**Exp. No. 11**

Implement a C program to perform symbol table operations.

**Program:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int cnt=0;

struct symtab

{

char label[20];

int addr;

}

sy[50];

void insert();

int search(char \*);

void display();

void modify();

int main()

{

int ch,val;

char lab[10];

do

{

printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

display();

break;

case 3:

printf("enter the label");

scanf("%s",lab);

val=search(lab);

if(val==1)

printf("label is found");

else

printf("label is not found");

break;

case 4:

modify();

break;

case 5:

exit(0);

break;

}

}while(ch<5);

}

void insert()

{

int val;

char lab[10];

int symbol;

printf("enter the label");

scanf("%s",lab);

val=search(lab);

if(val==1)

printf("duplicate symbol");

else

{

strcpy(sy[cnt].label,lab);

printf("enter the address");

scanf("%d",&sy[cnt].addr);

cnt++;

}

}

int search(char \*s)

{

int flag=0,i; for(i=0;i<cnt;i++)

{

if(strcmp(sy[i].label,s)==0)

flag=1;

}

return flag;

}

void modify()

{

int val,ad,i;

char lab[10];

printf("enter the labe:");

scanf("%s",lab);

val=search(lab);

if(val==0)

printf("no such symbol");

else

{

printf("label is found \n");

printf("enter the address");

scanf("%d",&ad);

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

{

if(strcmp(sy[i].label,lab)==0)

sy[i].addr=ad;

}

}

}

void display()

{

int i;

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

printf("%s\t%d\n",sy[i].label,sy[i].addr);

}

**Output:**

1.insert

2.display

3.search

4.modify

5.exit

1

enter the label a

enter the address 100

1.insert

2.display

3.search

4.modify

5.exit

2

a 100

1.insert

2.display

3.search

4.modify

5.exit

3

enter the label a

label is found

1.insert

2.display

3.search

4.modify

5.exit

4

enter the labe: a

label is found

enter the address 200

1.insert

2.display

3.search

4.modify

5.exit

2

a 200

1.insert

2.display

3.search

4.modify

5.exit

5

**Exp. No. 12**

Write a C program to construct recursive descent parsing for the given grammar

E → TE’

E’ → +TE’ / ∈

T → FT’

T’ → \*FT’ / ∈

F → ( E ) / id

**Program:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char input[100];

int i,l;

void main()

{

//clrscr();

printf("\nRecursive descent parsing for the following grammar\n"); printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->\*FT'/@\nF->(E)/ID\n"); printf("\nEnter the string to be checked:"); gets(input);

if(E())

{

if(input[i+1]=='\0')

printf("\nString is accepted");

else

printf("\nString is not accepted");

}

else

printf("\nString not accepted");

getch();

}

E()

{

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

EP()

{

if(input[i]=='+')

{

i++;

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

T()

{

if(F())

{

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

TP()

{

if(input[i]=='\*')

{

i++;

if(F())

{

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

F()

{

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

{

i++;

if(E())

{

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

{

i++;

return(1);

}

else

return(0);

}

else

return(0);

}

else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')

{

i++;

return(1);

}

else

return(0);

}

**Output:**

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->\*FT'/@

F->(E)/ID

Enter the string to be checked: (a+b)\*c

String is accepted

Enter the string to be checked: a/c+d

String is not accepted

**Exp. No. 13**

Write a C program to implement either Top Down parsing technique or Bottom Up Parsing technique to check whether the given input string is satisfying the grammar or not.

**Program:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main() {

char string[50];

int flag,count=0;

printf("The grammar is: S->aS, S->Sb, S->ab\n");

printf("Enter the string to be checked:\n");

gets(string);

if(string[0]=='a') {

flag=0;

for (count=1;string[count-1]!='\0';count++) {

if(string[count]=='b') {

flag=1;

continue;

} else if((flag==1)&&(string[count]=='a')) {

printf("The string does not belong to the specified grammar");

break;

} else if(string[count]=='a')

continue; else if((flag==1)&&(string[count]='\0')) {

printf("String not accepted…..!!!!");

break;

} else {

printf("String accepted");

}

}

}

}

**Output:**

The grammar is: S->aS, S->Sb, S->ab

Enter the string to be checked:

abb

String accepted

**Exp. No. 14**

Implement the concept of Shift reduce parsing in C Programming.

**Program:**

#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(); int 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

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;

}

**Output:**

SHIFT REDUCE PARSER

GRAMMER

E->E+E

E->E/E

E->E\*E

E->a/b

enter the input symbol: a+b

stack implementation table

stack input symbol action

$ a+b$ --

$a +b$ shift a

$E +b$ E->a

$E+ b$ shift+

$E+b $ shiftb

$E+E $ E->b

$E $ E->E+E

$E $ ACCEPT

**Exp. No. 15**

Write a C Program to implement the operator precedence parsing.

