Lab Assignment 1

ZedBoard Linux Continued

Zamir Johl and Michael Wong

[johl.z@husky.neu.edu](mailto:johl.z@husky.neu.edu)

[wong.mich@husky.neu.edu](mailto:wong.mich@husky.neu.edu)

Submit Date: 1/27/15

Due Date: 1/28/15

**Abstract**

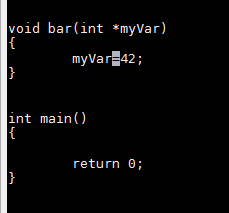
The purpose of this lab was to explore the use of pointers in C and to apply this to our programming. This involved the commenting of a pre-written program to demonstrate our understanding of the code. In addition, we experimented with different data formats and logical operations. A number of small programs were used to guide this particular exercise.

**Introduction**

The goal of this lab was to expand our understanding of the C programming language and to gain more familiarity with Linux. The pre-lab asked about the differences between pass-by-value and pass-by-reference which were a tie-in to the discussion of pointers in the lab. This was explored by printing octal and hexadecimal from decimal values. The lab itself started with an examination of two Linux concepts: file permissions and compiler options. Then, we focused on writing two programs that used different data formats and logical operations. Finally, the knowledge obtained in the pre-lab was applied to a program that utilized pointers. The goal was to accurately comment this code to demonstrate comprehension.

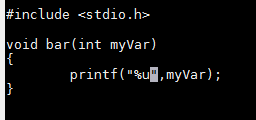
**Lab Discussion**

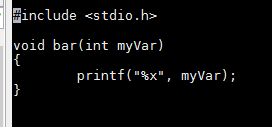
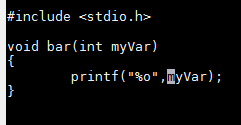
Pre 1.1:

1. No, changes to *myVar* within *foo* would not be visible to *main* because the value of *myVar* is passed to the function *foo* and then altered. Were the reference passed, the changes made by the function would be visible to *main*.
2. 
3. Yes, the result would be visible to the function calling *bar*
4. Pass by value is transfer of all the information contained in a parameter. Pass by reference is the transfer of the parameter itself in terms of a memory address or reference. The key is that a pass by reference will make any changes within the function overwrite the parameter which will make the changes visible to the preceding function.

Pre 1.2

1. Decimal Value



1. Hexadecimal Value
2. Octal Value

Pre 1.3:

Code located in Appendix

Hardware Used:

* ZedBoard and included power supply
* SD card with OS and file system
* Ethernet cable and adapter
* Dell/Windows 7 host computer
* COE Ubuntu computer (accessed via SSH)

Software Used:

* Windows 7
* Ubuntu
* Xilinux
* MobaXterm Personal Edition v 7.4

**Results and Analysis**

1.0

This portion of the lab concerned the compiling of the calculator program from the PreLab. This led into an examination of the file permissions for both the source and executable files. For the source file, we had read and write permission and the file itself was 2442 bytes. On the other hand, for the executable file, we had read, write and executable permission and the file was 8254 bytes in size.

1.1

In this section, we examined the function of the –S option in the gcc compiler. It seems that –S converts the source code into machine code or assembler code. When this new IntegerMath.s file was compiled it ran as expected.

1.2

To explore different data formats, we created a program that printed out two integers as decimal, octal and hexadecimal value. This was intended to lead into the use of pointers. Essentially, the printf function goes into the memory location and reads the binary value. This is then converted to the octal, decimal or hexadecimal formats based upon the option selected in the function.

1.3

Then, convert was modified to preform different bitwise operations and display them. What was found was that the output was a product of the bitwise operations but did not include a negative sign.

**Conclusion**

All in all, this program allowed us to gain a greater familiarity with the C programming language and with Linux as a whole. The pre lab gave us a solid understanding of pass-by-reference which was used during the lab proper to introduce pointers. Additionally, the inclusion of other data formats and bitwise logic were an introduction to concepts that will be further explored later in this class. Although the extra credit, a program that takes the transpose of a matrix, was attempted, our results were ultimately unsuccessful. The function using array indices was simple but we were beset by type errors when attempting to use pointers and did not have a chance to actually finish that portion of the program. All in all, in this lab we explored a variety of traits of C and Linux which will certainly pay off during the remainder of this course.

