CHAPTER 17

Static

The `static` keyword is used to create class members that exist in only one copy, which belongs to the class itself. These members are shared among all instances of the class. This is different from instance (non-static) members, which are created as new copies for each new object.

Static fields

A static field (class field) cannot be initialized inside the class like an instance field. Instead it must be defined outside of the class declaration. This initialization will only take place once, and the static field will then remain initialized throughout the life of the application.

```
class MyCircle
{
 public:
  double r;         // instance field (one per object)
  static double pi; // static field (only one copy)
};

double MyCircle::pi = 3.14;
```

To access a static member from outside the class, the name of the class is used followed by the scope resolution operator and the static member. This means that there is no need to create an instance of a class in order to access its static members.

```
int main()
{
  double p = MyCircle::pi;
}
```

Static methods

In addition to fields, methods can also be declared as `static`, in which case they can also be called without having to define an instance of the class. However, because a static method is not part of any instance it cannot use instance members. Methods should therefore only be declared `static` if they perform a generic function that is independent of any instance variables. Instance methods on the other hand, in contrast to static methods, can use both static and instance members.

```
class MyCircle
{
 public:
  double r;         // instance variable (one per object)
  static double pi; // static variable (only one copy)

  double getArea() { return pi * r * r; }
  static double newArea(double a) { return pi * a * a; }
};

int main()
{
  double a = MyCircle::newArea(1);
}
```

Static local variables

Local variables inside a function can be declared as `static` to make the function remember the variable. A static local variable is only initialized once when execution first reaches the declaration, and that declaration is then ignored every subsequent time the execution passes through.

```
int myFunc()
{
  static int count = 0; // holds # of calls to function
  count++;
}
```

Static global variables

One last place where the `static` keyword can be applied is to global variables. This will limit the accessibility of the variable to only the current source file, and can therefore be used to help avoid naming conflicts.

```
// Only visible within this source file
static int myGlobal;
```