COMP 1020: Sample Exam 1

UMASS - LOWELL

Name: ANS KEY

Student ID: ______________________

Question	Points
1	40
2	30
3	10
4	15
5	23
6	10
7	10
Total	138

Instructions:

1. This examination contains 13 pages, including this page.
2. Write your answers in this booklet. If you must write on the back page, please indicate **very** clearly on the front of the page that you have written on the back of the page.
3. You **may** use any resources, including lecture notes, books, other students or other engineers, but you should provide a reference.
4. You may use a calculator. You may not share a calculator with anyone.

Question 1: Linked Lists

[40 pts]

(a) (10 points) Please complete the following function to concatenate two circularly singly linked list into one circularly singly linked list.

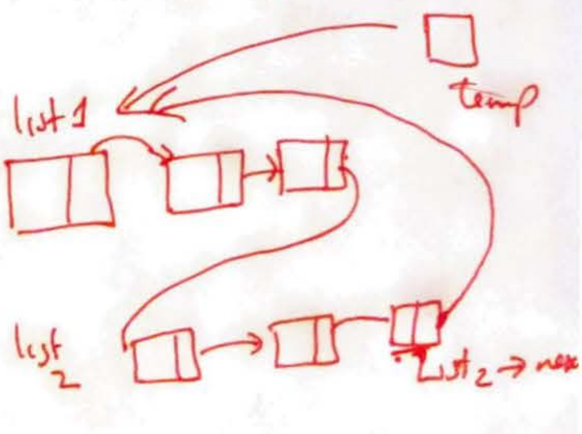
```
typedef struct node Node;

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

```
Node* concatenate(Node* list1, Node* list2) {
/*
Produce a new list which contains list1 followed by list2. Return the pointer which
    points to the new list. The arguments list1 and list2 point to the tails of the
    lists.
*/
```

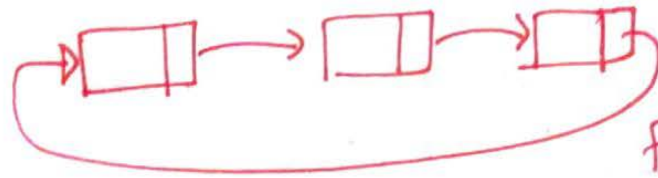
```
Node* temp;
if (List1 == NULL)
        return list2.
if (List2 == NULL)
        return List1;
temp = list1 -> next; // hold the head of list1
list1 -> next = list2 -> next; // Point to head of list 2
list2 -> next = temp;
return list2;

}
```



This Question assumes that you are using a Circular Singly Linked List

(b) (5 points) Assume the singly linked list formed in (a) above. Write a recursive function "Sum_recursive" that returns the total number of nodes in the list.



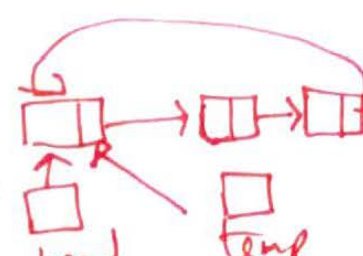
← Circularly Singly Linked List

for Recursion assume a pointer to tail is passed

```
int sum_recursive (Node* head, Node* tail){
   if (head == tail)
        return head→data;
   return head→data + sum_recursive(head→next, tail
}
```

(c) (5 points) Assume the singly linked list formed in (a) above. Write an iterative function "Sum_iterative" that returns the total number of nodes in the list.

```
int   sum_iterative (Node* head) {
 // head means pointer to the first Node.
  Node* temp = head; int sum = 0;
  if (temp != NULL) {
      head = head→next;
      sum = sum + temp→data;
   while ( head != temp) {
        sum = sum + head→data;
        head = head →next;
     }
 } return sum;
```



(d) (5 points) Assume the singly linked list formed in (a) above. Write an iterative function "destroy_iterative" that receives the address of the head node and frees' the memory of all nodes in the linked list.

