**Experiment No – 2**

**AIM: To implement a Symbol table with functions to create, insert, modify, search and displaying C language.**

ALGORITHM:

1. Start the program
2. Define the structure of the symbol table.
3. Enter the choice for performing the operations in the symbol table.
4. If choice is 1, search symbol table for the symbol to be inserted. If the symbol is already present display “Duplicate Symbol”, else insert symbol and corresponding address in the symbol table.
5. If choice is 2, symbols present in the symbols table are displayed.
6. If choice is 3, symbol to be deleted us searched in the symbol table. If found deletes else Displays “Not found”.
7. If choice is 5, the symbol to be modified is searched in the symbol table. The label or address or both can be modified.

PROGRAM:

/\* C program to implement SYMBOL TABLE \*/

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

#include <string.h>

#define null 0

int size = 0;

void insert();

void del();

int mysearch(char[]);

void modify();

void display();

struct symboltab

{

char label[10];

int addr;

struct symboltab \*next;

};

struct symboltab \*first, \*last;

void main(){

int op;

int y;

char la[10];

do{

printf("\nSYMBOL TABLE IMPLEMENTATION\n");

printf("1. INSERT\n");

printf("2. Display\n");

printf("3. Delete\n");

printf("4. Search\n");

printf("5. Modify\n");

printf("6. End\n");

printf("Enter your options: ");

scanf("%d",&op);

switch (op){

case 1:

insert();

display();

break;

case 2:

display();

break;

case 3:

del();

display();

case 4:

printf("Enter the label to be searched: ");

scanf("%s",la);

y = mysearch(la);

if(y == 1)

printf("The label is already in the symbol table\n");

else

printf("The label is not found in the symbol table\n");

break;

case 5:

modify();

display();

case 6:

break;

}

}while(op<6);

getch();

}

void insert(){

int n;

char l[10];

printf("Enter the label: ");

scanf("%s",l);

n = mysearch(l);

if (n == 1)

printf("The label is already in the symbol table. Duplicate cannot be inserted\n");

else{

struct symboltab \*p;

p = malloc(sizeof(struct symboltab));

strcpy(p->label,l);

printf("Enter the address: ");

scanf("%d",&p->addr);

p->next = null;

if (size == 0){

first = p;

last = p;

}

else{

last->next = p;

last = p;

}

size++;

}

}

void display(){

int i;

struct symboltab \*p;

p = first;

printf("Label\t\tAddress\n");

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

printf("%s\t%d\n",p->label,p->addr);

p = p->next;

}

}

int mysearch(char lab[]){

int i;

int flag = 0;

struct symboltab \*p;

p = first;

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

if(strcmp(p->label,lab) == 0){

flag = 1;

}

p = p->next;

}

return flag;

}

void modify(){

char l[10],nl[10];

int addr, choice, i, s;

struct symboltab \*p;

p = first;

printf("What do you want to modify?\n");

printf("1. Only the label\n");

printf("2. Only the address of a particular label\n");

printf("3. Both the label and address\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice){

case 1:

printf("Enter the old label\n");

scanf("%s",l);

printf("Enter the new label: ");

scanf("%s",nl);

s = mysearch(l);

if(s==0)

printf("\n No such label");

else{

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

if(strcmp(p->label,l)==0){

strcpy(p->label,nl);

}

p = p->next;

}

}

break;

case 2:

printf("Enter the label whose address is to be modified: ");

scanf("%s",l);

printf("Enter the new address: ");

scanf("%d",&addr);

s = mysearch(l);

if (s == 0)

printf("\nNo such Label");

else{

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

if(strcmp(p->label,l)==0){

p->addr = addr;

}

p = p->next;

}

}

break;

case 3:

printf("Enter the old label: ");

scanf("%s",l);

printf("Enter the new label: ");

scanf("%s",nl);

printf("Enter the new address: ");

scanf("%d",&addr);

s = mysearch(l);

if(s == 0)

printf("\nNo such label");

else{

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

if(strcmp(p->label,l)==0){

strcpy(p->label,nl);

p->addr = addr;

}

p = p->next;

}

}

break;

}

}

void del(){

int a;

char l[10];

struct symboltab \*p, \*q;

p = first;

printf("Enter the label to be deleted: ");

scanf("%s",l);

a = mysearch(l);

if (a == 0)

printf("Label not found! \n");

else{

if(strcmp(first->label,l)==0)

first = first->next;

else{

if(strcmp(last->label,l)==0)

{

q = p->next;

while(strcmp(q->label,l)!= 0){

p=p->next;

q = q->next;

}

p->next = null;

last = p;

}

else{

q= p->next;

while(strcmp(q->label,l)!=0)

{

p=p->next;

q=q->next;

}

p->next = q->next;

}

size--;

}

}

}

**Experiment no -3**

**IMPLEMENTATION OF SINGLE PASS ASSEMBLER**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <conio.h>

struct symtab

{

char symbol[20];

int addr;

char ch

}stab[20];

void main()

