PROGRAM NO: 1 DATE: 10-08-2023

**AIM**

To find minimum and maximum number present in an array.

**ALGORITHM**

1. START
2. Initialize min and max variable with first element of the array.
3. Start traversing the array from 1st element to last element (nth)
4. Compare each value with min and max variables.
5. If the value is higher than max, replace the max value.
6. If the value is lower than min, replace the min value.
7. Outside the for loop, print the min and max values.
8. STOP

**CODE**

// Find maximum and minimum element in an array.

#include <iostream>

using namespace std;

void findMinMax(int arr[], int size) {

int i, min = arr[0], max = arr[0];

for (i=1; i<size; i++) {

if (arr[i] < min) {

min = arr[i];

}

if (arr[i] > max) {

max = arr[i];

}

};

cout<< "Minimum value is : " << min <<endl;

cout<< "Maximum value is : " << max <<endl;

cout<<endl;

}

int main() {

int size, i;

cout<< "Enter size of array: ";

cin>> size;

int arr[size];

for (i=0; i<size; i++) {

cout<< "Enter element " <<i + 1 <<" : ";

cin>>arr[i];

}

cout<<endl;

findMinMax(arr, size);

cout<< "--------- Author ----------------" <<endl;

cout<< "Ali Izzath Shazin" <<endl;

cout<< "220071601028" <<endl;

cout<< "B. Tech CSE A" <<endl;

return 0;

}

**OUTPUT**

Enter size of array: 5

Enter element 1 : 5

Enter element 2 : -2

Enter element 3 : -3

Enter element 4 : 2

Enter element 5 : 6

Minimum value is : -3

Maximum value is : 6

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 2 DATE: 10-08-2023

**AIM**

To find consecutive letters if they appear in the given character array.

**ALGORITHM**

1. START
2. Initialize variables i, flag (int) and text (string)
3. Traverse through the char array starting from 1st index to last.
4. Inside for loop, check if ASCII value of current character and last character are consecutive.
5. If yes, and flag is 0, print the last and current character, and set flag to 1.
6. If flag is 1, print only the current character.
7. If ASCII value of current character and last character are not consecutive, and if flag is 1.
8. Print an endline and set flag to 0.
9. STOP

**CODE**

*// Find consecutive letters if they appear in the given character array.*

#include <iostream>

using namespace std;

int main() {

    int i, flag=0;

    string text;

    cout<< "Enter the text to check : ";

    cin>> text;

    int length = text.length();

    for (i=1; i<length; i++) {

        if (text[i] - 1 == text[i-1] || text[i] - text[i-1] - 1 == 32 || ftext[i-1] - text[i] + 1 == 32) {

            if (flag == 0) {

                cout<<text[i - 1] << text[i];

                flag = 1;

            } else {

                cout<< text[i];

            }

        } else {

            if (flag == 1) {

                cout<< endl;

                flag = 0;

            }

        }

    }

    cout<<endl;

    cout<< "--------- Author ----------------" <<endl;

    cout<< "Ali Izzath Shazin" <<endl;

    cout<< "220071601028" <<endl;

    cout<< "B. Tech CSE A" <<endl;

    return 0;

}

**OUTPUT**

Enter the text to check :AbcDhlmNquvWz

AbcD

lmN

uvW

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 3 DATE: 17-08-2023

**AIM**

To implement insert,delete and traverse operations on linear array.

**ALGORITHM**

1. START
2. For Displaying, initialize a for loop from i=0 to i=length of array (exclusive).
3. Print every element inside the array.
4. For Inserting, initialize a for loop from i=last index to i=index to insert to.
5. Inside loop, replace (i + 1)th element with ith element.
6. Outside loop, insert new element to the given index.
7. Return the incremented length pointer.
8. For Deleting, initialize a for loop from i=index given to i=length of array - 1 (exclusive)
9. Inside loop, replace ith element with (i + 1)th element.
10. Return the decremented length pointer.
11. STOP

**CODE**

#include <iostream>

using namespace std;

void Display(int arr[], int &n, int &size) {

    cout<<endl;

    cout<< "[";

    for (int i=0; i<n; i++) {

        if (i == n - 1) cout<<arr[i];

        else cout<<arr[i] << ", ";

    }

    cout<< "] Size = " << size <<endl<<endl;

}

void Insert(int arr[], int &n, int size, int index, int item) {

    if (n == size) {

        cout<< "Array overflow!" <<endl<<endl;

        return;

    }

    if (index > n || index < 0) {

        cout<< "Array out of bound" <<endl<<endl;

        return;

    }

    for (int i=n-1; i>= index; i--) {

        arr[i + 1] = arr[i];

    }

    arr[index] = item;

    n++;

}

void Delete(int arr[], int &n, int index) {

    if (index > n - 1 || index < 0) {

        cout<<endl;

        cout<< "Array out of bound" <<endl;

        return;

    }

    for (int i=index; i<n-1; i++) {

        arr[i] = arr[i + 1];

    }

    n--;

}

int main() {

    int size, n, i, choice, temp1, temp2;

    cout<< "Enter size of array: ";

    cin>> size;

    int \*arr = new int[size];

    while (true) {

        cout<< "Enter number of elements to insert: ";

        cin>> n;

        if (n > size) {

            cout<< "Number of elements to insert should be lesser than or

sequal to the given size!" <<endl;

            continue;

        }

        break;

    }

    for (i=0; i<n; i++) {

        cout<< "Enter (" <<i + 1 <<")th element: ";

        cin>>arr[i];

    }

    while (true) {

        cout<< "----------- Array Operation -----------" <<endl;

        cout<< "1. Display Array." <<endl;

        cout<< "2. Insert Array." <<endl;

        cout<< "3. Delete Array." <<endl;

        cout<< "4. Exit." <<endl;

        cout<< "Enter choice: ";

        cin>> choice;

        if (choice == 1) {

            Display(arr, n, size);

        } else if (choice == 2) {

            cout<<endl<< "Enter element to insert: ";

            cin>> temp1;

            cout<< "Enter index to insert to: ";

            cin>> temp2;

            cout<< endl;

            Insert(arr, n, size, temp2, temp1);

        } else if (choice == 3) {

            cout<<endl<< "Enter index to delete: ";

            cin>> temp1;

