Scope and Life Time of Variables

Gil Ben-Artzi

Scope and Life Time of Variables

- The scope of a variable is the part of the program in which the variable is visible (i.e. may be used).

- The life time of a variable is the time between the ‘birth’ of the variable on the computers memory and the time it ‘dies’.

- There are two broad classifications of scope:
  - Global scope : global.
  - Block (local) scope : automatic local, static local.
- The table in the next page describes the attributes of global and automatic local variables. The two static types will be discussed later.

Global vs. Automatic Local – cont'd

	Global	Automatic Local
declaration	outside any function	in a block { }
initialization	Unless specified otherwise, automatically initialized to 0	No automatic initialization. If specifically initialized - it occurs at each 'birth'
scope	functions beneath it, in the same module	its block
'birth'	once, before **`main()`**	each time the block is entered
'death'	once, after **`main()`** ends	each time the block is exited
address	on the data area	on the stack area

Scope Diagram

```
1   int num = 42;
2   void func(void)
3   {
4       int other_num = 10;
5       printf ("%d %d", num, other_num);
6   }
7   void main()
8   {
9       int num = 1, other_num = 2;
10      printf ("%d %d", num, other_num);
11      {
12         int other_num = 3;
13         printf ("%d %d", num, other_num);
14      }
15      printf ("%d %d", num, other_num);
16      func();
17  }
```

- When referencing a variable, the compiler searches for such a variable name within the immediate scope.
- If not found, then the next “higher” scope is searched.

Example: scope.c

```
1   /* scope.c
2   This program demonstrates the scope of variables */
3
4   #include <stdio.h>
5
6   void func(void);     /* function prototype */
7
8   /* global variables, defined outside any function. All the
9   functions in this file that are written below them can see them */
10  int Tom;
11  int Jerry;
12
13  void main(void)
14  {
15       int main_local = 100; /* local variable in main */
16
17       Tom = Jerry = main_local = 1;  /*  change the two
18                              globals and the local */
19
```

Example: scope.c – cont'd

```
20      printf("In main, before calling func(): Tom=%d,"
21      "Jerry=%d, main_local=%d\n",Tom, Jerry, main_local);
22      func();     /* call the function func() */
23
24      printf("In main, after calling func():  Tom=%d, "
25      "Jerry=%d, main_local=%d\n",Tom, Jerry, main_local);
26  }
27
28  void func(void)
29  {
30      int i; /* this variable (i) is local in function func(). */
31      int Jerry; /* local to the function func().When we refer to
32      Jerry, the compiler finds it in the function's scope, and
33      doesn't look any further. This function can never see the
34      global Jerry */
```

Example: scope.c – cont'd

```
35      Tom = Jerry = i = 1000; /* this increases the global
36                                 variable Tom */
37
38      printf("In func(), before block : Tom=%d, "
39      "Jerry=%d,i=%d\n", Tom, Jerry, i);
40
41      { /* The block starts here. */
42      int Jerry = 2000; /* local to the block. This block can't
43      see any Jerry that is out of this block */
44      Tom = Jerry = i = 2000;
45      printf("In the block : Tom = %d, Jerry = %d, i = "
46              "%d\n", Tom, Jerry, i);
47      } /* The block ends here */
48
49      printf("In func() after block : Tom=%d, Jerry=%d,"
50      " i=%d\n", Tom, Jerry, i);
51  }
```

Example: Exercise

```
1   /* scope2.c -
2   Fill in the blank spaces. If the line compiles, write the
3   output.  If it doesn't, write the reason. */
4
5   #include <stdio.h>
6
7   int  global = 100;
8   void func();  /* prototype */
9
10  void main(void)
11  {
12      int  main_scope = 10, i = 20;
13
14      printf("%d/n", j);   /* ______ */
15      printf("%d %d %d", main_scope, i, global);
16      /* _____ */
17      func();
18      printf("%d", j);   /* ______ */
```

