|  |
| --- |
| **Ex.No: 7 DATE: 27-10-23**  **IMPLEMENTATION OF PREDICTIVE PARSER AND SLR PARSER** |

1)Predictive parser

#include<stdio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

void followfirst(char , int , int);

void findfirst(char , int , int);

void follow(char c);

int count,n=0;

char calc\_first[10][100];

char calc\_follow[10][100];

int m=0;

char production[10][10], first[10];

char f[10];

int k;

char ck;

int e;

int main(int argc,char \*\*argv)

{

int jm=0;

int km=0;

int i,choice;

char c,ch;

printf("How many productions ? :");

scanf("%d",&count);

printf("\nEnter %d productions in form A=B where A and B are grammar symbols :\n\n",count);

for(i=0;i<count;i++)

{

scanf("%s%c",production[i],&ch);

}

int kay;

char done[count];

int ptr = -1;

for(k=0;k<count;k++){

for(kay=0;kay<100;kay++){

calc\_first[k][kay] = '!';

}

}

int point1 = 0,point2,xxx;

for(k=0;k<count;k++)

{

c=production[k][0];

point2 = 0;

xxx = 0;

for(kay = 0; kay <= ptr; kay++)

if(c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

findfirst(c,0,0);

ptr+=1;

done[ptr] = c;

printf("\n First(%c)= { ",c);

calc\_first[point1][point2++] = c;

for(i=0+jm;i<n;i++){

int lark = 0,chk = 0;

for(lark=0;lark<point2;lark++){

if (first[i] == calc\_first[point1][lark]){

chk = 1;

break;

}

}

if(chk == 0){

printf("%c, ",first[i]);

calc\_first[point1][point2++] = first[i];

}

}

printf("}\n");

jm=n;

point1++;

}

printf("\n");

printf("-----------------------------------------------\n\n");

char donee[count];

ptr = -1;

for(k=0;k<count;k++){

for(kay=0;kay<100;kay++){

calc\_follow[k][kay] = '!';

}

}

point1 = 0;

int land = 0;

for(e=0;e<count;e++)

{

ck=production[e][0];

point2 = 0;

xxx = 0;

for(kay = 0; kay <= ptr; kay++)

if(ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

follow(ck);

ptr+=1;

donee[ptr] = ck;

printf(" Follow(%c) = { ",ck);

calc\_follow[point1][point2++] = ck;

for(i=0+km;i<m;i++){

int lark = 0,chk = 0;

for(lark=0;lark<point2;lark++){

if (f[i] == calc\_follow[point1][lark]){

chk = 1;

break;

}

}

if(chk == 0){

printf("%c, ",f[i]);

calc\_follow[point1][point2++] = f[i];

}

}

printf(" }\n\n");

km=m;

point1++;

}

char ter[10];

for(k=0;k<10;k++){

ter[k] = '!';

}

int ap,vp,sid = 0;

for(k=0;k<count;k++){

for(kay=0;kay<count;kay++){

if(!isupper(production[k][kay]) && production[k][kay]!= '#' && production[k][kay] != '=' && production[k][kay] != '\0'){

vp = 0;

for(ap = 0;ap < sid; ap++){

if(production[k][kay] == ter[ap]){

vp = 1;

break;

}

}

if(vp == 0){

ter[sid] = production[k][kay];

sid ++;

}

}

}

}

ter[sid] = '$';

sid++;

printf("\n\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer :-");

printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");

printf("\n\t\t\t=====================================================================================================================\n");

printf("\t\t\t\t|\t");

for(ap = 0;ap < sid; ap++){

printf("%c\t\t",ter[ap]);

}

printf("\n\t\t\t=====================================================================================================================\n");

char first\_prod[count][sid];

for(ap=0;ap<count;ap++){

int destiny = 0;

k = 2;

int ct = 0;

char tem[100];

while(production[ap][k] != '\0'){

if(!isupper(production[ap][k])){

tem[ct++] = production[ap][k];

tem[ct++] = '\_';

tem[ct++] = '\0';

k++;

break;

