7. Write a function which evaluates an infix expression, without converting it to postfix. The input

string can have spaces, (,) and precedence of operators should be handled.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define MAX 100

typedef struct {
    int data[MAX];
    int top;
} StackInt;

typedef struct {
    char data[MAX];
    int top;
} StackChar;

void pushInt(StackInt* s, int val) {
    s->data[++s->top] = val;
}

int popInt(StackInt* s) {
    return s->data[s->top--];
}
```

```
int peekInt(StackInt* s) {
    return s->data[s->top];
}

int isEmptyInt(StackInt* s) {
    return s->top == -1;
}

void pushChar(StackChar* s, char val) {
    s->data[++s->top] = val;
}

char popChar(StackChar* s) {
    return s->data[s->top--];
}

char peekChar(StackChar* s) {
    return s->data[s->top];
}

int isEmptyChar(StackChar* s) {
    return s->top == -1;
}

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}
```

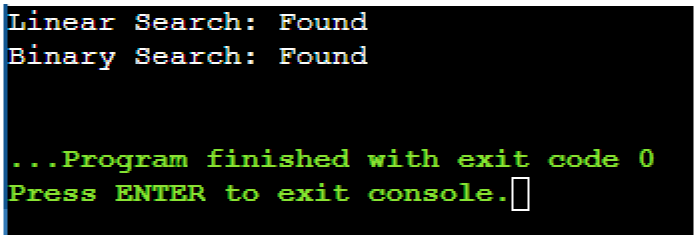
```
int applyOp(int a, int b, char op) {
    switch (op) {
        case '+': return b + a;
        case '-': return b - a;
        case '*': return b * a;
        case '/': return b / a;
    }
    return 0;
}

int evaluate(char* exp) {
    StackInt values = {.top = -1};
    StackChar ops = {.top = -1};

    for (int i = 0; i < strlen(exp); i++) {

        if (isspace(exp[i])) continue;

        else if (isdigit(exp[i])) {
            int val = 0;
            while (i < strlen(exp) && isdigit(exp[i])) {
                val = val * 10 + (exp[i] - '0');
                i++;
            }
            i--;
            pushInt(&values, val);
        }
        else if (exp[i] == '(') {
            pushChar(&ops, '(');
        }

        else if (exp[i] == ')') {
            while (!isEmptyChar(&ops) && peekChar(&ops) != '(') {
                int a = popInt(&values);
                int b = popInt(&values);
                char op = popChar(&ops);
                pushInt(&values, applyOp(a, b, op));
            }
            popChar(&ops);
        }

        else {
            while (!isEmptyChar(&ops) &&
                   precedence(peekChar(&ops)) >= precedence(exp[i])) {

                int a = popInt(&values);
                int b = popInt(&values);
                char op = popChar(&ops);

                pushInt(&values, applyOp(a, b, op));
            }
            pushChar(&ops, exp[i]);
        }
    }

    while (!isEmptyChar(&ops)) {
        int a = popInt(&values);
        int b = popInt(&values);
        char op = popChar(&ops);
        pushInt(&values, applyOp(a, b, op));
    }

    return popInt(&values);
}

int main() {
    char exp[] = "3 + 6 * (5 + 4) / 3 - 7";
    printf("Expression: %s\n", exp);
    printf("Result: %d\n", evaluate(exp));
    return 0;
}
```

DSA
Roll No:- SCOD09

1. Implement Linear search and Binary Search for a given array.
   Code:-

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
   for(int i = 0; i < n; i++)
      if(arr[i] == key) return i;
   return -1;
}

int binarySearch(int arr[], int n, int key) {
   int l = 0, r = n - 1;
   while(l <= r) {
      int m = l + (r - l)/2;
      if(arr[m] == key) return m;
      else if(arr[m] < key) l = m + 1;
      else r = m - 1;
   }
   return -1;
}
int main() {
   int arr[] = {2, 4, 6, 8, 10};
   int n = sizeof(arr)/sizeof(arr[0]);
   int key = 8;
   int lin = linearSearch(arr, n, key);
   int bin = binarySearch(arr, n, key);
   printf("Linear Search: %s\n", lin != -1 ? "Found" : "Not Found");
   printf("Binary Search: %s\n", bin != -1 ? "Found" : "Not Found");
   return 0;
}
```

