main Function

As shown in Code Snippet 4, we created an object **"dukaan"** of the shop data type, and the functions **"initCounter"** is called. The function **"setPrice"** is called three times. Loops can also be used to call the function multiple times. The **"displayPrice"** function is also called in the main function. The output of the following program is shown in figure 1.

```
Enter Id of your item no 1
1001
Enter Price of your item
12
Enter Id of your item no 2
1002
Enter Price of your item
23
Enter Id of your item no 3
1003
Enter Price of your item
34
The Price of item with Id 1001 is 12
The Price of item with Id 1002 is 23
The Price of item with Id 1003 is 34
```

Figure 1: Program Output

As shown in figure 1, for the item 1 we entered the ID **"1001"** and price **"12"**; for the item 2 we entered the ID **"1002"** and price **"23"**; for the item 3 we entered the ID **"1003"** and price **"34"**. The Output of the program has displayed the ID and the price of each item.

Code as described/written in the video

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

class Shop
{
    int itemId[100];
    int itemPrice[100];
    int counter;

public:
    void initCounter(void) { counter = 0; }
    void setPrice(void);
    void displayPrice(void);
};
```

```
void Shop ::setPrice(void)
{
    cout << "Enter Id of your item no " << counter + 1 << endl;
    cin >> itemId[counter];
    cout << "Enter Price of your item" << endl;
    cin >> itemPrice[counter];
    counter++;
}

void Shop ::displayPrice(void)
{
    for (int i = 0; i < counter; i++)
    {
        cout << "The Price of item with Id " << itemId[i] << " is
" << itemPrice[i] << endl;
    }
}

int main()
{
    Shop dukaan;
    dukaan.initCounter();
    dukaan.setPrice();
    dukaan.setPrice();
    dukaan.setPrice();
    dukaan.displayPrice();
    return 0;
}
```

Part: 24

Static Data Members & Methods in C++ OOPS

In this tutorial, we will discuss static data members and methods in C++

Static Data Members in C++

When a static data member is created, there is only a single copy of the data member which is shared between all the objects of the class. As we have discussed in our previous lecture that if the data members are not static then every object has an individual copy of the data member and it is not shared.

Static Methods in C++

When a static method is created, they become independent of any object and class. Static methods can only access static data members and static methods. Static methods can only be accessed using the scope resolution operator. An example program is shown below to demonstrate static data members and static methods in C++.

```
class Employee
{
    int id;
    static int count;

public:
    void setData(void)
    {
        cout << "Enter the id" << endl;
        cin >> id;
        count++;
    }
    void getData(void)
    {
        cout << "The id of this employee is " << id << " and this
is employee number " << count << endl;
    }

    static void getCount(void){
        // cout<<id; // throws an error
        cout<<"The value of count is "<<count<<endl;
    }
};
```

Code Snippet 1: Employee Class

As shown in Code Snippet 1, we created an employee class that has integer **"id"** variable and **"count"** static integer variable as private class members; and **"setData"** void function, **"getData"** void function, and **"getCount"** static void function as public class members. These functions are explained below.

We have defined a **"setData"** function. This function will take input for **"id"** from the user at runtime and increment in the count. The value of the counter will be incremented by one every time this function will run.

We have defined a **"getData"** function. This function will print the values of the variables **"id"** and **"count"**. We have defined a static **"getCount"** function. This function will print the value of the variable count. The main thing to note here is that **"getCount"** function is static, so if we try to access any data members or member functions which are not static the compiler will throw an error.

