**EXPERIMENT 1 (REGULAR EXPRESSION TO NFA)**

1. The NFA representing the empty string is:



2. If the regular expression is just a character, e.g., a, then the corresponding NFA is:



3. The union operator is represented by a choice of transitions from a node; thus a|b can be represented as:



4. Concatenation simply involves connecting one NFA to the other; eg. ab is:



5. The Kleene closure must allow for taking zero or more instances of the letter from the input; thus a\* looks like:



#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <ctype.h>

int ret[100];

static int pos = 0;

static int sc = 0;

void nfa(int st, int p, char \*s)

{

int i, sp, fs[15], fsc = 0;

sp = st;

pos = p;

sc = st;

while (\*s != NULL)

{

if (isalpha(\*s))

{

ret[pos++] = sp;

ret[pos++] = \*s;

ret[pos++] = ++sc;

}

if (\*s == '.')

{

sp = sc;

ret[pos++] = sc;

ret[pos++] = 238;

ret[pos++] = ++sc;

sp = sc;

}

if (\*s == '|')

{

sp = st;

fs[fsc++] = sc;

}

if (\*s == '\*')

{

ret[pos++] = sc;

ret[pos++] = 238;

ret[pos++] = sp;

ret[pos++] = sp;

ret[pos++] = 238;

ret[pos++] = sc;

}

if (\*s == '(')

{

char ps[50];

int i = 0, flag = 1;

s++;

while (flag != 0)

{

ps[i++] = \*s;

if (\*s == '(')

flag++;

if (\*s == ')')

flag--;

s++;

}

ps[--i] = '\0';

nfa(sc, pos, ps);

s--;

}

s++;

}

sc++;

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

{

ret[pos++] = fs[i];

ret[pos++] = 238;

ret[pos++] = sc;

}

ret[pos++] = sc - 1;

ret[pos++] = 238;

ret[pos++] = sc;

}

void main()

{

int i;

char \*inp;

clrscr();

printf("Enter the regular expression :");

gets(inp);

nfa(1, 0, inp);

printf("\nState input state\n");

for (i = 0; i < pos; i = i + 3)

printf("%d --%c--> %d\n", ret[i], ret[i + 1], ret[i + 2]);

printf("\n");

getch();

}

**EXPERIMENT 2 (NFA TO DFA)**

//1. Start the program.

//2. Accept the number of state A and B.

//3. Find the E-closure for node and name if as A.

//4. Find v(a,a) and (a,b) and find a state.

//5. Check whether a number new state is obtained.

//6. Display all the state corresponding A and B.

//7. Stop the program.

#include <stdio.h>

#include <conio.h>

#include <ctype.h>

#include <process.h>

typedef struct

{

int num[10], top;

} stack;

stack s;

int mark[16][31], e\_close[16][31], n, st = 0;

char data[15][15];

void push(int a)

{

s.num[s.top] = a;

s.top = s.top + 1;

}

int pop()

{

int a;

if (s.top == 0)

return (-1);

s.top = s.top - 1;

a = s.num[s.top];

return (a);

}

void epi\_close(int s1, int s2, int c)

{

int i, k, f;

for (i = 1; i <= n; i++)

{

if (data[s2][i] == 'e')

{

f = 0;

for (k = 1; k <= c; k++)

if (e\_close[s1][k] == i)

f = 1;

if (f == 0)

{

c++;

e\_close[s1][c] = i;

push(i);

}

}

}

while (s.top != 0)

epi\_close(s1, pop(), c);

}

int move(int sta, char c)

{

int i;

for (i = 1; i <= n; i++)

{

if (data[sta][i] == c)

return (i);

}

return (0);

}

void e\_union(int m, int n)

{

int i = 0, j, t;

for (j = 1; mark[m][i] != -1; j++)

{

while ((mark[m][i] != e\_close[n][j]) && (mark[m][i] != -1))

i++;

if (mark[m][i] == -1)

mark[m][i] = e\_close[n][j];

}

}

void main()

{

int i, j, k, Lo, m, p, q, t, f;

clrscr();

printf("\n Enter the NFA state table entries:");

scanf("%d", &n);

printf("\n");

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

printf("%d", i);

printf("\n");

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

printf("------");

printf("\n");

for (i = 1; i <= n; i++)

{

printf("%d|", i);

fflush(stdin);

for (j = 1; j <= n; j++)

scanf("%c", &data[i][j]);

}

for (i = 1; i <= 15; i++)

for (j = 1; j <= 30; j++)

{

e\_close[i][j] = -1;

mark[i][j] = -1;

}

for (i = 1; i <= n; i++)

{

e\_close[i][1] = i;

s.top = 0;

epi\_close(i, i, 1);

}

for (i = 1; i <= n; i++)

{

for (j = 1; e\_close[i][j] != -1; j++)

for (k = 2; e\_close[i][k] != -1; k++)

if (e\_close[i][k - 1] > e\_close[i][k])

{

t = e\_close[i][k - 1];

e\_close[i][k - 1] = e\_close[i][k];

e\_close[i][k] = t;

}

}

printf("\n The epsilon closures are:");

for (i = 1; i <= n; i++)

{

printf("\n E(%d)={", i);

for (j = 1; e\_close[i][j] != -1; j++)

printf("%d", e\_close[i][j]);

printf("}");

}

j = 1;

while (e\_close[1][j] != -1)

