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

_____ _____

CS 2123 Data Structures

Fall 2018 – Midterm 2

November 1, 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 5 questions in 10 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**.

Question	Topic	Possible Points	Student's Received Score
1	Review: struct-pointer, big-O analysis, recursive function, BST traversal	20	
2	Linked List - remove common elements from two linked lists	25	
3	bufferADT using double linked list	20	
4	queueADT and driver	25	
5	Binary Search Trees (recursive function)	10	
Total		100	

1. (20pt) Review Questions

   a. (**5pt**) Suppose one of your friends is struggling with the following code. While the declarations on the left column are OK and compile correctly, the statements in the right column give compiler errors. Fix (re-write) each statement so there will be no compiler error.

`typedef struct emp_data {`
`  char *name;`
`  char ssn[12];`
`  int age;`
`} emp_dataT;`
 	
`typedef struct magic {`
`  emp_dataT *staff[10];`
`  emp_dataT faculty[5];`
`  int count;`
`} magicT;`
 	
`magicT p, *q;`
	`q = p;  //1pt`
	
	`.....`
	
	`p->staff[2].name = "Abcd";  //2pt`
	
	`.....`
	
	`q->faculty[3]->age = 45;  //2pt`
	
	`.....`

   b. (**8pt**) In the following loops, first count/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

`for(i=1; i <= N; i++ )`
`  for(j=2; j <= N; j = j*2 )`
`  printf(" line1\n");`		

`for(i=2; i <= N*N; i = i*2 )`
`  for(j=1; j <= N; j++ )`
`  printf(" line2\n");`		

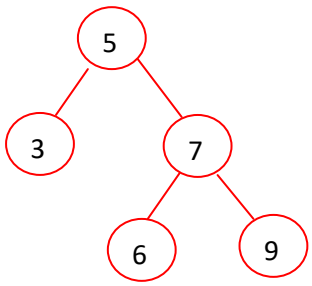
c. **(5pt)** We can easily write an iterative function that takes a positive <u>even</u> integer `(int n)` as a parameter and calculates/returns the sum of the even numbers from 2 to n (i.e., it finds 2+4+6+...+n), as follows:

```
int SumEven_Iter(int n)    /* iterative version */
{
  int sum = 0;
  for(i=2; i <= n; i=i+2){
        sum += i;
  }
  return sum;
}
```

Now you are asked to solve the same problem using a **<u>recursive</u>** function `SumEven_Rec(int n)` which takes a positive <u>even</u> integer (must be greater than or equal to 2), and finds/returns the sum of the even numbers from 2 to n (i.e., again find/return 2+4+6+...+n).

```
int SumEven_Rec(int n)   /* MUST be recursive */
{
```

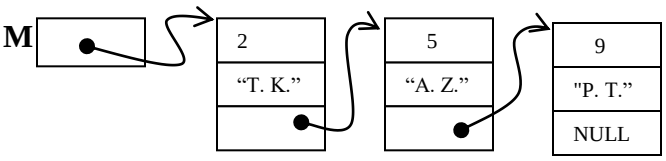
d. (**2pt**) If you print the values from the following binary search tree in PRE-ORDER, what will be the output?

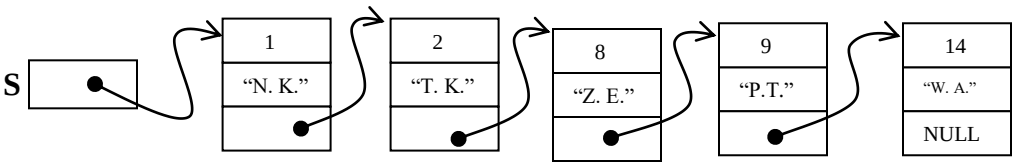


2. (25pt) We use the following cell structure to store the IDs and names of some students in a **single linked list** data structure

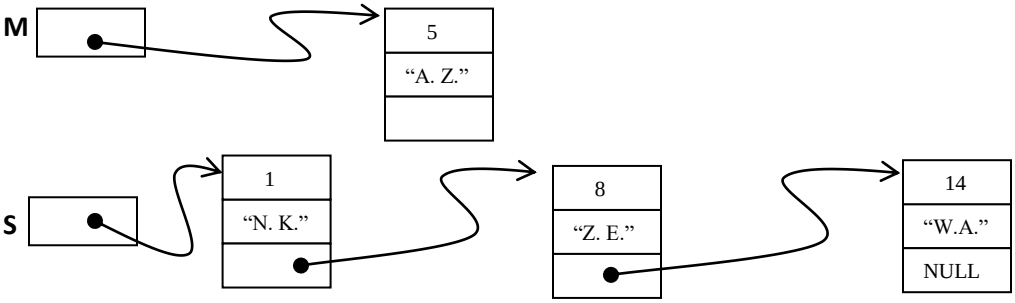
```
typedef struct cell  {
   int ID;
   char name[20];
   struct cell  *next;
}  cellT;
```

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 **M**ath and **S**cience, respectively. Both lists are **sorted** with respect to (w.r.t.) IDs.





