BBM 201 – Data Structures - Fall 2019

2nd Midterm

December 24, 2019

Name: ____

Student ID Number: __________________ Section:__________

Problem	Points	Grade
1	20	
2	15	
3	20	
4	15	
5	19	
6	16	
Total	105	

INSTRUCTIONS

- Do not open this exam booklet until you are directed to do so. Read all the instructions first.
- When the exam begins, write your name on every page of this exam booklet.
- The exam contains six multi-part problems. You have **120 minutes** to earn 105 points (5pts bonus).
- The exam booklet contains **7 pages** including this one.
- This exam is an **open book and notes exam**. You are allowed to have two pieces of **bound** books or notebooks.
- Please write your answers in the space provided on the exam paper.
- Be neat.
- Good luck!

QUESTIONS

Question 1) Evaluation of Expressions (20 pts):

(10) a. Convert the following fully parenthesized infix expression to postfix and prefix notations:

(x-((x+y)^((z/w)-q)))

postfix:	**xxy+zw/q-^-**
prefix:	**-x^+xy-/zwq**

(10) b. Fill in the table below accordingly to convert the infix expression

(E-F)^A/C

to a postfix expression using an operator stack. Note that you are supposed to leave some cells blank (In other words, you should not fill all blank cells).

Token	Operator Stack			Output
	[0]	[1]	[2]	
(	(			
E	(			E
-	(	-		E
F	(	-		EF
)				EF-
^	^			EF-
A	^			EF-A
/	/			EF-A^
C	/			EF-A^C
eos				EF-A^C/

Question 2) Array-based Linked Lists (15 pts):

According to the given array based linked list which is defined as follows:

```
#define MAX_LIST 10

typedef struct{
       char name[10];
       int link;
     }planet;
planet linkedlist[MAX_LIST];
int free_; //the index of the first free space in the list
int* list; //the index of the first item in the list
```

Imagine we begin with an alphabetically sorted list in the first table given below. In the sorted list, each element holds a link to the next element. That link points to the index of the next element. The first element in the list can be accessed through *list. The first available element in the list, where a new element can be inserted, is accessed through free_. The available items are also linked to each other in the same way.

After we insert “Jupiter” and delete “Neptune” successively (you will delete “Neptune” after you insert “Jupiter”) in the same sorted list, what will be the contents of the list? Please fill in the tables and write down the new values of free_ and *list.

Initial Table

	name	link
[0]	“Venus”	-1
[1]	“Saturn”	0
[2]	“Neptune”	4
[3]	“Mars”	2
[4]	“Pluto”	1
[5]	“Earth"	3
[6]		7
[7]		8
[8]		9
[9]		-1

Insert “Jupiter”

	name	link
[0]	“Venus”	-1
[1]	“Saturn”	0
[2]	“Neptune”	4
[3]	“Mars”	2
[4]	“Pluto”	1
[5]	“Earth"	6
[6]	“Jupiter”	3
[7]		8
[8]		9
[9]		-1

Delete “Neptune”

	name	link
[0]	“Venus”	-1
[1]	“Saturn”	0
[2]	“Neptune”	7
[3]	“Mars”	4
[4]	“Pluto”	1
[5]	“Earth"	6
[6]	“Jupiter”	3
[7]		8
[8]		9
[9]		-1

free_ = 6 free_ = 7 free_ = 2

*list = 5 *list = 5 *list = 5

Question 3) Circular Linked Lists (20 pts):

(10) a. Complete the given method that returns 1 if the given linked list is circular, otherwise returns 0. Please note that an empty linked list is supposed to be circular. You are allowed to define only an extra variable without doing any memory allocation.

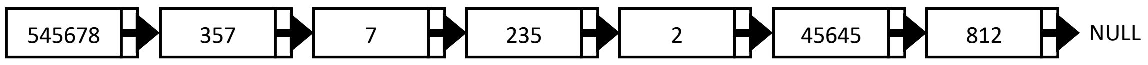
```
struct node
{
    int data;
    struct node* next;
};

int isCircular(struct node* head){

if (head==null)
      return 1;

struct node* next = head->next;
while(node!=null && node !=head)
      node = node->next;
return (node == head);

}
```

(10) b. This time write a <u>recursive</u> method that returns 1 if the given linked list is circular, otherwise returns 0. Again an empty linked list is supposed to be circular. The parameter ‘cur’ points to the head node in the first call. You are not allowed to define any other variable.

```
struct node
{
    int data;
    struct node* next;
};

int isCircular(struct node* head, struct node* cur){

if (head==null)
      return 1;

if(cur==null)
      return 0;

if(head==cur)
      return 1;

return isCircular(head, cur->next);

}
```

Question 4) Array of Linked Lists (15 pts):

For the program code given below, input of func is:



Hint: (int)log10(94634) = 4

(5) a. Write down the exact output line printed after == **PART 1 :** ==

2 >> 0 >> 3 >> 0 >> 1 >> 1 >> 0 >> 0 >> 0 >> 0 >>

(10) b. Write down the exact output line printed after == **PART 2 :** ==

>> 7 >> 2 >> 357 >> 235 >> 812 >> 45645 >> 545678

```
typedef struct ListNode* ListNode_pointer;
typedef struct ListNode {
    int element;
    ListNode_pointer next;
};

void func(ListNode_pointer *A) {
    if (*A == NULL) return;
    ListNode_pointer temp = *A;
    ListNode_pointer newHeads[10];
    ListNode_pointer newTails[10], tail;
    int co[10], om;
    for (int i=0; i<10; i++) {
        newHeads[i] = NULL;
        newTails[i] = NULL;
        co[i] = 0;
    }

    while (temp != NULL) {
        om = (int)log10(temp->element);

        co[om]++;
        if (newHeads[om] == NULL) {
            newHeads[om] = temp;
            newTails[om] = temp;
        }
        else {
            newTails[om]->next = temp;
            newTails[om] = temp;
        }
        temp = temp->next;
    }

    int i = 0;
    for( ; co[i] == 0; i++);
    *A = newHeads[i];
    tail = newTails[i];

    for (int j=i+1; j<10; j++) {
        if (newHeads[j] != NULL) {
            tail->next = newHeads[j];
            tail = newTails[j];
        }
    }
    tail->next = NULL;

    printf("\n== PART 1 : ==\n");
    for (int i=0; i<10; i++)
        printf("%d  >> ", co[i]);
    printf("\n************** \n");

    temp = *A;
    printf("\n== PART 2 : ==\n");
    for( ;temp != NULL; temp = temp->next)
        printf(" >> %d",temp->element);
    printf("\n************** \n");
}
```

Question 5) Linked List Implementation of Stack (19 pts):

Fill the following empty lines to implement a **recursive** function to find the minimum value stored in a linked list-based stack. You can assume that the stack is not empty. Each empty line is to be filled with one of the lines on the right. Only 6 lines from the right will be used. Write the corresponding line number inside the parenthesis.

```
struct Node {
   int data;
   struct Node* link;
};

int FindMin(struct Node* s) {
   ( 3 )
   ( 5 )
   ( 1 )
   (10 )
   {
      ( 6 )
      ( 2 )
   }
}
```

Each line is worth 1 point.
+3 points for 6 correct lines.
Alternatively, first 3 lines can be (5),(1),(3).

1	`return s->data;`
2	`return s->data<min?s->data:min;`
3	`int min;`
4	`return min;`
5	`if (s->link == NULL)`
6	`min = FindMin(s->link);`
7	`if (s == NULL)`
8	`s = s->link;`
9	`min = FindMin(s);`
10	`else`

Now implement the same **recursive** function assuming you have no access to the underlying linked list structure. You can only use the standard stack functions to access and modify a *global* stack. You can assume that the stack is not empty. Each empty line is to be filled with one of the lines on the right. Only 7 lines from the right will be used. Write the corresponding line number inside the parenthesis.

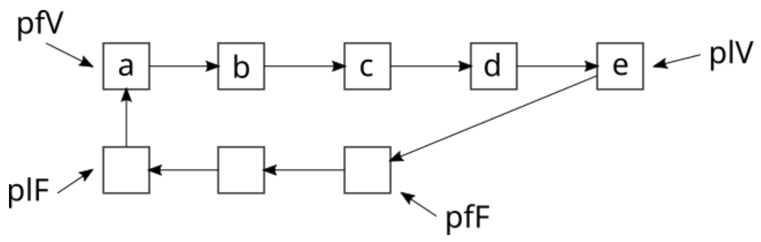
```
int FindMin()
{
   ( 9 )
   ( 1 )
   ( 7 )
   (10 )
   ( 4 )
   ( 2 )
   ( 8 )
}
```

Each line is worth 1 point.
+3 points for 7 correct lines.
Alternatively, (1) and (7) can be swapped.

1	`pop();`
2	`push(a);`
3	`return minVal;`
4	`minVal = FindMin();`
5	`if(isEmpty())`
6	`push(minVal);`
7	`int minVal = a;`
8	`return a<minVal?a:minVal;`
9	`int a = top();`
10	`if(!isEmpty())`

Question 6) Circular Linked List Implementation of Stacks and Queues (16 pts):

Consider a dynamically allocated linked list based circular data structure. As long as there are free nodes available, new node requests are answered by these nodes. The last node with a value is followed by the first free node and the last free node is followed by the first node with a value as shown in the figure below:



This structure can be implemented in many different ways based on the following possible pointers and/or values:

- store a pointer, pfV, to the first node with a value (top, front)
- store a pointer, pfF, to the first free node
- store a pointer, plV, to the last node with a value
- store a pointer, plF, to the last free node
- number of values stored in the list, nV
- number of free nodes, nF

Assuming that there are **m** nodes with values and **k** free nodes in the list (m>1, k>1), fill in the following table with the **number of nodes accessed or modified** (value or link fields are read or changed) during each ADT operation for the corresponding implementation variant, assuming a smart implementation. You can assume that the free nodes can be rearranged/reordered as necessary, but the nodes with values maintain their order at all times.

Stored pointers	enqueue	dequeue/pop	push	top/front
pfV & pfF	1	1	k	1
plV & plF	2	1	3	2
pfV & nV	m+1	1	m+k	1
plV & nF	2	0	k+1	k+2