**CD LAB COPY**

**MADDI HEMANTH**

**AP20110010296 CSE-E**

**Week1:**

**Aim**: C program to accept a string ending with same symbol

**Code:**

#include<stdio.h>

#define max 100

int main() {

char str[max],f='a';

int i;

printf("enter the string to be checked: ");

scanf("%s",str);

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

switch(f) {

case 'a': if(str[i]=='0') f='b';

else if(str[i]=='1') f='d';

break;

case 'b': if(str[i]=='0') f='c';

else if(str[i]=='1') f='d';

break;

case 'c': if(str[i]=='0') f='c';

else if(str[i]=='1') f='d';

break;

case 'd': if(str[i]=='0') f='b';

else if(str[i]=='1') f='e';

break;

case 'e': if(str[i]=='0') f='b';

else if (str[i]=='1') f='e';

}

}

if(f=='c'){

printf("\nString is accepted");

}

else if(f=='e'){

printf("\nString is accepted");

}

else{

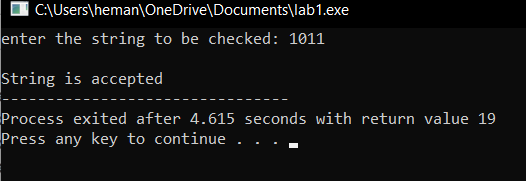
printf("\nString is not accepted");

return 0;

}

}

**Input,Output:**



**Week 2:**

**Aim:** Implementation of lexical analysis in C

**Code:**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

bool isValidDelimiter(char ch) {

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == ',' || ch == ';' || ch == '>' ||

ch == '<' || ch == '=' || ch == '(' || ch == ')' ||

ch == '[' || ch == ']' || ch == '{' || ch == '}')

return (true);

return (false);

}

bool isValidOperator(char ch){

if (ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == '>' || ch == '<' ||

ch == '=')

return (true);

return (false);

}

bool isvalidIdentifier(char\* str){

if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||

str[0] == '3' || str[0] == '4' || str[0] == '5' ||

str[0] == '6' || str[0] == '7' || str[0] == '8' ||

str[0] == '9' || isValidDelimiter(str[0]) == true)

return (false);

return (true);

}

bool isValidKeyword(char\* str) {

if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str,

"while") || !strcmp(str, "do") || !strcmp(str, "break") ||

!strcmp(str, "continue") || !strcmp(str, "int")

|| !strcmp(str, "double") || !strcmp(str, "float") ||

!strcmp(str, "return") || !strcmp(str, "char") || !strcmp(str,

"case") || !strcmp(str, "char")

|| !strcmp(str, "sizeof") || !strcmp(str, "long") ||

!strcmp(str, "short") || !strcmp(str, "typedef") || !strcmp(str,

"switch") || !strcmp(str, "unsigned")

|| !strcmp(str, "void") || !strcmp(str, "static") ||

!strcmp(str, "struct") || !strcmp(str, "goto"))

return (true);

return (false);

}

bool isValidInteger(char\* str) {

int i, len = strlen(str);

if (len == 0)

return (false);

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

if (str[i] != '0' && str[i] != '1' && str[i] != '2'&& str[i] != '3'

&& str[i] != '4' && str[i] != '5'

&& str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9'

|| (str[i] == '-' && i > 0))

return (false);

}

return (true);

}

bool isRealNumber(char\* str) {

int i, len = strlen(str);

bool hasDecimal = false;

if (len == 0)

return (false);

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

if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3'

&& str[i] != '4' && str[i] != '5' && str[i] != '6' && str[i] != '7'

&& str[i] != '8'

&& str[i] != '9' && str[i] != '.' || (str[i] == '-' && i > 0))

return (false);

if (str[i] == '.')

hasDecimal = true;

}

return (hasDecimal);

}

char\* subString(char\* str, int left, int right) {

int i;

char\* subStr = (char\*)malloc( sizeof(char) \* (right - left +2));

for (i = left; i <= right; i++)

subStr[i - left] = str[i];

subStr[right - left + 1] = '\0';

return (subStr);

}

void detectTokens(char\* str) {

int left = 0, right = 0;

int length = strlen(str);

while (right <= length && left <= right) {

if (isValidDelimiter(str[right]) == false)

right++;

if (isValidDelimiter(str[right]) == true && left == right) {

if (isValidOperator(str[right]) == true)

printf("Valid operator : '%c'\n", str[right]);

right++;

left = right;

} else if (isValidDelimiter(str[right]) == true && left != right

|| (right == length && left != right)) {

char\* subStr = subString(str, left, right - 1);

if (isValidKeyword(subStr) == true)

printf("Valid keyword : '%s'\n", subStr);

else if (isValidInteger(subStr) == true)

printf("Valid Integer : '%s'\n", subStr);

else if (isRealNumber(subStr) == true)

printf("Real Number : '%s'\n", subStr);

else if (isvalidIdentifier(subStr) == true

&& isValidDelimiter(str[right - 1]) == false)

printf("Valid Identifier : '%s'\n", subStr);

else if (isvalidIdentifier(subStr) == false

&& isValidDelimiter(str[right - 1]) == false)

printf("Invalid Identifier : '%s'\n", subStr);

left = right;