**Program:**

#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;");

}

**Output:**

Enter the string

i\*(i+i)\*i

STACK INPUT ACTION

$i \*(i+i)\*i$ Shift

$E \*(i+i)\*i$ Reduced: E->i

$E\* (i+i)\*i$ Shift

$E\*( i+i)\*i$ Shift

$E\*(i +i)\*i$ Shift

$E\*(E +i)\*i$ Reduced: E->i

$E\*(E+ i)\*i$ Shift

$E\*(E+i )\*i$ Shift

$E\*(E+E )\*i$ Reduced: E->i

$E\*(E )\*i$ Reduced: E->E+E

$E\*(E) \*i$ Shift

$E\*E \*i$ Reduced: E->)E(

$E \*i$ Reduced: E->E\*E

$E\* i$ Shift

$E\*i $ Shift

$E\*E $ Reduced: E->i

$E $ Reduced: E->E\*E

$E$ Shift

$E$ Shift

Accepted;

**Exp. No. 16**

Write a C Program to Generate the Three address code representation for the given input statement.

**Program:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

struct three

{

char data[10],temp[7];

}s[30];

int main()

{

char d1[7],d2[7]="t";

int i=0,j=1,len=0;

FILE \*f1,\*f2;

//clrscr();

f1=fopen("sum.txt","r");

f2=fopen("out.txt","w");

while(fscanf(f1,"%s",s[len].data)!=EOF)

len++;

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

strcpy(d1,"");

strcpy(d2,"t");

if(!strcmp(s[3].data,"+"))

{

fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

else if(!strcmp(s[3].data,"-"))

{

fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

for(i=4;i<len-2;i+=2)

{

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

if(!strcmp(s[i+1].data,"+"))

fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);

else if(!strcmp(s[i+1].data,"-"))

fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);

strcpy(d1,"");

strcpy(d2,"t");

j++;

}

fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);

fclose(f1);

fclose(f2);

getch();

}

**Output:**

**Input:** sum.txt

out = in1 + in2 + in3 - in4

**Output:** out.txt

t1=in1+in2

t2=t1+in3

t3=t2-in4

out=t3

**Exp. No. 17**

Write a C program for implementing a Lexical Analyzer to Scan and Count the number of characters, words, and lines in a file.

**Program:**

#include <stdio.h>

int main()

{

char str[100];//input string with size 100

int words=0,newline=0,characters=0; // counter variables

scanf("%[^~]",&str);//scanf formatting

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

{

if(str[i] == ' ')

{

words++;

}

else if(str[i] == '\n')

{

newline++;

words++;//since with every next line new words start. corner case 1

}

else if(str[i] != ' ' && str[i] != '\n'){

characters++;

}

}

if(characters > 0)//Corner case 2,3.

{

words++;

newline++;

}

printf("Total number of words : %d\n",words);

printf("Total number of lines : %d\n",newline);

printf("Total number of characters : %d\n",characters);

return 0;

}

**Output:**

void main()

{

int a;

int b;

a = b + c;

c = d \* e;

}

Total number of words : 18

Total number of lines : 7

**Exp. No. 18**

Write a C program to implement the back end of the compiler.

**Program:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int n,i,j;

char a[50][50];

printf("enter the no: intermediate code:");

scanf("%d",&n);

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

{

printf("enter the 3 address code:%d:",i+1);

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

{

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

}

}

printf("the generated code is:");

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

{

printf("\n mov %c,R%d",a[i][3],i);

if(a[i][4]=='-')

{

printf("\n sub %c,R%d",a[i][5],i);

}

if(a[i][4]=='+')

{

printf("\n add %c,R%d",a[i][5],i);

}

if(a[i][4]=='\*')

{

printf("\n mul %c,R%d",a[i][5],i);

}

if(a[i][4]=='/')

{

printf("\n div %c,R%d",a[i][5],i);

}

printf("\n mov R%d,%c",i,a[i][1]);

printf("\n");

}

return 0;

}

**Output:**

enter the no: intermediate code:2

enter the 3 address code:1:a=b+c

enter the 3 address code:2:d=n\*d

the generated code is:

mov b,R0

add c,R0

mov R0,a

mov n,R1

mul d,R1

mov R1,d

**Exp. No. 19**

Write a C program to compute LEADING( ) – operator precedence parser for the given grammar

E → E + T | T

T → T \* F | F

F → ( E ) | id

**Program:**

#include<conio.h>

#include<stdio.h>

char arr[18][3] ={{'E', '+', 'F'},{'E', '\*', 'F'},{'E', '(', 'F'}, {'E', ')', 'F'},{'E', 'i', 'F'},{'E', '$', 'F'},

