Computer Science 126 10/20/1999
First Midterm Exam with Answers 7pm - 8:30pm

A few meta-comments that are potentially helpful for future exams:

- Four (or five, depending on how you define them) of the following questions are derived directly from assigned exercises. Keeping up with exercises goes a long way towards doing well on about half of the exam.
- Two of the following questions are designed to reward people who attend lectures regularly and have understood the concepts explained in greater detail in lectures than from other sources.
- Most of the questions emphasize understanding "why" as oppposed to memorizing "what". So for example, memorizing a simple number of $\log_2 N$ is not as useful as knowing why it is so.

1. Debugging C [5]

The following program finds the value that occurs most often in an integer sequence of values between 0 and 99. Although this program will compile, there are some bugs in the code that will prevent it from calculating the correct answer. Find the errors and make the necessary corrections.

```
#include <stdio.h>
void
main()
{
   int num, i, maxi, a[99];

   while (scanf("%2d", num) != EOF)
      a[num]++;

   for (i = 0; i <= 99; i++);
      if (a[i] > a[maxi]) maxi = i;

   printf("%d\n", maxi);
}
```

Error 1:

To accommodate the numbers between 0 and 99, we need 100 elements. So the declaration of the array should be **`a[100]`**.

Error 2:

Need to initialize the array:

```
for (i=0; i < 100; i++) a[i] = 0;
```

Error 3:

Need an **`&`** in front of **`num`** inside **`scanf()`**.

Error 4:

maxi needs to be initialized:
`maxi = 0;`

Error 5:
Extraneous simicolon at the end of the `for()` line.

This problem is derived from problem #4 on page 14 of the course reader (exercises on arrays).

2. Unix Standard I/O Redirection [7]

The following program is compiled to obtain an “a.out” file.

```
#include <stdio.h>
void
main ()
{
    int n;
    scanf ("%d", &n);
    printf ("%d is output.\n", n+1);
}
```

Assume the following:

a) the file "input" has a single number "58" in it; and
b) if keyboard input is needed, you will type "58".

Answer what the following commands do by filling out the table. If the command results in an error, write "yes" in the "error?" column and leave everything else blank.

	command	error ?	keyboad input ?	terminal output ?	output file ?	output text
0	`a.out`		yes	yes		59 is output.
1	`input < a.out`	yes				
2	`input | a.out`	yes				
3	`a.out < input`			yes		59 is output.
4	`a.out > output`		yes		yes	59 is output.
5	`a.out > a.out`	yes[a]				
6	`a.out | a.out`		yes	yes		60 is output.
7	`a.out < input | a.out`			yes		60 is output.

a. The fifth line is a poorly designed question. My apologies. "**`a.out>a.out`**" usually results in an error but the error behavior is different depending on the platform and the environment of the user. The original intention was to test people's ability to tell the difference between line 5 and line 6. We will give credit for line 5 to everybody.

3. Structures [7]

Given the following definition:

```
typedef struct {float a; float b;} Interval;
```

Define a type for rectangles (**`Rect`**) that are parallel to the axes in a Cartesian coordinate system. Use the type **`Interval`** defined above.

```
typedef struct {Interval x; Interval y;} Rect;
```

Write a function that returns 1 if a point falls within a rectangle, 0 otherwise. Use the function prototype below and define your own **`Point`** type.

```
int inrect(Point, Rect);
```

```
typedef struct {float x; float y;} Point;

int inrect(Point p, Rect r) {
    return p.x > r.x.a && p.x < r.x.b &&
           p.y > r.y.a && p.y < r.y.b;
}
```

This is #5 and #6 of the structures exercises on page 22 in the course reader.

4. Pointers and Arrays [7]

What does the following program print?

```
#define N 6

main() {
    int i;
    int a[N];
    int *p, *q;

    p = &a[N-1];
    q = p - (N-1);

    for (i = 0; i < N; i++) {
        *(p-i) = i;
        *(q+i) = i;
    }

    for (i = 0; i < N; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}
```

5 4 3 3 4 5

p goes from end of array to the beginning. **q** goes from beginning to end. The earlier values set by each pointer are overwritten by the other pointer later.

5. Pointers and Linked Lists [6]

Suppose that a linked list is made up of nodes of type

```
typedef struct node *Link;
struct Node {int key; Link next; };
```

that you are given a pointer **list** of type Link, which points to the first node of the list; that the last node has **NULL** as its link; and that there are at least two nodes on the list. Write a code fragment to add a new node just after the second node of the list.

```
Link t = malloc(sizeof *t);
t->next = list->next->next;
list->next->next = t;
```

This is exercise #3 in course reader page 24.

6. Linked Lists and Recursion [6]

```
typedef struct node *Link;
struct Node {int key; Link next; };

int
f(Link list)
{
    int a,b;
    if (list == NULL)
        return 0;
    a = list->key;
    b = f(list->next);
    if (a > b)
        return a;
    else
        return b;
}
```

Suppose the above function is given a list whose content is **(9 8 1 12 8 2 5 7)**. (The head of the list contains the key 9. The **next** link of the last element is **NULL**.) What does the above function return?

12.

The function is supposed to find the maximum in a list. This is based on #5 of the recursion exercises in the course reader on page 29.

Suppose the above function is given a list whose content is
(-9 -8 -1 -12 -8 -2 -5 -7). What does the above function return?

0.

The value returned by the base case, 0, is greater than anything else on the list so it survives till the end. Question: how would you modify the function to make it work for any integers?

