Problem Number(s)	**Possible Points**	**Earned Points**	**Course Outcome(s)**
1	10		
2	10		
3	10		
4	10		
5	16		
6	10		
7	15		
	TOTAL POINTS 81		

Exam 1
100 points

1. Stacks (10 points)

   (a) Draw a sequence of diagrams, one for each problem segment, that represent the current state of the stack after each labeled set of operations. If an operation or instruction produces output then indicate what that output is. hStack is the handle of a stack opaque object that can hold characters. There is no diagram for init or destroy.

```
hStack = stack_init_default();
     i.   stack_push(hStack, 'a');
    ii.   stack_push(hStack, 'b');
   iii.   printf("%c ", stack_top(hStack)); stack_pop(hStack);
    iv.   printf("%c ", stack_top(hStack)); stack_push(hStack,
          'c');
     v.   stack_push(hStack, 'd');  stack_push(hStack, 'e');
    vi.   printf("%c ", stack_top(hStack)); stack_push(hStack,
          'f');
   vii.   stack_pop(hStack); stack_push(hStack, 'g');
stack_destroy(&hStack);
```

i. a

ii. b
a

iii. b a

iv. a c
a

v. e
d
c
a

vi. e f
e
d
c
a

vii. e g
d e
c d
a c
a

2. Queues (10 points)
   (a) Draw a sequence of diagrams, one for each problem segment, that represent the current state of the queue after each labeled set of operations. If an operation or instruction produces output then indicate what that output is. We will use the function enqueue to add to the queue and serve to remove from the queue. hQueue is the handle of a queue opaque object that can hold characters. There is no diagram for init or destroy.

```
     hQueue = queue_init_default();
  i.    queue_enqueue(hQueue, 'a');
 ii.    queue_enqueue(hQueue, 'b');
iii.    printf("%c, " queue_front(hQueue));
        queue_enqueue(hQueue, 'c');
 iv.    printf("%c ", queue_front(hQueue));
        queue_enqueue(hQueue, 'd');
  v.    queue_serve(hQueue); queue_serve(hQueue);
 vi.    printf("%c ", queue_front(hQueue));
        queue_enqueue(hQueue, 'e');
vii.    queue_serve(hQueue); queue_serve(hQueue);
     hQueue->destroy(&hQueue);
```

i. a

ii. a b

iii. a a b c

iv. a a b c d

v. c d

vi. c c d e

vii. d e e

3. (10 points) **Expression Evaluation**. 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
	9004	9016
	9008	5
	9012	100
	9016	87
	9020	9008
	9024	101
	9028	1
	9032	9000
	9036	9016

a. `v.other;` ____________

b. `(v.other->data) + 1;` ____________

c. `(p->other->data) << v.data;` ____________

d. `p->other[3].data;` ____________

e. `p->other->other->other->other` ____________

4. (10 points) Write a function called destroy that takes a Node pointer to the head of a list and will free up the memory associated with each node in the entire list.

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

```
void destroy(Node** pHead)
{
    Node* temp;
    while (*pHead != NULL)
    {
        temp = *pHead;
        *pHead = (*pHead)->next;
        free(temp);
    }
}
```

```
/* Recursive Function to delete the entire linked list */
void destroyRecursive(Node* head)
{
    if (head == NULL){
        return;
    }

    destroyRecursive(head->next);
    free(head);
}
```

5. (16 points) Given the following

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

(a) Write a recursive function called sum that given a Node pointer to the head of a list will return the sum of all data in the linked list.

```
// function to recursively find the sum of
// nodes of the given linked list
int sumOfNodes(Node* head)
{
    // if head = NULL
    if (!head){
        return;
    }

    int sum = head->data;
    // recursively compute the sum of data of the remaining nodes
    sum += sumOfNodes(head->next);

    return sum;
}
```

(b) Write the iterative version of the sum function that given a Node pointer to the head of a list will return the sum of all data in the linked list.

```
// function to find the sum of
// nodes of the given linked list
int sumOfNodes(Node* head)
{
    Node* ptr = head;
    int sum = 0;
    while (ptr != NULL) {
        sum += ptr->data;
        ptr = ptr->next;
    }

    return sum;
}
```

6. (10 points) Write a function called copy_list that, given a Node pointer to the head of a list will return a Node pointer containing the address of the head node of a new list that is an exact copy of the original list. Your copy should be independent of the first list and not share any nodes. You may write an iterative or recursive version of your function.

7. (15 points) In class we created an opaque object for a type called MY_VECTOR that had an internal structure called My_vector consisting of an integer size, an integer capacity, and an integer pointer data that held the address of the first element of a dynamic array of integers. Write a function called my_vector_init_default( ) that initializes the vector to have a size of zero, capacity of seven and an appropriate value in the data pointer. Your function should return the address of an opaque object upon success and NULL otherwise.