{'F', '+', 'F'},{'F', '\*', 'F'},{'F', '(', 'F'},{'F', ')', 'F'},{'F', 'i', 'F'},{'F', '$', 'F'}, {'T', '+', 'F'},

{'T', '\*', 'F'}, {'T', '(', 'F'},{'T', ')', 'F'},{'T', 'i', 'F'},{'T', '$', 'F'}};

char prod[] = "EETTFF";

char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0'}, {'T', '\*', 'F'}, {'F', '\0'}, {'(', 'E', ')'}, {'i', '\0'}};

char stack [5][2];

int top = -1;

void install(char pro, char re) {

int i;

for (i = 0; i < 18; ++i) {

if (arr[i][0] == pro && arr[i][1] == re) {

arr[i][2] = 'T';

break;

}

}

++top;

stack[top][0] = pro;

stack[top][1] = re;

}

int main() {

int i = 0, j;

char pro, re, pri = ' ';

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

for (j = 0; j < 3 && res[i][j] != '\0'; ++j) {

if (res[i][j] == '+' || res[i][j] == '\*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' || res[i][j] == '$') {

install(prod[i], res[i][j]);

break;

}

}

}

while (top >= 0) {

pro = stack[top][0];

re = stack[top][1];

--top;

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

if (res[i][0] == pro && res[i][0] != prod[i]) {

install(prod[i], re);

}

}

}

for (i = 0; i < 18; ++i) {

printf("\n\t");

for (j = 0; j < 3; ++j)

printf("%c\t", arr[i][j]);

}

getch();

printf("\n\n");

for (i = 0; i < 18; ++i) {

if (pri != arr[i][0]) {

pri = arr[i][0];

printf("\n\t%c -> ", pri);

}

if (arr[i][2] == 'T')

printf("%c ", arr[i][1]);

}

getch();

}

**Output:**

E + T

E \* T

E ( T

E ) F

E i T

E $ F

F + F

F \* F

F ( T

F ) F

F i T

F $ F

T + F

T \* T

T ( T

T ) F

T i T

T $ F

E -> + \* ( i

F -> ( i

T -> \* ( i

**Exp. No. 20**

Write a C program to compute TRAILING( ) – operator precedence parser for the given grammar

E → E + T | T

T → T \* F | F

F → ( E ) | id

**Program:**

#include<conio.h>

#include<stdio.h>

char arr[18][3] ={{'E', '+', 'F'}, {'E', '\*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}, {'E', 'i', 'F'},

{'E', '$', 'F'}, {'F', '+', 'F'}, {'F', '\*', 'F'}, {'F', '(', 'F'}, {'F', ')', 'F'}, {'F', 'i', 'F'},

{'F', '$', 'F'}, {'T', '+', 'F'}, {'T', '\*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'},

{'T', '$', 'F'},

};

char prod[6] = "EETTFF";

char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0', '\0'}, {'T', '\*', 'F'}, {'F', '\0', '\0'}, {'(', 'E', ')'}, {'i', '\0', '\0'},};

char stack [5][2];

int top = -1;

void install(char pro, char re) {

int i;

for (i = 0; i < 18; ++i) {

if (arr[i][0] == pro && arr[i][1] == re) {

}

}

++top;

arr[i][2] = 'T';

stack[top][0] = pro;

stack[top][1] = re;

}

int main() {

int i = 0, j;

char pro, re, pri = ' ';

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

for (j = 2; j >= 0; --j) {

if (res[i][j] == '+' || res[i][j] == '\*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' || res[i][j] == '$') {

install(prod[i], res[i][j]);

break;

} else if (res[i][j] == 'E' || res[i][j] == 'F' || res[i][j] == 'T') {

if (res[i][j - 1] == '+' || res[i][j - 1] == '\*' || res[i][j - 1] == '(' || res[i][j -

1] == ')' || res[i][j - 1] == 'i' || res[i][j - 1] == '$') {

install(prod[i], res[i][j - 1]);

break;

}

}

}

}

while (top >= 0) {

pro = stack[top][0];

re = stack[top][1];

--top;

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

for (j = 2; j >= 0; --j) {

if (res[i][0] == pro && res[i][0] != prod[i]) {

install(prod[i], re);

break;

} else if (res[i][0] != '\0') break;

}

}

}

for (i = 0; i < 18; ++i) {

printf("\n\t");

for (j = 0; j < 3; ++j)

printf("%c\t", arr[i][j]);

}

printf("\n\n");

for (i = 0; i < 18; ++i) {

if (pri != arr[i][0]) {

pri = arr[i][0];

printf("\n\t%c -> ", pri);

}

if (arr[i][2] == 'T')

printf("%c ", arr[i][1]);}

}

**Output:**

E + F

E \* F

E ( F

E ) F

E i F

E $ F

F + F

F \* F

F ( F

F ) F

F i F

F $ F

T + F

T \* F

T ( F

T ) F

T i F

T $ F

E ->

F ->

T ->

ss