Articles from Veda Solutions

Static keyword in C

2012-11-30 18:11:36   techveda

Introduction

static keyword has multiple uses in C code and slightly different behavior in each case. For a beginner it might seem like a total anarchy. So, let’s get to the point and understand implications of using static in our code.

static modifier has something to do with memory allocation and linking of your code. C standard refers to static keyword as **storage-class specifiers**. static allows you to specify where to allocate memory for your object and/or how to link it with the rest of the code.

1. Linkages

There are three types of linkage — external, internal and none. Each declared object in your program (i.e. variable or function) has some kind of linkage — usually specified by the circumstances of the declaration. Linkage of an object says how is the object propagated through the whole program. Linkage can be modified by both keywords extern and static.

External linkage

Objects with external linkage can be seen (and accessed) through the whole program across the modules. Anything you declare at file (or global) scope has external linkage by default. All global variables and all functions have external linkage by default.

Internal linkage

Variables and functions with internal linkage are accessible only from one compilation unit — the one they were defined in. Objects with internal linkage are **private** to a single module.

None linkage

None linkage makes the objects completely private to the scope they were defined in. As the name suggests, no linking is done. This applies to all local variables and function parameters, that are only accessible from within the function body, nowhere else.

2. Storage duration

Another area affected by **static** keyword is storage duration, i.e. the lifetime of the object through the program run time. There are two types of storage duration in C — **static** and **automatic**.

Objects with static storage duration are initialized on program startup and remain available through the whole runtime. All objects with external and internal linkage have also static storage duration. Automatic storage duration is default for objects with no linkage. These objects are allocated upon entry to the block in which they were defined and removed when the execution of the block is ended. Storage duration can be modified by the keyword static.

There are two different uses of this keyword in C language. In the first case, static modifies linkage of a variable or function. The ANSI standard states:

"If the declaration of an identifier for an object or a function has file scope and contains the storage-class specifier static, the identifier has internal linkage."

This means if you use the static keyword on a file level (i.e. not in a function), it will change the object's linkage to internal, making it private only for the file or more precisely, compilation unit.

```
/* This is file scope */

int one; /* External linkage. */
static int two; /* Internal linkage. */

/* External linkage. */
int f_one()
{
    return one;
}

/* Internal linkage. */
static void f_two()
{
    two = 2;
}

int main(void)
{
    int three = 0; /* No linkage. */

    one = 1;
    f_two();

    three = f_one() + two;

    return 0;
}
```

The `variable two)` and `function(f_two)` will have internal linkage and won't be visible from any other module.

The other use of static keyword in C is to specify storage duration. The keyword can be used to change automatic storage duration to static. A static variable inside a function is allocated only once (at program startup) and therefore it keeps its value between invocations.

```
#include <stdio.h>

void f1()
{
    int a = 10;
    static int sa = 10;

    a += 5;
    sa += 5;

    printf("a = %d, sa = %dn", a, sa);
}

int main()
{
    int i;

    for (i = 0; i < 10; ++i)
        f1();
}
```

The output will look like this:

```
root@ubuntu:/home/raghu/part1# ./a.out
a = 15, sa = 15
a = 15, sa = 20
a = 15, sa = 25
a = 15, sa = 30
a = 15, sa = 35
a = 15, sa = 40
a = 15, sa = 45
a = 15, sa = 50
a = 15, sa = 55
a = 15, sa = 60
```

Extern keyword

The extern keyword denotes, that "this identifier is **declared** here, but is **defined** elsewhere". In other words, you tell the compiler that some variable will be available, but its memory is allocated somewhere else. The thing is, where? Let's have a look at the difference between **declaration** and **definition** of some object first. By declaring a variable, you say what type the variable is and what name it goes by later in your program. For instance you can do the following:

```
extern int i; /* Declaration. */
extern int i; /* Another declaration. */
```

The variable virtually doesn't exist until you define it (i.e. allocate memory for it). The definition of a variable looks like this:

```
int i = 0; /* Definition. */
```

You can put as many declaration as you want into your program, but only one definition within one scope. Here is an example that comes from the C standard:

```
/*  definition, external linkage */
int i1 = 1;
/*  definition, internal linkage */
static int i2 = 2;
/*  tentative definition, external linkage */
int i3;

/*  valid tentative definition, refers to previous */
int i1;
/*  valid tenative definition, refers to previous */
static int i2;
/*  valid tentative definition, refers to previous */
int i3 = 3;

/* refers to previous, whose linkage is external */
extern int i1;
/* refers to previous, whose linkage is internal */
extern int i2;
/* refers to previous, whose linkage is external */
extern int i4;

int main(void) { return 0; }
```

Summary

Remember that `static` — the storage-class specifier and **static storage duration** are two different things. Storage duration is a attribute of objects that in some cases can be modified by `static`, but the keyword has multiple uses.

Also the `extern` keyword and **external linkage** represent two different areas of interest. External linkage is an object attribute saying that it can be accessed from anywhere in the program. The keyword on the other hand denotes, that the object declared is not defined here, but someplace else.

References

More on extern

-
-
-
-
- More
-