{

mark[1][j] = e\_close[1][j];

j++;

}

st = 1;

printf("\n DFA Table is:");

printf("\n a b ");

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

for (i = 1; i <= st; i++)

{

printf("\n{");

for (j = 1; mark[i][j] != -1; j++)

printf("%d", mark[i][j]);

printf("}");

while (j < 7)

{

printf(" ");

j++;

}

for (Lo = 1; Lo <= 2; Lo++)

{

for (j = 1; mark[i][j] != -1; j++)

{

if (Lo == 1)

t = move(mark[i][j], 'a');

if (Lo == 2)

t = move(mark[i][j], 'b');

if (t != 0)

e\_union(st + 1, t);

}

for (p = 1; mark[st + 1][p] != -1; p++)

for (q = 2; mark[st + 1][q] != -1; q++)

{

if (mark[st + 1][q - 1] > mark[st + 1][q])

{

t = mark[st + 1][q];

mark[st + 1][q] = mark[st + 1][q - 1];

mark[st + 1][q - 1] = t;

}

}

f = 1;

for (p = 1; p <= st; p++)

{

j = 1;

while ((mark[st + 1][j] == mark[p][j]) && (mark[st + 1][j] != -1))

j++;

if (mark[st + 1][j] == -1 && mark[p][j] == -1)

f = 0;

}

if (mark[st + 1][1] == -1)

f = 0;

printf("\t{");

for (j = 1; mark[st + 1][j] != -1; j++)

{

printf("%d", mark[st + 1][j]);

}

printf("}\t");

if (Lo == 1)

printf(" ");

if (f == 1)

st++;

if (f == 0)

{

for (p = 1; p <= 30; p++)

mark[st + 1][p] = -1;

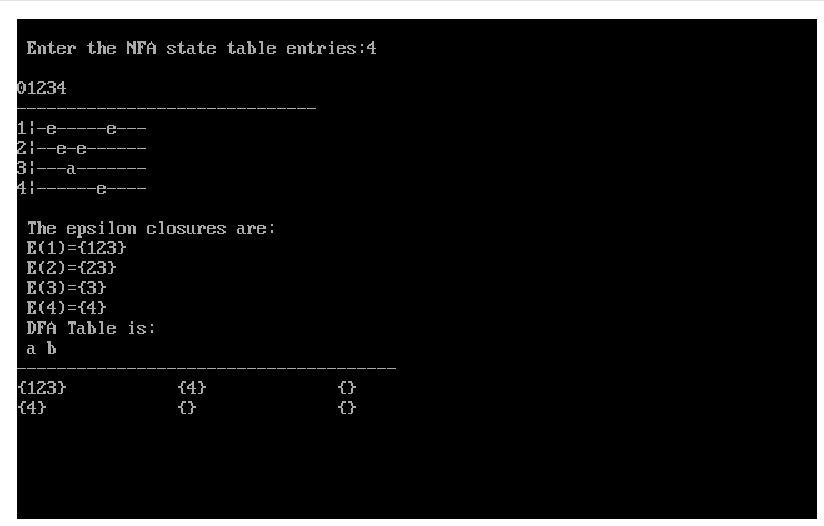
}

}

}

getch();

}



**EXPERIMENT 3 (FIRST)**

// Procedure First

// 1. Input the number of production N.

// 2. Input all the production rule PArray

// 3. Repeat steps a, b, c until process all input production rule i.e. PArray[N]

// a. If Xi ≠ Xi+1 then

// i. Print Result array of Xi which contain FIRST(Xi)

// b. If first element of Xi of PArray is Terminal or ε Then

// i. Add Result = Result U first element

// c. If first element of Xi of PArray is Non-Terminal Then

// i. searchFirst(i, PArray, N)

// 4. End Loop

// 5. If N (last production) then

// a. Print Result array of Xi which contain FIRST(Xi)

// 6. End

// Procedure searchFirst(i, PArray, N)

// 1. Repeat steps Loop j=i+1 to N

// a. If first element of Xj of PArray is Non-Terminal Then

// i. searchFirst(j, of PArray, N)

// b. If first element of Xj of PArray is Terminal or ε Then

// i. Add Result = Result U first element

// ii. Flag=0

// 2. End Loop

// 3. If Flag = 0 Then

// a. Print Result array of Xj which contain FIRST(Xj)

// 4. End

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

using namespace std;

void searchFirst(int n, int i, char pl[], char r[], char result[], int k)

{

int j, flag;

for (j = i + 1; j < n; j++)

{

if (r[i] == pl[j])

{

if (isupper(r[j]))

{

searchFirst(n, j, pl, r, result, k);

}

if (islower(r[j]) || r[j] == '+' || r[j] == '\*' || r[j] == ')' || r[j] == '(')

{

result[k++] = r[j];

result[k++] = ',';

flag = 0;

}

}

}

if (flag == 0)

{

for (j = 0; j < k - 1; j++)

cout << result[j];

}

}

int main()

{

char pr[10][10], pl[10], r[10], prev, result[10];

int i, n, k, j;

cout << "\nHow many production rule : ";

cin >> n;

if (n == 0)

exit(0);

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

{

cout << "\nInput left part of production rules : ";

cin >> pl[i];

cout << "\nInput right part of production rules : ";

cin >> pr[i];

r[i] = pr[i][0];

}

cout << "\nProduction Rules are : \n";