}

else{

int zap=0;

int tuna = 0;

for(zap=0;zap<count;zap++){

if(calc\_first[zap][0] == production[ap][k]){

for(tuna=1;tuna<100;tuna++){

if(calc\_first[zap][tuna] != '!'){

tem[ct++] = calc\_first[zap][tuna];

}

else

break;

}

break;

}

}

tem[ct++] = '\_';

}

k++;

}

int zap = 0,tuna;

for(tuna = 0;tuna<ct;tuna++){

if(tem[tuna] == '#'){

zap = 1;

}

else if(tem[tuna] == '\_'){

if(zap == 1){

zap = 0;

}

else

break;

}

else{

first\_prod[ap][destiny++] = tem[tuna];

}

}

}

char table[land][sid+1];

ptr = -1;

for(ap = 0; ap < land ; ap++){

for(kay = 0; kay < (sid + 1) ; kay++){

table[ap][kay] = '!';

}

}

for(ap = 0; ap < count ; ap++){

ck = production[ap][0];

xxx = 0;

for(kay = 0; kay <= ptr; kay++)

if(ck == table[kay][0])

xxx = 1;

if (xxx == 1)

continue;

else{

ptr = ptr + 1;

table[ptr][0] = ck;

}

}

for(ap = 0; ap < count ; ap++){

int tuna = 0;

while(first\_prod[ap][tuna] != '\0'){

int to,ni=0;

for(to=0;to<sid;to++){

if(first\_prod[ap][tuna] == ter[to]){

ni = 1;

}

}

if(ni == 1){

char xz = production[ap][0];

int cz=0;

while(table[cz][0] != xz){

cz = cz + 1;

}

int vz=0;

while(ter[vz] != first\_prod[ap][tuna]){

vz = vz + 1;

}

table[cz][vz+1] = (char)(ap + 65);

}

tuna++;

}

}

for(k=0;k<sid;k++){

for(kay=0;kay<100;kay++){

if(calc\_first[k][kay] == '!'){

break;

}

else if(calc\_first[k][kay] == '#'){

int fz = 1;

while(calc\_follow[k][fz] != '!'){

char xz = production[k][0];

int cz=0;

while(table[cz][0] != xz){

cz = cz + 1;

}

int vz=0;

while(ter[vz] != calc\_follow[k][fz]){

vz = vz + 1;

}

table[k][vz+1] = '#';

fz++;

}

break;

}

}

}

for(ap = 0; ap < land ; ap++){

printf("\t\t\t %c\t|\t",table[ap][0]);

for(kay = 1; kay < (sid + 1) ; kay++){

if(table[ap][kay] == '!')

printf("\t\t");

else if(table[ap][kay] == '#')

printf("%c=#\t\t",table[ap][0]);

else{

int mum = (int)(table[ap][kay]);

mum -= 65;

printf("%s\t\t",production[mum]);

}

}

printf("\n");

printf("\t\t\t---------------------------------------------------------------------------------------------------------------------");

printf("\n");

}

int j;

printf("\n\nPlease enter the desired INPUT STRING = ");

char input[100];

scanf("%s%c",input,&ch);

printf("\n\t\t\t\t\t===========================================================================\n");

printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");

printf("\n\t\t\t\t\t===========================================================================\n");

int i\_ptr = 0,s\_ptr = 1;

char stack[100];

stack[0] = '$';

stack[1] = table[0][0];

while(s\_ptr != -1){

printf("\t\t\t\t\t\t");

int vamp = 0;

for(vamp=0;vamp<=s\_ptr;vamp++){

printf("%c",stack[vamp]);

}

printf("\t\t\t");

vamp = i\_ptr;

while(input[vamp] != '\0'){

printf("%c",input[vamp]);

vamp++;

}

printf("\t\t\t");

char her = input[i\_ptr];

char him = stack[s\_ptr];

s\_ptr--;

if(!isupper(him)){

if(her == him){

i\_ptr++;

printf("POP ACTION\n");

}

else{

printf("\nString Not Accepted by LL(1) Parser !!\n");

exit(0);

}

}

else{

for(i=0;i<sid;i++){

if(ter[i] == her)

break;

}

char produ[100];

for(j=0;j<land;j++){

if(him == table[j][0]){

if (table[j][i+1] == '#'){

printf("%c=#\n",table[j][0]);

produ[0] = '#';

produ[1] = '\0';