{

char cc[10],opcode[25],label[25],operand[25],object[10];

int s, entry = 0, flag = 0, c=0, i, op,locctr, code,length;

char mne[20][20] = {"LDA","STA","ADD","COMP","J","JLT"};

char opval[20][20] = {"1A","2B","1E","33","L9"};

FILE \*fp1, \*fp2;

fp1 = fopen("sinput.txt","r");

fp2 = fopen("sobj.txt","w");

fscanf(fp1,"%s%s",&label,&opcode);

if(strcmp(opcode,"START")==0)

fscanf(fp1,"%d",&locctr);

s = locctr;

fscanf(fp1,"%s%s%s",&label,&opcode,&operand);

while(!eof(fp1))

{

if(strcmp(opcode,"RESW")==0){

op = atoi(operand);

locctr = locctr + (op\*3);

}

else if(strcmp(opcode,"RESB")==0){

op = atoi(operand);

locctr = locctr+op;

}

else{

if(strcmp(opcode,"BYTE")==0){

op = strlen(operand);

locctr = locctr+op;

}

else if(strcmp(opcode,"WORD")==0){

locctr = locctr+3;

code = atoi(operand);

}

else

locctr = locctr+3;

fscanf(fp1,"%s%s%s",&label,&opcode,&operand);

}

length = locctr-s;

rewind(fp1);

fscanf(fp1,"%s%s",&label,&opcode);

printf("%s%s",label,opcode);

getch();

if(strcmp(opcode,"START") == 0)

fscanf(fp1,"%d",&locctr);

s = locctr;

fprintf(fp2,"H^%s^00%d^00%d",label,locctr,length);

fprintf(fp2,"\nT^%d^IE",locctr);

fscanf(fp1,"%s%s%s",&label,&opcode,&operand);

while(!feof(fp1))

{

strcpy(object,"");

flag = 0;

if(strcmp(label,"-")!=0){

entry = 0;

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

if(strcmp(stab[i].symbol,label)==0 && stab[i].ch == '\*')

{

fprintf(fp2,"\nT^%d^02^%d",stab[i].addr,locctr);

stab[c].addr = locctr;

stab[c].ch = 'd';

entry = 1;

fprintf(fp2,"\nT^%d^IE",locctr);

}

}

if(entry == 0 ){

strcpy(stab[c].symbol,label);

stab[c].addr = locctr;

stab[c].ch = 'd';

}

c++;

}

entry =0;

for(i =0 ;i<7;i++){

if(strcmp(opcode,mne[i])==0)

{

strcpy(object,opval[i]);

entry = 1;

break;

}

}

if(entry == 1)

{

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

{

if(strcmp(operand,stab[i].symbol)==0){

code = stab[i].addr;

flag =1;

break;

}

}

if(flag == 0)

{

code = 0;

strcpy(stab[c].symbol,operand);

stab[c].addr = locctr+1;

stab[c].ch = '\*';

c++;

}

}

if(strcmp(opcode,"RESW")==0){

op=atoi(operand);

locctr = locctr+(op\*3);

}

else{

if(strcmp(opcode,"RESB")==0){

op = atoi(operand);

locctr = locctr+op;

}

else{

if(strcmp(opcode,"BYTE")==0){

op = strlen(operand);

locctr = locctr+op;

}

else if(strcmp(opcode,"WORD")==0){

locctr = locctr+3;

code = atoi(operand);

}

else{

locctr = locctr +3;

if(strcmp(opcode,"RESW")!=0 && strcmp(opcode,"RESB")!=0){

if(strcmp(opcode,"BYTE")==0){

fprintf(fp2,"^%s",object);

for(i=2;i<op-1;i++)

fprintf(fp2,"%d",toascii(operand[i]));

}

else{

fprintf(fp2,"^%s%d",object,code);

}

fscanf(fp1,"%s%s%s",&label,&opcode,&operand);

}

fprintf(fp2,"\nE^%d",s);

fclose(fp1);

fclose(fp2);

}

}

}

}

}

**Experiment no -4**

**Implementation of pass 2 compiler**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

void main()

{

char a[10],ad[10],label[10],opcode[10],operand[10],mnemonic[10],symbol[10];

int i,locctr,code,add,len,actual\_len;

FILE \*fp1,\*fp2,\*fp3,\*fp4;

clrscr();

fp1=fopen("twoout.dat","w");

fp2=fopen("symtab.dat","r");

fp3=fopen("out.dat","r");

fp4=fopen("optab.dat","r");

fscanf(fp3,"%s%s%s",label,opcode,operand);

if(strcmp(opcode,"START")==0)

{

fprintf(fp1,"\t%s\t%s\t%s\n",label,opcode,operand);

fscanf(fp3,"%d%s%s%s",&locctr,label,opcode,operand);

}

while(strcmp(opcode,"END")!=0)

{

if(strcmp(opcode,"BYTE")==0)

{

fprintf(fp1,"%d\t%s\t%s\t%s\t",locctr,label,opcode,operand);

len=strlen(operand);

actual\_len=len-3;

for(i=2;i<(actual\_len+2);i++)

{

itoa(operand[i],ad,16);

fprintf(fp1,"%s",ad);

}

fprintf(fp1,"\n");

