Final Exam

13 May, 1994

The exam is to be turned in at 12:30 PM. This is an open book exam.

Name: _____

Problem 1. (10 points)

How do you do the following in C?

(a), Declare `x` to be an integer.

(b), Declare `pI` to be a pointer to an integer.

(c), Set `pI` to the address of `x`.

(d), Declare `S` to be a structure with two integer fields, `r` and `s`.

(e), Declare `pS` to be a pointer to a structure of the type defined in part (d).

(f), Open a file called ``help.me''.

(g), Open a file called ``f:\dos\brock\help.me''.

Problem 2. (10 points)
Consider each of the following four loops

```
/* Loop A */
for (i=0; i<10000; i = i*2)
   for (j=0; j<i; ++j)
      s += A[i][j] ;
```

```
/* Loop B */
for (i=n; i>1; i = i/2)
   for (j=1; j<n; ++j)
      s += A[i][j] ;
```

```
/* Loop C */
for (i=0; i<n; ++i)
   for (j=i; j<i+10; ++j)
      s += A[i][j] ;
```

```
/* Loop D */
for (i=0; i<n; ++i)
   for (j=n; j>i; --j)
      s += A[i][j]
```

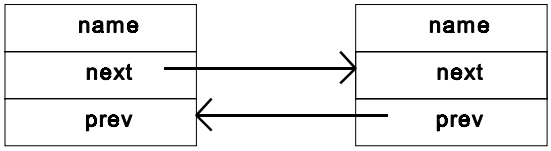
In big-O notation, the running times of these algorithms are O(1), O(n), O(n log n), and O(n^2). Match each algorithm with its running time.

Problem 3. (10 points)
Write a complete program which displays the number of lines in a text file. The program should prompt the user for the file name, then write the number of lines. A sample run of the program, where user entered data is underlined, is shown below:

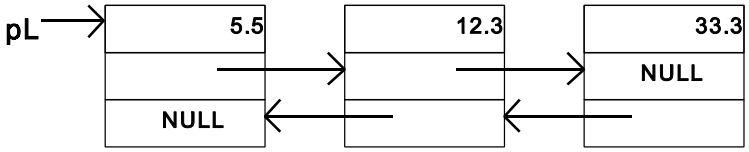
```
Enter filename: fred.c
File fred.c contains 16 lines
```

Problem 4. (20 points)
Use the following C definitions and declarations in your answers to this question.

```
typedef struct dl_node_tag {
   float name ;
   struct dl_node_tag *next ;
   struct dl_node_tag *prev ;
} DL_Node_type ;
struct dl_node_tag *pL, *pT ;
```



Starting with the variable `pL` pointing to the leftmost node of the doubly linked list on the right, show the result of executing the following ten C statements in order.




```
pL->prev = pL ;                        /*  1 */
pL->next = pL->next ;                  /*  2 */
pT = (struct dl_node_tag *)            /*  3 */
          malloc(sizeof(struct dl_node_tag));
pT->prev = pL ;                        /*  4 */
pT->next = pL->next ;                  /*  5 */
pL->next->next = pT ;                  /*  6 */
pT->next->name = 17.0 ;                /*  7 */
pT = pT->next->next ;                  /*  8 */
pT->next = pL ;                        /*  9 */
pL = pT ;                              /* 10 */
```

You may find it useful to use the numbers in the comments to document the changes.

In the last two assignments in this course, we have used collections of related procedures to solve a programming problem. For example, in the last assignment, we used procedures:

```
int CreateHashTable(hash_table_type *, int) ;
int UpdateHashTable(hash_table_type, char *) ;
int DumpHashTable(hash_table_type, FILE *) ;
int PrintHashTable(hash_table_type, FILE *) ;
```

to build a ``data base" of words read from a file. In the ``big number" (or ``number list") assignment, you used three related procedures that:

(1), read a big number from standard input;

(2), added two big numbers to create a new big number; and

(3), printed a big number to standard output.

Suppose the file `table.h` lists the following set of function prototypes:

```
/* table.h -- header file for Table processing facility */
typedef ____________ Table ;
void CreateTable (Table *ptable);
/* Creates and initializes a Table of integers. Must be called
 * before any attempt is made to insert or delete data.
 */
void DestroyTable (Table *ptable);
/* Destroys a Table. Call as soon as the Table is no
 * longer needed.
 */
int EmptyTable (Table *ptable);
/* Returns 1, if the Table is non-empty.
 * Returns 0, if the Table is empty.
 */
void Insert (Table *ptable, int data);
/* Inserts a specified data value into Table *ptable.
 */
int DeleteMin (Table *ptable);
/* Returns and deletes the smallest data value found in
 * the specified Table. Should only be called when
 * the Table actually contains data.
 */
int DeleteMax (Table *ptable);
/* Returns and deletes the largest data value found in
 * the specified Table. Should only be called when
 * the Table actually contains data.
 */
void ShowTable (FILE *file, Table *ptable);
/* Prints all data values currently stored in the Table
 * in ascending order to the file passed as the first argument.
 */
int SumTable (Table *ptable);
/* Returns the sum of all data values currently stored in the
 * specified Table. Has no effect on table contents.
 */
```

Note that we have left the type definition for the `Table` data type blank. That is intentional. In Problem 5, you will write code that calls these procedures without worrying about how the data in the `Table` is actually stored. In Problems 6 and 7, you'll get to worry.

You'll need to write your solutions to these last two problems on your own paper.

Problem 5. (20 points)
Write a program that does the following:

(a) declares a variable `T` of type `Table` and initializes the `T` by calling `CreateTable`;
(b) prompts a user to enter a sequence of 10 integers from the keyboard;
(c) reads and stores all these integers into `T`;
(d) prints a sorted list of all the integers in `T`;
(e) prints the sum of all the integers in `T`;
(f) removes the largest and smallest integers from `T` and prints both values; and
(g) destroys `T` and exits.

Problem 6. (10 points)
After reading the full set of function prototypes in `table.h`, let's turn to the problem of figuring out a good data structure for `Table`. Consider the possibilities of implementing `Table` either as a singly-linked list or as a doubly-linked list. In either case, the integers should be stored in ascending order within the list.

For each of the eight operations of `Table` decide if the operation is easier to implement with singly-linked or doubly-linked lists. Give a succinct argument for your viewpoint and, please, don't write any code.

Problem 7. (20 points)
Based on your conclusions in Problem (6), define a partial implementation for the `Table` procedures by:

(a) completing the missing `typedef` for `Table` in `table.h`;
(b) writing a matching implementation of the `Insert` function; and
(c) writing matching versions of `DeleteMin` and `DeleteMax`.