PES University
Department of Computer Science and Engineering

UE19CS202- Data Structures and its Applications(4-0-0-4-4)
UNIT - 2 : Question Bank

1. Draw a Sequence of stack frames showing the progress of each of the following segments of code. (s is a pointer to a stack of characters, and x, y, z are character variables)

   a)
   ```
   CreateStack(s);
   Push('a',s);
   Push('b', s);
   Push('c', s);
   Pop(&x,s);
   Pop(&y,s);
   Pop(&z,s);
   ```

   b)
   ```
   CreateStack(s);
   Push('a',s);
   Push('b', s);
   Push('c', s);
   Pop(&x,s);
   Pop(&y,s);
   Push(x,s);
   Push(y,s);
   Pop(&z,s);
   ```

   c)
   ```
   CreateStack(s);
   Push('a',s);
   Push('b', s);
   ClearStack(s);
   Push('c', s);
   Pop(&x,s);
   Push('a',s);
   Pop(&y,s);
   Push('b',s);
   Pop(&z,s);
   ```

   d)
   ```
   CreateStack(s);
   Push('a',s);
   Push('b', s);
   Push('c', s);
   while(!StackEmpty(s))
         Pop(&x,s);
   ```

2. Let stack be a stack of integers and x be an integer variable. Use the functions Push, Pop, CreateStack, StackEmpty to write a function that sets x to the top element of the stack **stack** and leaves the top element of stack unchanged. If the stack is empty, the function sets x to maxint.

3. A stack may be regarded as a railway switching network like the one in Below Figure. Cars numbered 1, 2, …, n are on the line at the left, and it is desired to rearrange (permute) the cars as they leave on the right-hand track. A car that is on the spur (stack) can be left there or sent on its way down the right track, but it can never be sent back to the incoming track. For example, if n = 3, and we have the cars 1,2,3 on the left track, then 3 first goes to the spur. We could then send 2 to

the spur, then on its way to the right, then send 3 on the way, then 1, obtaining the new order 1,3,2.

.

4. Use the operations push , pop, stacktop and empty to construct operations which do each of the following
    a) Set i to the second element from the top of the stack , leaving the stack without its top two elements
    b) Set i to the second element from the top of the stack, leaving the stack unchanged
    c) Given an integer n, set i to the nth element from the top of the stack, leaving the stack without its top n elements
    d) Given an integer n, set i to the nth element from the top of the stack, leaving the stack unchanged.
    e) Set i to the bottom element of the stack, leaving the stack empty
    f) Set i to the bottom element of the stack, leaving the stack unchanged (Hint : Use another auxiliary stack)
    g) Set i to the third element from the bottom of the stack.

5. Write an algorithm to determine if an input character string is of the form x C y where x is a string consisting of letters ‘A’ and ‘B’ and where y is the reverse of x (i.e if x= ABABBA , y= ABBABA) at each point you may read only the next character of the string.

6. Write an algorithm to determine if an input character string is of the form aDbDcD...Dz where each string a,b, . . .z is of the form of the string defined in the earlier problem.

7. Show how to implement a stack of integers in C by using an array int s[STACKSIZE] where s[0] is used to contain the index of the top element of the stack, where s[1] through s[STACKSIZE -1] contain the element of the stack. Write a declaration and routines pop, push, empty,,stacktop for this implementation.

8. Implement a stack in C in which each item on the stack is a varying number of integers. Choose a data structure for such a stack and design push and pop routines for it.
9. Design a method for keeping two stacks within a single linear array ${spacesize] so that neither stack overflows until all the memory is used and an entire stack is never shifted to a different location within the array. Write c routines push1, push2, pop1, pop2 to manipulate the two stacks.

10. Transform each of the following expressions to prefix and postfix
    a. A + B - C
    b. (A + B)* (C - D) $ E * F
    c. (A + B)* (C $ (D - E) + F) - G
    d. A + (((B - C) * (D - E) + F) / G) $ (H - J)

11. Transform each of the following prefix expressions to infix
    a) + - A B C
    b) + A - B C
    c) + + A - *$ B C D / + E F * G H I
    d)+ - $ A B C * D * * E F G

12. Transform each of the following postfix expressions to infix
    a) A B + C -
    b) A B C + -
    c) A B - C + D E F - + $
    d) A B C D E - + $ * E F * -

13. Write a program to evaluate an infix string. Use two stacks, one for the operands and the other for operators. Do not first convert infix string to postfix string, but rather evaluate as you go along.

14. Write a routine to accept as input a character string of operators and operands representing a postfix expression and to create the fully parenthesized infix form of the original postfix. For example A B + would be converted to (A + B). A B + C - would be converted to ((A + B) - C).

15. Write a c program to convert
    a. A prefix string to postfix
    b. A postfix string to prefix
    c. A prefix string to infix
    d. A postfix string to infix

16. Write a recursive program to compute
    a. Maximum element of the array
    b. The minimum element of the array
    c. The sum of elements of array
    d. The product of elements of array

e. The average of the elements of the array.

17. Evaluate each of the following using recursive definitions
    a. 6 !
    b. 9 !
    c. 100 * 3
    d. 6 * 4
    e. fib(10)
    f. fib(11)

18. Determine what the following recursive C function computes. Write an iterative function to accomplish the same.

```
int func(int n)
{
    if(n = =0)
        return 0;
    return ( n + func(n-1));
}
```

19. Show how to implement a queue of integers in C by using an array queue[100] , where queue[0] is used to indicate the front of the queue, queue[1] is used to indicate its rear and queue[2] trough queue[99] are used to contain the queue elements. Show how to initialize such an array to represent the empty queue and write the routines remove, insert and empty for such an implementation.

20. Show how to implement a queue in which each item represents a variable number of integers.

21. A deque is an ordered set of items from which items may be deleted at either end and into which items may be inserted at either end. Call the two ends of a deque left and right. How can a deque be represented as a Doubly LInked List. Write for routines remvleft, remvright, insrtleft, insrtright.

22. Define an **input-restricted deque** as a deque for which only operations remvleft, remvright and insertright are valid and **output-restricted queue** as a deque for which only the operations remvleft, insertleft and insertright are valid.

23. Implement an ascending priority queue and its operations pqinsert, pqmindelete and empty using an array

24. Show how to sort a set of input numbers using a priority queue and the operations pqinser, pqmindelete and empty.

25. Implement the routines empty, push,pop using the dynamic storage implementation of a linked stack.

26. Implement the routines empty, push,remove using the dynamic storage implementation of a linked queue.

27. Implement the routines empty, pqinsert and pqmindelete using the dynamic storage implementation of a linked priority queue.