1. Program count

--icg.l

%{

# include<stdio.h>

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

%}

%%

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

[ ] {cc++;sc++;}

[\t] {cc++; tc++;}

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

%%

int main()

{

yyin = fopen("a1.txt","r");

yyout=fopen("a2.txt","w");

yylex();

fclose(yyin);

fclose(yyout);

}

int yywrap()

{

fprintf(yyout,"no. of lines =%d\n",lc);

fprintf(yyout,"no. of tabs=%d\n",tc);

fprintf(yyout,"no. of characters = %d\n",cc);

fprintf(yyout,"no. of spaces = %d\n",sc);

fprintf(yyout,"no. of words=%d\n",wc);

}

----out

lex icg.l

ls

gcc lex.yy.c

./a.out > a2.txt

This is the output of the first pratical with a tab.

1. Lexical analyzer

--second.l

%{

# include<stdio.h>

# include<string.h>

struct symtab

{

char \*name;

};

struct symtab sym[10],\*k;

struct symtab \*install\_id(char \*s);

void disp();

%}

L [a-zA-Z]

D [0-9]

id {L}({L}|{D})\*

num {D}+(\.{D}+)?([eE][-+]?{D}+)?

bop [-+\*/=]

uop "++"|"--"

relop "<"|"<="|">"|">="|"!="|"=="

lop "&&"|"||"

bitlop [&|!]

kew "if"|"else"|"while"|"do"|"for"|"int"|"char"|"float"

pun [,;'"\[\]\{\}\)\(]

comment "//"

ws [ \t\n]+

st \"(.)\*\"

%%

{ws} {}

{kew} {printf("keyword=%s\n",yytext);}

{id} {k=install\_id(yytext);printf("identifier =%s\n",yytext);}

{num} {printf("constant =%s\n",yytext);}

{bop} {printf("binary op =%s\n",yytext);}

{uop} {printf("unary op =%s\n",yytext);}

{relop} {printf("relational op=%s\n",yytext);}

{lop} {printf("logical op =%s\n",yytext);}

{pun} {printf("punct =%s\n",yytext);}

{bitlop} {printf("bitwise logical op=%s\n",yytext);}

{comment} {printf("comment =%s\n",yytext);}

{st} {printf("string =%s\n",yytext);}

%%

int main()

{

yylex();

disp();

}

struct symtab \*install\_id(char \*s)

{

struct symtab \*p;

printf("in symbol table\n");

for(p=&sym[0];p<&sym[10];p++)

{

if(p->name && !strcmp(s,p->name))

return p;

if(!p->name)

{

p->name=strdup(s);

return p;

}

}

}

void disp()

{

struct symtab \*p;

printf("symbol table\n");

for(p=&sym[0];p<&sym[10];p++)

{

if(p->name)

printf("%s\n",p->name);

}

}

int yywrap()

{

return 1;

}

----- online run this file

lex second.l

gcc lex.yy.c

./a.out < prg1.c

keyword=int

in symbol table

identifier =main

punct =(

punct =)

punct ={

keyword=int

in symbol table

identifier =a

binary op ==

constant =10

punct =,

in symbol table

identifier =b

binary op ==

constant =20

punct =,

in symbol table

1. Checking of subset

--synatx.l

%{

#include"y.tab.h"

%}

NOUN "Tarkeshwari"|"Dog"|"Asha"|"Mayuri"|"Monali"|"Pratiksha"|"Car"

VERB "Play"|"Run"|"Sing"|"Act"|"Answer"|"Approve"|"Arrange"

PREPOSITION "In"|"On"|"Below"|"Above"|"Beside"

ADJECTIVE "Good"|"Bad"|"Big"|"small"|"Tall"|"Short"|"ugly"|"Beautiful"|"Adorable"

PRONOUN "He"|"She"|"It"|"They"|"Them"

DET "a"|"an"|"the"

CONJ ([aA][nN][dD])|([oO][rR])|([bB][uU][tT])