```
// Count is the static data member of class Employee
int Employee::count; // Default value is 0

int main()
{
    Employee harry, rohan, lovish;
    // harry.id = 1;
    // harry.count=1; // cannot do this as id and count are
private

    harry.setData();
    harry.getData();
    Employee::getCount();

    rohan.setData();
    rohan.getData();
    Employee::getCount();

    lovish.setData();
    lovish.getData();
    Employee::getCount();

    return 0;
}
```

Code Snippet 2: main Program

As shown in Code Snippet 2:

- The count variable is declared whose default value is **"0"**.
- Then we created objects **"harry"**, **"rohan"**, and **"lovish"** of the employee data type
- The functions **"setData"**, **"getData"** are called by the object **"harry"**, the function **"getCount"** is called by using class name and scope resolution operator because it is a static method.
- The functions **"setData"**, **"getData"** are called by the object **"rohan"**, the function **"getCount"** is called by using class name and scope resolution operator because it is a static method.
- The functions **"setData"**, **"getData"** are called by the object **"lovish"**, the function **"getCount"** is called by using class name and scope resolution operator because it is a static method.

The output of the following program is shown in figures 1 and 2.

```
Enter the id
1
The id of this employee is 1 and this is employee number 1
The value of count is 1
```

Figure 1: Program Output 1

```
Enter the id
2
The id of this employee is 2 and this is employee number 2
The value of count is 2
Enter the id
3
The id of this employee is 3 and this is employee number 3
The value of count is 3
```

Figure 2: Program Output 2

As shown in figures 1 and 2, for the **"harry"** object we entered the ID **"1"**; for the **"rohan"** object we entered the ID **"2"**; and for the **"lovish"** object we entered the ID **"3"**. The Output of the program has displayed the ID and the count of each employee.

Code as described/written in the video

```
#include <iostream>
using namespace std;
class Employee
{
    int id;
    static int count;
public:
    void setData(void)
    {
        cout << "Enter the id" << endl;
        cin >> id;
        count++;
    }
    void getData(void)
    {
        cout << "The id of this employee is " << id << " and this
is employee number " << count << endl;
    }
    static void getCount(void){
        // cout<<id; // throws an error
        cout<<"The value of count is "<<count<<endl;
    }
};
// Count is the static data member of class Employee
int Employee::count; // Default value is 0

int main()
{
    Employee harry, rohan, lovish;
    // harry.id = 1;
    // harry.count=1; // cannot do this as id and count are
private
    harry.setData();
    harry.getData();
    Employee::getCount();

    rohan.setData();
    rohan.getData();
    Employee::getCount();

    lovish.setData();
    lovish.getData();
    Employee::getCount();

    return 0;
}
```

Part: 25

Array of Objects & Passing Objects as Function Arguments in C

In this tutorial, we will discuss an array of objects and passing objects as a function-arguments in C++

An array of Objects in C++

An array of objects is declared the same as any other data-type array. An array of objects consists of class objects as its elements. If the array consists of class objects it is called an array of objects. An example program to demonstrate the concept of an array of objects is shown below.

```
class Employee
{
    int id;
    int salary;

public:
    void setId(void)
    {
        salary = 122;
        cout << "Enter the id of employee" << endl;
        cin >> id;
    }

    void getId(void)
    {
        cout << "The id of this employee is " << id << endl;
    }
};
```

Code Snippet 1: Employee Class

As shown in Code Snippet 1, we created an employee class that has integer **“id”** variable and **“salary”** integer variable as private class members; and **“setId”** void function, **“getId”** void function as public class members. These functions are explained below. We have defined a **“setId”** function. In this function, the **“salary”** variable is assigned by the value **“122”** and the function will take input for **“id”** from the user at runtime. We have defined a **“getId”** function. This function will print the values of the variable’s **“id”**.

```
int main()
{
    Employee fb[4];
    for (int i = 0; i < 4; i++)
    {
        fb[i].setId();
        fb[i].getId();
    }

    return 0;
}
```

Code Snippet 2: main program

As shown in Code Snippet 2, we created an array **"fb"** of size **"4"** which is of employee data-type. The **"for"** loop is used to run **"setId"** and **"getId"** functions till the size of an array. The main thing to note here is that the objects can also be created individually but it is more convenient to use an array if too many objects are to be created. The output of the following program is shown in figure 1.

