Andrew login ID: .....

Full Name: .....

CS 15-213, Fall 2001

Final Exam

December 13, 2001

Instructions:

Make sure that your exam is not missing any sheets, then write your full name and Andrew login ID on the front.

Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.

The exam has a maximum score of 120 points.

This exam is OPEN BOOK. You may use any books or notes you like. You may use a calculator, but no laptops or other wireless devices. Good luck!

1 (20):	
2 (10):	
3 (10):	
4 (8):	
5 (12):	
6 (6):	
7 (14):	
8 (10):	
9 (16):	
10 (14):	
TOTAL (120):	

Problem 1. (20 points):

We are running programs on a machine with the following characteristics:

> Values of type `int` are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type `unsigned` are 32 bits.
>
> Values of type `float` are represented using the 32-bit IEEE floating point format, while values of type `double` use the 64-bit IEEE floating point format.

We generate arbitrary values `x`, `y`, and `z`, and convert them to other forms as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

For each of the following C expressions, you are to indicate whether or not the expression *always* yields 1. If so, circle "Y". If not, circle "N". You will be graded on each problem as follows:

> If you circle no value, you get 0 points.
>
> If you circle the right value, you get 2 points.
>
> If you circle the wrong value, you get points (so don't just guess wildly).

Expression	Always True?
`(x<y) == (-x>-y)`	Y N
`((x+y)<<4) + y-x == 17*y+15*x`	Y N
`~x+~y+1 == ~(x+y)`	Y N
`ux-uy == -(y-x)`	Y N
`(x >= 0) || (x < ux)`	Y N
`((x >> 1) << 1) <= x`	Y N
`(double)(float) x == (double) x`	Y N
`dx + dy == (double) (y+x)`	Y N
`dx + dy + dz == dz + dy + dx`	Y N
`dx * dy * dz == dz * dy * dx`	Y N

Problem 2. (10 points):

A C function `looper` and the assembly code it compiles to on an IA-32 machine running Linux/GAS is shown below:

```
looper:
  pushl %ebp
  movl %esp,%ebp
  pushl %esi
  pushl %ebx
  movl 8(%ebp),%ebx
  movl 12(%ebp),%esi
  xorl %edx,%edx
  xorl %ecx,%ecx
  cmpl %ebx,%edx
  jge .L25
.L27:
  movl (%esi,%ecx,4),%eax
  cmpl %edx,%eax
  jle .L28
  movl %eax,%edx
.L28:
  incl %edx
  incl %ecx
  cmpl %ebx,%ecx
  jl .L27
.L25:
  movl %edx,%eax
  popl %ebx
  popl %esi
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int looper(int n, int *a)
  { int i;
  int x = _____;

  for(i = _____;

      _____;

      i++) {

    if (_____)

       x = _____;


    _____;
  }

  return x;
}
```

Based on the assembly code, fill in the blanks in the C source code.

Notes:

You may only use the C variable names `n`, `a`, `i` and `x`, not register names.

Use array notation in showing accesses or updates to elements of `a`.

Problem 3. (10 points):

Consider the following incomplete definition of a C struct along with the incomplete code for a function `func` given below.

```
typedef struct node {

  ____________________x;

  ____________________y;

  struct node *next;

  struct node *prev;

} node_t;
```

```
node_t n;

void func() {

  node_t *m;

  m = ____________________________;

  m->y /= 16;

  return;
}
```

When this C code was compiled on an IA-32 machine running Linux, the following assembly code was generated for function `func`.

```
func:
  pushl %ebp
  movl n+12,%eax
  movl 16(%eax),%eax
  movl %esp,%ebp
  movl %ebp,%esp
  shrw $4,8(%eax)
  popl %ebp
  ret
```

Given these code fragments, fill in the blanks in the C code given above. Note that there is a unique answer.

The types must be chosen from the following table, assuming the sizes and alignment given.

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
double	8	4

Problem 4. (8 points):

Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int array1[M][N];
int array2[N][M];

void copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code:

```
copy:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ecx
  movl 12(%ebp),%eax
  leal 0(,%eax,4),%ebx
  leal 0(,%ecx,8),%edx
  subl %ecx,%edx
  addl %ebx,%eax
  sall $2,%eax
  movl array2(%eax,%ecx,4),%eax
  movl %eax,array1(%ebx,%edx,4)
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

What are the values of `M` and `N`?

`M` =

`N` =