   Output:-



```
Linear Search: Found
Binary Search: Found


...Program finished with exit code 0
Press ENTER to exit console.
```

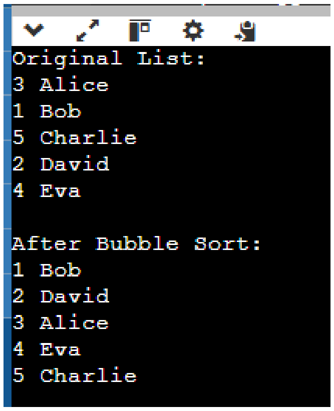
DSA
Roll No:- SCOD09

2. Arrange the list of students according to roll numbers in ascending order using a) Bubble Sort b) Insertion sort c) Quick sort

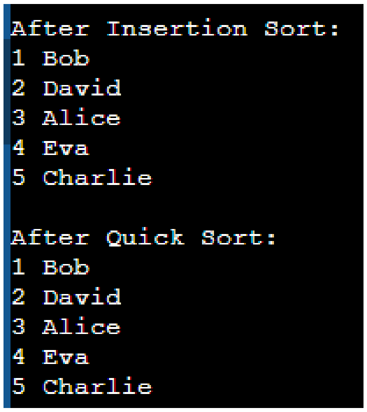
   Code:-

```
#include <stdio.h>
#include <string.h>
typedef struct {
   int roll;
   char name[50];
} Student;
void printStudents(Student s[], int n) {
   for(int i = 0; i < n; i++)
      printf("%d %s\n", s[i].roll, s[i].name);
}void bubbleSort(Student s[], int n) {
   for(int i = 0; i < n-1; i++)
      for(int j = 0; j < n-i-1; j++)
         if(s[j].roll > s[j+1].roll) {
            Student temp = s[j];s[j] = s[j+1];
            s[j+1] = temp;}}
void insertionSort(Student s[], int n) {
   for(int i = 1; i < n; i++) {
      Student key = s[i];
      int j = i - 1;
      while(j >= 0 && s[j].roll > key.roll) {
         s[j+1] = s[j];
         j--;}
      s[j+1] = key; }}
int partition(Student s[], int low, int high) {
   int pivot = s[high].roll;
   int i = low - 1;
   for(int j = low; j < high; j++) {
      if(s[j].roll < pivot) {
         i++;
         Student temp = s[i];
         s[i] = s[j];
         s[j] = temp; } }
   Student temp = s[i+1];
   s[i+1] = s[high];
   s[high] = temp;
   return i+1;}
void quickSort(Student s[], int low, int high)
{if(low < high) {
      int pi = partition(s, low, high);
      quickSort(s, low, pi-1);
      quickSort(s, pi+1, high); }}
int main() {
   Student s[] = {{3,"Alice"}, {1,"Bob"},
{5,"Charlie"}, {2,"David"}, {4,"Eva"}};
   int n = sizeof(s)/sizeof(s[0]);
   printf("Original List:\n");
   printStudents(s, n);
   bubbleSort(s, n);
   printf("\nAfter Bubble Sort:\n");
   printStudents(s, n);
   Student s2[] = {{3,"Alice"}, {1,"Bob"},
{5,"Charlie"}, {2,"David"}, {4,"Eva"}};
   insertionSort(s2, n);
   printf("\nAfter Insertion Sort:\n");
   printStudents(s2, n);
   Student s3[] = {{3,"Alice"}, {1,"Bob"},
{5,"Charlie"}, {2,"David"}, {4,"Eva"}};
   quickSort(s3, 0, n-1);
   printf("\nAfter Quick Sort:\n");
   printStudents(s3, n);

   return 0;}
```

   Output:-



```
Original List:
3 Alice
1 Bob
5 Charlie
2 David
4 Eva

After Bubble Sort:
1 Bob
2 David
3 Alice
4 Eva
5 Charlie
```

