COMP1020 Spring 2018 Exam 1- Computing II Name: Answer Key

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 (1)

ii) b (TOP) / a (1)

iii) OUTPUT : b (1)
a (1)

iv.) OUTPUT : a (1)
c
a (1)

v.)
e
d
c
a (1)

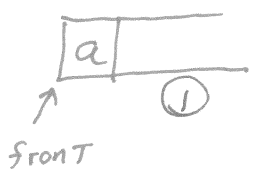
vi.) OUTPUT : e (1)
f
e
d
c
a (1)

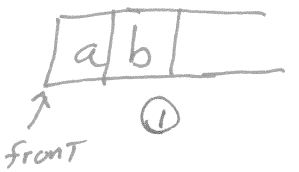
vii)
g
e
d
c
a (1)

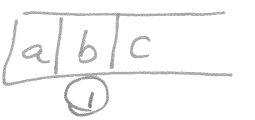
Diagram can have different format and orientation as long as TOP is labeled.

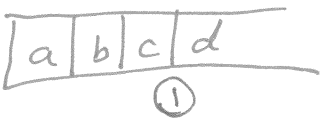
2. Queues (10 points)
   (a) Draw a sequence of diagrams, one for each problem segment, that represent the current state of the queue after each labeled set of operations. If an operation or instruction produces output then indicate what that output is. We will use the function enqueue to add to the 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.

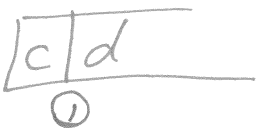
```
hQueue = queue_init_default();
 i.   queue_enqueue(hQueue, 'a');
 ii.  queue_enqueue(hQueue, 'b');
 iii. printf("%c, " queue_front(hQueue));
      queue_enqueue(hQueue, 'c');
 iv.  printf("%c ", queue_front(hQueue));
      queue_enqueue(hQueue, 'd');
 v.   queue_serve(hQueue); queue_serve(hQueue);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

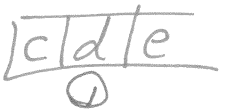
i) 

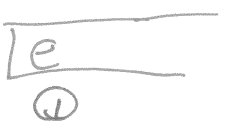
ii) 

iii) output: a (1)


iv) output: a (1)


v) 

vi) output: c (1)


vii) 

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	] [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 (2)

b. `(v.other->data) + 1;` 9033 (2)

c. `(p->other->data) << v.data;` 12 (2)

d. `p->other[3].data;` 101 (2)

e. `p->other->other->other->other` 100 (2)

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

(5) does it work?

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.

```
int sum(Node* head)
{
    if(head != NULL)
      return head->data + sum(head->next);
    else return 0;
}
```

(1) (1) (1) (1) (1) (3) Does it work?

(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.

```
int sum(Node* head)
{
    int total = 0;
    while(head != NULL)
    {
        total += head->data;
        head = head->next;
    }
    return total;
}
```

(1) (1) (1) (1) (1) (1) (2) Does it work?

same Node as 4 & 5

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.

Recursive way

```
Node * copy_list (Node* head)                    (1)
{
    Node * temp; =NULL                           (1)
    if (head != NULL)                            (1)
    {
        temp= (Node*) malloc (sizeof (Node));    (1)
        if (temp!= NULL)                         (1)
        {
            temp -> data = head -> data;         (1)
            temp -> next = copy_list (head -> next);  (1)
        }
        return temp;                             (1)
    }
}
```

(2) Does it work?

Does not need to recover well from out of memory errors.

For iterative method (5) Does it work (5) Partial credit to distribute.

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.

```
MY_VECTOR my_vector_init_default (void)
{
    My_vector* pVector = (My_vector*) malloc (sizeof(My_vector));
    if (pVector != NULL)
    {
        pVector->size = 0;
        pVector->capacity = 7;
        pVector->data = (int*) malloc (sizeof(int) * pVector->capacity);
        if (pVector->data == NULL)
        {
            free (pVector);
            return NULL;
        }
    }
    return pVector;
}
```