# **Laboratory-7**

## **Question**

## Perform DAG construction and simple code generation.

### **Dag.cpp:**

#include <bits/stdc++.h>

**using** **namespace** std;

**struct** label\_list

{

**char** value;

**struct** label\_list \*next;

};

**struct** dag\_node

{

**char** value;

**int** index;

**int** label;

**int** parent\_count;

**struct** label\_list \*labels;

**struct** dag\_node \*left, \*right;

};

**struct** root\_list

{

**struct** dag\_node \*root;

**struct** root\_list \*next;

};

**struct** three\_address

{

**char** op;

**char** op1;

**char** op2;

**char** lhs;

};

**int** index\_global = 0, n;

three\_address tac[100];

**bool** visited\_left\_most\_child = 0;

stack<**int**> rstack, tstack;

label\_list \*insert\_label(label\_list \*curr, **char** c)

{

label\_list \*temp = **new** label\_list;

temp->value = c;

temp->next = NULL;

**if** (curr == NULL)

{

**return** temp;

}

label\_list \*start = curr;

**while** (start->next)

{

start = start->next;

}

start->next = temp;

**return** start;

}

dag\_node \*create\_dag\_node(**char** op, **char** result)

{

dag\_node \*temp = **new** dag\_node;

temp->index = index\_global++;

temp->value = op;

temp->left = temp->right = NULL;

temp->labels = NULL;

temp->labels = insert\_label(temp->labels, result);

**return** temp;

}

**bool** search\_labels(dag\_node \*x, **char** value)

{

**if** (x->value == value)

{

**return** 1;

}

label\_list \*start = x->labels;

**while** (start)

{

**if** (start->value == value)

{

**return** 1;

}

start = start->next;

}

**return** 0;

}

**bool** check(dag\_node \*n1, dag\_node \*n2, **char** c1, **char** c2)

{

**if** (search\_labels(n1, c1) && search\_labels(n2, c2))

{

**return** 1;

}

**return** 0;

}

dag\_node \*search\_for\_value(root\_list \*root, **char** value)

{

**int** recent = -1;

queue<dag\_node \*> q;

dag\_node \*x, \*result = NULL;

vector<**bool**> visited(index\_global, 0);

root\_list \*start = root;

**while** (start)

{

q.push(start->root);

start = start->next;

}

**while** (!q.empty())

{

x = q.front();

q.pop();

visited[x->index] = 1;

**if** (search\_labels(x, value))

{

**if** (x->index > recent)

{

recent = x->index;

result = x;

}

}

dag\_node \*left = x->left;

dag\_node \*right = x->right;

**if** (left && !visited[left->index])

q.push(left);

**if** (right && !visited[right->index])

q.push(right);

}

**return** result;

}

dag\_node \*search\_for\_similar(root\_list \*root, **char** op, **char** op1, **char** op2)

{

**int** recent = -1;

queue<dag\_node \*> q;

dag\_node \*x, \*result = NULL;

vector<**bool**> visited(index\_global, 0);

root\_list \*start = root;

**while** (start)

{

q.push(start->root);

start = start->next;

}

**while** (!q.empty())

{

x = q.front();

q.pop();

visited[x->index] = 1;

**if** (search\_labels(x, op))

{

**if** (check(x->left, x->right, op1, op2))

{

**if** (x->index > recent)

{

result = x;

recent = x->index;

}

}

}

dag\_node \*left = x->left;

dag\_node \*right = x->right;

**if** (left && !visited[left->index])

q.push(left);

**if** (right && !visited[right->index])

q.push(right);

}

**return** result;

}

root\_list \*add\_to\_end(root\_list \*curr, dag\_node \*root)

{

root\_list \*temp = **new** root\_list;

temp->root = root;

temp->next = NULL;

**if** (curr == NULL)

{

**return** temp;

}

root\_list \*start = curr;

**while** (start->next)

{

start = start->next;

}

start->next = temp;

**return** curr;

}

root\_list \*create\_dag(root\_list \*root, **int** curr)