}

else if(table[j][i+1] != '!'){

int mum = (int)(table[j][i+1]);

mum -= 65;

strcpy(produ,production[mum]);

printf("%s\n",produ);

}

else{

printf("\nString Not Accepted by LL(1) Parser !!\n");

exit(0);

}

}

}

int le = strlen(produ);

le = le - 1;

if(le == 0){

continue;

}

for(j=le;j>=2;j--){

s\_ptr++;

stack[s\_ptr] = produ[j];

}

}

}

printf("\n\t\t\t=======================================================================================================================\n");

if (input[i\_ptr] == '\0'){

printf("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");

}

else

printf("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN REJECTED !!\n");

printf("\t\t\t=======================================================================================================================\n");

}

void follow(char c)

{

int i ,j;

if(production[0][0]==c){

f[m++]='$';

}

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

{

for(j=2;j<10;j++)

{

if(production[i][j]==c)

{

if(production[i][j+1]!='\0'){

followfirst(production[i][j+1],i,(j+2));

}

if(production[i][j+1]=='\0'&&c!=production[i][0]){

follow(production[i][0]);

}

}

}

}

}

void findfirst(char c ,int q1 , int q2)

{

int j;

if(!(isupper(c))){

first[n++]=c;

}

for(j=0;j<count;j++)

{

if(production[j][0]==c)

{

if(production[j][2]=='#'){

if(production[q1][q2] == '\0')

first[n++]='#';

else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))

{

findfirst(production[q1][q2], q1, (q2+1));

}

else

first[n++]='#';

}

else if(!isupper(production[j][2])){

first[n++]=production[j][2];

}

else {

findfirst(production[j][2], j, 3);

}

}

}

}

void followfirst(char c, int c1 , int c2)

{

int k;

if(!(isupper(c)))

f[m++]=c;

else{

int i=0,j=1;

for(i=0;i<count;i++)

{

if(calc\_first[i][0] == c)

break;

}

while(calc\_first[i][j] != '!')

{

if(calc\_first[i][j] != '#'){

f[m++] = calc\_first[i][j];

}

else{

if(production[c1][c2] == '\0'){

follow(production[c1][0]);

}

else{

followfirst(production[c1][c2],c1,c2+1);

