Introduction to Classes

- The concept of a **class** is the same as that of a structure: it is a user-defined data type that is used to create a group of variables that may have different data types.
- In addition to the data members, a **member of a class** can also be a function.
- **A member function** is in general specified in the class definition by its function prototype with its definition provided somewhere else in the program.
- **You define a class** as follows:

```
class   <class-name>
{
    <Data-members-and/or-member functions-declarations>
};
```

- A member of a class (data or function) can be **public** or **private**.
- In the definition of a class,
  - you use the label **public** to specify its **public members** and
  - the label **private** to specify its **private members**.

Example C1

```
class  Demo1
{
    public:
        double  getValue2(void);        // to return the value of the private data member
        void  setValue2(double num);    // to set the value of the private data member
        double  getAverage( );          // to compute the average of both values
        double val1;                    // the public data member
    private:
        double computeSum( );           // to compute the sum of both values
        double val2;                    // the private data member
};
```

Definition of a Class Member Function

- **When you write the definition of a class member function**, you must precede the name of the function in the function header by the name of that class followed by the **scope resolution operator** (**::**).
- A **class data member** (public or private) can be accessed in the body of a member function of that class.
- A **class member function** (private or public) can be called in another member function of that class.

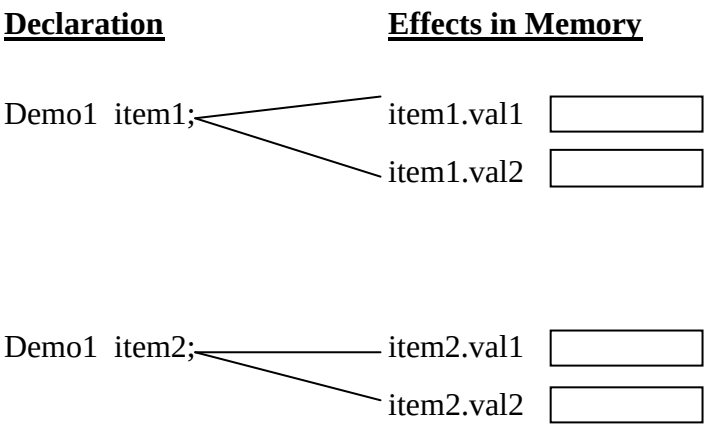
Example C2 Definitions of the member functions of class Demo1:

```
/*---------------------------------- function getValue2( ) ----------------------------------------------*/
/*  returns the value of the private member variable                                                    */
double  Demo1:: getValue2(void)
{
    return (val2);
}

/*---------------------------------- function setValue2( ) ----------------------------------------------*/
/*  set the value of the private member variable to the given value                                     */
    void  Demo1:: setValue2(double num)
    {
        val2 = num;
    }

/*---------------------------------- function getAverage( ) ----------------------------------------------*/
/*  compute the average of both values and return it                                                    */
    double  Demo1::  getAverage(void)
    {
        double total;
        total = computeSum( );
        return(total / 2);
    }

/*---------------------------------- function computeSum( ) ----------------------------------------------*/
/*  compute the sum of both values and return it                                                        */
    double Demo1:: computeSum(void)
    {
        return (val1 + val2 ) ;
    }
```

Declaration of a Class Variable (or Object)

- A class variable is called **object**.
- You **declare an object** in the same way that you declare a structure variable.
- As with structure variables, when you declare an object, a **member variable** is created for each data member of the class.

Example C3



Specifying the Members of an Object

- You **specify an object's public member variable** in the same way that you specify a member of a structure variable, by using the **dot** operator.
- You **call a class public member function on an object** by preceding the function call with the name of that object, followed by the **dot** operator.
- A **private member variable of an object** can only be specified in a member function or a *friend function* of the class of that object.
- A **private member function of a class** can be called on an object only in a member function or a *friend function* of the class of that object.
- A **class member function** can be called in a function that is not a class member function only on an object.

Example C4

Given the class *Demo1* of example C1 and object *item1* defined in function *main* as follows:

Demo1 item1;

We have the following access to the member functions and the member variable of object *item1*:

Valid Access to the Members of Object item1	**Effects**
1. cin >> item1.val1;	// read a value into the public member variable *val1*
2. item1.setValue2 (15);	// set the private member variable *val2* to 15
3. cout << endl << "The average of:\t"	
4. << item1.val1 << " and "	// output the value of the member variable *val1*
5. << item1.getValue2( ) << " is: "	// output the value of the member variable *val2*
6. << item1.getAverage( );	// output the average of the values of both variables

Invalid Access to the Members of Object item1	**Reasons**
1. cin >> item1.val2;	// *val2* is a private member variable
2. item1.computeSum( );	// *computeSum* is a private member function
3. getValue2( );	/* in a function that is not a class member function, a class member function must be called on an object */

Objects and the Assignment Operator

- It is legal in C++ to assign an object to another object of the same class.
- The assignment of an object to another generates the member-wise assignments of the member variables of both objects.

Example C5

Given the class *Demo1* of Example C1 and the following definitions of objects: *Demo1 item1, item2;*

The assignment: *item1 = item2;*

has the same effect as:

```
item1.val1  = item2.val1;
item1.setValue2( item2.getValue2( ) );
```

Notes

1. The concept of combining a number of items such as variables and functions into a package such as an object is called **encapsulation**.
2. In a class definition, you can have any number of occurrences of the labels **public:** and **private:** as shown in the following example:

```
class   SampleClass
    {
        Public:
            void  inputSomething( );
            int   stuff;
        private:
            int   computeResult( int val);
            int  val1;
        public:
            int  setvalue( );
            int val2;
     };
```

3. In a class definition, the section between the opening brace { and the first **access-specifier** label, **public:** or **private:** is a private section. For example, the following two class definitions are identical:

```
class   SampleClass1
{
        int   computeResult( int val);
        int  val1;
    public:
        int  setvalue( );
        int val2;
};
```

```
class   SampleClass2
{
    private:
        int   computeResult( int val);
        int  val1;
    public:
        int  setvalue( );
        int val2;
};
```

4. It is a common practice to specify the public members of a class before the private members and to list the member function prototypes before the member variables as we have done with the definition of the class *Demo1* of Example C1.

Exercise C1*

Given the following class definition:

```
class  Automobile
{
    public:
        void   setPrice ( double  newPrice );          // to set the value of price member variable
        void  setProfit ( double newProfit );          // to set the value of profit member variable
        double  getPrice (  );                         // to return the value of price member variable
    private:
        double  price;
        double  profit;
        double  getProfit (   );                       // to return the value of profit member variable
};
```

a. Write the definitions of the member functions *setPrice*, *setProfit*, *getPrice*, and *getProfit*.
b. Given the following definitions of objects *hyundai* and *jaguar* in function *main*:

    ```
    Automobile   hyunday,  jaguar;
    ```

    Which of the following statements are invalid? Also provide the reasons why a statement is invalid.

    ```
    hyunday.price  =  15000.00;
    jaguar.setPrice( 45000.00 );
    cout   <<   jaguar.getProfit (   );
    jaguar.setProfit(    );
    cout   <<  jaguar.getPrice (   );
    jaguar  =  hyunday;
    setPrice ( 16500 );
    ```

c. Write a statement to set the *profit* member variable of object *jaguar* to 3500.
d. Write a statement to read a price and set the *price* member variable of object *hyunday* to it.
e. Write a statement to output the value of the *price* member variable of object *jaguar*.
f. Is it possible to write a statement to output the value of the *profit* member variable of objet *jaguar*? Why?
g. Write the statements to do the following:
    1. Assign the value of each member variable of object *jaguar* to the corresponding member variable of object *hyunday*, and then,
    2. Change the value of the member variable *price* of object *hyunday* to 24000.00.

Exercise C2

Given the following class *Automobile* defined in exercise C1:

```
class  Automobile
{
    public:
        void   setPrice ( double  newPrice );          // to set the value of price member variable
        void  setProfit ( double newProfit );          // to set the value of profit member variable
        double  getPrice (  );                         // to return the value of price member variable
    private:
        double  price;
        double  profit;
        double  getProfit (   );                       // to return the value of profit member variable
};
```

a. And given the following definitions of objects *sable* and *jeep* in function *main*:

   ```
   Automobile   sable,  jeep;
   ```

   Which of the following statements are invalid? Also provide the reasons why a statement is invalid.

   ```
   cin   >>   jeep.profit;
   jeep.setPrice( 45000.00 );
   cout   <<   jeep.getProfit (   );
   cout   <<   jeep.getPrice (   );
   sable.setProfit(   );
   jeep  =  sable;
   cout   <<   getPrice(   );
   ```

b. Write a statement to set the *price* member variable of object *jeep* to 22000.00.
c. Write a statement to read a profit and set the *profit* member variable of object *sable* to it.
d. Write a statement to output the value of the *price* member variable of object *jeep*.
e. Write the statements to assign the value of each member variable of object *jeep* to the corresponding member variable of object *sable*, and then to change the value of member variable *profit* of object *sable* to 4000.00.

Exercise C3

Write the definition of function *main* that does the following:

a. Define an object of the class *Demo1* (defined in example C1above)
b. Read two values and set the two member variables of this object to those values.
c. Output the values of the member variables of this object with appropriate messages.
d. Compute and print the average value of the member variables of this object with an appropriate message.

Classes and Functions

- **A class can be the return type of a function**: a function can return an object.

Example C6

The following function reads the values for the two member variables of an object of the type class *Demo1* (defined in example C1 above) and then returns that object.

```
 Demo1  inputValues (void)
{
    Demo1 temp;
    double num;
    cin   >>   temp.val1;            // read a value for the first member variable of the object

    /*--- read the second value and set the second member variable of the object to it--------*/
    cin   >>  num;
    temp.setValue2(num);
    return (temp);
}

/*---------------------------------- the above function is called as follows ----------------------------*/
Demo1  item;
item = inputValues(  );
```

- **An object can be a value parameter of a function**.
- When a function with an object as value parameter is called, the value of each member variable of the object argument is copied to the corresponding member variable of the object parameter.

Example C7

The following function receives an object of the class *Demo1* (defined in example C1 above) as argument, and returns five times the average of the values of its member variables.

```
 double average5 (Demo1 item )
{
    double result;
    result =  5 * item.getAverage( );
    return result;
}
```

```
/*----------------------- the above function may be called as follows ---------------------------*/
Demo1 stuff;
stuff.val1  =  10;
stuff.setValue2 ( 12 );
cout  <<  endl  <<  average5( stuff );
```

After this call of function *average5( )*, its body is executed as if it was written as follows:

```
double average5 (Demo1 item )
{
    item.val1  =  stuff.val1;
    item.setValue2( stuff.getValue2( ) );
    double result;
    result =  5 * item.getAverage( );
    return result;
}
```

- **An object can be a reference parameter of a function**.
- When a function with an object as a reference parameter is called, the object argument replaces the object parameter in the body of the function.

Example C8

The following function receives an object of the class *Demo1* (defined in example C1 above) as argument, and adds 3 to the value of each of its member variables.

```
void  add3 (Demo1 &item)
{
    int num;
    item.val1 += 3;
    num  =  item.getValue2( );
    item.setValue2( num + 3 );
}
```

```
/*---------------------------------- the above function is called as follows ---------------------------*/
Demo1 someItem;
someItem.val1  =  10;
someItem.setValue2 ( 15 );
add3( someItem );              //  someItem.value1 = 13  and someItem.value2 = 18
```

After the above call of function *add3( )*, its body is executed as if it was written as follows:

```
{
    int num;
    someItem.val1 += 3;
    num = someItem.getValue2( );
    someItem.setValue2( num + 3 );
}
```

Exercise C4*

Using the definition of the class *Demo1* given in example C1, do the following:

a. Write a function *addDemo1* that receives as arguments two objects of the class *Demo1* and then builds and returns another object of the class *Demo1* such that the value of each of its member variable of is the sum of the values of the corresponding member variables of the objects received as arguments.
b. Write a code segment to do the following:
   i. Declare objects *obj1* and *obj2* of the class *Demo1*.
   ii. Set the values of the member variables of object *obj1* to 5 and 7 respectively and those of the member variables of object *obj2* to 10 and 15 respectively.
   iii. Create object *objR* of the class *Demo1* such that the value of each of its member variables is the sum of the values of the corresponding member variables of the objects *obj1* and *obj2* by calling function *addDemo1( )* defined in part a.

