Static Memory

Table of Contents

Memory Lifetime

Variable Lifetime:

The period during program execution during which a variable has memory allocated/assigned to it.

Automatic Storage:

Storage is assigned/allocated to automatic variables when execution enters the program block in which the variable definition occurs. Storage is automatically reclaimed/deallocated from automatic variables when execution exits the program block in which the variable definition occurs. Automatic variables are often assigned and reclaimed multiple times during execution.

All **`register`**, local variables and parameters are by default automatic variables. The keyword **`auto`** may be prefixed to a a variable definition to designate its storage.

Static Storage:

Storage is assigned/allocated to automatic variables when program execution begins and storage is not reclaimed/deallocated from static variables until execution completes. (The same lifetime as global variables.)

The keyword **`static`** may be prefixed to a a variable definition to designate its storage. (Extern variables are also considered static variables.)

Static Local Variable

Random Number Generator:

Functions that generate random numbers need to remember the last number generated. A local **`static`** variable is ideal for this purpose.

```
//Linear Congruential Generator
//returns a random integer
//between 0. . .65535
long random(const long InitialSeed = 13;) {
  const long MULITPLIER   = 25173;
  const long INCREMENT    = 13849;
  const long MODULUS      = 65536;

  static long seed = InitialSeed;

  seed = (MULITPLIER * seed + INCREMENT)
         % MODULUS;

  return ( seed );
}
```

The initialization of the seed is executed once only. After the first call to random() the subsequent calls will use the last returned value to determine the next random number.

Static Class Members

Class members may be declared **`static`**. Data members declared as static are "class-wide" data. Only one copy of the static data members exist and is shared by all objects of the class. Static data members are usually used to represent class properties or "states of the class.

Static data members are not global variables. Their scope is restricted to the containing class. They can only be defined and initialized once at the file scope level. They can be accessed when no class objects exist by using the class name and the scope resolution operator or by a static function member. (Static function members may only access other **`static`** members and contain no **`this`** pointer.)

Consider a class whose objects need to know how many other objects of the class are in existence at any given time. The following simple class could be incorporated for such a purpose.

Class Count

```
//ClassCount.h
#ifndef CLASSCOUNT_H
#define CLASSCOUNT _H
class ClassCount {
private:
  static int objects;
public:
  ClassCount();
  static int GetNumObjects() const;
  ~ClassCount();
};
#endif


//ClassCount.cpp
#include “ClassCount.h”

//static definition & initialization
int ClassCount::objects = 0; //file scope

ClassCount::ClassCount() { objects++; }

int ClassCount::GetNumObjects()const {return objects;}

ClassCount::~ClassCount() { objects--; }

// . . . client code outside of class . . .
#include “ClassCount.h”
// . . .
int Count = ClassCount::GetNumObjects();
         // call to static function member when
         // no class objects exist
```