{

**if** (curr == n)

{

**return** root;

}

**if** (root == NULL)

{

dag\_node \*parent, \*left, \*right;

parent = create\_dag\_node(tac[curr].op, tac[curr].lhs);

left = create\_dag\_node(tac[curr].op1, tac[curr].op1);

right = create\_dag\_node(tac[curr].op2, tac[curr].op2);

parent->left = left;

parent->right = right;

root = **new** root\_list;

root->root = parent;

root->next = NULL;

**return** create\_dag(root, curr + 1);

}

**if** (tac[curr].op == '=')

{

dag\_node \*temp = search\_for\_value(root, tac[curr].op1);

temp->labels = insert\_label(temp->labels, tac[curr].lhs);

**return** create\_dag(root, curr + 1);

}

**else**

{

dag\_node \*parent = search\_for\_similar(root, tac[curr].op, tac[curr].op1, tac[curr].op2);

dag\_node \*left = search\_for\_value(root, tac[curr].op1);

dag\_node \*right = search\_for\_value(root, tac[curr].op2);

{

**if** (left && right && parent)

{

**if** (parent->left->index == left->index && parent->right->index == right->index)

parent->labels = insert\_label(parent->labels, tac[curr].lhs);

**return** create\_dag(root, curr + 1);

}

}

parent = create\_dag\_node(tac[curr].op, tac[curr].lhs);

**if** (left == NULL)

{

left = create\_dag\_node(tac[curr].op1, tac[curr].op1);

}

**if** (right == NULL)

{

right = create\_dag\_node(tac[curr].op2, tac[curr].op2);

}

parent->left = left;

parent->right = right;

root\_list \*start = root;

root\_list \*temp = NULL;

**while** (start)

{

**if** (start->root != left && start->root != right)

{

temp = add\_to\_end(temp, start->root);

}

start = start->next;

}

temp = add\_to\_end(temp, parent);

root = temp;

**return** create\_dag(root, curr + 1);

}

}

**void** inorder(dag\_node \*curr, vector<**bool**> &visited)

{

curr->parent\_count++;

**if** (visited[curr->index])

{

**if** (curr->left != NULL)

{

inorder(curr->left, visited);

}

**if** (curr->right != NULL)

{

inorder(curr->right, visited);

}

**return**;

}

visited[curr->index] = 1;

**if** (curr->left == NULL && curr->right == NULL)

{

cout << "Leaf with Index: " << curr->index << " ,Value: " << curr->value << " ,Label: " << curr->label << "\n\n";

}

**else**

{

cout << "Index: " << curr->index << " ,Value: " << curr->value << " ,Label: " << curr -> label << '\n';

label\_list \*temp = curr->labels;

cout << "Labels are: ";

**while** (temp)

{

cout << temp->value << ' ';

temp = temp->next;

}

cout << '\n';

cout << "Left child has index " << curr->left->index << '\n';

cout << "Right child has index " << curr->right->index << '\n';

cout << '\n';

inorder(curr->left, visited);

inorder(curr->right, visited);

}

}

dag\_node \*assign\_labels(dag\_node \*curr, **bool** left\_child, vector<**bool**> &visited)

{

**if** (visited[curr->index])

{

**return** curr;

}

visited[curr->index] = 1;

**if** (curr->left == NULL && curr->right == NULL)

{

**if** (!left\_child)

{

curr->label = 0;

}

**else**

{

curr->label = 1;

}

**return** curr;

}

curr->left = assign\_labels(curr->left, 1, visited);

curr->right = assign\_labels(curr->right, 0, visited);

**if** (curr->left->label == curr->right->label)

{

curr->label = curr->left->label + 1;

}

**else**

{

curr->label = max(curr->left->label, curr->right->label);

}

**return** curr;

}

**void** swap\_registers()

{

**int** temp1, temp2;

temp1 = rstack.top();

rstack.pop();

temp2 = rstack.top();

rstack.pop();

rstack.push(temp1);

rstack.push(temp2);

}

**void** gen\_code(dag\_node \*curr, **bool** left\_child)

{

**if** (curr->left == NULL && curr->right == NULL)

{

**if** (left\_child)

{

printf("MOV %c R%d\n", curr->value, rstack.top());

}

}

**else**

{

**int** left\_label = curr->left->label;

**int** right\_label = curr->right->label;

**if** (right\_label == 0)

{

gen\_code(curr->left, 1);

printf("%c %c R%d\n", curr->value, curr->right->value, rstack.top());

}

**else** **if** (right\_label > left\_label && left\_label < rstack.size())

{

swap\_registers();

gen\_code(curr->right, 0);

**int** R = rstack.top();

rstack.pop();

gen\_code(curr->left, 1);

printf("%c R%d R%d\n", curr->value, R, rstack.top());

rstack.push(R);

swap\_registers();

}

**else** **if** (left\_label >= right\_label && right\_label < rstack.size())

{

gen\_code(curr->left, 1);

**int** R = rstack.top();

rstack.pop();

gen\_code(curr->right, 0);

printf("%c R%d R%d\n", curr->value, rstack.top(), R);

rstack.push(R);

}

**else** **if** (left\_label >= right\_label && left\_label > rstack.size() && right\_label > rstack.size())

{

gen\_code(curr->right, 0);

**int** T = tstack.top();

printf("MOV R%d T%d", rstack.top(), T);

gen\_code(curr->left, 1);

tstack.push(T);

printf("%c T%d R%d", curr->value, T, rstack.top());

}

}

}

**int** main()

{

**char** lhs, op, op1, op2;

string s;

**int** i = 0;

**while** (getline(cin, s))

{

**if** (s.size() <= 4)

{

tac[i].lhs = s[0];

tac[i].op1 = s[2];

tac[i].op = s[1];

tac[i].op2 = ' ';

}

**else**

{

tac[i].lhs = s[0];

tac[i].op1 = s[2];

tac[i].op = s[3];

tac[i].op2 = s[4];

}

i++;

}

n = i;

root\_list \*root = NULL;

root = create\_dag(root, 0);

root\_list \*start = root;

vector<**bool**> visited(index\_global, 0);

**while** (start)

{

start->root = assign\_labels(start->root, 1, visited);

start = start->next;

}

fill(visited.begin(), visited.end(), 0);

start = root;

**while** (start)

{

inorder(start->root, visited);

start = start->next;

}

rstack.push(0);

rstack.push(1);

**for** (**int** i = 0; i < 10; i++)

{

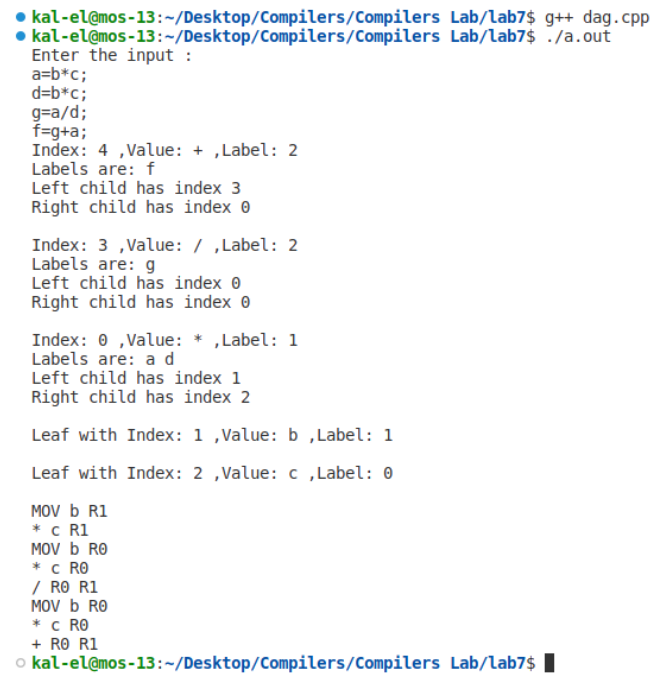
tstack.push(i);

}

gen\_code(root->root, 1);

}

### **Output:**



## **Result:**

DAG construction and Simple Code Generation was performed successfully.