            Delete(arr, n, temp1);

            cout<< endl;

        } else if (choice == 4) {

            cout<<endl<< "Thank you. Exiting.." <<endl;

cout<< "--------- Author ----------------" <<endl;

    cout<< "Ali Izzath Shazin" <<endl;

    cout<< "220071601028" <<endl;

    cout<< "B. Tech CSE A" <<endl;

            return 0;

        } else {

            cout<<endl<< "Invalid choice" <<endl<< endl;

            continue;

        }

    }

    return 0;}

**OUTPUT**

Enter size of array: 5

Enter number of elements to insert: 3

Enter (1)th element: 1

Enter (2)th element: 2

Enter (3)th element: 3

----------- Array Operation -----------

1. Display Array.

2. Insert Array.

3. Delete Array.

4. Exit.

Enter choice: 1

[1, 2, 3] Size = 5

----------- Array Operation -----------

1. Display Array.

2. Insert Array.

3. Delete Array.

4. Exit.

Enter choice: 2

Enter element to insert: 10

Enter index to insert to: 0

----------- Array Operation -----------

1. Display Array.

2. Insert Array.

3. Delete Array.

4. Exit.

Enter choice: 1

[10, 1, 2, 3] Size = 5

----------- Array Operation -----------

1. Display Array.

2. Insert Array.

3. Delete Array.

4. Exit.

Enter choice: 3

Enter index to delete: 1

----------- Array Operation -----------

1. Display Array.

2. Insert Array.

3. Delete Array.

4. Exit.

Enter choice: 1

[10, 2, 3] Size = 5

----------- Array Operation -----------

1. Display Array.

2. Insert Array.

3. Delete Array.

4. Exit.

Enter choice: 4

Thank you. Exiting..

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 4 DATE: 24-08-2023

**AIM**

To implement addition, subtraction and multiplication of matrix.

**ALGORITHM**

1. START
2. For storing matrix as pointer to pointer variable.
3. Firstly, initialize a pointer to pointer, and assign it with a pointer array of row size.
4. Loop through the pointer array and create array of column size in each index.
5. For addition and subtraction.
6. Declare two for loops. One traversing from i=0 to i=row size. Second one traversing from j=0 to j = col size.
7. Create a third zero matrix and assign the sum to each indices.
8. Finally, return the third matrix.
9. For multiplication of matrix.
10. Declare three for loops, One traversing from i=0 to i=row size of mat1. Second from j=0 to j=col size of mat2. Third one traversing from k=0 to k=col size of mat1 or row size of mat2
11. Then inside all three loops, mat3[i][j] += mat1[i][k] \* mat2[k][j];
12. STOP

**CODE**

#include <iostream>

using namespace std;

class Exception {

    public:

    virtual string msg() = 0;

};

class InvalidSize: public Exception {

    public:

    string msg() {

        return "Size should be greater than zero.";

    }

};

class OperationFailed: public Exception {

    public:

    string msg() {

        return "Both matrix should be of same size.";

    }

};

class MultiplicationFailed: public Exception {

    public:

    string msg() {

        return "Number of columns in the 1st matrix must be equal to the rows in the 2nd matrix.";

    }

};

class Matrix {

    int \_rows, \_cols;

    int \*\*mat;

    Matrix doAddOrSub(Matrix obj, int instance) {

        if (this->\_rows != obj.\_rows || this->\_cols != obj.\_cols) throw OperationFailed();

        Matrix returnMat = Matrix(this->\_rows, this->\_cols);

        returnMat.initializeMatrix();

        for (int i=0; i<this->\_rows; i++) {

            for (int j=0; j<this->\_cols; j++) {

                if (instance == 0) {

                    returnMat.mat[i][j] = this->mat[i][j] + obj.mat[i][j];

                } else if (instance == 1) {

                    returnMat.mat[i][j] = this->mat[i][j] - obj.mat[i][j];

                }

            }

        }

        return returnMat;

    }

    void initializeMatrix() {

        this->mat = new int\*[this->\_rows];

        for(int i = 0; i < this->\_rows; i++) {

            this->mat[i] = new int[this->\_cols];

        }

    }

    public:

        Matrix(int rows, int cols) {

            if (rows <= 0 || cols <= 0) throw InvalidSize();

            this->\_rows = rows;

            this->\_cols = cols;

            this->initializeMatrix();

        }

        Matrix operator+(Matrix obj) {

            return doAddOrSub(obj, 0);

        }

        Matrix operator-(Matrix obj) {

            return doAddOrSub(obj, 1);

        }

        Matrix operator\*(Matrix obj) {

            if (this->\_cols != obj.\_rows) throw MultiplicationFailed();

            Matrix returnMat = Matrix(this->\_rows, obj.\_cols);

            returnMat.initializeMatrix();

            for(int i=0; i<this->\_rows; i++) {

                for(int j=0; j<obj.\_cols; j++) {

                    returnMat.mat[i][j]=0;

                    for(int k=0; k<this->\_cols; k++) {

                        returnMat.mat[i][j] += this->mat[i][k] \* obj.mat[k][j];

                    }

                }

            }

            return returnMat;

        }

        void askElements() {

            for (int i=0; i<this->\_rows; i++) {

                for (int j=0; j<this->\_cols; j++) {

                    cout << "Enter element (" << i+1 << ", " << j+1 << "): ";

                    cin >> this->mat[i][j];

                }

            }

        }

        void display() {

            for (int i=0; i<this->\_rows; i++) {

                for (int j=0; j<this->\_cols; j++) {

                    cout << this->mat[i][j] << "\t";

                }

                cout << endl;

            }

            cout << "(" << this->\_rows << ", " << this->\_cols << ")" << endl;

        }

        int rows() {

            return this->\_rows;

        }

        int cols() {

            return this->\_cols;

        }

};

ostream& operator<<(ostream &out, Matrix obj) {

    obj.display();

    return out;

}

int main() {

    int mat1\_r, mat1\_c, mat2\_r, mat2\_c;

    cout << "Enter matrix 1 rows and columns: ";

    cin >> mat1\_r >> mat1\_c;

    Matrix mat1 = Matrix(mat1\_r, mat1\_c);

    cout << "Enter matrix 1 elements: " << endl;

    mat1.askElements();

    cout << mat1 << endl;

    cout << "Enter matrix 2 rows and columns: ";

    cin >> mat2\_r >> mat2\_c;

