In this lesson you will learn

- More about Initialization
- Friend Functions
- Overloading Operators

Initializing Arrays of Classes

You should already know how to *initialize* Arrays, objects of types that are classes with only public members and, possibly, Arrays of such objects, while declaring them:

1. **int A[5] = {50, 40, 30, 20, 10};**
2. **class pair { public: int x, y;};**
   pair p ={5,10};
3. **pair P[3] = { {1,2}, {2,4}, {3,6}};**

Note that
class pair { private: int x, y;};
pair p ={5,10};

does NOT work. You must use a constructor to initialize objects that have private members.


```
#include <iostream.h>
//Initialize1.cpp
//Examples of  initialization
class  pair
  { public: int x, y;};

void main()
{ int A[5] = {50, 40, 30, 20, 10};
  int i;

  for (i=0; i<=4; i++)
    cout << "A[" <<i <<"]=" << A[i] << endl;

  pair p ={5,10};
  cout << "p.x = "   << p.x;
  cout << "  p.y = " << p.y <<  endl;

  pair P[3] = { {1,2}, {2,4}, {3,6}};
  for (i=0; i<=2; i++)
    {  cout << "P[" <<i<<"].x = "  << P[i].x;
       cout << "  P["<<i<<"].y = " << P[i].y << endl;
    };
}
```



```
*****************OUTPUT*************

A[0]=50
A[1]=40
A[2]=30
A[3]=20
A[4]=10

p.x = 5  p.y = 10

P[0].x = 1  P[0].y = 2
P[1].x = 2  P[1].y = 4
P[2].x = 3  P[2].y = 6
```

We now see how to initialize an array of *Classes*
Let's rename *ComplexClass* as *CC*.
Suppose *CC* has three constructors:

```
CC(int x, int y);    //2 parameters
CC(int x);           //1 parameter
CC();                //Default constructor
```

To initialize an array of *CC* we write

CC A[4] ={CC(1,1), CC(), CC(3), CC(5,6)};

The syntax is

$\{ArrayName[n] = \mathrm{Cons}_0, \mathrm{Cons}_1, \ldots, \mathrm{Cons}_{n-1}\}$

where Cons_i is the constructor that should be applied to the ith object in the array.

Note: The *default constructor* must be called using $CC()$ and not CC


```
//Initialize2.cc
//Example of how to
//intialize an array of objects

#include <iostream.h>

class CC
{
  public:
         //Basic Constructor
         //Initializes number to be x + y*i
         CC(int x, int y);

         //Basic Constructor
         //Initializes number to be x + 0*i
         CC(int x);

         // Default Consructor
         CC();

         // Set number to be x + y*i
         void set(int x, int y);

         // Print in the form of: real+imagine*i.
         void print();

         // Add complex number C to the current number
         // and return value.
         CC Add(CC const &C);

         private:
                int  Real;
                int  Imag;
};
```



```
CC::CC(int x, int y)
{ cout << "Using Two Parameter  Constructor" << endl;
  Real =x; Imag =y;}

CC::CC(int x)
{ cout << "Using One Parameter  Constructor" << endl;
  Real =x; Imag =0;}

CC::CC()
{ cout << "Using Default Constructor" << endl;
  Real =0; Imag =0;}

void CC::set(int x, int y)
{ Real =x; Imag =y;}

void CC::print()
{cout << Real  << " + " << Imag << "*i"<<endl;}

CC CC::Add(CC const &C)
{ CC temp;
  temp.Real = Real + C.Real;
  temp.Imag = Imag + C.Imag;
  return temp;
}
```



```
int main()
{ cout << "Initialization of a" << endl;
  CC a;

  cout << endl << "Initialization of Array A"
       << endl;
  CC A[4] ={CC(1,1), CC(), CC(3), CC(5,6)};

  cout << endl << "Initialization of b"
       << endl;
  CC b(4);

  cout << endl << "Printing values in A"
       << endl;
  int i;
  for(i=0; i<4; i++)
    {cout <<"A[" << i << "]= ";
     A[i].print();
    }
}
```

```
**************OUTPUT******************

Initialization of a
Using Default Constructor

Initialization of Array A
Using Two Parameter  Constructor
Using Default Constructor
Using One Parameter  Constructor
Using Two Parameter  Constructor

Initialization of b
Using One Parameter  Constructor

Printing values in A
A[0]= 1 + 1*i
A[1]= 0 + 0*i
A[2]= 3 + 0*i
A[3]= 5 + 6*i
```


Friend Functions and Overloading

Suppose

ComplexClass A,B;

We saw before how to write $A.Add(B)$ so that it returns the expected value $A + B$.

In this section we will learn how to modify the $+$ operator to actually write $A + B$ so that it works for **ComplexClass**.