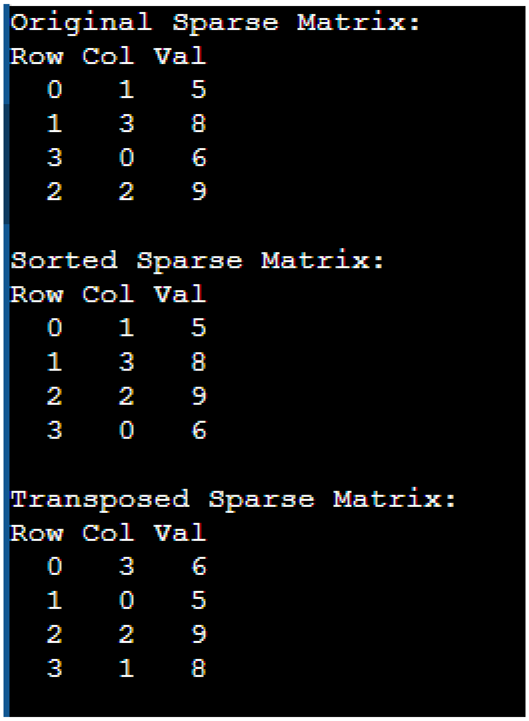


```
After Insertion Sort:
1 Bob
2 David
3 Alice
4 Eva
5 Charlie

After Quick Sort:
1 Bob
2 David
3 Alice
4 Eva
5 Charlie
```

3. Implement a sparse matrix with operations like initialize empty sparse matrix, insert an element, sort a sparse matrix on row-column, transpose a matrix, etc.
   Code:-

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int row;
    int col;
    int val;
} Element;
typedef struct {
    int rows;
    int cols;
    int size;       // number of non-zero elements
    Element *data;  // dynamic array of elements
} SparseMatrix;SparseMatrix initialize(int r, int c, int capacity) {
    SparseMatrix m;
    m.rows = r;
    m.cols = c;
    m.size = 0;
    m.data = (Element*)malloc(capacity * sizeof(Element));
    return m;
}void insert(SparseMatrix *m, int r, int c, int v) {
    if(v != 0) {
        m->data[m->size].row = r;
        m->data[m->size].col = c;
        m->data[m->size].val = v;
        m->size++;
    }
}void sortSparse(SparseMatrix *m) {
    for(int i = 0; i < m->size-1; i++)
        for(int j = 0; j < m->size-i-1; j++)
            if(m->data[j].row > m->data[j+1].row ||
              (m->data[j].row == m->data[j+1].row && m->data[j].col > m->data[j+1].col)) {
                Element temp = m->data[j];
                m->data[j] = m->data[j+1];
                m->data[j+1] = temp;
            }}SparseMatrix transpose(SparseMatrix *m) {
    SparseMatrix t = initialize(m->cols, m->rows, m->size);
    for(int i = 0; i < m->size; i++) {
        insert(&t, m->data[i].col, m->data[i].row, m->data[i].val); }
    sortSparse(&t);
    return t;}void display(SparseMatrix *m) {
    printf("Row Col Val\n");
    for(int i = 0; i < m->size; i++)
        printf("%3d %3d %3d\n", m->data[i].row, m->data[i].col, m->data[i].val);}
int main() {
    SparseMatrix mat = initialize(4, 5, 10);
    insert(&mat, 0, 1, 5);
    insert(&mat, 1, 3, 8);
    insert(&mat, 3, 0, 6);
    insert(&mat, 2, 2, 9);
    printf("Original Sparse Matrix:\n");
    display(&mat);
    sortSparse(&mat);
    printf("\nSorted Sparse Matrix:\n");
    display(&mat);
    SparseMatrix t = transpose(&mat);
    printf("\nTransposed Sparse Matrix:\n");
    display(&t);
    free(mat.data);
    free(t.data);
    return 0;}
```

Output:-



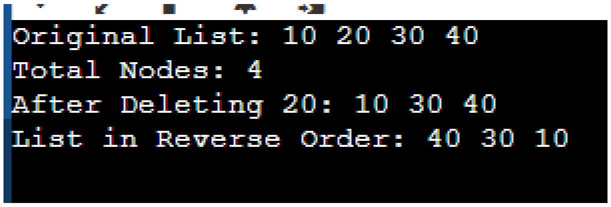
```
Original Sparse Matrix:
Row Col Val
  0   1   5
  1   3   8
  3   0   6
  2   2   9