    Matrix mat2 = Matrix(mat2\_r, mat2\_c);

    cout << "Enter matrix 2 elements: " << endl;

    mat2.askElements();

    cout << mat2 << endl;

    try {

        cout << "Matrix 1 + matrix 2" << endl;

        Matrix mat3 = mat1 + mat2;

        cout << mat3 << endl;

    } catch(Exception &e) {

        cout << "Addition Failed: " << e.msg() << endl;

    }

    try {

        cout << "Matrix 1 - matrix 2" << endl;

        cout << mat1 - mat2 << endl;

    } catch(Exception &e) {

        cout << "Subtraction Failed: " << e.msg() << endl;

    }

    try {

        cout << "Matrix 1 \* matrix 2" << endl;

        cout << mat1 \* mat2 << endl;

    } catch(Exception &e) {

        cout << "Multiplication Failed: " << e.msg() << endl;

    }

cout<< "--------- Author ----------------" <<endl;

    cout<< "Ali Izzath Shazin" <<endl;

    cout<< "220071601028" <<endl;

    cout<< "B. Tech CSE A" <<endl;

    return 0;

}

**OUTPUT**

Enter matrix 1 rows and columns: 2 2

Enter matrix 1 elements:

Enter element (1, 1): 1

Enter element (1, 2): 2

Enter element (2, 1): 3

Enter element (2, 2): 4

1 2

3 4

(2, 2)

Enter matrix 2 rows and columns: 2 2

Enter matrix 2 elements:

Enter element (1, 1): 1

Enter element (1, 2): 2

Enter element (2, 1): 3

Enter element (2, 2): 4

1 2

3 4

(2, 2)

Matrix 1 + matrix 2

2 4

6 8

(2, 2)

Matrix 1 - matrix 2

0 0

0 0

(2, 2)

Matrix 1 \* matrix 2

7 10

15 22

(2, 2)

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 5 DATE: 31-08-2023

**AIM**

To implement stack data structure with Push, Pop and Peek.

**ALGORITHM**

1. START
2. Declare function isEmpty()
3. If top == -1: Return true
4. Else: return false
5. End if. End Func
6. Declare function isFull()
7. If top == size - 1: Return true
8. Else: return false
9. End if. End Func
10. Declare push(item)
11. If isFull(): throw Overflow Error
12. Increment top
13. SET arr[top] = item
14. End Func
15. Declare pop()
16. If isEmpty(): throw Underflow Error
17. decrement top
18. Return arr[top + 1]
19. End Func
20. Declare peek()
21. If isEmpty(): throw StackEmpty Error
22. Return arr[top ]
23. End Func
24. STOP

**CODE**

#include <iostream>

using namespace std;

class Exception {

    public:

    virtual string msg() = 0;

};

class StackOverflow: public Exception {

    public:

    string msg() {

        return "Stack overflow!.";

    }

};

class StackUnderflow: public Exception {

    public:

    string msg() {

        return "Stack underflow!";

    }

};

class StackEmpty: public Exception {

    public:

    string msg() {

        return "Stack is empty.";

    }

};

class Stack {

    int \_size;

    int top;

    int \*array;

    public:

        Stack(int \_size) {

            this->\_size = \_size;

            this->top = -1;

            this->array = new int[\_size];

        }

        int isEmpty() {

            if (this->top == -1) return 1;

            return 0;

        }

        int isFull() {

            if (this->top == this->\_size - 1) return 1;

            return 0;

        }

        int size() {

            return this->\_size;

        }

        void push(int item) {

            if (this->isFull()) {

                throw StackOverflow();

            }

            this->top++;

            this->array[this->top] = item;

        }

        int pop() {

            if (this->isEmpty()) {

                throw StackUnderflow();

            }

            this->top--;

            return this->array[this->top + 1];

        }

        void display() {

            if (this->isEmpty()) {

                throw StackEmpty();

            }

            cout << endl;

            for (int i=this->top; i>-1; i--) {

                if (i == this->top) {

                    cout << this->array[i] << " <-- Top" << endl;

                } else {

                    cout << this->array[i] << endl;

                }

            }

            cout << "Size = " << this->\_size << endl << endl;

        }

        int peek() {

            if (this->isEmpty()) {

                throw StackEmpty();

            }

            return this->array[this->top];

        }

};

ostream& operator<<(ostream &out, Stack &obj) {

    obj.display();

    return out;

}

int main() {

    int size, choice, temp;

    cout << "Enter size of the stack: ";

    cin >> size;

    Stack stack = Stack(size);

    while (true) {

        cout << "---------- Stack Operations ----------" << endl;

        cout << "1. Push." << endl;

        cout << "2. Pop." << endl;

        cout << "3. Peek." << endl;

        cout << "4. Display." << endl;

        cout << "5. Exit." << endl;

        cout << "Enter choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter element to push: ";

            cin >> temp;

            try {

                stack.push(temp);

            } catch(Exception &e) {

                cout << endl << e.msg() << endl << endl;

            }

        } else if (choice == 2) {

            try {

                int temp = stack.pop();

                cout << endl << "Popped element: " << temp << endl << endl;

            } catch(Exception &e) {

                cout << endl << e.msg() << endl << endl;

            }

        } else if (choice == 3) {

            try {

                int temp = stack.peek();

                cout << endl << "Top element: " << temp << endl << endl;

            } catch(Exception &e) {

                cout << endl << e.msg() << endl << endl;

            }

        } else if (choice == 4) {

            try {

                cout << stack << endl;

            } catch(Exception &e) {

                cout << endl << e.msg() << endl << endl;

            }

        } else if (choice == 5) {

            cout << "Exiting..." << endl;

            cout << endl << "------ Author ------" << endl;

            cout << "Ali Izzath Shazin K" << endl;

            cout << "220071601028" << endl;

            cout << "B.Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option." << endl;

        }

    }

    return 0;

}

**OUTPUT**

Enter size of the stack: 4

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 1

Enter element to push: 2

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 1

Enter element to push: 3

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 1

Enter element to push: 1

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 4

1 <-- Top

3

2

Size = 4

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 2

Popped element: 1

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 3

Top element: 3

---------- Stack Operations ----------

1. Push.

2. Pop.

3. Peek.

4. Display.

5. Exit.

Enter choice: 5

Exiting...

