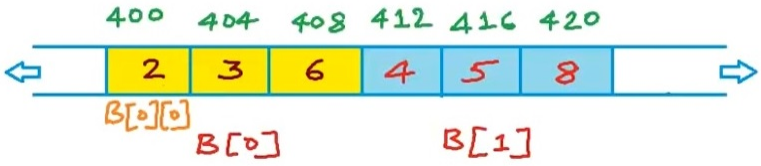
BBM 201
Exam 1
Time: 60 minutes

Answer the questions in the spaces provided on the question sheets. KEEP YOUR CELLPHONE TURNED OFF UNTIL THE EXAM IS OVER.

Name: _____

1. (15 points) If B is the array shown with its address above each node, write what the following lines of a program will print.

`int B[2][3]`:



   (a) (3 points) `printf`("%d", &B[1][1]);

   (b) (3 points) `printf`("%d", B+1);

   (c) (3 points) `printf`("%d", *(*B+1));

   (d) (3 points) `printf`("%d", B[1]+2);

   (e) (3 points) `printf`("%d", *(B+1)+1);

2. (12 points) Provide the time complexities below for the worst-case.

   (a) (4 points) Write the time complexity of inserting and deleting an element of an array.

   (b) (4 points) Write the time complexity of inserting and deleting an element of a linked-list.

   (c) (4 points) Write the time complexity of push(x), pop(), top() and IsEmpty() operations for an element x of a stack.

3. (9 points) How many bytes do the following data structures occupy in the memory?

   (a) (3 points) An INTEGER array of size 5.

   (b) (3 points) A CHARACTER doubly linked-list of size 5.

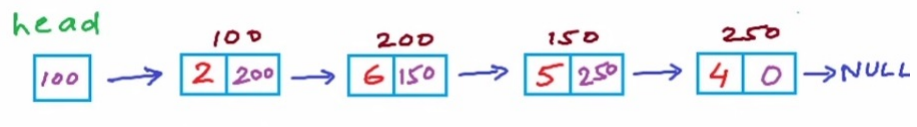
   (c) (3 points) A DOUBLE linked-list of size 5.

4. (12 points) (a) (6 points) Show how a stack looks after each of the following operations is applied consecutively. Write your answer in the stack column of the table below. Assume that this stack is initially empty.

   (b) (6 points) Write what `IsEmpty( )` and `Top( )` would return for the current stack if it was executed after each operation in the table below.

operation	current stack	IsEmpty( )	Top ( )
Push(2);			
Push(4);			
Pop( );			
Push(7);			
Push(3);			
Pop( );			

5. (16 points) (a) (12 points) Write each of the recursive calls made in their order when `Abc(head)` is executed in `main ()`. The address of each node is written above that node in this linked list.

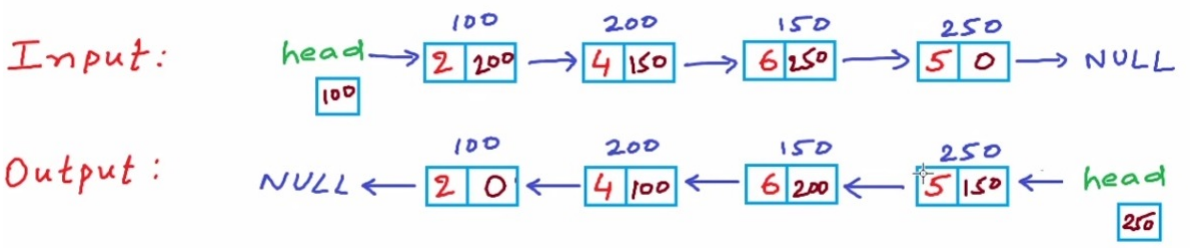



```
struct Node{
    int data;
    struct Node * next;
};
void Abc(struct Node * p)
{
    if(p == NULL)
    {
        return;
    }
    Abc(p → next);
    printf("%d", p → data);
}
int main()
{
    struct Node * head = NULL;
    head = Insert(head, 4);
    head = Insert(head, 5);
    head = Insert(head, 6);
    head = Insert(head, 2);
    Abc(head);
}
```

(b) (4 points) Write what `main()` will return in this program program.
(Note: `Insert(head, data)` inserts a new node in the beginning by putting `data` as the value of the head node.)

6. (36 points) The Reverse function below reverses any given linked list by using iteration method. The input argument of the Reverse function is the head of the linked list. In the following linked list, the number above each node indicates its address in the memory.

   (a) (20 points) If we execute the Reverse function for this linked list, redraw the changes on the input linked list after EACH ITERATION STEP of the while loop by showing the changes on the NODES and on the LINKS (arrows).




```
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* Reverse(struct Node* head) {
    struct Node *current,*prev,*next;
    current = head;
    prev = NULL;
    while(current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    return head;
}
```

   (b) (16 points) Write the value of addresses stored in "current", "prev" and "next" initially and after each iteration step of the while loop in the table below.

iteration	current	prev	next
1			
2			
3			
4			
5			