}

}

return;

}

int main(){

char str[100] = "int x=1; ";

printf("The Program is : '%s' \n", str);

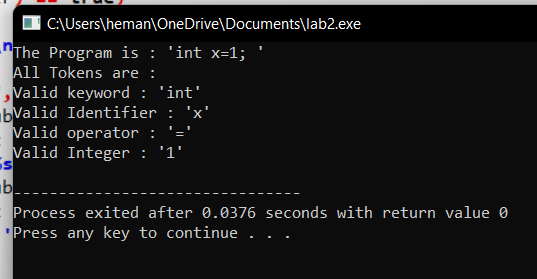
printf("All Tokens are : \n");

detectTokens(str);

return (0);

}

**Input,Output:**

****

**Week3:**

**Aim:** Identification of Vowels and Consonants

**Code:**

%option noyywrap

%{

#include<stdio.h>

%}

%%

[aeiouAEIOU] {printf("Vowel\n");}

[a-zA-z] {printf("consonent\n");}

{printf("special symbol\n");}

%%

int main()

{

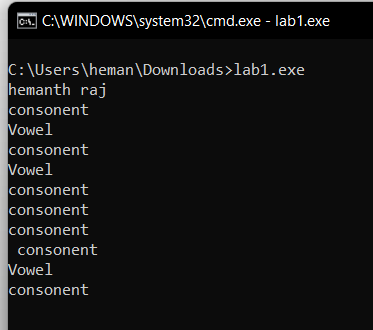
yylex();

getchar();

return 0;

}

**Input,Output:**

****

**Aim:** Count number of vowels and consonants

**Code:**

%{

    int vow\_count=0;

    int const\_count =0;

%}

%%

[aeiouAEIOU] {vow\_count++;}

[a-zA-Z] {const\_count++;}

%%

int yywrap(){}

int main()

{

    printf("Enter the string of vowels and consonants:");

    yylex();

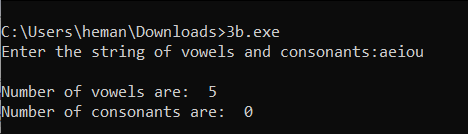
    printf("Number of vowels are: %d\n", vow\_count);

    printf("Number of consonants are: %d\n", const\_count);

    return 0;

}

**Input,Output:**

****

**Aim:** Count the number of Lines in given input

**Code:**

%{

#include<stdio.h>

int sc=0,wc=0,lc=0,cc=0;

%}

%%

[\n] { lc++; cc+=yyleng;}

[ \t] { sc++; cc+=yyleng;}

[^\t\n ]+ { wc++; cc+=yyleng;}

%%

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

{

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

yylex();

printf("The number of lines=%d\n",lc);

printf("The number of spaces=%d\n",sc);

printf("The number of words=%d\n",wc);

printf("The number of characters are=%d\n",cc);

}

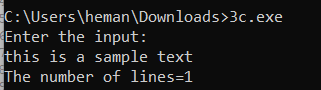
int yywrap( )

{

return 1;

}

**Input,Output:**

****

**Aim:**Recognize strings ending with 00

**Code:**

**%%**

**[0-9]\*00{printf(“string accepted”);**

**[0-9]\*{printf(“string rejected”);}**

**%%**

**main()**

**{**

**yylex();**

**}**

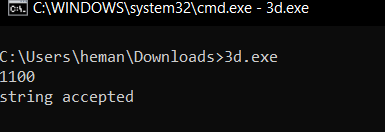
**int yywrap()**

**{**

**return 1;**

**}**

**Input,Output:**

****

**Aim:To recognize a string with 3 consecutive 0’s**

**Code:**

**%%**

**[0-9]\*000[0-9]\* { printf(“string accepted”);}**

**[0-9]\* { printf(“string rejected”);}**

**%%**

**main()**

**{**

**yylex();**

**}**

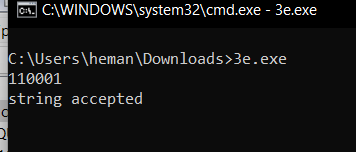
**int yywrap()**

**{**

**return 1;**

**}**

**Input,Output:**

****

**Week 4:**

**Aim:** Implementation of lexical analyser using LEX

**Code**:

%{

int COMMENT=0;

int cnt=0;

%}

identifier [a-zA-Z][a-zA-Z0-9]\*

%%

#.\* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}

int |

float |

char |

double |

while |

for |

do |

if |

break |

continue |

void |

switch |

case |

long |

struct |

const |

typedef |

return |

else |

goto {printf("\n\t%s is a KEYWORD",yytext);}

"/\*" {COMMENT = 1;}

"\*/" {COMMENT = 0; cnt++;}

{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}

\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}

\} {if(!COMMENT) printf("\n BLOCK ENDS");}

{identifier}(\[[0-9]\*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}

\".\*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}

[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}

\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\( ECHO;

= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |

\>= |

\< |

== |

\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%

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

{

if (argc > 1)

{

FILE \*file;

file = fopen(argv[1],"r");

if(!file)

{

printf("could not open %s \n",argv[1]);

exit(0);

