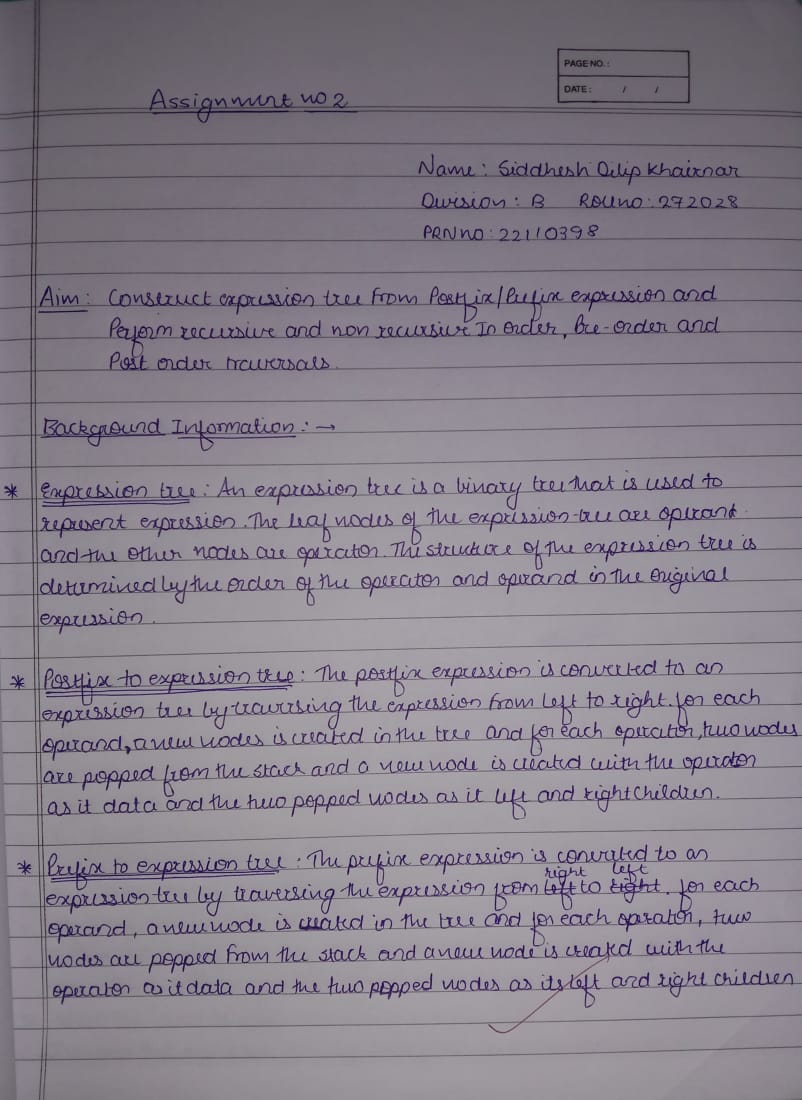
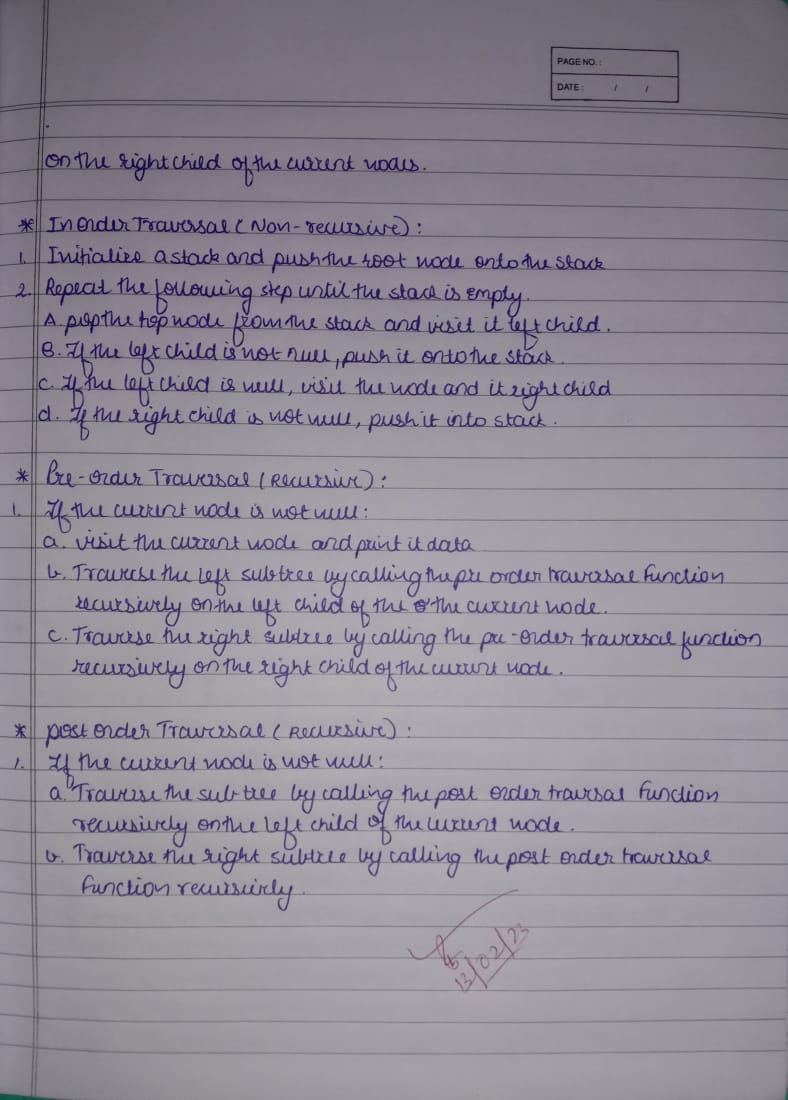
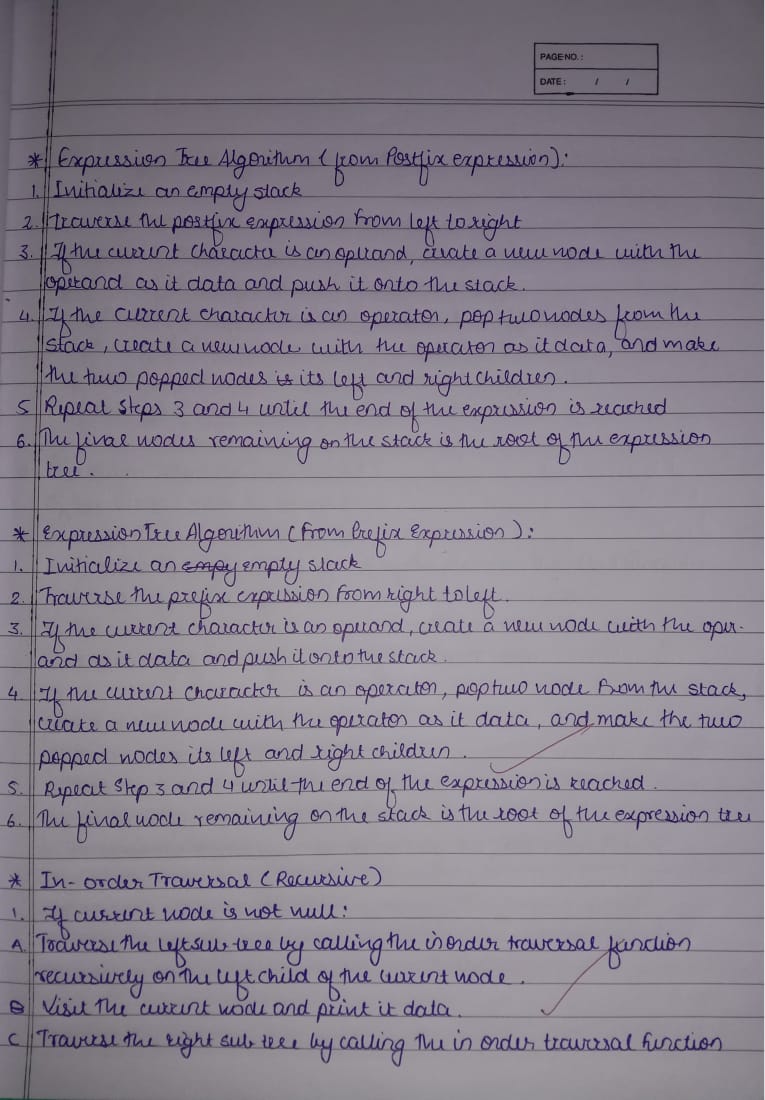
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Bansilal Ramnath Agarwal Charitable Trust's**  **Vishwakarma Institute of Information Technology**  **Department of**  **Artificial Intelligence and Data Science** | | |
| Student Name: Siddhesh Dilip Khairnar | | | |
| Class: S.Y. | Division: B | | Roll No: 272028 |
| Semester: IV | | Academic Year: 2022 - 23 | |
| Subject Name & Code: Advanced Data Structures: ADUA22202 | | | |
| Title of Assignment: Write a program to construct an expression tree from postfix/prefix expression and perform In-order, pre-order and post-order traversals. | | | |
| Date of Performance: 1/02/2023 | | Date of Submission: 10/02/2023 | |





**CODE AND OUTPUT**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

class Node

{

public:

    Node \*left;

    Node \*right;

    char value;

    Node(char value)

    {

        this->value = value;

        this->left = NULL;

        this->right = NULL;

    }

};

bool is\_operand(char c)

{

    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

    {

        return true;

    }

    return false;

}

Node \*build\_tree(std::string expression, bool is\_postfix)

{

    std::stack<Node \*> stack;

    for (int i = 0; i < expression.length(); i++)

    {

        char c = expression[i];

        if (is\_operand(c))

        {

            stack.push(new Node(c));

        }

        else

        {

            Node \*node = new Node(c);

            if (is\_postfix)

            {

                node->right = stack.top();

                stack.pop();

                node->left = stack.top();

                stack.pop();

            }

            else

            {

                node->left = stack.top();

                stack.pop();

                node->right = stack.top();

                stack.pop();

            }

            stack.push(node);

        }

    }

    return stack.top();

}

void in\_order\_traversal(Node \*root)

{

    if (root != NULL)

    {

        in\_order\_traversal(root->left);

        std::cout << root->value << " ";

        in\_order\_traversal(root->right);

    }

}

void pre\_order\_traversal(Node \*root)

{

    if (root != NULL)

    {

        cout << root->value << " ";

        pre\_order\_traversal(root->left);

        pre\_order\_traversal(root->right);

    }

}

void post\_order\_traversal(Node \*root)

{

    if (root != NULL)

    {

        post\_order\_traversal(root->left);

        post\_order\_traversal(root->right);

        cout << root->value << " ";

    }

}

int main()

{

    string expression = "ab+ef\*g\*-";

    Node \*root = build\_tree(expression, false);

    cout << "In-order: ";

    in\_order\_traversal(root);

    cout << endl;

    cout << "Pre-order: ";

    pre\_order\_traversal(root);

    cout << endl;

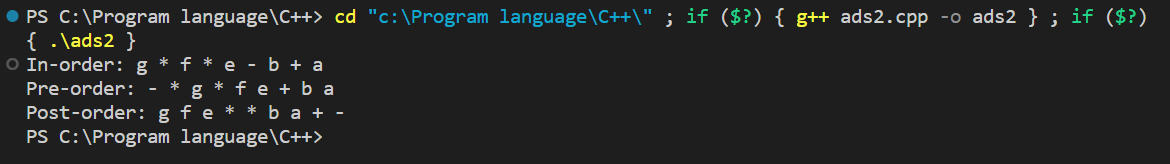
    cout << "Post-order: ";

    post\_order\_traversal(root);

    cout << endl;

    return 0;

}



Conclusion: Successfully implemented program for expression tree from postfix/prefix expression and perform In-order, pre-order and post-order traversals.