**Appendix**

**Pre 1.3:**

**IntegerMath.c:**

#include <stdio.h>///include io library

/\*\*

\*add\_num

\*Adds two numbers that are provided

\*

\*@param num1 first number to be added

\*@param num2 second number to be added

\*@return sum of num1 and num2

\*/

int add\_num(int num1, int num2) ///Function adds two parameters and returns the sum

{

return num1+num2; ///return sum of num1 and num2

}

/\*\*

\*sub\_num

\*Subtracts two numbers that are provided

\*

\*@param num1 number to be subtracted from

\*@param num2 number to be subtracted

\*@return difference between num1 and num2

\*/

int sub\_num(int num1, int num2) ///Function subtracts two parameters and returns the difference

{

return num1-num2; ///return difference between num1 and num2

}

/\*\*

\*mult\_num

\*Multiply two numbers that are provided

\*

\*@param num1 first number to be multiplied

\*@param num2 second number to be multiplied

\*@return product of num1 and num2

\*/

int mult\_num(int num1, int num2) ///Function multiplies two parameters and returns the product

{

return num1\*num2; ///return product of num1 and num2

}

/\*\*

\*div\_num

\*Divides two numbers that are provided

\*

\*@param num1 dividend

\*@param num2 divisor

\*@return num1 divided by num2

\*/

int div\_num(int num1, int num2) ///Function divides two parameters and returns the result

{

return num1/num2; ///return num1 divided by num2

}

int main() ///main function

{

int num1,num2,res; ///Initializes num1,num2 and res variables

char out; ///Intializes out variable

printf("Please enter an integer\n"); ///Prompt user for first number

scanf("%d",&num1); ///Stores input as num1

printf("\nPlease enter another integer\n"); ///Prompt user for second number

scanf("%d",&num2); ///Stores input as num2

printf("\nWhat operation would you like to be performed? (+,-,\*,or/)\n"); ///Prompts user for operation

scanf("%c",&out); ///Stores operation choice in out variable

if(out == '+'){ ///Determines if choice is addition

printf("%d",add\_num(num1,num2)); ///Performs addition function

}

if(out == '-'){ ///Determines if choice is subtraction

printf("%d",sub\_num(num1,num2)); ///Performs subtraction function

}

if(out == '\*'){ ///Determines if choice is multiplication

printf("%d",mult\_num(num1,num2)); ///Performs multiplication function

}

if(out == '/'){ ///Determines if choice is division

printf("%d",div\_num(num1,num2)); ///Performs division function

}

else{

printf("\nYour operation choice was not recognized\n"); ///Informs user that an error has occured

}

return 0; ///returns 0

}

**1.2:**

**convert.c:**

#include <stdio.h> ///Imports standard io library

/\*\*

\*@file convert.c

\*

\*@author Zamir Johl and Michael Wong

\*

\*This program asks the user to input two integers which

\*are then printed in decimal, octal and hexadecimal form

\*/

int main(void) ///Main method

{

int num1, num2; ///Variables to store user inputs

printf("Enter the first integer to convert: \n"); ///Prompts the user for an integer

scanf("%d", &num1); ///Stores integer as a decimal in num1

printf("Enter the second integer to convert: \n"); ///Prompts the user for an integer

scanf("%d", &num2); ///Stores integer as a decimal in num2

printf("%d is %X in hex. %d is %o in octal.\n", num1, num1, num1, num1); ///Prints num1 in decimal, hex and octal format

printf("%d is %X in hex. %d is %o in octal.\n", num2, num2, num2, num2); ///Prints num2 in decimal, hex and octal format