}

else if(strcmp(opcode,"WORD")==0)

{

len=strlen(operand);

itoa(atoi(operand),a,10);

fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",locctr,label,opcode,operand,a);

}

else if((strcmp(opcode,"RESB")==0)||(strcmp(opcode,"RESW")==0))

{

fprintf(fp1,"%d\t%s\t%s\t%s\n",locctr,label,opcode,operand);

}

else

{

rewind(fp4);

fscanf(fp4,"%s%d",mnemonic,&code);

while(strcmp(opcode,mnemonic)!=0)

fscanf(fp4,"%s%d",mnemonic,&code);

if(strcmp(operand,"\*\*")==0)

{

fprintf(fp1,"%d\t%s\t%s\t%s\t%d0000\n",locctr,label,opcode,operand,code);

}

else

{

rewind(fp2);

fscanf(fp2,"%s%d",symbol,&add);

while(strcmp(operand,symbol)!=0)

{

fscanf(fp2,"%s%d",symbol,&add);

}

fprintf(fp1,"%d\t%s\t%s\t%s\t%d%d\n",locctr,label,opcode,operand,code,add);

}

}

fscanf(fp3,"%d%s%s%s\n",&locctr,label,opcode,operand);

}

fprintf(fp1,"%d\t%s\t%s\t%s\n",locctr,label,opcode,operand);

printf("FINISHED");

fclose(fp1);

fclose(fp2);

fclose(fp3);

fclose(fp4);

getch();

}

**EXPERIMENT – 5**

**Write a program for dividing the given input program into lexemes.**

ALGORITHM

1. Start
2. Read the file and open the file as read mode.
3. Read the strings from token identifiers, variables
4. Take parenthesis also a token.
5. Parse the string.
6. Stop

#include<stdio.h>

#include<conio.h>

#include<string.h>

main()

{

int i,j,k,p,c;

char s[120],r[100];

char par[6]={'(',')','{','}','[',']'};

char sym[9]={'.',';',':',',','<','>','?','$','#'};

char key[9][10]={"main","if","else","switch","void","do","while","for","return"};

char dat[4][10]={"int","float","char","double"};

char opr[5]={'\*','+','-','/','^'};

char \*fp;

clrscr();

printf("\n\n\t enter file name");

scanf("%s",s);

fp=fopens(s,"r");

c=0;

do

{

fscanf(fp,"%s",r);

getch();

for(i=0;i<6;i++)

if(strchr(r,par[i])!=NULL)

printf("\n paranthesis:%c",par[i]);

for(i=0;i<9;i++)

if(strchr(r,sym[i])!=NULL)

printf("\n symbol:%c",sym[i]);

for(i=0;i<9;i++)

if(strstr(r,key[i])!=NULL)

printf("\n keyword:%s",key[i]);

for(i=0;i<4;i++)

if((strstr(r,dat[i])&&(!strstr(r,"printf")))!=NULL)

{

printf("\n data type:%s",dat[i]);

fscanf(fp,"%s",r);

printf("\n identifiers :%s",r);

}

for(i=0;i<5;i++)

if(strchr(r,opr[i])!=NULL)

printf("\n operator:%c",opr[i]);

p=c;

c=ftell(fp);

}while(p!=c);

return 0;

}

**EXPERIMENT-6**

**Write a program to compute FIRST function.**

#include<stdio.h>

#include<ctype.h>

void FIRST(char[],char );

void addToResultSet(char[],char);

int numOfProductions;

char productionSet[10][10];

main()

{

int i;

char choice;

char c;

char result[20];

printf("How many number of productions ? :");

scanf(" %d",&numOfProductions);

for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T

{

printf("Enter productions Number %d : ",i+1);

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

}

do

{

printf("\n Find the FIRST of :");

scanf(" %c",&c);

FIRST(result,c); //Compute FIRST; Get Answer in 'result' array

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

for(i=0;result[i]!='\0';i++)

printf(" %c ",result[i]); //Display result

printf("}\n");

printf("press 'y' to continue : ");

scanf(" %c",&choice);

}

while(choice=='y'||choice =='Y');

}

/\*

\*Function FIRST:

\*Compute the elements in FIRST(c) and write them

\*in Result Array.

\*/

void FIRST(char\* Result,char c)

{

int i,j,k;

char subResult[20];

int foundEpsilon;

subResult[0]='\0';

Result[0]='\0';

//If X is terminal, FIRST(X) = {X}.

if(!(isupper(c)))

{

addToResultSet(Result,c);

return ;

}

//If X is non terminal

//Read each production

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

{

//Find production with X as LHS

if(productionSet[i][0]==c)

{

//If X → ε is a production, then add ε to FIRST(X).

if(productionSet[i][2]=='$') addToResultSet(Result,'$');

//If X is a non-terminal, and X → Y1 Y2 … Yk

//is a production, then add a to FIRST(X)

//if for some i, a is in FIRST(Yi),

//and ε is in all of FIRST(Y1), …, FIRST(Yi-1).

else

{

j=2;

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

{

foundEpsilon=0;

FIRST(subResult,productionSet[i][j]);

for(k=0;subResult[k]!='\0';k++)

addToResultSet(Result,subResult[k]);

for(k=0;subResult[k]!='\0';k++)

if(subResult[k]=='$')

{

foundEpsilon=1;

break;

}

//No ε found, no need to check next element

if(!foundEpsilon)

break;

j++;

}

}

}

}

return ;

}

/\* addToResultSet adds the computed

\*element to result set.