}

yyin = file;

}

yylex();

printf("\n\n Total No.Of comments are %d",cnt);

return 0;

}

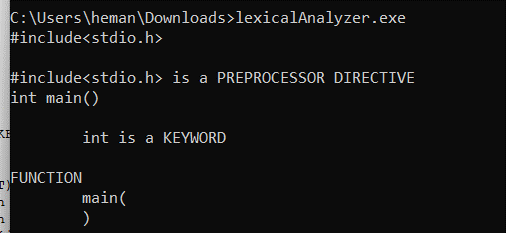
int yywrap()

{

return 1;

}

**Input,Output:**



**Week 5:**

**Aim:Lexical analyzer of a given code**

**Code:**

%{

int COMMENT=0;

int cnt=0;

%}

identifier [a-zA-Z][a-zA-Z0-9]\*

%%

#.\* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}

int |

float |

char |

double |

while |

for |

do |

if |

break |

continue |

void |

switch |

case |

long |

struct |

const |

typedef |

return |

else |

goto {printf("\n\t%s is a KEYWORD",yytext);}

"/\*" {COMMENT = 1;}

"\*/" {COMMENT = 0; cnt++;}

{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}

\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}

\} {if(!COMMENT) printf("\n BLOCK ENDS");}

{identifier}(\[[0-9]\*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}

\".\*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}

[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}

\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\( ECHO;

= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |

\>= |

\< |

== |

\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%

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

{

if (argc > 1)

{

FILE \*file;

file = fopen(argv[1],"r");

if(!file)

{

printf("could not open %s \n",argv[1]);

exit(0);

}

yyin = file;

}

yylex();

printf("\n\n Total No.Of comments are %d",cnt);

return 0;

}

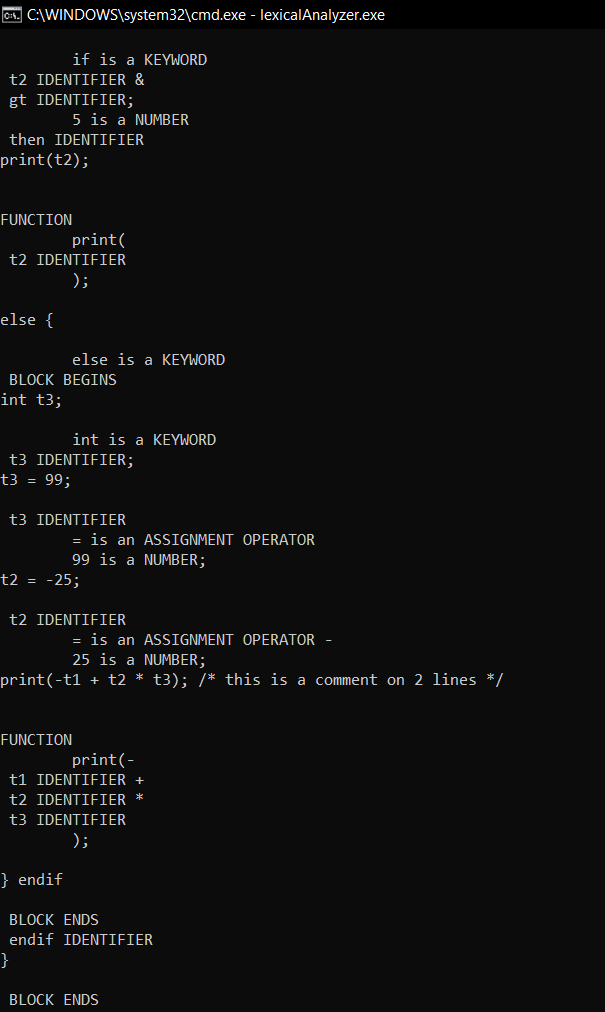
int yywrap()

{

return 1;

}

**Input,Output:**

****

**Week 6:**

**Aim:**Recursive Descent Parser

**Code:**

#include <stdio.h>

#include <string.h>

int S(), Ldash(), L(); char \*ip; char string[50];

int main()

{

printf("Enter the String : "); scanf("%s", string); ip = string; printf("\n\nInput\t\tAction\n---------------------\n"); if(S())

{

printf("\n--------------------------\n"); printf("\n String is successfully parsed\n");

}

else

{

printf("\n--------------------------\n"); printf("Error in parsing String\n");

}

}

int L()

{

printf("%s\t\tL->SL' \n", ip); if(S())

{

if(Ldash())

{

return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

int Ldash()

{

if(\*ip == ',')

{

printf("%s\t\tL'->,SL' \n", ip); ip++; if(S())

{

if(Ldash())

{

return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

else

{

printf("%s\t\tL'->ε \n", ip); return 1;

}

}

int S()

{

if(\*ip == '(')

{

printf("%s\t\tS->(L) \n", ip); ip++; if(L())

{

if(\*ip == ')')

{

ip++; return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

else if(\*ip == 'a')

{

ip++; printf("%s\t\tS->a \n", ip); return 1;