```
2
The id of this employee is 2
Enter the id of employee
3
The id of this employee is 3
Enter the id of employee
4
The id of this employee is 4
```

Figure 1: Employee Program Output

As shown in figure 1. As we input the Id for an employee it gives us the output of the employee Id.

Passing Object as Function Argument

Objects can be passed as function arguments. This is useful when we want to assign the values of a passed object to the current object. An example program to demonstrate the concept of passing an object as a function argument is shown below.

```
class complex{
    int a;
    int b;
    public:
        void setData(int v1, int v2){
            a = v1;
```

```
            b = v2;
        }
        void setDataBySum(complex o1, complex o2){
            a = o1.a + o2.a;
            b = o1.b + o2.b;
        }
        void printNumber(){
            cout<<"Your complex number is "<<a<<" +
"<<b<<"i"<<endl;
        }
};
```

Code Snippet 3: Complex Class

As shown in Code Snippet 3, we created a complex class that has integer **"a"** variable and **"b"** integer variable as private class members; and **"setData"** void function, **"setDataBySum"** void function, and **"printNumber"** void function as public class members. These functions are explained below.

We have defined a **"setData"** function. In this function the values are assigned to the variables **"a"** and **"b"** because they are private data members of the class and values cannot be assigned directly. We have defined a **"setDataBySum"** function. In this function, the values of two objects are added and then assigned to the variables **"a"** and **"b"**. We have defined a **"printNumber"** function. In this function, the values of the variable **"a"** and **"b"** are being printed.

```
int main(){
    complex c1, c2, c3;
    c1.setData(1, 2);
    c1.printNumber();

    c2.setData(3, 4);
    c2.printNumber();
    c3.setDataBySum(c1, c2);
    c3.printNumber();
    return 0;
}
```

Code Snippet 4: main program 2

As shown in Code Snippet 4:

- We have created object **"c1"**, **"c2"**, and**"c3"** of complex data-type.
- The object **"c1"** calls the **"setData"** and **"printNumber"** functions.

- The object **“c2”** calls the **“setData”** and **“printNumber”** functions.
- The object **“c3”** calls the **“setDataBySum”** and **“printNumber”** functions.

The output of the following program is shown in figure 2.

```
Your complex number is 1+2i
Your complex number is 3+4i
Your complex number is 4+6i
```

Figure 2: Complex Program Output

Code as described/written in the video

```
#include <iostream>
using namespace std;
class Employee
{
    int id;
    int salary;
public:
    void setId(void)
    {
        salary = 122;
        cout << "Enter the id of employee" << endl;
        cin >> id;
    }
    void getId(void)
    {
        cout << "The id of this employee is " << id << endl;
    }
};
int main()
{
    // Employee harry, rohan, lovish, shruti;
    // harry.setId();
    // harry.getId();
    Employee fb[4];
    for (int i = 0; i < 4; i++)
    {
        fb[i].setId();
        fb[i].getId();
    }
    return 0;
}
```

Code 25b as described/written in the video

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

class complex{
    int a;
    int b;

    public:
        void setData(int v1, int v2){
            a = v1;
            b = v2;
        }

        void setDataBySum(complex o1, complex o2){
            a = o1.a + o2.a;
            b = o1.b + o2.b;
        }

        void printNumber(){
            cout<<"Your complex number is "<<a<<" +
"<<b<<"i"<<endl;
        }
};

int main(){
    complex c1, c2, c3;
    c1.setData(1, 2);
    c1.printNumber();

    c2.setData(3, 4);
    c2.printNumber();

    c3.setDataBySum(c1, c2);
    c3.printNumber();
    return 0;
}
```

Part: 26

Friend Functions in C++

In this tutorial, we will discuss friend function in C++

Friend Function in C++

Friend functions are those functions that have the right to access the private data members of class even though they are not defined inside the class. It is necessary to write the prototype of the friend function. One main thing to note here is that if we have written the prototype for the friend function in the class it will not make that function a member of the class. An example program to demonstrate the concept of friend function is shown below.

```
class Complex{
    int a, b;
    friend Complex sumComplex(Complex o1, Complex o2);
    public:
        void setNumber(int n1, int n2){
            a = n1;
            b = n2;
        }
    // Below line means that non member - sumComplex funtion is
allowed to do anything with my private parts (members)
        void printNumber(){
            cout<<"Your number is "<<a<<" + "<<b<<"i"<<endl;
        }
};
Complex sumComplex(Complex o1, Complex o2){
    Complex o3;
    o3.setNumber((o1.a + o2.a), (o1.b+o2.b))
    ;
    return o3;
}
```