\*This code avoids multiple inclusion of elements

\*/

void addToResultSet(char Result[],char val)

{

int k;

for(k=0 ;Result[k]!='\0';k++)

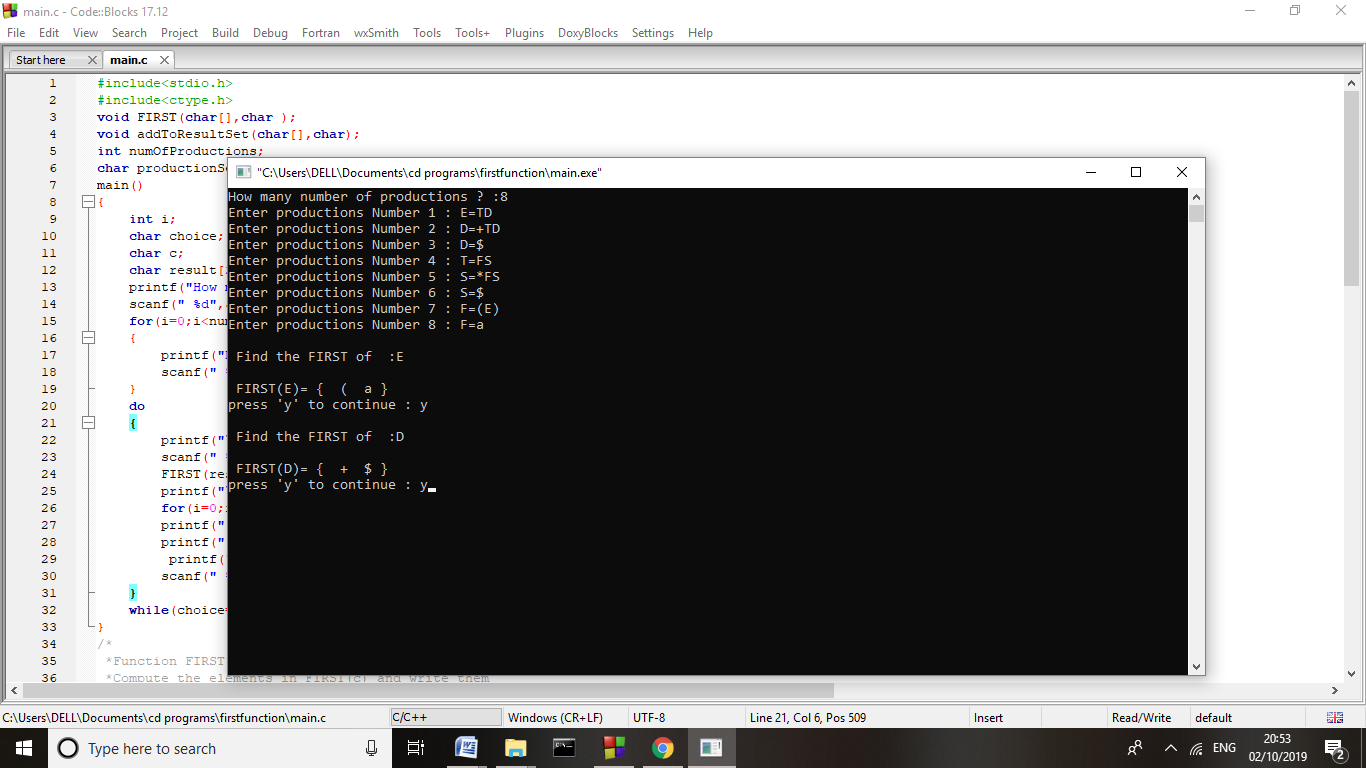
if(Result[k]==val)

return;

Result[k]=val;

Result[k+1]='\0';

}



**EXPERIMENT-7**

**Write a program to compute FOLLOW function.**

#include<stdio.h>  
#include<string.h>

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

char a[10][10],followResult[10];

void follow(char c);

void first(char c);

void addToResult(char);

int main()

{

int i;

int choice;

char c,ch;

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

scanf("%d", &n);

printf(" Enter %d productions\nProduction with multiple terms should be give as separate productions \n", n);

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

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

// gets(a[i]);

do

{

m=0;

printf("Find FOLLOW of -->");

scanf(" %c",&c);

follow(c);

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

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

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

printf(" }\n");

printf("Do you want to continue(Press 1 to continue....)?");

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

}

while(choice==1);

}

void follow(char c)

{

if(a[0][0]==c)addToResult('$');

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;

addToResult(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];

addToResult(a[k][2]);

else first(a[k][2]);

}

}

}

void addToResult(char c)

{

int i;