------ Author ------

Ali Izzath Shazin K

220071601028

B.Tech CSE A

PROGRAM NO: 6 DATE: 14-09-2023

**AIM**

To implement queue data structure with Enqueue, Dequeue, Peek and Display.

**ALGORITHM**

1. START
2. FUNC Queue(size): // Declaring queue
3. DECLARE arr by allocating required size.
4. SET rear = -1
5. SET fromt = 0
6. END FUNC
7. FUNC isEmpty():
8. IF rear = -1, return 1
9. ELSE, return 0
10. END IF, END FUNC
11. FUNC isFull():
12. IF rear = size – 1, return 1
13. ELSE, return 0
14. END IF, END FUNC
15. FUNC Enqueue(item):
16. IF isFull(), throw QueueOverflow
17. END IF
18. INCREMENT rear
19. SET arr[rear] = item
20. END FUNC
21. FUNC Dequeue(item):
22. IF isEmpty(), throw QueueUnderflow
23. END IF
24. SET popItem = arr[front]
25. INCREMENT front
26. IF front > rear
27. SET rear = -1, SET front = 0
28. END IF
29. Return popItem, END FUNC
30. FUNC Peek():
31. IF isEmpty(), throw StackEmpty
32. ELSE, return arr[front]
33. END IF, END FUNC
34. STOP

**CODE**

#include <iostream>

#include <cstdlib>

using namespace std;

class Queue {

    int \_size;

    int \_rear;

    int \_front;

    int \*arr;

    public:

    Queue(int size) {

        this->\_size = size;

        this->arr = (int\*) malloc(size \* sizeof(int));

        this->\_rear = -1;

        this->\_front = 0;

    }

    int isFull() {

        if (this->\_rear == this->\_size - 1) return 1;

        return 0;

    }

    int isEmpty() {

        if (this->\_rear == -1) return 1;

        return 0;

    }

    void Enqueue(int item) {

        if (this->isFull()) {

            cout << "Queue Overflow" << endl;

        } else {

            this->\_rear++;

            this->arr[this->\_rear] = item;

        }

    }

    void Dequeue() {

        if (this->isEmpty()) {

            cout << "Queue Underflow" << endl;

        } else {

            int poppedItem = this->arr[this->\_front];

            cout << "Popped Item: " << poppedItem << endl;

            this->\_front++;

            if (this->\_front > this->\_rear) {

                this->\_rear = -1;

                this->\_front = 0;

            }

        }

    }

    void Display() {

        cout << "[";

        for (int i=0; i<this->\_size; i++) {

            if (i < this->\_front) {

                cout << "\_";

            } else if (i <= this->\_rear) {

                cout << this->arr[i];

            } else {

                cout << "\_";

            }

            if (i != this->\_size - 1){

                cout << ", ";

            }

        }

        cout << "]" << endl;

    }

    void Peek() {

        if (this->isEmpty()) {

            cout << "Stack Underflow" << endl;

        } else {

            cout << "Front Element: " << this->arr[this->\_front] << endl;

        }

    }

};

int main() {

    int temp1, choice, size;

    cout << "Enter size of Queue: ";

    cin >> size;

    Queue q(size);

    while (1) {

        cout << "------ Queue Data Structure ------" << endl;

        cout << "1. Enqueue" << endl;

        cout << "2. Dequeue" << endl;

        cout << "3. Peek" << endl;

        cout << "4. Display" << endl;

        cout << "5. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter item to insert: ";

            cin >> temp1;

            q.Enqueue(temp1);

        } else if (choice == 2) {

            q.Dequeue();

        } else if (choice == 3) {

            q.Peek();

        } else if (choice == 4) {

            q.Display();

        } else if (choice == 5) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

        cout << endl;

    }

    return 0;

}

**OUTPUT**

Enter size of Queue: 3

------ Queue Data Structure ------

1. Enqueue

2. Dequeue

3. Peek

4. Display

5. Exit

Enter Choice: 1

Enter item to insert: 10

------ Queue Data Structure ------

1. Enqueue

2. Dequeue

3. Peek

4. Display

5. Exit

Enter Choice: 1

Enter item to insert: 20

------ Queue Data Structure ------

1. Enqueue

2. Dequeue

3. Peek

4. Display

5. Exit

Enter Choice: 4

[10, 20, \_]

------ Queue Data Structure ------

1. Enqueue

2. Dequeue

3. Peek

4. Display

5. Exit

Enter Choice: 2

Popped Item: 10

------ Queue Data Structure ------

1. Enqueue

2. Dequeue

3. Peek

4. Display

5. Exit

Enter Choice: 3

Front Element: 20

------ Queue Data Structure ------

1. Enqueue

2. Dequeue

3. Peek

4. Display

5. Exit

Enter Choice: 5

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 7 DATE: 21-09-2023

**AIM**

To implement Singly Linked List with Insertion, Deletion and Display Operations.

**ALGORITHM**

1. START
2. FUNC CreateNode(item):
3. DECLARE node\* with malloc()
4. SET node.value = item, node.next = NULL
5. RETURN node. END FUNC
6. FUNC LinkedList(): // constructor
7. SET this.Head = NULL, this.\_length = 0
8. END FUNC
9. FUNC Insert\_Start(node): // Insertion at the beginning
10. SET node.next = this.Head, this.Head = node
11. INCREMENT this.\_length. END FUNC
12. FUNC Insert\_End(node): // Insertion at the end
13. SET lastNode.next = node
14. INCREMENT this.\_length. END FUNC
15. FUNC Insert\_Middle(node): // Insertion at the middle
16. SET node.next = prevNode.next, prevNode.next = node
17. INCREMENT this.\_length. END FUNC
18. FUNC Delete(index):
19. IF index == 0:
20. DECLARE temp = this.Head, deletedItem = this.Head.value
21. SET this.Head = this.Head.next
22. DECREMENT this.\_length, free(temp), RETURN deletedItem
23. END IF
24. DECLARE temp = this.Head.next, prevValue = this.Head
25. FOR (i=1 to i<index):
26. SET prevValue = temp, temp = temp.next
27. END FOR
28. DECLARE deletedItem = temp.value
29. SET prevValue.next = temp.next, free(temp)
30. DECREMENT this.\_length, RETURN deletedItem. END FUNC
31. FUNC Display(): // Singly Linked List Traversal
32. DECLARE node = this.Head, count = 0
33. Print “[“
34. WHILE (node != NULL):
35. Print node.value
36. SET node = node.next
37. IF count != this.\_length - 1:
38. Print “]”. END IF
39. INCREMENT count. END WHILE
40. Print “]”. END FUNC
41. STOP