Code Snippet 1: Complex Class

As shown in Code Snippet 1, we created a complex class that has integer **“a”** variable and **“b”** integer variable as private class members; and **“setNumber”** void function, **“printNumber”** void function as public class members. The **“sumComplex”** friend function prototype is written as well in the complex class. These functions are explained below.

We have defined a **"setNumber"** function. In this function the values are assigned to the variables **"a"** and **"b"** because they are private data members of the class and values cannot be assigned directly. We have defined a **"printNumber"** function. In this function, the values of the variable **"a"** and **"b"** are being printed. We have defined a **"sumComplex"** friend function. In this function, the object "o3" is created which calls the **"setNumber"** function and passes the values of two objects after performing addition on them.

```
int main(){
    Complex c1, c2, sum;
    c1.setNumber(1, 4);
    c1.printNumber();

    c2.setNumber(5, 8);
    c2.printNumber();

    sum = sumComplex(c1, c2);
    sum.printNumber();

    return 0;
}
```

Code Snippet 2: main Program

As shown in Code Snippet 2:

- We have created object **"c1"**, **"c2"**, and **"sum"** of complex data-type.
- The object **"c1"** calls the **"setNumber"** and **"printNumber"** functions.
- The object **"c2"** calls the **"setNumber"** and **"printNumber"** functions.
- The function **"sumComplex"** is called and the values are assigned to the **"sum"**.
- The object **"sum"** calls the **"printNumber"** functions.

The output of the following program is shown in figure 1.

```
Your number is 1 + 4i
Your number is 5 + 8i
Your number is 6 + 12i
```

Figure 1: Complex Program Output

As shown in figure 1, the output of the complex number program is printed.

Properties of Friend Function

- Not in the scope of the class
- Since it is not in the scope of the class, it cannot be called from the object of that class, for example, **`sumComplex()`** is invalid
- A friend function can be invoked without the help of any object
- Usually contain objects as arguments
- Can be declared under the public or private access modifier, it will not make any difference
- It cannot access the members directly by their names, it needs (object_name.member_name) to access any member.

Code as described/written in the video

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

// 1 + 4i
// 5 + 8i
// -------
// 6 + 12i
class Complex{
    int a, b;
    friend Complex sumComplex(Complex o1, Complex o2);
    public:
        void setNumber(int n1, int n2){
            a = n1;
            b = n2;
        }

        // Below line means that non member - sumComplex funtion
is allowed to do anything with my private parts (members)
        void printNumber(){
            cout<<"Your number is "<<a<<" + "<<b<<"i"<<endl;
        }
};

Complex sumComplex(Complex o1, Complex o2){
    Complex o3;
    o3.setNumber((o1.a + o2.a), (o1.b+o2.b))
    ;
    return o3;
}
```

```
int main(){
    Complex c1, c2, sum;
    c1.setNumber(1, 4);
    c1.printNumber();

    c2.setNumber(5, 8);
    c2.printNumber();

    sum = sumComplex(c1, c2);
    sum.printNumber();

    return 0;
}

/* Properties of friend functions
1. Not in the scope of class
2. since it is not in the scope of the class, it cannot be called
from the object of that class. c1.sumComplex() == Invalid
3. Can be invoked without the help of any object
4. Usually contans the objects as arguments
5. Can be declared inside public or private section of the class
6. It cannot access the members directly by their names and need
object_name.member_name to access any member.

*/
```

Part: 27

Friend Classes & Member Friend Functions in C++

In this tutorial, we will discuss friend classes and member friend functions in C++

Member Friend Functions in C++

Friend functions are those functions that have the access to private members of the class in which they are declared. The main thing to note here is that only that function can access the member function which is made a friend of the other class. An example of the friend function is shown below.

