Data Structure

Lec 00 Review

Agenda

Functions

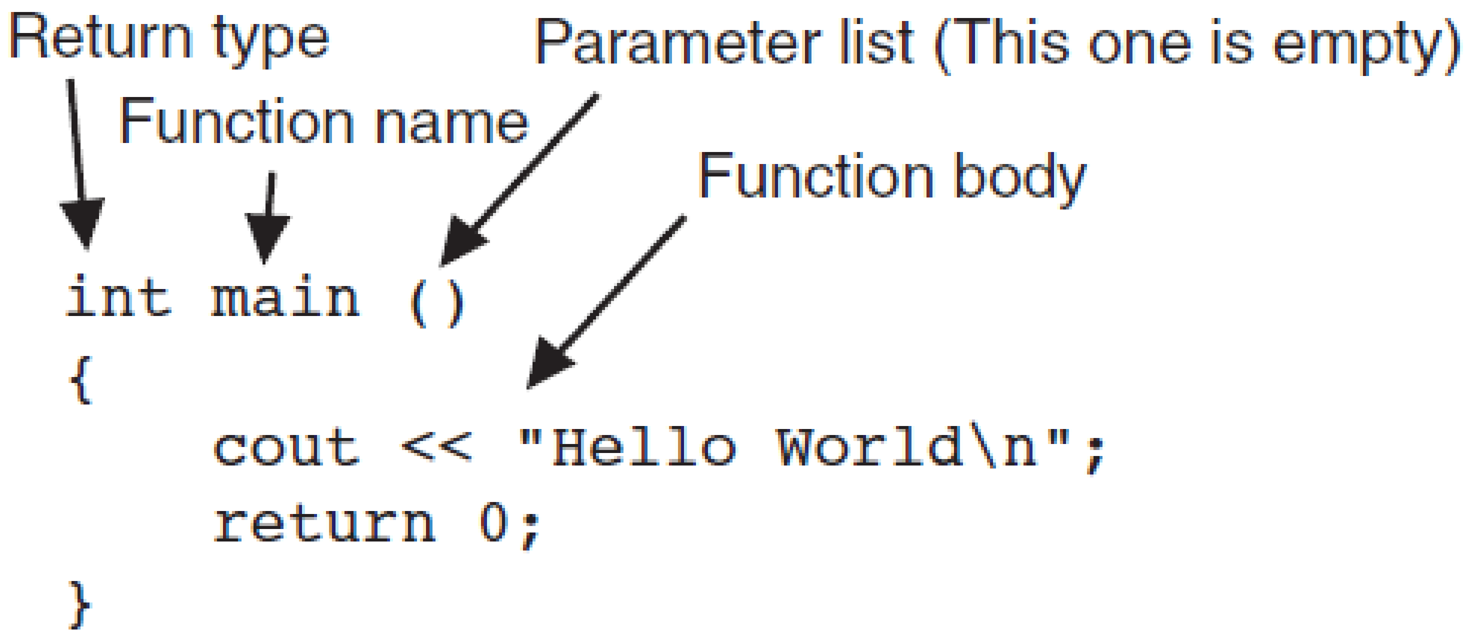
Array

Pointers

Structure

Class

Functions



```
int main ()
{
    cout << "Hello World\n";
    return 0;
}
```

Function Prototypes

A function prototype eliminates the need to place a function definition before all calls to the function.

```
// Function Prototypes
void first();
void second();

int main()
{
     cout << "I am starting in function main.\n";
     first();    // Call function first
     second();   // Call function second
     cout << "Back in function main again.\n";
     return 0;
}
void first()
{
     cout << "I am now inside the function first.\n";
}

void second()
{
     cout << "I am now inside the function second.\n";
}
```