}

}

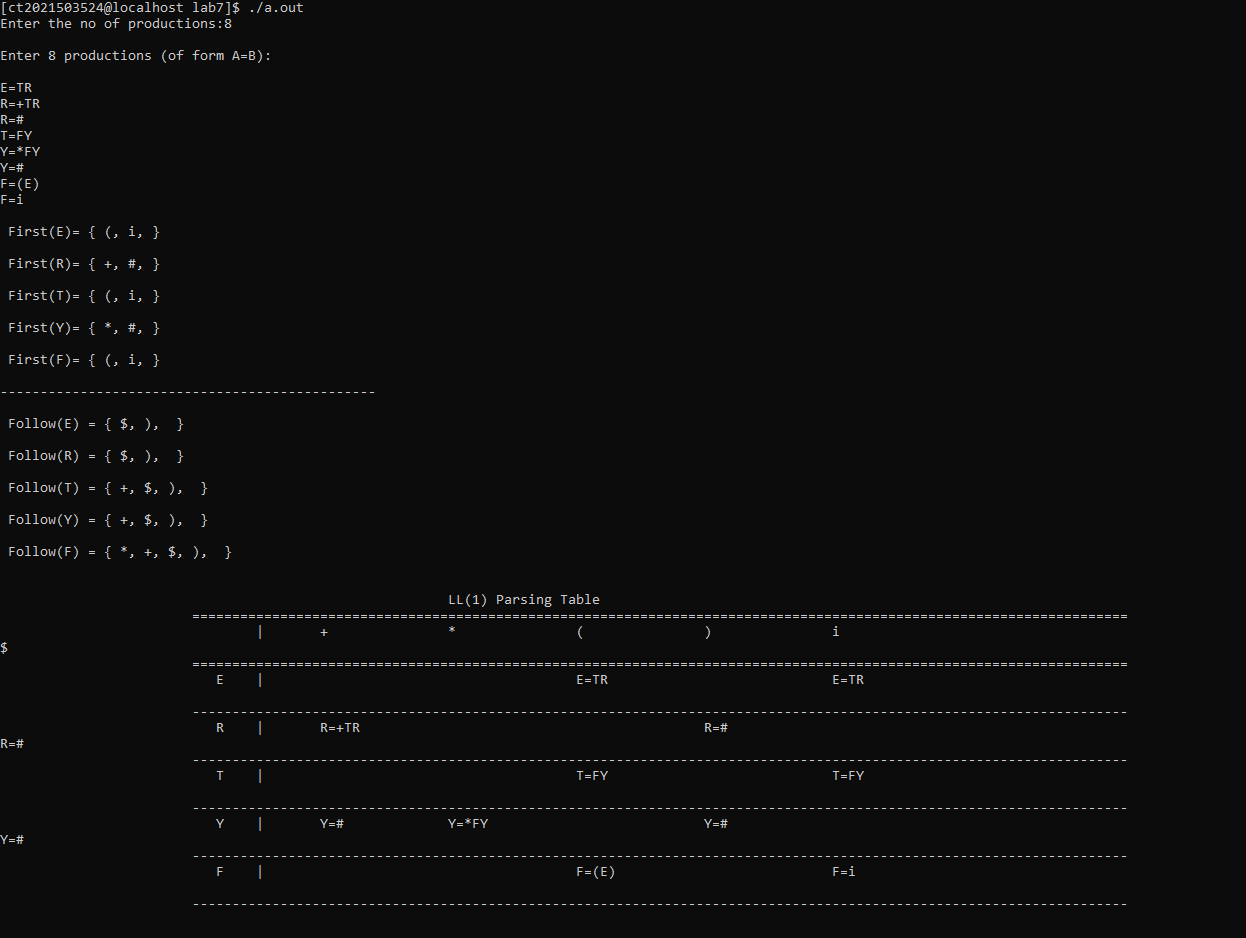
j++;

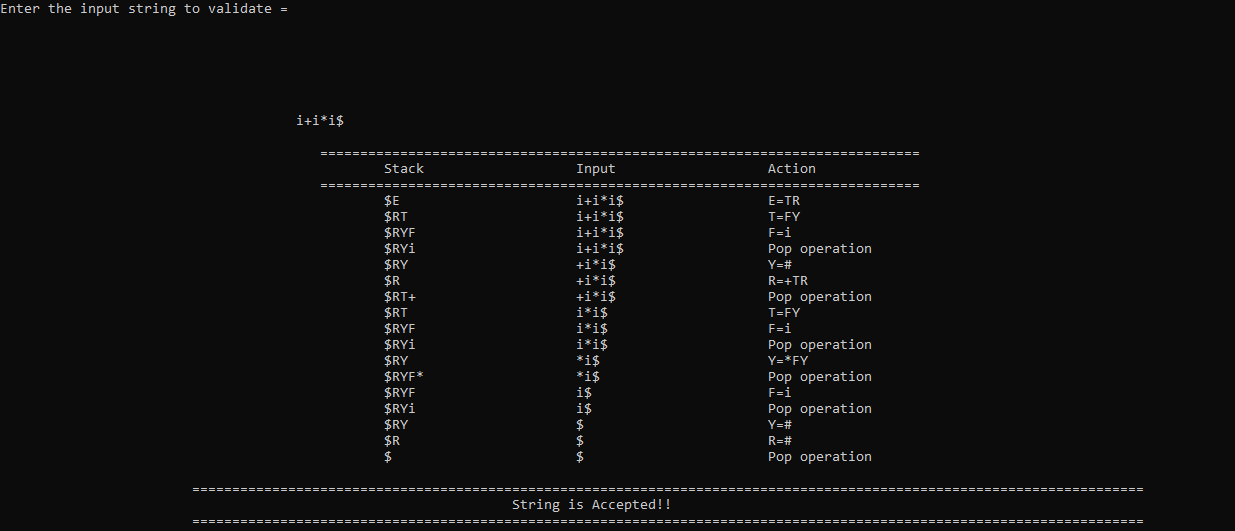
}

}

}

Output:





2) SLR Parser  
  
#include <bits/stdc++.h>

#define error(x) cerr << #x << " = " << x << '\n'

using namespace std; // Corrected "name space" to "namespace"

set<char> ss;

map<char, vector<vector<char>> > mp; // Added space after "<" and ">"

map<int, map<char, set<pair<deque<char>, deque<char>>> > > f; // Added space after "<" and ">"

map<int, vector<pair<int, char>> > g; // Added space after "<"

int num = -1;

bool dfs(char i, char org, char last, map<char, vector<vector<char>>>& mp) {

bool rtake = false;

for (auto r : mp[i]) {

bool take = true;

for (auto s : r) {

if (s == i)

break;

if (!take)

break;

if (!(s >= 'A' && s <= 'Z') && s != 'e') {

ss.insert(s);

break;

} else if (s == 'e') {

if (org == i || i == last)

ss.insert(s);

rtake = true;

break;

} else {

take = dfs(s, org, r[r.size() - 1], mp);

rtake |= take;

}

}

}

return rtake;

}

void dfs2(char c, char way, int last, pair<deque<char>, deque<char>> curr) {

map<char, set<pair<deque<char>, deque<char>>> > mp2;

int rep = -2;

if (last != -1) {

for (auto q : g[last]) {

if (q.second == way) {

rep = q.first;

mp2 = f[q.first];

}

}

}

mp2[c].insert(curr);

int count = 10; // Added missing int declaration for "count"

while (count--) {

for (auto q : mp2) {

for (auto r : q.second) {

if (!r.second.empty()) {

if (r.second.front() >= 'A' && r.second.front() <= 'Z') {

for (auto s : mp[r.second.front()]) {

deque<char> st, emp;

for (auto t : s)

st.push\_back(t);

mp2[r.second.front()].insert({emp, st});

}

}

}

}

}

}

for (auto q : f) {

if (q.second == mp2) {

g[last].push\_back({q.first, way});

return;

}

}

if (rep == -2) {

f[++num] = mp2;

if (last != -1)

g[last].push\_back({num, way});

} else {

f[rep] = mp2;

}

int cc = num; // Added missing int declaration for "cc"

for (auto q : mp2) {

for (auto r : q.second) {

if (!r.second.empty()) {

r.first.push\_back(r.second.front());

r.second.pop\_front();

dfs2(q.first, r.first.back(), cc, r);

}

}

}

}

int main()

{

int i, j;

ifstream fin("grammar.txt");

string num;

vector<int> fs;

vector<vector<int>> a;

char start;

bool flag = 0;

cout <<"Grammar: "<<'\n';

while (getline(fin, num))

{

if (flag == 0)

start = num[0], flag = 1;

cout <<num<<'\n';

vector<char> temp;

char s = num[0];

for (i = 3; i<num.size(); i++)

{

if (num[i] == '|')

{

mp[s].push\_back(temp);

temp.clear();

}

else

temp.push\_back(num[i]);

}

mp[s].push\_back(temp);

}

map<char, set<char>> fmp;

for (auto q : mp)

{

ss.clear();

dfs(q.first, q.first, q.first, mp);

for (auto g : ss)

fmp[q.first].insert(g);

}

cout <<'\n';

cout <<"FIRST: "<<'\n';

for (auto q : fmp)

{

string ans = "";

ans += q.first;

ans += " = {";

for (char r : q.second)

{

ans += r;

ans += ',';

}

ans.pop\_back();

ans += "}";

cout <<ans<<'\n';

}

map<char, set<char>> gmp;

gmp[start].insert('$');

int count = 10;

while (count--)

{

for (auto q : mp)

{

for (auto r : q.second)

{

for (i = 0; i<r.size() - 1; i++)

{

if (r[i] >= 'A'&&r[i] <= 'Z')

{

if (!(r[i + 1] >= 'A'&&r[i + 1] <= 'Z'))

{

gmp[r[i]].insert(r[i + 1]);

}

else

{

char temp = r[i + 1];

int j = i + 1;

while (temp>= 'A'&&temp<= 'Z')

{

if (\*fmp[temp].begin() == 'e')

{

for (auto g : fmp[temp])