```
class Complex{
    int a, b;
    // Individually declaring functions as friends
     friend int Calculator ::sumRealComplex(Complex, Complex);
     friend int Calculator ::sumCompComplex(Complex, Complex);
public:
    void setNumber(int n1, int n2){
        a = n1;
        b = n2;a
    }
    void printNumber(){
        cout << "Your number is " << a << " + " << b << "i" <<
endl;
    }
};
int Calculator ::sumRealComplex(Complex o1, Complex o2){
    return (o1.a + o2.a);
}
int Calculator ::sumCompComplex(Complex o1, Complex o2){
    return (o1.b + o2.b);
}
```

Code Snippet 1: Friend function example

As shown in a code snippet 1, a complex class is created which consists of two friend functions **“sumRealComplex”** and **“sumCompComplex”** of the calculator class. The main thing to note here is that **“sumRealComplex”** and **“sumCompComplex”** are the friend functions of complex class so they can access all the private members of the complex class.

Friend Classes in C++

Friend classes are those classes that have permission to access private members of the class in which they are declared. The main thing to note here is that if the class is made friend of another class then it can access all the private members of that class. An example program to demonstrate friend classes in C++ is shown below.

```
// Forward declaration
class Complex;

class Calculator
{
public:
    int add(int a, int b)
    {
        return (a + b);
    }

    int sumRealComplex(Complex, Complex);
    int sumCompComplex(Complex, Complex);
};
```

Code Snippet 2: Calculator Class

As shown in code snippet 2, a complex class is declared at the top which is known as forward declaration. Forward declaration hints to the compiler that this class is declared somewhere forward in the code. After that calculator class is defined this consists of three public member functions, **"add"**, **"sumRealComplex"**, and **"sumCompComplex"**. The **"add"** function will add the values of **"a"** and **"b"** and return the value. The **"sumRealComplex"** and **"sumCompComplex"** are taking two objects of the complex class. The code for the complex class is shown below.

```
class Complex
{
    int a, b;
    // Individually declaring functions as friends
    // friend int Calculator ::sumRealComplex(Complex, Complex);
    // friend int Calculator ::sumCompComplex(Complex, Complex);

    // Aliter: Declaring the entire calculator class as friend
    friend class Calculator;
```

```
public:
    void setNumber(int n1, int n2)
    {
        a = n1;
        b = n2;a
    }

    void printNumber()
    {
        cout << "Your number is " << a << " + " << b << "i" <<
endl;
    }
};

int Calculator ::sumRealComplex(Complex o1, Complex o2)
{
    return (o1.a + o2.a);
}

int Calculator ::sumCompComplex(Complex o1, Complex o2)
{
    return (o1.b + o2.b);
}
```

Code Snippet 3: Complex Class

As shown in code snippet 3, a complex class is defined which consists of, two private data members **"a"** and **"b"**, and two public member functions **"setNumber"** and **"printNumber"**. The function **"setNumber"** will assign the values to the variables **"a"** and **"b"**. The function **"printNumber"** will print the values of the variables **"a"** and **"b"**. Two functions **"sumRealComplex"** and **"sumCompComplex"** are defined at the end. The function **"sumRealComplex"** will add the real values and the function "**sumCompComplex"** will add the complex value. The main program is shown below.

```
int main()
{
    Complex o1, o2;
    o1.setNumber(1, 4);
    o2.setNumber(5, 7);
    Calculator calc;
    int res = calc.sumRealComplex(o1, o2);
    cout << "The sum of real part of o1 and o2 is " << res <<
endl;
    int resc = calc.sumCompComplex(o1, o2);
```