%%

{NOUN} {return NOUN;}

{VERB} {return VERB;}

{PREPOSITION} {return Preposition;}

{ADJECTIVE} {return ADJECTIVE;}

{PRONOUN} {return PRONOUN;}

{DET} {return DET;}

{CONJ} {return CONJ;}

"." {return yytext[0];}

.|\n {}

%%

int yywrap()

{

return 1;

}

---out

yacc -d -v syntax.y

lex syntax.l

gcc lex.yy.c y.tab.c

./a.out > abc.txt

--syntax.y

%{

#include <stdio.h>

%}

%token DET NOUN PRONOUN VERB Preposition ADJECTIVE

%nonassoc NOUN

%nonassoc Preposition

%nonassoc CONJ

%%

S1:S1 S

|S

;

S:SS '.' {printf("Simple Sentence");}

|CS'.' {printf("Compound Sentence");};

CS:SS CONJ SS {printf("Compound Sentence");}

|CS CONJ SS;

SS:NP VP {printf("SS\n");}| VP{printf("1\n");};

NP:DET NOUN {printf("Two No.\n");}

| NOUN {printf("3\n");}

| PRONOUN {printf("5\n");}

| DET NOUN PP {printf("6\n");};

VP:VERB NP {printf("7\n");}| VERB {printf("8\n");} |VP PP {printf("10\n");};

PP:Preposition NP {printf("11\n");};

%%

main()

{

yyparse();

}

int yyerror(char\* msg)

{

return 1;

}

1. Checking control statements

--control .l

%{

#include<stdio.h>

#include "y.tab.h"

%}

L [A-Za-z]

D [0-9]

id {L}({L}|{D})\*

%%

"if" {return IF;}

"else" {return ELSE;}

"for" {return FOR;}

"do" {return DO;}

"while" {return WHILE;}

"++" {return INC;}

"--" {return DEC;}

"||" {return OR;}

"&&" {return AND;}

"!" {return NOT;}

"switch" {return SWITCH;}

"case" {return CASE;}

"break" {return BREAK;}

"default" {return DEFAULT;}

[0-9]+(\.[0-9]+)? {return NUM;}

{id} {return id;}

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

[-/;=+\*,\(\)\{\}:] {return yytext[0];}

[ ] {}

\n {}

%%

int yywrap()

{

return 1;

}

---out

lex control.l

yacc -d -v control.y

gcc y.tab.c lex.yy.c

./a.out

a=b

:

Assignmenet statement accepted

if (a<b)

{

c=a;

Assignment statement accepted

}

c=a;

Assignment statement accepted

}

else

{c=b;

Assignmentstatement accepted

}

If else statement is accepted

--control.y

%{

#include <stdio.h>

%}

%token id NUM OR AND NOT relop TRUE FALSE INC DEC IF ELSE DO WHILE uminu

s FOR SWITCH CASE BREAK DEFAULT

%right '='

%left '+' '-'

%left '\*' '/'

%right '^'

%nonassoc uminus

%left OR%left AND

%nonassoc NOT

%%

S1 : S1 S

|S

;

S : AS ';' {printf("Assignment statement accepted \n");}

|IFS {printf("If statement is accepted \n");}

|IFES {printf("If else statement is accepted\n");}

|WS

{printf("While statement is accepted\n");}

|DWS {printf("Do while statement is accepted\n");}

|FORS {printf("For statement is accepted\n");}

|SS

{printf("Switch statement is accepted");}

;

SS : SWITCH'('E')''{' CV '}'

;

CV : CASE E ':' S1 BREAK ';'

| CASE E ':' S1 BREAK ';' CV

| CASE E ':' S1 BREAK ';' DEFAULT ':' S1

;

AS : id '=' E

;

E : E'+'E

|E'-'E

|E'\*'E

|E'/'E

|E'^'E

|'-' E %prec uminus

|id

|NUM

;

IFS: IF'('BE')''{'S1'}'

;

