NAME: ____________________________________________

1. (20 points) The C code for the "isGreater" function from project 1 is as follows:

```
1. int isGreater(char *s1,char *s2) {
2.    while(*s1) {
3.       if ((*s1)>(*s2)) return 1;
4.       if ((*s1)<(*s2)) return -1;
5.       s1++; s2++;
6.    }
7.    if ((*s2)==0) return 0;
8.    return -1;
9. }
```

For each of the following lines of x86 assembler code, identify the line number (or line numbers) of the C code which generated the x86 assembler code. Note – the x86 "movzbl" instruction copies from an 8 bit byte to a 32 bit long word, padding to the left with zeroes. The x86 "test" instruction performs a bitwise AND of its arguments, but does not save the results – just sets the condition codes.

A. ____4____

```
     movl      8(%ebp), %eax
     movzbl    (%eax), %edx
     movl      12(%ebp), %eax
     movzbl    (%eax), %eax
     cmpb      %al, %dl
     jge       .L27
     movl      $-1, %eax
     jmp       .L26
.L27
```

B. ___2,6______

```
     jmp       .L24
.L28
     …
.L24:
     movl      8(%ebp), %eax
     movzbl    (%eax), %eax
     testb     %al, %al
     jne       .L28
```

C. ____5_____

```
     addl      $1, 8(%ebp)
     addl      $1, 12(%ebp)
```

D. ____7_____

```
     movl      12(%ebp), %eax
     movzbl    (%eax), %eax
     testb     %al, %al
     jne       .L29
     movl      $0, %eax
     jmp       .L26
.L29
```

2. (20 points) The x86 instructions that enter and exit the "isGreater" function is as follows:

```
isGreater:
        pushl       %ebp
        movl        %esp, %ebp
        …
.L26:
        popl        %ebp
        ret
```

   a. How many bytes are required for the "isGreater" stack frame? ___4___

   b. If the "isGreater" function had any local variables, would the entry and exit code be different? If so, what other instructions would be required?
   Yes… would need subl $x,%esp at the top, and replace "popl %ebp" with "leave" at the bottom.

   c. If the "isGreater" function invoked any lower level functions which required argument values, would the entry and exit code be different? If so, what other instructions would be required?
   Yes… would need subl $x,%esp at the top, and replace "popl %ebp" with "leave" at the bottom.

   d. Would the "isGreater" function still work if we removed the pushl %ebp; movl %esp,%ebp; and popl %ebp instructions?
   No… we depend on the value of %ebp to read the parameters.

3. (10 points) Mark each statement below as either true (T) or false (F). Assume you are running GDB on the llist binary that includes the "isGreater" function, as defined above, with the associated x86 assembler code as defined in problems 1 and 2.

   a. __T__ If there is a breakpoint set at "isGreater", then GDB will open a prompt after the execution of the "movl %esp,%ebp" instruction.

   b. __T__ If GDB provides a prompt after executing the "move %esp,%ebp" instruction at the top of the "isGreater" function, you can type the command "x /3xw $ebp" to see 1) the caller's %ebp, 2) the return address to the caller, and 3) the first parameter to isGreater.

   c. __F__ If GDB provides a prompt at the .L24 label (as in problem 1.b above), the GDB "next" instruction would cause GDB to open a new prompt after executing the instruction "movl 8(%ebp),%eax

   d. __T__ If GDB provides a prompt at the .L24 label (as in problem 1.b above), and %ebp+8 has the value 0xffffdac0, then the GDB command "x /s 0xffffdac0" will print the value of the s1 string.

e. __F__ If GDB provides a prompt at the .L24 label (as in problem 1.b above), the GDB "info isGreater" command will print the author of the isGreater function.

4. (20 points) Consider the following Stack Information:

| Address | Hex Value | Comments |
|---|---|---|
| 0xff968a28 | 0xff968aa8 | |
| 0xff968a24 | 0x00000000 | |
| … | | |
| 0xff9689e4 | 0x08049e00 | |
| 0xff9689e0 | 0x73007369 | "is" |
| 0xff9689dc | 0x00000001 | |
| … | | |
| 0xff9689cc | 0x0804869d | |
| 0xff9689c8 | 0xff968a28 | |
| 0xff9689c4 | 0x00000000 | |
| … | | |
| 0xff968994 | 0x09992018 | -> "this" |
| 0xff968990 | 0xff9689e0 | |
| 0xff96898c | 0x08048806 | |
| 0xff968988 | 0xff9689c8 | %esp, %ebp |

a. Identify the starting and ending addresses of each stack frame in this stack

| Top of Frame | Bottom of Frame |
|---|---|
| 0xff968988 or xff96898c | 0xff988988 |
| 0xff9689c8 or xff9689cc | 0xff96898c |
| 0xff968a28 or xff968a2c | 0xff9689cc |

b. When the currently executing function returns to its caller, what address will it return to?
0x08048806

c. The ASCII value for 'i' is 0x69 and the ASCII value for 's' is 0x73. Was the above stack created from a big-endian or a little-endian machine?
Little Endian (see 0xff9689e0)

d. If the current function above is the "isGreater" function, what are the two parameters passed into the current invocation of that function?
"is" and "this" (0xff9689e0 an 0x09992018)

e. If the above represents main calling insertWord calling isGreater, then which function has a local variable with the value "is"?
main.

5. (30 points) Write a C function to insert a node into a binary tree. A node of the binary tree can be described by the following structure:

```
struct tnode {
        struct tnode * parent;
        int value;
        struct tnode * left;
        struct tnode * right;
};
```

You may use a function whose prototype is “struct tnode * makeTnode(int val)” which allocates space for a new instance of struct tnode, initializes the value field to the value of the val parameter, initializes all pointers to NULL, and returns a pointer to the new instance.

Your C function should have the prototype “void insertTree(int val, struct tnode *root)”, where val is an integer whose value is NOT already in the tree, and root is a pointer to the root node of an existing, non-empty binary tree. The tree has the property that the value of every node in the left sub-tree of any node is smaller than the value in the node itself, and the value in every node in the right sub-tree of a node is larger than the value in the node itself. The root’s parent pointer should be NULL, but every other node’s parent pointer should point at its parent node. Your function should add a new node to the tree for the “val” input, and maintain the ordering of the left and right sub-trees.

Hint: This function is simple if you write it recursively.

```
void insertTree(int val,struct tnode *root) {
        if (val < root->value) {
                if (root->left==NULL) { //Space at root... insert here
                        root->left=makeTnode(val);
                        root->left->parent=root;
                        return;
                }
                insertTree(val,root->left);
                return;
        }
        if (root->right==NULL) { // Space at root... insert here
                root->right=makeTnode(val);
                root->right->parent=root;
                return;
        }
        insertTree(val,root->right);
        return;
}
```