```
    cout << "The sum of complex part of o1 and o2 is " << resc <<
endl;
    return 0;
}
```

Code snippet 4: Main Program

As shown in code snippet 4, 1st two objects **"o1"** and **"o2"** of the **"complex"** data type are declared. 2nd **"setNumber"** function is called with the **"o1"** and **"o2"** objects and the values are passed. 3rd object **"calc"** of the calculator data type is declared. 4th **"sumRealComplex"** function is called by the **"calc"** object and the object **"o1"** and **"o2"** are passed to it. 5th **"sumCompComplex"** function is called by the **"calc"** object and the object **"o1"** and **"o2"** are passed to it. The output of the following program is shown in figure 1.

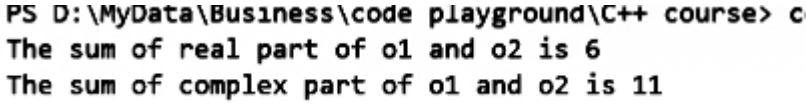

Figure 1: Program Output

As shown in figure 1, the sum of the real part is shown which is "6" and the sum of the complex part is shown which is **"11".**

Part: 28

More on C++ Friend Functions (Examples & Explanation)

In this tutorial, we will discuss more on friend functions in C++ with examples

Friend Functions in C++

As we have already discussed in previous lectures friend functions are those functions that can access the private data members of the other class. An example program to demonstrate friend functions in C++ is shown below.

Friend Function Example 1

```
class Y;
class X{
    int data;
    public:
        void setValue(int value){
            data = value;
        }
    friend void add(X, Y);
};
class Y{
    int num;
    public:
        void setValue(int value){
            num = value;
        }
    friend void add(X, Y);
};
void add(X o1, Y o2){
    cout<<"Summing data of X and Y objects gives me "<< o1.data +
o2.num;
}
```

Code Snippet 1: Friend Function Example 1

As shown in a code snippet 1,

- 1st class **"Y"** is declared at the top which is known as forward declaration to let the compiler know that this class is defined somewhere in the program.
- 2nd class **"X"** is defined which consists of private data member **"data"** and public member function **"setValue"** which assigns the value to the private data member **"data"**. At the end friend function **"add"** is declared.

- 3rd class **"Y"** is defined which consists of private data member **"num"** and public member function **"setValue"** which assigns the value to the private data member **"num"**. At the end friend function **"add"** is declared.
- 4th function **"add"** is defined which add the value of the objects of class **"X"** and **"Y"** and print it.

The main program is shown in Code Snippet 2.

```
int main(){
    X a1;
    a1.setValue(3);
    Y b1;
    b1.setValue(15);
    add(a1, b1);
    return 0;
}
```

Code Snippet 2: Main Program

As shown in Code Snippet 2,

- 1st object **"a1"** of the data type **"X"** is declared
- 2nd function **"setValue"** is called by the object **"a1"** and the value **"3"** is passed
- 3rd object **"b1"** of the data type **"Y"** is declared
- 4th function **"setValue"** is called by the object **"b1"** and the value **"15"** is passed
- 5th function **"add"** is called and the objects **"a1"** and **"b1"** are passed to it. The function **"add"** will add the values of both objects and print them.

The output of the following program is shown in figure 1.

```
Summing data of X and Y objects gives me 18
PS D:\MyData\Business\code playground\C++ course>
```

Figure 1: Program Output 1

As shown in figure 1, the sum of both values is shown which is "18".

Friend Function Example 2

```
class c2;

class c1{
    int val1;
    friend void exchange(c1
& , c2 &);
    public:
        void indata(int a){
            val1 = a;
        }

        void display(void){
            cout<< val1
<<endl;
        }
};

class c2{
    int val2;
    friend void exchange(c1
&, c2 &);
    public:
        void indata(int a){
            val2 = a;
        }

        void display(void){
            cout<< val2
<<endl;
        }
};

void exchange(c1 &x, c2
&y){
    int tmp = x.val1;
    x.val1 = y.val2;
    y.val2 = tmp;
}
```

Code Snippet 3: Friend Function Example 2

As shown in a code snippet 3,

- 1st class **"c2"** is declared at the top which is known as forward declaration to let the compiler know that this class is defined somewhere in the program.
- 2nd class **"c1"** is defined which consists of private data member **"val1"** and friend function **"exchange"** which takes reference variables **"c1"** and **"c2"** as parameters. The public member function **"indata"** is defined which assigns the value to the private data member **"val1"** and the function **"display"** prints the value of the data member **"val1"**.
- 3rd class **"c2"** is defined which consists of private data member **"val2"** and friend function **"exchange"** which takes reference variables **"c1"** and **"c2"** as parameters. The public member function **"indata"** is defined which assigns the value to the private data member **"val2"** and the function **"display"** prints the value of the data member **"val2"**.
- 4th function **"exchange"** is defined which swap the values.

The main program is shown in Code Snippet 4.

