601.220 Intermediate Programming

Lifetime and scope of variables

Outline

- Lifetime and scope of variables

Variable lifetime and scope

- *lifetime* of a variable: the period of time during which the value exists in memory

- *scope* of a variable: the region of code in which the variable name is usable

Types of variables in C

- Local ("stack") variables
  - Alive from point of declaration until end of block in which declared
  - Automatically created when enter that function, and destroyed when that function ends (more precisely - the curly brackets)
  - Lifetime and scope are similar, but not the same. . .

Lifetime and scope are related, but. . .

- Local variables go out of scope during calls to other functions

```c
// mainVarOutOfScope.c:
#include <stdio.h>
void outputHello() {
    printf("Hello!\n");   // Here, a is not in scope
}
int main() {
    int a = 12;
    outputHello();
    printf("%d", a);
    return 0;
}

$ gcc -std=c99 -Wall -Wextra -pedantic mainVarOutOfScope.c
$ ./a.out
Hello!
12
```

Lifetime and scope are related, but. . .

- A variable in scope may be temporarily "hidden" when an inner scope declares a variable with the same name

```c
// nestedScope.c:
#include <stdio.h>
int main() {
    int i = 12;                          // I'm a variable named i
    printf("i before loop: %d   ", i);
    for (int i = 0; i < 1; i++) {        // I'm a NEW variable named i
        printf("i inside loop: %d   ", i);
    }
    printf("i after loop: %d\n", i);     // I'm the old i
    return 0;
}
$ gcc -std=c99 -Wall -Wextra -pedantic nestedScope.c
$ ./a.out
i before loop: 12   i inside loop: 0   i after loop: 12
```

Types of variables in C, continued

- Static variables (declared using `static` keyword)
    - Have similar scope to local variables, but longer lifetime
    - Automatically created *and initialized to zero(!)*, but not destroyed at end of block of code. Persist, so that next time same block is executed, they come back into scope with the same value they had before

Local variables

```c
// localVar.c:
#include <stdio.h>
int addInt (int x, int y) {
  int result = 0;      // I'm a local variable
  result += x + y;     // Add (x+y) to existing result, which has value 0
  return result;
}
int main() {
    printf("addInt(3,4) returns %d\n", addInt(3,4));
    printf("addInt(2,1) returns %d\n", addInt(2,1));
    return 0;
}

$ gcc -std=c99 -Wall -Wextra -pedantic localVar.c
$ ./a.out
addInt(3,4) returns 7
addInt(2,1) returns 3
```

Static variables

```c
// staticVar.c:
#include <stdio.h>
int addInt (int x, int y) {
  static int result;  // I'm a static variable
  result += x + y;    // Add (x+y) to existing result
  return result;
}
int main() {
    printf("addInt(3,4) returns %d\n", addInt(3,4));
    printf("addInt(2,1) returns %d\n", addInt(2,1));
    return 0;
}

$ gcc -std=c99 -Wall -Wextra -pedantic staticVar.c
$ ./a.out
addInt(3,4) returns 7
addInt(2,1) returns 10
```

Types of variables in C, continued

- Global variables (declared outside any function)
  - Automatically created at program start *and initialized to zero(!)*
  - Have scope from point of declaration until end of program
  - Can be accessed by any function
  - Can even be accessed by functions in other files (after brought into scope using extern <type> <name> in that file)

Global variables

```c
// globalVar.c:
#include <stdio.h>

int result;  // I'm a global variable, in scope all over!

void addX (int x) {
    result += x;
}
void multiplyByX (int x) {
    result *= x;
}
int main() {
    addX(5);
    multiplyByX(10);
    printf("result equals %d\n", result);
    return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic globalVar.c
$ ./a.out
result equals 50
```

Beware of global variables

- Usage of global variables is generally discouraged
  - debugging is harder; difficult to track which function might have changed a global variable's value (since it could be any function!)
  - global variables cross boundaries between program modules, undoing benefits of modular code
    - readability
    - testability
    - reusability
  - usually, values should be conveyed via parameter passing and return values

Where do different types of variables get stored?

- Local variables live in a region of memory known as the stack
  - Stack frames are added/removed as functions get called and then return
- Both static and global variables live in a region of memory known as the data segment
  - The data segment is allocated when program begins, freed when program exits
- Dynamically-allocated memory lives in a third region of memory, called the heap
  - User is responsible for allocating and freeing memory in the heap

Zoom poll!

What output is printed by the program below?

```c
#include <stdio.h>

void m() {
    static int x = 5;
    x++;
    printf("%d ", x);
}

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

A. 6 7

B. 6 6

C. 5 6

D. 5 5

E. Code does not compile and/or produce any ouput