

Data Structures

Mid Term Solutions

Q1. Identify the data structure that allows deletions at both ends of the list but insertion at only one end (1)

1. Stack
2. Priority queues
3. Output restricted queue
4. Input restricted queue

Q2. How many queues are required to implement stack using queue (1)

1. 2
2. 3
3. 1
4. 4

Q3. A stack data structure cannot be used for (1)

1. Handling recursive calls
2. Reverse string
3. Implementation of Ticket reservation system
4. Evaluation of postfix expression

Q4. Evaluate the given postfix expression: $* - + 4\ 3\ 5 / + 2\ 4\ 3$ (1)

1. 4
2. 8
3. 1
4. None of the mentioned

Q5. If the elements A, B, C, D are placed in a queue in sequence and are deleted one at a time, in what order will they be removed? (1)

1. ABCD
2. DCBA

3. BCDA
4. ADBC

Q6. Which of the following information is stored in doubly linked list (1)

1. Value of node
2. Address of next node
3. Address of previous node
4. All of the mentioned

Q7. Insertion of an element at the middle of a linked list requires the modification of how many pointers (1)

1. 1
2. 2
3. 3
4. 0

Q8. A linked list in which none of the nodes contains a NULL pointer is (1)

1. Singly linked list
2. Doubly linked list
3. Circular linked list
4. None of the mentioned

Q9. Consider the following definition in c programming language. Which of the following c code is used to create new node? (1)

```
struct node
{
    int data;
    struct node * next;
}
typedef struct node NODE;
NODE *ptr;
```

1. `ptr = (NODE*)malloc(sizeof(NODE))`
2. `ptr = (NODE*)malloc(NODE)`
3. `ptr = (NODE*)malloc(sizeof(NODE*))`

4. `ptr = (NODE)malloc(sizeof(NODE))`

Q10. Which of the following is the valid way to implement stack using a linked list (1)

1. Add elements to the front of the list and remove elements from the back of the list
2. Add elements to the front of the list and remove elements from the front of the list
3. Add elements to the back of the list and remove elements from the front of the list
4. Add and remove elements from any end of the list

Q11) . Assume `QUEUE_SIZE` is 5. The circular queue contains four items 10, 20, 30 and 40. Show the contents of the circular queue after performing each of the following operations. (i) insert 50 (ii) insert 60 (iii) delete (iv) delete (v) insert 70 (vi) insert 80 (1*5=5 marks)

Solution:

| | | | | |
|----|----|----|----|--|
| 10 | 20 | 30 | 40 | |
|----|----|----|----|--|

(i) insert 50

| | | | | |
|----|----|----|----|----|
| 10 | 20 | 30 | 40 | 50 |
|----|----|----|----|----|

(ii) insert 60

Can't insert. Queue overflow

(iii) delete

| | | | | |
|--|----|----|----|----|
| | 20 | 30 | 40 | 50 |
|--|----|----|----|----|

(iv) delete

| | | | | |
|--|--|----|----|----|
| | | 30 | 40 | 50 |
|--|--|----|----|----|

(v) insert 70

| | | | | |
|----|--|----|----|----|
| 70 | | 30 | 40 | 50 |
|----|--|----|----|----|

(vi) insert 80

| | | | | |
|----|----|----|----|----|
| 70 | 80 | 30 | 40 | 50 |
|----|----|----|----|----|

Q12). Convert the following infix expressions to postfix using stack and write algorithm also.

(i) $A*(B*C+D*E)+F$ (ii) $A/(B-C)*D+E$.

(2+2+1) marks

Solution:

(i) `ABC*DE*+*F+`

| | Current Token | Operator Stack | Postfix String |
|----|---------------|----------------|----------------|
| 1 | A | | A |
| 2 | * | * | A |
| 3 | (| *(| A |
| 4 | B | *(| AB |
| 5 | * | *(* | AB |
| 6 | C | *(* | ABC |
| 7 | + | *(+ | ABC* |
| 8 | D | *(+ | ABC*D |
| 9 | * | *(+* | ABC*D |
| 10 | E | *(+* | ABC*DE |
| 11 |) | * | ABC*DE*+ |
| 12 | + | + | ABC*DE*+* |
| 13 | F | + | ABC*DE*+*F |
| 14 | | | ABC*DE*+*F+ |

(ii) $AB C-/D^*E+$

| Input String | Output Stack | Operator Stack |
|------------------|--------------|----------------|
| $(A/(B-C)^*D+E)$ | | (|
| $(A/(B-C)^*D+E)$ | A | (|
| $(A/(B-C)^*D+E)$ | A | (/ |
| $(A/(B-C)^*D+E)$ | A | ((|
| $(A/(B-C)^*D+E)$ | AB | ((|
| $(A/(B-C)^*D+E)$ | AB | ((- |
| $(A/(B-C)^*D+E)$ | AB | ((- |
| $(A/(B-C)^*D+E)$ | AB C | ((- |
| $(A/(B-C)^*D+E)$ | AB C- | (/ |
| $(A/(B-C)^*D+E)$ | AB C-/ | (* |
| $(A/(B-C)^*D+E)$ | AB C-/D | (* |
| $(A/(B-C)^*D+E)$ | AB C-/D* | (+ |
| $(A/(B-C)^*D+E)$ | AB C-/D*E | (+ |
| $(A/(B-C)^*D+E)$ | AB C-/D*E+ | |

Algorithm Infix→postfix

Step 1: If the scanned character is an operand, put it into postfix expression.

