CL-217 OBJECT ORIENTED PROGRAMMING LAB

LAB MANUAL 12

INSTRUCTOR: MUHAMMAD HAMZA

SEMESTER SPRING 2020

Static and Automatic Variables:

The variables discussed so far have followed two simple rules:

1. Memory for global variables remains allocated as long as the program executes.

2. Memory for a variable declared within a block is allocated at block entry and deallocated at block exit. For example, memory for the formal parameters and local variables of a function is allocated when the function is called and deallocated when the function exits.

A variable for which memory is allocated at block entry and deallocated at block exit is called an automatic variable. A variable for which memory remains allocated as long as the program executes is called a static variable. Global variables are static variables, and by default, variables declared within a block are automatic variables.

You can declare a static variable within a block by using the reserved word static .

The syntax for declaring a static variable is:

static dataType identifier;

The statement:

static int x;

declares x to be a static variable of type int .

Static variables declared within a block are local to the block, and their scope is the same as that of any other local identifier of that block.

Most compilers initialize static variables to their default values. For example, static int variables are initialized to 0 . However, it is a good practice to initialize static variables yourself, especially if the initial value is not the default value. In this case, static variables are initialized when they are declared.

The statement:

static int x = 0;

declares x to be a static variable of type int and initializes x to 0 , the first time the function is called.

The following program shows how static and automatic variables behave.

```
//Program: Static and automatic variables
#include <iostream>
using namespace std;
void test();

int main()
{
int count;
for (count = 1; count <= 5; count++)
test();
return 0;
}
void test()
{
static int x = 0;
int y = 10;
x = x + 2;
y = y + 1;
cout << "Inside test x = " << x << " and y = "
<< y << endl;
}
```

Sample Run:

```
Inside test x = 2 and y = 11
Inside test x = 4 and y = 11
Inside test x = 6 and y = 11
Inside test x = 8 and y = 11
Inside test x = 10 and y = 11
```

In the function test , x is a static variable initialized to 0 , and y is an automatic variable initialized to 10 . The function main calls the function test five times. Memory for the variable y is allocated every time the function test is called and deallocated when the function exits. Thus, every time the function test is called, it prints the same value for y . However, because x is a static variable, memory for x remains allocated as long as the program executes. The variable x is initialized once to 0 , the first time the function is called. The subsequent calls of the function test use the value x had when the program last left (executed) the function test .

Reference: C++ Programming from problem analysis to problem designing, Eighth Edition, DS Malik.

Static Members of a Class:

We can define class members static using static keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

Let us try the following example to understand the concept of static data members −

Live Demo

```
#include <iostream>

using namespace std;

class Box {
   public:
      static int objectCount;

      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;
```

```
         // Increase every time object is created
         objectCount++;
      }
      double Volume() {
         return length * breadth * height;
      }

   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};

// Initialize static member of class Box
int Box::objectCount = 0;

int main(void) {
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2

   // Print total number of objects.
   cout << "Total objects: " << Box::objectCount << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Constructor called.
Constructor called.
Total objects: 2
```

Static Function Members

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Let us try the following example to understand the concept of static function members –

Live Demo

```
#include <iostream>

using namespace std;

class Box {
   public:
      static int objectCount;

      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;

         // Increase every time object is created
         objectCount++;
      }
      double Volume() {
         return length * breadth * height;
      }
      static int getCount() {
         return objectCount;
      }

   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};

// Initialize static member of class Box
int Box::objectCount = 0;

int main(void) {
   // Print total number of objects before creating object.
   cout << "Inital Stage Count: " << Box::getCount() << endl;
```

```
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2

   // Print total number of objects after creating object.
   cout << "Final Stage Count: " << Box::getCount() << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Inital Stage Count: 0
Constructor called.
Constructor called.
Final Stage Count: 2
```

Reference: https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm

LAB TASK

Q. Write a class named **Enemy**. The class keeps track of all alive **Enemy** objects at the current time. Create the class and figure out how you can achieve the goal. Write a test program to test your class for the required functionality. Create several objects of the class in main and delete them. After each creation and deletion you must print out the current number of alive **Enemy** objects.