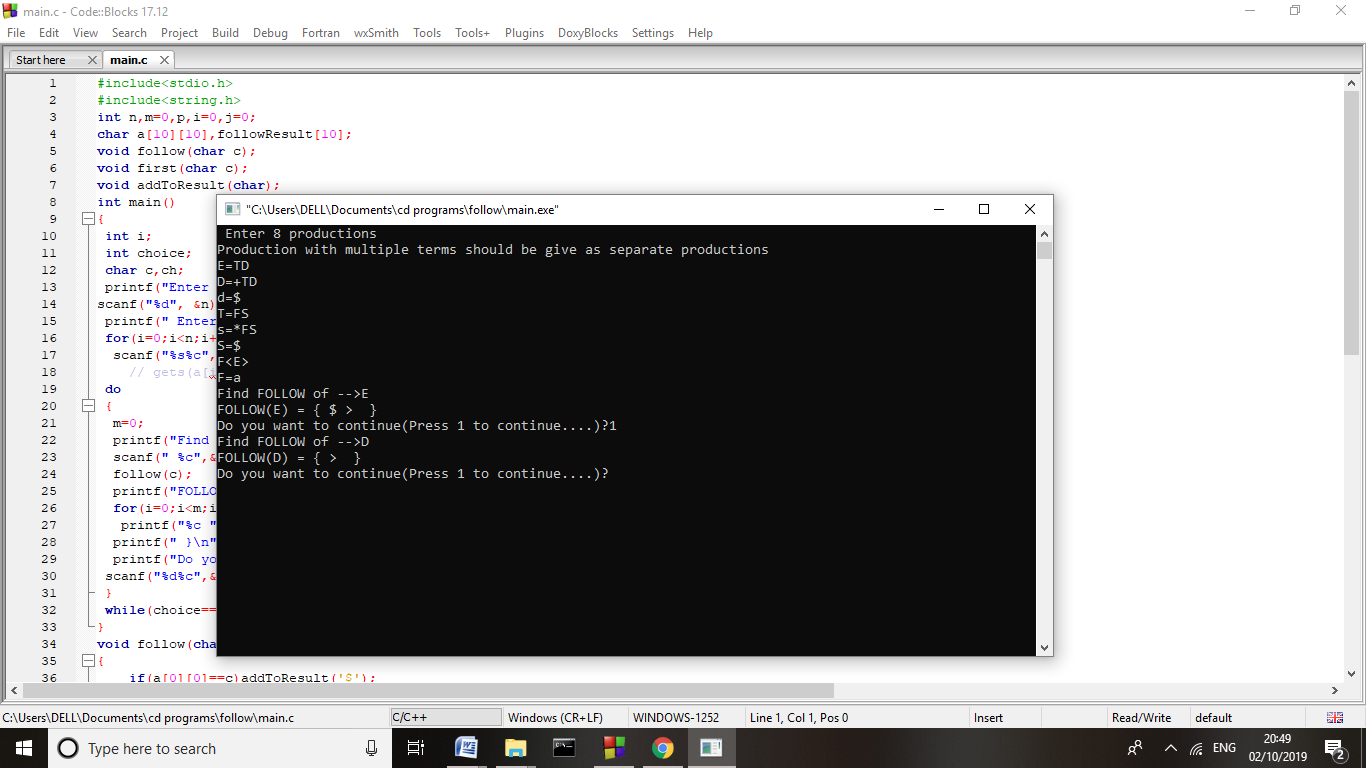
for( i=0;i<=m;i++)

if(followResult[i]==c)

return;

followResult[m++]=c;

}



**EXPERIMENT- 8**

**Write a program to implement operator precedence parsing**.

#include<stdio.h>

#include<string.h>

char \*input;

int i=0;

char lasthandle[6],stack[50],handles[][5]={")E(","E\*E","E+E","i","E^E"};

//(E) becomes )E( when pushed to stack

int top=0,l;

char prec[9][9]={

                            /\*input\*/

            /\*stack    +    -   \*   /   ^   i   (   )   $  \*/

            /\*  + \*/  '>', '>','<','<','<','<','<','>','>',

            /\*  - \*/  '>', '>','<','<','<','<','<','>','>',

            /\*  \* \*/  '>', '>','>','>','<','<','<','>','>',

            /\*  / \*/  '>', '>','>','>','<','<','<','>','>',

            /\*  ^ \*/  '>', '>','>','>','<','<','<','>','>',

            /\*  i \*/  '>', '>','>','>','>','e','e','>','>',

            /\*  ( \*/  '<', '<','<','<','<','<','<','>','e',

            /\*  ) \*/  '>', '>','>','>','>','e','e','>','>',

            /\*  $ \*/  '<', '<','<','<','<','<','<','<','>',

                };

int getindex(char c)

{

switch(c)

    {

    case '+':return 0;

    case '-':return 1;

    case '\*':return 2;

    case '/':return 3;

    case '^':return 4;

    case 'i':return 5;

    case '(':return 6;

    case ')':return 7;

    case '$':return 8;

    }

}

int shift()

{

stack[++top]=\*(input+i++);

stack[top+1]='\0';

}

int reduce()