}

else

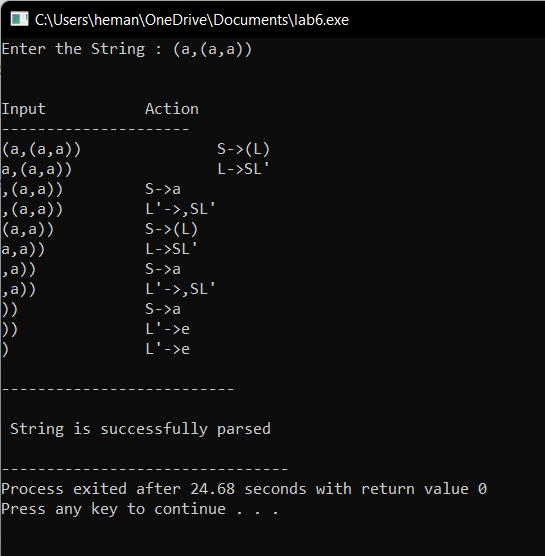
{

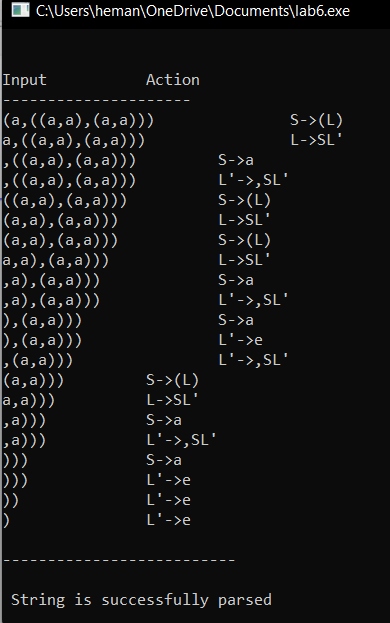
return 0;

}

}

**Input,Output:**





**Week7:**

**Aim:** Predictive parser

**Code:**

#include<stdio.h>

#include<string.h>

int i,j,l,m,n=0,o,p,nv,z=0,x=0;

char str[10],temp,temp2[10],temp3[20],\*ptr;

struct prod

{

char lhs[10],rhs[10][10],ft[10],fol[10];

int n;

}pro[10];

void findter()

{

int k,t;

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

{

if(temp==pro[k].lhs[0])

{

for(t=0;t<pro[k].n;t++)

{

if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )

pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];

else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t][0]<=90 )

{

temp=pro[k].rhs[t][0];

if(temp=='S')

pro[i].ft[strlen(pro[i].ft)]='#';

findter();

}

}

break;

}

}

}

void findfol()

{

int k,t,p1,o1,chk;

char \*ptr1;

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

{

chk=0;

for(t=0;t<pro[k].n;t++)

{

ptr1=strchr(pro[k].rhs[t],temp);

if( ptr1 )

{

p1=ptr1-pro[k].rhs[t];

if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t][p1+1]<=90)

{

for(o1=0;o1<n;o1++)

if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])

{

strcat(pro[i].fol,pro[o1].ft);

chk++;

}

}

else if(pro[k].rhs[t][p1+1]=='\0')

{

temp=pro[k].lhs[0];

if(pro[l].rhs[j][p]==temp)

continue;

if(temp=='S')

strcat(pro[i].fol,"$");

findfol();

chk++;

}

else

{

pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t][p1+1];

chk++;

}

}

}

if(chk>0)

break;

}

}

int main()

{

FILE \*f;

//clrscr();

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

pro[i].n=0;

f=fopen("tab5.txt","r");

while(!feof(f))

{

fscanf(f,"%s",pro[n].lhs);

if(n>0)

{

if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )

{

pro[n].lhs[0]='\0';

fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);

pro[n-1].n++;

continue;

}

}

fscanf(f,"%s",pro[n].rhs[pro[n].n]);

pro[n].n++;

n++;

}

printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");

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

for(j=0;j<pro[i].n;j++)

printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

pro[0].ft[0]='#';

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

{

for(j=0;j<pro[i].n;j++)

{

if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )

{

pro[i].ft[strlen(pro[i].ft)]=pro[i].rhs[j][0];

}

else if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j][0]<=90 )

{

temp=pro[i].rhs[j][0];

if(temp=='S')

pro[i].ft[strlen(pro[i].ft)]='#';

findter();

}

}

}

printf("\n\nFIRST\n");

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

{

printf("\n%s -> ",pro[i].lhs);

for(j=0;j<strlen(pro[i].ft);j++)

{

for(l=j-1;l>=0;l--)

if(pro[i].ft[l]==pro[i].ft[j])

break;

if(l==-1)

printf("%c",pro[i].ft[j]);

}

}

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

temp2[i]=pro[i].lhs[0];

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

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

{

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

{

for(j=0;j<pro[i].n;j++)

{

ptr=strchr(pro[l].rhs[j],temp2[i]);

if( ptr )

{

p=ptr-pro[l].rhs[j];

if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j][p+1]<=90)

{

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

if(pro[o].lhs[0]==pro[l].rhs[j][p+1])

strcat(pro[i].fol,pro[o].ft);

}

else if(pro[l].rhs[j][p+1]=='\0')

