

Name / ID (please PRINT) Sequence #: _____ Seat Number: _____

KEY

CS 2123.002 Data Structures

Spring 2018 – Midterm 2 -- March 22, 2018
You have 75 min. Good luck.

- This is a **closed book/note** examination. But *You can use the reference card(s) given to you.*
- This exam has 4 questions in 9 pages. Please read each question carefully and answer all the questions, which have 100 points in total. Feel free to ask questions if you have any doubts.
- **Partial credit will be given, so do not leave questions blank.**

You can get **1pt bonus** credit if you complete the **boldfaced column** of the following table. Please do this after answering all the questions in the exam. You will also get **1pt bonus** if the total expected score is within ± 5 of total received score.

| Question | Topic | Possible Points | Student Expects to receive out of | Student's Received Score |
|--------------|---|-----------------|-----------------------------------|--------------------------|
| 1 | Review: pointer, string, big-O analysis, recursive function | 25 | /25 | |
| 2 | linked list - difference of two sets | 25 | /25 | |
| 3 | bufferADT and double linked list | 25 | /25 | |
| 4 | queueADT and driver | 25 | /25 | |
| | Bonus If this table is completed | 1 | | |
| | If the total expected score is within ± 5 of total received score | 1 | | |
| | | | | |
| Total | | 100+2 | | |

1. (25 pt) Review Questions

a. (5pt) Show how the memory content changes as the below code is executed. Keep the old values in the table along with new values so I can see the changes.

| <pre>int *p, **q; int arr[4]={5,8,3,7}; p = arr; q = &p; **q = 11; *++p = 9;</pre> | Variable name | Memory Address | Memory content | |
|---|------------------|-------------------|-------------------|-----|
| | P | 100 | 108 | 112 |
| | Q | 104 | 100 | |
| | arr[0] | 108 | 5 | 11 |
| | arr[1] | 112 | 8 | 9 |
| | arr[2] | 116 | 3 | |
| | arr[3] | 120 | 7 | |
| | | | | |

b. (5pt) Suppose one of your friends is struggling with the following code segment. It compiles OK, but gives a segmentation fault when executed. What is the problem there and how would you fix it if we want to keep strcpy statement as is?

```
char *a = "CS 2123";
/* ... */
strcpy(a, "CS 4321");
```

The problem is (2pt):

a is declared as a char pointer and it is pointing to a constant whose content cannot be changed... we can define a as an array or dynamically allocate memory

Fixed version is (3pt):

```
char a[8] = "CS 2123";    // OR
                           char *a = malloc(8);
                           if(a==NULL) exit();
                           strcpy(a, "CS 2123");
                           /* ... */
strcpy(a, "CS 4321");
```

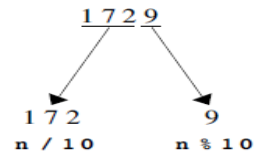
- c. (6pt) In the following loops, first count the number of operations (i.e., find exactly how many lines will be printed out) when N is 32. Then give the computational complexity using big-O notation for any value of N. (justify your answers)

| int i, j, N=32; | Number of lines printed when N=32 | big-O notation |
|--|-----------------------------------|---|
| <pre>for(i=1; i <= N; i++) for(j=N*N; j >= 2; j = j/2) printf(" line1\n");</pre> | 32*10 = 320 | $N \log_2 N^2$ $N^2 \log_2 N$ $O(N \log_2 N)$ |
| <pre>for(i=1; i <= N; i++) for(j=N; j >= 2; j = j/2) printf(" line2\n");</pre> | 32*5 = 160 | $O(N \log_2 N)$ |

- d. (9pt) Write a **recursive** function DigitMin(n) that takes a nonnegative positive integer and returns the min digit in it. For example, calling DigitMin(1729) should return 1 because min{1, 7, 2, 9} is 1.

Hint: As in DigitSum that we did in class, think about breaking an integer down into two components using division by 10.

For example, given the integer 1729, you can divide it into two pieces as follows:



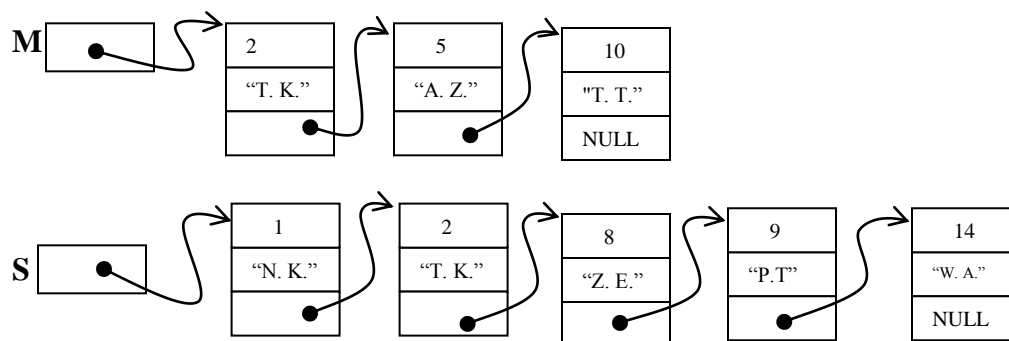
```

int DigitMin(int n)          /* MUST be recursive */
{
    int part1, part2;        // 1pt
    if (n < 10)               // 2pt base case
        return n;
    else{                     // 6pt
        part1 = DigitMin(n/10);
        part2 = n % 10;
        // return (part1 < part2) ? part1 : part2;
        if (part1 < part2)
            return part1;
        else
            return part2;
    }
}
  
```

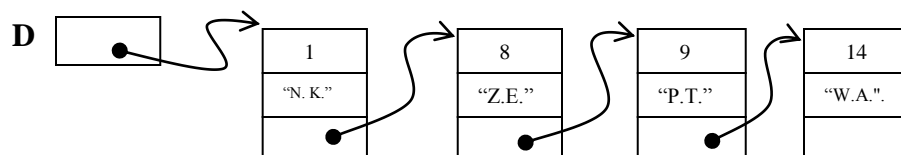
2. (25pt) We use the following cell structure to store the IDs and names of some students in a **single linked list** data structure

```
typedef struct cell {
    int ID;
    char name[20];
    struct cell *next;
} cellT;
```

Suppose somehow we have already created the following two single linked lists, namely M and S, to store the IDs and names of the students who are taking **Math** and **Science**, respectively. Both lists are **sorted** with respect to (w.r.t.) IDs.



Now we are interested in creating a new list, say D, to store the IDs and names of the students who are taking Science **but NOT** Math. As you may know this operation is known as **difference** of two sets ($S - M$) such that D contains the elements that appear in S but NOT in M. So list D will look like as follow (elements in list D should be **sorted** w.r.t. IDs, as in the original lists):



You are asked to implement `cellT *difference_S_M(cellT *S, cellT *M)`, which can be called as follows to find the difference between S and M.

```
cellT *M, *S, *D;
```

```
/* somehow the lists pointed by M and S are created */
```

```
D = difference_S_M(S, M);
```

After your function, M and S should be intact. So do not remove the cells from M or S! When needed, create new cells and copy the IDs and names from the cells in S.

```

cellT *difference_S_M(cellT *S, cellT *M)
{
    cellT *D=NULL, *tmp, *tail;

    while( S ){
        if (M && M->ID == S->ID) {
            // they are the same skip both ...
            M = M->next;
            S = S->next;
        } else if (M && S->ID > M->ID) {
            // S->ID might be in M, so just skip M
            M = M->next;
        } else { // M is NULL OR S->ID < M->ID
            // S->ID is not in M, so add/insert its info to D
            tmp = (cellT *) malloc(sizeof(cellT));
            if (tmp==NULL) exit(0);

            tmp->next = NULL;
            tmp->ID = S->ID;
            strcpy(tmp->name, S->name);
            // tmp->name = S->name; // is wrong! Why?

            if (D==NULL)
                D = tmp;
            else
                tail->next = tmp;
            tail = tmp;

            S = S->next;
        }
    }
    return D;
}

```

3. (25pt) Recall the bufferADT which had the same interface `buffer.h` with four different implementations (i.e., array, stack, single linked list and **circular double linked list**).

Suppose `buffer.h` has a function `void BackspaceBuffer(bufferADT buffer);`, which **deletes** the character before the cursor.

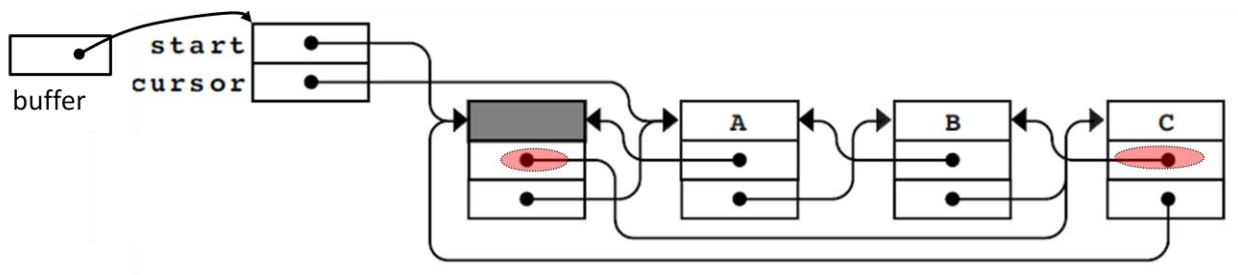
If the cursor is at the beginning of the buffer, then there is will be no change!

