Overloading and Constructors

Our next major topic will be how to initialize new objects using **Constructors**. Before doing so we take a sort break to introduce another concept that we will need in that discussion, that of function **overloading**. This is a technique that allows the same function name to have many "meanings".

You actually use this all of the time. For example, 1/2 is not the same thing as 1.0/2.0 in C++.

As another example suppose you want to write one function to compute the average of two numbers and another to compute the average of three numbers:

```
double ave(double n1, double n2)
{ return ((n1 + n2)/2.0); }

double ave3(double n1, double n2, double n3)
{ return ((n1 + n2 + n3)/3.0); }
```

In C++ You can use the same name for both functions!

```
//Illustrates overloading the function name ave.
//Savitch 3.14
#include <iostream.h>

double ave(double n1, double n2);
//Returns the average of n1 and n2.

double ave(double n1, double n2, double n3);
//Returns the average of n1, n2, and n3.

int main( )
{    cout << "The average of 2.0, 2.5, and 3.0 is "
         << ave(2.0, 2.5, 3.0) << endl;
     cout << "The average of 4.5 and 5.5 is "
         << ave(4.5, 5.5) << endl;
    return 0; }

double ave(double n1, double n2)
{ return ((n1 + n2)/2.0); }
double ave(double n1, double n2, double n3)
{ return ((n1 + n2 + n3)/3.0); }
-------------------------------------------
The average of 2.0, 2.5, and 3.0 is 2.5
The average of 4.5 and 5.5 is 5
```

```
//illustrates overloading with different types
//From Deitel & Deitel, Figure 3.25

#include <iostream.h>

int     square(int x)    { return x * x;}
double  square(double y) { return y * y;}

void main()
{
  cout << "The square of 7 is ";
  cout << square(7) << endl;
  cout << "The square of 7.5 is ";
  cout << square(7.5) << endl;
}

-------------------------------------------
172> a.out
The square of 7 is 49
The square of 7.5 is 56.25
```

If you have two or more function definitions for the same function name that is called **overloading**

When you overload a function name the different definitions must have different numbers of formal parameters or some formal parameters of different types.

The compiler checks each function call and matches it with the particular function definition whose number and type of formal parameters matches.

The use of the same name to mean different things is called **Polymorphism** (Greek for "many forms").

Overloading in Classes

```
//Complex.cc
//Basic example of how to use classes
//Overloaded set function
#include <iostream.h>

class ComplexClass
{
  public:
         // Set number to be x + y*i
         void set(int x, int y);

         // set number to be z +0*i
         void set(int z);

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

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

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

void ComplexClass::set(int z)
{ Real =z; Imag =0;}

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

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

        b.set(2);
        cout << "b = "; b.print();
        cout << endl;
}
-----------------------------------------------
a = Number is 1 + 1*i

b = Number is 2 + 0*i
```

Constructors

A variable of predefined data type such as **int** can be initialized when it is declared, e.g.,

int i=5;

The same sort of thing can be done for an object of a (user) defined class by using a **constructor**.

A **constructor** is a special *member function* of the class that is automatically called whenever an object of that class is declared.

- A **constructor** must have the same name as the class
- A **constructor** CAN NOT return a value.
  No return type, not even void, can be given in the function prototype.

```
//Constructor Prototype (in class)
 ComplexClass(int x, int y);

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

```
//Constructor1.cc
//Basic Constructor Example
#include <iostream.h>

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

         // 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.
         ComplexClass Add(const ComplexClass & C);


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

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

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

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

int main()
{       ComplexClass a(2,5), b(9,4);
        cout << "a = "; a.print();
        cout << "b = "; b.print();
        cout << endl;

        a.set(1,1);
        cout << "a = "; a.print();}
*******************************************
a = Number is 2 + 5*i
b = Number is 9 + 4*i

a = Number is 1 + 1*i
```

A constructor can be *overloaded* in the same way as any other function.

