CS 240 Programming in C

Block, Scope, extern and sscanf

Sep 9, 2019

the GROUR folder

1, login on server
2. cd cs240
3. ln -s /courses/cs240/f19/haoyu/GROUP group

All the codes grading can be found in this folder.

Block and Scope

- Block: A section of code that is grouped together
  - In C, blocks are delimited by curly braces

    `{ [block statements] }`
  - or the parenthesis of for loop

```
for (int i=0 ; i<10; i++);
for (int i=0 ; i<10; i++){\textbf{}int j = 0; j = 1};
```

- Variable: A name used to refer to some location in memory that holds a value we want to work with
- Scope: the area of a program where a variable can be referenced
  - For each different entity that an identifier designates, the identifier is visible (i.e., can be used) only within a region of program text called its *scope*

LoopBlock Variables

- They are local to the block where they are defined.
- They come and go with the loop's invocation and exit.
- If the variables used inside a loop were not declared inside this loop, they will retain their value after loop terminated.
- See demos 1 and 2.

Internal Variables (Local Automatic)

- Arguments and variables defined inside functions are internal
- They are local to the block where they are defined
- Internal variables come into existence when the function is entered and disappear when it is left
- These variables are said to be *automatic*
- Scope: The block where the variable was declared
- Initialized to: Undefined (i.e., garbage) value unless explicitly initialized in the source code
- Initialization happens: Each time the function or block is entered
- The parameters of a function are, in effect, local variables

External Variables

- The adjective "external" is used in contrast to "internal", which describes the arguments and variables defined inside functions.
- External variables are defined outside of any function, and are thus available to many functions.
- Functions themselves are always external, because C does not allow functions to be defined inside other functions.
- External variables are globally accessible, but C can also define static external variables and functions that are visible only within a single source file, we will see later.

External Variables

Question:

Must external variables be defined by the "extern" keyword ?

External Variables and the "extern" key word

- No.
- An external variable is just a variable being defined outside any functions. See demo 4, 5.
- The "extern" keyword is used for searching the global variable reference somewhere else. It means no variable definition here.
  - An "extern" variable inside a function means searching for this variable reference outside this function but within this source file. demo 6.
  - An "extern" variable outside a function means searching for this variable reference from the global variables in this and other provided source code. demo 7&8.

External Variables

Advantages:

- If a large number of variables must be shared among functions, external variables are more convenient and efficient than long argument lists.
- External variables also retain their values after the exit of a function call, since no function owns it solely.
- External (global) variables are favored in high performance computing. They allow additional optimization by compilers.

Disadvantages:

- It is problematic for decoupling a program structure, which makes a big software into less dependent parts such that it is easy for maintaining and testing etc.
- If their value gets corrupted, hard to trace the reason. They make functions dependent on their external environment
- In fact, software architecture/design standards often prohibit use of external variables
- see demo 9&10

External Variables (External Static)

- External variables can be accessed by any function in the program.
- what if we want to limits its scope ?
- The static declaration, applied to an external variable or function, limits the scope of that object to the source file being compiled.

Example: External Static

```
#include <stdlib.h>
double drand48(void);
void srand48(long int seedval);
```

- The pseudorandom number generator `drand48()` is a family of functions
- They keep an external static variable X as the seed of the generators
- The contants are a $=$ 0x5DEECE66D, c $=$ 0xB, and m $= 2^{48}$
- The calculation is as follows

$$X_{i+1} = (aX_i + c) \mod m$$

- You must call `srand48()` to initialize the seed

Internal Variables (Local Static)

- Local alternative to automatic
- Static variables declared inside a function are preserved in memory
- Scope: The block where the variable was declared
- Initialized to 0
  - Also can be explicitly initialized otherwise, in which case the initializer must be a constant expression
- Initialization happens: If initialized, it is done once, before the program starts execution
- Function may behave differently when it is called with different values preserved in local static variables
- Makes it harder to test a function because you need to test with all possible values of local static variables

Example: Internal Variables, Local Static

```
unsigned int keepCnt(unsigned int boolean)
{
  static unsigned int count = 0;

  if (boolean)
    count++;
  return count;
}
```

- The parameter `boolean` is local automatic
- The variable `count` is local static

The `register` Variables

- A register declaration advises the compiler that this variable will be heavily used
- We want it placed in a machine register, but the compiler is free to ignore this suggestion if it needs registers
- Can only be applied to automatic variables

The `register` Variables

- The register declaration can only be applied to automatic variables and to the formal parameters of a function.

```
f(register unsigned m, register long n)
{
    register int i;
    ...
}
```

- In practice, there are restrictions on register variables, reflecting the realities of underlying hardware.
- And it is not possible to take the address of a register variable regardless of whether the variable is actually placed in a register.
- The specific restrictions on number and types of register variables vary from machine to machine.

Availability

- Local automatic: within the block where it is created
- Local static: within the block where it is created
- External: to all functions within the program
- External static: to functions within only the file where it is defined

Duration

- Local automatic: Comes into existence between the braces and disappears once return is performed
- Local static: Perserved in memory
- External: Perserved like local static
- External static: Perserved like local static

Initialization

- In the absence of explicit initialization, external and static variables are guaranteed to be initialized to zero; automatic and register variables have undefined (i.e., garbage) initial values.
- For external and static variables, the initializer must be a constant expression; the initialization is done once, conceptionally before the program begins execution.
- For automatic and register variables, the initializer is not restricted to being a constant: it may be any expression involving previously defined values, even function calls.
- In effect, initialization of automatic variables are just shorthand for assignment statements.

fgets and sscanf

- See code demo 21, 22. And textbook.