{

temp=pro[l].lhs[0];

if(pro[l].rhs[j][p]==temp)

continue;

if(temp=='S')

strcat(pro[i].fol,"$");

findfol();

}

else

pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];

}

}

}

}

printf("\n\nFOLLOW\n");

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

{

printf("\n%s -> ",pro[i].lhs);

for(j=0;j<strlen(pro[i].fol);j++)

{

for(l=j-1;l>=0;l--)

if(pro[i].fol[l]==pro[i].fol[j])

break;

if(l==-1)

printf("%c",pro[i].fol[j]);

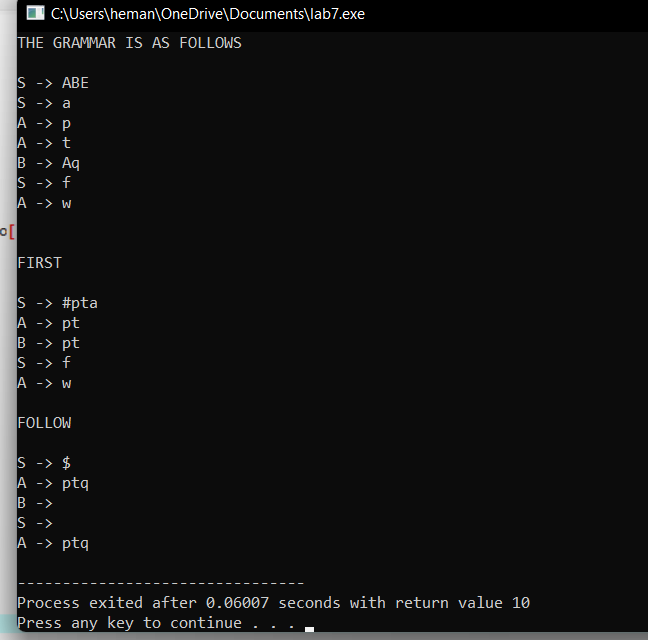
}

}

printf("\n");

}

**Input,Output:**



**Week8:**

**Aim:**Predictive parser

**Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

int i=0,top=0;

char stack[20],ip[20]; void push(char c)

{

if (top>=20) printf("Stack Overflow"); else

stack[top++]=c;

}

void pop(void)

{

if(top<0)

printf("Stack underflow"); else

top--;

}

void error(void)

{

printf("\n\nSyntax Error!!!! String is invalid\n"); exit(0);

}

int main()

{

int n;

printf("The given grammar is\n\n");

printf("E -> TG\n");

printf("G -> +TG | e \n");

printf("T -> FU \n");

printf("U -> \*FU | e\n"); printf("F -> (E) | d \n");

printf("Enter the string to be parsed:\n"); scanf("%s",ip);

n=strlen(ip); ip[n]='$';

ip[n+1]='\0';

push('$');

push('E');

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

{ if(ip[i]=='$' && stack[top-1]=='$')

{

printf("\n\n Successful parsing of string \n"); return 1;

}

else if(ip[i]==stack[top-1])

{

printf("\nmatch of %c ",ip[i]); i++;pop();

}

else

{

if(stack[top-1]=='E')

{

printf(" \n E -> TG"); pop();

push('G');

push('T');

}

else if(stack[top-1]=='G' && ip[i]=='$')

{

printf("\n G-> epsilon"); pop();

}

else if(stack[top-1]=='G')

{

printf("\n G -> +TG"); pop();

push('G');

push('T');

push('+');

}

else if(stack[top-1]=='T' )

{

printf("\n T -> FU"); pop();

push('U');

push('F');

}

else if(stack[top-1]=='U' )

{

printf("\n U -> \*FU"); pop();

push('U');

push('F');

push('\*');

}

else if(stack[top-1]=='U' && ip[i]=='$')

{

printf("\n U -> epsilon"); pop();

}

else if(stack[top-1]=='F' )

{

printf("\n F -> (E) "); pop();

push(')');

push('E');

push('(');

}

else if(stack[top-1]=='F' && ip[i]=='d')

{

printf("\n F -> d "); pop();

push('d');

}

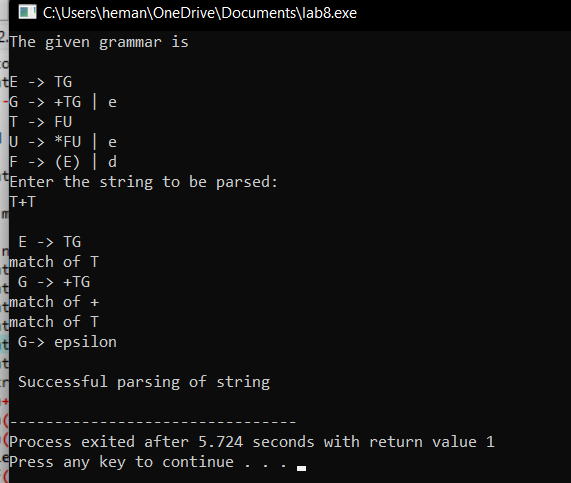
else error();

}

}

}

**Input,Output:**



**Week9:**

**Aim**: Shift Reduce parser

**Code:**

#include<stdio.h>

#include<stdlib.h>

void pop(),push(char),display();

char stack[100]="\0", input[100], \*ip;

int top=-1;

void push(char c)

