**ex.no 1 lex. Analyzer to create sym,table rec. ident…**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

main(){

char a[100],temp[100];

char\*word;

char delim[]=";><{}+)(&%#,=";

char variable[10][20],datatype[10][20];

int k,noofvar=0;

void\*i;

FILE\*p;

p=fopen("add.c","r");

fscanf(p,"%s",a);

strcpy(temp,"NULL");

printf("\nLexeme\tToken\n\n");

while(strcmp(a,"END")!=0){

for(k=0;k<strlen(a);k++){

if(a[k]==';'||a[k]=='<'||a[k]=='{'||a[k]=='>'||a[k]==')'||a[k]=='}'||a[k]=='#'||a[k]=='>'||a[

k]==','||a[k]=='&'||a[k]=='='||a[k]=='(')

{

printf("\033[0;37m");

printf("\n%c\t Special Character",a[k]);

}

}

word=strtok(a,delim);

while(word!=NULL){

if((strcmp(word,"scanf")==0)||(strcmp(word,"printf")==0)||(strcmp(word,"main")==0))

{

printf("\033[0;36m");

printf("\n%s\t",word);

printf("BUILD IN FUNCTION\t");

}

else

if((strcmp(word,"int")==0)||(strcmp(word,"float")==0)||(strcmp(word,"char")==0)||(strc

mp(word,"void")==0))

{

printf("\033[0;32m");

printf("\n%s\t",word);

printf("KEYWORD\t");

}

else if((strcmp(word,"include")==0))

{

printf("\033[0;32m");

printf( "\n%s\t",word);

printf("PROCESSOR\t");

}

else if((strcmp(temp,"int")==0)||(strcmp(temp,"float")==0)||(strcmp(temp,"char")==0))

{

printf("\033[0;36");

printf("\n%s\t",word);

printf("VARIABLE\t");

strcpy(variable[noofvar],word);

strcpy(datatype[noofvar],temp);

noofvar++;

}

word=strtok(NULL,delim);

}

strcpy(temp,a);

fscanf(p,"%s",a);}

fclose(p);

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

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

printf("symbol\taddr\ttype\n");

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

{

i=malloc(variable[k][0]);

printf("\033[0;31m");

printf("%s\t%d\t%s\n",variable[k],i,datatype[k]);

}

return 0;

}

ADD.C

#include<stdio.h>

#include<string.h>

void main()

{

int a,b,sum;

printf("enter number:");

scanf("%d",&a);

printf("enter number:");

scanf("%d",&b);

sum=a+b;

printf("sum=%d",sum);

}

END

**ex.no 2 lexi..anal..using lex tool**

%{

#include <stdio.h>

%}

%%

int|char|float|return { printf("%s=> Keywords\n", yytext); }

#.\* { printf("\n%s=>Preprocessor Directive", yytext); }

printf|scanf|main { printf("\n%s=>functions", yytext); }

[a-zA-Z]+ { printf("\n%s=>Identifiers", yytext); }

\"[a-zA-Z]\*\" { printf("\n%s=>String Literals", yytext); }

[0-9]+ { printf("\n%s=>Integer Literals", yytext); }

[+\-\*/%] { printf("\n%s=>Operators", yytext); }

[,\(\)\[\]\{\};&] { printf("\n%s=>Special Characters", yytext); }

%%

int main() {

FILE \*fp;

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

yyin = fp;

yylex();

return 0;

}

int yywrap() {

return 1;

}

INPUT.TXT:

int main()

{

int a,b; printf("hello"); float c;

char d;

return 0;

}

**ex.no 3a valid expression**

ao.l

%{

#include<stdio.h>

#include"aro.tab.h"

extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext);

return NUM;

}

[\t] ;

\n return 0;

. return yytext[0];

%%

aro.y

%{

#include<stdio.h>

#include<conio.h>

%}

%token NUM

%left '+' '-'

%left '\*' '/'

%left '(' ')'

%%

expr: e{

printf("result:%d\n",$$);

return 0;

}

e:e'+'e {$$=$1+$3;}

|e'-'e {$$=$1-$3;}

|e'\*'e {$$=$1\*$3;}

|e'/'e {$$=$1/$3;}

|'('e')' {$$=$2;}

| NUM {$$=$1;}

;

%%

int main()

{

printf("\n enter the arithematic expression:\n");

yyparse();

printf("\nvalid expression\n");

}

yyerror()

{

printf("\n invalid expression\n");

}

int yywrap()

{

return 1;

}

flex ao.l

bison -d aro.y

gcc lex.yy.c aro.tab.c

**EX.NO 3B valid variable**

LEX PROGRAM

%{

#include"4b.tab.h"

