COMP1020 Summer 2019 Exam 1- Computing II Name: PHONG VO

Problem Number(s)	Possible Points	Earned Points	Course Outcome(s)
1	10	7	Understanding data structures in C
2	10	10	Stacks
3	10	10	Queues
4	10	4	Linked Lists and memory utilization
5	10	10	Implementation of functions in a linked list
~~5~~ 6	16	14	Implementation of functions in a linked list / Recursion/Iteration
~~6~~ 7	15	11	Implementation of initializer functions for vector/stack
	TOTAL POINTS 81	66	

COMP1020 Summer 2019 Exam 1- Computing II Name: Phong Vo

Exam 1
100 points

1. Questions bonanza (10 points)
   a. What is a void pointer?
   is a pointer that holds a value (non) type of memory address.

   b. What is an opaque handle
   - has opaque data type in a ".h" file what do you mean?
   - must declare the return datatype at each function that uses opaque handle.

   c. What is an abstract data structure?
   ADS is a data structure that a function may access without knowing of datatype of structure. (2)

   d. Differentiate an Array from a Linked List

Array	Linked list
- has finitely determined capacity	- has unfinitely capacity.
- has size (number of elements)	- has number of nodes (2)
- may use pointers techniques	- may use pointer techniques
- properties: data, size, capacity	- properties: data, next node

   e. Compare and contrast Stack and Queue data structures

Stack	Queue
- FILO philosophy (2)	- FIFO philosophy.
- may use pointer techniques	- may use pointer technique.
- properties: capacity, top, data	- properties: data, front, tail, capacity

2. 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 integers.

```
hStack = stack_init_default();
      i.   stack_push(hStack, 13);
     ii.   stack_push(hStack, 17);
    iii.   printf("%d ", stack_top(hStack)); stack_pop(hStack);
     iv.   printf("%d ", stack_top(hStack)); stack_push(hStack,
           21);
      v.   stack_push(hStack, 29);  stack_push(hStack, 35);
     vi.   printf("%d ", stack_top(hStack)); stack_push(hStack,
           42);
    vii.   stack_pop(hStack); stack_push(hStack, 9);
stack_destroy(&hStack);
```

i

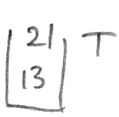
ii

iii

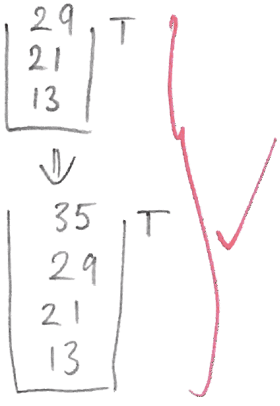
output: 17

iv

output: 13

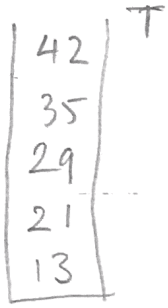


v

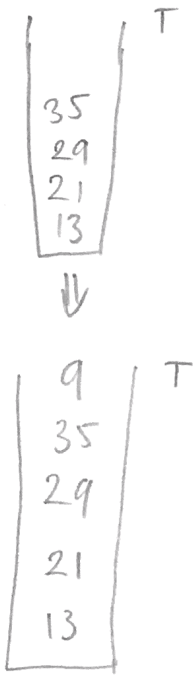


vi

output: 35



vii

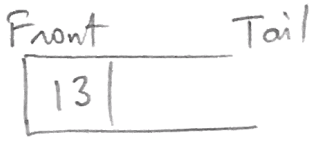


10

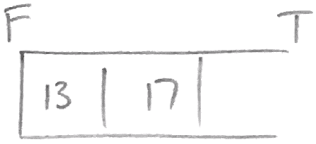
3. Queues (10 points)
   (a) Draw a sequence of diagrams, one for each problem segment, that represent the current state of the queue after each labeled set of operations. If an operation or instruction produces output then indicate what that output is. We will use the function enqueue to add to the queue and serve to remove from the queue. hQueue is the handle of a queue opaque object that can hold integers.

```
hQueue = queue_init_default();
 i.   queue_enqueue(hQueue, 13);
ii.   queue_enqueue(hQueue, 17);
iii.  printf("%d, " queue_front(hQueue));
      queue_enqueue(hQueue, 21);
iv.   printf("%d ", queue_front(hQueue));
      queue_enqueue(hQueue, 29);
 v.   queue_dequeue(hQueue); queue_dequeue(hQueue);
vi.   printf("%d ", queue_front(hQueue));
      queue_enqueue(hQueue, 35);
vii.  queue_dequeue(hQueue); queue_dequeue(hQueue);
hQueue->destroy(&hQueue);
```