Sorted Sparse Matrix:
Row Col Val
  0   1   5
  1   3   8
  2   2   9
  3   0   6

Transposed Sparse Matrix:
Row Col Val
  0   3   6
  1   0   5
  2   2   9
  3   1   8
```

DSA
Roll No:- SCOD09

4. Write functions to a) Add and delete the nodes in a linked list b) Compute total number of nodes in the linked list c) Display list in reverse order using recursion
   Code:-

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;
void addNode(Node** head, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    if(*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while(temp->next) temp = temp->next;
    temp->next = newNode;
}
void deleteNode(Node** head, int value) {
    if(*head == NULL) return;
    Node *temp = *head, *prev = NULL;
    if(temp->data == value) {
        *head = temp->next;
        free(temp);
        return;
    }
    while(temp && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
    if(temp) {
        prev->next = temp->next;
        free(temp);
    }
}
int countNodes(Node* head) {
    int count = 0;
    while(head) {
        count++;
        head = head->next;
    }
    return count;
}
void displayReverse(Node* head) {
    if(head == NULL) return;
    displayReverse(head->next);
    printf("%d ", head->data);
}
void displayList(Node* head) {
    while(head) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}
int main() {
    Node* head = NULL;
    addNode(&head, 10);
    addNode(&head, 20);
    addNode(&head, 30);
    addNode(&head, 40);
    printf("Original List: ");
    displayList(head);
    printf("Total Nodes: %d\n", countNodes(head));
    deleteNode(&head, 20);
    printf("After Deleting 20: ");
    displayList(head);
    printf("List in Reverse Order: ");
    displayReverse(head);
    printf("\n");
    return 0;
}
```

Output:



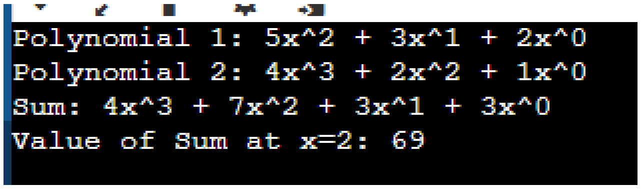
```
Original List: 10 20 30 40
Total Nodes: 4
After Deleting 20: 10 30 40
List in Reverse Order: 40 30 10
```

5. Implement a data type to represent a Polynomial with the operations like create an empty polynomial, insert an entry into polynomial, add two polynomials and return the result as a polynomial, evaluate a polynomial, etc.

   Code:-

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct Term {
    int coeff;
    int exp;
    struct Term* next;
} Term;

// Create an empty polynomial
Term* createPolynomial() {
    return NULL;
}

// Insert a term in descending order of exponent
void insertTerm(Term** poly, int coeff, int exp) {
    if(coeff == 0) return; // ignore zero coefficient
    Term* newTerm = (Term*)malloc(sizeof(Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;

    if(*poly == NULL || exp > (*poly)->exp) {
        newTerm->next = *poly;
        *poly = newTerm;
        return;
    }

    Term* temp = *poly;
    Term* prev = NULL;
    while(temp && temp->exp > exp) {
        prev = temp;
        temp = temp->next;
    }
    if(temp && temp->exp == exp) { // combine like terms
        temp->coeff += coeff;
        free(newTerm);
        if(temp->coeff == 0) { // remove zero coefficient term
            if(prev) prev->next = temp->next;
            else *poly = temp->next;
            free(temp);
        }
    } else {
        newTerm->next = temp;
        if(prev) prev->next = newTerm;
    }
}

// Display polynomial
void displayPolynomial(Term* poly) {
    if(!poly) { printf("0\n"); return; }
    while(poly) {
        printf("%dx^%d", poly->coeff, poly->exp);
        if(poly->next) printf(" + ");
        poly = poly->next;
    }
    printf("\n");
}

// Add two polynomials
Term* addPolynomials(Term* p1, Term* p2) {
    Term* result = createPolynomial();
    Term* t1 = p1;
    Term* t2 = p2;
    while(t1) {
        insertTerm(&result, t1->coeff, t1->exp);
        t1 = t1->next;
    }
    while(t2) {
        insertTerm(&result, t2->coeff, t2->exp);
        t2 = t2->next;
    }
    return result;
}

