Lab 9 : Write a program for code generation.

```
#include <iostream>
#include <stack>
#include <string>

struct bin_tree
{
   char data;
   int label;
   bin_tree* right;
   bin_tree* left;
};

typedef bin_tree node;

void insertNode(node** tree, char val)
{
   node* temp = nullptr;

   if (!(*tree))
   {
      temp = new node;
      temp->left = nullptr;
      temp->right = nullptr;
      temp->data = val;
      temp->label = -1;
      *tree = temp;
   }
}

void insert(node** tree, char val)
{
   char l, r;
   int numofchildren;

   insertNode(tree, val);

   std::cout << "\nEnter number of children of " << val << ": ";
   std::cin >> numofchildren;

   if (numofchildren == 2)
   {
      std::cout << "\nEnter Left Child of " << val << ": ";
      std::cin >> l;
      insertNode(&(*tree)->left, l);

      std::cout << "\nEnter Right Child of " << val << ": ";
```

```
        std::cin >> r;
        insertNode(&(*tree)->right, r);

        insert(&(*tree)->left, l);
        insert(&(*tree)->right, r);
    }
}

void findLeafNodeLabel(node* tree, int val)
{
    if (tree->left != nullptr && tree->right != nullptr)
    {
        findLeafNodeLabel(tree->left, 1);
        findLeafNodeLabel(tree->right, 0);
    }
    else
    {
        tree->label = val;
    }
}

void findInteriorNodeLabel(node* tree)
{
    if (tree->left->label == -1)
    {
        findInteriorNodeLabel(tree->left);
    }
    else if (tree->right->label == -1)
    {
        findInteriorNodeLabel(tree->right);
    }
    else
    {
        if (tree->left != nullptr && tree->right != nullptr)
        {
            if (tree->left->label == tree->right->label)
            {
                tree->label = tree->left->label + 1;
            }
            else
            {
                if (tree->left->label > tree->right->label)
                {
                    tree->label = tree->left->label;
                }
                else
                {
                    tree->label = tree->right->label;
```

```
            }
          }
        }
    }
}

void printInorder(node* tree)
{
    if (tree)
    {
        printInorder(tree->left);
        std::cout << tree->data << " with Label " << tree->label << std::endl;
        printInorder(tree->right);
    }
}

void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}

int pop(int* R, int& top)
{
    int temp = R[top];
    top--;
    return temp;
}

void push(int* R, int& top, int temp, int numOfRegisters)
{
    if (top == numOfRegisters - 1)
    {
        std::cout << "Stack overflow! Storing in temporary variable T." << std::endl;
        top = numOfRegisters; // Reset top to indicate overflow
        R[top] = temp; // Store value in T
    }
    else
    {
        top++;
        R[top] = temp;
    }
}

std::string operationName(char temp)
{
    switch (temp)
```

```
    {
    case '+':
        return "ADD";
    case '-':
        return "SUB";
    case '*':
        return "MUL";
    case '/':
        return "DIV";
    default:
        return "";
    }
}

void generateCode(node* tree, int* R, int& top, int numOfRegisters)
{
    if (tree->left != nullptr && tree->right != nullptr)
    {
        if (tree->left->label == 1 && tree->right->label == 0 && tree->left->left == nullptr && tree->left-
>right == nullptr && tree->right->left == nullptr && tree->right->right == nullptr)
        {
            std::cout << "MOV " << tree->left->data << ", ";
            if (top == numOfRegisters)
                std::cout << "T";
            else
                std::cout << "R[" << top << "]";
            std::cout << "\n";
            std::string op = operationName(tree->data);
            std::cout << op << " " << tree->right->data << ", ";
            if (top == numOfRegisters)
                std::cout << "T";
            else
                std::cout << "R[" << top << "]";
            std::cout << "\n";
        }
        else if (tree->left->label >= 1 && tree->right->label == 0)
        {
            generateCode(tree->left, R, top, numOfRegisters);
            std::string op = operationName(tree->data);
            std::cout << op << " " << tree->right->data << ", ";
            if (top == numOfRegisters)
                std::cout << "T";
            else
                std::cout << "R[" << top << "]";
            std::cout << "\n";
        }
        else if (tree->left->label < tree->right->label)
        {
```

```
            swap(R[top], R[top - 1]);
            generateCode(tree->right, R, top, numOfRegisters);
            int temp = pop(R, top);
            generateCode(tree->left, R, top, numOfRegisters);
            push(R, top, temp, numOfRegisters);
            swap(R[top], R[top - 1]);
            std::string op = operationName(tree->data);
            std::cout << op << " R[" << top - 1 << "], ";
            if (top == numOfRegisters)
               std::cout << "T";
            else
               std::cout << "R[" << top << "]";
            std::cout << "\n";
         }
         else if (tree->left->label >= tree->right->label)
         {
            int temp;
            generateCode(tree->left, R, top, numOfRegisters);
            temp = pop(R, top);
            generateCode(tree->right, R, top, numOfRegisters);
            push(R, top, temp, numOfRegisters);
            std::string op = operationName(tree->data);
            std::cout << op << " R[" << top - 1 << "], ";
            if (top == numOfRegisters)
               std::cout << "T";
            else
               std::cout << "R[" << top << "]";
            std::cout << "\n";
         }
      }
      else if (tree->left == nullptr && tree->right == nullptr && tree->label == 1)
      {
         std::cout << "MOV " << tree->data << ", ";
         if (top == numOfRegisters)
            std::cout << "T";
         else
            std::cout << "R[" << top << "]";
         std::cout << "\n";
      }
   }

   void deleteTree(node* tree)
   {
      if (tree)
      {
         deleteTree(tree->left);
         deleteTree(tree->right);
         delete tree;
```

```
    }
}

int main()
{
    node* root = nullptr;
    node* tmp;
    char val;
    int i, temp;

    std::cout << "Enter root of tree: ";
    std::cin >> val;

    insert(&root, val);

    findLeafNodeLabel(root, 1);

    while (root->label == -1)
        findInteriorNodeLabel(root);

    int numOfRegisters;
    std::cout << "Enter the number of registers available: ";
    std::cin >> numOfRegisters;

    int* R = new int[numOfRegisters];
    temp = numOfRegisters - 1;
    for (i = 0; i < numOfRegisters; i++)
    {
        R[i] = temp;
        temp--;
    }

    std::cout << "\nInorder Display:\n";
    printInorder(root);

    std::cout << "\nAssembly Code:\n";
    int top = root->label - 1;
    generateCode(root, R, top, numOfRegisters);

    deleteTree(root);
    delete[] R;

    return 0;
}
```

OUTPUT:

```
Enter root of tree: -

Enter number of children of -: 2

Enter Left Child of -: +

Enter Right Child of -: -

Enter number of children of +: 2

Enter Left Child of +: a

Enter Right Child of +: b

Enter number of children of a: 0

Enter number of children of b: 0

Enter number of children of -: 2

Enter Left Child of -: e

Enter Right Child of -: +

Enter number of children of e: 0

Enter number of children of +: 2

Enter Left Child of +: c

Enter Right Child of +: d

Enter number of children of c: 0

Enter number of children of d: 0
Enter the number of registers available: 1
```

```
Enter the number of registers available: 1

Inorder Display:
a with Label 1
+ with Label 1
b with Label 0
- with Label 2
e with Label 1
- with Label 2
c with Label 1
+ with Label 1
d with Label 0

Assembly Code:
MOV e, T
MOV c, R[0]
ADD d, R[0]
Stack overflow! Storing in temporary variable T.
SUB R[0], T
MOV a, R[0]
ADD b, R[0]
Stack overflow! Storing in temporary variable T.
SUB R[0], T


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