

ECE 199 Final Exam Fall 2002

Thursday, December 19 2002

Name:

- **Be sure your exam booklet has 8 pages.**
- **Write your name at the top of each page.**
- **This is a closed book exam.**
- **You are allowed three 8.5 x 11 sheet of notes.**
- **Absolutely no interaction of any form between students is allowed.**
- **Show all of your work.**
- **Don't panic, and good luck!**

Problem 1	20 points	_____
Problem 2	20 points	_____
Problem 3	20 points	_____
Problem 4	20 points	_____
Problem 5	20 points	_____
Total	100 points	_____

Name: _____

Problem 1 (20 points): Short Answer

Part A (5 points): For this question, refer to the LC-2 code below:

```
        .ORIG    x3000
        LD      R0, A
        ADD     R2, R0, #0
        ADD     R0, R0, #2
        JSR    S
        HALT
S       LDR     R2, R2, #0
        JMPR   R2, #0
A       .FILL   x0021
        .END
```

This code does something very simple. Describe it in 10 words or less. Answers of the form : “First this program loads a value from memory location...” will get 0 credit.

Part B (5 points): For this question, refer to the code below:

```
void main()
{
    int x[10];
    int *y;

    y = &x[5];
    for (i = 0; i < 9; i = i + 1) {
        x[i] = i;
    }
}
```

What is the value of y[3]?

Name: _____

Problem 1 (continued)

Part C (5 points): For each item highlighted in the code below, indicate where that item is allocated in memory. Is it on the run-time stack or is it on the heap?

```
void main()
{
    int x;
    int *y;

    x = 3;

    y = (int *) malloc(sizeof(int));

    *y = 4;
}
```

Part D (5 points): Below is a code snippet that creates a new node, initializes it, and then returns the address of this new node to the calling function. Very briefly describe what is wrong with this code. Hint: there are no syntax errors.

```
struct node_struct {
    int value;
    struct node_struct *link;
}
typedef struct node_struct Node;

Node *CreateNode( int value, Node *next )
{
    Node newnode;

    newnode.value = value;
    newnode.link = next;
    return &newnode;
}
```

Name: _____

Problem 2 (20 Points): Recursion

The function **Balanced** below is designed to match parentheses. It is supposed to return 0 if the parenthesis in the character array **string** are balanced, or return something non-zero if the parentheses are not balanced. The initial call to **Balanced** would look something like:

Balanced(string, 0, 0);

Balanced, however, is missing a few key pieces of code. Fill in the three underlined missing portions in the code below:

```
int Balanced(char string[], int position, int count)
{
    if ( _____ )
        return count;

    else if (string[position] == _____)
        return Balanced( string, ++position, ++count);

    else if (string[position] == _____)
        return Balanced( string, ++position, --count);

    else
        return Balanced( string, ++position, count);
}
```

Name: _____

Problem 3 (20 Points): Problem Solving with Arrays

For this problem, you are given an array of size $n - 1$ that contains, in no particular order, all but one of the first n consecutive integers (1, 2, ..., $n - 1$, n). Your job is to write a function called **FindMissing** that returns the integer that is missing from the array.

Example: For the array below, $n = 6$ and `FindMissing()` should return 4.

5	1	3	6	2
---	---	---	---	---

For the array below, $n = 8$ and `FindMissing()` should return 8.

1	4	3	7	2	6	5
---	---	---	---	---	---	---

Hint: Before writing any code, spend some time thinking about this problem. Some solutions are more difficult than others. A simple 10-15 line program should suffice.

```
int FindMissing( int list[], int n )  
{
```

Name: _____

Problem 4 (20 Points): Functions involving linked lists

Part A (10 points): Joe has written a function named **SameNode** that determines if two nodes in a linked list are actually the same node—that is, they are physically the same object in memory. **SameNode** returns 1 if the two nodes are the same, and 0 if they are not. Below is the code Joe wrote, but unfortunately it is not *functionally* correct (i.e., it compiles, but doesn't work correctly in all cases).

```
struct node_struct {
    int value;
    struct node_struct *link;
}
typedef struct node_struct Node;

int SameNode( Node *a, Node *b )
{
    if( a->value == b->value )
        return 1;
    else
        return 0;
}
```

Provide a correctly functioning version of this **SameNode** below:

```
int SameNode( Node *a, Node *b )
{
```

Name: _____

Problem 4 (continued)

Part B (10 points): Given the following code and a linked list, complete the function **DeleteNode**, which deletes the node pointed to by **remove**. Assume that **remove** points to a node that is neither the head nor the tail of the linked list (i.e., it is somewhere in the middle.)

```
struct node_struct {
    int value;
    struct node_struct *link;
}
typedef struct node_struct Node;
```

```
void DeleteNode( Node *remove )
{
```

Name: _____

Problem 5 (20 points): Data structures and memory

Given the following code below:

```
struct node_struct {  
    int value;  
    struct node_struct *link;  
}  
typedef struct node_struct Node;
```

Node ***list**;

And the following contents of LC-2 memory:

x8000	x8002
x8001	x8006
x8002	x801A
x8003	x8008
x8004	x8006
x8005	x8000
x8006	x8009
x8007	x8002
x8008	x8011
x8009	x0000

If the the value of **list** is x8004, draw a picture of the entire data structure it points to below, clearly indicating the value of **value** for each node.