// Evaluate polynomial at x
int evaluatePolynomial(Term* poly, int x) {
    int sum = 0;
    while(poly) {
        sum += poly->coeff * pow(x, poly->exp);
        poly = poly->next;
    }
    return sum;
}

int main() {
    Term* p1 = createPolynomial();
    Term* p2 = createPolynomial();

    insertTerm(&p1, 5, 2);
    insertTerm(&p1, 3, 1);
    insertTerm(&p1, 2, 0);

    insertTerm(&p2, 4, 3);
    insertTerm(&p2, 2, 2);
    insertTerm(&p2, 1, 0);

    printf("Polynomial 1: ");
    displayPolynomial(p1);
    printf("Polynomial 2: ");
    displayPolynomial(p2);

    Term* sum = addPolynomials(p1, p2);
    printf("Sum: ");
    displayPolynomial(sum);

    int x = 2;
    printf("Value of Sum at x=%d: %d\n", x, evaluatePolynomial(sum, x));

    return 0;
}
```

Output:-



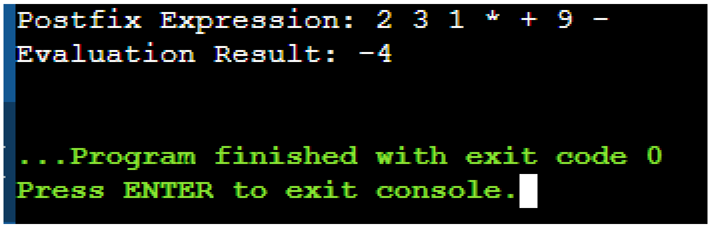
```
Polynomial 1: 5x^2 + 3x^1 + 2x^0
Polynomial 2: 4x^3 + 2x^2 + 1x^0
Sum: 4x^3 + 7x^2 + 3x^1 + 3x^0
Value of Sum at x=2: 69
```

6. Implement Stack using a linked list. Use this stack to perform evaluation of a postfix expression

   Code:-

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;void push(Node** top, int val) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = val;
    newNode->next = *top;
    *top = newNode;}
int pop(Node** top) {
    if(*top == NULL) { printf("Stack underflow\n"); return 0; }
    Node* temp = *top;
    int val = temp->data;
    *top = temp->next; free(temp); return val;}
int evaluatePostfix(char* exp) {
    Node* stack = NULL;
    for(int i=0; exp[i]; i++) {
        if(isspace(exp[i])) continue;else if(isdigit(exp[i])) {int num = 0;
            while(isdigit(exp[i])) { num = num*10 + (exp[i]-'0'); i++; }  i--; // adjust for outer loop
            push(&stack, num);
        } else {
            int b = pop(&stack);
            int a = pop(&stack);
            switch(exp[i]) {
                case '+': push(&stack, a+b); break; case '-': push(&stack, a-b); break;case '*': push(&stack,
a*b); break;case '/': push(&stack, a/b); break;}}}  return pop(&stack);}
int main() {
    char exp[] = "2 3 1 * + 9 -"; // Example postfix expression
    printf("Postfix Expression: %s\n", exp);
    printf("Evaluation Result: %d\n", evaluatePostfix(exp));
    return 0;
}
```

Output:-



```
Postfix Expression: 2 3 1 * + 9 -
Evaluation Result: -4


...Program finished with exit code 0
Press ENTER to exit console.
```

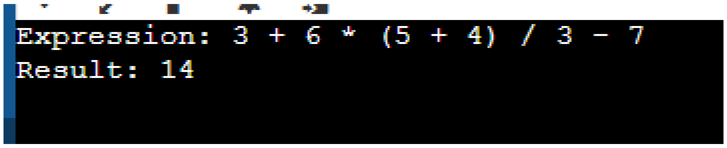
7. Write a function which evaluates an infix expression, without converting it to postfix. The input string can have spaces, (,) and precedence of operators should be handled.

