System Software (CS306)

Assignment - 1

U20CS135

Write a program to convert Regular Expression to DFA.

```
#include<bits/stdc++.h>
#define vec(i) vector<int>  //Vector macro
#define pb push_back

using namespace std;

int gpos,fl_pos;        // for finding  position and follow position index
vector<int> follow_pos[30],state[30];   // follow_pos store the follow position and state store total no of Dstates
map<int,char> alpha_int;
set<char> in_alpha;

vec(int) operator +(vec(int) a,vec(int) b)  // operator Overloading for vector Additions
{
    a.insert(a.end(),b.begin(),b.end());
    sort(a.begin(),a.end());        // required for making unique element
    a.erase(unique(a.begin(),a.end()),a.end()); //for finding unique element within vector
    return a;
}

struct tree         // Structure for Treenode
{
    char alpha;
    int pos;
    bool nullable;
    vec(int) fpos,lpos;
    tree *left,*right;
};

bool is_op(char ch)
{
    if(ch == '|' || ch == '/'  || ch == '*' || ch == '.')
        return true;
    return false;
}

tree *create(char ch,int pos)       // Creating Node Memory and initialization
{
    tree *node = new tree;
    node->alpha = ch;
```

```
    node->pos = pos;
    node->left = node->right = NULL;
    node->lpos.clear();node->fpos.clear();
    return node;
}

void vec_print(vec(int) v)
{
    for(int i = 0 ; i < v.size() ; i++)
        cout<<v[i]<<" ";
}

void postfix(tree *root)        // For Traversing The tree
{
    if(root)
    {
        string s("          ");
        postfix(root->left);
        postfix(root->right);
        cout<<root->alpha<<s<<root->pos<<s<<root->nullable<<s<<"{ ";vec_print(root->fpos);cout<<"
}"<<s<<s<<"{ ";vec_print(root->lpos);cout<<" }"<<endl;
    }
}

void dfa(tree *root,string input)   // Finding DFA
{
    int num_state = 1,cur_state = 1;
    char ch = 'A';
    vec(int) temp;
    map< vector<int> , char> out_state; // Out_state used for removing Redundant States
    map< vector<int> , map< char , vector<int> > >  re_dfa;  //for Storing The final DFA state
    state[num_state++] = root->fpos;     position
    out_state[root->fpos] = ch++ ;
    while(1)
    {
        for(int i = 0 ; i < input.size() ; i++)
        {
            for(int j = 0 ; j < state[cur_state].size() ; j++)
            {
                if(alpha_int[state[cur_state][j]] == input[i])
                    temp = temp + follow_pos[state[cur_state][j] ];
                if(out_state[temp] == 0 && temp.size() > 0)
                {
                    out_state[temp] = ch++;
                    state[num_state++] = temp;
                }
            }
            re_dfa[state[cur_state]][input[i]] = temp;
            temp.clear();
        }
        if(cur_state == num_state - 1)
```

```
            break;
        cur_state++;
    }
    cout<<"\n\nThe Final State Table :\n\n";
    for(auto an : re_dfa)
    {
        cout<<"{ ";
        for(auto jn : an.first)
            cout<<jn<<" ";
        cout<<" } ";
        for(auto jn : an.second)
        {
            cout<<" at : "<<jn.first<<" { ";
            for(auto kn:jn.second)
                cout<<kn<<" ";
            cout<<" } ";
        }
        cout<<endl;
    }
}

int main()
{
    tree *temp;
    stack<tree *> s;
    string str,sp("     "),input;
    cout<<"\nEnter the Postfix Expression = ";
    cin>>str;
    for(int i = 0 ; i < str.size() ; i++)
    {
        if(!is_op(str[i]))
        {
            gpos++;
            if(str[i] != '#')
            {
                fl_pos++;
                alpha_int[fl_pos] = str[i];
                in_alpha.insert(str[i]);
            }
            temp = create(str[i],gpos);
            temp->nullable = false;
            temp->fpos.pb(gpos);temp->lpos.pb(gpos);
        }
        else if(str[i] == '*')
        {
            temp = create(str[i],0);
            temp->left = s.top() , s.pop();
            temp->nullable = true;
            temp->fpos = temp->left->fpos;
            temp->lpos = temp->left->lpos;
            for(int i = 0 ; i < temp->lpos.size() ; i++)
```

```
                follow_pos[temp->lpos[i]] = follow_pos[temp->lpos[i]] + temp->fpos;

        }
        else if(str[i] == '.')
        {
            temp = create(str[i],0);
            temp->right = s.top() , s.pop();
            temp->left = s.top() , s.pop();
            temp->nullable = temp->right->nullable && temp->left->nullable;
            if(temp->left->nullable)
                temp->fpos = temp->right->fpos + temp->left->fpos;
            else
                temp->fpos = temp->left->fpos;
            if(temp->right->nullable)
                temp->lpos = temp->right->lpos + temp->left->lpos;
            else
                temp->lpos = temp->right->lpos;
            for(int i = 0 ; i < temp->left->lpos.size() ; i++)
                follow_pos[temp->left->lpos[i]] = follow_pos[temp->left->lpos[i]] + temp->right-
>fpos;

        }
        else
        {
            temp = create(str[i],0);
            temp->right = s.top() , s.pop();
            temp->left = s.top() , s.pop();
            temp->nullable = temp->right->nullable && temp->left->nullable;
            temp->fpos = temp->right->fpos + temp->left->fpos;
            temp->lpos = temp->right->lpos + temp->left->lpos;
        }
        s.push(temp);
    }
    for(auto temp:in_alpha)
        input.pb(temp);
    cout<<"\n\nNODE"<<sp<<"Position"<<"       "<<"Nullable"<<"        "<<"First
position"<<"           "<<"Last position"<<endl;
    postfix(temp);
    cout<<"\n\nFollow Position"<<endl;
    for(int i = 1 ; i <= fl_pos ; i++)  // Display of Follow Position
    {
        cout<<i<<sp<<alpha_int[i]<<sp<<"{ ";
        for(int j = 0; j < follow_pos[i].size() ; j++)
        {
            cout<<follow_pos[i][j]<<" ";
        }
        cout<<" }\n";
    }
    dfa(temp,input);
    return 0;
}
```

SUBMITTED BY:

U20CS135

Shivam Mishra