Variable Scope

Miss your loved ones?
YOU DON'T HAVE TO.
RJX-21 LASER SCOPE

Scope

The places in your code that can read and/or write a variable.

- Scope starts at the location where you declare the variable
  - There may be holes in the scope!
- Scope ends at the end of the block in which the declare occurs
  - Usually, the function in which the variable was declared

Simple Example

Block in which j is declared

```
#include <stdio.h>
int main(int argc, char **argv) {
        int j;
        for(j=0; j<argc; j++) {
                printf("Argument %d = %s\n",j,argv[j]);
        }
        return 0;
}
```

Scope of j

“Block” doesn’t have to be a function!

- You can define a variable inside a sub-block of a function

Internal Example

Block in which k is declared

```
#include <stdio.h>
int main(int argc, char **argv) {
        int j;
        for(j=0; j<argc; j++) {
                int k=j+1;
                printf(“Argument %d = %s\n”,k,argv[j]);
        }
        return 0;
}
```

Scope of k

Global Variable

- Declared outside of a block
- Scope is from declaration to the end of the C file!

Global Example

Block in which nc is declared

```
#include <stdio.h>
int nc=0;
int myfunc(int n) { nc++; return n; }
int main(int argc, char **argv) {
        int j;
        for(j=0; j<argc; j++) myFunc(2);
        printf("myfunc called %d times\n",nc);
        return 0;
}
```

Scope of nc

Global Variable Pros & Cons

Advantages

- Simple and intuitive
- Enables functions to communicate data with each other
- Remembers between function calls as well as within function calls

Disadvantages

- Increases the “outside” information a function needs to be aware of (binding)
- Prevents re-use of functions
- Remembers between function calls as well as within function calls

Holes in Scopes

- If the same variable is declared inside a sub-block, the internal declaration temporarily replaces the external definition!

Example Scope Hole

```
{ int i; int j=7;
  for (i=0; i<3; i++) {
      int j=i+1;
      printf(“j=%d\n”,j);
  }
  printf(“j=%d\n”);
}
```

Scope of outside j

Scope of inside j (Hole for outside j)

```
j=1
j=2
j=3
j=7
```

Variable Class

- Automatic
  - Created/Initialized on entry to block
  - Destroyed on exit from block
- Static
  - Created/Initialized when program starts
  - Destroyed when program ends

Default Class

- Function/Block Variables are automatic
    - Created/Initialized on entry to that function/block
    - Deleted when that function/block ends
- Global Variables are Static
    - Created/Initialized when the program starts
    - Deleted when the program ends
    - “Automatic” has no meaning!

Local Static Variables

- We can specify “static” keyword in a declaration
- Implies variable is created when program starts, deleted when program ends
- Does NOT mean that the scope is global!!!!
  - Scope is still within that function

Example Local Static

```
char * flipflop() {
        static int flip=1;
        if (flip) { flip=0; return “flip”; }
        else { flip=1; return “flop”; }
}
for(i=0;i<8;i++) printf(“%s “,flipflop());
```

```
flip flop flip flop flip flop flip flop
```

BAD FORM : Pseudo-Globals

- It is legal in C to nest a function inside another function
- This allows the variables in the outside function to be visible (in scope) for the inside function
- C Coders frown on this practice!
  - Nested functions have other complications
  - Nested functions cannot be re-used
  - It’s ugly and confusing

Example Nested Functions

```
int main(int argc, char **argv) {
	char firstArgLetter(int i) {
		return argv[i][0];
	}
	int j;
	for(j=0;j<argc;j++) printf(“Arg start: %c\n”,
		firstArgLetter(j));
	return 0;
}
```

Resources

- Programming in C, Chapter 7
- Wikipedia Variable https://en.wikipedia.org/wiki/Variable_(computer_science)
- Scope Tutorial http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm