

C: an introduction

Scope

Key Point: Modularity

- Functions: permit modularity within a program.
- Writing quality modular software
 - Learning the basic syntax of C is easy. But large-scale software is much more.
 - Modularity is the key to managing complexity.
 - Without care, code becomes highly interconnected.
 - Testing and debugging different components separately. Once you are sure a component is correct, you (almost) never have to test it again. Future bugs are likely to be elsewhere.
 - Grouping related functionality. Controlling visibility.
 - Difficult to teach modularity

Definitions

- **Scope** refers to an identifier's *visibility* throughout the program: where in the program can it be used.
- **Extent** refers to the *lifetime* of a variable. When is it created and when is it destroyed. (when is memory allocated for it, and when is the memory released).

Global variable

- global variable, is recognized by its name across all functions in the program
- must be declared outside of any function definition, including main().
- generally discouraged, because their unlimited accessibility can be dangerous

```
double dist = 0.0; // defined outside of any function
int main()
{
    double a, b;

    ...
    return 0;
}
```

```

1 #include<stdio.h>
2 #include<math.h>
3
4 /*
5 demo_global.c
6 taken from Gonzalez, computer programming in C for beginners
7 */
8
9 void dist(double x1, double y1, double x2, double y2);
10 double square(double x);
11
12 double distance = 0.0; // global variable defined outside of any function
13
14 int main()
15 {
16     double x_one = 5.37, y_one = 9.6, x_two = 11.16, y_two = 21.78;
17     dist(x_one, y_one, x_two, y_two);
18     printf("The distance between the two points is %lf \n", distance);
19     return 0;
20 }
21
22 void dist(double x1, double y1, double x2, double y2)
23 // dist() doesn't return anything
24 // It sets the value of the global distance directly
25 {
26     distance = sqrt(square(x1-x2) + square(y1-y2));
27     // variable distance is global
28 }
29
30 double square(double x)
31 {
32     double y = 0.0;
33     y = x * x;
34     return y;
35 }

```

```

frankvp@CRD-L-08004:~/Scope$ gcc demo_global.c -o demo_global -lm
frankvp@CRD-L-08004:~/Scope$ ./demo_global
The distance between the two points is 13.486160
frankvp@CRD-L-08004:~/Scope$

```

Variable declaration: scope

- The scope of a variable: the part(s) of the program in which it is known.
- **Basic rule 1:** identifiers are accessible only within the block in which they are declared
 - a name is local to the block in which it is declared
 - outside the boundaries of that block it is not known
 - storage space for a local variable is provided when its block is entered and released when the block ends
- **Basic rule 2:** a declaration in an inner block hides all declarations of the same name in surrounding blocks

Name Hiding

- It is possible to have two variables with same name, and overlapping scope, without conflict.
- C has scope resolution rules that state the variable with more-restricted scope will hide the other.

```
1#include <stdio.h>
2// demo_scope01.c
3
4void f(int a);
5
6int a, b; // global - - test this
7
8void main(void)
9{
10
11//int a, b; // local to main - test this
12
13
14a = 1;
15b = 2;
16printf("a = %d, b = %d \n", a, b);
17f(a);
18printf("a = %d, b = %d \n", a, b);
19}
20
21void f(int a)
22{
23    a = 3;
24    {
25        int b = 4;
26        printf("a = %d, b = %d \n", a, b);
27    }
28    printf("a = %d, b = %d \n", a, b);
29    b = 5;
30}
```

```
frankvp@CRD-L-08004:~/Scope$ gcc demo_scope01.c -o demo_scope01
frankvp@CRD-L-08004:~/Scope$ ./demo_scope01
a = 1, b = 2
a = 3, b = 4
a = 3, b = 2
a = 1, b = 5
frankvp@CRD-L-08004:~/Scope$
```

```

1 #include<stdio.h>
2
3 /*
4 demo_scope02.c
5 taken from: http://www.c4learn.com/c-programming/c-file-scope/
6 */
7
8 void message ();
9 int num1 = 1;    // Global
10
11 int main ()
12 {
13     int num1 = 6;
14     printf ("%d \n", num1); // Local variable is accessed
15     message ();
16
17     return 0;
18 }
19
20 void
21 message ()
22 {
23     printf ("%d \n", num1); // Global variable is accessed
24 }

```

```

frankvp@CRD-L-08004:~/Scope$ gcc demo_scope02.c -o demo_scope02
frankvp@CRD-L-08004:~/Scope$ ./demo_scope02
6
1
frankvp@CRD-L-08004:~/Scope$

```

The lifetime of the global variable `num1` extends for the duration of the whole program. The scope is however not in `main`, since it is hidden by the local variable `num1`.

Static Variables

- Variables declared *within* a function are **local** to that function. This is known as **automatic** local variables (they are automatically created and then destroyed as the function is called, and then finishes).
- Static** local variables: The variable will not be destroyed when the function exits, but it (and its value) will persist. The variable is initialized only once.

```

1 /*
2 static_var.c
3 static variable
4 taken from http://gribblelab.org/cbootcamp/5_Functions.html
5 */
6 #include <stdio.h>
7
8 void myFunc(void);
9 void myFunc2(void);
10
11 int main() {
12     myFunc();
13     myFunc2();
14     myFunc();
15     myFunc2();
16     myFunc();
17     // printf("num = %d\n", num); // THIS WOULD NOT WORK
18     return 0;
19 }
20 void myFunc(void) {
21     static int num = 0;
22     // int num = 0;
23     num++;
24     printf("myFunc() has been called %d times so far\n", num);
25 }
26 void myFunc2(void) {
27     int num = 0;
28     num++;
29     printf("myFunc2() has been called %d times so far\n", num);
30 }
31

```

```

frankvp@CRD-L-08004:~/Scope$ gcc static_var.c -o static_var
frankvp@CRD-L-08004:~/Scope$ ./static_var
myFunc() has been called 1 times so far
myFunc2() has been called 1 times so far
myFunc() has been called 2 times so far
myFunc2() has been called 1 times so far
myFunc() has been called 3 times so far
frankvp@CRD-L-08004:~/Scope$

```