SOFTWARE CPC 2

MAX MARKS: 45

Section A
Basics of programming and OOP
(20 marks)

1) Can a static method use non static members? (2)
2) What is the default access modifier in a class? (2)
3) Name any three operators that cannot be overloaded. (2)
4) Is it possible to have a recursive inline function? Explain (2)
5) Out of all basic arithmetic operations (+, -, *, /,) and comparison operations, which are valid on pointers? (2)
6) The following postfix expression with single digit operands is evaluated using a stack. Write the expected output. (3)

```
6 2 3 * / 2 3 * ^
```

7) Suppose a stack is to be implemented with a linked list instead of an array. What would be the effect on the time complexity of the push and pop operations of the stack implemented using linked list (Assuming stack is implemented efficiently)? (2)
8) Mention any two applications each of Queue and Stack Data structures (In Computer Science) (5)

Section B
Data structures and algorithms
(25 marks)

Note: Marks will be awarded for correct and efficient code. A well written code with a better time and space complexity will fetch you more marks.

Assume the definition of “node” as the following structure for Q1 and Q2.

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

1) Write a function that takes as argument the head to a singly linked list and checks if a cycle exists in it. If a cycle is found, print the value of the node where the cycle begins. If cycle is not found, print “No cycle found” without the quotes.
Function prototype: (5)

```
void cycle_detect(node * head)
{
        //Definition here
}
```

```
Example input list: 1->2->3->4->5
                          ^     |
                          |     *    (Not the best ASCII artist :( ) <- comment1
                          7<-6<-5
Output : 3 (3 is the node where the cycle begins)

Example 2: 1->2->3->NULL
Output : No cycle found
```

2) Write a function that finds the node at which two linked lists merge. Print the value of the node if they do else print “No merge detected” without quotes.
Function prototype is as follows: (5)

```
        void merge_detect(node * h1, node * h2)
        {
                //Definition here
        }
Example input: (h1) 1->2->3->4->5->NULL
                              ^             (check comment 1 above)
                              |
               (h2)  3->1->7->5
Output: 4 (4 is the node where these two lists merge)

Example 2: (h1) 1->4->3->4->NULL
           (h2) 1->5->3->7->6->NULL
Output: No merge detected
```

3) Design a stack that supports the following operations: (8)

```
        void push(val){} -> Pushes val in the stack in O(1). Prints an error upon
        overflow.
        void pop(){} -> Pops the value on top of the stack in O(1). Prints error
        statement if stack is empty.
        int max_diff(){} -> Returns the maximum difference b/w any two elements
        currently present in the stack in O(1). Returns 0 if there are no or just 1
        element in the stack.
```

You may write and use auxiliary functions and / or data structures.
Example (Assume empty stack in the beginning)

```
        Operation      Stack status          Function return value/ print
        pop()          top->                 Stack is empty (or whatever)
        push(1)        top->1
        max_diff()     top->1                0 (just one element in stack)
        push(3)        top->3->1
        push(2)        top->2->3->1
        max_diff()     top->2->3->1          2 (3 - 1 = 2)
        push(0)        top->0->2->3->1
        max_diff()     top->0->2->3->1       3 (3 - 0 = 3)
        pop()          top->2->3->1
        push(5)        top->5->3->1
        push(3)        top->3->5->3->1
        max_diff()     top->3->5->3->1       4(5 - 1 = 4)
```

4) Implement a queue using only stacks. Assume push(), pop() is already implemented for the stack ADT. You may use multiple stacks if you need to. push(), pop() and get_top() operations must be supported by the queue.

(7)

Stack details

Assume stack class is provided to you and you can declare a stack as follows:

```
stack_class s;
```

You can push values into s as follows:

```
s.push(val)      //Pushes val on top
```

You can get the top value of the stack as follows

```
Int top_val = s.get_top()         //returns value on top
```

You can pop from the stack as follows:

```
s.pop()           //pops the top element
```

ALL THE BEST :)