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

{

cout << pl[i] << "->" << pr[i] << "\n"; //<<";"<<r[i]<<"\n";

}

cout << "\n----O U T P U T---\n\n";

prev = pl[0];

k = 0;

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

{

if (prev != pl[i])

{

cout << "\nFIRST(" << prev << ")={";

for (j = 0; j < k - 1; j++)

cout << result[j];

cout << "}";

k = 0;

prev = pl[i];

// cout<<"\n3";

}

if (prev == pl[i])

{

if (islower(r[i]) || r[i] == '+' || r[i] == '\*' || r[i] == ')' || r[i] == '(')

{

result[k++] = r[i];

result[k++] = ',';

}

if (isupper(r[i]))

{

cout << "\nFIRST(" << prev << ")={";

searchFirst(n, i, pl, r, result, k);

cout << "}";

k = 0;

prev = pl[i + 1];

}

}

}

if (i == n)

{

cout << "\nFIRST(" << prev << ")={";

for (j = 0; j < k - 1; j++)

cout << result[j];

cout << "}";

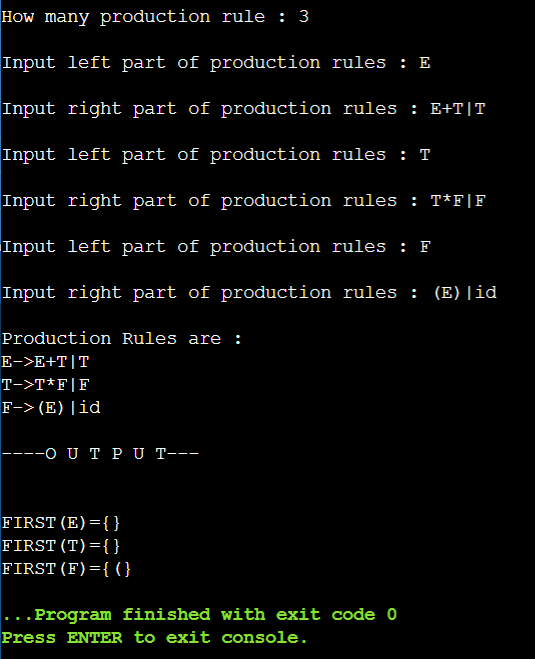
k = 0;

prev = pl[i];

}

return 0;

}



**EXPERIMENT 4 (FOLLOW)**

// 1. Declare the variables.

// 2. Enter the production rules for the grammar.

// 3. Calculate the FOLLOW set for each element call the user defined

// function follow().

// 4. If x->aBb

// a. If x is start symbol then FOLLOW(x)={$}.

// b. If b is NULL then FOLLOW(B)=FOLLOW(x).

// c. If b is not NULL then FOLLOW(B)=FIRST(b).

// END.

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <ctype.h>

using namespace std;

int n, m = 0, p, i = 0, j = 0;

char a[10][10], f[10];

void follow(char c);

void first(char c);

int main()

{

int i, z;

char c, ch;

printf("Enter the no.of productions:");

scanf("%d", &n);

printf("Enter the productions(epsilon=$):\n");

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

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

do

{

m = 0;

printf("Enter the element whose FOLLOW is to be found:");

scanf("%c", &c);

follow(c);

printf("FOLLOW(%c) = { ", c);

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

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

printf(" }\n");

printf("Do you want to continue(0/1)?");

scanf("%d%c", &z, &ch);

} while (z == 1);

}

void follow(char c)

{

if (a[0][0] == c)

f[m++] = '$';

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

{

for (j = 2; j < strlen(a[i]); j++)

{

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

{

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

first(a[i][j + 1]);

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

follow(a[i][0]);

}

}

}

}

void first(char c)

{

int k;

if (!(isupper(c)))

f[m++] = c;

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

{

if (a[k][0] == c)

{

if (a[k][2] == '$')

follow(a[i][0]);

else if (islower(a[k][2]))

f[m++] = a[k][2];

else

first(a[k][2]);

}

}

}

**EXPERIMENT 5 (LEADING)**

// 1. Start the program.

// 2. Get the Set of Productions for the grammar from the user. No redundant & cyclic

// productions must be given.

// 3. The conditions to be checked are:

// Conditions Inclusions in result

// S->Sa add a

// S->Aa add a, production of A

// S->ab add a

// S->AB Production of A

// S->SA none

// S->a take a

// S->SA\* none taken

// S->\*a take \* leave a

// 4. Print the Leading edges.

// 5. Stop the program.

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

#include <string.h>

char av[100], av1[100];

int v = 0, j = 0, v1 = 0;

void disp(int);

struct pro

{

char h, t, t1;

} p[100];

int search(char x)

{

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

if (av[i] == x)

return 1;

return 0;

}

int search1(char x)

{

for (int i = 0; i < v1; i++)

if (av1[i] == x)

return 1;

return 0;

}

void disp1(char x)

{

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

if (p[i].h == x)

disp(i);

}

void disp(int px)

{

if (int(p[px].t) >= 65 && int(p[px].t) <= 90)

{

if (p[px].t1 != '\0' && search1(p[px].t1) == 0)

{

if (p[px].t1 != '\n')

cout << p[px].t1;

av1[v1] = p[px].t1;

v1++;

}

disp1(p[px].t);

}

else if (p[px].t != '#')

{

if (search1(p[px].t) == 0)