```
void  destroy (Node** head) {
  Node* temp1 = *head;   Node* temp2;
  while ( *head→next != temp1) {
        temp2 = *head→next;
        *head→next = *head→next→next;
        free(temp2);
   }
   free (temp1);
}  *head = = NULL)
```

(e) (5 points) Assume the singly linked list formed in (a) above. Write a recursive function "destroy_recursive" that receives the address of the head node and frees' the memory of all nodes in the linked list.

void destroy_recursive(Node** head, Node** tail){
  if (*head == *tail){
    free(*head); *head = NULL; *tail == NULL;
    return;
  }
  destroy_recursive(&(*head->next), &(*tail));
  return;
}

(f) (5 points) Assume the singly linked list formed in (a) above. Write a function "InsertHead" that inserts nodes to the front of the linked list.

void InsertHead(Node** head, int item){
  Node* new = (Node*) malloc(sizeof(Node));
  ~~if (new != NULL){~~ ~~if (*head == NULL){~~
  if (new != NULL){
    new->data = item;}
    new->next = *head;
  }
  *head = new;
}

~~if(*head==NULL)~~
~~[illegible]~~
~~else~~

(g) (5 points) Assume the singly linked list formed in (a) above. Write an iterative function "InsertTail" that inserts nodes to the tail of the linked list.

Check class Code

Question 2: STACK: Part I

[30 pts]

(a) (12 points) You are given the following requirements for a stack abstract data type:

(a) It must be able to set up enough memory to accommodate the stack

(b) It must be possible to make a stack empty, i.e. without freeing the memory.

(c) It must be possible to push a given element on to a stack.

(d) It must be possible to pop the topmost element from a stack.

(e) It must be possible to free all the memory used by the stack.

Write a contract for a stack abstract data type. Express your contract in the form of an application programming interface (what we called API in class) using an opaque object, with a comment specifying the expected behavior of each function. Note: You shall need to specify the opaque object before using it.

typedef void* MY_STACK;
MY_STACK stack_init_default(); // initialize the stack
void clear_stack(); // make stack Empty
Status stack_push(MY_STACK hStack, int k);
Status stack_Pop(MY_STACK hstack);
void destroy(MY_STACK* phStack);

(b) (3 points) Briefly describe a possible representation for a stack.

ANY

① for Unbounded stack — Vector elementary data type, NOTE use a top fla

② Unbounded Stack → Singly linked list

(c) (5 points) For the describe representation in (b above, write c code that fully represents the stack structure.

```c
typedef struct stack {
    int top;
    int capacity;
(void *) int* data
} STACK;
```

OR

```c
typedef struct node Node;
struct node {
    int item;
    Node* next;
}
typedef struct stack {
    Node* top;
} STACK;
```

(d) (10 points) Given the stack representation you gave in (c) and your API (in a), complete the definition of the function that reserves enough memory for the stack structure, i.e. an initialization function. This function should return an opaque handle.

(USING VECTOR ARRAY)

```c
MY_STACK stack_init_default () {
 STACK* pstack = (STACK*) malloc (sizeof(STACK);
 if (!pstack)
        return Null;