{

if (g == 'e')

continue;

gmp[r[i]].insert(g);

}

j++;

if (j<r.size())

{

temp = r[j];

if (!(temp>= 'A'&&temp<= 'Z'))

{

gmp[r[i]].insert(temp);

break;

}

}

else

{

for (auto g : gmp[q.first])

gmp[r[i]].insert(g);

break;

}

}

else

{

for (auto g : fmp[temp])

{

gmp[r[i]].insert(g);

}

break;

}

}

}

}

}

if (r[r.size() - 1] >= 'A'&&r[r.size() - 1] <= 'Z')

{

for (auto g : gmp[q.first])

gmp[r[i]].insert(g);

}

}

}

}

cout <<'\n';

cout <<"FOLLOW: "<<'\n';

for (auto q : gmp)

{

string ans = "";

ans += q.first;

ans += " = {";

for (char r : q.second)

{

ans += r;

ans += ',';

}

ans.pop\_back();

ans += "}";

cout <<ans<<'\n';

}

string temp = "";

temp += '.';

temp += start;

deque<char> emp;

deque<char> st;

st.push\_back(start);

dfs2('!', 'k', -1, {emp, st});

cout <<"\nProductions: "<<'\n';

int cc = 1;

set<char> action, go;

map<pair<char, deque<char>>, int> pos;

for (auto q : mp)

{

go.insert(q.first);

for (auto r : q.second)

{

cout <<"r"<<cc<<": ";

string ans = "";

ans += q.first;

ans += "->";

deque<char>temp;

for (auto s : r)

ans += s, temp.push\_back(s);

pos[{q.first, temp}] = cc;

for (auto s : r)

{

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

go.insert(s);

else

action.insert(s);

}

cout <<ans<<'\n';

cc++;

}

}

cout <<"\nGraph: "<<'\n';

for (auto mp2 : f)

{

cout <<'\n';

cout <<"I";

cout <<mp2.first<<": \n";

for (auto q : mp2.second)

{

string ans = "";

ans += q.first;

ans += "->";

for (auto r : q.second)

{

for (auto t : r.first)

ans += t;

ans += '.';

for (auto t : r.second)

ans += t;

ans += '|';

}

ans.pop\_back();

for (auto tt : ans)

{

if (tt == '!')

cout <<start<<'\'';

else

cout <<tt;

}

cout <<'\n';

}

}

cout <<'\n';

cout <<"Edges: "<<'\n';

for (auto q : g)

{

for (auto r : q.second)

{

cout <<"I"<<q.first<<" -> "<<r.second<<" -> "

<<"I"<<r.first<<"\n";

}

}

action.insert('$');

cout <<"\nParsing Table:"<<'\n';

cout <<"St.\t\tAction & Goto"<<'\n';

int tot = f.size();

cout <<" \t";

for (auto q : action)

cout <<q<<'\t';

for (auto q : go)

cout <<q<<'\t';

cout <<'\n';

for (i = 0; i<tot; i++)

{

cout <<"I"<<i<<'\t';

for (auto q : action)

{

if (g.count(i))

{

int flag = 0;

for (auto r : g[i])

{

if (r.second == q)

{

flag = 1;

cout <<"S"<<r.first<<"\t";

break;

}

}

if (!flag)

cout <<"-"<<'\t';

}

else

{

int flag = 0;

for (auto r : f[i])

{

if (r.first == '!')

{

if (q == '$')

{

cout <<"AC\t";

flag = 1;

}

else

cout <<"-\t";

}

}

if (!flag)

{

for (auto r : f[i])

{

char ccc = r.first;

deque<char> chk = (\*r.second.begin()).first;

int cou = 1;

for (auto r : gmp[ccc])

{

if (q == r)

{

cout <<"r"<<pos[{ccc, chk}] <<"\t";

}

cou++;

}

}

}

}

}

for (auto q : go)

{

if (g.count(i))

{

int flag = 0;

for (auto r : g[i])

{

if (r.second == q)

{

flag = 1;

cout <<r.first<<"\t";

break;

}

}

if (!flag)

cout <<"-"<<'\t';

}

else

{

cout <<"-"<<'\t';

}

}

cout <<'\n';

}

return 0;

}

Output:

