Name:____

Problem #1: Short Answers [20 points] (15 minutes)

a) Pointers and Structure [5 points] (5 minutes)

```
struct node{
int value;
struct node *next; };

struct list{
char foo;
char *ptr;
struct node *listHead;
int *bar;
int size; };
struct list linklist;
```

a.1) [2 points] What is the correct C syntax to access the field value in `struct` node through the variable `linklist`?

a) `linklist.listHead.value`
b) `linklist->listHead->value`
c) `linklist->listHead.value`
d) `linklist.listHead->value`
e) `none of the above`

a.2) [3 points] On a MIPS machine with 32 bit addressing, and every word in memory must be aligned to 4 byte addressing, how many bytes does the compiler allocate for the variable
linklist?
a) 14
b) 16
c) 17
d) 20
e) 24
f) none of the above

b) [5 points] (5 minutes) Consider the following fragment of MIPS.

```
la $a0, buffer             #a
loop:
beq $a0, $a1, continue     #b
lw $t0, 0($a0)             #c
…
addiu $a0, $a0, 4          #d
jal foo                    #e
j loop                     #f
continue:
add $t0, $0, $0
…
```

Of the lines commented [a-f], which of these instructions are not completely determined until linking?

Name:____

c) Executing a program [10 points] (5 minutes)
Following is a list of tasks that must be done to run a program written in C. Sort the tasks into the order in which they are executed and for each step, indicate which tool performs the step (where the tool is the compiler, assembler, linker or loader).

a. Store resulting code in a.s file
b. Combine text and data segments from multiple input files.
c. Store resulting code in an executable file
d. Load text and data segments in to memory.
e. Compute offsets for all branch instructions.
f. Compute absolute address for all relocation table entries; rewrite instruction(s) to hold the new address.
g. Replace pseudoinstructions with one or more real instructions.
h. Translate C code to equivalent assembly language code.
i. Store resulting code in a.o

Write your answers here:

1. Task: ____________________________Tool:____________________________

2. Task: ____________________________Tool:____________________________

3. Task: ____________________________Tool:____________________________

4. Task: ____________________________Tool:____________________________

5. Task: ____________________________Tool:____________________________

6. Task: ____________________________Tool:____________________________

7. Task: ____________________________Tool:____________________________

8. Task: ____________________________Tool:____________________________

9. Task: ____________________________Tool:____________________________

Name:____

Problem #2: Number Representation [15 points] (15 minutes)

a) [5 points] (5 minutes) Take the number -3/16 (base ten) and convert it to 32 bit IEEE 754 floating-point standard (the one discussed in class)

Reminder: A 32 bit floating point number has 1 sign bit, 8 bits for the exponent and 23 bits for the significand.

b) [5 points] (5 minutes) Convert `0x43340000` – a single precision floating point number – to binary scientific notation (e.g., ½ (base 10) would be 1.0 x 2^{-1}).

c) [5 points] (5 minutes)
For the following MIPS assembly language program:

```
loop: addi $t0, $t0, -1
bne $t0, $zero, loop
```

Translate the second instruction into MIPS machine language and write it in hex.

Name:____

Problem #3: C Programming [25 points] (20 minutes)
Write the code for the C function:
*char * insertChar(char *s, char c, int pos)*,
which returns the result of inserting the given character at the given position. *s* is a pointer to the first element of a string and *c* is the character that you should insert at the *pos* element of *s*. You must return the pointer to the first element of a **new** string, which is identical to *s* except with the one additional character inserted at the specified position. You are not allowed to modify the string *s*. You are free to use any of the following functions from string.h library:

*int strlen(char *s)* – returns the length of the string

*char *strncat(char *sl, char *s2, int n)* - The function copies the string `s2`, *not* including its terminating null character, to successive elements of the array of *char* whose first element has the address `s1`. It copies no more than `n` characters from `s2`. The function then stores zero or more null characters in the next elements to be altered in `s1` until it stores a total of `n` characters. It returns `s1`.

