Lexical Analyzer

#include<stdio.h>

#include<string.h>

#include<ctype.h>

int k\_count=0, i\_count=0, str\_count=0;

void keyword(char str[10])

{

if(strcmp("for",str)==0||strcmp("printf",str)==0||

strcmp("while",str)==0||strcmp("do",str)==0||

strcmp("int",str)==0||strcmp("float",str)==0||

strcmp("char",str)==0||strcmp("double",str)==0||

strcmp("static",str)==0||strcmp("switch",str)==0||

strcmp("case",str)==0||strcmp("include",str)==0)

{

printf("\n%s is a keyword\n",str);

k\_count++;

}

else

{

printf("\n%s is an identifier\n", str);

i\_count++;

}

}

void main()

{

FILE \*f1,\*f2,\*f3;

char c,str[100],str1[10];

int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0,count=0,s\_count=0;

/\*printf("Enter the C program: ");

f1=fopen("input.txt","w");

while((c=getchar()!='$'))

putc(c,f1);

fclose(f1);\*/

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

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

f3=fopen("special\_char.txt", "w");

if (!f1 || !f2 || !f3)

{

printf("Error opening file.\n");

// return 0;

}

printf("\nString literals found:\n\n");

while((c = getc(f1))!=EOF)

{

if(isdigit(c))

{

tokenvalue = c-'0';

c=getc(f1);

while(isdigit(c))

{

tokenvalue\*=10+c-'0';

c=getc(f1);

}

num[i++]=tokenvalue;

ungetc(c,f1);

count = i;

}

else if(isalpha(c))

{

putc(c,f2);

c=getc(f1);

while(isdigit(c)||isalpha(c)||c=='\_'||c=='$')

{

putc(c,f2);

c=getc(f1);

}

putc(' ',f2);

ungetc(c,f1);

}

else if(c == '"')

{

k=0;

str[k++] = c;

putc(c,f3);

c = getc(f1);

while(c != '"' && c != EOF)

{

str[k++] = c;

c = getc(f1);

}

str[k++] = c;

str[k] = '\0';

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

putc(c,f3);

str\_count++;

}

else if(c==' '||c=='\t')

printf(" ");

else if(c=='\n')

lineno++;

else

putc(c,f3);

}

fclose(f2);

fclose(f3);

fclose(f1);

printf("\nThe no's in the program are: ");

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

printf("%d\t", num[i]);

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

{

printf("\n");

f2=fopen("identifier.txt","r");

k=0;

printf("\nThe keywords & identifiers are: ");

while((c=getc(f2))!=EOF)

{

if(c!=' ')

str[k++]=c;

else

{

str[k] = '\0';

keyword(str);

k=0;

}

}

fclose(f2);

f3 = fopen("special\_char.txt", "r");

printf("\nThe special characters are: ");

while((c=getc(f3))!=EOF)

{

printf("%c", c);

s\_count++;

}

printf("\n");

fclose(f3);

printf("\nTotal no: of lines are: %d\n", lineno);

printf("\nCount of numbers are: %d\n", count);

printf("\nTotal no: of keywords are: %d\n", k\_count);

printf("\nTotal no. of identifiers are: %d\n", i\_count);

printf("\nTotal no. of special characters are: %d\n", s\_count);

printf("\nTotal no. of strings are: %d\n", str\_count);

}

}

input.txt

#include<stdio.h>

void main()

{

printf("Hello World");

}

identifier.txt

include stdio h void main printf

special\_char.txt

#<.>(){("");}

Operator Precedence Parser

#include<stdio.h>

#include<string.h>

void main()

{

char stack[20],ip[20],opt[10][10][1],ter[10];

int i,j,k,n,top=0,row,col;

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

{

stack[i]=0;

ip[i]=0;

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

{

opt[i][j][1]=0;

}

}

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

scanf("%d",&n);

printf("\nEnter the terminals:");

scanf("%s",ter);

printf("\nEnter the table values:\n");

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

{

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

{

printf("Enter the value for %c %c:",ter[i],ter[j]);

scanf("%s",opt[i][j]);

}

}

printf("\nOPERATOR PRECEDENCE TABLE:\n");

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

{

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

}

printf("\n");

printf("\n");

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

{

printf("\n%c |", ter[i]);

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

{

printf("\t%c",opt[i][j][0]);

}

}

stack[top]='$';

printf("\n\nEnter the input string(append with $):");

scanf("%s",ip);

i=0;

printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");

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

while(i<=strlen(ip))

{

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

{

if(stack[top]==ter[k]) row=k;

if(ip[i]==ter[k]) col=k;

}

if((stack[top]=='$')&&(ip[i]=='$'))

{

printf("String is ACCEPTED"); break;

}

else if((opt[row][col][0]=='<') ||(opt[row][col][0]=='='))