{

top++;

stack[top]=c;

}

void pop()

{

stack[top]='\0';

top--;

}

void display()

{

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

}

void main()

{

printf("S -> 0S0 | 1S1 | 2\n");

printf("Enter the input string followed by $ \n");

scanf("%s",input);

ip=input;

push('$');

printf("STACK\t BUFFER \t ACTION\n");

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

display();

if(stack[top]=='$' && \*ip=='$'){

printf("Null Input");

exit(0);

}

do

{

if((stack[top]=='S' && stack[top-1]=='$') && (\*(ip)=='$'))

{

display();

printf(" Valid\n\n\n");

break;

}

if(stack[top]=='$')

{

push(\*ip);

ip++;

printf("Shift");

}

else if(stack[top]=='2')

{

display();

pop();

push('S');

printf("Reduce S->2");

}

else if(stack[top]=='2' && stack[top-1]=='S' && stack[top-2]=='2')

{

display();

pop();

pop();

pop();

push('S');

printf("Reduce S->2S2");

}

else if(stack[top]=='1' && stack[top-1]=='S' && stack[top-2]=='1')

{

display();

pop();

pop();

pop();

push('S');

printf("Reduce S->1S1");

}

else if(\*ip=='$')

{

printf(" Invalid\n\n\n");

break;

}

else

{

display();

push(\*ip);

ip++;

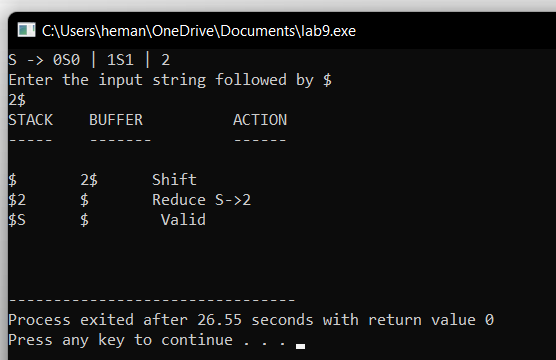
printf("shift");

}

}while(1);

}

**Input,Output:**



**Week 10:**

**Aim:LALR Parser**

**Code:**

**LEX file:**

%{

#include "y.tab.h"

%}

%%

[0-9]+ {yylval=atoi(yytext);

return NUMBER;

}

[\t] ;

\n return 0;

. return yytext[0];

%%

**YACC FILE:**

%{

#include<stdio.h>

%}

%token NUMBER

%%

S: E { printf("The result is =%d\n",$1);}

;

E: E'+'T { $$ = $1 + $3; }

| T { $$ = $1;}

;

T: T'\*'F { $$ = $1 \* $3; }

| F { $$ = $1;}

;

F: '('E')' { $$ = $2;}

| NUMBER { $$ = $1;}

;

%%

int main(){

yyparse();

}

int yywrap(){

return 1;

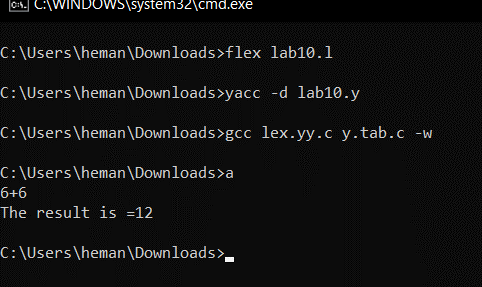
}

void yyerror(char \*s){

printf("Error %s",s);

}

**Input,Output:**



**WEEK 11:**

**AIM:**

Generate quadruples for given arithmetic expression using LEX and YACC.

**CODE:**

**LEX FILE**:

%{

#include<stdio.h>

#include "y.tab.h"

#include<string.h>

%}

%%

[a-z]([a-z]|[0-9])\* { strcpy(yylval.exp,yytext);

return VAR;

}

\t ;

\n return 0;

. return yytext[0];

%%

**YACC FILE:**

%{

#include<stdio.h>

#include<string.h>

struct quad

{

char op[5];

char arg1[10];

char arg2[10];

char result[10];

}QUAD[30];

int i=0,j;

%}

%union

{

char exp[10];

}

%token <exp> VAR

%type <exp> S E T F

%%

S: E

{

printf("\n There are %d quadrupls \n", i);

printf("\n List of Quadruples are: \n");

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

printf("%s\t%s\t%s\t%s\n",QUAD[j].op,QUAD[j].arg1,QUAD[j].arg2,QUAD[j].re

sult);

}

;

E: E'+'T { printf("\n E -> E+T, $1=%s, $3=%s, $$=%s\n",$1,$3,$$);

strcpy(QUAD[i].op,"\*");

strcpy(QUAD[j].arg1,$1);

strcpy(QUAD[j].arg2,$3);

strcpy(QUAD[j].result,$$); i++;

i++;

}

| T { printf("\n E -> T, $1=%s, $$=%s\n",$1,$$);}

;

T: T'\*'F { printf("\n T -> T\*F, $1=%s, $3=%s, $$=%s\n",$1,$3,$$);

strcpy(QUAD[i].op,"\*");

strcpy(QUAD[j].arg1,$1);

strcpy(QUAD[j].arg2,$3);

strcpy(QUAD[j].result,$$);

i++;

