Code: analysis, bugs, and security

supported by Bitdefender

Linking and loading

Marius Minea

marius@cs.upt.ro

18 October 2017

Identifiers in C: scope, lifetime, linkage

Scope of identifiers: where is identifier *visible* ?

block scope: from declaration to end of enclosing }

file scope: if declared outside any block

also: *function prototype* scope (ID in function header)

function scope (`goto` labels: can't jump out)

if redeclared, *outer* scope *hidden* while *inner* scope in effect

Linkage: how do same names in different scopes/files link ?
do they refer to the same object ?

external: same in all *translation units* (files) making up program
default for functions and file scope identifiers;
explicit with `extern` declaration

internal: same within one translation unit; if declared `static`

none: each declaration denotes distinct object (for block scope)

Linkage and `static`

Identifiers declared with **`static`** keyword have *internal linkage*
(are not linked to objects with same name in other files)

Storage duration if declared **`static`** is lifetime of program.

`static` in function: local scope but preserves value between calls
initialization done only once, at start of lifetime

```
#include <stdio.h>
int counter(void) {
  static int cnt = 0;
  return cnt++;
}
int main(void) {
  printf("counter is %d\n", counter()); // 0
  printf("counter is %d\n", counter()); // 1
  return 0;
}
```

Storage duration of objects (variables)

automatic, for variables declared with block scope
  lifetime: from block entry to exit; re-initialized every time

static: lifetime is program execution; initialized once

allocated: with `malloc`

thread: for `_Thread_local` objects (since C11)

Declarations and definitions

An identifier can be *declared* multiple times, only *defined once*

A declaration with initializer is a definition.

A file scope declaration with no initializer and no storage class specifier or with `static` is a *tentative definition*
several tentative definitions for same object must match
become definition by end of translation unit

How to use in practice

functions: define in one file, declare in all others
variables: define in one file, declare `extern` in all others

Can put declarations in a *header file*, and include where needed