{

stack[++top]=opt[row][col][0];

stack[++top]=ip[i];

ip[i]=' ';

printf("Shift %c",ip[i]);

i++;

}

else

{

if(opt[row][col][0]=='>')

{

while(stack[top]!='<')

{

top = top - 1;

}

top=top-1; printf("Reduce");

}

else

{

printf("\nString is not accepted"); break;

}

}

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

}

getchar();

printf("\n");

}

First and Follow

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

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;

//clrscr();

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

scanf("%d",&n);

printf("Enter the productions:\n");

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

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

do{

m=0;

printf("Enter the elements whose first & follow is to be found:");

scanf("%c",&c);

first(c);

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

for(i=0;i<m-1;i++)

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

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

printf("}\n");

strcpy(f, " ");

//flushall();

m=0;

follow(c);

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

for(i=0;i<m-1;i++)

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

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

printf("}\n");

printf("Continue(0/1)?");

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

}while(z==1);

return 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[k][0]);

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

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

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

}

}

}

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]);

}

}

}

}

Shift Reduce Parsing

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<string.h>

char ip\_sym[15], stack[15], stack2[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\tSHIFT REDUCE PARSER\n");

printf("\nGRAMMAR\n");

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

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

printf("\nEnter the input symbol:\t");

scanf("%s",ip\_sym);

printf("\n\tstack implementation table");

printf("\nstack\t\tinput symbol\t\taction");

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();

printf("\n");

}

void check()

{

int flag=0;

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

temp2[1] = '\0';

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

{

stack[st\_ptr] = 'E';

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

flag=1;

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

{

strcpy(stack2, stack);

strcpy(stack,"E");

st\_ptr=0;

if(!strcmp(stack2,"E+E"))

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

else

if(!strcmp(stack2,"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(!strcmp(stack,"E")&&ip\_ptr==len)

{

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

getchar();

exit(0);

}

if(flag==0)

{

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

exit(0);

}

return;

}

Recursive Descent 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();

}

}

Intermediate Code Generation

#include<stdio.h>

#include<string.h>

#include<ctype.h>

char tempvariables[] = {'z', 'y', 'x', 'w', 'v', 'u', 't'};

int length, top=-1, count=0, tvar=0;

char input[20], tempinput[20];

char prearray[20];

char stack[20];

char threeaddress[10][10];

char concatarray[20];

char \*strrev(char \*str)

{

if(!str||!\*str) return str;

int i=strlen(str) - 1, j=0;

char ch;

while(i>j)

{

ch=str[i];

str[i] = str[j];

str[j] = ch;

i--;

j++;

}

return str;

}

int prec(char op)

{

switch(op)

{

case '+': return 1;

break;

case '-': return 1;

break;

case '\*': return 2;

break;

case '/': return 2;

break;

}

}

int isoperator(char sym)

{

if(sym=='+'||sym=='-'||sym=='\*'||sym=='/')

return 1;

else

return 0;

}

void push(char sym)

{

top++;

stack[top]=sym;

}

char pop()

{

top--;

return(stack[top + 1]);

}

void display()

{

int i;

printf("\nStack");

for(i=top;i>-1;i--)

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

}

void del(char sym, int pos)

{

int j, c=0, k;

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

tempinput[j] = prearray[j];

tempinput[j] = tempvariables[tvar - 1];

k=j+1;

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

tempinput[k++] = prearray[j];

strcpy(prearray, tempinput);

length = strlen(prearray);

}

void prefix()

{

int i, k=0;

char popval;

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

{

if(isalpha(input[i])||isdigit(input[i]))

prearray[k++] = input[i];

else

{

if(top == -1)

push(input[i]);

else

{

while(prec(stack[top])>=prec(input[i]))

prearray[k++] = pop();

push(input[i]);

}

}

}

if(top!= -1)

for(i=top;i>-1;i--)

prearray[k++] = pop();

}

void generator(char op, char sym1, char sym2)

{

int len=0;

concatarray[len++] = tempvariables[tvar++];

concatarray[len++] = '=';

concatarray[len++] = sym1;

concatarray[len++] = op;

concatarray[len++] = sym2;

strcpy(threeaddress[count++], concatarray);

}

void main()

{

int i;

printf("\nEnter the input expression: ");

scanf("%s", input);

length=strlen(input);

strcpy(input, strrev(input));

prefix();

strcpy(prearray, strrev(prearray));

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

if(isoperator(prearray[i]))

if((isalpha(prearray[i+1])||isdigit(prearray[i+1]))&&

(isalpha(prearray[i+2])||isdigit(prearray[i+2])))

{

generator(prearray[i],prearray[i+1],prearray[i+2]);

del(prearray[i], i);

i = -1;

}

printf("\nThree Address Code for the expression:\n\n");

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

printf("%s\n", threeaddress[i]);

}