Andrew login ID:____________________________________

Full Name:____________________________________

CS 15-213, Fall 2002

Final Exam

December 17, 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 105 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 (08):
3 (04):
4 (10):
5 (08):
6 (12):
7 (09):
8 (16):
9 (08):
10 (10):
TOTAL (105):

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();
/* Convert to other forms */
unsigned ux = (unsigned) x;
double   dx = (double) x;
```

For each of the following C-like expressions, you are to indicate whether or not the expression *always* yields 1. For expressions that include an implication operator `=>` indicate whether or not the right-hand expression yields 1 for all values that satisfy the left-hand expression. 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 −1 points (so don't just guess wildly). The minimum total score for this entire problem is 0.

Expression	Always True?
`x < 0 => (x*3) < 0`	Y N
`x < 0 => (dx*dx*dx < 0.0)`	Y N
`((y-x)<<3) + x-(2*y) == 6*y - 7*x`	Y N
`((x-y)<<3) + (x>>1) - y == 8*x - 9*y + x/2`	Y N
`(x-y) > 0 => -x < -y`	Y N
`ux - x == 0`	Y N
`x > 0 => !( x >> 31 | !x) == 1`	Y N
`((ux >> 1) << 1) == ux`	Y N
`dx / 3.0 == x / 3`	Y N
`x - (int)dx == 0`	Y N

Problem 2. (8 points):

This problem tests your understanding of casting and pointer de-referencing.

Consider the following code, being executed on a Little Endian Pentium machine where

- `sizeof(int) == 4`
- `sizeof(int *) == 4`
- `sizeof(char) == 1`

For each of the following assignment statements, fill in the blanks in the comments to indicate the result of the assignment. All answers must be in hex.

```
int main() {
    int array[2];
    int * ptr;
    int x;
    char c;

    array[0] = 0xaabbccdd;
    array[1] = 0x44556677;

    ptr = array;

    x = *((int *)ptr + 1);

    /* x = 0x____________________*/

    c = *((char *)ptr + 1);

    /* c = 0x____________________*/

    x = *((char *)ptr + 1);

    /* x = 0x____________________*/

    c = *((int *)ptr + 1);

    /* c = 0x____________________*/

 }
```

Problem 3. (4 points):

This problem concerns the indexing of C arrays.

Consider the C code below, where `N` is a constant declared with `#define`.

```
int foo (int A[16][N], int i, int j)
{
    return A[i][j];
}
```

Suppose the above C code generates the following assembly code:

```
foo:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  movl 12(%ebp),%edx
  movl 16(%ebp),%ecx
  sall $2,%ecx
  leal (%edx,%edx,4),%edx
  leal (%ecx,%edx,8),%edx
  movl %ebp,%esp
  popl %ebp
  movl (%eax,%edx),%eax
  ret
```

What is the value of `N`?

`N` =

Problem 4. (10 points):

This problem tests your understanding of C structures and pointers.

```
typedef char mystr[8];

typedef struct {
    char **strs;
    int num;
} s1;

typedef struct {
    char *str;
    int len;
} s2;



int func1(char *x, int i)
{
    return strlen (&(x[i]));
}


int func2(char **x,  int i)
{
    return strlen (x[i]);
}


int func3(mystr x[], int i)
{
    return strlen (x[i]);
}


int func4(s1 *x, int i)
{
    return strlen (x->strs[i]);
}


int func5(s2 *x, int i)
{
    return strlen (x[i].str);
}
```

A. Which C function corresponds to the assembly code shown below?

`func________`

```
pushl %ebp
movl %esp,%ebp
subl $8,%esp
movl 8(%ebp),%edx
movl 12(%ebp),%eax
addl $-12,%esp
movl (%edx,%eax,4),%eax
pushl %eax
call strlen
movl %ebp,%esp
popl %ebp
ret
```

B. Which C function corresponds to the assembly code shown below?

`func________`

```
pushl %ebp
movl %esp,%ebp
subl $8,%esp
movl 8(%ebp),%edx
movl 12(%ebp),%eax
addl $-12,%esp
movl (%edx,%eax,8),%eax
pushl %eax
call strlen
movl %ebp,%esp
popl %ebp
ret
```

Problem 5. (8 points):

This problem tests your understanding of how recursion in C relates to IA32 machine code.

Consider the following IA32 assembly code for a procedure `foo()`:

```
foo:
  pushl %ebp
  movl %esp,%ebp
  subl $12,%esp
  pushl %edi
  pushl %esi
  pushl %ebx
  movl 8(%ebp),%edi
  movl 12(%ebp),%esi
  testl %edi,%edi
  jne .L3
  leal 1(%esi),%eax
  jmp .L6
.L3:
  testl %esi,%esi
  je .L4
  addl $-8,%esp
  pushl %esi
  leal -1(%edi),%eax
  pushl %eax
  call foo
  movl %eax,%ebx
  addl $-8,%esp
  leal -1(%esi),%eax
  pushl %eax
  pushl %edi
  call foo
  imull %eax,%ebx
  movl %ebx,%eax
  jmp .L5
.L4:
  leal 1(%edi),%eax
.L6:
.L5:
  leal -24(%ebp),%esp
  popl %ebx
  popl %esi
  popl %edi
  movl %ebp,%esp
  popl %ebp
  ret
```

Based on the assembly code, fill in the blanks below in its corresponding C source code. (Note: you may only use symbolic variables `x` and `y` from the source code in your expressions below — do *not* use register names.)

```
int foo(int x, int y)
{
  if (______)

    return ______;

  if (______)

    return ______;

  else

    return foo(______,______) * foo(_____,______);

}
```

Problem 6. (12 points):

Suppose the following code

```
/* N is a compile-time constant */
int a[N][N];

/* Sum first 4 columns in a */
int sum4col()
{
    int i,j;
    int sum = 0;
    for (j = 0; j < 4; j++)
        for (i = 0; i < N; i++)
            sum += a[i][j];
    return sum;
}
```

is compiled and executed on a machine with the following characteristics:

- `sizeof(int)` is 4.
- The cache is 512 bytes, direct mapped, with 16-byte blocks.
- Array `a` starts at address `0x800000`.
- During the execution of the inner two loops, the only accesses to the memory or data cache are to read the elements of array `a`.

With `i = 0` and `j = 0`, a block will be read into the cache containing array elements `a[0][0]`, `a[0][1]`, `a[0][2]`, and `a[0][3]`.

Fill in the table below, showing for different values of `N` whether this block will be evicted from the cache during the execution of `sum4col` (answer “Y” or “N”), and if so, what will be the values of `i` and `j` that cause this eviction (leave this blank when no eviction occurs).

N	Evicted? (Y/N)	i	j
16			
12			
14			
10			

Problem 7. (9 points):

This problem tests your understanding of Unix process control. Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

Part 1

```
int main () {
    if (fork() == 0) {
        if (fork() == 0) {
            printf("9");
            exit(1);
        }
        else
            printf("5");
    }
    else {
        pid_t pid;
        if ((pid = wait(NULL)) > 0) {
             printf("3");
        }
    }
    printf("0");
    return 0;
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program.

A.	93050	Y	N
B.	53090	Y	N
C.	50930	Y	N
D.	39500	Y	N
E.	59300	Y	N

Part 2

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
int i = 0;

int main () {
    int j;
    pid_t pid;

    if ((pid = fork()) == 0) {
        for (j = 0; j < 20; j++)
            i++;
    }
    else {
        wait(NULL);
        i = -1;
    }

    if (i < 0)
        i = 10;

    if (pid > 0)
        printf("Parent: i = %d\n", i);
    else
        printf("Child: i = %d\n", i);

    exit(0);
}
```

What are the outputs of the two `printf` statements?

`Parent: i = __________`

`Child : i = __________`

Problem 8. (16 points):

Consider the following four different designs for a memory allocator:

IL (Implicit List). The free list is represented implicitly. Each block has a header, but no footer. Coalescing of the blocks is deferred to occur during block allocation.

ILBT (Implicit List with Boundary Tags). The free list is represented implicitly. Each block has a header and an identical boundary-tag footer. Coalescing of the blocks occurs as part of the free operation.

E1LBT (Explict List with Boundary Tags, Singly-Linked). The free list is represented explicitly as a singly-linked list. Each block has a header and an identical boundary-tag footer. Coalescing of the blocks occurs as part of the free operation. A newly freed block is placed at the head of the free list.

E2LBT (Explict List with Boundary Tags, Doubly-Linked). The free list is represented explicitly as a doubly-linked list. Each block has a header and an identical boundary-tag footer. Coalescing of the blocks occurs as part of the free operation. A newly freed block is placed at the head of the free list.

Define the following operations. Here `bp` denotes some pointer to the beginning of the payload section of a block, while `p` denotes an arbitrary pointer.

`malloc(cnt)` Allocate a block of `cnt` bytes. You may assume that sufficient space is available.

`free(bp)` Free an allocated block

`isRightFree(bp)` Determine whether the block with the next higher address is free.

`isLeftFree(bp)` Determine whether the block with the next lower address is free.

`isValidAddress(p)` Determine whether `p` points to a word within an allocated block.

Assume at some point in the program execution that there are m allocated blocks and n free blocks. Fill in the table showing the *worst-case asymptotic performance* (big-Oh notation) for the optimum implementation of each of the following operations. Example entries are $O(1)$ (constant time), $O(m)$, $O(m+n)$, etc.

Operation	IL	ILBT	E1LBT	E2LBT
`malloc(cnt)`				
`free(bp)`				
`isRightFree(bp)`				
`isLeftFree(bp)`				
`isValidAddress(p)`				

Problem 9. (8 points):

This problem tests your understanding of how the Unix kernel represents open files, and how files are shared.

There are four questions to answer for this problem. You can assume that the C programs in all questions are executed by a standard Unix shell. The `O_RDONLY` and `O_WRONLY` flags are used by the `Open` function to open files read-only and write-only, respectively.

Part 1

Consider the following C program:

```
int main()
{
    int fd1, fd2, fd3;

    fd1 = Open("foo.txt", O_RDONLY, 0);
    fd2 = Open("bar.txt", O_RDONLY, 0);
    Close(fd1);
    fd3 = Open("baz.txt", O_RDONLY, 0);
    printf("fd3 = %d", fd3);
    exit(0);
}
```

The output is: `fd3 =` ___________

Part 2

Consider the following C program, where the disk file `barfoo.txt` consists of the 6 ASCII characters "`barfoo`".

```
int main()
{
    int fd1, fd2;
    char c;

    fd1 = Open("barfoo.txt", O_RDONLY, 0);
    fd2 = Open("barfoo.txt", O_WRONLY, 0);
    Read(fd1, &c, 1);
    c = 'z';
    Write(fd2, &c, 1);
    Read(fd1, &c, 1);
    printf("c = %c", c);
    exit(0);
}
```

The output is: `c =` ___________

Part 3

Consider the following C program, where the disk file `barfoo.txt` consists of the 6 ASCII characters “`barfoo`”.

```
int main()
{
    int fd;
    char c;

    fd = Open("barfoo.txt", O_RDONLY, 0);
    Read(fd, &c, 1);
    if (Fork() == 0) {
       Read(fd, &c, 1);
       exit(0);
    }
    Wait(NULL);  /* wait for child to terminate */
    Read(fd, &c, 1);
    printf("c = %c", c);
    exit(0);
}
```

The output is: `c =` ___________

Part 4

Consider the following C program, where the disk file `barfoo.txt` consists of the 6 ASCII characters “`barfoo`”. Recall that the `dup2(int srcfd, int dstfd)` function copies descriptor table entry `srcfd` to descriptor table entry `dstfd`.

```
int main()
{
    int fd1, fd2;
    char c;

    fd1 = Open("barfoo.txt", O_RDONLY, 0);
    fd2 = Open("barfoo.txt", O_RDONLY, 0);
    Read(fd1, &c, 1);
    Dup2(fd2, fd1);
    Read(fd1, &c, 1);
    printf("c = %c", c);
    exit(0);
}
```

The output is: `c =` ___________

Problem 10. (10 points):

This problem tests your understanding of race conditions in concurrent programs.

Consider a simple concurrent program with the following specification: The main thread creates two peer threads, passing each peer thread a unique integer *thread ID* (either 0 or 1), and then waits for each thread to terminate. Each peer thread prints its thread ID and then terminates.

Each of the following programs attempts to implement this specification. However, some are incorrect because they contain a race on the value of `myid` that makes it possible for one or more peer threads to print an incorrect thread ID. Except for the race, each program is otherwise correct.

You are to indicate whether or not each of the following programs contains such a race on the value of `myid`. You will be graded on each subproblem follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 2 points.
- If you circle the wrong answer, you get -1 points (so don't just guess wildly).

A. **Does the following program contain a race on the value of `myid`?** `Yes` `No`

```
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    Free(vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i, *ptr;

    for (i = 0; i < 2; i++) {
        ptr = Malloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], 0, foo, ptr);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

B. **Does the following program contain a race on the value of `myid`?** Yes No

```
void *foo(void *vargp) {
    int id;
    id = *((int *)vargp);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], NULL, foo, &i);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

C. **Does the following program contain a race on the value of `myid`?** Yes No

```
void *foo(void *vargp) {
    int id;
    id = (int)vargp;
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], 0, foo, i);
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

D. **Does the following program contain a race on the value of `myid`?** Yes No

```
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int id;
    id = *((int *)vargp);
    V(&s);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    sem_init(&s, 0, 0); /* S=0 INITIALLY */

    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], 0, foo, &i);
        P(&s);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

E. **Does the following program contain a race on the value of `myid`?** Yes No

```
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int id;
    P(&s);
    id = *((int *)vargp);
    V(&s);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    sem_init(&s, 0, 1); /* S=1 INITIALLY */

    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], 0, foo, &i);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```