Add Two Numbers

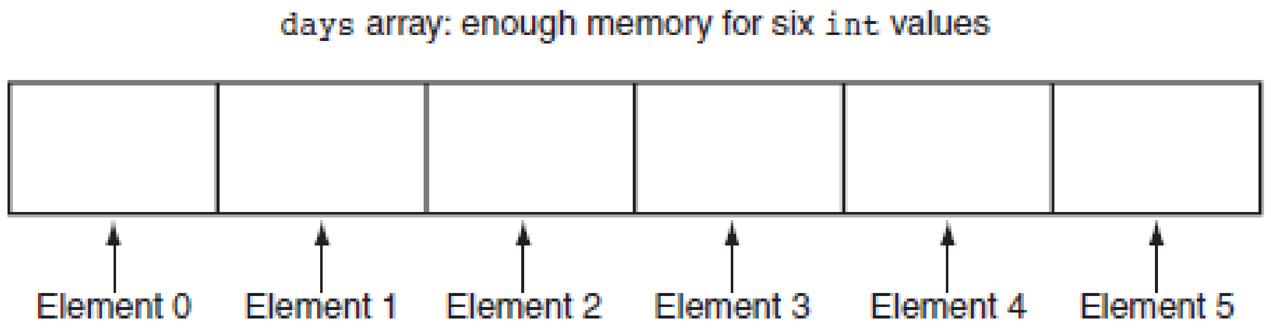
```
int sum = 0;
void print()
{
        cout << "The Sum is = " << sum << endl;
}
int sum_func(int , int );
void main()
{
        int a, b;
        cout << "Enter two numbers : ";
        cin >> a >> b;
        sum = sum_func(a, b);
        print();
}
int sum_func(int x, int y)
{
        int z = x + y;
        return z;
}
```

Arrays

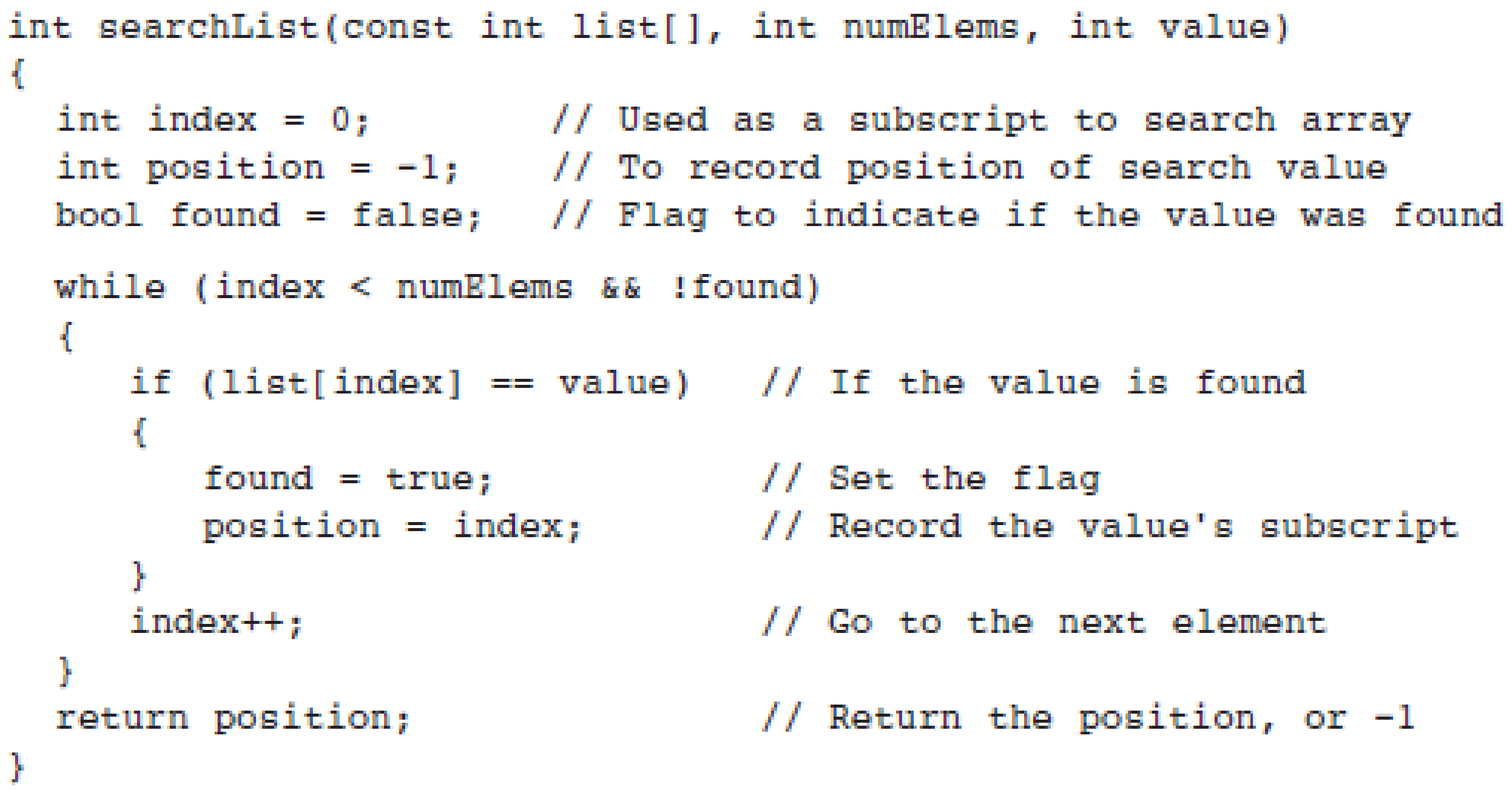
An array allows you to store and work with multiple values of the same data type.

```
int days[6];
```

```
const int NUM_DAYS = 6;
int days[NUM_DAYS];
```