 pstack -> top = 0;
 pstack -> capacity = 7;
 pstack -> data = (int*) malloc (sizeof (int)* pstack->capacity);
 if (! pstack -> data) {
    free (pstack)
    pstack = NULL;
 }
 return (MY_STACK) pstack;
}
```

(e) (10 points) A stack machine has instructions that push integers on to a stack and pop integers off the stack. A typical stack machine has instructions such as those summarized in Table ?? below.

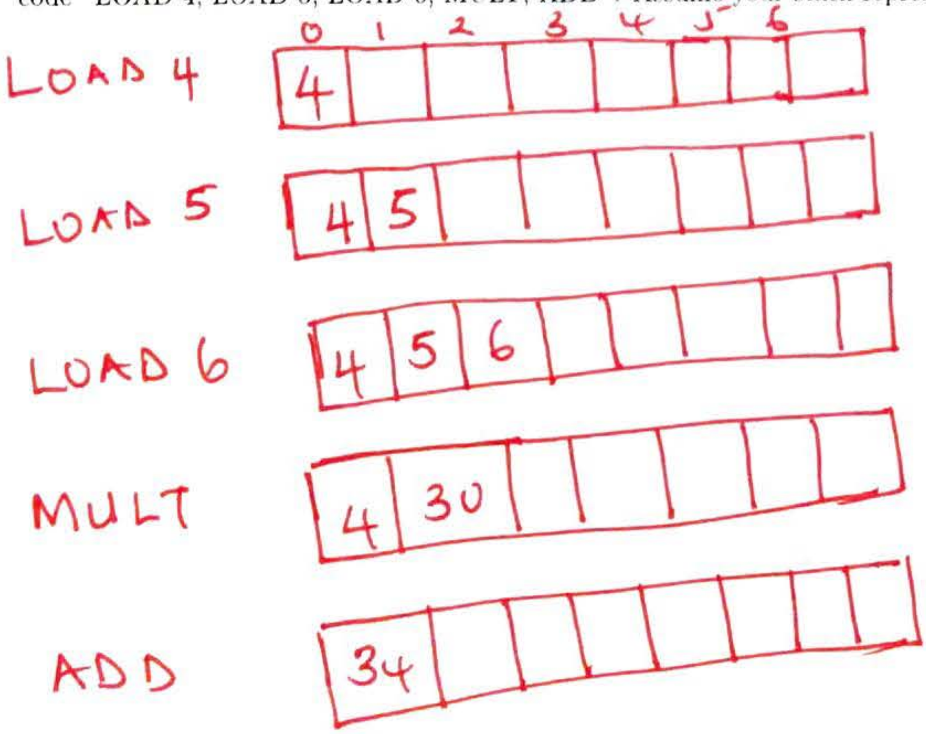
Instruction	Effect
LOAD i	Push the integer i on to the stack.
ADD	Pop two integers off the stack, add them, and push the result back on to the stack.
SUB	Pop two integers off the stack, subtract the top most integer from the integer, and push the result back on to the stack.
MULT	Pop two integers off the stack, multiply them, and push the result back on to the stack.

Table 1: Summary of stack machine instructions

A stack machine makes it easy to evaluate complicated expressions. Any integer expression can be translated to stack machine code. After the code is executed, the stack will contain a single integer, which is the result of evaluating the expression. For example:

Expression	Stack machine code	Expected result
$4 + (5 \times 6)$	LOAD 4; LOAD 5; LOAD 6; MULT; ADD	+34
$2-(3 \times 4) + 5$	LOAD 2; LOAD 3; LOAD 4; MULT; SUB; LOAD 5; ADD	-5
$(2-3) \times 4 + 5$	LOAD 2; LOAD 3; LOAD 4; MULT; SUB; LOAD 5; ADD	+1

Draw diagrams showing the contents of the stack after executing each instruction in the stack machine code "LOAD 4; LOAD 5; LOAD 6; MULT; ADD". Assume your stack representation of part (C).



Question 3: Stack: Part II

[10 pts]

(a) (4 pts) Suppose the following code from the book is executed. Write the order of the integers in myStack (from bottom to top):

```
Stack myStack = Stack initStack();
push(myStack, 8);
push(myStack, 11);
push(myStack, 3);
push(myStack, 5);
```

BOTTOM TOP

8 11 3 5

(b) (4 pts) Suppose the code continues from (a) above. Write the order of the integers in myStack (from bottom to top).

```
pop(myStack);
push(myStack, 4);
push(myStack, 7);
push(myStack, 10);
pop(myStack);
```

BOTTOM

8 11 3 4 7 TOP

(c) (2 pts) (circle answer) What is an abstract data type?

(a) a data type written in pseudocode only

(b) a data type the programmer can use without knowing its underlying implementation

(c) a data type that can change during the execution of a program

(d) a data type's underlying implementation

Question 4: Queue: Array based

[15 pts] Queues are a FIFO data structure and allow items that are first in the queue to exit first from the queue. Think of a line of waiting people. Refer to the lecture code for queues with an array as the underlying representation. Update the values the queue, head, and tail as items are enqueued and dequeued.

You may also want to keep track of the items in the queue as a simple list, crossing out dequeued items

```
#define MAXQ 8  // max for the queue
typedef char QueueData;  // storing chars into the queue

Queue q = initQueue();  // done for you
```

Recall We can add data as: tail ++
tail = tail % capacity
A [tail] = item;
in this case we add from 1

Initial queue:

Tail = 0   Head = 0

'junk'	'junk'	'junk'	'junk'	'junk'	'junk'	'junk'	'junk'

(a) (4 points) Draw the array diagram showing the contents of the queue after the following operations. Where is the Head and the Tail, indicate on your drawing.

```
enqueue(q, 'A');
enqueue(q, 'C');
enqueue(q, 'G');
enqueue(q, 'T');
```

Junk	A	C	G	T	Junk	Junk	Junk

(b) (8 points) Draw the array diagram showing the contents of the queue after the following operations. Where is the Head and the Tail, indicate on your drawing.