```
int main(){
    c1 oc1;
    c2 oc2;

    oc1.indata(34);
    oc2.indata(67);
    exchange(oc1, oc2);

    cout<<"The value of c1 after exchanging becomes: ";
    oc1.display();
    cout<<"The value of c2 after exchanging becomes: ";
    oc2.display();

    return 0;
}
```

Code Snippet 4: Main program

As shown in Code Snippet 4,

- 1st object **"oc1"** of the data type **"c1"** is declared

- 2nd object **"oc2"** of the data type **"c2"** is declared
- 3rd function **"indata"** is called by the object **"oc1"** and the value **"34"** is passed
- 4th function **"indata"** is called by the object **"oc2"** and the value **"67"** is passed
- 5th function **"exchange"** is called and the objects **"oc1"** and **"oc2"** are passed to it. The function **"exchange"** will swap both values and
- 6th function **"display"** is called by the objects **"oc1"** and **"oc2"** which will print their values.

The output of the following program is shown in figure 2.

```
The value of c1 after exchanging becomes: 67
The value of c2 after exchanging becomes: 34
```

Figure 2: Program Output 2

As shown in figure 2, the values are swapped.

Part: 29

Constructors In C++

In this tutorial, we will discuss constructors in C++

Constructors in C++

A constructor is a special member function with the same name as the class. The constructor doesn't have a return type. Constructors are used to initialize the objects of its class. Constructors are automatically invoked whenever an object is created.

Important Characteristics of Constructors in C++

- A constructor should be declared in the public section of the class
- They are automatically invoked whenever the object is created
- They cannot return values and do not have return types
- It can have default arguments
- We cannot refer to their address

An example program to demonstrate the concept of the constructor is shown below.

```
#include <iostream>
using namespace std;
class Complex{
    int a, b;
public:
    // Creating a Constructor
    // Constructor is a special member function with the same name
as of the class.
    //It is used to initialize the objects of its class
    //It is automatically invoked whenever an object is created
  Complex(void); // Constructor declaration
  void printNumber(){
        cout << "Your number is " << a << " + " << b << "i" <<
endl;
    }
};
Complex ::Complex(void) // ----> This is a default constructor as
it takes no parameters
{
    a = 10;
    b = 0;
    // cout<<"Hello world";
}
```

Code Snippet 1: Constructor Example Program

As shown in a code snippet 1,

- 1st **"complex"** class is defined which consists of private data members **"a"** and **"b"**.
- 2nd default constructor of the **"complex"** class is declared.
- 3rd function **"printNumber"** is defined which will print the values of the data members **"a"** and **"b"**.
- 4th default constructor is defined which will assign the values to the data members **"a"** and **"b"**. The main things to note here are that whenever a new object will be created this constructor will run and if the parameters are not passed to the constructor it is called a default constructor.

The main program is shown in code snippet 2.

```
int main(){
    Complex c1, c2, c3;
    c1.printNumber();
    c2.printNumber();
    c3.printNumber();
    return 0;
}
```

Code Snippet 2: Main Program

As shown in Code Snippet 2,

- 1st objects **"c1"**, **"c2"**, and **"c3"** of the complex data type are created. The main thing to note here is that when we are creating objects the constructor will run for each object and will assign the values.
- 2nd function **"printNumber"** is called by the objects **"c1"**, **"c2"**, and **"c3"**.

The output for the following program is shown in figure 1.

