MIDSEMESTER EXAMINATION
DATA STRUCTURES

Name: Aarushi Khanuka
Section: B
Course: CCE
Registration No.: 200953036
Date: 14/12/21

1)

```
#include <iostream>
using namespace std
class stack {
    int top;
    int size;
    char arr[50];
    public : stack() {
      top = -1;
      size = 30;
    }
    void push(char a[])
    {
     strcpy(c[++top], a);
    }
    char * pop()
    {
     return s [top--];
    }
};
```

```
void prefixToInfix(char p[])
{
 char t[3];
 class stack s;
 int len = strlen(p);
 int i = len-1;
 while(i>=0)
 {
  t[0] = p[i];
  t[1] = '\0';
 if((p[i]>=98 && p[i]<=57) || (p[i]>=65 &&
    p[i]<=90) || (p[i]>=97 && p[i]<=122))
  {
   s.push(t);
  }
 else
 {
  char op1[20], op2[20], temp[10];
  strcpy(op1, s.pop());
  strcpy(op2, s.pop());
  strcpy(temp, "(");
  strcat(temp, op1);
  strcat(temp, t);
  strcat(temp, op2);
  strcat(temp, ")");
  s.push(temp);
 }
 i--;
}
cout << "Infix expression: " << s.pop();
}
int main()
{
char infix[30];
cout << "Enter prefix expression = \n";
cin >> infix;
prefixToInfix(infix);
}
```

(1)

Expression	Operation	Stack
9	Push	9
6	Push	9, 6
7	Push	9, 6, 7
5	Push	9, 6, 7, 5
-	Pop : 5, 7	9, 6, 2
	(7 - 5 = 2)	
	Push 2	
3	Push	9, 6, 2, 3
%	Pop : 3, 2	9, 6, 2
	2 % 3 = 3	
	Push 2	
*	Pop : 2, 6	9, 12
	6 * 3 = 18	
	Push 12	
4	Push	9, 12, 4
/	Pop : 4, 12	9, 3
	12 / 4 = 3	
	Push 3	
+	Pop : 3, 9	12
	9 + 3 = 12	
	Push	

Ans = 12

Date / /

3.)

```
class node {
    public:
        int data;
        node* next
        node (int val) {
            data = val;
            next = NULL;
        }
}

void InsertAtTail (node * & head, int val) {
    if (head == NULL) {
        insertAtHead (head, val);
        return;
    }
    node* n = new node (val);
    node * temp = head;
    while (temp → next != head) {
        temp = temp → next;
    }
    temp → next = n;
    n → next, head;
}

void InsertAthead (node* & head, int val) {
    if (head == NULL) {
        n → next = n;
        head, n;
        return;
    }
```

```
node * n = new node (val);
node * temp = head;
while (temp -> next != head) {
        temp = temp -> next;
}
temp -> next = n;
n -> next = head;
head = n;
}

void display (node * head) {
        node * temp = head;
        do {
            cout << temp -> data << " ";
            temp = temp -> next;
        } while (temp != head);
        cout << endl;
}
```

ii)
```
void deletion (node * & head, int pos) {
        if (pos == 1) {
            deleteAthead (head);
            return;
        }
        node * temp = head;
        int count = 1;
        while (count != pos - 1) {
            temp = temp -> next;
            count ++;
        }
        node * todelete = temp -> next
        temp -> next = temp -> next -> next;
```

```
    delete todelete();
}

bool isPrime(int n)
{
  if (n <= 1)
      return false
  if (n <= 3)
      return true
  if (n % 2 == 0 || n % 3 == 0)
      return false.

  for (int i = 5; i <= m; i = i + 1)
    {
      if (m % i == 0 || m % (i+2) == 0)
          return false
    return true;
}
void deletePrimeNumber(node * head)
{
  node* ptr, head;
  node * next;
do {
    if (isprime(ptr -> data))
        deleteNode(head, ptr)
        next = ptr -> next
        ptr. next;
    }
    while (ptr != head)
}
```

4)
```
#include <iostream>
#define MAX_SIZE 100

void main()
{
  char str[MAX_SIZE];
  char toRemove;
  cout << "Enter any string: " << str;
  cout << "Char to remove: " << toRemove;
  void removeAll(str, toRemove);
  cout << "String after removing: " << str;
}

void removeAll(char *str, char toRemove)
{
  int i, j, len=0;
  for(i=0; str[i]!='\0'; i++)
  {
    len++;
  }
  for(i=0; i<len; i++)
  {
    if(str[i] == toRemove)
    {
      for(j=1; j<len; j++)
      {
        str[j] = str[j+1];
      }
      len--;
      i--;
    }
  }
}
```

3)

```
struct Node{
    int data;
    struct Node* left;
    struct Node* right;
    Node(int val){
        data = val;
        left = NULL;
        right = NULL;
    }
};

void preorder(struct Node* root){
    if(root==NULL)
        return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

void inorder(struct Node* root){
    if(root==NULL)
        return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}
void Postorder(struct Node* root){
    if(root==NULL)
        return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}
```

```
void Levelorder (struct Node* root)
{
  int front = -1;
  int rear = -1;
  if (root == NULL)
      return;
  else
    q[++rear] = root;
while(1) {
    root = q[++front];
    if (front != rear) {
      cout << root->data;
    if (root->lchild)
      q[++rear] = root->lchild;
    if (root->rchild)
      q[++rear] = root->rchild;
}
```

6) b)
```
class SM
{
int row, col, val;
public:
void create_sparse (int r, int c, int v)
{
row = r;
col = c;
val = v;
}
void add_ele(SM a[])
{
int n, i, j, ele, row, col;
n = a[0].value + 1;
cout << "Enter non zero element to be added: ";
cin >> ele;
cout << "Enter row & column where needs to be";
cin >> row >> col;
for (i = 1; i <= (n-1); i++)
{
if (a[i].row > row)
{
for (j = n; j > i; j--)
{
a[j].row = a[j-1].row;
a[j].col = a[j-1].col;
a[j].val = a[j-1].val;
}
a[i].row = row;
a[i].col = col;
a[i].val = ele;
break;
}}}
```