{

cout << "\t" << p[px].t;

av1[v1] = p[px].t;

v1++;

}

}

}

void main()

{

clrscr();

cout << "Enter the production: end with ~" << endl

<< endl;

char a1[100];

for (int i = 0; (a1[i] = getc(stdin)) != '~'; i++)

;

a1[i] = '\0';

clrscr();

cout << a1;

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

{

if (a1[k] == '-' && a1[k + 1] == '>')

{

p[j].h = a1[k - 1];

p[j].t = a1[k + 2];

p[j].t1 = '\0';

if (p[j].h == p[j].t)

{

p[j].t = a1[k + 3];

if (int(p[j].t) >= 65 && int(p[j].t) <= 90)

p[j].t = '#';

p[j].t1 = '\0';

}

else if (int(p[j].t) >= 65 && int(p[j].t) <= 90)

{

p[j].t1 = a1[k + 3];

if ((int(p[j].t1) >= 65) &&

(int(p[j].t1) <= 90))

p[j].t1 = '\0';

}

j++;

}

}

cout << endl

<< "The Leading edges r as follows: " << endl;

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

{

if (search(p[j].h) == 0)

{

av[v] = p[i].h;

cout << endl

<< av[v] << ": {";

disp1(av[v]);

cout << " }" << endl

<< endl;

for (k = 0; k < v1; k++)

av1[k] = '\0';

v1 = 0;

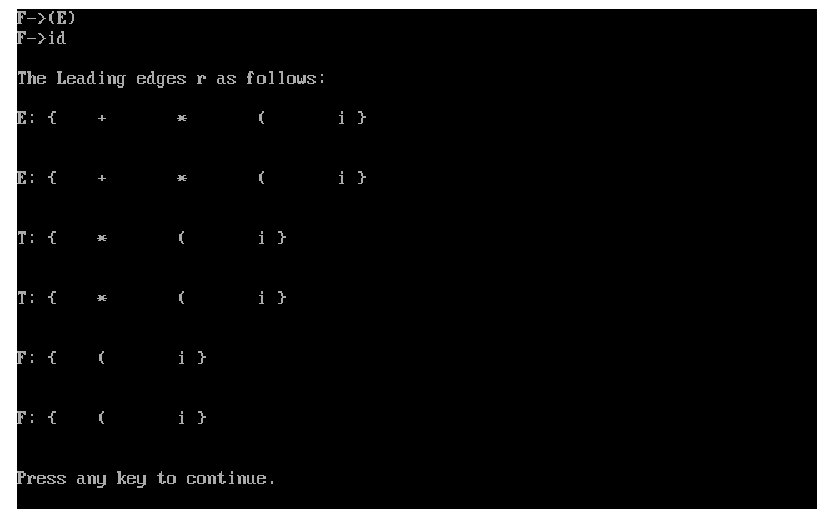
v++;

}

}

getch();

}



**EXPERIMENT 6 (TRAILING)**

// 1. Start the program.

// 2. Get the Set of Productions for the grammar from the user. No redundant & cyclic

// productions must be given.

// 3. Reverse each input productions and print it.

// 4. The conditions to be checked according to the reversed inputs are:

// Conditions Inclusions in result

// S->Sa add a

// S->Aa add a, production of A

// S->ab add a

// S->AB Production of A

// S->SA none

// S->a take a

// S->SA\* none taken

// S->\*a take \* leave a

// 5. Print the Trailing edges.

// 6. Stop the program.

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

#include <string.h>

char b1[100];

char a1[100];

char av[100], av1[100];

int v = 0, j = 0, v1 = 0;

void disp(int);

struct pro

{

char h, t, t1;

} p[100];

void revpro(int l)

{

int k1, k2, j;

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

{

a1[i] = b1[i];

if (b1[i] == '>')

{

for (j = i + 1;; j++)

{

if (int(b1[j]) == 10)

{

a1[j] = b1[j];

break;

}

}

for (k1 = i + 1, k2 = j - 1; k1 < j; k1++, k2--)

a1[k1] = b1[k2];

i = j;

}

}

}

int search(char x)

{

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

if (av[i] == x)

return 1;

return 0;

}

int search1(char x)

{

for (int i = 0; i < v1; i++)

if (av1[i] == x)

return 1;

return 0;

}

void disp1(char x)

{

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

if (p[i].h == x)

disp(i);

}

void disp(int px)

{

if (int(p[px].t) >= 65 && int(p[px].t) <= 90)

{

if (p[px].t1 != '\0' && search1(p[px].t1) == 0)

{

if (p[px].t1 != '\n')

cout << p[px].t1;

av1[v1] = p[px].t1;

v1++;

}

disp1(p[px].t);

}

else if (p[px].t != '#')

{

if (search1(p[px].t) == 0)

{

cout << "\t" << p[px].t;

av1[v1] = p[px].t;

v1++;

}

}

}

void main()

{

clrscr();

cout << "Enter the production: end with ~" << endl

<< endl;

for (int i = 0; (b1[i] = getc(stdin)) != '~'; i++)

;

b1[i] = '\0';

revpro(i);

clrscr();

cout << a1;

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

{

if (a1[k] == '-' && a1[k + 1] == '>')

{

p[j].h = a1[k - 1];

p[j].t = a1[k + 2];

p[j].t1 = '\0';

if (p[j].h == p[j].t)

