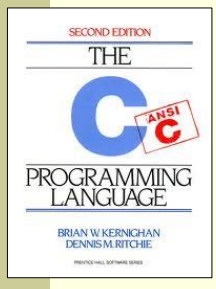
Functions and Scope with Arrays

CS 241
Data Organization using C

Instructor: **Joel Castellanos**
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office: Farris Engineering Center, Room 2110

9/25/2019

Read: Kernighan & Ritchie



- Due Thursday, Sept 5
  - 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
  - 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)
3)  int foo(int x);
4)
5)  void main(void)
6)  { int n=5;
7)    printf("%d\n", foo(n));
8)  }
9)
10) int foo(int n)
11) { return 2*n;
12) }
```

Function Prototype

Must Agree

Function Definition

```
Output:
10
```

3

Function Prototype and Definition

```
1)  #include <stdio.h>
2)
3)  int foo(int x);
4)
5)  void main(void)
6)  { int n=5;
7)    printf("%d\n",foo(n));
8)  }
9)
10) int foo(int n)
11) { return 2*n;
12) }
```

```
1)  //Prototype of foo not needed.
2)  #include <stdio.h>
3)
4)  int foo(int n)
5)  { return 2*n;
6)  }
7)
8)  void main(void)
9)  { int n=5;
10)   printf("%d\n",foo(n));
11) }
```

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***.

4

Quiz: Section 1.7: Functions

```
1)  #include <stdio.h>
2)
3)  int foo(float x);
4)
5)  void main(void)
6)  { int n=5;
7)    printf("%d\n", foo(n));
8)  }
9)
10) int foo(int n)
11) { return 2*n;
12) }
```

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.

5

5

No Overloaded Functions in C

```
1)  #include <stdio.h>
2)
3)  int foo(int n)
4)  { return 2*n;
5)  }
6)
7)  int foo(int k, int n)
8)  { return k*n;
9)  }
10)
11) void main(void)
12) { int n=5;
13)   printf("%d\n", foo(n));
14)   printf("%d\n", foo(3,n);
15) }
```

```
foo.c:7: error:
conflicting
types for 'foo'
```

6

6

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

7

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.

8

Program Scope and Function Scope

```
1.  #include <stdio.h>
2.  int a=4;
3.  int b=7;
4.  void foo()
5.  {
6.    int b = 12;
7.    a++;
8.    printf("foo: a=%d, b=%d\n", a, b);
9.  }
10.
11. void main(void)
12. {
13.   foo();
14.   printf("main: a=%d, b=%d\n", a, b);
15. }
```

`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`.

Output:

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

9

9

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.

10

10

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.
3.  #define DATA_COUNT 4
4.  #define MAX_VALUE 32
5.
6.  int x[DATA_COUNT];
7.
8.  int increment(void)
9.  { // Adds 1 to each element of global array x[].
⇓     // Returns error if any element of x[] is > MAX_VALUE
15. }

16. void main(void)
17. { //Sets initial values of x[].
⇓     //Calls increment() some number of times.
25. }
```

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

```
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)
17. { x[0] = 20; x[1] = 15; x[2] = 30; x[3] = 2;
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
```

13

13

Increment Elements of Global Array: 2 of 2

```
8.  int increment()
9.  { int i;
10.   for (i=0; i<DATA_COUNT; i++)
11.   { if (x[i] >= MAX_VALUE) return 1;
12.   }
13.
14.   for (i=0; i<DATA_COUNT; i++)
15.   { x[i]++;
16.   }
17.   return 0;
18. }
```

Output:

```
21 16 31 3
22 17 32 4
ERROR
ERROR
ERROR
```

14

14

What Does the fibonacci Function Do?

```
1.  #include <stdio.h>
2.
3.  void fibonacci(int n0, int n1)
4.  { int n2 = n0 + n1;
5.    n0 = n1;
6.    n1 = n2;
7.  }
8.
9.  void main(void)
10. { int n0 = 1;
11.   int n1 = 1;
12.   int i;
13.   for (i=1; i<10; i++)
14.   { printf("%d ", n0);
15.     fibonacci(n0, n1);
16.   }
17.   printf("\n");
18. }
```

Nothing!!!
fibonacci does not return a value.
fibonacci has no side effects.

Output:

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

15

15

Fibonacci on Global Variables

```
1.  #include <stdio.h>
2.  int n0, n1;
3.
4.  void fibonacci()
5.  { int n2 = n0 + n1;
6.    n0 = n1;
7.    n1 = n2;
8.  }
9.
10. void main(void)
11. { n0 = 1; n1 = 1;
12.   int i;
13.   for (i=1; i<10; i++)
14.   { printf("%d ", n0);
15.     fibonacci();
16.   }
17.   printf("\n");
18. }
```

- 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
```

16

16

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. }
```

Allocates new memory for an integer and the ***value*** passed to fibonacci is ***copied*** into that new memory.

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
```

17

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.   for (i=0; i<11; i++)
11.   {
12.     n[i] = i*2;
13.   }
14.   i=4;
15.   foo(n, i);
16.   printf("%d\n", n[i]);
17. }
```

What is the output of this program:

a) 8
b) 10
c) 12
d) 14
e) 16

18

Quiz Solution: 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.   for (i=0; i<11; i++)
11.   { n[i] = i*2;
12.   }
13.   i=4;
14.   foo(n, i);
15.   printf("%d\n", n[i]);
16. }
```

n[]: Passed by *Reference*

i: Passed by *Value*

Dead Store

```
n[4] = n[3] + n[2]
     =   6  +  4
     = 10
```


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

```
1)  void main(void)
2)  { int i;
3)    int x=2, y=4, pow=1;
4)    for (i=0; i<y; i++)
5)    { pow *= x; //pow = pow * x;
6)    }
7)     printf("%d\n",pow);     //output: 16
8)
9)    x=3; y=4; pow = 1;       // already declared
10)   for (i=0; i<y; i++)
11)   { pow *= x;
12)   }
13)   printf("%d\n",pow);  //output: 81
14) }
```

X Raised to the Yth Power: by Function

```
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)
10) void main(void)
11) {
12)   printf("%d\n",xToPowerY(2,4)); // 16
13)   printf("%d\n",xToPowerY(3,4)); // 81
14) }
```

21

21

String Length: (Section 2.3)

```
1)  #include <stdio.h>
2)  int strLength(char s[])
3)  {
4)    int n=0;
5)    while (s[n]) n++;
6)    return n;
7)  }
8)
9)  int main(void)
10) {
11)   char str[] = "Euler";
12)   printf("%s: %d\n", str, strLength(str));
13) }
```

22

22

Segmentation fault (core dumped) ?

```
#include <stdio.h>

char* getString()
{
   char str[] =
     "Never return a pointer to stack memory";
   return str;
}

void main(void)
{
  char* str = getString();
  printf("%s\n", str);
}
```

23

How about this?

```
#include <stdio.h>

char getChar()
{
   char c = 'N';
   return c;
}

void main(void)
{
  char str = getChar();
  printf("%c\n", str);
}
```

24