Example: Exercise – cont'd

```
19      printf("%d %d %d\n", main_scope, i, global);
20      /* _____ */
21      printf("%d\n", other_global);
22  }
23  int other_global = 200;
24
25  void func()
26  {
27      int i = 3, j = 4;
28      {
29              int i = 5;
30
31              printf("%d %d %d\n", i, j, other_global);
32             /* ______ */
33      }
34      printf("%d %d %d %d\n", i, j, global, other_global);
35      /* ______ */
36      printf("%d", main_scope);  /* ______ */
37  }
```

Global Variables – Bad News

- Global variables are not healthy for your program,
  try to avoid them.
- Global variables are ‘exposed’. It is hard to predict the contents of a global variable, since many functions can change it.
- It captures memory (on the data) throughout the lifetime of the program.
- The behavior of a function should be predicted, by the arguments that it gets. If the functions behavior also depends on a global variable, it is not as independent as it should be.

Static Local Variables

- Sometimes it is desirable for a variable in a function to maintain its value from one function call to another.
- In other words, the variable should be born once and die once, regardless of the number of times the function is called.
- A static local variable behaves this way.
- The characteristics of a static local variable are the same as those of the global, except from the scope, which is like automatic local.

- For example :

```
void func(void)
{
      static int num = 3;
       /* What happens without the initialization ? */
}
```

Storage Classes

- There are two storage classes :
  - automatic
  - static.

- The automatic local variables belong to the automatic storage class.

- The static local variables, global variables and static global variables (which will be discussed soon) belong to the static storage class.

- The characteristics of the automatic storage class are those described in the table on page 25 - attributes of the automatic local variables.

- The characteristics of the static storage class are those described in the table on page 25 - attributes of the global variables.

Example: static-vs-local1.c

```
1   /* static-vs-local1.c
2   This program illustrates the difference between:
3   static local and automatic local variables */
4
5   #include <stdio.h>
6
7   void func(void); /* function prototype */
8
9   void main(void)
10  {
11      int i;
12
13      for (i = 1; i <= 5; i++) /* call func() five times */
14              func();
15      printf("\n");
16  }
```

Example: static-vs-local1.c – cont'd

```
17 void func(void)
18 {
19      int aut = 0; /* local variable. Automatic
20                           storage class by default.Created each
21                           time this block is entered, destroyed
22                           each time this block ends */
23      static int stat = 0; /* (initialized to 0 any way…).
24                                Created before main(), destroyed
25                                after the program ends */
26      aut++;
27      stat++;
28      printf("automatic = %d      stat = %d\n", aut,stat);
29 }
30

        aut = 1      stat = 1
        aut = 1      stat = 2
        aut = 1      stat = 3
        aut = 1      stat = 4
        aut = 1      stat = 5
```

Example: static-vs-local2.c

```
1   /* static-vs-local2.c
2   This program illustrates the difference between:
3   static local and automatic local variables */
4
5   #include <stdio.h>
6
7   void func(void); /* function prototype */
8
9   void main(void)
10  {
11      int i;
12
13      for (i = 1; i <= 5; i++) /* call func() five times */
14              func();
15      printf("\n");
16  }
```

Example: static-vs-local2.c – cont'd

```
17 void func(void)
18 {
19      int aut = 0;            /* local variable. Automatic storage
20                              class by default. Created each time
21                              this block is entered, destroyed
22                              each time this block ends */
23      static int stat;    /* (initialized to 0 any way…).Created
24                              before main(), destroyed after the
25                              program ends */
26      stat = 0;/*This line didn't appear in the previous program */
27      aut++;
28      stat++;
29      printf("automatic = %d      stat = %d\n", aut,stat);
30 }
31
        aut = 1      stat = 1
        aut = 1      stat = 1
        aut = 1      stat = 1
        aut = 1      stat = 1
        aut = 1      stat = 1
```

Summary

- There are 2 storage classes :
  - **Static**
    - static local.
    - global (may be called – extern global).
    - static global.
  - **Automatic**
    - automatic local.
- The arguments received by a function are automatic local variables of that function.
- The keyword **`extern`** help sharing global variables or functions between different modules of the program.
- The keyword **`static`** makes a global variable or a function private to a certain module.