{

p[j].t = a1[k + 3];

if (int(p[j].t) >= 65 && int(p[j].t) <= 90)

p[j].t = '#';

p[j].t1 = '\0';

}

else if (int(p[j].t) >= 65 && int(p[j].t) <= 90)

{

p[j].t1 = a1[k + 3];

if ((int(p[j].t1) >= 65) &&

(int(p[j].t1) <= 90))

p[j].t1 = '\0';

}

j++;

}

}

cout << endl

<< "The Trailing edges r as follows: " << endl;

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

{

if (search(p[j].h) == 0)

{

av[v] = p[i].h;

cout << endl

<< av[v] << ": {";

disp1(av[v]);

cout << " }" << endl

<< endl;

for (k = 0; k < v1; k++)

av1[k] = '\0';

v1 = 0;

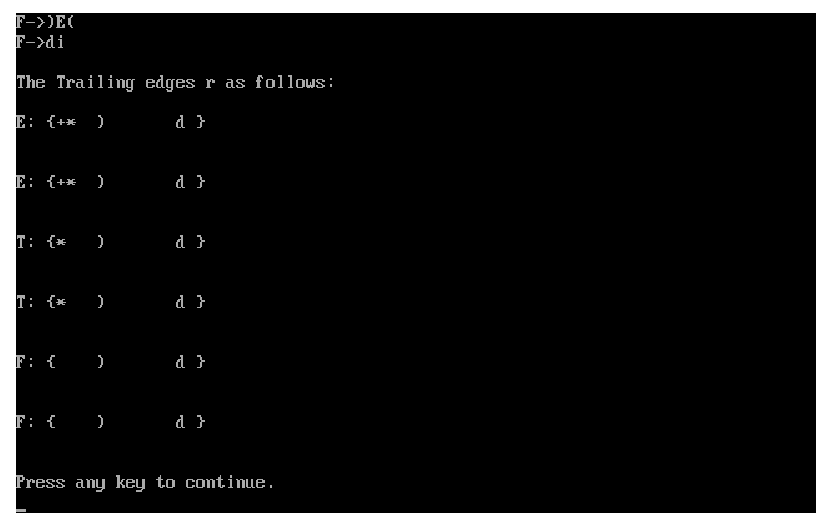
v++;

}

}

getch();

}



**EXPERIMENT 7 (PREDICTION PARSING)**

// INPUT: Grammar G.

// OUTPUT: Parsing table M.

// METHOD:

// 1. For each production A   of the grammar, do the following:

// a. For each terminal a in FIRST(A), add A   to M[A, a] .

// b. If  is in FIRST() , then for each terminal b in FOLLOW(A) , add A   to M [A,

// b] . If  is in FIRST() and $ is in FOLLOW(A), add A   to M[A, $] as well.

// 2. If, after performing the above, there is no production at all in M[A, a], then set M[A,

// a] to error (which we normally represent by an empty entry in the table).

// 3. End.

#include <stdio.h>

#include <conio.h>

#include <string.h>

void main()

{

char fin[10][20], st[10][20], ft[20][20], fol[20][20];

int a = 0, e, i, t, b, c, n, k, l = 0, j, s, m, p;

clrscr();

printf("enter the no. of coordinates\n");

scanf("%d", &n);

printf("enter the productions in a grammar\n");

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

scanf("%s", st[i]);

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

fol[i][0] = '\0';

for (s = 0; s < n; s++)

{

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

{

j = 3;

l = 0;

a = 0;

l1:

if (!((st[i][j] > 64) && (st[i][j] < 91)))

{

for (m = 0; m < l; m++)

{

if (ft[i][m] == st[i][j])

goto s1;

}

ft[i][l] = st[i][j];

l = l + 1;

s1:

j = j + 1;

}

else

{

if (s > 0)

{

while (st[i][j] != st[a][0])

{

a++;

}

b = 0;

while (ft[a][b] != '\0')

{

for (m = 0; m < l; m++)

{

if (ft[i][m] == ft[a][b])

goto s2;

}

ft[i][l] = ft[a][b];

l = l + 1;

s2:

b = b + 1;

}

}

}

while (st[i][j] != '\0')

{

if (st[i][j] == '|')

{

j = j + 1;

goto l1;

}

j = j + 1;

}

ft[i][l] = '\0';

}

}

printf("first pos\n");

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

printf("FIRS[%c]=%s\n", st[i][0], ft[i]);

fol[0][0] = '$';

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

{

k = 0;

j = 3;

if (i == 0)

l = 1;

else

l = 0;

k1:

while ((st[i][0] != st[k][j]) && (k < n))

{

if (st[k][j] == '\0')

{

k++;

j = 2;

}

j++;

}

j = j + 1;

if (st[i][0] == st[k][j - 1])

{

if ((st[k][j] != '|') && (st[k][j] != '\0'))

{

a = 0;

if (!((st[k][j] > 64) && (st[k][j] < 91)))

{

for (m = 0; m < l; m++)

{

if (fol[i][m] == st[k][j])

goto q3;

}

fol[i][l] = st[k][j];

l++;

q3:

}

else

{

while (st[k][j] != st[a][0])

{

a++;

}

p = 0;

while (ft[a][p] != '\0')

{

if (ft[a][p] != '@')

{

for (m = 0; m < l; m++)

{

if (fol[i][m] == ft[a][p])

goto q2;