```
enqueue(q, 'R');
QueueData remove = dequeue(q);
enqueue(q, 'W');
remove = dequeue(q);
enqueue(q, 'X');
enqueue(q, 'B');
enqueue(q, 'H');
remove = dequeue(q);
```

0	1	2	3	4	5	6	7
B	H ~~A~~	~~C~~	~~G~~	T	R	W	X
	Tail		Head				

(c) (3 points) What does q have in it now as a list of items?

T R W X B H

NOTE: We shall other implementations in class.

Question 5: Queue: Linked list_based

[23 pts]

```
struct node;
typedef struct node QNode;

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

typedef struct queue{
   QNode* front;
   QNode* rear;
}Queue;
```

Use this representation to answer the questions below:

(a) (6 points) Write a function called 'MakeNode' that on invocation creates a new node and assigns an element, k, and returns a pointer to the created node.

```
Node*   MakeNode (int k) {
 QNode* newNode = (QNode*) malloc(Size of(QNode));
  If (newNode) {
      newNode -> data;
      new Node -> next = NULL;
  }
} return newNode;
```

(b) (9 points) Write a function called 'enqueue' that on utilizes the 'MakeNode' created in (A) above, to add a new Node to the Queue. Your function should return a Status. (Assume this has been specified in a status.h file).

```
Status   enqueue (QUEUE hQueue, int k) {
  Queue* pQueue = (Queue*) hQueue;
  QNode* new = MakeNode( k),                assuming new is not NU
  if ( pQueue -> front == NULL) {
        pQueue -> front = pQueue -> rear = new;
  }
else{ pQueue -> rear -> next = new;
      pQueue -> rear = new;
      return Sucess;
}
```

NOTE.
if new == N
return Fa

(c) (8 points) Write a recursive function 'destroy' that frees all the memory utilized by the Queue. The function should take an address of a handle of a queue opaque object and does not return anything. NOTE: Do not assume the availability of a dequeue function.

```
void destroy(MY_QUEUE * phqueue) {
 Queue* pQueue= (Queue*) * phQueue;
 QNode* temp;
 if ( pQueue == NULL) {
   while ( pQueue -> front != NULL) {
      temp = pQueue -> front;
      pQueue -> front = pQueue -> front -> next;
      free (temp);
   }
   if (pQueue -> front == NULL)
      pQueue -> rear = NULL;}
 }
 *phQueue = NULL;
}
```

Question 6: Express Evaluation: Vectors

[10 pts] Evaluate the following expressions assuming 32 bit integers and 32 bit pointers. Variables are declared as listed but after some unknown number of operations the current state of the memory is given by the supplied memory diagram.

```
struct my_vector
{
    int size;
    int capacity;
    int* data;
};
typedef struct my_vector My_vector;
My_vector v;
My_vector* t;
```

Variable Name	Address	Memory Value
v	8000	2
	8004	8016
	8008	9004
*t	8012	9028
	8016	9032
	8020	9020
	...	...
	9000	3
	9004	9016
	9008	5
	9012	100
	9016	87
	9020	9008
	9024	101
	9028	1
	9032	9000
	9036	9016

(a) *v.data*; → 9004

(b) (*v.data*[2]) << 2; $100 * 2^2 = 400$

(c) &*t*; 8012

(d) *t* − > *data*[1]; 9008

(e) (**t*).*capacity*; 9000

Question 7: Express Evaluation: Linked Lists

[10 pts] Evaluate the following expressions assuming 32 bit integers and 32 bit pointers. Variables are declared as listed but after some unknown number of operations the current state of the memory is given by the supplied memory diagram.

```
struct node
{
  int data;
  struct node* other;
};
typedef struct node Node;
Node v;
Node* p;
```

Variable Name	Address	Memory Value	
v	8000	2	
	8004	8016	
	8008	9004	
p	8012	9028	
	8016	9032	
	8020	9020	
	...	...	
	9000	3	0
	9004	9016	
	9008	5	1
	9012	100	
	9016	87	2
	9020	9008	
	9024	101	3
	9028	1	
	9032	9000	
	9036	9016	

(a) *v.other*: 8016

(b) (*v.other*− > *data*) + 1; 9033

(c) (*p*− > *other*− > *data*) << *v.data*; $3 * 2^2 = 12$

(d) *p*− > *other*[3].*data*; 101

(e) *p*− > *other*− > *other*− > *other*− > *other*

9020 9016 9008 <u>100</u>