return 0;///Returns 0, ends program

}

**1.3:**

**logicalOp.c:**

#include <stdio.h> ///Imports standard io library

/\*\*

\*@file logicalOp.c

\*

\*@author Zamir Johl & Michael Wong

\*

\*This program prompts the user for two integers and then performs

\*a series of bitwise operations and displays the result

\*/

int main(void) ///Main function

{

int num1, num2; ///Variables to store user inputs

printf("Enter the first integer to convert: \n"); ///Prompts the user for an integer

scanf("%d", &num1); ///Stores integer as a decimal in num1

printf("Enter the second integer to convert: \n"); ///Prompts the user for an integer

scanf("%d", &num2); ///Stores integer as a decimal in num2

int and = num1 & num2; ///Applies the bitwise AND operation to the numbers and stores in a new variable

int or = num1 | num2; ///Applies the bitwise OR operation to the numbers and stores in a new variable

int xor = num1 ^ num2; ///Applies the bitwise XOR operation to the numbers and stores in a new variable

printf("The value of %d AND %d is %X\n", num1, num2, and); ///Displays the value of num1 AND num2

printf("The value of %d OR %d is %X\n", num1, num2, or); ///Displays the value of num1 OR num2

printf("The value of %d XOR %d is %X\n", num1, num2, xor); ///Displays the value of num1 XOR num2

return 0; ///Returns 0 and ends the program

}

**1.4:**

**logicalOp.c:**

///@file twosComplememt.cab

///@author Michael Wong and Zamir Johl

#include <stdio.h> ///include standard io library for console inputs and outputs

///main function

///takes a user inputed int and prints the one's and two's complement of its binary representation

///@return int indicated a terminated program

int main(void) ///required main function

{

int num; ///number to be converted

printf("Enter an integer: "); ///prompt the user for an int input

scanf("%d", &num); ///take the user input from console and store its value at num

int ones = ~num; ///assign the bitwise negation of num to ones for the one's complement

int twos = -1 \* num; ///negative numbers are already stored in two's complement form so no special method is required

printf("One's complement of %d is %X.\n", num, ones); ///print the value of the one's complement in hex and the original decimal number

printf("Two's complement of %d is %X.\n", num, twos); ///print the value of the two's complement in hex and the original decimal number

return 0; ///end the program

}

**1.5:**

**pointers.c:**

///@file pointers.c

///@author Michael Wong and Zamir Johl

#include <stdio.h> ///include io library

///perform various operations with pointers and references

///@return int when the program terminates

int main(void) ///main function

{

char ch = 'T'; ///assign character T to ch

char \*chptr = &ch; ///create a pointer chptr to the address of the value of ch

char name[6]; ///create an array called name of characters, aka a string

int a = 1000; ///assign 1000 to integer a

int \*intptr = &a; ///create a pointer named intptr to the address of a

float fnumber = 1.20000; ///create a float called fnumber and assign 1.2 to it

float \*fptr = &fnumber; ///create a pointer called fptr to the address of fnumber

char \*ptr = "My dog has fleas!"; ///create a pointer, ptr, to the string "My dog has fleas!"

printf("\n [%c],[%d],[%f],[%c],[%s]\n", \*chptr, \*intptr, \*fptr, \*ptr, ptr); ///print the values of the above variables by referencing their addresses

///a pointer to an array or string only points to the first element in it, so \*ptr prints 'M' but ptr refers to the entire string

chptr = ptr; /// assign a copy of the value of ptr to chptr

printf("\n [%c], [%s] \n" , \*chptr, chptr); ///print the value \*chptr points to (the first element of the string) and then the entire string

name[0] = 75; ///assign the character with the number 75 to the first index of name

name[1] = 97; ///assign char 97 to the second index of name

name[2] = 0x65; ///assign char 0x65 to the third inded of name

name[3] = 0154; ///assign char 154 to the fourth index of name

name[4] = 105; ///assign char 105 to the fifth index of name

name[5] = 0; ///null terminator, ends the string

printf("\n [%s]\n", name); ///print the string

///this prints a string because the array is of type char, so the numbers are interpreted as ASCII values

return 0; ///end the program

}