```
1.  #include <iostream>
2.  using namespace std;
3.  struct Rectangle {
4.      double length;
5.      double width;
6.  };
7.  int main()
8.  {   Rectangle R={12.5,7};
9.      cout << "length=" << R.length;
10.     cout << " ";
11.     cout << " width=" << R.width;
12.     cout  << endl;
13.     return 0;
14. }
```

Figure 1. Program **p01**

Program **p01** produces**:**
length= 12.5 width=7

Program **p01** notes:

1. Program **p01** illustrates how the *structure-type-specifier* can be used to create a type in C++.
2. The grammar for C++ has been enhanced to permit the *tag-name* to be used everywhere. This is a welcome addition eliminating the more cumbersome *typedef* declaration (not shown).
3. *Rectangle R* is initialized on line 8. Note the initialization syntax. Values enclosed in curly braces and separated by commas are assigned to members in the order that they are declared.

```
1.  #include <iostream>
2.  using namespace std;
3.  struct Rectangle {
4.      double length;
5.      double width;
6.      double Area(void){return length*width;}
7.  };
8.  int main()
9.  {   Rectangle R={12.5,7};
10.     cout << "length=" << R.length ;
11.     cout << " ";
12.     cout << " width=" << R.width ;
13.     cout << " ";
14.     cout << " Area="  << R.Area();
15.     cout << endl;
16.     return 0;
17. }
```

Figure 2. Program **p02**

Program **p02** produces**:**
length= 12.5 width=7 Area=87.5

Program **p02** notes:

1. Program **p02** illustrates member functions.
2. The grammar for C++ permits members that are functions as well as members that contain data. Observe function *Area* that is defined on line 6 on figure 3.
3. Member functions are accessed in exactly the same way as member data. The structure name is given first, followed by a period, and the member function name appears last. Do not forget the argument list even if it is empty.
4. Note references to members *length* and *width* in member function *Area* on line 6 in figure 3. Members of a structure do not need to be qualified by their structure name in member functions.

```
 1.  #include <iostream>
 2.  using namespace std;
 3.  struct Rectangle {
 4.      private:
 5.      double length;
 6.      double width;
 7.      public:
 8.      Rectangle(double l, double w):length(l),width(w){};
 9.      double Area(void){return length*width;}
10.      double Length(void){return length;}
11.      double Width(void){return width;}
12.  };
13.  int main()
14.  {   Rectangle R(12.5,7);
15.      cout << "length=" << R.Length() ;
16.      cout << " ";
17.      cout << "width=" << R.Width() ;
18.      cout << " ";
19.      cout << "Area="  << R.Area();
20.      cout << endl;
21.      return 0;
22.  }
```

Figure 3. Program **p03**

Program **p03** produces**:**
length= 12.5 width=7 Area=87.5

Program **p03** notes:

1. Program **p03** illustrates private and public members of a structure.
2. The directive **private** makes members *length* and *width* available to member functions only.
3. The directive **public** makes members available to any function in the file.
4. Since private members cannot be accessed in function main alternative arrangements must be made to obtain their values. Create public member functions *Length* and *Width* to return the values of corresponding private members.

5. Program **p03** illustrates a *constructor*.
   5.1. A *constructor* is a member function having the same name as the structure name.
   5.2. A *constructor* is a member function having no return type.
   5.3. A *constructor* is called when a structure is allocated. A constructor is called when the structure name is used to declare a variable of that type. Structure *R*, having type *Rectangle*, is declared on line 18 of figure 3. The constructor (a member function) is called on line 14 of figure 3.
   5.4. The values of the arguments of a constructor are used to initialize private member data. A special syntax is given to facilitate this goal. Members of a structure can be initialized between the parameter list and the body of the function. This syntax is only available for constructors. A colon follows the closing parenthesis of the parameter list. Then, member names are listed. Commas separate member names. After each member name parentheses enclose the value to be assigned to the member. Formal parameter *l* is assigned to private member *length* and formal parameter *w* is assigned to member *width* using the special syntax to initialize members in constructor *Rectangle*.

```
 1.  #include <iostream>
 2.  using namespace std;
 3.  class Rectangle {
 4.      double length;
 5.      double width;
 6.  public:
 7.      Rectangle(double l, double w) :length(l), width(w) {}
 8.      double Area(void){return length*width;}
 9.      double Perimeter(void) {return 2*(length + width);}
10.      double Length(void){return length;}
11.      double Width(void) {return width;}
12.  };
13.  int main()
14.  {   Rectangle R(12.5,7);
15.      cout << "length=" << R.Length() << ;
16.      cout << " ";
17.      cout << "width=" << R.Width() ;
18.      cout << " ";
19.      cout << "Area="  << R.Area();
20.      cout << " ";
21.      cout << "Perimeter=" << R.Perimeter() ;
22.      cout << endl;
23.      return 0;
24.  }
```