{

int i,len,found,t;

for(i=0;i<5;i++)//selecting handles

{

    len=strlen(handles[i]);

    if(stack[top]==handles[i][0]&&top+1>=len)

        {

        found=1;

        for(t=0;t<len;t++)

            {

            if(stack[top-t]!=handles[i][t])

                {

                found=0;

                break;

                }

            }

        if(found==1)

            {

            stack[top-t+1]='E';

            top=top-t+1;

            strcpy(lasthandle,handles[i]);

            stack[top+1]='\0';

            return 1;//successful reduction

            }

        }

}

return 0;

}

void dispstack()

{

int j;

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

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

}

void dispinput()

{

int j;

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

printf("%c",\*(input+j));

}

void main()

{

int j;

input=(char\*)malloc(50\*sizeof(char));

printf("\nEnter the string\n");

scanf("%s",input);

input=strcat(input,"$");

l=strlen(input);

strcpy(stack,"$");

printf("\nSTACK\tINPUT\tACTION");

while(i<=l)

{

shift();

printf("\n");

dispstack();

printf("\t");

dispinput();

printf("\tShift");

if(prec[getindex(stack[top])][getindex(input[i])]=='>')

{

while(reduce())

{

printf("\n");

dispstack();

printf("\t");

dispinput();

printf("\tReduced: E->%s",lasthandle);

}}}

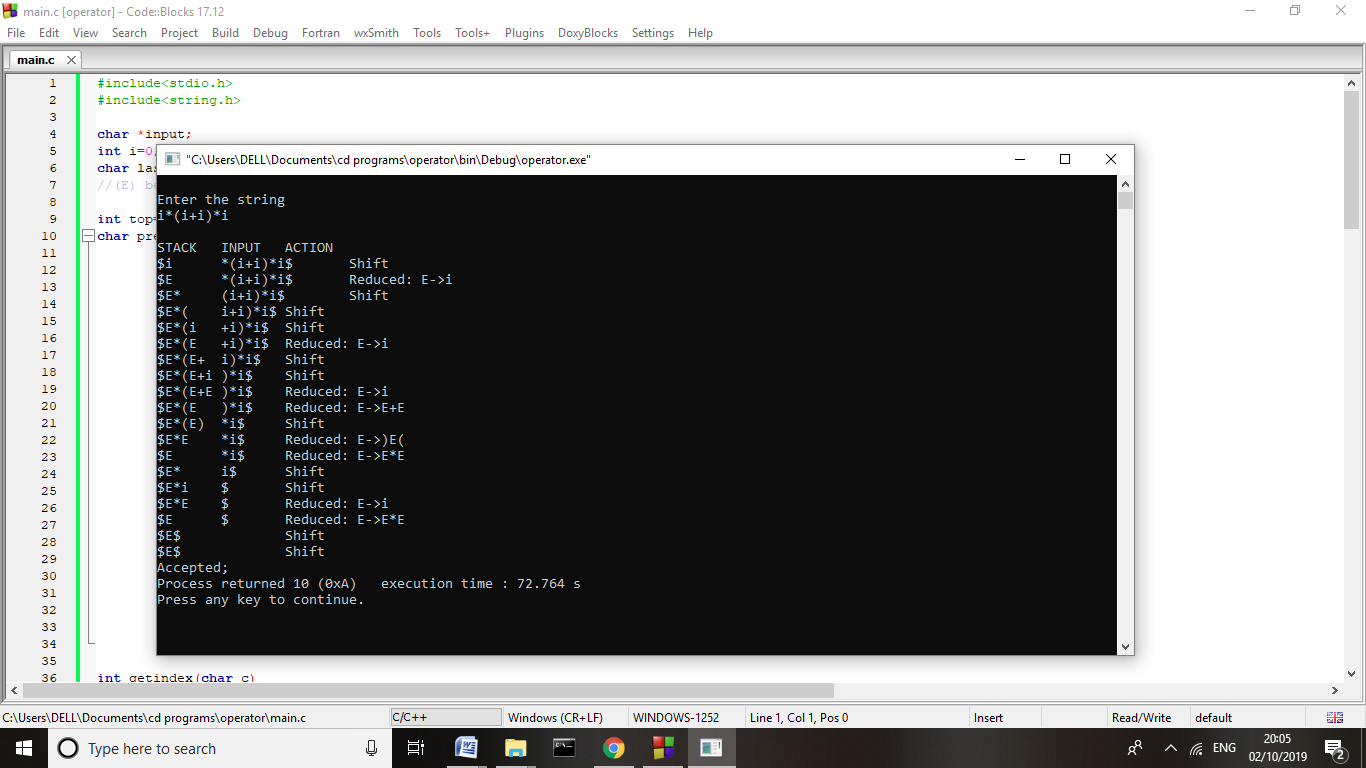
if(strcmp(stack,"$E$")==0)

    printf("\nAccepted;");

else

    printf("\nNot Accepted;");

}

****

**EXPERIMENT-9**

**Write a program on recursive descendent parsing.**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

void Tprime();

void Eprime();

void E();

void check();

void T();

char expression[10];

int count, flag;

 int main()

