Overview These problems are representative of the types of exam problems you’ll see next week. The number of problems here, and the difficult level of the problems are what you can expect in Exam 1. I did not include any problems from the book on this study guide. However, you are highly encouraged to study the homework problems in the book for the relevant sections.

We’ll go over these problems in class next Monday.
This first problem will test your understanding of stack frames. It is based on the following recursive C function:

```c
int silly(int n, int *p)
{
    int val, val2;

    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;

    *p = val + val2 + n;

    return val + val2;
}
```

This yields the following machine code:

```
silly:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
pushl %ebx
movl 8(%ebp),%ebx
testl %ebx,%ebx
jle .L3
addl $-8,%esp
leal -4(%ebp),%eax
pushl %eax
leal (%ebx,%ebx),%eax
pushl %eax
call silly
jmp .L4
.p2align 4,,7
.L3:
xorl %eax,%eax
movl %eax,-4(%ebp)
.L4:
movl -4(%ebp),%edx
addl %eax,%edx
movl 12(%ebp),%eax
addl %edx,%ebx
movl %ebx,(%eax)
movl -24(%ebp),%ebx
movl %edx,%eax
movl %ebp,%esp
popl %ebp
ret
```
Problem 1. (25 points):

A. Is the variable val stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?

B. Is the variable val2 stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?

C. What (if anything) is stored at $-24 (%ebp)$? If something is stored there, why is it necessary to store it?

D. What (if anything) is stored at $-8 (%ebp)$? If something is stored there, why is it necessary to store it?
Problem 2. (15 points):
Consider the following assembly code for a C for loop:

```assembly
loop:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%ecx
    movl 12(%ebp),%edx
    xorl %eax,%eax
    cmpl %edx,%ecx
    jle .L4
.L6:
    decl %ecx
    incl %edx
    incl %eax
    cmpl %edx,%ecx
    jg .L6
.L4:
    incl %eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables \( x \), \( y \), and \( \text{result} \) in your expressions below — do not use register names.)

```c
int loop(int x, int y)
{
    int result;
    for (____________; __________; result++ ) {
        __________;
        __________;
    }
    __________;
    return result;
}
```
Problem 3. (10 points):
Consider the following datatype definitions on an IA32 (x86) machine running Linux.

```c
typedef struct {
    char c;
    int i;
    char *b;
    short s;
    double d;
} struct1;

typedef union {
    char c;
    double *p;
    int i;
    double d;
    short *s;
} 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 class: data types of size $x$ must be aligned on $x$-byte boundaries. Clearly indicate the right hand boundary of the data structure with a vertical line.

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 $x$.)
Consider the following C program `prog.c`

```c
main( int argc, char * argv[] )
{
    char **p = argv;
    int *i = ((int *)&argv)-1;

    while( (*i)-- )
        printf( "%s%s", *(p++), *i ? "":"" );
    putchar( '\n' );
}
```

which is compiled on an Intel/32 based Linux machine using `cc -S prog.c` to yield the following `prog.s`:

```
.file    "prog.c"
.version  "01.01"
gcc2_compiled.:
.section    .rodata
.LC0:
    .string    " "
.LC1:
    .string    ""
.LC2:
    .string    "%s%s"
.text
   .align  4
.globl   main
   .type    main,@function
main:
   pushl  %ebp
   movl   %esp,%ebp
   subl   $24,%esp
   movl   12(%ebp),%eax
   movl   %eax,-4(%ebp)
   leal   12(%ebp),%eax
   leal   -4(%eax),%edx
   movl   %edx,-8(%ebp)
   .p2align 4,,7
.L3:
   movl   -8(%ebp),%eax
   decl   (%eax)
   cmpl    $-1,(%eax)
jne    .L5
jmp    .L4
   .p2align 4,,7
```

Page 6 of 10
Problem 4. (15 points):

For each of the four (4) variables, argc, argv, p, i used in prog.c, label the first place it is used in prog.s by writing the variable name (e.g., i next to this first use on the listing above.)
Problem 5. (10 points):
Consider the following short “C” procedure:

```c
strcpy( char *d, char *s )
{
    while( *d++ = *s++ )
    ;
}
```

Write an equivalent procedure which uses a `do while` loop instead of the `while` loop used here.
Problem 6. (10 points):

Consider the following C functions and assembly code:

```c
int fun4(int *ap, int *bp)
{
    int a = *ap;
    int b = *bp;
    return a+b;
}

int fun5(int *ap, int *bp)
{
    int b = *bp;
    *bp += *ap;
    return b;
}

int fun6(int *ap, int *bp)
{
    int a = *ap;
    *bp += *ap;
    return a;
}
```

Which of the functions compiled into the assembly code shown?
Problem 7. (15 points):

In the following questions assume the variables a and b are signed integers and that the machine uses two’s complement representation. Also assume that MAX_INT is the maximum integer, MIN_INT is the minimum integer, and W is one less than the word length (e.g., W = 31 for 32-bit integers).

Match each of the descriptions on the left with a line of code on the right (write in the letter). You will be given 2 points for each correct match.

1. One’s complement of a
   - a. \( \text{\sim} (\text{\sim} a \mid (b \text{\sim} (\text{MIN\_INT} + \text{MAX\_INT}))) \)
   - b. \( ((a \text{\^} b) \& \text{\sim} b) \mid (\text{\sim} (a \text{\^} b) \& b) \)
   - c. \( 1 + (a \ll 3) + \text{\sim} a \)

2. a.
   - d. \( (a \ll 4) + (a \ll 2) + (a \ll 1) \)
   - e. \( ((a < 0) ? (a + 3) : a) \gg 2 \)

3. a \& b.
   - f. \( a \text{\^} (\text{MIN\_INT} + \text{MAX\_INT}) \)
   - g. \( \text{\sim} ((a \mid (\text{\sim} a + 1)) \gg W) \& 1 \)

4. a * 7.
   - h. \( \text{\sim} ((a \gg W) \ll 1) \)

5. a / 4.
   - i. \( a \gg 2 \)

6. (a < 0) ? 1 : -1.
   - a. \( a \gg 2 \)