Now you are asked to write a function that **removes/eliminates the common students from both lists** such that each list will only contain the students who are not in the other list. After calling your function, the resulting lists M and S should be as follows (still sorted w.r.t. IDs).



Implement `void remove_common_students(cellT **ptrM, cellT **ptrS)`, which can be called as follows to remove/eliminate the common students from both lists .

```
cellT *M, *S;

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

remove_common_students( &M, &S);
```

After your function, M and S should be changed as you need to remove the common cells from M and S!

use the next page to answer question 2.

```
void remove_common_students (cellT **ptrM, cellT **ptrS)
{
   cellT *M, *S, *prevM=NULL, *prevS=NULL, *tmp;
```

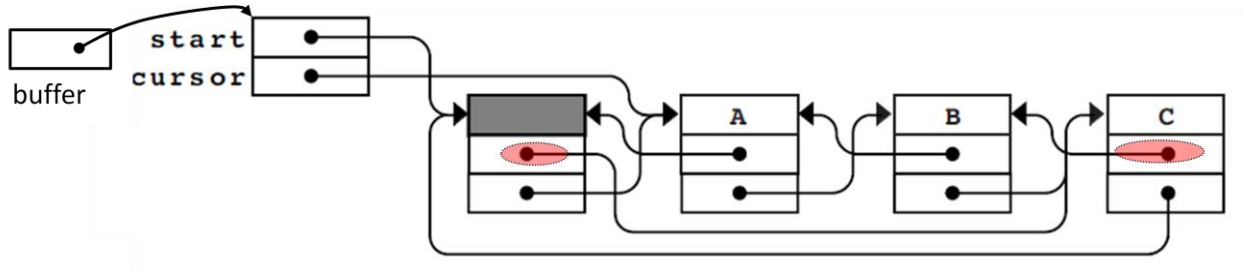
3. (20pt) Recall the bufferADT which had the same interface `buffer.h` with four different implementations (i.e., array, stack, single linked list and **circular double linked list**).

Suppose we include a new function into `buffer.h` that moves the cursor to a position just before the beginning of the next word after the cursor. Suppose words are separated by at least one space char, which is `' '`. If there is no new word after the cursor, leave the cursor at the original place.

For Example:

Suppose `buffer` has **A | B C**

Here is how the buffer conceptually looks like:



After we call `GoToNextWordBuffer(buffer);`

The `buffer` should still have : A **| B C**

because there is no new word after the cursor

Example 2:

Suppose `buffer` has **A | B C K L M X Y Z**

After we call `GoToNextWordBuffer(buffer);`

The `buffer` should have : **A B C | K L M X Y Z**

If we call `GoToNextWordBuffer(buffer);` again

The `buffer` should have : **A B C K L M | X Y Z**

Now implement this function under **circular double linked list** representation, which is shown above. You need to see at least one space char `' '` before the next word. If you have more space characters, skip all them and make sure the cursor is placed just before the first char of the next word, as in the above examples.

use the next page to answer question 3.

/* 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 GoToNextWordBuffer(bufferADT buffer)
{
  DcellT *cp;
```

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.

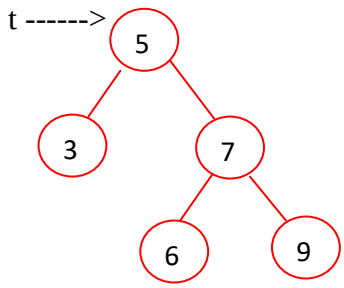
The driver/application simply reads the name and salary of each employee from a text file into a dynamically allocated **empT** structure and inserts (Enqueue) the address of this structure into the queue. File name is given as a command line argument. *(Anyway, this part is already done for you, just inspect the code in the next page.)*

Now you are asked to find and print the *name* and *salary* of the employee who has the largest (**maximum)** salary. Do not dequeue or remove the employees from the queue yet.

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"
typedef struct emp {
  char name[30];
  double salary;
} empT;

int main(int argc, char *argv[])
{
   FILE *fp;           queueADT myQ;
   empT *ptr, *max;    double max_salary;    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();
   while(!feof(fp)){
        ptr = (empT *) malloc(sizeof(empT));    if(ptr==NULL) exit(-1);
        fscanf(fp, "%s %lf", ptr->name, &ptr->salary);
        Enqueue( myQ, ptr);
   }
   // [15pt] find/print the name and salary of the employee who has the
   // larges (maximum) salary in myQ. Do not dequeue employees from myQ, yet.









    // [10pt] release/free all the dynamically allocated memory spaces









}
```

5. (10pt) *[Binary Search Tree (BST)]* You are asked to implement the following function: `void FreeBST(nodeT *t);` which frees all the nodes in a given BST.




```
typedef struct node  {
   int key;
   struct node *left, *right;
}  nodeT, *treeT;
```

After calling your function, all the nodes in a given BST should be freed. Assume that each node was allocated using a single malloc.

```
void FreeBST(nodeT *t)
{
```