Code:-

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct { int data[MAX]; int top;
} StackInt;
typedef struct {
    char data[MAX];int top;
} StackChar;
void pushInt(StackInt* s, int val) { s-
>data[++s->top] = val; }
int popInt(StackInt* s) { return s->data[s-
>top--]; }
int peekInt(StackInt* s) { return s->data[s-
>top]; }
int isEmptyInt(StackInt* s) { return s->top
== -1; }void pushChar(StackChar* s, char
val) { s->data[++s->top] = val; }
char popChar(StackChar* s) { return s-
>data[s->top--]; }
char peekChar(StackChar* s) { return s-
>data[s->top]; }
int isEmptyChar(StackChar* s) { return s-
>top == -1; }
int precedence(char op) {
    if(op=='+'||op=='-') return 1;
    if(op=='*'||op=='/') return 2;
    return 0;}
int applyOp(int a,int b,char op){
    switch(op){
        case '+': return a+b;case '-': return a-b;
        case '*': return a*b;
        case '/': return a/b; }
    return 0;}
```

```
int evaluate(char* exp){
    StackInt values = {.top=-1};
    StackChar ops = {.top=-1};
    for(int i=0;i<strlen(exp);i++){
        if(isspace(exp[i])) continue;
        else if(isdigit(exp[i])){
            int val=0;
            while(i<strlen(exp) &&
isdigit(exp[i])){ val=val*10 + (exp[i]-'0');
i++;}i--;pushInt(&values,val);}
        else if(exp[i]=='(') pushChar(&ops,'(');
        else if(exp[i]==')'){
            while(!isEmptyChar(&ops) &&
peekChar(&ops)!='(')pushInt(&values,
applyOp(popInt(&values), popInt(&values),
popChar(&ops)));
            popChar(&ops); // remove '('  }
        else{
            while(!isEmptyChar(&ops) &&
precedence(peekChar(&ops)) >=
precedence(exp[i]))
                pushInt(&values,
applyOp(popInt(&values), popInt(&values),
popChar(&ops)));pushChar(&ops, exp[i]);}}
    while(!isEmptyChar(&ops))
        pushInt(&values,
applyOp(popInt(&values), popInt(&values),
popChar(&ops)));
    return popInt(&values);}
int main(){
    char exp[] = "3 + 6 * (5 + 4) / 3 - 7";
    printf("Expression: %s\n", exp);
    printf("Result: %d\n", evaluate(exp));
    return 0;
}
```

Output:-



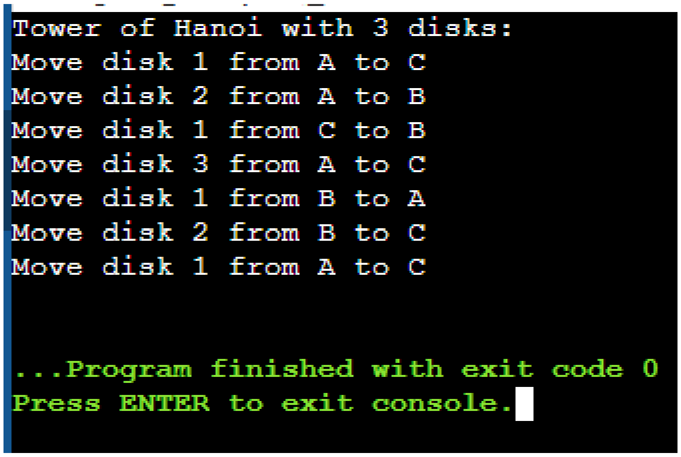
```
Expression: 3 + 6 * (5 + 4) / 3 - 7
Result: 14
```

8. Implement Tower of Hanoi using Recursion

Code:-

```
#include <stdio.h>
void towerOfHanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", from, to);
        return;
    }
    towerOfHanoi(n - 1, from, aux, to);
    printf("Move disk %d from %c to %c\n", n, from, to);
    towerOfHanoi(n - 1, aux, to, from);
}

int main() {
    int n = 3; // number of disks
    printf("Tower of Hanoi with %d disks:\n", n);
    towerOfHanoi(n, 'A', 'C', 'B'); // A = source, C = destination, B = auxiliary
    return 0;
}
```

Output:-



```
Tower of Hanoi with 3 disks:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C