```
//In Class definition
//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);

//Constructor Definitions
ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;}

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

```
//Constructor2.cc
//Overloaded Constructor Example
#include <iostream.h>
class ComplexClass
{ public:
         //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);

         void set(int x, int y);

         void print();

         ComplexClass Add(const ComplexClass & C);

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

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

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

void ComplexClass::set(int x, int y)
{ Real =x; Imag =y;}
void ComplexClass::print()
{cout << "Number is " << Real
  << " + " << Imag << "*i"<<endl;}
int main()
{       ComplexClass a(2), b(9,4);
        cout << "a = "; a.print();
        cout << "b = "; b.print();
        cout << endl;

        a.set(1,1);
        cout << "a = "; a.print();}
*******************************************
a = Number is 2 + 0*i
b = Number is 9 + 4*i

a = Number is 1 + 1*i
```

If no constructor is defined C++ defines a **default constructor** that is used, e.g., the declaration

ComplexClass a,b

actually calls the default constructor.
BUT if the user defines at least one constructor for a class then the default constructor is not automatically defined. Furthermore C++ will attempt to match every declaration to some constructor definition. This can lead to errors.

Example: If

ComplexClass(int x, int y);

is the only constructor defined then all declarations must use that form, e.g.,

ComplexClass a(2,5), b(9,4);

but declarations of form

ComplexClass a,b;

will cause an error.

```
//Constructor3.cc
//No Default Constructor
//THIS PROGRAM DOES NOT WORK

#include <iostream.h>
class ComplexClass
{ public:
         //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);

         void set(int x, int y);

         void print();

         ComplexClass Add(ComplexClass C);

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

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

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

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

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

int main()
{       ComplexClass a;  //THIS IS ILLEGAL

        a.set(1,1);
        cout << "a = "; a.print();
}
```

Because of this problem you should always include a default constructor in your class definition.

```
class ComplexClass
{ public: //3 construtors
         ComplexClass(int x, int y);
         ComplexClass(int x);
         ComplexClass();
             .
             .
             .
}


//Function definitions
ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;}

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

ComplexClass::ComplexClass()
{// Do Nothing
}
```

```
//Constructor4.cc
//Including Default Constructor
#include <iostream.h>
class ComplexClass
{ public:
         //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 Consructor
         ComplexClass();

         void set(int x, int y);
         void print();
         ComplexClass Add(const ComplexClass &C);

         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()
{// Do Nothing
}

void ComplexClass::set(int x, int y)
{ Real =x; Imag =y;}
void ComplexClass::print()
{cout << "Number is " << Real
   << " + " << Imag << "*i"<<endl;}

int main()
{       ComplexClass a;

        a.set(1,1);
        cout << "a = "; a.print();}
*******************************************
a = Number is 1 + 1*i
```

Weird C++ Syntax

The default constructor is a function with no parameters so you might think that it should actually be called using

ComplexClass a();

the same way as any other function without parameters.

This in not correct. A default constructor should be called as

ComplexClass a;

without using the ().

Constructors are called *anywhere* an object is declared, even when the object is only declared temporarily, e.g., as a local variable.

```
//Constructor5.cc
//Including Default Constructor
//
#include <iostream.h>
class ComplexClass
{  public:
         //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 Consructor
         ComplexClass();

         void set(int x, int y);
         void print();
         ComplexClass Add(const ComplexClass &C);

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

```
ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;
  cout<< "Constructing with two variables: ";
  cout <<Real  << " + " << Imag << "*i"<<endl << endl;;
}

ComplexClass::ComplexClass(int x)
{ Real =x; Imag =0;
  cout<< "Constructing with one variable: ";
  cout <<Real  << " + "
       << Imag << "*i"<<endl << endl;;
}

ComplexClass::ComplexClass()
{ Real =0; Imag =0;
  cout<< "Constructing with no variables: ";
  cout <<Real  << " + "
    << Imag << "*i"<<endl << endl;
}

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

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


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

```
int main()
{       ComplexClass a(6,9),b(1),c(2);

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

        c = a.Add(a);
        cout << "a = "; c.print();
}

***********************************************
Constructing with two variables: 6 + 9*i

Constructing with one variable: 1 + 0*i

Constructing with one variable: 2 + 0*i

Constructing with no variables: 0 + 0*i

c = Number is 7 + 9*i

Constructing with no variables: 0 + 0*i

a = Number is 12 + 18*i
```