}

fol[i][l] = ft[a][p];

l = l + 1;

}

else

e = 1;

q2:

p++;

}

if (e == 1)

{

e = 0;

goto a1;

}

}

}

else

{

a1:

c = 0;

a = 0;

while (st[k][0] != st[a][0])

{

a++;

}

while ((fol[a][c] != '\0') && (st[a][0] != st[i][0]))

{

for (m = 0; m < l; m++)

{

if (fol[i][m] == fol[a][c])

goto q1;

}

fol[i][l] = fol[a][c];

l++;

q1:

c++;

}

}

goto k1;

}

fol[i][l] = '\0';

}

printf("follow pos\n");

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

printf("FOLLOW[%c]=%s\n", st[i][0], fol[i]);

printf("\n");

s = 0;

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

{

j = 3;

while (st[i][j] != '\0')

{

if ((st[i][j - 1] == '|') || (j == 3))

{

for (p = 0; p <= 2; p++)

{

fin[s][p] = st[i][p];

}

t = j;

for (p = 3; ((st[i][j] != '|') && (st[i][j] != '\0')); p++)

{

fin[s][p] = st[i][j];

j++;

}

fin[s][p] = '\0';

if (st[i][k] == '@')

{

b = 0;

a = 0;

while (st[a][0] != st[i][0])

{

a++;

}

while (fol[a][b] != '\0')

{

printf("M[%c,%c]=%s\n", st[i][0], fol[a][b], fin[s]);

b++;

}

}

else if (!((st[i][t] > 64) && (st[i][t] < 91)))

printf("M[%c,%c]=%s\n", st[i][0], st[i][t], fin[s]);

else

{

b = 0;

a = 0;

while (st[a][0] != st[i][3])

{

a++;

}

while (ft[a][b] != '\0')

{

printf("M[%c,%c]=%s\n", st[i][0], ft[a][b], fin[s]);

b++;

}

}

s++;

}

if (st[i][j] == '|')

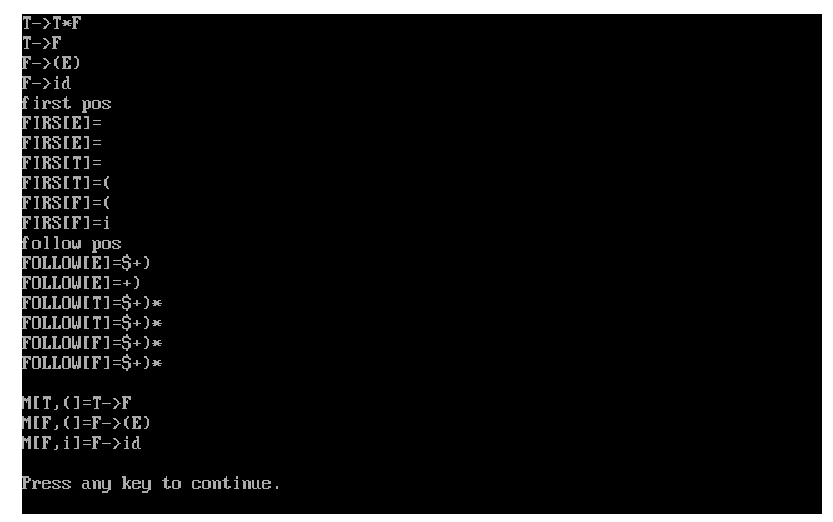
j++;

}

}

getch();

}



**EXPERIMENT 8 (RECURSIVE DECENT)**

#include <iostream>

#include <map>

#include <vector>

using namespace std;

int main()

{

int flag = 0;

map<char, vector<string>> rules;

string exp, test;

rules['S'].push\_back("aAc");

rules['A'].push\_back("cd");

rules['A'].push\_back("d");

cout << "Enter the string: ";

cin >> exp;

string start = "aAc";

if (start[0] != exp[0])

cout << "Not Accepted";

else

{

cout << "S" << endl

<< start << endl;

string a = (rules['A'])[0];

string b = (rules['A'])[1];

string t;

t = start[0] + a + start[2];

cout << t << endl;

if (t == exp)

{

flag = 1;

cout << "Accepted";

}

else

{

cout << start << endl;

t = start[0] + b + start[2];

cout << t << endl;

if (t == exp)

{

flag = 1;

cout << "Accepted";

}

}

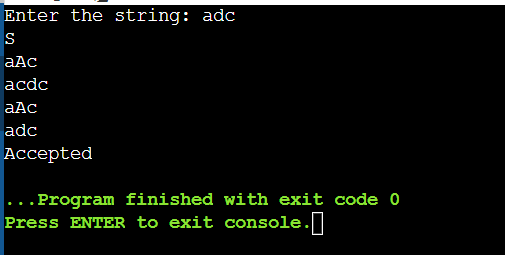
}

if (flag == 0)

cout << "Not accepted";

return 0;

}



**EXPERIMENT 9 (SHIFT REDUCING)** ***1***

// 1. Start the Process.

// 2. Symbols from the input are shifted onto stack until a handle appears on top of

// the stack.

// 3. The Symbols that are the handle on top of the stack are then replaces by the left

// hand side of the production (reduced).

// 4. If this result in another handle on top of the stack, then another reduction is done,

// otherwise we go back to shifting.