This will require learning two new concepts

1. **Friend Functions:** these are *non-member* functions that are permitted to access private members.
2. **Operator Overloading:** permits adding/redefining meaning of standard C++ operators such as $+, -, >>, <<, ==, =, \ldots$

Friend Functions
(Lippman/Lajoie, 13.1.4, 15.2)

It would be very convenient if we could write a function such as

```
int Equal(ComplexClass A, ComplexClass B)
{ return ( (A.Real ==B.Real) && (A.Imag==B.Imag) );}
```

Unfortunately this is illegal because *Equal()* is not a *ComplexClass* member function and is accessing private members of *ComplexClass* objects.

We will now see a way around this difficulty, a way to give a non-member function the same access privileges as a member function.

This is done by declaring the non-member function as a **friend** to the class.

To make a non-member function a friend you must declare it as a friend in the class definition. More particularly you must include the function prototype preceded by the word **friend** in the class definition.

```
class ComplexClass
{public:
friend ComplexClass Add(ComplexClass A, ComplexClass B)
friend int Equal(ComplexClass A, ComplexClass B);

  ComplexClass(int x, int y);
  ComplexClass(int x);
  ComplexClass();
  void set(int x, int y);
  void print();

 private:
  int Real, Imag;
};
```

A friend function is *not* a member function. In particular

1. The dot operator is not used to call it.
2. A type-qualifier is not used in its definition.


```
//Complex5.cc
//Introduction to friend functions
#include <iostream.h>
class ComplexClass
{ public:
        // Return A+B
        // and return value.
        friend ComplexClass Add(ComplexClass A, ComplexClass B);

        // Return A==B
        // and return value.
        friend int Equal(ComplexClass A, ComplexClass B);

        //Basic Constructor
        //Initializes number to be x + y*i
        ComplexClass(int x, int y);

        //Basic Constructor
        //Initializes number to be x + 0*i
        ComplexClass(int x);

        //Default Constructor
        //Initializes number to be 0 + 0*i
        ComplexClass();

        // Set number to be x + y*i
        void set(int x, int y);

        // Print in the form of: real+imagine*i.
        void print();

  private:
        int  Real;
        int  Imag;
};
```



```
ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;}

ComplexClass::ComplexClass(int x)
{ Real =x; Imag =0;}

ComplexClass::ComplexClass()
{ Real =0; Imag =0;}

void ComplexClass::set(int x, int y)
{ Real =x; Imag =y;}

void ComplexClass::print()
{cout << "Number is " << Real  << " + " << Imag << "*i"<<endl;}

//***************************************************
// A new addition function
// works because it uses friend declaration
ComplexClass Add(ComplexClass A, ComplexClass B)
{ ComplexClass temp;
  temp.Real = A.Real + B.Real;
  temp.Imag = A.Imag + B.Imag;
  return temp;
    }

// A new Equal function
// works because it uses friend declaration
int Equal(ComplexClass A, ComplexClass B)
{ return ( (A.Real ==B.Real) && (A.Imag==B.Imag) );}
```



```
int main()
{
        ComplexClass a(1,1), b(2,2), c;
        cout << "a = "; a.print();
        cout << "b = "; b.print();
        cout << "c = "; c.print();
        cout << endl;

        c = Add(a,b);
        cout << "After addition c = "; c.print();
        cout << endl << endl;

        b=a;
        cout << "a = "; a.print();
        cout << "b = "; b.print();
        cout << "c = "; c.print();
        cout << endl << endl;

        cout << "a";
        if (Equal(a,b))
           cout << " = ";
        else cout << " != ";
        cout << "b" << endl;
```

```
        cout << "a";
        if (Equal(a,c))
           cout << " = ";
        else cout << " != ";
        cout << "c" << endl;
}
```



```
a = Number is 1 + 1*i
b = Number is 2 + 2*i
c = Number is 0 + 0*i

After addition c = Number is 3 + 3*i


a = Number is 1 + 1*i
b = Number is 1 + 1*i
c = Number is 3 + 3*i


a = b
a != c
```


Overloading Operators
(Lippman/Lajoie, 15)

Suppose **ComplexClass A,B;**

We now know how to write
$Add(A, B)$ and $Equal(A, B)$.

We will now see how to redefine $+$ and $==$ so that they return the appropriate values, i.e., $A + B$ and $A == B$.

Symbols such as $+$, $-$, $<<$, etc. are known as operators. Adding new utility to them in this fashion is known as **operator overloading.**

Operators can be overloaded in almost exactly the same way as functions. The only difference is that in place of the function name we write **operator op** where **op** is the symbol used by the operator. For example

CC operator+(CC A, CC B)
and
int operator==(CC A, CC B)

Almost all operators in C++ can be overloaded including $<<$ and $>>$.