Exercise C5*

Using the definition of the class *Demo1* given in example C1, do the following:

a. Write a *void* function *updateDemo1* that receives as argument an object of the class *Demo1* and adds 10 to the value of its first member variable, and subtract 5 from the value of its second member variable.
b. Write a code segment to do the following:
   — Declare object *obj* of the class *Demo1*.
   — Set the values of the member variables of object *obj* to 15 and 70 respectively.
   — Add 10 to the value of the first member variable of object *obj* and subtract 5 from the value of its second member variable by calling the function *updaDemo1( )* defined in part a.

Exercise C6

The class *Demo2* is defined as follows:

```
class  Demo2
{
    public:
        void  setValues(double num1, double num2);  // to set the values of both member variables
        double  getValue1(void);                    // to return the value of member variable val1
        double  getValue2(void);                    // to return the value of member variable val2
        double  getAverage( );            // to compute the average of  the values of both variables
    private:
        double val1;                      // the first data member
        double val2;                      // the second data member
};

    /*---------------------------------- function setValues( ) ----------------------------------------------*/
    /*  set the values of the member variables to the given values                                          */
    void  Demo2:: setValues(double num1, double num2)
    {
        val1 = num1;
        val2 = num2;
    }

    /*---------------------------------- function getValue1( ) ----------------------------------------------*/
    /*  returns the value of the first member variable                                                      */
    double  Demo2:: getValue1(void)
    {
        return (val1);
    }

    /*---------------------------------- function getValue2( ) ----------------------------------------------*/
    /*  returns the value of the second member variable                                                     */
    double  Demo2:: getValue2(void)
    {
        return (val2);
    }

    /*---------------------------------- function getAverage( ) ---------------------------------------------*/
    /*  compute the average of both values and return it                                                    */
    double  Demo2::  getAverage(void)
    {
      return(  (valu1 + valu2) / 2);
    }
```

Use the definition of the class *Demo2* to do the following:

1. Declare object *item* of the class *Demo2*.
2. Read the values for the member variables of object *item*, and set the values of its member variables.
3. Compute the average of the values of the member variables of object *item* and print it.
4. Write a function named *addDemo2( )* that receives as arguments two objects of the class *Demo2* and then builds and returns another object of the class *Demo2* such that the value of each of its member variables is the sum of the values of the corresponding member variables of its arguments.
5. Write the sequence of statements to do the following:
   i. Declare object *obj1* and set its member variables to 5 and 7 respectively
   ii. Declare object *obj2* and set its member variables to 14 and 9 respectively.
   iii. Create a third object named *objR* such that the value of each of its member variables is the sum of the values of the corresponding member variables of objects *obj1* and *obj2* by calling function *addDemo2( )*.
6. Write a function *incrDemo2* that receives as argument an object of the class *Demo2* and increments the value of each of its member variables by 5.
7. Write the statement(s) to increment the value of each member variable of object *obj1* by 5 by calling function *incrDemo2( )*.

Object-Oriented Programming

- Object-Oriented programming is a programming methodology with the following three fundamental characteristics:
  - Data abstraction and information hiding
  - Inheritance
  - Dynamic binding of the messages (function calls) to the methods (definitions of functions)

Data abstraction and information hiding

- The data is encapsulated with its processing methods (functions) with a controlled access to the data.

Inheritance

- The definition of a new class (called **derived class**) can be derived from that of an existing class (called **base class**).
- Inheritance provides an effective way to reuse programming code.

Example: the class to represent graduate students can be derived from the class to represent students.

Dynamic (run-time) Binding of the Messages to the Methods

- Messages (function calls) are dynamically bound to specific method (function) definitions: it means that they are bound during the execution of the program.

Data Abstraction and Information Hiding in C++

- **Data abstraction and information hiding is implemented in C++ by using classes** as follows:
  1. Make all data members of a class private.
  2. For each data member, provide a public member function called **set function** or **mutator** that a user of the class must call to modify or set its value.
  3. For each data member, provide a public member function called **get function** or **accessor** that a user of the class must call to access or obtain its value.

Example O1

```
class Demo3
{
    public:
        void readValues(void);          // to read the values of the member variables
        void  setValue1(double num);    // to set the value of the first member variable
        void  setValue2(double num);    // to set the value of the second member variable
        double  getValue1(void);        // to return the value of the first member variable
        double  getValue2(void);        // to return the value of the second member variable
        double  getAverage( );          // to compute the average of both values
    private:
        double computeSum( );           // to compute the sum of both values
        double val1;                    // the first member variable
        double val2;                    // the second member variable
};


/*---------------------------------- function readValues( ) ----------------------------------------------*/
/*  read the values of the member variables ----------------------------------------------------------*/
void  Demo3:: readValues(void)
{
    cin  >>  val1  >>  val2;
}


/*---------------------------------- function getValue1( ) ----------------------------------------------*/
/*  return the value of the first member variable  ---------------------------------------------------*/
double  Demo3:: getValue1(void)
{
    return (val1);
}


/*---------------------------------- function setValue1( ) ----------------------------------------------*/
/*  set the value of the first member variable to the given value ------------------------------------*/
void  Demo3:: setValue1(double num)
{
    val1 = num;
}
```

```
/*---------------------------------- function getValue2( ) ----------------------------------------------*/
/*  return the value of the second member variable ---------------------------------------------------*/
double  Demo3:: getValue2(void)
{
      return (val2);
}


/*---------------------------------- function setValue2( ) ----------------------------------------------*/
/*  set the value of the second member variable to the given value                                  */
void  Demo3:: setValue2(double num)
{
    val2 = num;
}


/*---------------------------------- function getAverage( ) ---------------------------------------------*/
/*  compute the average of both values and return it -------------------------------------------------*/
double  Demo3::  getAverage(void)
{
    double total;
    total = computeSum( );
    return(total / 2);
}


/*---------------------------------- function computeSum( ) ---------------------------------------------*/
/*  compute the sum of both values and return it -----------------------------------------------------*/
double Demo3:: computeSum(void)
{
    return (val1 + val2 ) ;
}


/*------  The following code segment defines an object of the class Demo3,
    reads values for its member variables, and computes and prints their average  ----------------*/

Demo3  item1;
item1.readValues( );                              // read the values for the member variables
cout  <<  endl   <<   "The avareage of:\t"
        <<   item1.getValue1( )   <<   " and   "    //  output the first value
        <<   item1.getValue2( )   <<   " is:  "     //  output the second value
        <<   item1.getAverage( );                   //  output their average
```

```
/*------  The following code segment defines an object of the class Demo3,
    sets the values for its members variables to 57 and 86, and computes and prints their average--*/

Demo3  item2;
item2.setValue1(57 );                    // set the value of the first member variable to 57
item2.setValue2(86 );                    // set the value of the second member variable to 86
cout  <<  endl   <<   "The avareage of:\t"
        <<   item2.getValue1( )   <<   " and   "    // output the first value
        <<   item2.getValue2( )   <<   " is:  "     // output the second value
        <<   item2.getAverage( );                   // output their average
```

Notes

1. You do not have to define a set function for each member variable: this is usually the case when the member variable is initialized or its value is input and will not be modified.
2. A set function may be used to set the values of two or more member variables. For example, the class **Demo3** of example O1 could only have one set function **setValues (double num1, double num2)** that sets the values of both member variables of an object instead of the two set functions **setVal1(double num)** and **setVal2(double num)**.
3. You do not have to define a get function for each member variable: this is usually the case when the value of the member variable is not needed by the user of the class.
4. The **essence of object-oriented programming** is to solve a problem as follows:
   a. First identify the real world objects of that problem and the processing required of each object.
   b. Then create those objects simulations and processes, and the required communications between them.

Initializing Object with constructors

- A **constructor** is a special member function of a class that is used to initialize the member variables of the objects of that class.
- A **constructor** of a class has the following characteristics:
  1. Its name is the same as the name of the class.
  2. It has no return type (not even **void**).
  3. It may or may not have parameters.
- A class may have more than one constructor, and
- A **constructor of a class** is automatically called when an object of that class is created.

Default Constructor

- The **default constructor of a class** is the constructor with no parameters.
- The **default constructor of a class** is called when an object of that class is defined without initial values.
- When you define a class without any constructor, the compiler creates the default constructor for that class.
- The default constructor created by the compiler does nothing.
- If you define a class with at least one constructor, you must also define the default constructor for that class if you want to define objects of that class without initial value(s): the compiler does not create a default constructor for a class that has another constructor.

Example O2

The following class named *Demo4* is the class *Demo3* of example O1 to which we have added the default constructor.

```
class  Demo4
{
    public:
        Demo4(  );                  // default constructor
        void readValues(void);      // to read the values of the member variables
        void  setValue1(double num);// to set the value of the first member variable
        void  setValue2(double num);// to set the value of the second member variable
        double  getValue1(void);    // to return the value of the first member variable
        double  getValue2(void);    // to return the value of the second member variable
        double  getAverage( );      // to compute the average of both values
    private:
        double computeSum( );       // to compute the sum of both values
        double val1;                // the first member variable
        double val2;                // the second member variable
};


/*-------------------------Definition of the default constructor-----------------------------------------*/
Demo4 :: Demo4(  )
{
    val1 = 0;
    val2 = 0;
}
```

```
/*------------------------------ Examples of object definitions-----------------------------------------*/
Demo4  item,                  // item.val1 is set to 0 and item.val2 is set to 0
       stuff;                 // stuff.val1 is set to 0 and stuff.val2 is set to 0
```

Constructors with Parameters

- **The parameter list of a class constructor** specifies how the initial values for the member variables of an object of that class must be specified when that object is declared.
- **When you declare an object**, you specify the initial values of its member variables in parentheses following the object's name.

Example O3

The following class named *Demo5* is the class *Demo4* of example O2 to which we have added a constructor with parameters.

```
class  Demo5
{
    public:
        Demo5(  );                          // default constructor
        Demo5( double num1, double num2);   // constructor with parameters
        void readValues(void);              // to read the values of the member variables
        void  setValue1(double num);        // to set the value of the first member variable
        void  setValue2(double num);        // to set the value of the second member variable
        double  getValue1(void);            // to return the value of the first member variable
        double  getValue2(void);            // to return the value of the second member variable
        double  getAverage( );              // to compute the average of both values
    private:
        double computeSum( );               // to compute the sum of both values
        double val1;                        // the first member variable
        double val2;                        // the second member variable
};


/*------------------------------Definition of the constructor-----------------------------------------*/
Demo5 :: Demo5( double num1, double num2  )
{
    val1 = num1;
    val2 = num2;
}
```

```
/*----------------------------- Examples of object definitions-----------------------------------------*/
Demo5  item1,              // item1.val1 is set to 0 and item1.val2 is set to 0
       stuff( 10, 15),     // stuff.val1 is set to 10 and stuff.val2 is set to 15
       obj( -1, 5 ),       // obj.val1 is set to -1 and obj.val2 is set to 5
       tobj;               // tobj.val1 is set to 0 and tobj.val2 is set to 0
```

Note The following declaration:

```
Demo5   someStuff(   );
```

is the declaration of the function *someStuff( )* without parameters, that returns an object of the class *Demo5*: it is not the definition of the object *someStuff* with a call to the default constructor.

Exercise O1*

1. A class named *OurClass* has two private data members named *first* (integer), and *second* (character). Write the definition of the class *OurClass* with the following member functions:
   - The default constructor: it sets the first data member to 0, and the second to 'A'.
   - The constructor with parameters.
   - The set functions: *setFirst( )* for data member *first*, and *setSecond( )* for data member *second*.
   - The get functions: *getFirst( )* for data member *first*, and *getSecond( )* for data member *second*.
   - The function *read( )* that reads the values for the two data members.
   - The function *print( )* that outputs the values of the two data members.
2. Write the following two functions:
   - Function *maxObj( )* that receives as arguments two objects of the class *OurClass* and returns another object of the class *OurClass* such that the value of each of its member variables is the maximum value of the corresponding member variables of its arguments.
   - Function *incrObj( )* that receives as argument an object of the class *OurClass* and add 1 to the value of its first member variable.
3. Write the declaration of the following objects of the class *OurClass*:
   - Objects *objA* and *objB* with their member variables set to the default values.
   - Object *objC* with its member variables set to 100 and 'P' respectively.
4. Write the statements to do the following:
   a. Read the values for the two member variables of object *objA*.
   b. Set the value of the first member variable of object *objB* to 15.
   c. Add 10 to the value of the first member variable of object *objA*.
   d. Output the values of the member variables of object objC.
   e. Call function *maxObj( )* with the objects *objA* and *objB* as arguments.
   f. Call function *incrObj( )* with object *objC* as argument.