```
Your number is 10 + 0i
Your number is 10 + 0i
Your number is 10 + 0i
PS D:\MyData\Business\code playground\C++ course>
```

Figure 1: Program Output

As shown in figure 1, whenever a **"printNumber"** function is called it prints the values which are being assigned through the constructor.

Part: 30

Parameterized and Default Constructors In C++

In this tutorial, we will discuss parameterized and default constructors in C++

Parameterized and Default Constructors in C++

Parameterized constructors are those constructors that take one or more parameters. Default constructors are those constructors that take no parameters. The main things to note here are that constructors are written in the public section of the class and the constructors don't have a return type. An example program to demonstrate the concept of the constructor is shown below.

Parameterized Constructors Example Program 1

```
#include<iostream>
using namespace std;
class Complex{
    int a, b;
public:
    Complex(int, int); // Constructor declaration
    void printNumber(){
        cout << "Your number is " << a << " + " << b << "i" <<
endl;
    }
};
Complex ::Complex(int x, int y) // ----> This is a parameterized
constructor as it
{                                       takes 2 parameters
    a = x;
    b = y;
    // cout<<"Hello world";
}
```

Code Snippet 1: Parameterized Constructor Example Program 1

As shown in a code snippet 1,

- 1st **"complex"** class is defined which consists of private data members **"a"** and **"b"**.
- 2nd parameterized constructor of the **"complex"** class is declared which takes two parameters.

- 3rd function **"printNumber"** is defined which will print the values of the data members **"a"** and **"b"**.
- 4th parameterized constructor is defined which takes two parameters and assigns the values to the data members **"a"** and **"b"**. The main things to note here are that whenever a new object will be created this constructor will run.

The main program is shown in code snippet 2.

```
int main(){
    // Implicit call
    Complex a(4, 6);
    a.printNumber();
    // Explicit call
    Complex b = Complex(5, 7);
    b.printNumber();

    return 0;
}
```

Code Snippet 2: Main Program

As shown in Code Snippet 2,

- 1st parameterized constructor is called implicitly with the object **"a"** and the values **"4"** and **"6"** are passed
- 2nd function **"printNumber"** is called which will print the values of data members
- 3rd parameterized constructor is called explicitly with the object **"b"** and the values **"5"** and **"7"** are passed
- 4th function **"printNumber"** is called again which will print the values of data members

The output for the following program is shown in figure 1.

```
o tut30 } ; if ($?) { .\tut30
Your number is 4 + 6i
Your number is 5 + 7i
PS D:\MyData\Business\code pla
```

Figure 1: Program Output 1

Parameterized Constructors Example Program 2

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

class Point{
    int x, y;
    public:
        Point(int a, int b){
            x = a;
            y = b;
        }
        void displayPoint(){
            cout<<"The point is ("<<x<<", "<<y<<")"<<endl;
        }

};
```

Code Snippet 3: Parameterized Constructor Example Program 2

As shown in Code Snippet 3,

- 1st **"point"** class is defined which consists of private data members **"x"** and **"y"**.
- 2nd parameterized constructor of the **"point"** class is defined which takes two parameters and assigns the values to the private data members of the class.
- 3rd function **"displayPoint"** is defined which will print the values of the data members **"x"** and **"y"**.

The main program is shown in code snippet 4.

```
int main(){
    Point p(1, 1);
    p.displayPoint();

    Point q(4, 6);
    q.displayPoint();
    return 0;
}
```

Code Snippet 4: Main Program

As shown in Code Snippet 4,

- 1st parameterized constructor is called implicitly with the object **"p"** and the values **"1"** and **"1"** are passed

- 2nd function **"displayPoint"** is called which will print the values of data members
- 3rd parameterized constructor is called implicitly with the object **"q"** and the values **"4"** and **"6"** are passed
- 4th function **"displayPoint"** is called which will print the values of data members

The output for the following program is shown in figure 2.

```
-o tut30b } ; if ($?) { .
The point is (1, 1)
The point is (4, 6)
PS D:\MyData\Business\cod
```

Figure 2: Program Output **2**