...Program finished with exit code 0
Press ENTER to exit console.
```

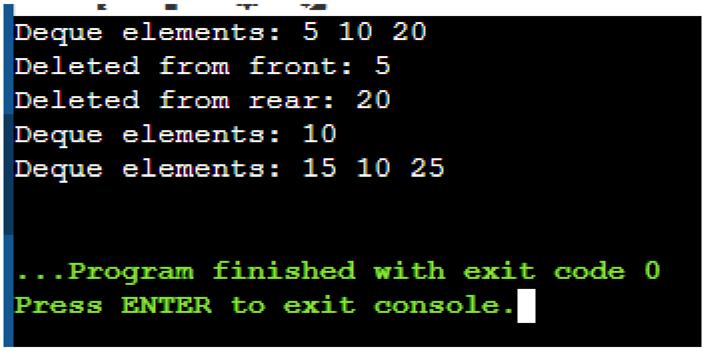
9. Write a program to simulate deque with functions to add and delete elements from either end of the deque

   Code:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100
typedef struct {
    int data[MAX];
    int front, rear;
} Deque;
void initDeque(Deque* dq) {
    dq->front = -1;
    dq->rear = -1;
}
int isEmpty(Deque* dq) {
    return dq->front == -1;
}
int isFull(Deque* dq) {
    return (dq->front == 0 && dq->rear == MAX-1) || (dq->front == dq->rear + 1);
}
void addFront(Deque* dq, int val) {
    if(isFull(dq)) { printf("Deque Overflow\n"); return; }
    if(isEmpty(dq)) dq->front = dq->rear = 0;
    else if(dq->front == 0) dq->front = MAX-1;
    else dq->front--;
    dq->data[dq->front] = val;
}
void addRear(Deque* dq, int val) {
    if(isFull(dq)) { printf("Deque Overflow\n"); return; }
    if(isEmpty(dq)) dq->front = dq->rear = 0;
    else if(dq->rear == MAX-1) dq->rear = 0;
    else dq->rear++;
    dq->data[dq->rear] = val;
}
int deleteFront(Deque* dq) {
    if(isEmpty(dq)) { printf("Deque Underflow\n"); return -1; }
    int val = dq->data[dq->front];
    if(dq->front == dq->rear) dq->front = dq->rear = -1;
    else if(dq->front == MAX-1) dq->front = 0;
    else dq->front++;
```

```
   return val;
}
int deleteRear(Deque* dq) {
   if(isEmpty(dq)) { printf("Deque Underflow\n"); return -1; }
   int val = dq->data[dq->rear];
   if(dq->front == dq->rear) dq->front = dq->rear = -1;
   else if(dq->rear == 0) dq->rear = MAX-1;
   else dq->rear--;
   return val;}
void display(Deque* dq) {
   if(isEmpty(dq)) { printf("Deque is empty\n"); return; }
   printf("Deque elements: ");
   int i = dq->front;
   while(1) {
      printf("%d ", dq->data[i]);
      if(i == dq->rear) break;
      i = (i + 1) % MAX; }
   printf("\n");}
int main() {
   Deque dq;
   initDeque(&dq);
   addRear(&dq, 10);
   addRear(&dq, 20); addFront(&dq, 5);display(&dq);
   printf("Deleted from front: %d\n", deleteFront(&dq));
   printf("Deleted from rear: %d\n", deleteRear(&dq));
   display(&dq);
   addFront(&dq, 15);
   addRear(&dq, 25);
   display(&dq);
   return 0;
}
```

Output:-



```
Deque elements: 5 10 20
Deleted from front: 5
Deleted from rear: 20
Deque elements: 10
Deque elements: 15 10 25


...Program finished with exit code 0
Press ENTER to exit console.
```

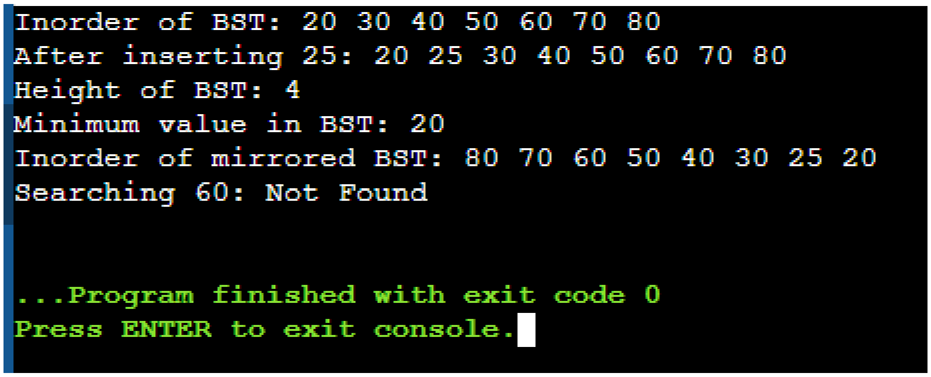
DSA
Roll No:- SCOD09

10. Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given. After constructing a binary tree i. Insert new node ii. Find number of nodes in longest path iii. Minimum data value found in the tree iv. Change a tree so that the roles of the left and right pointers are swapped at every node v. Search a value

