Often, programmers confuse the scope, lifetime and visibility of variables. So I’ll cover these three important concepts in this month’s column.

Whenever you declare a variable, you determine its scope, lifetime and visibility. These three are important concepts associated with any variable declared in C. Understanding the difference between them, and how they are related to each other, will help avoid mistakes in writing code.

**Scope**

Scope is defined as the area in which the declared variable is ‘available’. There are five scopes in C: program, file, function, block, and prototype. Let us examine a dummy program to understand the difference (the comments indicate the scope of the specific variable):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void foo() {}  // "foo" has program scope  static void bar() {      // "bar" has file scope      printf("hello world");      int i;      // "i" has block scope  }  void baz(int j);  // "j" has prototype scope  print:  // "print" has function scope |

The foo function has program scope. All non-static functions have program scope, and they can be called from anywhere in the program. Of course, to make such a call, the function needs to be first declared using extern, before being called, but the point is that it is available throughout the program.

The function bar has file scope — it can be called from only within the file in which it is declared. It cannot be called from other files, unlike foo, which could be called after providing the external declaration of foo.

The label print has function scope. Remember that labels are used as a target for jumps using goto in C. There can be only one print label inside a function, and you can write a goto print statement anywhere in the function, even before the label appears in the function. Only labels can have function scope in C.

The variable i has block scope, though declared at the same level/block as print. Why is that so? The answer is, we can define another variable with the same name i inside another block within the bar function, whereas it is not possible for print, since it is a label.

The variable j has prototype scope: you cannot declare any other parameter with the same name j in the function baz. Note that the scope of j ends with the prototype declaration: you can define the function baz with the first argument with any name other than j.

**Lifetime**

The lifetime of a variable is the period of time in which the variable is allocated a space (i.e., the period of time for which it “lives”). There are three lifetimes in C: static, automatic and dynamic. Let us look at an example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int foo() {      static int count = 0;      // "count" has static lifetime      int \* counter = malloc(sizeof(int));      // "counter" has automatic lifetime      free(counter);      // malloc’ed memory has dynamic lifetime  } |

In this code, the variable count has a static lifetime, i.e., its lifetime is that of the program. The variable counter has an automatic lifetime — its life is till the function returns; it points to a heap-allocated memory block — its life remains till it is explicitly deleted by the program, which is not predictable, and hence it has a dynamic lifetime.

**Visibility**

Visibility is the “accessibility” of the variable declared. It is the result of hiding a variable in outer scopes. Here is a dummy example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int i;  // the "i" variable is accessible/visible here  void foo() {      int i;      // the outer "i" variable      // is not accessible/visible here      {          int i;          // two "i" variables at outer scopes          // are not accessible/visible here      }      // the "i" in this block is accessible/visible      // here and it still hides the outer "i"  }  // the outermost "i" variable  //is accessible/visible here |

**Summery of differences**

As you can see, scope, lifetime and visibility are related to each other, but are distinct. Scope is about the ‘availability’ of the declared variable: within the same scope, it is not possible to declare/define two variables of the same type with the same name. Lifetime is about the duration in which the variable is ‘alive’: it determines how long the named or unnamed variable has memory allocated to it.

Visibility is about the ‘accessibility’ of the declared variables: it arises because of the possibility of variables in outer scope having the same name as the ones in inner scopes, resulting in ‘hiding’.