**CODE**

#include <iostream>

#include <cstdlib>

using namespace std;

class LinkedListException {

    public:

    virtual string msg() = 0;

};

class LinkedListIndexOutOfBound: public LinkedListException {

    public:

    string msg() {

        return "Linked List Out of Bound";

    }

};

typedef struct Node {

    int value;

    Node\* next;

} Node;

class LinkedList {

    Node\* Head;

    int \_length;

    Node\* createNode(int item) {

        Node\* node = (Node\*) malloc(sizeof(Node));

        node->value = item;

        node->next = NULL;

        return node;

    }

    void Insert\_Start(Node\* node) {

        node->next = this->Head;

        this->Head = node;

        this->\_length++;

    }

    void Insert\_End(Node\* node, Node\* lastNode) {

        lastNode->next = node;

        this->\_length++;

    }

    void Insert\_Middle(Node\* node, Node\* prevNode) {

        node->next = prevNode->next;

        prevNode->next = node;

        this->\_length++;

    }

    public:

    LinkedList() {

        this->Head = NULL;

        this->\_length = 0;

    }

    int length() {

        return this->\_length;

    }

    void Insert(int index, int item) {

        Node\* node = this->createNode(item);

        if (index < 0 || (index > this->\_length)) throw LinkedListIndexOutOfBound();

        if (index == 0) {

            this->Insert\_Start(node);

            return;

        }

        int count = 1;

        Node\* temp = this->Head;

        while (temp->next != NULL) {

            if (count == index) {

                this->Insert\_Middle(node, temp);

                return;

            }

            temp = temp->next;

            count++;

        }

        if (this->\_length == index) {

            this->Insert\_End(node, temp);

        }

    }

    int Delete(int index) {

        if (index < 0 || (index >= this->\_length)) throw LinkedListIndexOutOfBound();

        if (index == 0) {

            // Deleting from the beginning

            Node\* temp = this->Head;

            int deletedItem = this->Head->value;

            this->Head = this->Head->next;

            this->\_length--;

            free(temp);

            return deletedItem;

        }

        // Accessing the node to delete

        Node\* temp = this->Head->next;

        Node\* prevValue = this->Head;

        for (int i=1; i<index; i++) {

            prevValue = temp;

            temp = temp->next;

        }

        // Deleting from the middle or end

        int deletedItem = temp->value;

        prevValue->next = temp->next;

        free(temp);

        this->\_length--;

        return deletedItem;

    }

    void Display() {

        Node\* node = this->Head;

        int count = 0;

        cout << "[";

        while (node != NULL) {

            cout << node->value;

            node = node->next;

            if (count != this->\_length - 1) {

                cout << ", ";

            }

            count++;

        }

        cout << "]";

    }

};

int main() {

    int choice, temp1, temp2;

    LinkedList List;

    while (1) {

        cout << "---- Linked List Data Structure ----" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Delete" << endl;

        cout << "3. Display" << endl;

        cout << "4. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter item to insert : ";

            cin >> temp1;

            cout << "Enter index to insert to : ";

            cin >> temp2;

            try {

                List.Insert(temp2, temp1);

            } catch(LinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 2) {

            cout << "Enter index to delete : ";

            cin >> temp1;

            try {

                cout << "Deleted Item: " << List.Delete(temp1);

            } catch(LinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 3) {

            List.Display();

        } else if (choice == 4) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

        cout << endl;

    }

    return 0;

}

**OUTPUT**

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 1

Enter item to insert : 10

Enter index to insert to : 0

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 1

Enter item to insert : 20

Enter index to insert to : 1

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 1

Enter item to insert : 30

Enter index to insert to : 1

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 2

Enter index to delete : 0

Deleted Item: 10

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 3

[30, 20]

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 4

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 8 DATE: 28-09-2023

**AIM**

To implement Doubly Linked List with Insertion, Deletion and Display Operations.

**ALGORITHM**

1. START
2. FUNC CreateNode(item):
3. DECLARE node\* with malloc()
4. SET node.value = item, node.next = NULL, node.prev = NULL
5. RETURN node. END FUNC
6. FUNC DoublyLinkedList(): // constructor
7. SET this.Head = NULL, this.End = NULL, this.\_length = 0
8. END FUNC
9. FUNC Insert\_Start(node): // Insertion at the beginning
10. SET node.next = this.Head, this.Head = node
11. IF node.next == NULL: SET this.End = node
12. ELSE: SET node.next.prev = node, ENDIF
13. INCREMENT this.\_length. END FUNC
14. FUNC Insert\_End(node): // Insertion at the end
15. SET node.prev = this.End, this.End.next = node, this.End = node
16. INCREMENT this.\_length. END FUNC
17. FUNC Insert\_Middle(node, prevNode): // Insertion at the middle
18. SET node.next = prevNode.next, node.prev = prevNode, node.next.prev = node, prevNode.next = node
19. INCREMENT this.\_length. END FUNC
20. FUNC Delete\_Start():
21. DECLARE temp = this.Head, deletedItem = this.Head.value
22. SET this.Head = this.Head.next
23. IF this.Head == NULL: SET this.End = NULL
24. ELSE: SET this.Head.prev = NULL, ENDIF
25. DECREMENT this.\_length, free(temp)
26. RETURN deletedItem, END FUNC
27. FUNC Delete\_End():
28. DECLARE temp = this.Head, deletedItem = this.Head.value
29. SET this.End = this.End.prev
30. IF this.End == NULL: SET this.Head = NULL
31. ELSE: SET this.End.next= NULL, ENDIF
32. DECREMENT this.\_length, free(temp)
33. RETURN deletedItem, END FUNC
34. FUNC Delete\_Middle(index, nodeToDelete)
35. DECLARE deletedItem = nodeToDelete.value
36. SET nodeToDelete.prev.next = nodeToDelete.next, nodeToDelete.next.prev = nodeToDelete.prev
37. DECREMENT this.\_length, free(temp)
38. RETURN deletedItem, END FUNC
39. FUNC Display():
40. DECLARE node = this.Head, count = 0
41. Print “[“
42. WHILE (node != NULL):
43. Print node.value
44. SET node = node.next
45. IF count != this.\_length - 1:
46. Print “]”. END IF
47. INCREMENT count. END WHILE
48. Print “]”. END FUNC
49. FUNC DisplayReverse():
50. DECLARE node = this.End, count = 0
51. Print “[“
52. WHILE (node != NULL):
53. Print node.value
54. SET node = node.prev
55. IF count != this.\_length - 1:
56. Print “]”. END IF
57. INCREMENT count. END WHILE
58. Print “]”. END FUNC
59. STOP