7. ADT: Stacks and Queues [5]

Suppose we have an empty stack of integers and an empty queue of integers to begin with, what will the following code fragment print?

```
for (i = 0; i < 100; i++) stackPush(i);
for (i = 0; i < 100; i++) queuePut(stackPop());
for (i = 0; i < 100; i++) stackPush(queueGet());
for (i = 0; i < 100; i++) printf("%d ", stackPop());
```

It prints 0,1,2,3,...,99.

Pushing and popping things on the stack once reverses the order. Pushing and popping things on the stack again restores the order. The queue has no effect on ordering.

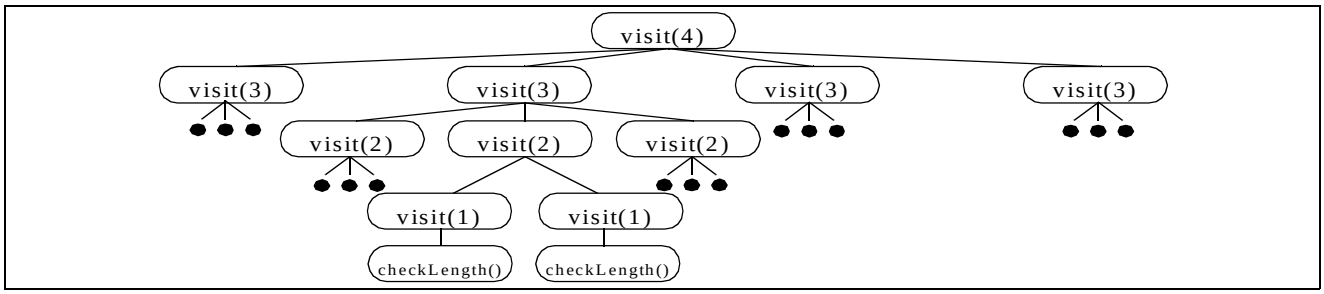
8. Recursion [6]

Given the recursive solution of the "Traveling Salesman Problem" discussed in class:

```
visit(int N) {
    int i;
    if (N == 1) {
        checkLength();
        return;
    }
    for (i = 1; i <= N; i++) {
        swap(i, N);
        visit(N-1);
        swap(i, N);
    }
}
```

When executing **`visit(4)`**, how many times does it invoke **`checkLength()`**?

24.



Refer to the annotated lecture slides for recursion (p23, and p24). The number of checks is **`4!=24`**. Without memorizing this fact, the key is to know how to trace the execution of a recursive function that makes multiple recursive calls per invocation. Page 24 of the lecture slides shows how to do this with a tree-like diagram. The knowledge of "Traveling Salesman" is not necessary, neither are the details of the function **`swap()`**.

When executing **`visit(4)`**, how many times does it invoke **`visit(2)`**?

12.

Trace the tree-like diagram, but stopping short of the bottom level.

9. Searching Arrays [4]

You have learned binary search. Let us extend it to "ternary search". In binary search, we repeatedly divide the search range into two roughly equal halves and search only one of them. In a ternary search, we repeatedly divide the search range into three roughly equal parts and search only one of them. In the worst case, how many comparisons do we make to search an array of N elements?

A) $Log_2(N)$

B) $Log_3(N)$

C) 3*Log$_2$(N)

D) 2*Log$_3$(N)

E) 2*Log$_2$(N)

F) 3*Log$_3$(N)

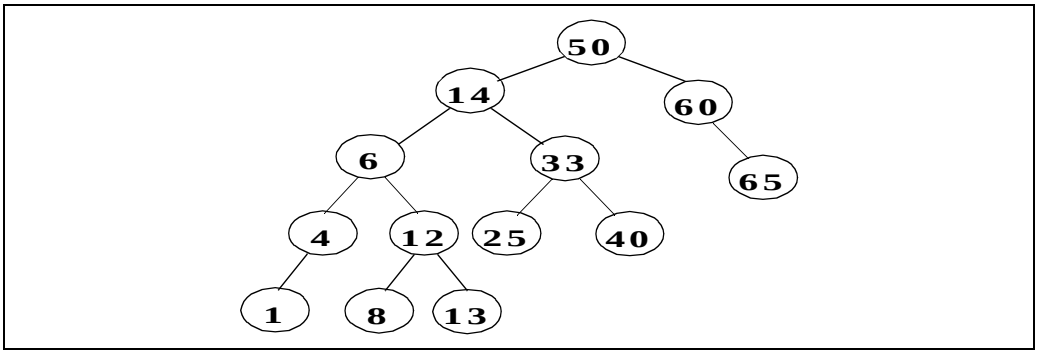
Briefly justify your answer (with one or two sentences or a simple picture).

D) 2*Log$_3$(N)

We can perform a maximum of **Log$_3$(N)** divisions and need to perform 2 comparisons after each division to determine which subpart of the array to search next.

10. Trees [7]

You are given the following binary search tree:



We perform a preorder traversal using a stack. During the traversal, at the moment when the top of the stack is 8, what is the content of the entire stack? As a reminder, here is the traversal code:

```
void
preorder(Link h) {
    stackPush(h);
    while (!stackEmpty()) {
        h = stackPop();
        visit(h);
        if (h->r != NULL)
            stackPush(h->r);
        if (h->l != NULL)
            stackPush(h->l);
    }
}
```

8
13
33
60

Note that this is **not** recursive. Solving the problem requires tracing the execution.