Note: Properly defining overloaded operators often requires making them **friend** functions to the class in the class definition.

friend CC operator+(CC A, CC B)
and
friend int operator==(CC A, CC B)

```
//Complex6.cc
//Introduction to friend functions with overloading
#include <iostream.h>

class ComplexClass
{
  public:
        // Return A+B
        friend ComplexClass operator+(ComplexClass A, ComplexClass B);

        // Return A==B
        friend int operator==(ComplexClass A, ComplexClass B);

        //Basic Constructor
        //Initializes number to be x + y*i
        ComplexClass(int x, int y);

        //Basic Constructor
        //Initializes number to be x + 0*i
        ComplexClass(int x);

        //Default Constructor
        //Initializes number to be 0 + 0*i
        ComplexClass();

        // Set number to be x + y*i
        void set(int x, int y);

        // Print in the form of: real+imagine*i.
        void print();

  private:
        int  Real;
        int  Imag;
};
```



```
ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;}

ComplexClass::ComplexClass(int x)
{ Real =x; Imag =0;}

ComplexClass::ComplexClass()
{ Real =0; Imag =0;}

void ComplexClass::set(int x, int y)
{ Real =x; Imag =y;}

void ComplexClass::print()
{cout << "Number is " << Real  << " + " << Imag << "*i"<<endl;}


ComplexClass operator+(ComplexClass A, ComplexClass B)
{ ComplexClass temp;
  temp.Real = A.Real + B.Real;
  temp.Imag = A.Imag + B.Imag;
  return temp;
}

int operator==(ComplexClass A, ComplexClass B)
{ return ( (A.Real ==B.Real) && (A.Imag==B.Imag) );}
```



```
int main()
{
        ComplexClass a(1,1), b(2,2), c;
        cout << "a = "; a.print();
        cout << "b = "; b.print();
        cout << "c = "; c.print();
        cout << endl;

        c = a+b;
        cout << "After addition c = "; c.print();
        cout << endl << endl;

        b=a;
        cout << "a = "; a.print();
        cout << "b = "; b.print();
        cout << "c = "; c.print();
        cout << endl << endl;

        cout << "a";
        if (a==b)
           cout << " = ";
        else cout << " != ";
        cout << "b" << endl;
```



```
        cout << "a";
        if (a==c)
           cout << " = ";
        else cout << " != ";
        cout << "c" << endl;
}
```

```
a = Number is 1 + 1*i
b = Number is 2 + 2*i
c = Number is 0 + 0*i

After addition c = Number is 3 + 3*i


a = Number is 1 + 1*i
b = Number is 1 + 1*i
c = Number is 3 + 3*i


a = b
a != c
```


The basic rule is that, when overloading an operator, at least one parameter must be of a class type e.g., you can not overload + so that $5 + 3$ has a different meaning.

Note that overloading == so that $A == B$ is now defined **does not** mean that if $A = 1+0*i$ then $A == 1$ will return true (1).

This is because == was only overloaded for two ComplexClass parameters. If we want it to work for $A == 1$ we must also overload it for one ComplexClass and one int.

```
int operator==(ComplexClass A, ComplexClass B)
{ return ( (A.Real==B.Real) && (A.Imag==B.Imag) );}

int operator==(ComplexClass A, int B)
{ return ( (A.Real==B) && (A.Imag==0) );}

int operator==(int A, ComplexClass B)
{ return ( (A==B.Real) && (0==B.Imag) );}
```



```
//Complex7.cc
//Introduction to friend functions with overloading
#include <iostream.h>
class ComplexClass
{
  public:
    // Return A==B
    friend
      int operator==(ComplexClass A, ComplexClass B);
    friend int operator==(ComplexClass A, int B);
    friend int operator==(int A, ComplexClass B);

    //Basic Constructor
    //Initializes number to be x + y*i
    ComplexClass(int x, int y);

    // Print in the form of: real+imagine*i.
    void print();


  private:
    int  Real;
    int  Imag;
};
```



```
ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;}

void ComplexClass::print()
{cout << "Number is " << Real
      << " + " << Imag << "*i"<<endl;}

int operator==(ComplexClass A, ComplexClass B)
{ return ( (A.Real==B.Real) && (A.Imag==B.Imag) );}

int operator==(ComplexClass A, int B)
{ return ( (A.Real==B) && (A.Imag==0) );}

int operator==(int A, ComplexClass B)
{ return ( (A==B.Real) && (0==B.Imag) );}
```

```
int main()
{       ComplexClass a(1,1), b(1,0);

        cout << "a";
        if (a==1)  cout << " = ";
           else cout << " != ";
        cout << "1" << endl << endl;

        cout << "b";
        if (b==1)  cout << " = ";
           else cout << " != ";
        cout << "1" << endl << endl;

        cout << "1";
        if (1==b)  cout << " = ";
           else cout << " != ";
        cout << "b" << endl << endl;

        cout << "b";
        if (b==2)  cout << " = ";
           else cout << " != ";
        cout << "2" << endl << endl;
}
```



```
a != 1

b = 1

1 = b

b != 2
```