**CODE**

#include <iostream>

#include <cstdlib>

using namespace std;

class DoublyLinkedListException {

    public:

    virtual string msg() = 0;

};

class DoublyLinkedListIndexOutOfBound: public DoublyLinkedListException {

    public:

    string msg() {

        return "Linked List Out of Bound";

    }

};

typedef struct Node {

    int value;

    Node\* next;

    Node\* prev;

} Node;

class DoublyLinkedList {

    Node\* Head;

    Node\* End;

    int \_length;

    Node\* createNode(int item) {

        Node\* node = (Node\*) malloc(sizeof(Node));

        node->value = item;

        node->next = NULL;

        node->prev = NULL;

        return node;

    }

    void Insert\_Start(Node\* node) {

        node->next = this->Head;

        this->Head = node;

        if (node->next == NULL) {

            // If first item

            this->End = node;

        } else {

            // If not first item

            node->next->prev = node;

        }

        this->\_length++;

    }

    void Insert\_End(Node\* node) {

        node->prev = this->End;

        this->End->next = node;

        this->End = node;

        this->\_length++;

    }

    void Insert\_Middle(Node\* node, Node\* prevNode) {

        node->next = prevNode->next;

        node->prev = prevNode;

        node->next->prev = node;

        prevNode->next = node;

        this->\_length++;

    }

    int Delete\_Start() {

        Node\* temp = this->Head;

        int deletedItem = this->Head->value;

        this->Head = this->Head->next;

        if (this->Head == NULL) {

            // If all items deleted

            this->End = NULL;

        } else {

            // If more items are left

            this->Head->prev = NULL;

        }

        this->\_length--;

        free(temp);

        return deletedItem;

    }

    int Delete\_End() {

        Node\* temp = this->End;

        int deletedItem = this->End->value;

        this->End = this->End->prev;

        if (this->End == NULL) {

            // If all items deleted

            this->Head = NULL;

        } else {

            // If more items are left

            this->End->next = NULL;

        }

        this->\_length--;

        free(temp);

        return deletedItem;

    }

    int Delete\_Middle(int index, Node\* nodeToDelete) {

        int deletedItem = nodeToDelete->value;

        nodeToDelete->prev->next = nodeToDelete->next;

        nodeToDelete->next->prev = nodeToDelete->prev;

        this->\_length--;

        free(nodeToDelete);

        return deletedItem;

    }

    public:

    DoublyLinkedList() {

        this->Head = NULL;

        this->\_length = 0;

    }

    int length() {

        return this->\_length;

    }

    Node\* Get\_Node(int index, int instance) {

        if (index < 0 || (index >= this->\_length)) throw DoublyLinkedListIndexOutOfBound();

        Node\* temp;

        int count = 0;

        if (index <= ((this->\_length - 1) / 2)) {

            temp = this->Head;

            for (int i=0; i<index; i++) {

                count++;

                temp = temp->next;

            }

            if (instance == 1) cout << "Number of nodes traversed from left: " << count << endl;

        } else {

            temp = this->End;

            for (int i=this->\_length - 1; i>index; i--) {

                count++;

                temp = temp->prev;

            }

            if (instance == 1) cout << "Number of nodes traversed from right: " << count << endl;

        }

        return temp;

    }

    void Insert(int index, int item) {

        Node\* node = this->createNode(item);

        if (index < 0 || (index > this->\_length)) throw DoublyLinkedListIndexOutOfBound();

        if (index == 0) {

            this->Insert\_Start(node);

            return;

        }

        if (index == this->\_length) {

            this->Insert\_End(node);

            return;

        }

        Node\* temp = this->Get\_Node(index, 0);

        this->Insert\_Middle(node, temp->prev);

        return;

    }

    int Delete(int index) {

        if (index < 0 || (index >= this->\_length)) throw DoublyLinkedListIndexOutOfBound();

        if (index == 0) {

            return this->Delete\_Start();

        }

        if (index == this->\_length - 1) {

            return this->Delete\_End();

        }

        Node\* temp = this->Get\_Node(index, 0);

        return this->Delete\_Middle(index, temp);

    }

    void Display() {

        Node\* node = this->Head;

        int count = 0;

        cout << "[";

        while (node != NULL) {

            cout << node->value;

            node = node->next;

            if (count != this->\_length - 1) {

                cout << ", ";

            }

            count++;

        }

        cout << "]";

    }

    void DisplayReverse() {

        Node\* node = this->End;

        int count = 0;

        cout << "[";

        while (node != NULL) {

            cout << node->value;

            node = node->prev;

            if (count != this->\_length - 1) {

                cout << ", ";

            }

            count++;

        }

        cout << "]";

    }

};

int main() {

    int choice, temp1, temp2;

    DoublyLinkedList List;

    while (1) {

        cout << "---- Doubly Linked List Data Structure ----" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Delete" << endl;

        cout << "3. Display" << endl;

        cout << "4. Display Reverse" << endl;

        cout << "5. Access A Node" << endl;

        cout << "6. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter item to insert : ";

            cin >> temp1;

            cout << "Enter index to insert to : ";

            cin >> temp2;

            try {

                List.Insert(temp2, temp1);

            } catch(DoublyLinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 2) {

            cout << "Enter index to delete : ";

            cin >> temp1;

            try {

                int deletedValue = List.Delete(temp1);

                cout << "Deleted Item: " << deletedValue;

