 Name: ____

ECE 190 Final Exam
Fall 2012

Wednesday, December 19th, 2012

- **Be sure your exam booklet has 4 double-sided pages.**
- **Ask TA if you need paper for scratch work.**
- **You are allowed three handwritten 8.5 x 11" sheets of notes.**
- **This is a closed book exam. You may not use a calculator.**
- **Absolutely no interaction between students is allowed.**
- **Be sure to clearly indicate any assumptions that you make.**
- **The questions have different level of difficulty. Budget your time accordingly.**
- **Don't panic, and good luck!**

Instructions for accessing/working on the programming problems

- Log into an EWS Linux machine.
- Appendixes A and D from the textbook are provided for you on the desktop.
- Open a terminal window. Check if directory **finalexam** exists.
- Save your code in the files indicated by the problem statements. Do not rename the files. **We will not grade your problem if it is not saved as required.**
- We are NOT using a subversion repository for the exam. We will grade the saved files that you leave in your **finalexam** directory.
- Your code will be graded by an autograder. You may receive zero points if your code does not assemble, does not compile, or does not behave as specified.
- To begin work on problem X, use the **cd** command to get to the **projectX** directory.
- To edit the file, type **gedit givenfilename** &
  (Replace **givenfilename** with the name of the file where you will add your code.)
- Save your work.

LC-3 Tools reference information

- To assemble your code and produce the object file, type **lc3as givenfilename.asm**
- To run the LC-3 graphical simulator, type **lc3sim-tk givenfilename.obj**
- To run the LC-3 command-line simulator, type **lc3sim givenfilename.obj**

C Tools reference information

- If **Makefile** is provided, use it for compiling the project code and follow the instructions provided with the problem statement for running and testing your program
- If no **Makefile** is provided, use **gcc** or **clang** compiler as follows:
  - To compile your code using clang compiler, type **clang -ansi -Wall -g -lm <c file name>**
  - To compile your code using gcc compiler, type **gcc -ansi -Wall -g -lm <c file name>**
  - To execute your code, type **./a.out**

Programming Problem 1 (25 pts): C to LC-3 Assembly Conversion

Convert the following function from C to LC-3. This function recursively traverses a tree and returns the value stored in the left-most leaf node.

```
char left_most(treenode *root)
{
    if (!root->left) {
        return root->data;
    }

    return left_most(root->left);
}
```

The tree node is defined as follows:

```
typedef struct treenode
{
    struct treenode *left;
    struct treenode *right;
    char data;
} treenode;
```

Remember that structures merely consist of simple data types where one element always comes after another in memory. That is if `root` points to an address of x2003, the `left` pointer address is contained within x2003, the `right` pointer address is contained within x2004, and data value is contained within x2005.

Your program should be saved in **finalexam/project1/left_most.asm** file. The file already contains an LC-3 assembly implementation of *MAIN* function that calls `left_most` function. **You should not modify *MAIN* function**.

data.asm file is provided for your convenience for testing. It contains a short hard-coded tree consisting of 6 nodes. As you can see from *MAIN's* implementation, tree's root node is stored starting at address x2003.

To receive full credit, your program should correctly run in lc3sim (or lc3sim-tk) and should produce the correct output. You must use the run-time stack convention presented in the textbook for subroutine invocation. Your function does not need to save and restore the registers. You may not make any assumptions about the behavior of *MAIN* other than that it follows the runtime convention presented in the textbook.

To test your `left_most` function using the provided *MAIN* and **data.asm**, assemble both **left_most.asm** and **data.asm**, load them in the lc3 simulator, set PC to x3000, and run it.

Your code will be graded by an autograder for functionality of the `left_most` function as well as the correct construction of the activation record in the run-time stack. Consequently, you MAY NOT perform any optimizations that impact the contents of the stack. For example, ALL local variables used in the C function, if any, should be stored in the run-time stack appropriately. The local variables may NOT be stored locally in memory reserved by .FILL, .BLKW, etc. You may receive zero points if your code does not assemble or does not behave as specified.

Programming Problem 2 (20 pts): Linked Data Structures

Implement a **recursive** function for counting the number of negative values stored in a tree:

```
int countNegatives(struct node* root);
```

The function requires one argument: pointer to the root node of the tree. It should traverse the tree and return the number of negative value stored in the tree. Tree node data structure is defined as follows:

```
struct node
{
    int data;
    struct node* left;
    struct node* right;
}
```

Provided files (in finalexam/project2)

problem2.c contains `main` function that is used for setting up a simple tree and testing the function you write. You can modify it as needed for testing.

tree.c should contain the function that you are required to write. All the code you write should be in this file.

tree.h contains the function prototypes necessary to use your function. Do not modify this file, do not change function prototypes!

Makefile is provided to compile your code by typing **make** when in **finalexam/problem2** directory. Do not modify this file!

Compiling and running

To compile your code, simply type **make**

To test your code, type **./problem2**

 Name: ____

Programming Problem 4 (35 pts): Problem Solving with Files and Arrays

In this assignment, you are required to implement a program for computing perimeter of a polygon based on its vertex coordinates stored in a file. The computed results should be stored into another file.

Input file format

Input file contains *the number of vertices* record followed by space-separated *x* and *y coordinate* records for each vertex. For example, the following file contains a polygon consisting of 4 vertices:

```
4
1 1
1 5
2 4
3 2
```

Note that the polygon is formed by connecting vertices in the order they appear in the input file.

Vertice data structure

```
typedef struct {
    int x, y;           /* vertex coordinates */
    float length;       /* polygon's side length */
} vertex;
```

Functions to implement

```
vertex * read_polygon(char *file_name, int *count);
```

This function should read the *number of vertices* record from the file with the supplied *`file_name`* and populate an array of *vertex* records with data read from the file. Memory for the array of *vertex* records should be allocated dynamically. Note that only `x` and `y` fields need to be populated by this function. The function should return a pointer to the populated array of *vertex* records and the number of read records in `count`. If the function fails to read data from the file, it should return NULL and `count` should be set to 0.

```
float calc_perimeter(vertex* vrtx, int count);
```

This function should calculate length of each side of the polygon as well as the polygon's perimeter. The length of polygon's side is computed as $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. This value should be stored in the `length` field of the *vertex* with coordinates (x_1, y_1). Perimeter is computed by adding up all sides of the polygon. This value should be computed and returned by the function. (Do not forget to add ALL sides when computing the perimeter.)

```
int record_polygon(char *file_name, vertex *vrtr, int count, float
perimeter);
```

This function should write to file `file_name` the *number of vertices* record followed by records for each vertex consisting of the space-separated *x*, *y*, and *length* fields, followed by the perimeter. For example, the following output file should be created for the above input file:

```
4
1 1 4.00
1 5 1.41
2 4 2.24
3 2 2.24
9.89
```

The function should also free memory allocated for vertex array. It should return 1 if the data is successfully stored in the output file, or 0 otherwise.

<u>Provided files</u> (in finalexam/project4)

problem4.c contains `main` function that is used for testing the functions you write. It calls your functions and also prints the results of `calc_perimeter` to the screen. You should not modify it.

polygon.c should contain all the functions that you are required to write. All the code you write should be in this file.

polygon.h contains the function prototypes necessary to use your functions. Do not modify this file, do not change function prototypes!

Makefile is provided to compile your code by typing **make** when in **problem4** directory. Do not modify this file!

input.txt is a sample input file used to test your code. The input file we use when grading will have a different name and content. Make sure you do not hardcode values!

<u>Compiling and testing</u>

To compile your code, simply type **make**

To test your code, type **./problem4 input.txt output.txt**

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

Opcode	Encoding	Assembly	RTL
ADD	0001 | DR | SR1 | 0 | 00 | SR2	ADD DR, SR1, SR2	DR ← SR1 + SR2, Setcc
ADD	0001 | DR | SR1 | 1 | imm5	ADD DR, SR1, *imm5*	DR ← SR1 + SEXT(imm5), Setcc
AND	0101 | DR | SR1 | 0 | 00 | SR2	AND DR, SR1, SR2	DR ← SR1 AND SR2, Setcc
AND	0101 | DR | SR1 | 1 | imm5	AND DR, SR1, *imm5*	DR ← SR1 AND SEXT(imm5), Setcc
BR	0000 | n | z | p | PCoffset9	BR{nzp} *PCoffset9*	((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)
JMP	1100 | 000 | BaseR | 000000	JMP BaseR	PC ← BaseR
JSR	0100 | 1 | PCoffset11	JSR *PCoffset11*	R7 ← PC, PC ← PC + SEXT(PCoffset11)
TRAP	1111 | 0000 | trapvect8	TRAP *trapvect8*	R7 ← PC, PC ← M[ZEXT(trapvect8)]
LD	0010 | DR | PCoffset9	LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010 | DR | PCoffset9	LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110 | DR | BaseR | offset6	LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110 | DR | PCoffset9	LEA DR, *PCoffset9*	DR ← PC + SEXT(PCoffset9), Setcc
NOT	1001 | DR | SR | 111111	NOT DR, SR	DR ← NOT SR, Setcc
ST	0011 | SR | PCoffset9	ST SR, *PCoffset9*	M[PC + SEXT(PCoffset9)] ← SR
STI	1011 | SR | PCoffset9	STI SR, *PCoffset9*	M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111 | SR | BaseR | offset6	STR SR, BaseR, *offset6*	M[BaseR + SEXT(offset6)] ← SR