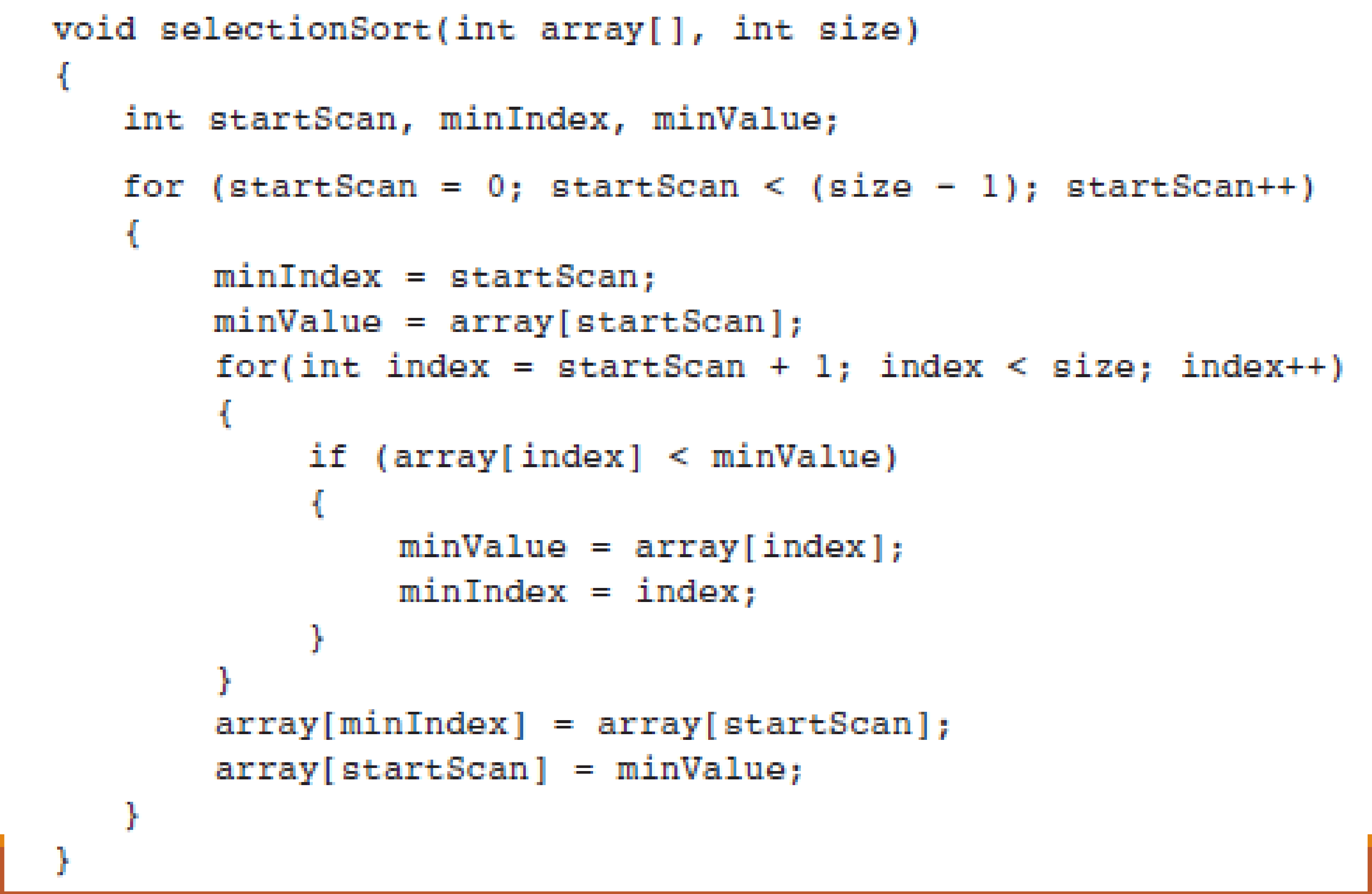


Searching Unsorted Array (Linear Search)



```
int searchList(const int list[], int numElems, int value)
{
   int index = 0;          // Used as a subscript to search array
   int position = -1;      // To record position of search value
   bool found = false;     // Flag to indicate if the value was found

   while (index < numElems && !found)
   {
      if (list[index] == value)   // If the value is found
      {
         found = true;            // Set the flag
         position = index;        // Record the value's subscript
      }
      index++;                    // Go to the next element
   }
   return position;               // Return the position, or -1
}
```

Sorting Arrays (Selection Sort)



```
void selectionSort(int array[], int size)
{
    int startScan, minIndex, minValue;

    for (startScan = 0; startScan < (size - 1); startScan++)
    {
        minIndex = startScan;
        minValue = array[startScan];
        for(int index = startScan + 1; index < size; index++)
        {
            if (array[index] < minValue)
            {
                minValue = array[index];