            } catch(DoublyLinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 3) {

            List.Display();

        } else if (choice == 4) {

            List.DisplayReverse();

        } else if (choice == 5) {

            cout << "Enter index to access : ";

            cin >> temp1;

            try {

                int accessedValue = List.Get\_Node(temp1, 1)->value;

                cout << "Accessed Node: " << accessedValue;

            } catch(DoublyLinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 6) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

        cout << endl;

    }

    return 0;

}

**OUTPUT**

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 1

Enter item to insert : 10

Enter index to insert to : 0

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 1

Enter item to insert : 20

Enter index to insert to : 1

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 1

Enter item to insert : 30

Enter index to insert to : 1

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 3

[10, 30, 20]

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 4

[20, 30, 10]

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 5

Enter index to access : 0

Number of nodes traversed from left: 0

Accessed Node: 10

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 5

Enter index to access : 2

Number of nodes traversed from right: 0

Accessed Node: 20

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 2

Enter index to delete : 1

Deleted Item: 30

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 3

[10, 20]

---- Doubly Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 6

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 9 DATE: 19-10-2023

**AIM**

To implement Binary Search Tree with Insertion, Inorder, Postorder and Preorder Traversal.

**ALGORITHM**

1. START
2. FUNC CreateNode(value):
3. DECLARE node with malloc()
4. SET node.value = value
5. SET node.left = NULL
6. SET node.right = NULL
7. RETURN node, END FUNC
8. FUNC INSERT(value):
9. DECLARE newNode = this.CreateNode(value)
10. DECLARE root = this.Root
11. DECLARE prevRoot = NULL
12. DECLARE WHILE root != NULL:
13. SET prevRoot = root
14. IF newNode.value <= root.value:
15. SET root = root.left
16. ELSE: SET root = root.right
17. END WHILE
18. IF newNode.value <= prevRoot.value: SET prevRoot.left = newNode
19. ELSE: SET prevRoot.right = newNode
20. END FUNC
21. FUNC In\_Order\_Traversal(root)
22. IF root == NULL: RETURN
23. END IF
24. INVOKE In\_Order\_Traversal(root.left)
25. Print root.value
26. INVOKE In\_Order\_Traversal(root.right)
27. END FUNC
28. FUNC Pre\_Order\_Traversal(root)
29. IF root == NULL: RETURN
30. END IF
31. Print root.value
32. INVOKE Pre\_Order\_Traversal(root.left)
33. INVOKE Pre\_Order\_Traversal(root.right)
34. END FUNC
35. FUNC Post\_Order\_Traversal(root)
36. IF root == NULL: RETURN
37. END IF
38. INVOKE Post\_Order\_Traversal(root.left)
39. INVOKE Post\_Order\_Traversal(root.right)
40. Print root.value
41. END FUNC
42. STOP

**CODE**

#include <iostream>

#include <cstdlib>

using namespace std;

typedef struct Node {

    int value;

    Node\* left;

    Node\* right;

} Node;

class BinarySearchTree {

    Node\* CreateNode(int value) {

        Node\* node = (Node\*) malloc(sizeof(Node));

        node->value = value;

        node->left = NULL;

        node->right = NULL;

        return node;

    }

    public:

    Node\* Root;

    BinarySearchTree(int value) {

        Node\* newNode = this->CreateNode(value);

        this->Root = newNode;

    }

    void Insert(int value) {

        Node\* newNode = this->CreateNode(value);

        Node\* root = this->Root;

        Node\* prevRoot = NULL;

        while (root != NULL) {

            prevRoot = root;

            if (newNode->value <= root->value) {

                root = root->left;

            } else {

                root = root->right;

            }

        }

        if (newNode->value <= prevRoot->value) {

            prevRoot->left = newNode;

        } else {

            prevRoot->right = newNode;

        }

    }

    void DisplaySubTree(Node\* node, int indent, int side) {

        for (int i=0; i<indent; i++) {

            cout << "    ";

        }

        if (side == 1) {

            cout << "L";

        } else if (side == 2) {

            cout << "R";

        }

        if (node->left == NULL) {

            cout << "|" << node->value << endl;

        } else {

            cout << "\\" << node->value << endl;

        }

        if (node->left != NULL) {

            this->DisplaySubTree(node->left, indent + 1, 1);

        }

        if (node->right != NULL) {

            this->DisplaySubTree(node->right, indent + 1, 2);

        }

    }

    void In\_Order\_Traversal(Node\* root) {

        if (root == NULL)

            return;

        In\_Order\_Traversal(root->left);

        cout<<root->value<<" ";

        In\_Order\_Traversal(root->right);

    }

    void Pre\_Order\_Traversal(Node\* root) {

        if (root == NULL)

            return;

        cout<<root->value<<" ";

        Pre\_Order\_Traversal(root->left);

        Pre\_Order\_Traversal(root->right);

    }

    void Post\_Order\_Traversal(Node\* root) {

        if (root == NULL)

            return;

        Post\_Order\_Traversal(root->left);

        Post\_Order\_Traversal(root->right);

        cout<<root->value<<" ";

    }

};

int main() {

    int choice, temp;

    cout << "Enter the root node value: ";

    cin >> temp;

    BinarySearchTree bst(temp);

    while (true) {

        cout << "\n---- Binary Search Tree (BST) ----" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Inorder Traversal" << endl;

        cout << "3. Preorder Traversal" << endl;

        cout << "4. Postorder Traversal" << endl;

        cout << "5. Display BST" << endl;

        cout << "6. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter value to insert: ";

            cin >> temp;

            bst.Insert(temp);

        } else if (choice == 2) {

            cout << "Inorder => ";

            bst.In\_Order\_Traversal(bst.Root);

            cout << endl;

        } else if (choice == 3) {

            cout << "Preorder => ";

            bst.Pre\_Order\_Traversal(bst.Root);

            cout << endl;

        } else if (choice == 4) {

            cout << "Postorder => ";

            bst.Post\_Order\_Traversal(bst.Root);

            cout << endl;

        } else if (choice == 5) {

            bst.DisplaySubTree(bst.Root, 0, 0);