i

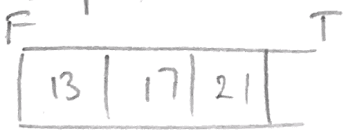


ii



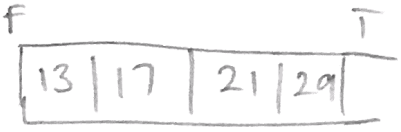
iii

output: 13

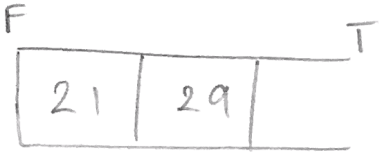


iv

output 13

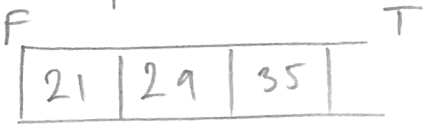


v

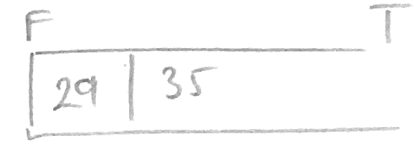


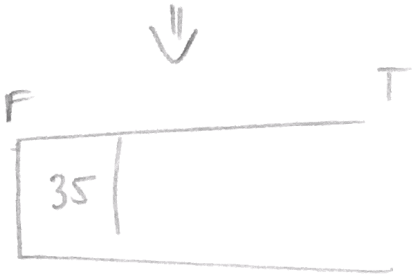
vi

output: 21



vii





4. (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	3
	8004	9000
	8008	9004
p	8012	9028
	8016	9032
	8020	9020
	…	…
	9000	74
	9004	9016
	9008	5
	9012	100
	9016	87
	9020	9008
	9024	101
	9028	1
	9032	8000
	9036	9016

number of bytes added to (v. after a pointer) = 9000 + 1x(8) = 9008

a. `v.other +1;` 9001 ×

b. `(v.other->data) +1;` 74 + 1 = 75 ✓

c. `(p->other->data) << v.data;` 3 << 3 = 3x8 = 24 ✓

d. `p->other[1].data;` 9000 × 9004

e. `p->other->other->other->other` 9008

8000 → 9000 → 9016 → 9008

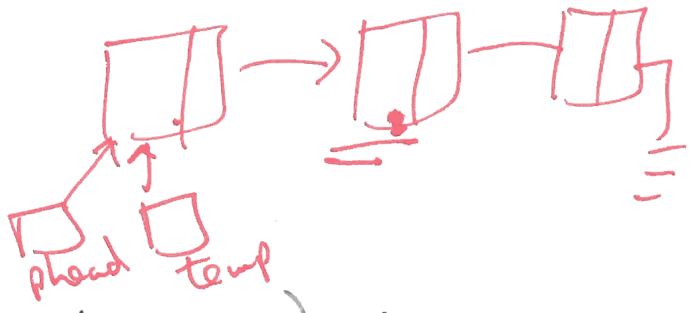
5. (10 points) Write a function called destroy that takes a Node pointer to the head of a list and will free up the memory associated with each node in the entire list.

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

```
void destroy(Node* pHead) {

    if (pHead == NULL) {
        return;
    }

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



6. (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 nodes in the linked list.

```
int Sum (Node* pHead) {
    if (!pHead) {
        return 0;
    }
    return (pHead->data + Sum(pHead->next));
}
```

number of nodes ≠ total of data values

```
return ( 1 + Sum(pHead->next));
```

(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 nodes in the linked list.

```
int Sum (Node* pHead) {
    if (!pHead) {return 0;}
    int sum = 0;
    Node* temp;
    while (!pHead->next) {
        temp = pHead;
        sum += temp->data;
        pHead = pHead->next;
    }
    return sum;
}
```

Sum++;

7. (15 points) In class we created an opaque object 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 ( ) {

    Struct  My_vector {
        int  size     ;
        int  capacity ;
        int* data     ;
    };
    typedef  struct  My_vector  pVector;

    if (pVector == NULL) {
        printf("Unable to allocate memory\n");
        return NULL;
    }
    else {
        pVector->size = 0;
        pVector->capacity = 7;
        pVector->data = (int*) malloc (sizeof(int) * pVector->capacity);
    }
    if (pVector->data != NULL) {
        return pVector;
    }
}
```

should use a pointer

-1

pVector is not necessary if there is Function

No return -1