Assignment for Lab 10
Data Structure Representations
Computer Science 240

In lab this week, you will write some assembly language programs to study how data structures are stored in memory. To investigate this concept, it is useful to write some X86 assembly code directly (rather than producing it by compiling C code, as we have been doing up to now). Below on the left is an example of a simple C program, and on the right is the corresponding X86 program which performs the equivalent task.

1. Now that you have had some experience with disassembly, it should be fairly straightforward to correlate the C code below to the X86 code. Draw some lines between the two versions of the program below and make some notes to indicate how the C code is implemented in X86.

simple.c: (C code)

```
#include <stdio.h>
int total = 0;

int sum(int x,int y)
{
  int t = x + y;
  total +=t;
  return t;
}

int main()
{
   int x = 2;
   int y = 3;
   printf("Sum = %d\n",sum(x,y));
   printf("Total =  %d\n",total);
   return 0;
}
```

simple.s: (X86 code)

```
          .data
         .globl  total
total:    .long  0
fstr1:    .string  "Sum =  %d\n"
fstr2:     .string  "Total = %d\n"



          .text
          .global main
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $32, %esp
        movl    $2, -4(%ebp)
        movl    $3, -8(%ebp)
        movl   -8(%ebp),%edx
        movl   %edx,4(%esp)
        movl    -4(%ebp),%edx
        movl  %edx,(%esp)
        call    sum
        movl    $fstr1, %edx
        movl    %eax, 4(%esp)
        movl    %edx, (%esp)
        call    printf
        movl    $fstr2, %edx
        movl   $total,%eax
        movl    (%eax),%eax
       movel    %eax, 4(%esp)
        movl    %edx, (%esp)
        call    printf
        movl    $0, %eax
        leave
        ret
```

```
sum:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    12(%ebp), %eax
        movl    8(%ebp), %edx
        leal    (%edx,%eax), %eax
        movl    %eax, -4(%ebp)
        movl    $total,%edx
        movl   (%edx),%eax
        addl    -4(%ebp), %eax
        movl    %eax, (%edx)
        movl    -4(%ebp), %eax
        leave
        ret
```

NOTE: There are some lines you probably do not completely understand in the X86 code. The lines beginning with periods, like ".data", ".long", or ".string" are *assembler directives* -- commands that tell the assembler how to assemble the file. The lines beginning with some text followed by a colon, like "main:", are labels, or named locations in the code. To see a list of possible directives, visit: http://tigcc.ticalc.org/doc/gnuasm.html#SEC67

2. Using the previous program as a guide, write an X86 program which implements the following C program (do NOT use the computer to compile the C program and produce the X86 code; instead, write it from scratch).

simple1.c: (C code)

```
#include <stdio.h>
int z;

int square(int n)
{
        return n*n;
}

int main()
{
        int x = square(3);
        int y = square(4);
        z = x + y;
        printf("Calculation produces %d\n",z);
}
```