// 5. This combination of shifting input symbols onto the stack and reducing

// productions when handles appear on the top of the stack continues until all of the

// input is consumed and the goal symbol is the only thing on the stack - the input

// is then accepted.

// 6. If we reach the end of the input and cannot reduce the stack to the goal symbol,

// the input is rejected.

// 7. Stop the process.

#include <stdio.h>

#include <conio.h>

#include <string.h>

void check();

void check1();

void copy();

void print(int val);

char stack[20];

char temp[10];

char result[10];

int i, j;

void main()

{

clrscr();

printf("Enter Your Expression:");

scanf("%s", &stack);

check();

getch();

}

void check()

{

for (; i < strlen(stack) + 1; i++)

{

if (stack[i] == '+' || stack[i] == '-' || stack[i] == '\*' || stack[i] == '/' || stack[i] == '\0')

{

temp[j] = 'E';

j++;

temp[j] = stack[i];

j++;

}

}

check1();

}

void check1()

{

printf("\n STACK VALUES\tINPUT \n");

l:

for (j = 0, i = 0; i < strlen(temp);)

{

if (temp[i] == '+' || temp[i] == '-' || temp[i] == '\*' ||

temp[i] == '/')

{

printf("\n\t %c", temp[i]);

i++;

print(i);

printf("\n\t %c", temp[i]);

i++;

print(i);

i--;

copy();

goto l;

}

else

{

printf("\n\t %c", temp[i]);

i++;

print(i);

}

}

printf("\n\n\t Expressions Output:%s", temp);

}

void copy()

{

j = 0;

while (temp[i] != '\0')

{

temp[j] = temp[i];

j++;

i++;

}

temp[j] = '\0';

}

void print(int val)

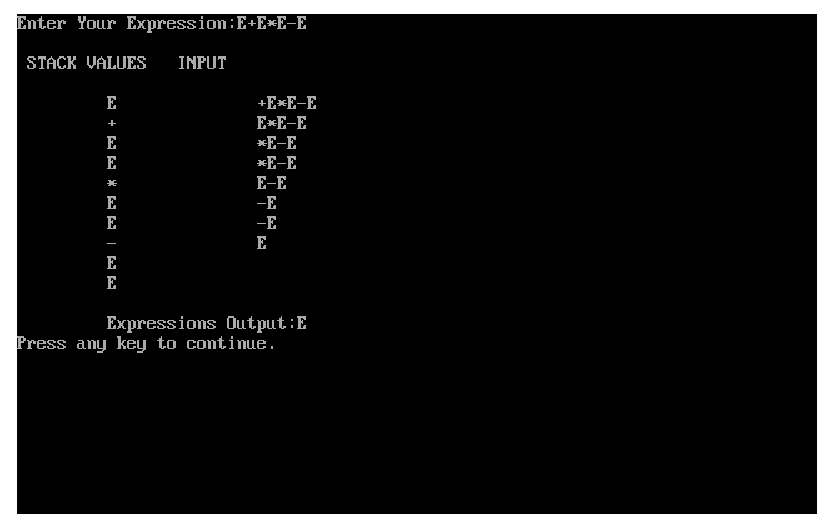
{

printf("\t\t");

for (; val < strlen(temp); val++)

printf("%c", temp[val]);

}



**EXPERIMENT 9 (SHIFT REDUCING)** ***2***

// 1. Start the Process.

// 2. Symbols from the input are shifted onto stack until a handle appears on top of

// the stack.

// 3. The Symbols that are the handle on top of the stack are then replaces by the left

// hand side of the production (reduced).

// 4. If this result in another handle on top of the stack, then another reduction is done,

// otherwise we go back to shifting.

// 5. This combination of shifting input symbols onto the stack and reducing

// productions when handles appear on the top of the stack continues until all of the

// input is consumed and the goal symbol is the only thing on the stack - the input

// is then accepted.

// 6. If we reach the end of the input and cannot reduce the stack to the goal symbol,

// the input is rejected.

// 7. Stop the process.

#include <stdio.h>

#include <string.h>

int k = 0, z = 0, i = 0, j = 0, c = 0;

char a[16], ac[20], stk[15], act[10];

void check();

int main()

{

puts("GRAMMAR is \n E->E+E \n E->E\*E \n E->(E) \n E->id");

puts("Enter input string ");

gets(a);

c = strlen(a);

strcpy(act, "SHIFT->");

puts("STACK \t INPUT \tCOMMENT");

// puts("$ \t");

// puts(a);

printf("$ \t%s$\n", a);

for (k = 0, i = 0; j < c; k++, i++, j++)

{

if (a[j] == 'i' && a[j + 1] == 'd')

{

stk[i] = a[j];

stk[i + 1] = a[j + 1];

stk[i + 2] = '\0';

a[j] = ' ';

a[j + 1] = ' ';

// printf("$ \t%s$\n",a);

printf("\n$%s\t%s$\t%sid", stk, a, act);

check();

}

else

{

stk[i] = a[j];

stk[i + 1] = '\0';

a[j] = ' ';

printf("\n$%s\t%s$\t%ssymbols", stk, a, act);

check();

}

}

}

void check()

{

strcpy(ac, "REDUCE TO E");

for (z = 0; z < c; z++)

if (stk[z] == 'i' && stk[z + 1] == 'd')

{

stk[z] = 'E';

stk[z + 1] = '\0';

printf("\n$%s\t%s$\t%s", stk, a, ac);

j++;