        } else if (choice == 6) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

    }

    return 0;}

**OUTPUT**

Enter the root node value: 10

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 1

Enter value to insert: 5

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 1

Enter value to insert: 15

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 1

Enter value to insert: 8

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 1

Enter value to insert: 12

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 5

\10

L|5

R|8

R\15

L|12

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 2

Inorder => 5 8 10 12 15

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 3

Preorder => 10 5 8 15 12

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 4

Postorder => 8 5 12 15 10

---- Binary Search Tree (BST) ----

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Display BST

6. Exit

Enter Choice: 6

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A

PROGRAM NO: 10 DATE: 26-10-2023

**AIM**

To implement AVL Tree with Insertion (Balancing) and Display Operations

**ALGORITHM**

1. START
2. FUNC newNode(key):
3. DECLARE node = new Node()
4. SET node.value = key
5. SET node.left = NULL
6. SET node.right = NULL
7. SET node.height = 1
8. RETURN node.
9. END FUNC
10. FUNC rightRotate(y):
11. DECLARE x = y.left, T2 = x.right
12. SET x.right = y, y.left = T2
13. SET y.height = max(height(y.left), height(y.right)) + 1
14. SET x.height = max(height(x.left), height(x.right)) + 1
15. RETURN x. END FUNC
16. FUNC leftRotate(x):
17. DECLARE y = x.left, T2 = y.left
18. SET y.left = x, x.right= T2
19. SET x.height = max(height(x.left), height(x.right)) + 1
20. SET y.height = max(height(y.left), height(y.right)) + 1
21. RETURN x. END FUNC
22. FUNC getBalance(N):
23. IF N == NULL: RETURN 0
24. RETURN height(N.left) - height(N.right)
25. END FUNC
26. FUNC Insert(node, value):
27. IF node == NULL: RETURN newNode(value). END IF
28. IF value <= node.value: node.left = Insert(node.left, value)
29. ELSE IF value > node.value: node.right = Insert(node.right, value)
30. END IF
31. node.height = 1 + max(height(node.left), height(node.right))
32. DELCARE balance = getBalance(node)
33. IF balance > 1 AND value < node.left.value: RETURN rightRotate(node). END IF
34. IF balance < -1 AND value > node.right.value: RETURN leftRotate(node). END IF
35. IF balance > 1 AND value > node.left.value:
36. node.left = leftRotate(node.left)
37. RETURN rightRotate(node). END IF
38. IF balance < -1 AND value < node.right.value:
39. node.right = rightRotate(node.left)
40. RETURN leftRotate(node). END IF
41. RETURN node
42. END FUNC
43. STOP

**CODE**

#include <bits/stdc++.h>

using namespace std;

typedef struct Node {

    int value;

    Node \*left;

    Node \*right;

    int height;

} Node;

class AVLTree {

    public:

    Node\* rootNode;

    AVLTree() {

        this->rootNode = NULL;

    }

    int height(Node \*N) {

        if (N == NULL) return 0;

        return N->height;

    }

    int max(int a, int b) {

        return (a > b)? a : b;

    }

    Node\* newNode(int key) {

        Node\* node = new Node();

        node->value = key;

        node->left = NULL;

        node->right = NULL;

        node->height = 1;

        return node;

    }

    Node \*rightRotate(Node \*y) {

        Node \*x = y->left;

        Node \*T2 = x->right;

        // Perform rotation

        x->right = y;

        y->left = T2;

        // Update heights

        y->height = this->max(this->height(y->left), this->height(y->right)) + 1;

        x->height = this->max(this->height(x->left), this->height(x->right)) + 1;

        // Return new root

        return x;

    }

    Node \*leftRotate(Node \*x) {

        Node \*y = x->right;

        Node \*T2 = y->left;

        // Perform rotation

        y->left = x;

        x->right = T2;

        // Update heights

        x->height = this->max(this->height(x->left), this->height(x->right)) + 1;

        y->height = this->max(this->height(y->left), this->height(y->right)) + 1;

        // Return new root

        return y;

    }

    // Get Balance factor of node N

    int getBalance(Node \*N) {

        if (N == NULL) return 0;

        return this->height(N->left) - this->height(N->right);

    }

    Node\* insert(Node\* node, int value) {

        if (node == NULL) {

            return this->newNode(value);

        }

        if (value <= node->value)

            node->left = this->insert(node->left, value);

        else if (value > node->value)

            node->right = this->insert(node->right, value);

        node->height = 1 + this->max(this->height(node->left), this->height(node->right));

        int balance = this->getBalance(node);

        // Left Left Case

        if (balance > 1 && value < node->left->value)

            return this->rightRotate(node);

        // Right Right Case

        if (balance < -1 && value > node->right->value) {

            return this->leftRotate(node);

        }

        // Left Right Case

        if (balance > 1 && value > node->left->value) {

            node->left = this->leftRotate(node->left);

            return this->rightRotate(node);

        }

        // Right Left Case

        if (balance < -1 && value < node->right->value) {

            node->right = this->rightRotate(node->right);

            return this->leftRotate(node);

        }

        return node;

    }

    void Insert(int value) {

        this->rootNode = this->insert(this->rootNode, value);

    }

    void DisplaySubTree(Node\* node, int indent, int side) {

        if (this->rootNode == NULL) return;

        for (int i=0; i<indent; i++) {

            cout << "    ";

        }

        if (side == 1) {

            cout << "L";

        } else if (side == 2) {

            cout << "R";

        }

        if (node->left == NULL) {

            cout << "|" << node->value << endl;

        } else {

            cout << "\\" << node->value << endl;

        }

        if (node->left != NULL) {

            this->DisplaySubTree(node->left, indent + 1, 1);

        }

        if (node->right != NULL) {

            this->DisplaySubTree(node->right, indent + 1, 2);

        }

    }

};

int main() {

    int choice, temp;

    AVLTree avl;

    while (true) {

        cout << "\n---- Binary Search Tree (BST) ----" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Display" << endl;

        cout << "3. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter value to insert: ";

            cin >> temp;

            avl.Insert(temp);

        } else if (choice == 2) {

            avl.DisplaySubTree(avl.rootNode, 0, 0);

        } else if (choice == 3) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

    }

    return 0;

}

**OUTPUT**

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 50

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 60

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 70

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 80

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 90

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 40

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 30

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 2

\60

L\40

L|30

R|50

R\80

L|70

R|90

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 3

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A