                minIndex = index;
            }
        }
        array[minIndex] = array[startScan];
        array[startScan] = minValue;
    }
}
```

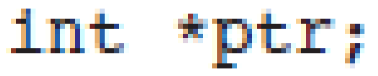
Sum array of integers

```
void arr_input(int ar[], int s)
{
		cout << "Enterthe arr elements\n";
		for (int i = 0; i < s; i++)
		{
				cout << "Enter " << i << "element : ";
				cin >> ar[i];
		}
}
int arr_sum(int ar[], int s)
{
		int z = ar[0];
		for (int i = 1; i < s; i++)
		{
				z += ar[i];
		}
		return z;
}
```

```
int sum = 0;
void print()
{
		cout << "The Sum is = " << sum << endl;
}
int sum_func(int , int );
void arr_input(int[], int);
int arr_sum(int[], int);
void main()
{
		int size;
		cout << "Enter array size : ";
		cin >> size;
		int * arr = NULL;
		arr = new int[size];
		arr_input(arr, size);
		sum = arr_sum(arr, size);
		print();
}
```

Pointers

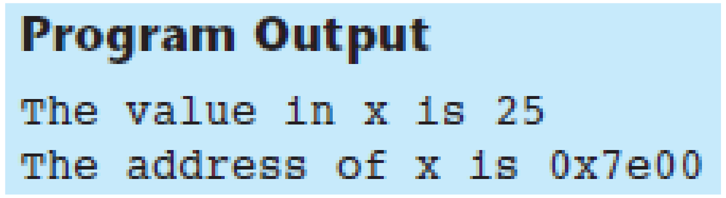
Pointer variables , which are often just called pointers , are designed to hold memory addresses. With pointer variables you can indirectly manipulate data stored in other variables.



```
int *ptr;
```

