1. True/False. Circle the appropriate choice (there are no trick questions).

(a) T F In C, a pointer variable is used for storing an address.
(b) T F In x86 assembly, the eax register can be used to hold integers but not addresses.
(c) T F In x86 assembly, the instruction “shl eax,3” (or “shl $3,%eax” in AT&T syntax) multiplies the value in eax by 8.
(d) T F If, in C, an array is declared by “int a[10];”, then accessing a[11] during execution will generate an error message.
(e) T F A callee-saved register should be saved before it is written to in a procedure.
(f) T F C has no built-in boolean type, rather the value 0 is used to represent false and all other values represent true.
(g) T F ebp is a caller-saved register.
(h) T F In x86 assembly, local variables in a procedure are generally declared in the .data section.
(i) T F Compiling C into x86 assembly comprises translating each C statement into a single assembly instruction.
(j) T F In x86 assembly, a label corresponds to an address.

2. Answer this question on this sheet

Consider the following x86 code fragment for computing the sum of an array of 10 32-bit integers, pointed to by ecx.

```
#Intel Syntax               #AT&T Syntax
mov eax,0                  mov $0,%eax  #
mov edx,0                  mov $0,%edx  #
TOP:                        TOP:
cmp edx,10                 cmp $10,%edx  #
jl OUT                     jl OUT      #
add eax,[ecx+edx]          add (%ecx,%edx,)%eax  #
inc edx                   inc %edx      #
jmp TOP                   jmp TOP      #
OUT:                       OUT:
```

(a) There are two bugs in the code. What are they?

(b) Put a comment following each “#” in the above code to describe what that instruction does (or is supposed to do, in the case of a bug).

3. Given the following C procedure
int sum(int *a, int num)
{
    int i;
    int sum = 0;
    for(i=0; i<num; i++)
        sum = sum + a[i];
    return sum;
}

using the appropriate x86 addressing modes, fill in the missing code in the assembly procedure below that corresponds to the procedure `sum()`, above. Choose either Intel or AT&T syntax.

```assembly
._sum:
    push ebp
    mov ebp,esp

    mov ecx,[ebp+8]  #ecx holds a
    mov edx,[ebp+12] #edx holds num
    mov eax,0        #eax holds sum
    mov ebx,0        #ebx holds i

    TOP:
        jge OUT        #jump out if i >= num
        inc ebx        #increment i
        jmp TOP        #jump to top of loop

    OUT:
        .....

    pop ebp
    ret
```

4. (a) Define a C struct type `CELL` that contains the following fields: an integer `x`, a string `y`, and a `next` field that points to another structure of type `CELL`.

(b) Write in C a procedure corresponding to the declaration
   ```c
   int list_length(CELL *head);
   ```
   that returns the length of a linked list whose first element is pointed to by `head`. 
5. A simple C procedure for computing the integer square root of a number (i.e. the greatest integer less than or equal to the square root of the number) is shown below (where “>>” is the shift-right operator):

```c
int square_rt(int x, int low, int high)
{
    if (low >= high - 1)
        return low;
    int mid = (low + high) >> 1;
    if ((mid * mid) > x)
        return square_rt(x,low,mid);
    else
        return square_rt(x,mid,high);
}
```

where the initial call to square_rt would be square_rt(n,1,n). Translate square_rt into x86 assembly, so that it could be called from C.