

Name:

Huang, D.

ID#

01521888

COMPUTING II**MIDTERM 1**

You have 50 minutes to complete the midterm. The total number of points is 88. You are not allowed to use any books, notes, calculator, or electronic devices. Write your answer carefully and clearly. Incorrect answers will receive little to no points. When you are asked to write a program the program should be clear and include indentations and comments to make it easier to read. Be sure to *state any assumptions* on which you have based your answers.

Problem Number(s)	Possible Points	Earned Points
1	14	14
2	14	12
3	10	8
4	5	4
5	10	7
6(a)	10	9
6(b)	6	6
6(c)	17	1
7	2	2
8(Extra)	2	1
	TOTAL POINTS	64
	88 (+2)	

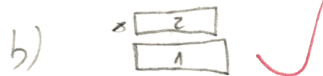
Stack

1. (14 points) 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.

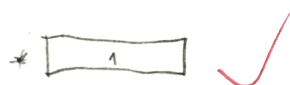
```
hStack = stack_init_default();
```

- `stack_push(hStack, '1');`
- `stack_push(hStack, '2');`
- `printf("%d ", stack_top(hStack)); stack_pop(hStack);`
- `printf("%d ", stack_top(hStack)); stack_push(hStack, '3');`
- `stack_push(hStack, '4');` `stack_push(hStack, '5');`
- `printf("%d ", stack_top(hStack)); stack_push(hStack, '6');`
- `stack_pop(hStack); stack_push(hStack, '5');`

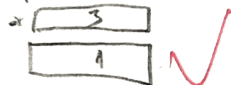
```
stack_destroy(&hStack);
```



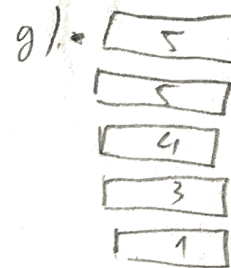
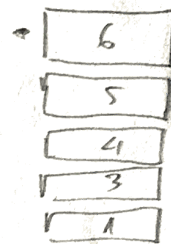
- c) print the top item of the stack: 2 ✓
and then remove it.



- d) print the top item of stack: 1 ✓
push item 3 to stack



- f) print the top item of stack: 5 ✓
push item 6 to stack



Queues

2. (14 points)



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.

```
hQueue = queue_init_default();
```

- `queue_enqueue(hQueue, '1');`
- `queue_enqueue(hQueue, '2');`
- `printf("%c ", queue_front(hQueue)); queue_enqueue(hQueue, '3');`
- `printf("%c ", queue_front(hQueue)); queue_enqueue(hQueue, '4');`
- `queue_dequeue(hQueue); queue_dequeue(hQueue);`
- `printf("%c ", queue_front(hQueue)); queue_enqueue(hQueue, '5');`
- `queue_dequeue(hQueue); queue_dequeue(hQueue);`

```
hQueue->destroy(&hQueue);
```

a)  

b)  

c) print the front item in queue: 2 (1)

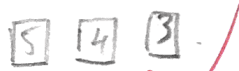


d) print the front item in queue: 3 (1)



e)  

f) print the front item in queue: 4 (3)



g)  

Expression Evaluation

3. (10 points) 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	42
	9004	9016
	9008	5
	9012	100
	9016	87
	9020	9008
	9024	101
	9028	1
	9032	8000
	9036	9016

a. v.data;

3 ✓

b. v.other->data;

42 ✓

c. p->other->data;

3 ✓

d. p->other[2]->data;

~~87~~ Invalid.

e. p->other->other->other->data

87 ✓

Stack

4. (5 points) Write a C declaration for a stack of floating point numbers stored in an array.

```

struct stack
{
    int capacity;
    int size;
    float data;
}

```

⇒ answer

typedef struct stack Stack;

5. (10 points) Using the above stack declaration, write a C function that implements the pop operation including printing a message and exiting if stack underflow occurs. You may assume the existence of an empty() function that takes a pointer to a stack as an argument.

void pop (Stack *p, Head *size)

```

{
Node *temp = (Node *) malloc(sizeof(Node));

```

```

    if (empty(p))

```

```

    {
        printf("Nothing to pop from the stack!\n");

```

```

        return FAILURE;
    }

```

```

    else

```

```

    {
temp = p->next;

```

```

free(p->next);

```

```

p->next = temp;

```

```

    while (int i = 0, i < n, i++)

```

```

    { Head[i+1] = Head[i]; }

```

```

    size--;

```

```

}

```

-3

Linked list

6. (33 points) Given the following:

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

- a. (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.

```
void destroy (Node * * pHead)  
{  
    Node * temp = (Node *) * pHead;  
  
    while (temp != NULL)  
    {  
        temp = temp->next;  
        free (*pHead);  
        pHead = temp;  
    }  
    return -1;  
}
```


- b. (6 points) 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 * pHead )  
{  
    Node * temp = Node * ( pHead )  
    if ( (temp -> next) == NULL )  
    {  
        return temp -> data;  
    }  
    return (temp -> data) + sum ( temp -> next );  
}
```

1/17

- c. (17 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.

```
Node * copy_list ( Node ** pHead ).
```

```
{ Node * temp = (Node *) * pHead;
```

```
Node * new Node = (Node *) malloc (size of (Node));
```

```
while ((temp -> next -> data) != NULL)
```

```
{ new Node = temp;
```

```
new Node = new Node - next }
```

```
return new Node;
```


- 2 7. (2 points) Following is an *incorrect* pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
declare a character stack
while (more input is available)
{
    read a character
    if (the character is a '(' )
        push it on the stack
    else if (the character is a ')' and the stack is not empty)
        pop a character off the stack
    else
        print "unbalanced" and exit
}
print "balanced"
```

Which of these unbalanced sequences does the above code think is *balanced*?

- a. ((()
- b. ()()
- c. (())
- d. (())()

Extra

8. (2 points) What is a stack?

stack is a data structure that we use to store data in a particular order. two function that are used to insert and remove items from stack. It follow the principle First in Last out

pop and push - 1