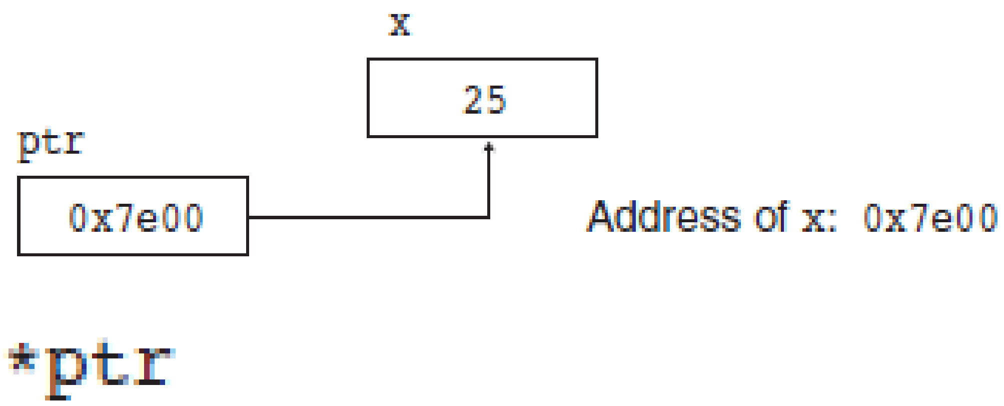
Example

```
#include <iostream>
using namespace std;
int main()
{
    int x = 25;
    int *ptr = NULL;

    ptr = &x;

    cout << "The value in x is " << x << endl;
    cout << "The address of x is " << ptr << endl;

    return 0;
}
```



Program Output

```
The value in x is 25
The address of x is 0x7e00
```



Array and pointer

```
void arr_input(int ar[], int s)
{
        cout << "Enter the arr elements\n";
        for (int i = 0; i < s; i++)
        {
                cout << "Enter " << i << “ element : ";
                cin >> ar[i];
        }
}
```

```
void arr_input_pointer(int * ar, int s)
{
        cout << "Enterthe arr elements\n";
        for (int i = 0; i < s; i++)
        {
                cout << "Enter " << i << “ element : ";
                cin >> *(ar + i);
        }
}
```

Dynamic Memory Allocation

Variables may be created and destroyed while a program is running.

```
int *iptr = NULL;


iptr = new int;


*iptr = 25;


cout << *iptr; // Display the contents of the new variable.
cin >> *iptr; // Let the user input a value.
total += *iptr; // Use the new variable in a computation.
delete iptr;
```

Free Memory

```
void main()
{
        int size;
        cout << "Enter array size : ";
        cin >> size;
        int * arr = NULL;
        arr = new int[size];
        arr_input(arr, size);
        sum = arr_sum(arr, size);
        print();
        delete[] arr;
        arr = NULL;
}
```

Structure

Structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

```
struct type_name {
member_type1    member_name1;
member_type2    member_name2;
member_type3    member_name3;
…
…
} object_names;
```

Example 1

```
struct product {
  int weight;
  double price;
} ;

product apple;
product banana, melon;
```

```
apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price
```

Example 2

```
struct movies_t {
  string title;
  int year;
};

movies_t amovie;
movies_t * pmovie;
pmovie = &amovie;
```

```
pmovie->title
```

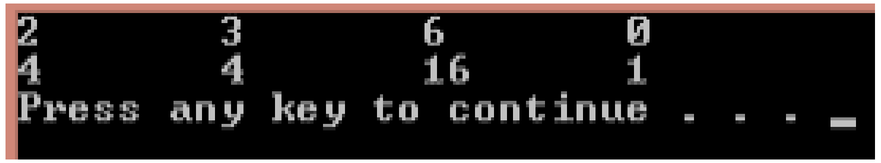
is, for all purposes, equivalent to:

```
(*pmovie).title
```

Example 3 (Rectangle)

```
struct rectangle {
    int len;
    int wid;
    bool isSquare() { return (len == wid);
};
    int area();
};
int rectangle::area()
{
    return len*wid;
}
```

```
void main()
{
   rectangle a, *b = NULL;
   a.len = 2, a.wid = 3;
   cout << a.len << "\t" << a.wid << '\t' << a.area() << '\t' << a.isSquare() << endl;
   b = new rectangle;
   b->len = 4, b->wid = 4;
   cout << b->len << "\t" << b->wid << '\t' << b->area() << '\t' << b->isSquare() << endl;
   delete b;
   b = NULL;
}
```



```
2       3       6       0
4       4       16      1
Press any key to continue . . . _
```

Classes

A class is similar to a structure. It is a data type defined by the programmer, consisting of variables and functions. Here is the general format of a class declaration:

class ClassName

{

- declaration ;
- // ... more declarations
- // may follow...

