Functions and Scope with Arrays

CS 241
Data Organization using C

Instructor: Joel Castellanos
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office: Farris Engineering Center (FEC) room 319

Read: Kernighan & Ritchie

- Due Tuesday, Jan 31
  2.3: Constants
  2.4: Declarations
  2.5: Arithmetic Operators
  2.6: Relational and Logical Operators
  2.7: Type Conversions
  2.8: Increment and Decrement Operators

- Due Thursday, Feb 2
  2.9: Bitwise Operations
  Supplemental reading on class website:
  Math is Fun: binary Number System
  Lab 3: Variation of setbits (Exercise 2-6)
Function Prototype and Definition

1) `#include <stdio.h>`
2) `int foo(int x);`
3) `void main(void)`
4) `{ int n=5;
5) printf("%d\n", foo(n));
6) }`
7) `int foo(int n)`
8) `{ return 2*n;`
9) `}`
10) `output: 10`

A Prototype is needed when:
1) A function is used in a line above where it is defined.
2) A function is defined in a different file.
Quiz: Section 1.7: Functions

This code will not compile because:

a) The version of foo in line 3 accepts a float, but returns an int.
b) The function foo in line 3 has no body.
c) The version of foo in line 3 should not end with a semicolon.
d) The variable n is declared in two different places.
e) The prototype of foo does not agree with the definition.

No Overloaded Functions in C

foo.c:7: error: conflicting types for `foo'
Quiz: Section 1.7: Functions

1) 
#include <stdio.h>
2) int foo(int n)
3) { n = 2*n;
4) printf("foo: n=%d ", n);
5) return n;
6) }
7)
8) void main(void)
9) { int n=5;
10) printf("main: foo(n)=%d, n=%d\n", foo(n),n);
11) }

The output of this C program is:

a) foo: n=5  main: foo(n)=5, n=5
b) foo: n=10 main: foo(n)=10, n=5
c) foo: n=10 main: foo(n)=10, n=10
d) foo: n=10 main: foo(n)=20, n=10

Scope of a Variable in C

All constants and variables have scope:

- The values they hold are accessible in some parts of the program, where as in other parts, they don’t appear to exist.

**Block Scope:** variables declared in a block are visible between an opening curly bracket and the corresponding closing bracket.

**Function Scope:** variables visible within a whole function.

**File Scope:** variables declared static and outside all function blocks.

**Program Scope (global variables):** variables declared outside all function blocks.
Program Scope and Function Scope

```
#include <stdio.h>
int a=4;
int b=7;
void foo()
{
  int b = 12;
  a++;
  printf("foo: a=%d, b=%d\n", a, b);
}

void main(void)
{
  foo();
  printf("main: a=%d, b=%d\n", a, b);
}
```

Output: foo: a=5, b=12
main: a=5, b=7

foo does not return a value
but it has two side effects:
1) Sends data to the standard output stream.
2) Changes a global field: a.

Quiz (sect 1.8): Call by Value

In the C Programming Language, call by value means:

a) When two functions have the same name, the compiler determines which to call by the value of the arguments.

b) The called function is given the address of its arguments so that the function can both read and set the argument’s values.

c) Each called function is assigned a value that is used by the operating system to determine the function’s priority. This is most useful on multi-core systems.

d) The called function is given the values of its arguments which are copied into temporary variables.
Quiz (sect 1.10): Automatic Variables

In the C Programming Language, an automatic variable is:

a) A local variable in a function which comes into existence at the time the function is called, and disappears when the function is exited.

b) A variable that is automatically initialized.

c) A global variable that is automatically available to all functions within the source file.

d) A global variable that is available to all functions within any source file that declare the variable as extern.

e) A variable that is automatically defined by the compiler such as PI, E, and HBAR.

Increment Elements of Global Array: 1 of 2

```
1. #include <stdio.h>
2. #define DATA_COUNT 4
3. #define MAX_VALUE 32
4. int x[DATA_COUNT];
5. int increment(void)
6. { // Adds 1 to each element of global array x[].
7.   // Returns error if any element of x[] is > MAX_VALUE
8. } //Calls increment() some number of times.
```

Global array
Note: this violates our standard: x is too short a name for a global variable.
Increment Elements of Global Array: 2 of 2

```c
8. int increment()
9. { int i;
10. for (i=0; i<DATA_COUNT; i++)
11. { if (x[i] >= MAX_VALUE) return 1;
12.     x[i]++;
13. }
14. return 0;
15. }
16. void main(void)
18. int i;
19. for (i=0; i<5; i++)
20. { if (increment()) printf("ERROR\n");
21.     else
22.        { printf("%d %d %d %d\n", x[0],x[1],x[2],x[3]);
23.         }
24. }
25. }
```

