CSci 2021, Spring 2015
Homework Assignment II
Due: Friday, February 27th 2015, at beginning of lecture

Problem 0: (1 point)

Clearly label your assignment with the time of your recitation section (8:00, 9:05, 10:10, 11:15, 12:20, 1:25, 2:30). This will help us turn back your graded assignments more efficiently.

Problem 1:

Textbook problem 3.57 (p. 296). Hints: design your rewritten function so that it *always* performs a dereference. If GCC is not producing a conditional move when you expect it to, try varying whether any extra variables are local or global.

Problem 2:

Consider the following assembly code for a function with a loop:

```
prob2:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        cmpl    $1, %eax
        je      .L1
.L6:
        testb   $1, %al
        je      .L3
        leal    1(%eax,%eax,2), %eax
        jmp     .L4
.L3:
        shrl    %eax
.L4:
        cmpl    $1, %eax
        jne     .L6
.L1:
        popl    %ebp
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may only use the source-level C variable `n`: don't use register names!

```
void prob2(unsigned n)
{
    while (________________) {

        if (_______________) {

            ______________;
        } else {

            ________________;
        }
    }
}
```

Problem 3:

Consider the following C code, and the corresponding assembly code produced by a C compiler:

```
#define SIZE 10                         prob3:
void prob3(int mat[SIZE][SIZE]) {               pushl   %ebp
    int r, c;                                   movl    %esp, %ebp
                                                pushl   %edi
                                                pushl   %esi
                                                pushl   %ebx
                                                subl    $4, %esp
                                                movl    8(%ebp), %eax
    mat[0][0] = 1;                              movl    $1, (%eax)
                                                leal    40(%eax), %ebx
                                                movl    $1, 40(%eax)
                                                addl    $80, %eax
                                                movl    %eax, -16(%ebp)
                                                movl    $1, %edi
                                                jmp     .L2
    for (r = 1; r < SIZE; r++) {        .L5:
                                                movl    -16(%ebp), %ebx
        mat[r][0] = 1;                          movl    $1, (%ebx)
                                                cmpl    $1, %edi
                                                jle     .L3
                                                movl    %ebx, %edx
                                                subl    $40, %edx
                                                leal    -1(%edi), %esi
                                                movl    $0, %eax
        for (c = 1; c < r; c++) {       .L4:
            mat[r][c] = mat[r-1][c]             movl    4(%edx,%eax,4), %ecx
                                                addl    (%edx,%eax,4), %ecx
                   + mat[r-1][c-1];             movl    %ecx, 4(%ebx,%eax,4)
                                                addl    $1, %eax
                                                cmpl    %esi, %eax
        }                                       jne     .L4
                                        .L3:
                                                addl    $40, -16(%ebp)
                                        .L2:
        mat[r][r] = 1;                          movl    $1, (%ebx,%edi,4)
                                                addl    $1, %edi
                                                cmpl    $10, %edi
    }                                           jne     .L5
                                                addl    $4, %esp
                                                popl    %ebx
                                                popl    %esi
                                                popl    %edi
                                                popl    %ebp
}                                               ret
```

Because the compiler has optimized some of the accesses to the array, the registers don't all correspond exactly to variables in the source code. (And the statements and instructions don't line up exactly one-to-one either, so don't put too much significance in the way we've spaced the lines.) For each of the following registers, as it is used in a particular range of instructions (shown by their assembly code line number), write a C expression that corresponds to the value in the register. Your expressions should be written using the C variables `mat`, `r`, and `c`, together with C operators and constants; don't use register names.

Register	C Expression
`%eax`, lines 8-11	
`%edi`, lines 14-37	
`%ebx`, lines 10-35	
`%edx`, lines 22-27	
`%esi`, lines 23-30	
`%eax`, lines 23-30	

Problem 4:

Textbook problem 3.68 (p. 306).

Problem 5: (based on textbook problem 3.69)

The following function declaration defines a class of structures for use in constructing binary trees:

```
typedef struct ELE *tree_ptr;

struct ELE {
    int val;
    tree_ptr left;
    tree_ptr right;
};
```

For a function with the prototype `int trace(tree_ptr tp);`, GCC generates the following IA32 code:

```
trace:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %edx
        movl    $0, %eax
        testl   %edx, %edx
        je      .L2
.L5:
        movl    (%edx), %eax
        movl    8(%edx), %edx
        testl   %edx, %edx
        jne     .L5
.L2:
        popl    %ebp
        ret
```

A. Generate a C version of the function, using a `while` loop.

B. Explain in English what this function computes.

Problem 6:

Consider the following datatype definitions on an IA32 (x86) machine.

```
typedef struct {                typedef union {
  short s;                        short s;
  double *p;                      double *p;
  int i;                          int i;
  char c;                         char c;
  int a[2];                       int a[2];
} struct1;                      } union1;
```

A. Using the template below (allowing a maximum of 32 bytes), indicate the allocation of data for a structure of type `struct1`. Mark off and label the areas for each individual element (there are 5 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment).

Assume the alignment rules discussed in lecture: primitive data values of size x must be aligned on x-byte boundaries. **Clearly indicate the right hand boundary of the data structure with a vertical line**.

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                                                                                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

B. How many bytes are allocated for an object of type `struct1`?

C. What alignment is required for an object of type `struct1`? (If an object must be aligned on an x-byte boundary, then your answer should be x.)

D. If we define the fields of `struct1` in a different order, we can reduce the number of bytes wasted by each variable of type `struct1`. What is the number of **unused, allocated** bytes in the best case?

E. How many bytes are allocated for an object of type `union1`?

F. What alignment is required for an object of type `union1`? (If an object must be aligned on an x-byte boundary, then your answer should be x.)

Problems 0, 2, and 6 should be submitted for grading.