}

| F { printf("\n T -> F, $1=%s, $$=%s\n",$1,$$);}

;

F: VAR { printf("\n F -> VAR and $1=%s, $$=%s\n",$1,$$);}

;

%%

main()

{

yyparse();

}

int yywrap(){

return 1;

}

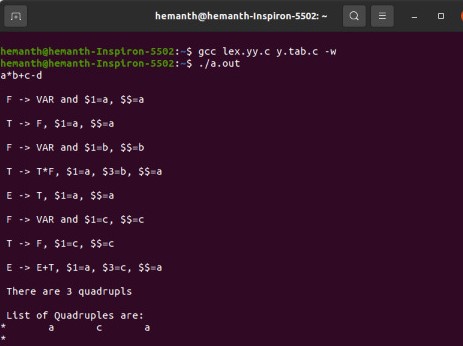
void yyerror(char \*s)

{

printf("%s", s);

}

**OUTPUT:**

****

**WEEK 12:**

**AIM:**

Generate 3-address code for while statement using LEX and YACC.

**CODE:**

**LEX CODE:**

%%

while return WHILE;

[A-Za-z]([A-Za-z]|[0-9])\* return ID;

[0-9]+ {return NUM;}

[ \t] ;

\n yyterminate();

. return yytext[0];

%%

**YACC CODE:**

%token ID NUM WHILE

%right '='

%left '+' '-'

%left '\*' '/'

%left MINUS

%%

S : WHILE{L1();} '(' E ')' {Lcond();} E ';' {End();}

E :V '='{push();} E{codegen\_assign();}

| E '+'{push();} E{codegen\_assign();}

| E '-'{push();} E{codegen\_assign();}

| E '\*'{push();} E{codegen\_assign();}

| E '/'{push();} E{codegen\_assign();}

| '(' E ')'

| '-'{push();} E{codegen\_assign();} %prec MINUS

| V

| NUM{push();}

;

V : ID {push();}

;

%%

#include "lex.yy.c"

#include<stdio.h>

char st[100][10];

int top=0;

char temp[3]="t0";

main()

{

printf("Enter the expression : ");

yyparse();

}

push()

{

strcpy(st[++top],yytext);

}

codegen()

{

printf("%s = %s %s %s\n", temp, st[top-2], st[top-1],st[top]);

top-=2;

strcpy(st[top],temp);

temp[1]++;

}

codegen\_umin()

{

printf("%s = -%s\n", temp, st[top]);

top--;

strcpy(st[top],temp);

temp[1]++;

}

codegen\_assign()

{

printf("%s = %s\n", st[top-2],st[top]);

top-=2;

}

L1()

{

printf("\nL1: \n");

}

Lcond()

{

printf("%s = not %s\n", temp,st[top]);

printf("if %s goto End\n", temp);

temp[1]++;

}

End()

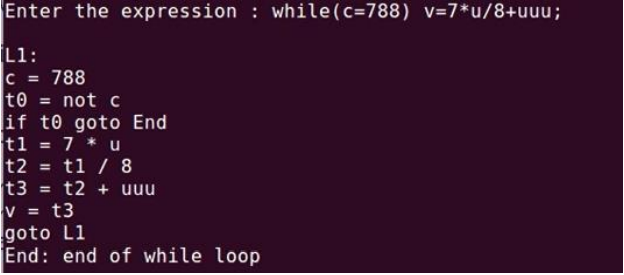
{

printf("goto L1\n");

printf("End: end of while loop \n\n");

}

**OUTPUT:**



**WEEK13:**

**AIM:** Code optimization

**CODE:**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

void input();

void output();

void change(int p,char \*res);

void constant();

struct expr

{

char op[2],op1[5],op2[5],res[5];

int flag;

}arr[10];

int n;

void main()

{

input();

constant();

output();

}

void input()

{

int i;

printf("\n\nEnter the maximum number of expressions : ");

scanf("%d",&n);

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

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

{

scanf("%s",arr[i].op);

scanf("%s",arr[i].op1);

scanf("%s",arr[i].op2);

scanf("%s",arr[i].res);

arr[i].flag=0;

}

}

void constant()

{

int i;

int op1,op2,res;

char op,res1[5];

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

{

if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0)

/\*if both digits, store them in variables\*/

{

op1=atoi(arr[i].op1);

op2=atoi(arr[i].op2);

op=arr[i].op[0];

switch(op)

{

case '+':

res=op1+op2;

break;

case '-':

res=op1-op2;

break;

case '\*':

res=op1\*op2;

break;

case '/':

res=op1/op2;

break;

case '=':

res=op1;

break;

}

sprintf(res1,"%d",res);

arr[i].flag=1;

change(i,res1);

}

}

}

void output()

{

int i=0;

printf("\nOptimized code is : ");

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

{

if(!arr[i].flag)

{

printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);

}

}

}

void change(int p,char \*res)

{

int i;

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

{

if(strcmp(arr[p].res,arr[i].op1)==0)

strcpy(arr[i].op1,res);

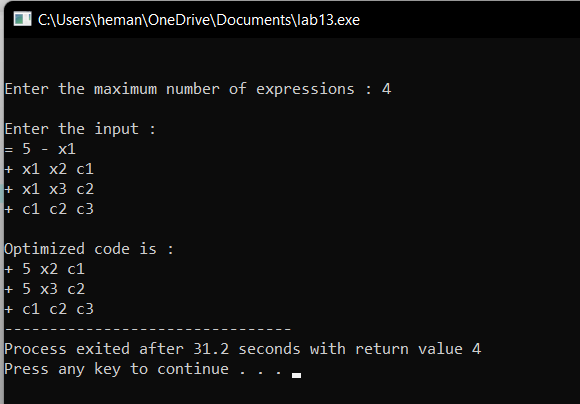
else if(strcmp(arr[p].res,arr[i].op2)==0)

strcpy(arr[i].op2,res);

}

}

**OUTPUT:**



**Week14:**

**Aim:** Write a program to perform loop unrolling

**CODE:**

#include<stdio.h>

#include<conio.h>

void main()

{

unsigned int n;

int x;

char ch;

printf("\nEnter N\n");

scanf("%u",&n);

printf("\n1. Loop Roll\n2. Loop UnRoll\n");

printf("\nEnter ur choice\n");

scanf(" %c",&ch);

switch(ch)

{

case '1':

x=countbit1(n);

printf("\nLoop Roll: Count of 1's : %d" ,x);

break;

case '2':

x=countbit2(n);

printf("\nLoop UnRoll: Count of 1's : %d" ,x);

break;

default:

printf("\n Wrong Choice\n");

}

getch();

}

int countbit1(unsigned int n)

{

int bits = 0,i=0;

while (n != 0)

{

if (n & 1) bits++;

n >>= 1;

i++;

}

printf("\n no of iterations %d",i);

return bits;

}

int countbit2(unsigned int n)

{

int bits = 0,i=0;

while (n != 0)

{

if (n & 1) bits++;

if (n & 2) bits++;

if (n & 4) bits++;

if (n & 8) bits++;

n >>= 4;

i++;

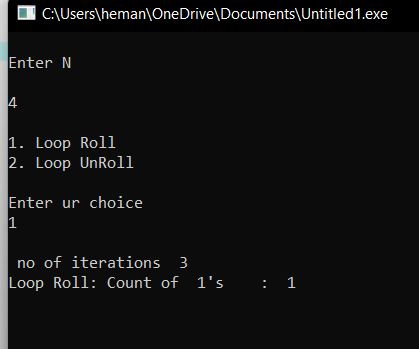
}

printf("\n no of iterations %d",i);

return bits;

}

**OUTPUT:**



**WEEK15:**

**Aim:** A program to generate machine code from the abstract syntax tree  
generated by the parser.

**CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int label[20];

int no=0;

int main()

{

FILE \*fp1,\*fp2;

char fname[10],op[10],ch;

char operand1[8],operand2[8],result[8];

int i=0,j=0;

printf("\n Enter filename of the intermediate code");

scanf("%s",&fname);

fp1=fopen(fname,"r");

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

if(fp1==NULL || fp2==NULL)

{

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

exit(0);

}

while(!feof(fp1))

{

fprintf(fp2,"\n");

fscanf(fp1,"%s",op);

i++;

if(check\_label(i))

fprintf(fp2,"\nlabel#%d",i);

if(strcmp(op,"print")==0)

{

fscanf(fp1,"%s",result);

fprintf(fp2,"\n\t OUT %s",result);

}

if(strcmp(op,"goto")==0)

{

fscanf(fp1,"%s %s",operand1,operand2);

fprintf(fp2,"\n\t JMP %s,label#%s",operand1,operand2);

label[no++]=atoi(operand2);

}

if(strcmp(op,"[]=")==0)

{

fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n\t STORE %s[%s],%s",operand1,operand2,result);

}

if(strcmp(op,"uminus")==0)

{

fscanf(fp1,"%s %s",operand1,result);

fprintf(fp2,"\n\t LOAD -%s,R1",operand1);

fprintf(fp2,"\n\t STORE R1,%s",result);

}

switch(op[0])

{

case '\*': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t MUL R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '+': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t ADD R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '-': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t SUB R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '/': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t DIV R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '%': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t DIV R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '=': fscanf(fp1,"%s %s",operand1,result);

fprintf(fp2,"\n\t STORE %s %s",operand1,result);

break;

case '>': j++;

fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n\t JGT %s,label#%s",operand2,result);

label[no++]=atoi(result);

break;

case '<': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n\t JLT %s, label#%d",operand2,result);

label[no++]=atoi(result);

break;

}

}

fclose(fp2); fclose(fp1);

fp2=fopen("target.txt","r");

if(fp2==NULL)

{

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

exit(0);

}

do

{

ch=fgetc(fp2);

printf("%c",ch);

}while(ch!=EOF);

fclose(fp1);

return 0;

}

int check\_label(int k)

{

int i;

for(i=0;i<no;i++)

{

if(k==label[i])

return 1;

}

return 0;

}

**OUTPUT:**