{

      count = 0;

      flag = 0;

      printf("\nEnter an Algebraic Expression:\t");

      scanf("%s", expression);

      E();

      if((strlen(expression) == count) && (flag == 0))

      {

            printf("\nThe Expression %s is Valid\n", expression);

      }

      else

      {

            printf("\nThe Expression %s is Invalid\n", expression);

      }

}

void E()

{

      T();

      Eprime();

}

void T()

{

      check();

      Tprime();

}

void Tprime()

{

      if(expression[count] == '\*')

      {

            count++;

            check();

            Tprime();

     }}

void check()

{

      if(isalnum(expression[count]))

      {

            count++;

      }

      else if(expression[count] == '(')

      {

            count++;

            E();

            if(expression[count] == ')')

            {

                  count++;

            }

            else

            {

                  flag = 1;

            }

      }

      else

      {

            flag = 1;

      }}

void Eprime()

{

      if(expression[count] == '+')

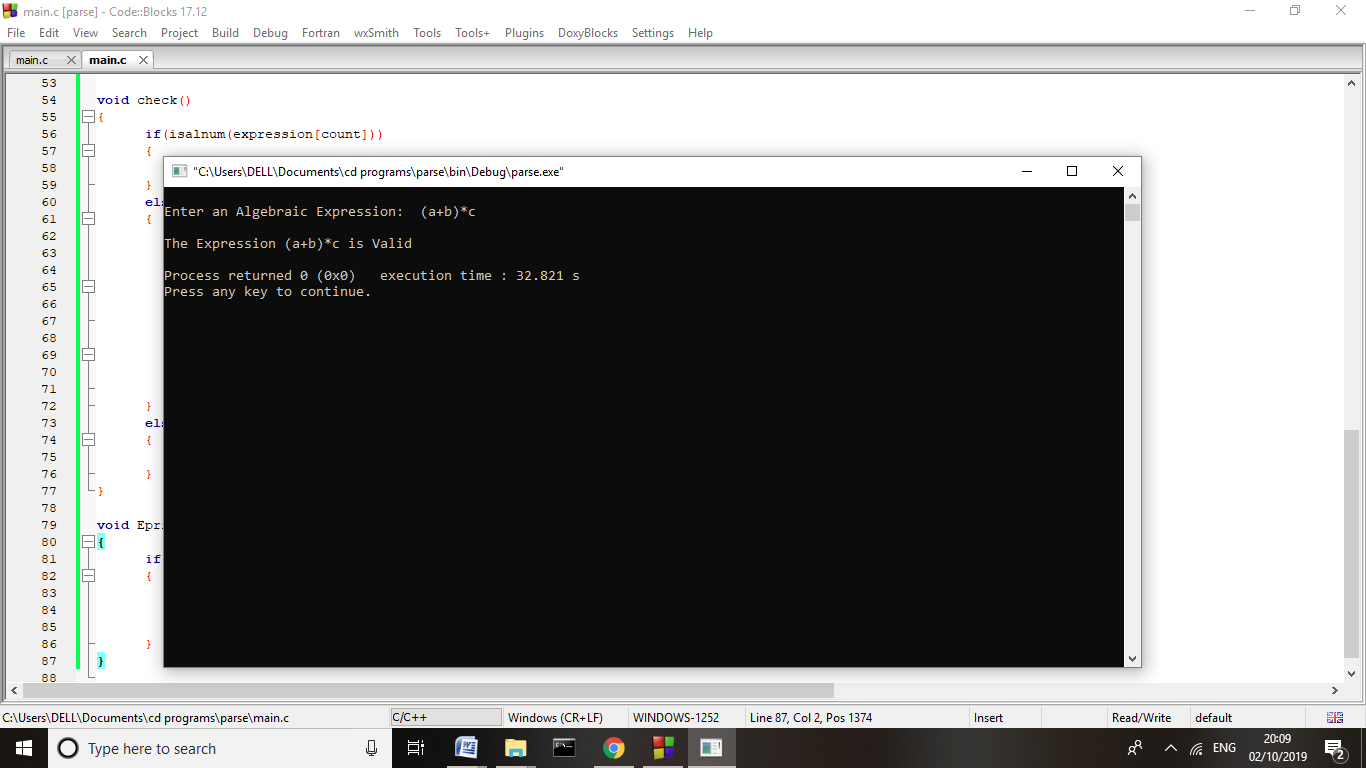
      {

            count++;

            T();

            Eprime();

      }}

****

**EXPERIMENT- 10**

**Write a program for implementing Shift Reduce Parsing using C**

## #include"stdio.h"

#include"stdlib.h"

#include"conio.h"

#include"string.h"

char ip\_sym[15],stack[15];

int ip\_ptr=0,st\_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main()

{

clrscr();

printf("\n\t\t SHIFT REDUCE PARSER\n");

printf("\n GRAMMER\n");

printf("\n E->E+E\n E->E/E");

printf("\n E->E\*E\n E->a/b");

printf("\n enter the input symbol:\t");

gets(ip\_sym);

printf("\n\t stack implementation table");

printf("\n stack\t\t input symbol\t\t action");

