Nodes.c

```
struct Node {
      void *data;
      struct Node *left, *right;
      };

/* do the entire function in assembler */
long count(struct Node *ptr)
{
      long rval = 0;

      if(ptr)
      {
          rval = 1;    /* count the node itself */
          /* add the child counts */
          rval += count(ptr->left);
          rval += count(ptr->right);
      }
      return rval;
}

count:                        # points off if you forget he labe when I
                        # ask for a funciton
      pushq %rbp
      movq %rsp, %rbp         # stack frame will be required for functions
                        # assumed to be there for other questions

      xorq  %rax, %rax  # in case I don't have a node
      testq %rdi, %rdi  # is my pointer zero?
      jz    done        # no pointer, no work to do

      pushq %rbx        # we make function calls and have 2 values that
      pushq %r12        # need to survive the calls

      movq  %rdi, %rbx  # survive the funciton call pointer
      movq  $1, %r12    # count the current node


      movq  8(%rbx), %rdi     # go to memory for the pointer
      call count        # left count
      addq %rax, %r12         # add the left count


      movq 16(%rbx), %rdi     # go to memory for right pointer
      call count        # right count
      addq %r12, %rax         # no more calls, put sum where I really want
it


      popq %r12         # make sure the order is right
      popq  %rbx

done:
      leave             # required for complete functions, not for
```

```
                        # single lines fo code
      ret
```

P7

```
long fx( long p1, long p2, long p3,
      long p4, long p5, long p6,
      long p7)
{
      long rval;

      /* render the following line of C in assembler */
      rval = p1 + p7;

      /*
      movq %rdi, %rax         # rax = p1
      addq 16(%rbp), %rax     # add p7 to that
      */

      return rval;
}
```

Ptr

```
/* be able to get into and out of memory with a pointer */

void inc(int *ip)
{

      /* think this one over carefully, it hides a bunch of stuff */
      /* just do this one line */
      *ip += 1;

      /*
      addl $1, (%rdi)
      */

}
```

Structs

```
/* given a struct, get any element or the address of any element
** This means you have to know how structs are laid out, alignment,
** padding etc. Lay it out in class with length and offsets*/

struct Record {
    char name[15];
    short scores[2][6];
    int win, loss, tie;
};

void fx( struct Record *ptr)
{

      int i, *ip;
      short *sp;

      i = ptr->win;

      /*
      movl 40(%rdi), %eax
      */

      ip = &ptr->tie;

      /*
      leaq 48(%rdi), %rax
      */

      /* be able to do these, they are more involved */
      i = ptr->scores[0][3];
      sp = &ptr->scores[1][1];
      /* do these with variables in the subscripts as well! */

}
```

While

```
void wtest( char *bytePointer, long count)
{
      /* render the entire while loop in assembler */
      while(count)
      {
            count--;
            bytePointer[count] = 0;
      }


      /*
loop:
      testq %rsi, %rsi
      jz done

      decq %rsi
      movb $0, (%rdi, %rsi)

      jmp loop
done:
      */
}
```

Can also do with a jump to the test and the test at the end for a shorter loop (only one jump) – something useful for L6 bonus