Step 2: If the scanned character is an operator and operator's stack is empty, push operator into operators' stack.

Step 3: If the operator's stack is not empty, there may be following possibilities.

If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operator 's stack.

If the precedence of scanned operator is less than the top most operator of operator's stack, pop the operators from operator's stack until we find a low precedence operator than the scanned character.

If the precedence of scanned operator is equal then check the associativity of the operator. If associativity left to right then pop the operators from stack until we find a low precedence operator. If associativity right to left then simply put into stack.

If the scanned character is opening round bracket ('('), push it into operator's stack.

If the scanned character is closing round bracket (')'), pop out operators from operator's stack until we find an opening bracket ('(').

Repeat Step 1,2 and 3 till expression has character

Step 4: Now pop out all the remaining operators from the operator's stack and push into postfix expression.

Step 5: Exit

Q13). Write an algorithm/pseudocode to count the number of elements in a singly linked list.

(3 marks)

Solution:

1. ptr = start, count = 0
2. while ptr <> NULL do steps 3 and 4
3. count = count+1
4. ptr = ptr -> link
5. print "No of Nodes", count.

Q14). Why do we need doubly linked lists? Write functions to insert and delete elements in doubly linked list.

(3marks)

- In a doubly linked list, nodes are equipped with two pointers: one pointing to the previous node and the other to the next node. This dual-pointer structure facilitates seamless traversal in both forward and reverse directions, ensuring swift insertion and deletion of nodes. (1mark)
- **insert elements in doubly linked list.** (1mark)

```
void insertbeginning(struct Node** head_ref, int new_data)
{
```

```

// 1. allocate node
struct Node* new_node
    = (struct Node*)malloc(sizeof(struct Node));

// 2. put in the data
new_node->data = new_data;

// 3. Make next of new node as head and previous as NULL
new_node->next = (*head_ref);
new_node->prev = NULL;

// 4. change prev of head node to new node
if ((*head_ref) != NULL)
    (*head_ref)->prev = new_node;

// 5. move the head to point to the new node
(*head_ref) = new_node;
}

```

Note: insertion can be at the end/middle of the list also....

- **delete elements in doubly linked list.** (1 mark)

```

/* a node of the doubly linked list */
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

/* Function to delete a node in a Doubly Linked List.
head_ref --> pointer to head node pointer.
del --> pointer to node to be deleted. */
void deleteNode(struct Node** head_ref, struct Node* del)
{
    /* base case */
    if (*head_ref == NULL || del == NULL)
        return;

    /* If node to be deleted is head node */
    if (*head_ref == del)
        *head_ref = del->next;

    /* Change next only if node to be deleted is NOT the last node */
    if (del->next != NULL)
        del->next->prev = del->prev;
}

```

```

/* Change prev only if node to be deleted is NOT the first node */
if (del->prev != NULL)
    del->prev->next = del->next;

/* Finally, free the memory occupied by del*/
free(del);
return;
}

```

```

/* delete nodes from the doubly linked list */
deleteNode(&head, head); /*delete first node*/
deleteNode(&head, head->next); /*delete middle node*/
deleteNode(&head, head->next); /*delete last node*/

```

Q15). How polynomial can be represented with a linked list? Write a C program to add two given polynomials P1: $5x^4+3x^2+1x^0$ and P2: $4x^4+2x^2+1x^1$. (4 marks)

- Polynomial representation using linked list (1 mark)**

A node can be created for each term in the polynomial . Coefficient and exponent can be stored as data in the node and all the nodes corresponding to terms in polynomial can be linked by pointers as shown below:

- Representation**

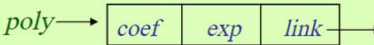
$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \dots + a_0x^{e_0}$$

- Type declaration**

```

typedef struct poly_node *poly_pointer;
typedef struct poly_node {
    int coef;
    int expon;
    poly_pointer link;
};
poly_pointer a, b, d;

```



- C program to add two given polynomials P1: $5x^4+3x^2+1x^0$ and P2: $4x^4+2x^2+1x^1$ (3 marks)**

```

struct Node {
    int coef;
    int exp;
    struct Node* next;
};

typedef struct Node Node;

```

```

Node* add(Node* poly1, Node* poly2) {
    Node* result = NULL;

    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->exp == poly2->exp) {
            insert(&result, poly1->coef + poly2->coef, poly1->exp);
            poly1 = poly1->next;
            poly2 = poly2->next;
        } else if (poly1->exp > poly2->exp) {
            insert(&result, poly1->coef, poly1->exp);
            poly1 = poly1->next;
        } else {
            insert(&result, poly2->coef, poly2->exp);
            poly2 = poly2->next;
        }
    }

    while (poly1 != NULL) {
        insert(&result, poly1->coef, poly1->exp);
        poly1 = poly1->next;
    }

    while (poly2 != NULL) {
        insert(&result, poly2->coef, poly2->exp);
        poly2 = poly2->next;
    }

    return result;
}

```



```
int main() {
    Node* poly1 = NULL;
    insert(&poly1, 5, 4);
    insert(&poly1, 3, 2);
    insert(&poly1, 1, 0);

    Node* poly2 = NULL;
    insert(&poly2, 4, 4);
    insert(&poly2, 2, 2);
    insert(&poly2, 1, 1);

    Node* result = add(poly1, poly2);
    printf("Result: ");
    print(result);

    return 0;
}
```