%}

%%

[a-zA-Z] {return LETTER;}

[0-9] {return DIGIT;}

[\_] {return UND;}

[\n] {return NL;}

. {return yytext[0];}

%%

YACC PROGRAM

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token DIGIT LETTER UND NL

%%

stmt: variable NL {printf("valid identifiers\n"); exit(0);}

;

variable: LETTER alphanumeric

;

alphanumeric: LETTER alphanumeric | DIGIT alphanumeric | UND alphanumeric

| LETTER | DIGIT | UND

;

%%

int yyerror(char \*msg)

{

printf("Invalid variable\n");

exit(0);

}

main()

{

printf("enter the variable: \n");

yyparse();

}

int yywrap()

{

return 1;

}

**EX.NO 3C c language stmts**

3c.l

%{

#include "3c.tab.h"

%}

%%

"if" {return IF;}

[sS][0-9]\* {return S;}

"<"|">"|"=="|"!="|"<="|">=" {return RELOP;}

[0-9]+ {return NUMBER;}

[a-z][a-zA-Z0-9\_]\* {return ID;}

\n {return NL;}

. {return yytext[0];}

%%

3c.y

%{

#include<stdio.h>

#include<stdlib.h>

int count=0;

%}

%token IF RELOP S NUMBER ID NL

%%

stmt: if\_stmt NL {printf("No of nested if statements=%d \n",count); exit(0);}

;

if\_stmt: IF'('cond')''{'if\_stmt'}' {count++;}

|S

;

cond: x RELOP x

;

x: ID | NUMBER

;

%%

int yyerror(char \*msg)

{

printf("Invalid Statement\n");

exit(0); }

main() {

printf("Enter the statement\n");

yyparse();

}

int yywrap() {

return 1; }

**EX.NO 3D calculator**

3d.l

%{

#include <stdlib.h>

#include <stdio.h>

#include "3d.tab.h"

void yyerror(char\*);

extern int yylval;

%}

%%

[ \t]+ ;

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

return INTEGER;}

[-+\*/] {return \*yytext;}

"(" {return \*yytext;}

")" {return \*yytext;}

\n {return \*yytext;}

. {char msg[25];

sprintf(msg,"%s <%s>","invalid character",yytext);

yyerror(msg);

}

3d.y

%{

#include<stdlib.h>

#include<stdio.h>

%}

%token INTEGER

%%

stmt: expr '\n' { printf("%d\n",$1); }

| 'n'

expr:

expr '+' mulex { $$ = $1 + $3; }

| expr '-' mulex { $$ = $1 - $3; }

| mulex { $$ = $1; }

mulex:

mulex '\*' term { $$ = $1 \* $3; }

| mulex '/' term { $$ = $1 / $3; }

| term { $$ = $1; }

term:

'(' expr ')' { $$ = $2; }

| INTEGER { $$ = $1; }

%%

int yyerror(char \*s)

{

fprintf(stderr,"%s\n",s);

return;

}

yywrap()

{

return(1);

}

int main(void)

{

yyparse();

return 0;

}

**EX.NO 4 three address codes using lex and yacc**

4.l

%{

#include"4.tab.h"

extern char yyval;

%}

%%

[0-9]+ {yylval.symbol=(char)(yytext[0]);return NUMBER;}

[a-z] {yylval.symbol= (char)(yytext[0]);return LETTER;}

. {return yytext[0];}

\n {return 0;}

%%

4.y

%{

#include<string.h>

#include<stdio.h>

char addtotable(char,char,char);

int index1=0;

char temp ='A'-1;

struct expr

{

char operand1;

char operand2;

char operator;

char result;

};

%}

%union

{

char symbol;

}

%left '+' '-'

%left '/' '\*'

%token <symbol> LETTER NUMBER

%type <symbol> exp

%%

statement: LETTER'='exp';' {addtotable((char)$1,(char)$3,'=');};

exp: exp '+' exp {$$ = addtotable((char)$1,(char)$3,'+');}

|exp '-' exp {$$ = addtotable((char)$1,(char)$3,'-');}

|exp '/' exp {$$ = addtotable((char)$1,(char)$3,'/');}

|exp '\*' exp {$$ = addtotable((char)$1,(char)$3,'\*');}

|'('exp')' {$$= (char)$2;}

|NUMBER {$$ = (char)$1;}

|LETTER {(char)$1;};

%%

struct expr arr[20];

int yyerror(char \*s)

{

printf("Errror %s",s);

}

char addtotable(char a, char b, char o)

{

temp++;

arr[index1].operand1 =a;

arr[index1].operand2 = b;

arr[index1].operator = o;

arr[index1].result=temp;

index1++;

return temp;