Output:
```
21 16 31 3
22 17 32 4
ERROR
ERROR
ERROR
```
What Does the fibonacci Function Do?

```c
#include <stdio.h>

void fibonacci(int n0, int n1)
{
    int n2 = n0 + n1;
    n0 = n1;
    n1 = n2;
}

void main(void)
{
    int n0 = 1;
    int n1 = 1;
    int i;
    for (i=1; i<10; i++)
    {
        printf("%d ", n0);
        fibonacci(n0, n1);
    }
    printf("\n");
}
```

Nothing!!!

fibonacci does not return a value.
fibonacci has no side effects.

Output:

```
1 1 1 1 1 1 1 1 1
```

Fibonacci on Global Variables

```c
#include <stdio.h>

int n0, n1;

void fibonacci()
{
    int n2 = n0 + n1;
    n0 = n1;
    n1 = n2;
}

void main(void)
{
    n0 = 1; n1 = 1;
    int i;
    for (i=1; i<10; i++)
    {
        printf("%d ", n0);
        fibonacci();
    }
    printf("\n");
}
```

- The body of `fibonacci` is unchanged from the last program.
- In the last version, `n0` and `n1` were local to `fibonacci`.
- In this version, `n0` and `n1` are global.
- Therefore, this version of `fibonacci` has side effects.

Output:

```
1 1 2 3 5 8 13 21 34
```
Fibonacci on Array Parameter

1. #include <stdio.h>
2.
3. void fibonacci(int n[], int a)
4. { //int n2 = n0 + n1;
5.     n[a] = n[a-2] + n[a-1];
6. }
7.
8. void main(void)
9. { int i, n[11];
10.    n[0] = 1; n[1] = 1;
11.    for (i=2; i<11; i++)
12.    { fibonacci(n, i);
13.         printf("%d ", n[i-2]);
14.    }
15.    printf("\n");
16. }

Arrays are passed by reference.

Arrays are passed by reference. fibonacci does not allocate memory for a new array. n in fibonacci points to the same memory as n in main.

Output:

1 1 2 3 5 8 13 21 34

Quiz: Argument Passing

1. #include <stdio.h>
2.
3. void foo(int n[], int i)
4. { n[i] = n[i-2] + n[i-1];
5.     i = 6;
6. }
7.
8. void main(void)
9. { int i, n[11];
10.    n[0] = 1; n[1] = 1;
11.    for (i=2; i<11; i++)
12.    { fibonacci(n, i);
13.         printf("%d ", n[i-2]);
14.    }
15.    printf("\n");
16. }

What is the output of this program:

a) 8
b) 10
c) 12
d) 14
e) 16
Quiz Solution: Argument Passing

```c
#include <stdio.h>

void foo(int n[], int i)
{ n[i] = n[i-2] + n[i-1];
  i = 6;
}

void main(void)
{ int i, n[11];
  for (i=0; i<11; i++)
    { n[i] = i*2;
    }
  i=4;
  foo(n, i);
  printf("%d\n", n[4]);
}
```

**X Raised to the Yth Power: all in main()**

```c
void main(void)
{ int i;
  int x=2, y=4, pow=1;
  for (i=0; i<y; i++)
  { pow *= x; //pow = pow * x;
  }
  printf("%d\n",pow); //output: 16
}

x=3; y=4; pow = 1; // already declared
for (i=0; i<y; i++)
{ pow *= x;
}
printf("%d\n",pow); //output: 81
```
### X Raised to the Yth Power: by Function

```c
1) int xToPowerY(int x, int y)  
2) {                         
3)   int pow = 1;            
4)   for (int i=0; i<y; i++) 
5)     { pow *= x;          
6)     }                    
7)   return pow;            
8) }                        
9) void main(void)          
10) {                       
11)   printf("%d\n",xToPowerY(2,4)); // 16  
12)   printf("%d\n",xToPowerY(3,4)); // 81  
13) }                       
```

### String Length: (Section 2.3)

```c
1) #include <stdio.h>       
2) int strLength(char s[])  
3) {                        
4)   int n=0;               
5)   while (s[n]) n++;     
6)   return n;             
7) }                       
8) int main(void)          
9) {                       
10)   char str[] = "Euler";  
11)   printf("%s: %d\n", str, strLength(str));  
12) }                      
```