Exercise O2

1. A class named *Product* has two private data members named *unitPrice* (double precision value) and *quantity* (integer value). Write the definition of this class with the following member functions:
   - The default constructor: it sets the values of both data members to 0.
   - The constructor with parameters.
   - The set functions: *setUPrice( )* for the unit price, and *setQtity( )* for the quantity.
   - The get functions: *getUPrice( )* for the unit price, and *getQtity( )* for the quantity.
   - The function *read( )* that reads the values for the unit price and the quantity data members.
   - The function *getPrice( )* that computes and returns the price of the product which is its unit price times its quantity.
   - The function *print( )* that outputs the unit price, the quantity, and the price of the product.
2. Write the following two functions:
   - Function *avgProd( )* that receives as arguments two objects of the class *Product* and returns another object of the class *Product* such that the value of *unitPrice* member variable is the average of the values of the *unitPrice* member variables of the arguments and the value of the *quantity* member variable is the sum of the values of the *quantity* member variables of the arguments.
   - Function *discountProd( )* that receives as argument an object of the class *Product* and reduces the value of its *unitPrice* member variable by 10%.
3. Write the declaration of the following objects of the class *Product*:
   - Objects *itemA* and *itemB* with their member variables set to the default values.
   - Object *itemC* with its member variables set to $1.75 and 60 respectively.
4. Write the statements to do the following:
   a. Read the values for the two member variables of object *itemA*.
   b. Set the values of the member variables of object *itemB* to $2.5 and 100 respectively.
   c. Add 10 to the value of the *quantity* member variable of object *itemC*.
   d. Output the unit price, the quantity, and the price of object *itemC*.
   e. Call function *avgProd( )* with objects *itemA* and *itemB* as arguments.
   f. Call function *discountProd( )* with object *itemC* as argument.

Copy Constructor and Object Initialization with another Object

- The **copy constructor** of a class is a constructor that has a reference parameter with that class as data type.
- The copy constructor of a class copies the values of each member variable of the object that it receives as argument to the corresponding member variable of the object on which it is called.

Example O4

The function prototype of the copy constructor of the class *Demo5* defined in example O3 follows:

Demo5 (Demo5 & objectInit);

It is defined as follows:

```
/*-----------------------------Definition of the constructor-----------------------------------------*/
Demo5 :: Demo5( Demo5 & objectInit  )
{
    val1 = objectInit.val1;
    val2 = objectInit.val2;
}
```

- A class copy constructor can be used to initialize an object with another one as follows:

```
Demo5  item( 10, 15 );
Demo5 stuff( item );          // stuff.val1 = item.val1 = 10  and   stuff.val2 = item.val2 = 15
```

With the same effects as the following two statements:

```
Demo5  item( 10, 15 );
Demo5 stuff =  item;          // stuff.val1 = item.val1 = 10  and   stuff.val2 = item.val2 = 15
```

Note1

Each class that you define in C++ has a **default copy constructor** created by the compiler: you define a copy constructor for a class only if you do not want to use the one provided by the compiler.

Note 2

- The copy constructor of a class is called automatically in the following situations:
  - When a function returns an object of that class.
  - When an object of that class is passed by value as an argument to a function.

Explicit Constructor Call

- In C++, a constructor is implicitly a function that returns an object of the class in which it is a member.
- A constructor can then be called in the same way that any other function that returns a value (in this case, an object) is called.

Example O5

```
Demo5 item, stuff;
item  =  Demo5 ( );        // calling the default constructor:  item.val1 = 0, item1.val2 = 0
stuff  =  Demo5(9, 20);    //calling the constructor with parameters:  stuff.val1 = 9,  stuff.val2 = 20
```

Note 3

- Since a constructor can be explicitly called to set the values of the member variables of an object, you can define a class with constructors without providing its set functions.

Example O6

The following class *Demo6* is the class *Demo5* of example O3 without the set functions.

```
class  Demo6
{
    public:
        Demo6(  );                      // default constructor
        Demo6( int num1, int num2);     // constructor with parameters
        void readValues(void);          // to read the values of the member variables
        double  getValue1(void);        // to return the value of the first member variable
        double  getValue2(void);        // to return the value of the second member variable
        double  getAverage( );          // to compute the average of both values
    private:
        double computeSum( );           // to compute the sum of both values
        double val1;                    // the first member variable
        double val2;                    // the second member variable
};
```

- The constructors of this class will be used to set the values of the two member variables of an object. The only problem with this class is that an object's member variables cannot be set individually.

The constructors of this class are explicitly called to set the values of the member variables of an object as follows:

```
Demo6   item1,              // item1.val1 = 0     itme1.val2  = 0
        item2,              // item2.val1 = 0     itme2.val2  = 0
item1  =  Demo6(6, 4 );     // item1.val1 = 6     itme1.val2  = 4

/*------ read the values for the member variables of object item2 and set them ----------*/
int   num1, num2;
cin   >>   num1   >>   num2;
itme2   =   Demo6( num1,  num2 );
```

Exercise O3*

Given the following class definition:

```
class   MyClass
{
   public:
        MyClass (    );                          // default constructor
        MyClass ( int num, char symb );          // constructor
        void process( );                         // a member function
  private:
        int   val;
        char  type;
};
```

Which of the following declarations of objects are invalid?

a. MyClass obj1;
b. MyClass obj2 (34, 'A');
c. MyClass obj3 ( );
d. MyClass obj4;
   obj4 = MyClass ( );
e. MyClass obj5;
   obj5 = MyClass (21, 'Z');
f. MyClass obj6 (10, 'C');
   MyClass obj7 (obj6);
g. MyClass obj8 (5, 'X');
   MyClass obj9 = obj8 ;
h. MyClass obj10 = MyClass;

Exercise O4

Given the following class definition:

```
class   YourClass
{
   public:
         YourClass (    );                              // default constructor
         YourClass (  int  num1,  double num2 );        //  constructor
         void  process(  );                             //  a member function
  private:
         int   ival;
         double dval ;
};
```

Which of the following declarations of objects are invalid?

a. YourClass obj1;

b. YourClass obj2 (34, 12.75);

c. YourClass obj3 ( );

d. YourClass obj4;
   obj4 = YourClass ( );

e. YourClass obj5;
   obj5 = YourClass (25, 3.87);

f. YourClass obj6 (10, -5.20);
   YourClass obj7 (obj6);

g. YourClass obj8 (-7, -87.0);
   YourClass obj9 = obj8 ;

h. YourClass obj10 = YourClass;

Constructors with Default Arguments

- A constructor may be specified in a class definition with defaults arguments like any other function.
- Using a constructor with default arguments is an alternative way to define a default constructor.

Example O7

The following class *Demo7* is the class *Demo5* of example O3 in which the default constructor and the constructor with parameters are replaced with a constructor with default arguments.

```
class  Demo7
{
    public:
        Demo7( int num1= 0, int num2 = 0);    // constructor with default arguments
        void readValues(void);                // to read the values of the member variables
        void  setValue1(double num);          // to set the value of the first member variable
        void  setValue2(double num);          // to set the value of the second member variable
        double  getValue1(void);              // to return the value of the first member variable
        double  getValue2(void);              // to return the value of the second member variable
        double  getAverage( );                // to compute the average of both values
    private:
        double computeSum( );                 // to compute the sum of both values
        double val1;                          // the first member variable
        double val2;                          // the second member variable
};


/*------------------------------Definition of the constructor-----------------------------------------*/
Demo7 :: Demo7( int num1, int num2  )
{
    val1 = num1;
    val2 = num2;
}


/*------------------------------ Examples of object definitions-----------------------------------------*/
Demo7  item1,                 // item1.val1 is set to 0 and item1.val2 is set to 0
       stuff( 10, 15),        // stuff.val1 is set to 10 and stuff.val2 is set to 15
       obj( 5 ),              // obj.val1 is set to 5 and obj.val2 is set to 0
       tobj;                  // tobj.val1 is set to 0 and tobj.val2 is set to 0
```

Writing a Constructor with the Initialization Section

- An alternative way to write the definition of a constructor is to follow the right parenthesis of the function header with a colon and a list of some or all data members of the class with their initializing expressions in parentheses.

Example O8

The following is an alternative way to write the definitions of the constructors of the class *Demo5* of example O3.

```
/*-----------------------------Definition of the default constructor-------------------------------------*/
Demo5 :: Demo5(  ) :  val1( 0 ), val2( 0 )
{
}


/*-----------------------------Definition of the constructor with parameters----------------------------*/
Demo5 :: Demo5( int num1, int num2  ) :  val1( num1 )
{
    val2 =  num2;
}
```

Exercise O5*

1. Rewrite the definition of the class *OurClass* that you wrote in exercise O1such that the default constructor and the constructor with parameters are replaced with one constructor with default arguments.
2. Write the definition of the new constructor with the initialization section.

Exercise O6

1. Rewrite the definition of the class *Product* that you wrote in exercise O2 such that the default constructor and the constructor with parameters are replaced with one constructor with default arguments.
2. Write the definition of the new constructor with the initialization section.

Data Validation: exit( ) Library Function

- It is in general essential to make sure that the value of an object's member variable is a valid data before it is assigned to that member variable.
- This can be done in a class definition in one of the following two ways:
  1. Define a member function that does the data validation and which is called to validate a value before it is assigned to the member variable.
  2. Do the data validation in the corresponding set function. In this case, the set function is called by any member function that would like to modify the value of the member variable.
- When an invalid data is found, one of the following actions can be taken:
  1. Print an error message and then set the value of the member variable to a default value.
  2. Print an error message and then call the library function **exit( )** to terminate the execution of the program.

exit Library Function

Prototype: void exit(int status); // Use status 0 if the program has terminated normally;
// any other value indicate an error.
Header file: <cstdlib>
Name space: std
Operation: terminates the execution of the program.

Example O9

The following class *DayOfYear* has a member function named *checkDate( )* that validates a month and a day in a month. This function is called each time the value of the month and/or that of the day data members of the class are set.

```
class  DayOfYear
{
    public:
        DayOfYear(int newMonth = 1, int  newDay = 1);    // constructor with default arguments
        void  input( );                                  // to input the month and the day
        void  output( );                                 // to output the month and the day
        int getMonth( );                                 // to return the month
        int getDay( );                                   // to return the day
    private:
        void  checkDate( );                      // to validate the month and the day
        int month;                               // to hold the month (1 – 12)
        int day;                                 // to hold the day (1 – 31)
    };


/*-----------------------------------Definitions of the member functions ----------------------------------*/
#include  <iostream>
#include <cstdlib>
using  namespace  std;

/*---------------------------------------------- Constructor ----------------------------------------------------*/
/*  set the month and the day to the provided values and do the data validation                    */
DayOfYear :: DayOfYear(int newMonth,  int newDay)
{
    month  =  newMonth
    day = newDay;
    checkDate( );
}
```

```
/*-----------------------------------------function input( ) ----------------------------------------------------*/
/*   read the month and the day in this order and do the data validation                                        */
void  DayOfYear :: input( )
{
    cin  >>  month  >>  day;
    checkDate( );
}


/*-----------------------------------------function output( ) ----------------------------------------------------*/
/*   output the month and the day                                                                               */
void  DayOfYear :: output( )
{
    cout  <<  endl  <<  "month =\t"  <<  month
          <<  "\t day =\t"  <<  day;
}


/*-----------------------------------------function getMonth( ) ----------------------------------------------*/
/*   return the month                                                                                           */
void  DayOfYear :: getMonth( )
{
    return ( month);
}


/*-----------------------------------------function getDay( ) ----------------------------------------------*/
/*   return the day                                                                                           */
void  DayOfYear :: getDay( )
{
    return ( day);
}


/*-----------------------------------------function checkDate( ) ----------------------------------------------*/
/*   validate the date                                                                                          */
void  DayOfYear :: checkDate( )
{
    if ( (month < 1) ||  (month > 12) || (day < 1) || (day > 31) )
    {
        cout  <<  endl  <<  "Invalid date";
        exit ( 1 );
    }
}
```

```
/*-----------------------------------------function main( ) ----------------------------------------------------*/
/*  read a month and a day and find out if it is John Doe's birth day which is 4/25                  */

#include  <iostream>
using  namespace  std;

int  main( )
{
    DayOfYear  today,  johnBirthday;
    cout  <<  endl  <<  "Enter today\'s date:\t";
    today.input( );
    cout  <<  endl  <<  "Today\'s date is:\t";
    today.output( );
    johnBirthday  =  DayOfYear(4, 25);                    // set John Doe's birthday
    cout  <<  endl  <<  "John\'s birthday is:\t";
    johnBirthday.output(  );
    if (today.getMonth( ) == johnBirthday.getMonth( ) && today.getDay( ) == johnBirthday.getDay( ))
        cout  <<  endl  <<  "Happy Birthday John Doe";
    return 0;
}
```

Exercise O7

Define a class named **Date** with three private data members named **month** (integer), **day** (integer), and **year** (integer) as follows:

- This class has a private member function **void checkDate( )** that validates a date as follows:
  - The month must be an integer value from 1 to 12.
  - The day must be an integer value from 1 to 31.
  - The year must be an integer value from 1960 to 2011.
- Function **checkDate( )** calls the library function **exit( )** to terminate the program if any of the above conditions is not satisfied.
- This class default constructor sets the **month** data member to 1, the **day** data member to 1, and the **year** data member to 1960: The default date is 1/1/1960.
- The class constructor with parameters calls function **checkDate( )** to check the date after it has set the values for the data members **month**, **day**, and **year**.
- The class also has the following public member functions:
  - **void inputDate( )** that reads the values for the data members month, day and year, and then calls function **checkDate( )** to check the date.
  - **void outputDate( )** that prints the date in the format: **month/day/year**.
  - **int getMonth( )**, **int getDay( )**, and **int getYear( )**. **These functions** return the value of the *month* data member, the value of the *day* data member, and the value of the *year* data member respectively.
- Place the definition of the class in the header file **Date.h**, and the definitions of the functions in the source file **Date.cpp**.
- **Note:** the header files **iostream**, **iomanip**, **cstdlib**, and **Date.h** must be included in the source file **Date.cpp**.

Preprocessor Wrapper

- The C++ compiler generates an error message when there is an attempt to include a header file in a source module more than once.
- You prevent multiple inclusions of a header file in a source module by using the *#ifndef*, *#define*, and *#endif* preprocessor directives to form a **preprocessor wrapper** of that header file as follows:

```
#ifndef    <Name>
#define    <Name>

<content-of-the header-file>

#endif
```

Where **<Name>** is a valid identifier. But in general programmers use the name of the header file in upper case with the period replaced by an underscore.

Example

```
#ifndef   DAYOFYEAR_H
#define   DAYOFYEAR_H

class  DayOfYear
{
    public:
        DayOfYear(int newMonth = 1, int  newDay = 1);    // constructor with default arguments
        void  input( );                                  // to input the month and the day
        void  output( );                                 // to output the month and the day
        int getMonth( );                                 // to return the month
        int getDay( );                                   // to return the day
    private:
        void  checkDate( );                  // to validate the month and the day
        int month;                           // to hold the month (1 – 12)
        int day;                             // to hold the day (1 – 31)
        };

#endif
```

— If the symbolic constant **DAYOFYEAR_H** is not yet defined in the source module, then the text between *#ifndef* (which means "if not defined") and *#endif* is inserted into the source module; otherwise, nothing is inserted.

— Note that if this code is already inserted in the source module, it will not be inserted the second time; because **DAYOFYEAR_H** will have already been defined in the source module.

Objects as Data Members of another Class

- **A class can have an object of another class as a data member**.
- When a class has one or more objects of other classes as data members, it is a good programming practice to initialize these objects in the initialization section of the class constructors.
- When an object that is a member of a class is not initialized in the initialization section of the constructors, the default constructor of the class of that object is automatically called to initialize that object.

Example O10

As an example of a class with objects of other classes as data members, we define the following class *Employee* with the following private data members:

- first name - class string
- last name - class string
- birth day - class DayOfYear (of Example O9)
- hours - integer
- pay rate - double precision

In addition to the constructors, the class also has the member functions *input( )* that reads the values of an object's member variables and *print( )* that output the name, birth day and gross pay of an object.

```
class  Employee
{
    public:
        Employee(   );              // default constructor
        Employee(  string  fname, string lname, DayOfYear bday, int hrs, double prate );
        void  input(   );           // read the values for the data memebers
        void  print(   );           //  output the values of the data memebers
    private:
        string firstName;           // first name (object)
        string lastName;            // last name (object)
        DayOfYear  birthDay;        // birth day (object)
        int  hours;                 // number of hours of work
        double payRate;             // Employee's pay rate
};
```

Note

- Objects that are data members of a class are created in the order in which they are declared in the class definition and not in the order in which they appear in the initialization section of constructors.

Writing the Default Constructor of the Class Employee:

- When you write the default constructor of a class that has one or more objects as data members, you do not have to initialize those objects: each object is initialized by the default constructor of its class when it is not initialized in the initialization section of the constructor.
- Two implementations of the default constructor of the class *Employee* are provided as follows:

```
/*----------------------------------------default constructor ----------------------------------------*/
   /* initialize member objects firstName and lastName to the null string; the birth day to the
      default month = 1 and day = 1; and the hours and pay rate to 0
   */

   /*---------------- Implementation with initialization section -------------------------*/
   Employee  :: Employee (   ) :  hours( 0 ),  payRate( 0 )
   {
   }


   /*-----------------Implementation without initialization section -------------------*/
   Employee  :: Employee (   )
   {
        hours =  0;
        payRate  = 0;
   }
```

Writing the Constructor with Parameters of the Class Employee:

- When you write a constructor with parameter of a class that has one or more objects as data members, it is a good programming practice to initialize those objects in the initialization section of the constructor: each object is initialized by the default constructor of its class when it is not initialized in the initialization section of the constructor.
- Two implementations of the constructor with parameters of the class *Employee* are provided as follows:

```
/*----------------------Good implementation of the constructor ----------------------------------------------------*/
/* use member initializer list to initialize the object data members    */

    Employee :: Employee( string fname, string lname, DayOfYear bday, int hrs, double prate) :
            firstName( fname ),          // initialize firstName
            lastName( lname ),           // initialize lastname
            birthDay( bday ),            // initialize birthday
            hours( hrs ),                // initialize hours
            payRate( prate )             // initialize payRate
    {
    }


/*----------------------Bad implementation of the constructor ----------------------------------------------------*/
/*------------------- double initializations of the member objects ----------------------------------------------*/

Employee :: Employee( string fname, string lname, DayOfYear bday, int hrs, double prate)
{
    firstName =  fname ;         // initialize firstName for the second time
    lastName  =  lname;          // initialize lastname for the second time
    birthday  =  bday;           // initialize birthday for the second time
    hours  =  hrs;               // initialize hours
    payRate =  prate;            // initialize payRate
}
```

Definitions of the other Member Functions:

```
/*---------------------------------- Member function input( ) ----------------------------------------------------*/
/*  read the values for the member variables  */
    void  Employee  :: input(  )
    {
        cin  >>  firstName   >>   lastName;
        birthDay.input( );                      // read the values for the birth day month and day
        cin  >>  hours   >>  payRate;
    }
```

```
/*--------------------------------- Member function print( ) ----------------------------------------------*/
/*  output the name, birth day and gross pay of an object   */
void  Employee  :: print(   )
{
    cout  <<  endl  <<   "Name:\t"  <<  lastName  +  ",  "  +  firstName;
    cout  <<  endl  <<  "Birth Day:\t";
    birthDay.output( );                  // print the values for the birth day month and day
    cout  <<  endl  <<  "Gross Pay:\t"  <<  ( hours *  payRate );
}
```

This class is used in function *main* as follows:

— To read the information about a welder and to compute and print his gross pay.

— To compute and print the gross pay of a secretary with the following information:
first name: "John"; last name: "Doe"; day of birth: 3/25; hours of work: 35; pay rate: 10.75.

```
 /*------read the information about a welder and compute his gross pay and output his information-*/
Employee  welder;

/*------------------------read his information ----------------------------------------------*/
Welder.input(   );

/*------- compute his gross pay and output his information -----------------------------*/
Welder.print( );

/*-----------Define the object secretary and set his information ---------------------------*/
DayOfYear  tempday(3, 25);
Employee  secretary( "John", "Doe", tempday,  35,  10.75  );

/*------- compute his gross pay and output his information -----------------------------*/
secretary.print( );
```

Note: An alternative way to define and initialize object *secretary* follows:

```
/*-----------Define the object secretary and set his information ---------------------------*/
Employee  secretary( "John", "Doe", DayOfYear (3, 25),  35,  10.75  );
```

Exercise O8*

1. Write the definition of the class named *Student* with the following private data members: first name and last name (class string); ID number (integer) with the default value 999999; birth day (class *DayOfYear* above); and GPA (double precision value) with the default value 0. In addition to the constructors, it has the public member functions *read( )* that inputs the values of an object's member variables, and *print( )* that outputs the values of an object's member variables.

2. Write the definitions of the constructors and the member functions *read( )* and *print( ).*

3. Write the statements to do the following:

- define an object of the class *Student,*
- read the values for its member variables and
- output the values of its member variables.

4. Define an object of the class *Student* and initialize its member variables as follows:

  - First name = Mark
  - Last name = Depaul
  - ID number = 1123
  - Birth day = 3/25
  - GPA = 3.25

5. Output the values of the member variables of the above object.

Exercise O9

Using the class **Date** that you have defined in exercise O7, write the definition of the class named **Employee** with the following private data members:

- first name - class string (the default first name is the null string " ")
- last name - class string (the default last name is the null string " ")
- ID number - integer (the default ID number is 999999)
- birth day - class Date (the default birth day is the default date: 1/1/1960)
- date hired - class Date (the default date of hire is 1/1/20011)
- base pay - double precision (the default base pay is $0.00)

In addition to the constructors, the class has the following public member functions:

- *void readPInfo( )* that reads the values for the data members first name, last name, ID number, birth day, and date of hire.
- *void readPayInfo( )* that reads the value for the base pay data member.
- *void printPInfo( )* that outputs the values of the data members first name, last name, ID number, birth day, and date of hire.
- *void setBpay(double newBpay)* that sets the value of the base pay to the new value, *newBpay*.
- *double getBpay( )* that returns the value of the base pay data member.
- *double getGpay( )* that returns the value of the gross pay (which is the base pay).
- *double computeTax( )* that computes the tax deduction on the gross pay and returns it as follows:

If gross pay is greater than or equal to 1000, 20% of the gross pay;
If 800 <= gross pay < 1000, 18% of gross pay
If 600 <= gross pay < 800, 15% of gross pay
Otherwise, 10 % of the gross pay.

- *void printPayInfo( )* that outputs the gross pay, tax deduction, and net pay (gross pay - tax deduction).
- Place the definition of this class in the header file **Employee.h**, and the definitions of the member functions in the source file **Employee.cpp**.
- **Note**:
  - the header files **string** and **Date.h** must be included in the header file **Employee.h** and the header file **Employee.h** must be included in the source file **Employee.cpp**.
  - The header file **Employee.h** must look like the following:

```
#ifndef  EMPLOYEE_H
#define  EMPLOYEE_H

#include  <string>
using namespace  std;

#include  "Date.h"

        Enter the class here

#endif
```

Inheritance

- **Inheritance** is the process by which one class called **derived class** is created from an existing class called **base class**.
- The base class is sometime called **parent class** or **superclass** whereas the derived class is sometime called **child class** or **subclass**.
- The definition of a derived class begins like any other class definition, but the name of the class is followed by a colon, then an *access-specifier* (**public/protected/private**) and the name of the base class.

Example O11

Definition of the Base class:

```
class Parent
{
    public:
      Parent( );                  // default constructor
      Parent( int n );            // constructor
      void  setValue1( int n );
      int getValue1( );
      void print(  );             // print the value of the data member
    private:
      int num1;
};
```

```
/*----------------------------- Default constructor ---------------------------------------*/
Parent ::  Parent(  ) : num1( 0 )
{
}
```

```
/*----------------------------- Constructor with parameters -----------------------------*/
Parent ::  Parent( int n ) : num1( n )
{
}
```

```
/*-----------------------------------set function-----------------------------------------*/
void  Parent ::  setValue1( int n )
{
    num1 = n;
}


/*-------------------------------- get function ----------------------------------------------*/
int  Parent ::  getValue1( void )
{
    return ( num1 );
}


/*-----------------------------------------print function ----------------------------------*/
void  Parent ::  print( )
{
    cout   <<   endl   <<   "num1 ="    <<    num1;
}
```

Definition of the Derived Class:

```
class Child   :   public   Parent        // inherits class Parent
{
    public:
        Child( );                        // default constructor
        Child ( int n1, int n2 );        // constructor
        void  setValue2( int n );
        int  getValue2( );
        void  print( );                  // print values of both data members
    private:
        int num2;
};
```

- Class *Child* is viewed by the system as if it was defined as follows:

```
class Child
{
    public:
        Child(  );                          // default constructor
        Child ( int n1, int n2 );           // constructor
        void  setValue1( int n );
        int getValue1( );
        void  setValue2( int n );
        int  getValue2(  );
        void  print(  );                    // print values of both data members
    private:
        int num1;
        int num2;
};
```

However, inherited private data member *num1* is not directly accessible in class *Child*.

Constructors of a Derived Class

- The constructors of the base class are not inherited in a derived class.
- The constructors of the base class are invoked (called) in the initialization section of the constructors of a derived class to initialize the inherited data members.
- The constructors of the class *Child* above are defined as follows:

```
/*----------------------Default constructors of the derived class Child ------------------------------*/
Child ::  Child(  ) : Parent(  )          // call of base class default constructor:   num1 = 0
{
    num2 =  0;
}


/*----------------------Constructor with parameters of the derived class Child -------------------------*/
Child ::  Child( int n1, int n2 ) : Parent( n1 )      // call of base class constructor:  num1  = n1
{
    num2 = n2;
}
```

Public Member Functions and Data Members of the Base Class

- A **public data member** of a *public* base class becomes a public data member of a derived class.
- A **public member function** of a *public* base class becomes a public member function of a derived class if it is not **redefined** in that derived class.
- When a public member function of the base class is redefined in a derived class, any call of that function on an object of the derived class will invoked the definition of the function in the derived class.

Example

- The public member functions *setValue1( )* and *getValue1( )* of the base class *Parent* are also public member functions of the derived class *Child*.
- The public member function *print( )* of the base class *Parent* is not a member function of the derived class *Child* because this function is redefined in the derived class *Child*.

Private Member Functions and Data Members of the Base Class

- A **private member function** of a *public* base class is not accessible in a member function of a derived class.
- A **private data member** of a *public* base class becomes a private data member of a derived class but it is not directly accessible in a member function of the derived class: it can be accessed only by using a get or a set function.

Example

Private data member *num1* of the base class *Parent* can only be accessed in the member function *print* of the derived class *Child* through the get function, *getValue1( )* as follows:

```
void  Child ::  print( )
{
    cout   <<   endl   <<   "num1 ="   <<   getValue1( ) ;
    cout   <<   endl   <<   "num2 ="   <<   num2;
}
```

Exercise O10*

Assume given the following class definition:

```
class   Item
{
    public:
        Item (    );                          //default name is " " and default quantity is 0
        Item ( string itsname, int qtity );
        void  read(    );                     // read the name and the quantity of an item
        void setName( string newName );       // set the new name of an item
        void setQtity( int newQtity);         //  set the new quantity of an item
        string getName(   );                  // returns the name of an item
        int  getQtity(   );                   //returns the quantity of an item
        void  print(   );                     // prints the name and the quantity of an item
    private:
        string name;
        int  quantity;
};
```

1. Write the definitions of the constructors and the other member functions.
2. Write the definition of the derived class of class *Item* named *Item4Sale* as follows:
   a. It has an additional private data member named *unitPrice* (double precision value).
   b. In addition to the constructors, it has the following public member functions:
      — *double getUnitPrice( )* that returns the value of the data member *unitPrice*.
      — *void setUnitPrice(double uprice)* that sets the new value of the unit price.
   c. Member function *void read( )* is redefined in the class *Item4Sale*. It reads the unit price in addition to the name and the quantity of an item.
   d. Member function *void print( )* is redefined in the class *Item4Sale*. It prints the unit price and the price of an item (the unit price times the quantity) in addition to the name and the quantity.
3. List the following:
   a. All member functions of class *Item4Sale* (each one with its access-specifier and description).
   b. All data members of the class *Item4Sale*. List each one with its access-specifier and also indicate whether or not it is accessible in a new/redefined member function of class *Item4Sale*.
4. Write the definitions of the constructors (the default unit price is $1.00), the functions *read( )* and *print( )*, and the new member functions.
5. Assume given the following definitions of objects *t1* and *t2*:

   ```
   Item  t1("Apple", 25);
   Item4Sale  t2("Orange", 34, 0.20);
   ```

   What is the output produced by the following two statements:

   ```
   t1.print(  );
   t2.print(  );
   ```

Exercise O11

Assume given the following class definition:

```
class   RectangularLot
{
    public:
        RectangularLot(    );                  //default length is 1 and default width is 1
        RectangularLot( double len, double wid );
        void  read(    );                      // read the length and the width of the lot
        void setLength( double len);           //  set the new value of the data member length
        void  setWidth( double wid );          // set the new value of the data member width
        double getLength(   );                 // returns the length of the lot
        double  getWidth(   );                 //returns the width of the lot
        double  getArea(   );                  // computes and returns the area of the lot
        void  print(   );                      // prints the length, the width and the area of the lot
    private:
        double  length;
        double  width;
};
```

1. Write the definitions of the constructors and the other member functions of this class.
2. Write the definition of the derived class of class *RectangularLot* named *Lot4Sale* as follows:
   a. It has an additional private data member named *unitPrice* (double precision value).
   b. In addition to the constructors, it has the following public member functions:
      — *double getUnitPrice( )* returns the value of the data member *unitPrice*.
      — *void setUnitPrice(double uprice)* sets the new value of the unit price.
      — *double getPrice( )* returns the price of the lot (the unit price times the area).
   c. Member function *void read( )* is redefined in class *Lot4Sale*. It reads the unit price in addition to the length, and the width.
   d. Member function *void print( )* is redefined in class *Lot4Sale*. It prints the unit price and the price of the lot (the unit price times the area) in addition to the length, the width, and the area.
3. List the following:
   a. All member functions of class *Lot4Sale* (each one with its access-specifier and its description).
   b. All data members of class *Lot4Sale*. List each one with its access-specifier and also indicate whether or not it is accessible in a new/redefined member function of class *Lot4Sale*.
4. Write the definitions of the constructors (the default unit price is $1000.00), the functions *read( )* and *print( )*, and the new member functions of the class *Lot4Sale*.
5. Assuming that objects *lot1* and *lot2* are defined as follows:

   RectangularLot lot1(80.00, 55);
   Lot4Sale lot2(120, 50, 5000.00);

   What is the output produced by the following two statements:

   lot1.print( );
   lot2.print( );

Exercise O12

Write the definition of the derived class of class **Employee** (that you have defined in exercise O9) named **BonusEmployee** as follows:

1. It has one additional private data members named *bonus* (double precision) with the default value $0.0.
2. In addition to the constructors, it has the public member function:
   double getBonus(void) that returns the value of the data member *bonus*.
3. Member function *void readPayInfo( )* is redefined in the class *BonusEmploye*: it now reads the values for the data members *base pay* and *bonus*.
4. Member function *double getGpay( )* is redefined in the class *BonusEmploye*: it now returns the value of the data member *base pay* plus the value of the data member *bonus*.
5. Member function *double computeTax( )* is redefined in the class *BonusEmploye*: It now calls the redefined member function *getGpay( )* to get the gross pay and then computes the tax deduction in the same way as in exercise O9.
6. Member function *void printPayInfo( )* is redefined in the class *BonusEmploye*: it now prints the values of the data members *bonus* and *base pay*, in addition to the gross pay, tax deduction, and net pay.
7. Place the definition of this class in the header file **BonusEmploye.h** and the definitions of the member functions in the source file **BonusEmploye.cpp**.
8. **Note**: the header file **Employee.h** must be included in the header file **BonusEmploye.h**.

Exercise O13 Extra Credit

Write the definition of the derived class of class **Employee** (that you have defined in exercise O9) named **HourlyEmployee** as follows:

1. It has three additional private data members named *hours* (integer) and *payRate* (double precision), and *overtime* (double precision). Their default values are 0 for *hours*, and $0.00 for *payRate* and *overtime*.
2. It also has a private member function *void computeBaseOvertimePay( )* that computes the base pay and the overtime pay as follows:
   a. If the value of the data member *hours* is less than or equal to 35, then it does the following:
      i. Set the value of the inherited data member *base pay* to *payRate* times *hours*.
      ii. Set the value of the data member *overtime* to 0.00.
   b. Otherwise, it does the following:
      i. Set the value of the inherited data member *base pay* to *payRate* times *35*.
      ii. Set the value of the data member *overtime* to *payRate* times 1.5 times (*hours – 35)*.
3. The constructor with parameters has the following function header:
   HourlyEmployee(string fn, string ln, int iD, Date bd, Date hd, int hrs, double prate)
   Where *hrs* represents the number of hours of work and *prate* the pay rate.
   It calls class *Employee*'s constructor with 0 as the argument for the data member *base pay*, and then calls the private member function *computeBaseOvertimePay( )* to set the values of the data members *base pay* and *overtime*.
4. In addition to the constructors, it has the following public member functions:
   a. *int getHours(void)* returns the value of the data member *hours*.
   b. *double getPayRate(void)* returns the value of the data member *payRate*.
   c. *double getOvertime(void)* returns the value of the data member *overtime*.
5. Member function *void readPayInfo( )* is redefined in the class *HourlyEmployee*: it now reads the values of the data members *hours* and *payRate*, and then calls the private member function *computeBaseOvertimePay( )* to set the values of the data members *base pay* and *overtime*.
6. Member function *double getGpay( )* is redefined in the class *HourlyEmployee*: it now returns the sum of the values of the data members *base pay* and *overtime*.
7. Member function *double computeTax( )* is redefined in the class *HourlyEmployee*: It now calls the redefined member function *getGpay( )* to get the gross pay and then computes the tax deduction in the same way as in exercise O9.
8. Member function *void printPayInfo( )* is redefined in the class *HourlyEmployee*: it now outputs the values of the data members *hours*, *payRate*, *base pay*, and *overtime*, in addition to the gross pay, tax deduction, and net pay.
9. Place the definition of this class in the header file **HourlyEmployee.h** and the definitions of the member functions in the source file **HourlyEmployee.cpp**.
10. **Note**: the header files **Employee.h** must be included in the header file **HourlyEmployee.h**.

Accessing a Redefined Member Function of the Base Class

- In C++, you can call a member function of the base class (that is redefined in a derived class) in the definition of a member function of that derived class.
- You do this by preceding the function call with the name of the base class followed by the scope resolution operator.

Example

Given the base class *Parent* and the derived class *Child* of example O11, member function *print( )* that is redefined in the derived class *Child* can be rewriting by calling the member function *print( )* of the base class *Parent* as follows:

```
void  Child ::  print(  )
    {
        Parent :: print(  ) ;          // print the first data member
        cout   <<   endl   <<   "num2 ="   <<    num2;      // print the second data member
    }
```

- In C++, you can call a member function of the base class (that is redefined in a derived class) on an object of that derived class by preceding the function call with the name of the base class followed the scope resolution operator.
- The object of a derived class on which a member function of the base class (that is redefined in the derived class) is called, is treated as an object of the base class.

Example

Given the base class *Parent* and the derived class *Child* of example O11 and the following declarations of objects:

```
Parent   pOject( 5 );
Child  cObject( 2, 3);
```

The statement	**will produce the following output**
pObject.print( );	*num1 = 5*
cObject.print( );	*num1 = 2* *num2 = 3*
cObject.Parent :: print ( );	*num1 = 2*

Exercise O14*

In exercise O10, member functions *void read( )* and *void print( )* of the base class *Item* are redefined in the derived class *Item4Sale*.

1. Write a new definition of the member function *read( )* in which there is a call to the member function *read( )* of the base class *Item* and a new definition of the member function *print( )*in which there is a call to the member function *print( )* of the base class *Item*.
2. Given the following definitions of objects *t1*, *t2*, and *t3*:

   ```
   Item  t1( "Apple", 25);
   Item4Sale t2("Banana", 50, .10), t3;
   ```

   a. What is produced by the following output statements:

   ```
   t1.print( );
   t2.print( );
   t2.Item :: print( );
   ```

   b. Write a statement to read the name and the quantity of object *t3* by calling the *read( )* member function of the base class *Item*.

Exercise O15

In exercise O11, member functions *void read( )* and *void print( )* of the base class *RectangularLot* are redefined in the derived class *Lot4Sale*.

1. Write a new definition of the member function *read( )* in which there is a call to the member function *read( )* of the base class *RectangularLot* and a new definition of the member function *print()* in which there is a call to the member function *print( )* of the base class *RectangularLot*.
2. Given the following definitions of objects *lot1*,*lot2*, and *lot3*:

   ```
   RectangularLot  lot1( 80, 25);
   Lot4Sale lot2(100, 50, 10000), lot3;
   ```

   a. What is produced by the following output statements:

   ```
   lot1.print( );
   lot2.print( );
   lot2.RectangularLot :: print( );
   ```

   b. Write a statement to read the length and the width of object *lot3* by calling the *read( )* member function of the base class *RectangularLot*.

Exercise O16

In exercise O12, member functions *void readPayInfo( )* of the base class *Employee* is redefined in the class *BonusEmployee*.

1. Write a new definition of the member function *readPayInfo( )* in which there is a call to the member function *readPayInfo( )* of the base class *Employee*.
2. Given the following definitions of objects *emp1,emp2,* and *emp3*:

   Employee emp1("John", "Doe", 111111, Date(10, 25, 1991), Date(5, 10, 2010), 1250);
   BonusEmployee emp2("Job", "Daly", 222222, Date(1, 5, 1990), Date(6, 20, 2011), 850, 250), emp3;

   a. What is produced by the following output statements:

   emp1.printPayInfo( );
   emp2.printprintPayInfo( );
   emp2.Employee :: printPayInfo( );

   b. Write a statement to read the base pay of object *emp3* by calling the *readPayInfo( )* member function of the base class *Employee*.

Protected members of a Class

- When you write the definition of a class, you can make a member function or a data member of that class a **protected member** by preceding its declaration/definition with the label **protected:** instead of the label **private:**
- A **protected data member/member function** of a base class is like a private data member/member function of that class except that it can be directed accessed in a member function of a derived class.

Example

If we redefine the base class *Parent* of example O11 as follows:

```
class Parent
{
    public:
      Parent( );                  // default constructor
      Parent( int n );            // constructor
      void  setValue1( int n );
      int getValue1( );
      void print( );              // print the value of the member variable
    protected:
      int num1;
};
```

Then we can now access the protected data member *num1* in the member function *print( )* of the derived class *Child* as follows:

```
void  Child ::  print( )
{
    cout   <<   endl   <<   "num1 ="   <<   num1;
    cout   <<   endl   <<   "num2 ="   <<   num2;
}
```

Exercise O17*

Assuming that the private data members *name* and *quantity* of the class *Item* defined in exercise O10 are protected data members, rewrite the definitions of the member functions *void read( )* and *void print( )* that are redefined in the class *item4Sale*.

Exercise O18

Assuming that the private data members *length* and *width* of the class *RectangularLot* defined in exercise O11 are protected data members, rewrite the definitions of the member functions *void read( )* and *void print( )* that are redefined in the class *Lot4Sale*.

Exercise O19

Assuming that the private data member *base pay* of the class *Employee* defined in exercise O9 is a protected data member, rewrite the definitions of the member functions *void readPayInfo( ), double getGpay( )*, and *void printPayInfo( )* that are redefined in the class *BonusEmploye*.

Data Type of an Object of a Derived Class

- **An object of a derived class is also an object of the base class (without the additional data members and member functions)**:
  - It can be used anywhere (such as arguments to functions) that an object of its base class is allowed.
  - It can also be assigned to an object of the base class.
- However, **an object of the base class is not an object of a derived class**.

Example

Given the base class *Parent* and the derived class *Child* of example O11, the following function definition:

```
void  funct( Parent obj )
{
    cout   <<   10 * obj.getValue1(  );
}
```

And the following definitions of objects:

```
Parent  pobj1( 2 ) , pobj2;
Child  cobj1( 3, 4 ) , cobj2;
```

The following assignment statement is legal:

```
pobj2  = cobj1;          //  pobj2.num1 is set to 3
```

The following assignment statement is illegal:

```
cobj2  =  pobj1.    //  you cannot assign an object of the base class to an object of a derived class
```

The following call statements are legal:

```
funct( pobj1 );          // output will be:    20
funct( cobj1 );          //  output will be:  30
```

Exercise O20*

Using the definitions of the classes *Item* and *Item4Sale* of exercise O10, we define function *helper( )* as follows:

```
void   helper ( Item t )
        {
            t.print (   );
        }
```

And the object *t1*, *t2*, and *t3* as follow:

```
Item  t1( "Pie", 60 ), t2;
Item4Sale t3( "hat", 100, 30.00);
```

Execute the following statements and show their output:

a. t1.print( );
b. t3.print( );
c. helper (t1);
d. helper (t3);
e. t2 = Item4Sale(“Shirt”, 150, 10.0);
   t2.print( );

Exercise O21

Using the definitions of the classes *RectangularLot* and *Lot4Sale* of exercise O11, we define function *helper( )* as follows:

```
void   helper ( RectangularLot  lot )
        {
            lot.print (   );
        }
```

And the object *myLot*, *yourLot*, and *hisLot* as follow:

```
RectangularLot  myLot( 30, 200 ), hisLot;
Lot4Sale  yourLot ( 50, 100, 2000);
```

Execute the following statements and show their output:

a. myLot.print( );
b. yourLot.print( );
c. helper (myLot);
d. helper (yourLot);
e. hisLot = Lot4Sale(40, 150, 3000);
   hisLot.print( );

Virtual Functions: Dynamic/Late Binding

- A function of the base class that will be redefined in derived classes can be made a **virtual function** by adding the keyword *virtual* to its function prototype in the base class definition.
- **It is necessary to make a function of a base class a virtual function in the following situation**:

  functA( ) and *functB( )* are member functions of a base class such that:

  - Member function *functA* is called in member function *functB( )*.
  - Member function *functA* is redefined in a derived class.
  - Member function *functB* is not redefined in that derived class.
  - When member function *functB( )* is called on an object of the base class, we want the definition of the member function *functA( )* in the base class to be used in the function call, and
  - When member function *functB( )* is called on an object of the derived class, we want the definition of the member function *functA( )* in the derived class to be used in the function call
- This can only be done by making member function *functA( )* a *virtual* function.
- Note that if you do not want to make member function *functA( )* a virtual function, you will have to redefine member function *functB( )* in all derived classes as we did for the member function *computeTax( )* of the classes *Employee* of exercise O9, *BonusEmployee* of exercise O12 and *HourlyEmployee* of exercise O13. By doing this, the definition of member function *functA( )*in each class is bound to the function call in the member function *functB( )* defined in that class.
- By making a member function of a base class a virtual function, you are saying to the compiler that it should not use the definition of that function that is provided in the base class when it is processing that function call: the binding of the function call to the proper definition of the virtual function must be delayed until the function that call the virtual function is called in the program: C++ chooses the definition of the virtual function that corresponds to the object been processed.
- Delaying the binding of a virtual function call to its definition is referred to as **late binding**.
- A late binding is referred to as **dynamic binding** if it occurs during the execution of the program.

Note

- The definition of a virtual function in a base class serves as a default definition: It will be used on objects of a derived class only if its definition is not overridden in that derived class.

Example O12

A company sales two shapes of tiles (rectangular, and triangular) and the price of a tile is the unit price per square inch times the area of the tile. To process the information about these tiles, we design a base class named *Tile* to represent the information common to all tiles and then create derived classes *RectangleTile* and *TriangleTile* to represent the information about each shape of tile as follows:

```
class  Tile
{
    public:
        Tile(  );
        Tile( double uprice );
        double  getUprice(   );
        virtual  double computeArea(  );       //a place holer function
        double  computePrice(  );
        void  print(  );
    private:
        double unitPrice;
};


/*-------------------------------- Definition of  the default constructor -------------------------*/
Tile ::  Tile(  ) : unitPrice ( 2.0 )
{
}

/*-------------------------------- Definition of  the constructor with parameters -------------------------*/
Tile ::  Tile ( double uprice ) : unitPrice( uprice )
{
}

/*-------------------------------- Definition of member function getUprice(  ) -------------------------*/
double  Tile ::  getUprice( void )
{
    return unitPrice ;
}

/*-------------------------------- Definition of  member function computeArea(  ) -------------------------*/
double  Tile ::  computeArea( void )
{
    return ( 1 );                              // return a dummy area of 1sqrt inch
}

/*-------------------------------- Definition of  member function computePrice(   ) -------------------------*/
double  Tile :: computePrice(  )
{
    double  area;
    area  =  computeArea(  );          // call to the virtual function compute Area is delayed
    return( unitPrice  *  area );
}
```

Note

Although member function *computeArea( )* is not needed in the class *Tile,* we need its definition in order to write the definition of member function *computePrice( )*.

```
/*-------------------------------- Definition of  member function print(  ) ------------------------*/
void  Tile ::  print( void )
{
    cout  <<  endl  <<  "The price of the tile is:\t"  <<  computePrice ( );
}
```

Derived Classes:

```
    class RectangleTile  :  public  Tile      // inherits class Tile
    {
        public:
            RectangleTile ( );                          // default constructor
            RectangleTile ( double uprice , double len , double wth);     // constructor
            virtual  double computeArea( );             // function to be called for Rectangle objects
            void  print( );
        private:
            double length;
            double  width;
    };
```

```
/*-------------------------------- Definition of  the default constructor ------------------------*/
RectangleTile ::  RectangleTile( ) : Tile( ) , length( 0.5 ) , width( 1.0 )
{
}
```

```
/*-------------------------------- Definition of  the constructor with parameters ------------------------*/
RectangleTile :: RectangleTile(double len , double wth, double uprice ):Tile( uprice ) , length( len ) ,
width( wth )
{
}
```

```
/*-------------------------------- Re-definition of  member function computeArea( ) ------------------------*/
double  RetangleTile ::  computeArea( void )
{
    return ( length * width );                  // return the area of a rectangular tile
}
```

```
/*------------------------------- Re-definition of  member function print( ) -------------------------*/
void  RectangleTile ::  print( void )
{
    cout  <<  endl  <<  "The length is:\t"  <<  length
          <<  endl  <<  "The width is:\t"  <<  width;
    cout  <<  endl  <<  "The price of the tile is:\t"  <<  computePrice ( );
}


class TriangleTile  :  public  Tile       // inherits class Tile

{
    public:
        TriangleTile ( );                                          // default constructor
        TriangleTile (double ht , double bse, double uprice );     // constructor
        virtual  double computeArea( );          // function to be called for triangle objects
        void  print( );
    private:
        double height;
        double  base;
};


/*------------------------------- Definition of  the default constructor -------------------------*/
TriangleTile ::  TriangleTile ( ) : Tile( ) , height( 0.5 ) , base( 1.0 )
{
}

/*------------------------------- Definition of  the constructor with parameters -------------------------*/
TriangleTile ::  TriangleTile (double ht , double bse, double uprice ) : Tile( uprice ) , height( ht ) , base(
bse )
{
}


/*------------------------------- Redeefinition of  member function computeArea( ) -------------------------*/
double  TriangleTile ::  computeArea( void )
{
    return ( height * base / 2.0 );              // return the area of a rectangular tile
}
```

```
/*------------------------------ Re-definition of  member function print( ) -------------------------*/
void  TriangleTile ::  print( void )
{
    cout  <<  endl  <<  "The height is:\t"  <<  height
        <<  endl  <<  "The base is:\t"  <<  base;
    cout  <<  endl  <<  "The price of the tile is:\t"  <<  computePrice ( );
}
```

Exercise O22*

Given the following definitions of objects,

Tile obj1, obj2(10.0);

RectangleTile robj1, robj2(3.0, 5.0, 4.0);

Show the following outputs:

a. obj1.print( );
b. obj2.print ( );
c. robj1.print( );
d. robj2.print( );

Exercise O23

Given the following definitions of objects,

Tile obj1(5.0), obj2 (20.0);

TriangleTile tobj1, tobj2(2.0, 6.0, 8.0);

Show the following outputs:

a. obj1.print( );
b. obj2.print ( );
c. tobj1.print( );
d. tobj2.print( );

Exercise O24

1. Modify class *Employee* that you have defined in exercise O9 by making the member function *double getGpay( )* a virtual function.
2. With this modification of the class *Employee*, explain why you do not need to redefine the member function *double computeTax( )* in the class *BonusEmployee*.
3. Assuming that member function *double getGpay( )* is a virtual member function of the class *Employee*, rewrite the definition of the class *BonusEmploye* without redefining the member function *double computeTax( )* of the base class *Employee*.

Exercise O25 Extra Credit

Assuming that member function *double getGpay( )* is a virtual member function of the class *Employee*, rewrite the definition of the class *HourlyEmployee* without redefining the member function *double computeTax( )* of the base class *Employee*.

Pure Virtual Functions and Abstract Classes

- A *virtual member function of a base class* whose definition is not needed in that class can be made a **pure virtual function** by following its function prototype in the class definition with the expression: **= 0**.
- Note that you do not have to write the definition of a *pure virtual function* in the base class: its definitions are provided in the derived classes of the base class.
- A class with one or more pure virtual functions is referred to as an **abstract base class**.
- An *abstract base class* cannot be used to instantiate objects.

Example

The base class *Tile* of example O12 can be rewritten by making the member function *double computeArea( )* a pure virtual function as follows:

```
class  Tile
{
    public:
        Tile( );
        Tile( double uprice );
        double  getUprice( );
        virtual  double computeArea( ) = 0 ;      //a place holer function
        double  computePrice( );
        void  print( );
    private:
        double unitPrice;
};
```

Polymorphic Functions

- A **polymorphic function** is a function that behaves in different ways for different kinds of objects.
- A virtual function can be used to create a polymorphic function as in the following example:

Example O13

Assume given the class *Tile* and its derived classes *RectangleTile* and *TriangleTile* of example O12 with the following modification: member function *print( )* is now a virtual function and a new member function *void readDimensions( )* is added.

```
class  Tile
{
    public:
            Tile( );
            Tile( double uprice );
            double  getUprice( );
virtual void readDimensions( ) = 0;     // a place holder function to read the dimensions of a tile
            virtual  double computeArea( );      //default definition of  member function computeArea( )
            double  computePrice( );
            virtual void  print( );
        private:
            double unitPrice;
    };

/*-------------------------------- Definition of constructors and member functions -------------------------*/
Tile ::  Tile( ) : unitPrice ( 2.0 )
{
}

Tile ::  Tile ( double uprice ) : unitPrice( uprice )
{
}

double  Tile ::  getUprice( void )
{
    return unitPrice ;
}

double  Tile ::  computeArea( void )
{
    return ( 1 );          // return a dummy area of 1sqrt inch
}
```

```
double  Tile :: computePrice(  )
{
    double  area;
    area  =  computeArea(  );          // call to the virtual function compute Area is delayed
    return( unitPrice  *  area );
}

void  Tile ::  print( void )
{
    cout  <<  endl  <<  "The price of the tile is:\t"  <<  computePrice (  );
}
```

Derived Class *RectangleTile*:

```
class RectangleTile   :   public   Tile       // inherits class Tile
{
    public:
        RectangleTile (  );                                          // default constructor
        RectangleTile ( double uprice , double len , double wth);    // constructor
        virtual  double computeArea(  );              // function to be called for Rectangle objects
        virtual void  print(  );
    private:
        double length;
        double  width;
};

/*-------------------------------- Definition of constructors and member functions -------------------------*/
RectangleTile ::  RectangleTile(  ) : Tile(  ) , length( 0.5 ) , width( 1.0 )
{
}

RectangleTile :: RectangleTile(double len , double wth, double uprice ):Tile( uprice ) , length( len ) ,
width( wth )
{
}

double  RetangleTile ::  computeArea( void )
{
    return ( length * width );                  // return the area of a rectangular tile
}
```

```
void  RectangleTile ::  print( void )
{
    cout  <<  endl  <<  "The length is:\t"  <<  length
          <<  endl  <<  "The width is:\t"  <<  width;
    cout  <<  endl  <<  "The price of the tile is:\t"  <<  computePrice ( );
}
```

Derived Class *TriangleTile*:

```
class TriangleTile  :  public  Tile        // inherits class Tile
{
    public:
        TriangleTile ( );                                          // default constructor
        TriangleTile (double ht , double bse, double uprice );     // constructor
        virtual  double computeArea( );          // function to be called for triangle objects
        virtual void  print( );
    private:
        double height;
        double  base;
};

/*-------------------------------- Definition of constructors and member functions -------------------------*/
TriangleTile ::  TriangleTile ( ) : Tile( ) , height( 0.5 ) , base( 1.0 )
{
}

TriangleTile::TriangleTile (double ht , double bse, double uprice ):Tile( uprice) , height( ht) , base(bse )
{
}

double  TriangleTile ::  computeArea( void )
{
    return ( height * base / 2.0 );              // return the area of a rectangular tile
}

void  TriangleTile ::  print( void )
{
    cout  <<  endl  <<  "The height is:\t"  <<  height
          <<  endl  <<  "The base is:\t"  <<  base;
    cout  <<  endl  <<  "The price of the tile is:\t"  <<  computePrice ( );
}
```

Given the following definition of function *helper( )*:

```
void  helper( const  Tile & obj )
{
    obj.print( );
}
```

And the following definitions of objects:

```
RectangleTile item2( 5.0, 3.0, 2.0);
TriangleTile item3( 4.0, 6.0, 1.0 );
```

The execution of the following statements will produce the following output:

```
helper( item2 );    cout  <<  endl;
helper( item3 );    cout  <<  endl;
```

OUTPUT

```
The length is:  5.0
The width is:   3.0
The price of the tile is:     30.0
The height is:  4.0
The base is:    6.0
The price of the tile is:     12.0
```

Note:

Function *helper ( )* is a polymorphic function: it behaves in different ways for different kinds of objects.

➢ friend Functions

- An ordinary function that has an object of a class as a parameter or that processes an object of a class as a local/global variable can be made a **friend function** of that class.
- The only purpose of making an ordinary function a **friend function** of a class is for that function to have access to the private data members and member functions of that class.
- You make an ordinary function a **friend function** of a class by preceding its prototype in that class definition with the keyword **friend**.
- This declaration may appear in any section (private/public) of a class definition.

Example O14

Given the following definition of class *Demo8*:

```
class  Demo8
{
    public:
        Demo8(int n1 = 0, int n2 = 0);    // constructor
        int getFirst( );                  // returns the value of the first member variable
        int getSecond( );                 // returns the value of the second member variable
    private:
        int val1;
        int val2;
};
```

Function *addDemo8(Demo8 ob1, Demo8 obj2)* that receives as arguments two objects of the class *Demo8* and returns another object of the class *Demo8* such that the value of each of its member variables is the sum of the values of the correspond member variables of the objects that it receives as arguments and function *updateDemo8(Demo8 &obj)* that receives as argument an object of the class *Demo8* and adds 1 to the value of each of its member variables are defined as ordinary functions as ordinary functions as follows:

```
/*----------------------------------------function addDemo8( ) ----------------------------------------------------*/
/*  receives two objects of the class Demo8 and returns another object of the same class with
   each member variable the sum of thevalues of the corresponding member variables of both objects */
Demo8  addDemo8( Demo8 obj1, Demo8 obj2)
{
    Demo8 objResult;
    int num1 = obj1.getFirts( )  + obj2.getFirst( );
    int num2 = obj1.getSecond( ) + obj2.getSecond(  );
    objResult  =  Demo8( num1, num2 );
    return ( objResult);
}
```

```
/*---------------------------------------function updateDemo8( ) ----------------------------------------------*/
/*  receives an object of the class Demo8 and adds 1 to the value of  each of its member variables */
void updateDemo8( Demo8 &obj )
{
    int  num1  = obj.getFirst(   ) ;
    int  num2  = obj.getSecond(   ) ;
    obj  =  Demo8( num1+1, num2+1 );
}
```

These two functions are defined as friend functions of the class *Demo8* as follows:

```
class  Demo8
{
    public:
        Demo8(int n1 = 0, int n2 = 0);    // constructor
        int getFirst( );                  // returns the value of the first member variable
        int getSecond( );                 // returns the value of the second member variable
        friend  Demo8  addDemo8( Demo8 ob1, Demo8 obj2);
        friend  void  updateDemo8( Demo8 &obj);
    private:
        int val1;
        int val2;
};
```

```
/*---------------------------------------function addDemo8( ) ----------------------------------------------------*/
/*  receives two objects of the class Demo8 and returns another object of the same class with
   each member variable the sum of the corresponding member variables of both objects          */
Demo8  addDemo8( Demo8 obj1, Demo8 obj2)
{
    Demo8 objResult;
    objResult.val1 = obj1.val1 + obj2.val1;
    objResult.val2 = obj1.val2 + obj2.val2;
    return ( objResult);
}
```

```
/*---------------------------------------function updateDemo8( ) ----------------------------------------------*/
/*  receives an object of the class Demo8 and increments each of its member variables by 1          */
void updateDemo8( Demo8 &obj )
{
    obj.val1 ++ ;
    obj.val2 ++ ;
}
```

```
/*---------------------- calling functions addDemo8( ) and updateDemo8( ) ---------------------------------*/
Demo8  tobj(14, 25), sobj(5, 9), robj;

robj = addDemo8 ( tobj, sobj );
cout  <<  endl  <<  "first value is:\t"  <<  robj.getFirst( )
      <<  endl  <<  "second value is:\t"  <<  robj.getSecond( );
updateDemo8 ( tobj );
cout  <<  endl  <<  "first value is:\t"  <<  tobj.getFirst( )
      <<  endl  <<  "second value is:\t"  <<  tobj.getSecond( );
```

Exercise O26

Given the following definition of class *Demo8*:

```
class  Demo8
{
    public:
        Demo8(int n1 = 0, int n2 = 0);    // constructor
        int getFirst( );                  // returns the value of the first member variable
        int getSecond( );                 // returns the value of the second member variable
    private:
        int val1;
        int val2;
};
```

Function *subDemo8(Demo8 ob1, Demo8 obj2)* receives as arguments two objects of the class *Demo8* and returns another object of the class *Demo8* such that the value of each of its member variables is the difference of the value of the corresponding member variable of the first object minus the value of the corresponding member variable of the second object that it receives as arguments and function *decDemo8(Demo8 &obj)* receives as argument an object of the class *Demo8* and subtracts 1 from the value of each of its member variables.

1. Provide the definitions of these two functions as ordinary functions.
2. Provide the definitions of these two functions as friend functions of the class *Demo8*.
3. Given the following objects:
   Demo8 obj1(5, 8), obj2(12, 21), obj3(9, 6), obj4;

   Write the statements to assign values the member variables of object *obj4* such that the value of each member variable is the difference of the value of the corresponding member variable of object *obj1* minus the value of the corresponding member variable of object *obj2* (by calling function *subDemo8( )*). Also write the statements to decrement the value of each member variable of object *obj3* by 1 (by calling function *decDemo8( )*).

Solution Exercises

Exercise C1

a.

```
void  Automobile : : setPrice(   double newPrice )
{
     price  = newPrice;
}

void  Automobile : : setProfit(   double newProfit )
{
     profit  = newProfit;
}

double  Automobile : : getPrice(   )
{
     return price;
}

double  Automobile : : getProfit(   )
{
     return profit;
}
```

b.

