**SLR PARSER**

**BY-**

**AAKRITI MEHROTRA**

**22BCE1954**

**CODE:**

#include <iostream>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

char terminals[100] = {};

int no\_t;

char non\_terminals[100] = {};

int no\_nt;

char goto\_table[100][100];

char reduce[20][20];

char follow[20][20];

char fo\_co[20][20];

char first[20][20];

struct state

{

int prod\_count;

char prod[100][100] = {{}};

};

void add\_dots(struct state \*I)

{

for (int i = 0; i < I->prod\_count; i++)

{

for (int j = 99; j > 3; j--)

I->prod[i][j] = I->prod[i][j - 1];

I->prod[i][3] = '.';

}

}

void augument(struct state \*S, struct state \*I)

{

if (I->prod[0][0] == 'S')

strcpy(S->prod[0], "Z->.S");

else

{

strcpy(S->prod[0], "S->.");

S->prod[0][4] = I->prod[0][0];

}

S->prod\_count++;

}

void get\_prods(struct state \*I)

{

cout << "Enter the number of productions:\n";

cin >> I->prod\_count;

cout << "Enter the number of non terminals:" << endl;

cin >> no\_nt;

cout << "Enter the non terminals one by one:" << endl;

for (int i = 0; i < no\_nt; i++)

cin >> non\_terminals[i];

cout << "Enter the number of terminals:" << endl;

cin >> no\_t;

cout << "Enter the terminals (single lettered) one by one:" << endl;

for (int i = 0; i < no\_t; i++)

cin >> terminals[i];

cout << "Enter the productions one by one in form (S->ABc):\n";

for (int i = 0; i < I->prod\_count; i++)

{

cin >> I->prod[i];

}

}

bool is\_non\_terminal(char a)

{

if (a >= 'A' && a <= 'Z')

return true;

else

return false;

}

bool in\_state(struct state \*I, char \*a)

{

for (int i = 0; i < I->prod\_count; i++)

{

if (!strcmp(I->prod[i], a))

return true;

}

return false;

}

char char\_after\_dot(char a[100])

{

char b;

for (int i = 0; i < strlen(a); i++)

if (a[i] == '.')

{

b = a[i + 1];

return b;

}

}

string move\_dot(const string &input)

{

string result = input;

size\_t dot\_pos = result.find('.');

if (dot\_pos != string::npos && dot\_pos < result.length() - 1)

{

swap(result[dot\_pos], result[dot\_pos + 1]);

}

return result;

}

bool same\_state(struct state \*I0, struct state \*I)

{

if (I0->prod\_count != I->prod\_count)

return false;

for (int i = 0; i < I0->prod\_count; i++)

{

int flag = 0;

for (int j = 0; j < I->prod\_count; j++)

if (strcmp(I0->prod[i], I->prod[j]) == 0)

flag = 1;

if (flag == 0)

return false;

}

return true;

}

void closure(struct state \*I, struct state \*I0)

{

bool changed;

do

{

changed = false;

for (int i = 0; i < I0->prod\_count; i++)

{

char a = char\_after\_dot(I0->prod[i]);

if (is\_non\_terminal(a))

{

for (int j = 0; j < I->prod\_count; j++)

{

if (I->prod[j][0] == a)

{

if (!in\_state(I0, I->prod[j]))

{

if (I0->prod\_count < 100)

{ // Check array bounds

strcpy(I0->prod[I0->prod\_count], I->prod[j]);

I0->prod\_count++;

changed = true;

}

else

{

cerr << "Error: Too many productions in closure" << endl;

return;

}

}

}

}

}

}

} while (changed);

}

void goto\_state(struct state \*I, struct state \*S, char a)

{

int time = 1;

for (int i = 0; i < I->prod\_count; i++)

{

if (char\_after\_dot(I->prod[i]) == a)

{

if (time == 1)

{

time++;

}

strcpy(S->prod[S->prod\_count], move\_dot(I->prod[i]).c\_str());

S->prod\_count++;

}

}

}

void print\_prods(struct state \*I)

{

for (int i = 0; i < I->prod\_count; i++)

printf("%s\n", I->prod[i]);

cout << endl;

}

bool in\_array(char a[20], char b)

{

for (int i = 0; i < strlen(a); i++)

if (a[i] == b)

return true;

return false;

}

string chars\_after\_dots(struct state \*I)

{

string result;

for (int i = 0; i < I->prod\_count; i++)

{

char c = char\_after\_dot(I->prod[i]);

if (result.find(c) == string::npos)

{

result += c;

}

}

return result;

}

void cleanup\_prods(struct state \*I)

{

char a[100] = {};

for (int i = 0; i < I->prod\_count; i++)

strcpy(I->prod[i], a);

I->prod\_count = 0;

}

int return\_index(char a)

{

for (int i = 0; i < no\_t; i++)

if (terminals[i] == a)

return i;

for (int i = 0; i < no\_nt; i++)

if (non\_terminals[i] == a)

return no\_t + i;

}

void print\_shift\_table(int state\_count)

{

cout << endl

<< "\*\*\*\*\*\*\*\*Shift Actions\*\*\*\*\*\*\*\*\*" << endl

<< endl;

cout << "\t";

for (int i = 0; i < no\_t; i++)

cout << terminals[i] << "\t";

for (int i = 0; i < no\_nt; i++)

cout << non\_terminals[i] << "\t";

cout << endl;

for (int i = 0; i < state\_count; i++)

{

int arr[no\_nt + no\_t] = {-1};

for (int j = 0; j < state\_count; j++)

{

if (goto\_table[i][j] != '~')

{

arr[return\_index(goto\_table[i][j])] = j;

}

}

cout << "I" << i << "\t";

for (int j = 0; j < no\_nt + no\_t; j++)

{

if (i == 1 && j == no\_t - 1)

cout << "ACC" << "\t";

if (arr[j] == -1 || arr[j] == 0)

cout << "\t";

else

{

if (j < no\_t)

cout << "S" << arr[j] << "\t";

else

cout << arr[j] << "\t";

}

}

cout << "\n";

}

}

int get\_index(char c, char \*a)

{

for (int i = 0; i < strlen(a); i++)

if (a[i] == c)

return i;

}

void add\_dot\_at\_end(struct state \*I)

{

for (int i = 0; i < I->prod\_count; i++)

{

strcat(I->prod[i], ".");

}

}

void add\_to\_first(int n, char b)

{

for (int i = 0; i < strlen(first[n]); i++)

if (first[n][i] == b)

return;

first[n][strlen(first[n])] = b;

}

void add\_to\_first(int m, int n)

{

for (int i = 0; i < strlen(first[n]); i++)

{

int flag = 0;

for (int j = 0; j < strlen(first[m]); j++)

{

if (first[n][i] == first[m][j])

flag = 1;

}

if (flag == 0)

add\_to\_first(m, first[n][i]);

}

}

void add\_to\_follow(int n, char b)

{

for (int i = 0; i < strlen(follow[n]); i++)

if (follow[n][i] == b)

return;

follow[n][strlen(follow[n])] = b;

}

void add\_to\_follow(int m, int n)

{

for (int i = 0; i < strlen(follow[n]); i++)

{

int flag = 0;

for (int j = 0; j < strlen(follow[m]); j++)

{

if (follow[n][i] == follow[m][j])

flag = 1;

}

if (flag == 0)

add\_to\_follow(m, follow[n][i]);

}

}

void add\_to\_follow\_first(int m, int n)

{

for (int i = 0; i < strlen(first[n]); i++)

{

int flag = 0;

for (int j = 0; j < strlen(follow[m]); j++)

{

if (first[n][i] == follow[m][j])

flag = 1;

}

if (flag == 0)

add\_to\_follow(m, first[n][i]);

}

}

void find\_first(struct state \*I)

{

for (int i = 0; i < no\_nt; i++)

{

for (int j = 0; j < I->prod\_count; j++)

{

if (I->prod[j][0] == non\_terminals[i])

{

if (!is\_non\_terminal(I->prod[j][3]))

{

add\_to\_first(i, I->prod[j][3]);

}

}

}

}

}

void find\_follow(struct state \*I)

{

for (int i = 0; i < no\_nt; i++)

{

for (int j = 0; j < I->prod\_count; j++)

{

for (int k = 3; k < strlen(I->prod[j]); k++)

{

if (I->prod[j][k] == non\_terminals[i])

{

if (I->prod[j][k + 1] != '\0')

{

if (!is\_non\_terminal(I->prod[j][k + 1]))

{

add\_to\_follow(i, I->prod[j][k + 1]);

}

}

}

}

}

}

}

int get\_index(int \*arr, int n)

{

for (int i = 0; i < no\_t; i++)

{

if (arr[i] == n)

return i;

}

return -1;

}

void print\_reduce\_table(int state\_count, int \*no\_re, struct state \*temp1)

{

cout << "\*\*\*\*\*\*\*\*\*\*Reduce actions\*\*\*\*\*\*\*\*\*\*" << endl

<< endl;

cout << "\t";

int arr[temp1->prod\_count][no\_t] = {-1};

for (int i = 0; i < no\_t; i++)

{

cout << terminals[i] << "\t";

}

cout << endl;

for (int i = 0; i < temp1->prod\_count; i++)

{

int n = no\_re[i];

for (int j = 0; j < strlen(follow[return\_index(temp1->prod[i][0]) - no\_t]); j++)

{

for (int k = 0; k < no\_t; k++)

{

if (follow[return\_index(temp1->prod[i][0]) - no\_t][j] == terminals[k])

arr[i][k] = i + 1;

}

}

cout << "I" << n << "\t";

for (int j = 0; j < no\_t; j++)

{

if (arr[i][j] != -1 && arr[i][j] != 0 && arr[i][j] < state\_count)

cout << "R" << arr[i][j] << "\t";

else

cout << "\t";

}

cout << endl;

}

}

int main()

{

struct state init;

struct state temp;

struct state temp1;

int state\_count = 1;

get\_prods(&init);

temp = init;

temp1 = temp;

add\_dots(&init);

for (int i = 0; i < 100; i++)

for (int j = 0; j < 100; j++)

goto\_table[i][j] = '~';

struct state I[50];

augument(&I[0], &init);

closure(&init, &I[0]);

cout << "\nI0:\n";

print\_prods(&I[0]);

char characters[20] = {};

for (int i = 0; i < state\_count; i++)

{

char characters[20] = {};

for (int z = 0; z < I[i].prod\_count; z++)

if (!in\_array(characters, char\_after\_dot(I[i].prod[z])))

characters[strlen(characters)] = char\_after\_dot(I[i].prod[z]);

for (int j = 0; j < strlen(characters); j++)

{

goto\_state(&I[i], &I[state\_count], characters[j]);

closure(&init, &I[state\_count]);

int flag = 0;

for (int k = 0; k < state\_count - 1; k++)

{

if (same\_state(&I[k], &I[state\_count]))

{

cleanup\_prods(&I[state\_count]);

flag = 1;

cout << "I" << i << " on reading the symbol " << characters[j] << " goes to I" << k << ".\n";

goto\_table[i][k] = characters[j];

;

break;

}

}

if (flag == 0)

{

state\_count++;

cout << "I" << i << " on reading the symbol " << characters[j] << " goes to I" << state\_count - 1 << ":\n";

goto\_table[i][state\_count - 1] = characters[j];

print\_prods(&I[state\_count - 1]);

}

}

}

int no\_re[temp.prod\_count] = {-1};

terminals[no\_t] = '$';

no\_t++;

add\_dot\_at\_end(&temp1);

for (int i = 0; i < state\_count; i++)

{

for (int j = 0; j < I[i].prod\_count; j++)

for (int k = 0; k < temp1.prod\_count; k++)

if (in\_state(&I[i], temp1.prod[k]))

no\_re[k] = i;

}

find\_first(&temp);

for (int l = 0; l < no\_nt; l++)

{

for (int i = 0; i < temp.prod\_count; i++)

{

if (is\_non\_terminal(temp.prod[i][3]))

{

add\_to\_first(return\_index(temp.prod[i][0]) - no\_t, return\_index(temp.prod[i][3]) - no\_t);

}

}

}

find\_follow(&temp);

add\_to\_follow(0, '$');

for (int l = 0; l < no\_nt; l++)

{

for (int i = 0; i < temp.prod\_count; i++)

{

for (int k = 3; k < strlen(temp.prod[i]); k++)

{

if (temp.prod[i][k] == non\_terminals[l])

{

if (is\_non\_terminal(temp.prod[i][k + 1]))

{

add\_to\_follow\_first(l, return\_index(temp.prod[i][k + 1]) - no\_t);

}

if (temp.prod[i][k + 1] == '\0')

add\_to\_follow(l, return\_index(temp.prod[i][0]) - no\_t);

}

}

}

}

print\_shift\_table(state\_count);

cout << endl

<< endl;

print\_reduce\_table(state\_count, &no\_re[0], &temp1);

return 0;

}

**OUTPUT:**