*char *strcat(char *s1, char *s2)* - The function copies the string `s2`, including its terminating null character, to successive elements of the array of *char* that stores the string `s1`, beginning with the element that stores the terminating null character of `s1`. It returns `s1`.

Name:____

Problem #4: Assembly Language [25 points] (30 minutes)
On the midterm, you were given the following question:

A list is a series of elements that are sequentially connected to each other. A list node is a structure that represents a single element of a list. A list node is formally defined as follows:

```
struct ListNode {
      int         data;  // this is the data contained in this element
      ListNode*   next;  // this is a pointer to the next element in the list
      ListNode*   prev;  // this is a pointer to the previous element in the list
};
```

A list is simply a series of these nodes connected using pointers.

Write a C function that completely reverses the ordering of the elements in the list. The function should be named `reverse` and should take one argument: a pointer to the first `ListNode` object of the list. It should return a pointer to the new first element of the list (which was previously the last element)

Now, you must write a *recursive* MIPS assembly code for this C function. You must follow register conventions as well as standard procedure calling conventions for full credit on this question. In other words, make no assumptions about the calling procedure.

For you convenience, here is the answer to the midterm question that accomplishes the desired task.

```
ListNode * reverse(ListNode *aNode)
{
      ListNode * tempNode;

      if(aNode == NULL)
            {return NULL;}        //check for incorrect input

      tempNode = aNode->next;     //save the next pointer
      aNode->next = aNode->prev;  //switch the prev and next ptrs
      aNode->prev = tempNode;

      if(aNode->prev == NULL)     //stop if end of list
            { return aNode; }     //    has been reached
      else
            { return Reverse(aNode->prev); } //otherwise continue
                                             //    recursively
}
```

1. Solutions that are not recursive will not get any credit
2. You must follow all register conventions and procedure calling conventions
3. No pseudoinstructions. TAL (true assembly language) only. This means you can only use instructions from the back cover of the textbook.
4. You must write comments. Code that is not adequately commented will be penalized.

Name:____

```
Reverse:
```

Name:____

Problem #5: Decompilation [25 points] (20 minutes)

```
stewie:
      addi $t0, $a2, 1
brian:
      bge $t0, $a1, chris
      mul $t1, $t0, 4
      add $t1, $t1, $a0
      lw $t2, 0($t1)
      sub $t1, $t1, 4
      sw $t2, 0($t1)
      addi $t0, $t0, 1
      j brian
chris:
      jr $ra
```

a) [20 points]

Translate the *stewie* function above into C. You should include a header that lists the types of any arguments and return values. Also, your code should be as concise as possible, without any gotos or explicit pointers. We will not deduct points for syntax errors unless they are significant enough to alter the meaning of your code.

b) [5 points] Describe, in English, what this function computes.

Name:____

Problem #6: Self Modifying Program [15 points] (10 minutes)

The following code illustrates how a program can modify itself. For simplicity, there is only one instruction that is modified by the program, and assumes you run this code on SPIM.

```
1 __start:
2 addiu $t0, $0, 0      # $t0 = 0
3 addiu $t1, $0, 0      # $t1 = 0
4 addiu $t2, $0, 2      # $t2 = 2
5 loop:
6 addiu $t0, $t0, 1     # $t0 = $t0 + 1
7 beq $t0, $t2, next    # if ($t0 == $t2) branch to next
8 jal loop
9 next:
10 addu $t1, $t1, $ra   # $t1 += $ra
11 beq $t1, $0, exit    # if ($t1 == 0) branch to exit
12 lw $t3, 0($t1)       # $t3 = mem[$t1]
13 addiu $t3, $t3, 1    # $t3 += 1
14 sw $t3, 0($t1)       # store word $t3 to memory
15 j next               # jump to next
16 exit:
17 done                 # program finishes
```

6.1 [5 points] Please identify which line is modified by the program?

6.2 [10 points] Write out the new MIPS instruction in TAL for this line.