    Code:-

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
Node* newNode(int data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
Node* insert(Node* root, int data) {
    if(root == NULL) return newNode(data);
    if(data < root->data) root->left = insert(root->left, data);
    else root->right = insert(root->right, data);
    return root;
}
int height(Node* root) {
    if(root == NULL) return 0;
    int l = height(root->left);
    int r = height(root->right);
    return (l > r ? l : r) + 1;
}
int findMin(Node* root) {
    Node* current = root;
    while(current && current->left != NULL) current = current->left;
    return current ? current->data : -1;
}
void mirror(Node* root) {
    if(root == NULL) return;
    Node* temp = root->left;
    root->left = root->right;
    root->right = temp;
    mirror(root->left);
    mirror(root->right);}
int search(Node* root, int key) {
    if(root == NULL) return 0;
    if(root->data == key) return 1;
    if(key < root->data) return search(root->left, key);
    else return search(root->right, key);}
void inorder(Node* root) {
    if(root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);}
int main() {
    Node* root = NULL;
    int values[] = {50, 30, 20, 40, 70, 60, 80};
    int n = sizeof(values)/sizeof(values[0]);
    for(int i=0;i<n;i++) root = insert(root, values[i]);
    printf("Inorder of BST: ");
    inorder(root);
    printf("\n");
    root = insert(root, 25);
    printf("After inserting 25: ");inorder(root);printf("\n");
    printf("Height of BST: %d\n", height(root));
    printf("Minimum value in BST: %d\n", findMin(root));
    mirror(root);
    printf("Inorder of mirrored BST: ");
    inorder(root);
    printf("\n"); int key = 60;
    printf("Searching %d: %s\n", key, search(root, key) ? "Found" : "Not Found");
    return 0;
}
```

Output:-



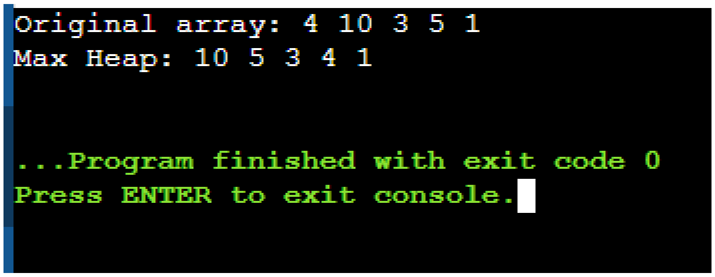
```
Inorder of BST: 20 30 40 50 60 70 80
After inserting 25: 20 25 30 40 50 60 70 80
Height of BST: 4
Minimum value in BST: 20
Inorder of mirrored BST: 80 70 60 50 40 30 25 20
Searching 60: Not Found


...Program finished with exit code 0
Press ENTER to exit console.
```

11. Develop C program to build a MAX_HEAP with given input data set

Code:-

```
#include <stdio.h>
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;
    if(left < n && arr[left] > arr[largest]) largest = left;
    if(right < n && arr[right] > arr[largest]) largest = right;
    if(largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);}}
void buildMaxHeap(int arr[], int n) {
    for(int i = n/2 - 1; i >= 0; i--) {
        heapify(arr, n, i);}}
void printHeap(int arr[], int n) {
    for(int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int arr[] = {4, 10, 3, 5, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Original array: ");
    printHeap(arr, n);
    buildMaxHeap(arr, n);
    printf("Max Heap: ");
    printHeap(arr, n);
    return 0;
}
```

Output:-



```
Original array: 4 10 3 5 1
Max Heap: 10 5 3 4 1


...Program finished with exit code 0
Press ENTER to exit console.
```