}

void threeAdd(){

int i=0;

char temp='A';

while(i<index1){

printf("%c:=\t",arr[i].result);

printf("%c\t",arr[i].operand1);

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

printf("%c\t",arr[i].operand2);

i++;

temp++;

printf("\n");

}

}

int yywrap()

{

return 1;

}

int main(){

printf("Enter the expression: ");

yyparse();

threeAdd();

printf("\n");

return 0;

}

**ex.no 5 imp…type chcking**

5.l

%{

#include "5.tab.h"

%}

%%

[0-9]+ { yylval.integer = atoi(yytext); return INTEGER; }

true|false { yylval.boolean = (\*yytext == 't'); return BOOLEAN; }

[a-zA-Z][a-zA-Z0-9]\* { yylval.string = strdup(yytext); return ID; }

[-+\*/=;()] { return \*yytext; }

[ \t\n] ; // Skip whitespace

. { printf("Float value not allowed: %s\n"); }

%%

5.y

%{

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int yylex();

void yyerror(const char \*s);

%}

%union {

int integer;

char\* string;

int boolean;

}

%token ID INTEGER BOOLEAN

%left '+' '-'

%left '\*' '/'

%%

program : statements

statements : statement statements

| /\* empty \*/

statement : expression ';' { printf("Statement OK\n"); }

expression : ID '=' expression {

// Type checking code here

printf("Assignment OK\n");

}

| expression '+' expression

| expression '-' expression

| expression '\*' expression

| expression '/' expression

| '(' expression ')'

| INTEGER

| BOOLEAN

%%

void yyerror(const char \*s) {

printf("Syntax error: %s\n", s);

}

int main() {

yyparse();

return 0;

}

int yywrap() {

return 1;

}

**ex.no 6 simple code optimization**

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct op

{

char l;

char r[20];

}

op[10],pr[10];

void main()

{

int a,i,k,j,n,z=0,m,q;

char \*p,\*l;

char temp,t;

char \*tem;

clrscr();

printf("Enter the Number of Values:");

scanf("%d",&n);

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

{

printf("left: ");

op[i].l=getche();

printf("\tright: ");

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

}

printf("Intermediate Code\n") ;

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

{

printf("%c=",op[i].l);

printf("%s\n",op[i].r);

}

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

{

temp=op[i].l;

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

{

p=strchr(op[j].r,temp);

if(p)

{

pr[z].l=op[i].l;

strcpy(pr[z].r,op[i].r);

z++;

}

}

}

pr[z].l=op[n-1].l;

strcpy(pr[z].r,op[n-1].r);

z++;

printf("\nAfter Dead Code Elimination\n");

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

{

printf("%c\t=",pr[k].l);

printf("%s\n",pr[k].r);

}

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

{

tem=pr[m].r;

for(j=m+1;j<z;j++)

{

p=strstr(tem,pr[j].r);

if(p)

{

t=pr[j].l;

pr[j].l=pr[m].l;

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

{

l=strchr(pr[i].r,t) ;

if(l)

{

a=l-pr[i].r;

printf("pos: %d",a);

pr[i].r[a]=pr[m].l;

}}}}

}

printf("Eliminate Common Expression\n");

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

{

printf("%c\t=",pr[i].l);

printf("%s\n",pr[i].r);

}

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

{

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

{

q=strcmp(pr[i].r,pr[j].r);

if((pr[i].l==pr[j].l)&&!q)

{

pr[i].l='\0';

strcpy(pr[i].r,'\0');

}

}

}

printf("Optimized Code\n");

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

{

if(pr[i].l!='\0')

{

printf("%c=",pr[i].l);

printf("%s\n",pr[i].r);

}}

getch();

}

**ex.no 7 back end compiler**

#include<stdio.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char icode[10][30],str[20],opr[10];

int i=0;

clrscr();

printf("\n Enter the set of intermediate code (terminated by exit):\n");

do

{

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

} while(strcmp(icode[i++],"exit")!=0);

printf("\n target code generation");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

i=0;

do

{

strcpy(str,icode[i]);

switch(str[3])

{

case '+':

strcpy(opr,"ADD");

break;

case '-':

strcpy(opr,"SUB");

break;

case '\*':

strcpy(opr,"MUL");

break;

case '/':

strcpy(opr,"DIV");

break;

}

printf("\n\tMov %c,R%d",str[2],i);

printf("\n\t%s%c,R%d",opr,str[4],i);

printf("\n\tMov R%d,%c",i,str[0]);

}while(strcmp(icode[++i],"exit")!=0);

getch();

}