

Functions, Arguments, and Scope

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 321

Lab Instructor: **Dongye Meng**

e-mail: dymeng@cs.unm.edu



3/6/2009

Binary Output of unsigned char

```
1. #include <stdio.h>
2. int main()
3. { unsigned char a=22;
4.   int i;
5.
6.   for (i=7; i>=0; i--)
7.     { unsigned char n = 1 << i;
8.       if (a & n) printf("1");
9.       else printf("0");
10.    }
11. printf("\n");
```

Output: 00010110

2

Function Declarations

In C, functions must be declared lexically before being used.

```
1. void printBinary(unsigned char a)
2. { ...
3. }
4. int main()
5. { printBinary(62);
6. }
```

Originally, C was a one pass compiler.

3

Function: printBinary

```
1. void printBinary(unsigned char a)
2. { int i;
3.   for (i=7; i>=0; i--)
4.   { unsigned char n = 1 << i;
5.     if (a & n) printf("1"); else printf("0");
6.   }
7.   printf("\n");
8. }
9. int main()
10. { unsigned char a=66;
11.   unsigned char b=55;
12.   printBinary(a);
13.   printBinary(b);
14.   printBinary(44);
15. }
```

Allocates new memory on stack.

Passed by value

Output:

```
01000010
00110111
00101100
```

4

Pass by Value verses Pass by Address

```
1. void foo(int x, int* y)
2. { printf("foo:: x=%d, y=%d\n", x, *y);
3.   x = 5;
4.   *y = 6;
5.   printf("foo:: x=%d, y=%d\n", x, *y);
6. }
7.
8. int main()
9. { int a = 2;
10.  int b = 3;
11.
12.  foo(a, &b);
13.  printf("main:: a=%d, b=%d\n", a, b);
14. }
```

***y is the value at Address y**

Output:

```
foo:: x=2, y=3
foo:: x=5, y=6
main:: a=2, b=6
```

a: Value

b: Address

⁵

#1 of 2: What is the Error?

```
1. void foo(int x, int *y)
2. { printf("foo:: x=%d, y=%d\n", x, *y);
3.   x = 5;
4.   *y = 6;
5. }
6. int main()
7. { int a = 2;
8.   int b = 3;
9.   foo(a, b);
10. }
```

Error: 9: warning: passing argument 2 of foo makes pointer from integer without a cast

6

#2 of 2: What is the Error?

```
1. void foo(int x, int y)
2. { printf("foo:: x=%d, y=%d\n", x, *y);
3.   x = 5;
4.   *y = 6;
5. }
6. int main()
7. { int a = 2;
8.   int b = 3;
9.   foo(a, &b);
10. }
```

Error: 2: error: invalid type argument of unary *
4: error: invalid type argument of unary *
9: warning: passing argument 2 of foo makes integer from pointer without a cast

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.

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

8

Program and Function Scope Variables

```
1. int a=4;
2. int b=7;
3. void foo()
4. { int b = 12;
5.   a++;
6.   printf("foo:: a=%d, b=%d\n", a, b);
7. }
8.
9. int main()
10. { foo(a, b);
11.   printf("main:: a=%d, b=%d\n", a, b);
12. }
```

Output:

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

9

Variable Qualifier: static

```
1. void foo()
2. { int static a = 12;
3.   int b = 12;
4.   printf("foo:: a=%d, b=%d\n", a, b);
5.   a++;
6. }
7.
8. int main()
9. { int a = 88;
10.  foo();
11.  printf("main:: a=%d\n", a);
12.  foo();
13.  printf("main:: a=%d\n", a);
14. }
```

Function Scope, with
Global Persistence

Output:

```
foo:: a=12, b=12
main:: a=88
foo:: a=13, b=12
main:: a=88
```

10

Global Scope: setBinaryStr()

```
char binaryStr[9];
void setBinaryStr(unsigned char n)
{ int i;
  binaryStr[8] = 0;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    if (n & bitMask) binaryStr[i] = '1';
    else binaryStr[i] = '0';
  }
}

int main()
{ setBinaryStr(75);
  printf("%s\n",x, binaryStr);
}
```

Output: 01001011

11

Walking Out of Bounds

```
void setRunAwayStr(char str[])
{ int i;
  for (i=0; i<20; i++)
  { str[i] = '1';
  }
  str[20]=0;
}

int main()
{ char binaryStr[9];
  setRunAwayStr(binaryStr);
  printf("%s\n", binaryStr);
}
```

11111111111111111111

or

Output: Segmentation fault

or

?