}

for (z = 0; z < c; z++)

if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E')

{

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 2] = '\0';

printf("\n$%s\t%s$\t%s", stk, a, ac);

i = i - 2;

}

for (z = 0; z < c; z++)

if (stk[z] == 'E' && stk[z + 1] == '\*' && stk[z + 2] == 'E')

{

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 1] = '\0';

printf("\n$%s\t%s$\t%s", stk, a, ac);

i = i - 2;

}

for (z = 0; z < c; z++)

if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')')

{

stk[z] = 'E';

stk[z + 1] = '\0';

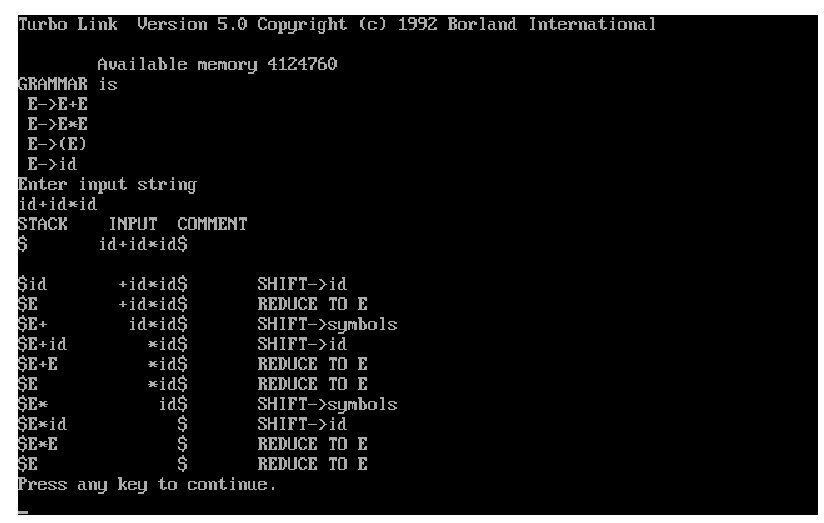
stk[z + 1] = '\0';

printf("\n$%s\t%s$\t%s", stk, a, ac);

i = i - 2;

}

}



**EXPERIMENT 10 (INTERMEDIATE CODE GENERATION)**

// 1. Start the process.

// 2. Input an expression EXP from user.

// 3. Process the expression from right hand side to left hand side.

// 4. FLAG:=0; TOP = -1;

// 5. IF EXP = ‘=’ then

// i. IF EXP(index – 1) = 0 then

// 1. PRINT EXP element from index to (index – 1) and POP

// STACK[TOP]. Terminate

// Else

// i. PRINT Wrong Expression

// [EndIF]

// IF an operator is found and FLAG = 0 then

// i. TOP:= TOP + 1

// ii. add to STACK[TOP].

// iii. FLAG:=1

// Else

// i. pop twice the STACK and result add to the newID(identifier) and

// PRINT.

// ii. TOP:=TOP-2. Save newID to STACK[TOP]

// iii. FLAG:=0

// [EndIF]

// 6. IF an operand is found then

// i. TOP:=TOP+1

// ii. move to STACK [TOP]

// iii. IF TOP > 1 then

// 1. pop twice the STACK and result add to the

// newID(identifier) and PRINT.

// 2. TOP:=TOP-2. Save newID to STACK[TOP]

// 3. FLAG:=0

// [End]

// 7. End the process

#include <iostream>

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <ctype.h>

using namespace std;

int main()

{

char g, exp[20], stack[20];

int m = 0, i, top = -1, flag = 0, len, j;

cout << "\nInput an expression : ";

gets(exp);

cout << "\nIntermediate code generator\n";

len = strlen(exp);

// If expression contain digits

if (isdigit(exp[len - 1]))

{

cout << "T = inttoreal(";

i = len - 1;

while (isdigit(exp[i]))

{

i--;

}

for (j = i + 1; j < len; j++)

{

cout << exp[j];

}

cout << ".0)\n";

exp[i + 1] = 'T';

len = i + 2;

}

else // If expression having no digit

{

cout << "T = " << exp[len - 1] << "\n";

exp[len - 1] = 'T';

}

for (i = len - 1; i >= 0; i--)

{

if (exp[i] == '=')

{

if ((i - 1) == 0)

{

// If expression contains unary operator in RHS near = operator

if (isalpha(stack[top]))

{

cout << exp[i - 1] << " " << exp[i] << " " << stack[top];

}

else

{

cout << exp[i - 1] << " " << exp[i] << "" << stack[top] << stack[top - 1];

}

break;

}

else

{

cout << "\nWrong Expression !!!";

break;

}

}

if (exp[i] == '+' || exp[i] == '/' || exp[i] == '\*' || exp[i] == '-' || exp[i] == '%')

{

if (flag == 0)

{

flag = 1;

top = top + 1;

stack[top] = exp[i];

}

else

{

g = char('A' + m);

m++;

cout << g << " = " << stack[top] << stack[top - 1] << "\n";

stack[top - 1] = g;

stack[top] = exp[i];

flag = 0;

}

}

else

{

top = top + 1;

stack[top] = exp[i];

if (top > 1)

{

g = char('A' + m);

m++;

cout << g << " = " << stack[top] << stack[top - 1] << stack[top - 2] << "\n";

top = top - 2;

stack[top] = g;

flag = 0;

}

}

}

return 0;

}