Invalid Statement	Reasons
hyunday.price = 15000.00;	*price* is a private data member of the class
cout << jaguar.getProfit ( );	*getProfit( )* is a private member function of the class
jaguar.setProfit( );	The class member function *setProfit( )* is called without an argument
setPrice (16500);	The class member function *setPrice( )* is not called on an object

a.
```
jaquar.setProfit(  3500 );
```

b.
```
double  carPrice;
cin  >>  carPrice;
hynday.setPrice( carPrice );
```

c.
```
cout  <<  jaguar.getPrice(  );
```

d. No! Because data member *profit* and member function *getProfit( )* that returns the value of data member profit are private.

e.
```
hyunday  =  jaguar;
hynday.setPrice( 24000.00 );
```

Exercise C4

a.
```
Demo1  addDemo1 ( Demo1 o1, Demo1  o2 )
{
    Demo1  temp;
    double res2;
    temp.val1  =  o1.val1  +  o2.val1;
    res2  =  o1.getValue2( )  +  o2.getValue2(  );
    temp.setValue2 ( res2 );
    return ( temp );
}
```
b.
```
Demo1  obj1, obj2, objR;
obj1.Val1 = 5;
obj1.setValue2 ( 7 );
obj2.Val1 = 10;
obj2.setValue2 ( 15 );
objR  = addDemo1 ( obj1, obj2 );
```

Exercise C5

a.
```
void   updateDemo1 ( Demo1 & o )
{
    double  num2;
    o.val1 += 10;
    num2  =  o.getValue2 (  );
    o.setValue2 ( num2 - 5 );
}
```
b.
```
Demo1  obj;
obj.val1  =  15;
obj.setValue2 ( 70 );
updateDemo1( obj );
```

Exercise O1

1.
```
class  OurClass
{
    public:
        OurClass (  );                          // default constructor
        OurClass ( int num, char symb);         // constructor with parameters
        void  setFirst(int newVal);             // to set the value of data member first
        void  setSecond(char newSymb);          // to set the value of  data member second
        int  getFirst(void);                    // to return the value of data member first
        char  getSecond(  );                    // to return the value of data member second
        void read(void);                        // to read the values of the two data members
        void print(  );                         // to output the values of the two data members
    private:
        int first;                              // the first data member
        char second;                            // the second data member
};
```

```
/*------------------------------Definition of the default constructor----------------------------------*/
OurClass :: OurClass(   )
{
     first = 0;
     second = 'A';
}

/*------------------------------Definition of the constructor with parameters--------------------------*/
OurClass :: OurClass ( int num, char symb)
{
     first = num;
     second = symb;
}
/*------------------------------member function setFirst(  )  -----------------------------*/
void OurClass :: setFirst(int newVal)
{
     first = newVal;
}
/*------------------------------member function setSecond(  )  -----------------------------*/
void OurClass :: setSecond(char newSymb)
{
     second = newSymb;
}
/*------------------------------member function getFirst(  )  -----------------------------*/
int  OurClass :: getFirst(void);
{
     return first;
}

/*------------------------------member function getSecond(  )  -----------------------------*/
int  OurClass :: getSecond(void);
{
     return second;
}

/*------------------------------member function read(  )  -----------------------------*/
void  OurClass :: read(void);
{
     cin   >>   first   >>   second;
}

/*------------------------------member function print(  )  -----------------------------*/
void  OurClass :: print(void);
{
     cout   <<   endl    <<   first    <<   endl   <<   second;
}
```

2.

```
/*------------------------------ function maxObj( ) ----------------------------*/
OurClass   maxObj(  OurClass obj1,  OurClass obj2 )
{
    OurClass objR;
    int  val1  =  obj1.getFirst( ),
         val2   = obj2.getFirst( );
    char symb1  = obj1.getSecond( ),
          symb2  = obj2.getSecond( );
    if(  val1 >  val2 )
         objR.setFirst( val1 );
    else
         objR.setFirst( val2 );
    if(  symb1 >  symb2 )
         objR.setSecond( symb1 );
    else
         objR.setSecond( symb2 );
    return objR;
}

/*------------------------------ function incrObj( ) ----------------------------*/
void  incrObj(  OurClass & obj )
{
    int  val  =   obj.getFirst(   );
    obj.setFirst( val + 1 );
}
```

3.

```
OurClass   objA,   objB,   objC( 100,  'P' );
```

4.

a. objA.read( );
b. objB.setFirst(15);
c. int val = objC.getFirst( );
   objC.setFirst (val + 10);
d. objC.print( );
e. OurClass newObj = maxObj(objA, objB);
f. incrObj(objC);

Exercise O3

a. MyClass obj1;
b. MyClass obj2 (34, 'A');
c. MyClass obj3 ( ); Valid (but declaration of function obj3)
d. MyClass obj4;
   obj4 = MyClass ( );
e. MyClass obj5;
   obj5 = MyClass (21, 'Z');
f. MyClass obj6 (10, 'C');
   MyClass obj7 (obj6);
g. MyClass obj8 (5, 'X');
   MyClass obj9 = obj8 ;
h. MyClass obj10 = MyClass; Invalid

Exercise O5

```
class  OurClass
{
    public:
        OurClass ( int num = 0, char symb = 'A');      // constructor with parameters
        void  setFirst(int newVal);                    // to set the value of data member first
        void  setSecond(char newSymb);                 // to set the value of  data member second
        int  getFirst(void);                           // to return the value of data member first
        char  getSecond( );                            // to return the value of data member second
        void read(void);                               // to read the values of the two data members
        void print( );                                 // to output the values of the two data members
    private:
        int first;                                     // the first data member
        char second;                                   // the second data member
};

/*------------------------------Definition of the constructor with parameters----------------------------*/
OurClass :: OurClass ( int num, char symb) :  first( num ), second( symb )
{
}
```

Exercise O8

1.

```
class  Student
{
    Public:
        Student( );
        Student(  string  fname,  string lname,  int id,  DayOfYear  dbay, double g );
        void  read( void );
        void print( );
    private:
        string firtName;
        string  lastName;
        int  idNum;
        DayOfyear birthday;
        double gpa;
};
```

2.

```
/*---------------------------default constructor ----------------------------------------------------*/
Student : :  Student (  ) :  idNum( 999999 ), gpa( 0 )
{

}
```

```
/*-----------------------------------------constructor with parameters ----------------------------------------------*/
Student : :  Student(  string  fname,  string lname,  int id,  DayOfYear  dbay, double g ) :  firstName( fname),
              lastName( lname ), birthday (bday )
{
    idNum  = id;
    gpa = g;
}

/*---------------------------------- Member function read( ) ----------------------------------------------*/
void  Student : : read(   )
{
    cin  >>  firstName   >>   lastName  >>  idNum;
    birthDay.input( );                  // read the values for the birth day month and day
    cin  >>  gpa;
}

/*---------------------------------- Member function print( ) ----------------------------------------------*/
void  Student  :: print(   )
{
    cout  <<  endl  <<  "Name:\t"  <<  lastName + ", " +  firstName;
    cout  <<  endl  <<  "ID NUM:\t"  <<  idNum;
    cout  <<  endl  <<  "Birth Day:\t";
    birthDay.output( );                 // print the values for the birth day month and day
    cout  <<  endl  <<  "GPA:\t"  <<  gpa;
}
```

3.

```
Student   toto;              // define object toto
toto.read( );                // read values for the member variable of object toto
toto.print( );               // output the values of the member variables of object toto
```

4.

```
Student  newStu ( "Mark", "Depaul", 1123, DayOfYear( 3, 25 ), 3.25 );
```

5.

```
newStu.print( );
```

Exercise O10

1.

```
/*----------------------------default constructor ----------------------------------------------*/
Item :: Item(   )
{
    quantity = 0;
}

/*-----------------------------------------constructor with parameters ----------------------------------------------*/
Item ::  Item( string  NewName,  int newQtity ) :  name( NewName),  quantity( newQtity )
{
}
```

```
/*---------------------------------- Member function read( ) ----------------------------------------------*/
void  Item :: read(   )
{
     cin   >>   name    >>    quantity;
}

/*---------------------------------- Member function setName( ) ----------------------------------------------*/
void  Item :: setName( string newName )
{
     name = newName;
}

/*---------------------------------- Member function setQtity( ) ----------------------------------------------*/
void  Item :: setQtity(  int newQtity )
{
     quantity  = newQtity;
}

/*---------------------------------- Member function getName( ) ----------------------------------------------*/
string  Item :: getName( void )
{
     return ( name );
}

/*---------------------------------- Member function getQtity( ) ----------------------------------------------*/
int  Item :: getQtity(  void )
{
     return ( quantity );
}

/*---------------------------------- Member function print( ) ----------------------------------------------*/
void  Item  :: print(   )
{
     cout  <<  endl  <<   "Name:\t"  <<  name;
     cout  <<  endl  <<  "Quantity:\t"  <<  quantity;
}
```

2.

```
class   Item4Sale  :  public Item
{
     public:
          Item4Sale(   );
          Item4Sale( string newName, int newQtity, double newUPrice );
          double getUnitPrice(    );
          void setUnitPrice( double newUPrice );
          void  read(   );
          void  print(   );
     private:
          double  unitPrice;
};
```

3.

Member Functions of class Item4Sale	Access Specifiers	Descriptions
setName( )	public	Set the new name of an item
setQtity( )	public	Set the new quantity of an item
getName( )	public	Return the name of an item
getQtity( )	public	Return the quantity of an item
getUnitPrice( )	public	Return the unit price of an item
setUnitPrice( )	public	Set the new unit price of an item
Redefined function read( )	public	Read the name, quantity, and unit price of an item
Redefined function print( )	public	Output the name, the quantity, and unit price of an item

Data Member of class Item4Sale	Access Specifiers	
name	private	Not accessible in a new/redefined member function
quantity	private	Not accessible in a new/redefined member function
unitPrice	private	Accessible in a new/redefined member function

4.

```
/*----------------------------default constructor ----------------------------------------------------*/
Item4Sale :: Item4Sale( ) :  Item( ), unitPrice( 1.00 )
{
}

/*-----------------------------------------constructor with parameters ----------------------------------------------*/
Item4Sale ::  Item4Sale( string  NewName,  int newQtity, double newUPrice ) :  Item( NewName, newQtity)
{
     unitPrice  = newUPrice;
}

/*---------------------------------- Member function read( ) ----------------------------------------------*/
void  Item4Sale :: read( )
{
     string  iName;
     int iqtity;
      cin  >>  iname  >>  iqtity  >>  unitPrice;
     setName( iName );
     setQtity( iqtity );
}

/*---------------------------------- Member function print( ) ----------------------------------------------*/
void  Item4Sale  :: print( )
{
     cout  <<  endl  <<  "Name:\t"  <<  getName( );
     cout  <<  endl  <<  "Quantity:\t"  <<  getQtity( );
     cout  <<  endl  <<  Unit Price:\t"  <<  unitPrice;
}
```

```
/*---------------------------------- Member function setUnitPrice( ) ----------------------------------------*/
void  Item4Sale :: setUnitPrice( double newUPrice )
{
     unitPrice = newUPrice;
}

/*---------------------------------- Member function getUnitPrice(    )----------------------------------------*/
double  Item4Sale :: getUnitPrice(    );
{
     return (unitPrice) ;
}
```

5.

Output produced by: t1.print( )	**Output produced by: t2.print( )**
Name: Apple Quantity: 25	Name: Orange Quantity: 34 Unit Price: 0.20

Exercise O14

1.

```
/*---------------------------------- Member function read( ) ----------------------------------------------*/
void  Item4Sale :: read(  )
{
     Item :: read(  );
      cin  >>  unitPrice;
}

/*---------------------------------- Member function print( ) ----------------------------------------------*/
void  Item4Sale  :: print(  )
{
     Item :: print(  );
     cout  <<  endl  <<  Unit Price:\t"  <<  unitPrice;
}
```

2.

a.

Output produced by: t1.print( )	**Output produced by: t2.print( )**	**Output produced by: t2.Item :: print( )**
Name: Apple Quantity: 25	Name: Banana Quantity: 50 Unit Price: 0.10	Name: Banana Quantity: 50

b.

```
t3.Item :: read(  );
```

Exercise O17

```
/*---------------------------------- Member function read( ) ----------------------------------------------*/
void  Item4Sale :: read(   )
{

      cin  >>  name   >>   quantity   >>   unitPrice;
}

/*---------------------------------- Member function print( ) ----------------------------------------------*/
void  Item4Sale  :: print(   )
{
     cout  <<  endl  <<  "Name:\t"  <<  Name;
     cout  <<  endl  <<  "Quantity:\t"  <<  quantity;
     cout  <<  endl  <<  Unit Price:\t"  <<  unitPrice;
}
```

Exercise O20

Statements:	**Output:**
t1.print( );	Name: Pie Quantity: 60
t3.print( );	Name: hat Quantity: 100 Unit Price: 30.00
helper (t1);	Name: Pie Quantity: 60
helper (t3);	Name: hat Quantity: 100
t2 = Item4Sale("Shirt", 150, 10.0); t2.print( );	Name: Shirt Quantity: 150

Exercise O22

Statements:	**Output:**
obj1.print( );	The price of the tile is: 2.00
obj2.print( );	The price of the tile is: 10.00
robj1.print( );	The length is: 0.5 The width is: 1.0 The price of the tile is: 1.00
robj2.print( );	The length is: 3.0 The width is: 5.0 The price of the tile is: 60.00