};

Example 4

```
class Rectangle
{
   int width;
   int length;
}; // Don't forget the semicolon.
```

There is a problem with this class,

Unlike structures, the members of a class are private by default.

Private class members cannot be accessed by programming statements outside the class.

So, no statements outside this Rectangle class can access the width and length members.

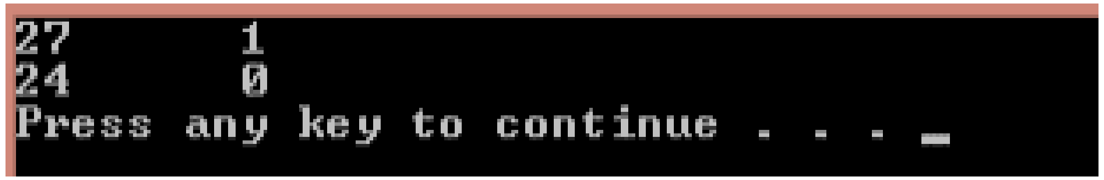
Access Specifiers

```
class ClassName
{
   private:
      // Declarations of private
      // members appear here.
   public:
      // Declarations of public
      // members appear here.
};
```

Example 5

```
class box {
private:
  int length;
  int width;
  int high;
public:
  box(int, int, int);
  int volume();
  bool isEquale() { return (length == width && length == high); };
};
box::box(int l = 3, int w = 3, int h = 3)
{
            length = l;
            width = w;
            high = h;
}
int box::volume()
{
            return (length*width*high);
}
```

```
void main()
{
  box a, *b = NULL;
  b = new box(2, 3, 4);
  cout << a.volume() << '\t' << a.isEquale() << endl;
  cout << b->volume() << '\t' << b->isEquale() << endl;
  delete b;
  b = NULL;
}
```



```
27      1
24      0
Press any key to continue . . . _
```

Example 6 (class Box with struct Rectangle)

```
struct rectangle {
        int len;
        int wid;
        bool isSquare() { return (len == wid); };
        int area();
};
int rectangle::area()
{
        return len*wid;
}
```

```
class boxRect {
private:
        rectangle *r = NULL;
        int high;
public:
        boxRect(int, int, int);
        ~boxRect();
        int volume();
        bool isEquale() { return (r->isSquare() && r->len ==
high); };
};
```

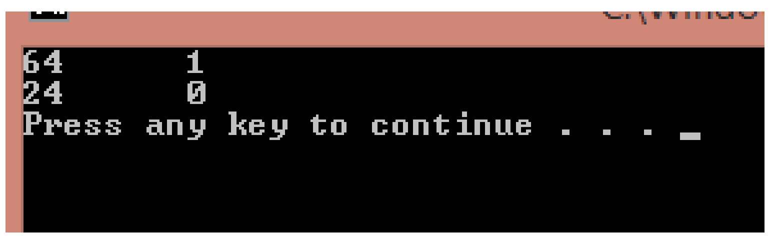
Example 6 (class Box with struct Rectangle)

```
boxRect::boxRect(int l = 3, int w = 3, int h = 3)
{
        r = new rectangle;
        r->len = l;
        r->wid = w;
        high = h;
}
```

```
int boxRect::volume()
{
        return (r->area()*high);
}
boxRect::~boxRect()
{
        delete r;
        r = NULL;
}
```

Example 6 (class Box with struct Rectangle)

```
void main()
{
        boxRect a(4, 4, 4), *b = NULL;
        b = new boxRect(2, 3, 4);
        cout << a.volume() << '\t' << a.isEquale() << endl;
        cout << b->volume() << '\t' << b->isEquale() << endl;
        delete b;
        b = NULL;
}
```



```
64      1
24      0
Press any key to continue . . .
```