12

Array Argument: setBinaryStr()

```
void setBinaryStr(unsigned char n, char str[])
{ int i;
  str[8] = 0;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    if (n & bitMask) str[i] = '1';
    else str[i] = '0';
  }
}

int main()
{ char binaryStr[9];
  unsigned char x = 160;
  setBinaryStr(x, binaryStr);
  printf("%s\n", binaryStr);
}
```

Output: 10100000

13

Changing char str[] to char*

```
//void setBinaryStr(unsigned char n, char str[])
void setBinaryStr(unsigned char n, char* str)
{ int i;
  str[8] = 0;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    if (n & bitMask) str[i]='1'; else str[i]='0';
  }
}

int main()
{ char binaryStr[9];
  unsigned char x = 160;
  setBinaryStr(x, binaryStr);
  printf("%s\n", binaryStr);
}
```

Output: 10100000
no compiler
messages.
A char array is a
pointer to a char.

14

- Exam Thursday after spring break
- Review on Tuesday after spring break
- Exam: like clicker questions – but short answer.

15

Quiz 3-1

```
void setBinaryStr(unsigned char n, char* str)
{ int i; str[8] = 0;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    printf("%d, ", n & bitMask);
  }
}
int main()
{ char binaryStr[9];
  setBinaryStr(155, binaryStr);
}
```

- a) 128, 0, 0, 16, 8, 0, 2, 1,
- b) 128, 64, 32, 16, 8, 4, 2, 1,
- c) 0, 64, 32, 0, 0, 4, 0, 0,
- d) 0, 0, 0, 0, 0, 0, 0, 0,
- e) 1, 0, 0, 1, 1, 0, 1, 1,

16

Changing char str[9] to char* str

```
void setBinaryStr(unsigned char n, char str[])
{ int i;
  str[8] = 0;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    if (n & bitMask) str[i]='1'; else str[i]='0';
  }
}

int main()
{ char* str; //char str[9];
  unsigned char x = 160;
  setBinaryStr(x, str);
  printf("%s\n", str);
}
```

gcc compiles this with
neither errors nor warnings.

10100000
sometimes

Output:

17

Changing a Pointer Argument

```
void setBinaryStr(unsigned char n, char inStr[])
{ int i;
  char str[9];
  str[8] = 0;
  inStr = str;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    if (n & bitMask) str[i]='1'; else str[i]='0';
  }
}

int main()
{ char* str; //char str[9];
  unsigned char x = 160;
  setBinaryStr(x, str);
  printf("%s\n", str);
}
```

gcc compiles this with
neither errors nor warnings.

Output:

*Something
random*

18

What is Wrong with setBinaryStr()?

```
char* getBinaryStr(unsigned char n)
{ int i;
  char str[8]; str[8] = 0;
  for (i=0; i<8; i++)
  { unsigned char bitMask = 1 << (7-i);
    if (n & bitMask) str[i] = '1';
    else str[i] = '0';
  }
  return str;
}

int main()
{ char* binaryStr = getBinaryStr(180);
  printf("%s\n", binaryStr);
}
```

Error:

gcc: warning: function
returns address of
local variable

19

Function Return: int

```
1. int foo(int n)
2. { int i;
3.   int answer = 1;
4.   for (i=2; i<=n; i++)
5.     { answer *= i;
6.     }
7.   return answer;
8. }

9. int main()
10. { printf("%d\n", foo(7));
11. }
```

- Is it ok to return a local variable? ✓
- Will this compile without warnings? ✓

Output: 5040

20

Quiz: 3-2: What is the Output?

```
int foo(int n)
{ int i;
  int answer = 1;
  for (i=2; i<=n; i++)
  { printf("%d, ", answer);
    answer += i;
  }
  return answer;
}

int main()
{ printf("%d\n", foo(6));
}
```

- a) 1, 1, 3, 4, 5, 9
- b) 1, 3, 4, 5, 9
- c) 2, 3, 4, 5, 6, 7
- d) 1, 3, 6, 10, 15, 21
- e) 3, 6, 10, 15, 21

21

Function that Needs to Return 2 Values

```
1. #include <stdio.h>
2. void getMinMax(int center, int radius,
3.               int* min, int* max)
4. { *min = center - radius;
5.   *max = center + radius;
6. }
7.
8. int main()
9. { int min, max;
10.  getMinMax(10, 3, &min, &max);
11.  printf("min=%d, max=%d\n", min, max);
12. }
```

Output: min=7, max=13

22

Quiz 3-3:

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

a) 5, 15
b) 5, 75
c) 25, 15
d) 25, 75
e) 25, (unpredictable)