Figure 4. Program **p04**

Program **p04** produces:
length= 12.5 width=7 Area=87.5 Perimeter=39

Program **p04** notes:

1. Program **p04** illustrates the relationship between a class and a structure.
2. A class is a structure where members are private by default. Public members must be explicitly identified Members of a structure are public by default. Note the reserve word class on line 3 of figure 4.
3. Observe the additional member function Perimeter defined on line 9 and called on line 21.

```
1.  #include <iostream>
2.  using namespace std;
3.  #include "Rectangle05.h"
4.  int main()
5.  {   Rectangle R(12.5,7);
6.      cout << "length=" << R.Length() ;
7.      cout << " ";
8.      cout << " width=" << R.Width() ;
9.      cout << " ";
10.     cout << " Area="  << R.Area();
11.     cout << " ";
12.     cout << " Perimeter=" << R.Perimeter() ;
13.     cout << endl;
14.     return 0;
15. }
```

Figure 5. File **p05.cpp**

```
1.  #ifndef Rectangle05_h
2.  #define Rectangle05_h 1
3.  class Rectangle {
4.      double length;
5.      double width;
6.  public:
7.      Rectangle(double l, double w);
8.      double Area(void);
9.      double Perimeter(void);
10.     double Length(void);
11.     double Width(void);
12. };
13. #endif
```

Figure 6. File **Rectangle05.h**

```
1.  #include "Rectangle05.h"
2.  Rectangle::Rectangle(double l, double w):length(l), width(w) {}
3.  double Rectangle::Perimeter(void){return 2*(length + width);}
4.  double Rectangle::Area(void){return length*width;}
5.  double Rectangle::Length(void){return length;}
6.  double Rectangle::Width(void){return width;}
```

Figure 7. File **Rectangle05.cpp**

```
#-------------------------------------------------------------------------------------------------------------------
# Object files
#-------------------------------------------------------------------------------------------------------------------
obj      =          p05.o Rectangle05.o
#-------------------------------------------------------------------------------------------------------------------
# Create executable file p05
#-------------------------------------------------------------------------------------------------------------------
p05:                ${obj}
                    g++ -o p05 ${obj} -lm
#-------------------------------------------------------------------------------------------------------------------
# Compile file p05.cpp
#-------------------------------------------------------------------------------------------------------------------
p05.o:              p05.cpp Rectangle05.h
                    g++ -c -g p05.cpp
#-------------------------------------------------------------------------------------------------------------------
# Compile file Rectangle05.cpp
#-------------------------------------------------------------------------------------------------------------------
Rectangle05.o:      Rectangle05.cpp Rectangle05.h
                    g++ -c -g Rectangle05.cpp
```

Figure 8. File **p05make**.

Program **p05** produces**:**
length= 12.5 width=7 Area=87.5 Perimeter=39

Program **p05** notes:

1. Program **p05** is created by issuing the Linux command
   $ make –f p05make
   1.1. After entering the foregoing command one can expect LINUX to respond
   g++ -c -g p05.cpp
   g++ -c -g Rectangle05.cpp
   g++ -o p05 p05.o Rectangle05.o -lm
2. Program **p05** illustrates program structure for a class in C++.
3. A class has an interface and an implementation.
   3.1. **Interface**: The interface to a class is the **.h** file in C++. The **.h** file contains the definition of the class. Member data and member functions are defined in the class. Member functions define the valid operations on a variable that has the type given by the class. Member functions, *Length*, *Width*, *Area*, and *Perimeter* define the valid operations on a variable of type *Rectangle*. Member data define the representation of the type. Member data define the set of values that a variable of the type specified by the class can take on. Member data are hidden from the users of the class. Native type double simulates a real number. For example, a variable of type **double** is represented by an IEEE 754 double binary specification. A value that conforms to the IEEE 754 double binary specification has three components, a sign, a characteristic, and a fraction. The programmer is completely unaware of these three components. The operations

defined for a variable of type double are complete. These operations are addition, subtraction, multiplication, division, and the like.

3.2. **Implementation**: A class is implemented in a file having a **.cpp** suffix. The bodies of member functions appear in the **.cpp** file.

3.2.1. **:: - The global resolution operator.** The class name and the global resolution operator must qualify all member function names when the function is defined in the **.cpp** file. For example function *Area*.

double *Rectangle*::*Area***(void){return** *length****** *width***;}**

The class name is *Rectangle* and the global resolution operator is **::**. Note that the name *Area* is qualified using *Rectangle***::**Area.

4. **Separate files**. It is probably best to begin developing your programs in a single file. When the program functions satisfactorily, the program can be separated into separate files. The steps below can be used to split a single file containing one class into three files.

4.1. Make two additional copies of the source file. For example, assume file **p05.cpp**, initially is identical to file **p04.cpp**.

$ cp p05.cpp Rectangle05.cpp
$ cp p05.cpp Rectangle05.h

One of the copies will be edited to become the class interface, the **.h** file.
The copy will be edited to become the class implementation, the **.cpp** file.
Finally, the original file will be edited to become the file that exercises the class.

4.2. **Making the .h file**.

4.2.1. At the top of the file insert the two directives #ifndef …, #define. For example, in file **Rectangle05.h**, insert

```
#ifndef Rectangle05_h
#define Rectangle05_h 1
```

4.2.2. At the bottom of the file insert the directive #endif. For, example, in file **Rectangle05.h** insert.

```
#endif
```

4.2.3. Keep the author identification comment, the copyright notice, C++ include file directives and, most important of all, keep the class definition. For example, in file **Rectangle05.h**, keep the following.

```
#ifndef Rectangle05_h
#define Rectangle05_h 1
//-----------------------------------------------------------------------------------------
//File description comment
//-----------------------------------------------------------------------------------------
//Author identification comment
//-----------------------------------------------------------------------------------------
//Copyright notice
//-----------------------------------------------------------------------------------------
```

```
//C++ include files
//-------------------------------------------------------------------------------------------
#include <iostream>
…
using namespace std;
//-------------------------------------------------------------------------------------------
//class description comment
//-------------------------------------------------------------------------------------------
class Rectangle {
---
};
#endif
```

4.2.4. Remove everything after the semicolon that terminates the class as shown above.

4.2.5. Make all the member functions function prototypes. Replace the bodies of all member functions with a semicolon as shown in figure 6.

5. **Making the .cpp file**.

5.1.1. Remove everything that follows the **};** that terminates the class. Then, remove the **};** also.

5.1.2. Remove everything from the class definition that is not a member function.

5.1.3. Qualify all member function names in the function header with the class name and the global resolution operator. For example, change

double *Area***(void){return** *length*******width***;}**

to

double *Rectangle***::***Area***(void){return** *length*******width***;}**

5.1.4. After the C++ include files but before the member functions insert an include directive that causes the class interface definition to be included. In file **Rectangle05.cpp**,

```
//-------------------------------------------------------------------------------------------
//File description comment
//-------------------------------------------------------------------------------------------
//Author identification comment
//-------------------------------------------------------------------------------------------
//Copyright notice
//-------------------------------------------------------------------------------------------
//C++ include files
//-------------------------------------------------------------------------------------------
#include <iostream>
…
using namespace std;
//-------------------------------------------------------------------------------------------
//Application include files
//-------------------------------------------------------------------------------------------
```

```
#include "Rectangle05.h"
//-----------------------------------------------------------------------------------------------
Rectangle::Rectangle(double l, double w):length(l), width(w) {}
double Rectangle::Perimeter(void){return 2*(length + width);}
double Rectangle::Area(void){return length*width;}
double Rectangle::Length(void){return length;}
double Rectangle::Width(void){return width;}
```

5.1.5. Finally, remove all default parameter initialization expressions. For example, if the constructor had initial values for parameters l and w, they would have to be removed.

change

```
Rectangle::Rectangle(double l=0, double w=0):length(l), width(w) {}
```

to

```
Rectangle::Rectangle(double l, double w):length(l), width(w) {}
```