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

printf("\n $\t\t%s$\t\t\t--",ip\_sym);

strcpy(act,"shift ");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

len=strlen(ip\_sym);

for(i=0;i<=len-1;i++){

stack[st\_ptr]=ip\_sym[ip\_ptr];

stack[st\_ptr+1]='\0';

ip\_sym[ip\_ptr]=' ';

ip\_ptr++;

printf("\n $%s\t\t%s$\t\t\t%s",stack,ip\_sym,act);

strcpy(act,"shift ");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st\_ptr++;

}

st\_ptr++;

check();

}

void check()

{

int flag=0;

temp2[0]=stack[st\_ptr];

temp2[1]='\0';

if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))

{

 stack[st\_ptr]='E';

 if(!strcmpi(temp2,"a"))

  printf("\n $%s\t\t%s$\t\t\tE->a",stack, ip\_sym);

 else

  printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip\_sym);

 flag=1;

}

if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"\*"))||(!strcmpi(temp2,"/")))

{

 flag=1;

}

if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E\*E")))

{

strcpy(stack,"E");

st\_ptr=0;

if(!strcmpi(stack,"E+E"))

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

else

if(!strcmpi(stack,"E\E"))

printf("\n $%s\t\t %s$\t\t\tE->E\E",stack,ip\_sym);

else

printf("\n $%s\t\t%s$\t\t\tE->E\*E",stack,ip\_sym);

flag=1;

}

if(!strcmpi(stack,"E")&&ip\_ptr==len)

{

printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip\_sym);

getch();

exit(0);

}

if(flag==0)

{

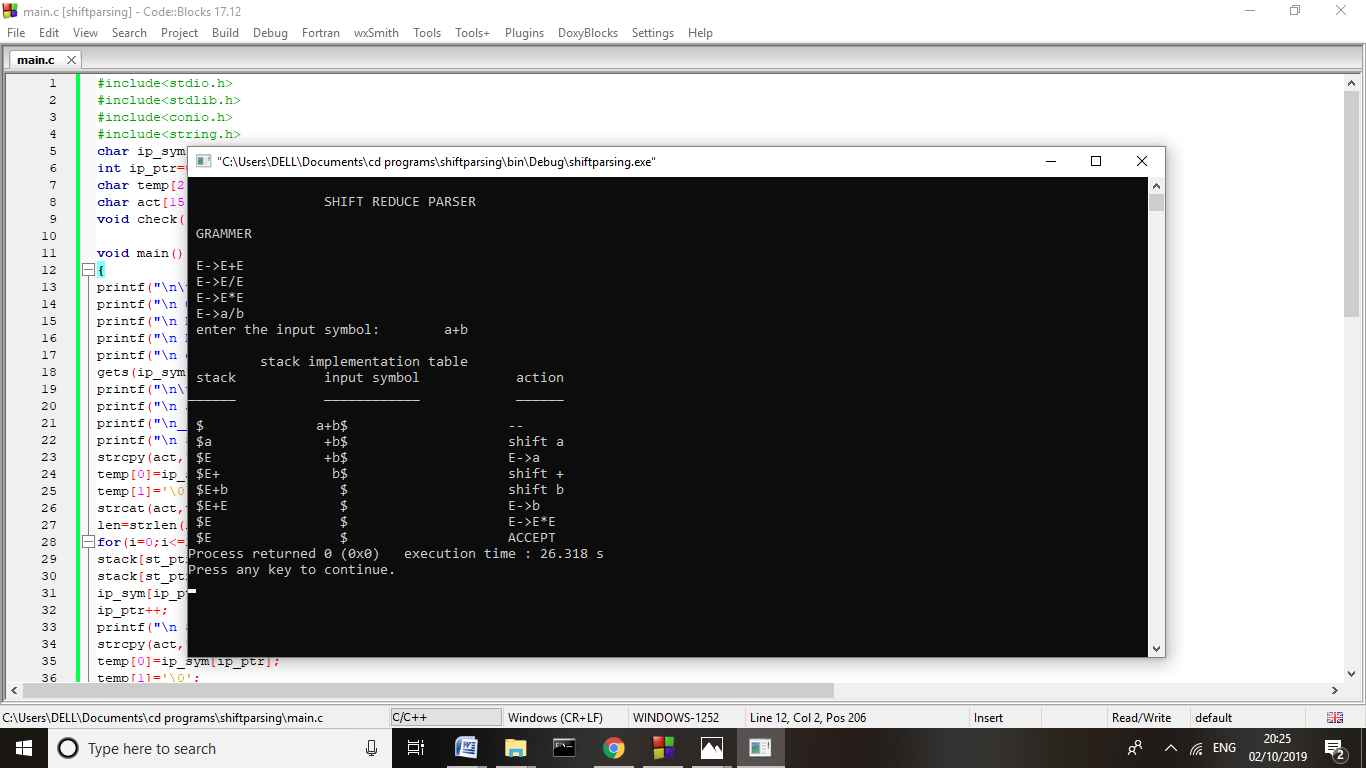
printf("\n%s\t\t\t%s\t\t reject",stack,ip\_sym);

exit(0);

}

return;

}

****