For Example:

Suppose `buffer` has

A | B C

Here is how the buffer conceptually looks like:



After we call `BackspaceBuffer(buffer);`

The `buffer` should have :

| B C

- a. [5pt] As in the above figure, first draw how the buffer conceptually looks like now.

b. [20pt] Then implement this function under **circular double linked list** representation.

```
/* doublelinkedlistbuf.c */
#include "buffer.h"

typedef struct DcellT {
    char ch;
    struct DcellT *prev;
    struct DcellT *link;
} DcellT;

struct bufferCDT {
    DcellT *start;
    DcellT *cursor;
};

bufferADT NewBuffer(void)
{
    bufferADT buffer;
    DcellT *dummy;

    buffer = New(bufferADT);
    dummy = New(DcellT *);

    buffer->start = dummy;
    buffer->cursor = dummy;
    dummy->link = dummy;
    dummy->prev = dummy;

    return (buffer);
}

/* implementations of other functions */
```

```
void BackspaceBuffer(bufferADT buffer)
{
    DcellT *cp;

    if (buffer->start == buffer->cursor)
        return; // 3pt

    cp = buffer->cursor; // 2pt

    buffer->cursor = buffer->cursor->prev;
    // OR buffer->cursor = cp->prev; // 4pt

    cp->prev->link = cp->link; // 4pt
    cp->link->prev = cp->prev; // 4pt

    free(cp); // 3pt
}
```

4. (25pt) Recall queueADT structure which had the following `queue.h` interface:

```
/* queue.h */
#ifndef _queue_h
#define _queue_h
#include "genlib.h"

typedef void *queueElementT;

typedef struct queueCDT *queueADT;

queueADT NewQueue(void);
void FreeQueue(queueADT queue);

void Enqueue(queueADT queue, queueElementT element);
queueElementT Dequeue(queueADT queue);

bool QueueIsEmpty(queueADT queue);
bool QueueIsFull(queueADT queue);
int QueueLength(queueADT queue);

queueElementT GetQueueElement(queueADT queue, int index);

#endif
```

Suppose its implementation is available as `queue.o`, so you can use all the functions in `queue.h`, but you cannot change their implementation.

Now you are asked to complete the driver/application program in the next page by using the above `queue.h` interface and implementing the necessary piece of codes needed for your driver/application as defined below.

The driver/application simply reads all the **double** numbers from a file whose name is given as a command line argument (*this part is already done for you*) and inserts (Enqueue) them into the queue (*you will do this part*). Please note that `queueElementT` is defined to be **void ***. So you cannot directly enqueue the **double** numbers into the queue. In this case, remember you should allocate memory for your double numbers and then enqueue their addresses in the queue!

Then you are asked to find and print the min of the double values in the queue, but do not dequeue or remove the numbers from the queue. So the queue still contains all the values.

Finally, before exiting from the program, make sure all the dynamically allocated memory spaces and structures are released/freed.


```

/* driver.c */      /* assume all the necessary standard C libraries and booklibs are included here too */
#include "queue.h"
int main(int argc, char *argv[])
{
    FILE *fp;
    queueADT myQ;
    double value, *ptr, min;
    int i;

    if(argc!=2 || (fp=fopen(argv[1],"r")) == NULL){
        printf("not enough argument or file cannot be opened\n"); exit(0);
    }
    myQ = NewQueue();

    // [9pt] get each double value from the file, insert it into myQ
    while(fscanf(fp, "%lf", &value) == 1){

        ptr = (double *) malloc(sizeof(double)); // 3pt
        if(ptr==NULL) exit(-1); // 1pt
        *ptr = value; // 2.5pt
        Enqueue( myQ, ptr); // 2.5pt

    }
    // [9pt] find/print the min of the values in myQ.
    // but do not dequeue or remove the numbers from myQ.
    // min = INF;
    for(i=0; i < QueueLength(myQ); i++){ // 3pt
        ptr = (double *) GetQueueElement(myQ, i); // 2.5pt
        if (i==0) min=*ptr; // 0.5pt
        else if (min < *ptr) min = *ptr; // 2.5pt
    }
    printf("The min is %lf\n", min); // 0.5pt

    // [7pt] release/free all the dynamically allocated memory spaces
    for(i=0; i < QueueLength(myQ); i++){ // 1pt
        ptr = (double *) GetQueueElement(myQ, i); // 2pt
        free(ptr); // 2pt
    }
    // while(!QueueIsEmpty(myQ)) free(Dequeue(myQ)); // is OK too, 5pt
    FreeQueue (myQ); // 2pt
}

```