```
1.  #include <iostream>
2.  #include <iomanip>
3.  using namespace std;
4.  struct Rectangle {
5.      double length;
6.      double width;
7.  };
8.  int main()
9.  {   Rectangle R={12.5,7};
10.     cout << "length=" << R.length;
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 shown in Figure 1.
3. *Rectangle R* is initialized on line 8. Note the initialization syntax.

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

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 declared on line 7 on figure 2.

3. The problem with member functions is finding a syntax that distinguishes member functions from functions declared outside a structure. The global resolution operator, **::**, and the structure name are used to qualify member function names. Member function *Area* is defined on lines 9 – 11 in figure 2.
4. 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.
5. Note references to members *length* and *width* in member function *Area* on line 10 in figure 2. Members of a structure do not need to be qualified by their structure name in member functions.

```
#include <iostream>
#include <iomanip>
using namespace std;
struct Rectangle {
private:
     double length;
     double width;
public:
     Rectangle(double l, double w);
     double Area(void);
     double Length(void);
     double Width(void);
};
Rectangle::Rectangle(double l, double w)
{    length=l; width=w;
}
double Rectangle::Area(void)
{    return length*width;
}
double Rectangle::Length(void)
{    return length;
}
double Rectangle::Width(void)
{    return width;
}
int main()
{    Rectangle R(12.5,7);
     cout << "length=" << R.Length() ;
     cout << " width=" << R.Width() ;
     cout << " Area="  << R.Area();
     cout << endl;
     return 0;
}
```

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. Members do not need to be qualified by the structure in which they are declared to be referenced in a member function. Note members *length* and *width* are not qualified by a reference to a structure of type *Rectangle*.
6. Program **p03** illustrates a *constructor*.
   6.1. A *constructor* is a member function having the same name as the structure name.
   6.2. A *constructor* is a member function having no return type.
   6.3. 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 27 of figure 3. The constructor (a member function) is called on line 27 of figure 3.
   6.4. The values of the arguments of a constructor are used to initialize private member data. Formal parameter *l* is assigned to private member *length* in member function *Rectangle*. Formal parameter *w* is assigned to private member *width* in member function *Rectangle*.

```
#include <iostream>
#include <iomanip>
using namespace std;
class Rectangle {
      double length;
      double width;
public:
      Rectangle(double l, double w);
      double Area(void);
      double Perimeter(void);
      double Length(void);
      double Width(void);
};
Rectangle::Rectangle(double l, double w):length(l), width(w) {}

double Rectangle::Perimeter(void)
{     return 2*length + 2*width;
}
double Rectangle::Area(void)
{     return length*width;
}
double Rectangle::Length(void)
{     return length;
}
double Rectangle::Width(void)
{     return width;
}
int main()
{     Rectangle R(12.5,7);
      cout << "length=" << R.Length() ;
      cout << " width=" << R.Width() ;
      cout << " Area="  << R.Area();
      cout << " Perimeter=" << R.Perimeter() ;
      cout << endl;
      return 0;
}
```

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 every member is private. Public members must be explicitly identified. Members of a structure are public by default. Note the reserve word **class** on line 4 of figure 4.
3. Initialization using a constructor has been extended to native types of C++. Observe that private members *length* and *width* are initialized using standard constructors for type **double** on line 14 of figure 4.
4. Observe the additional member function *Perimeter* declared on line 10, defined on lines 16, 17, and 18, and called on line 33 in figure 4.

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "Rectangle.h"
int main()
{    Rectangle R(12.5,7);
     cout << "length=" << R.Length() ;
     cout << " width=" << R.Width() ;
     cout << " Area="  << R.Area();
     cout << " Perimeter=" << R.Perimeter() ;
     cout << endl;
     return 0;
}
```

Figure 5. File **p05.cpp**

```
#ifndef Rectangle_h
#define Rectangle_h 1
class Rectangle {
     double length;
     double width;
public:
Rectangle(double l, double w);
     double Area(void);
     double Perimeter(void);
     double Length(void);
     double Width(void);
};
#endif
```

Figure 6. File **Rectangle.h**

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

Figure 7. File **Rectangle.cpp**

```
#-----------------------------------------------------------------------------------------------------
# File p05make contains instructions for creating executable file p05 from
# files p05.cpp, Rectangle.cpp, and Rectangle.h
#-----------------------------------------------------------------------------------------------------
obj             =           p05.o Rectangle.o
p05:                        ${obj}
g++ -o p05 ${obj} –lm
p05.o:                      p05.cpp Rectangle.h
g++ -c –g  p05.cpp
Rectangle.o:                Rectangle.cpp Rectangle.h
g++ -c –g  Rectangle.cpp
```

Figure 8. File **p05make**

Program **p05** produces**:**

```
length= 12.5 width=7 Area=87.5 Perimeter=39
```

Program **p05** notes:

1. Program **p05** illustrates program structure for an Abstract data type (ADT). Abstract data types are defined by the programmer and implemented as classes in C++.
2. An ADT has an interface and an implementation.
   2.1. The interface to an ADT 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. 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.

2.1.1. Program **p05** illustrates the Rectangle ADT. **class** *Rectangle* is contained in file **Rectangle.h**.

2.2. The implementation of an ADT is contained in the **.cpp** file in C++. Member functions are implemented in the **.cpp** file. Member functions are *declared* in the **.h** file and *defined* in the **.cpp** file.

2.2.1. Program **p05** illustrates an implementation of a Rectangle. Member functions of class Rectangle are implemented in file **Rectangle.cpp**

3. Program **p05** is composed of four source files **p05.cpp**, **Rectangle.cpp**, **Rectangle.h**, and **p05make**. Executable file **p05** is created from the four source files.

3.1. File **p05.cpp** exercises **class** *Rectangle*.

3.2. File **Rectangle.h** contains the definition of **class** *Rectangle* and serves as the interface for **class** *Rectangle*.

3.3. File **Rectangle.cpp** contains the implementations of member functions of **class** *Rectangle*.

3.4. File **p05make** contains instructions for the Unix/Linux utility ***make***. Instructions in file **p05make** direct the make utility to create executable file **p05**.