BE : BE OR BE

| BE AND BE

| NOT BE

|id relop id

|TRUE

|FALSE

;

IFES : IF'('BE')''{'S1'}'ELSE'{'S1'}'

;WS : WHILE '('BE')''{'S1'}'

;

DWS : DO '{'S1'}'WHILE'('BE')'';'

;

FORS : FOR'('IS';'BE';'MS')''{'S1'}'

;

IS : AS

| IS ',' AS

;

MS : IS

| id INC

| INC id

| id DEC

| DEC id

;

%%

void main()

{

yyparse();

}

int yyerror(char \*msg)

{

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

}

1. Checking declaration

--declaration.l

%{

#include "y.tab.h"

%}

letter [a-zA-Z]

num [0-9]+

float {num}+\.{num}+

bools "true"|"false"

identifier {letter}({letter}|{num})\*

%%

"int" {return Int;}

"char" {return Char;}

"float" {return Float;}

"bool" {return Bool;}

{num} {return IntV;}

{float} {return FloatV;}

{bools} {return BoolV;}

{identifier} {return Id;}

"'"(.)"'" {return CharV;}

[,;=] { return yytext[0]; }

%%

int yywrap() {

return 0;

}

--out

yacc -d -v declaration.y

lex declaration.l

gcc lex.yy.c y.tab.c

./a.out

int a;

int declaration accepted

int a=4;

int declaration accepted

char s='r';

char declaration accepted

bool f=true;

bool declaration accepted.

--declaration.y

%{

#include <stdio.h>

%}

%token Int Char Float Bool String IntV CharV FloatV BoolV StringV Id Am Const

%%

S1: S1 S

|S

;

S: Int Iv';' { printf("int declaration accepted"); }

| Char Cc';' { printf("char declaration accepted"); }

| Float Ff';' { printf("float declaration accepted"); }

| Bool Bb';' { printf("bool declaration accepted."); }

;

Iv: IdM

| Iv ',' Id

| Id '=' IntV

| Iv ',' Id '=' IntV

;

Cc: IdM

| Cc ',' Id

| Id '=' CharV

| Cc ',' Id '=' CharV

;

Ff: IdM

| Ff ',' Id

| Id '=' FloatV

| Ff ',' Id '=' FloatV

;

Bb: IdM| Bb ',' Id

| Id '=' BoolV

| Bb ',' Id '=' BoolV

;

Ss: IdM

| Ss ',' Id

| Id '=' StringV

| Ss ',' Id '=' StringV

;

IdM: Id

;

%%

void yyerror(char\*s) {

printf("%s", s);

}

int main() {

yyparse();

return 0;

}

1. Calculator

--l6.l

%{

# include<stdio.h>

# include<stdlib.h>

# include "y.tab.h"

%}

num ([0-9]+)(\.[0-9]+)?([eE][-+]?[0-9]+)?

%%

"sin" {return SIN;}

"cos" {return COS;}

"tan" {return TAN;}

[A-Za-z][A-Za-z0-9]\* {yylval.p=install\_id(yytext);return id;}

{num} {yylval.v=atof(yytext); return NUM;}

[\n] {return yytext[0];}

. {return yytext[0];}

%%

int yywrap()

{

return 1;

}

--out

yacc -d -v l6.y

lex l6.l

gcc lex.yy.c y.tab.c

./a.out

tan(45)

ans = 0.999204

cos(60)

ans = 0.500460

--l6.y

%{

# include<math.h>

# include<stdio.h>

struct symtab

{

char \*name;

double val;

}SYM[20];

void disp();

struct symtab \*install\_id(char \*s);

%}

%union{

double v;

struct symtab \*p;

}

%token SIN COS TAN

%token<v> NUM

%token<p> id

%right '='

%left '+' '-'

%left '\*' '/'

%nonassoc UMINUS

%type<v> E

%%

lines: lines S

|S;

S: id '=' E '\n' {$1->val=$3;}

|E '\n' {printf("ans = %lf\n",$1);}

E: E'+'E

{$$ = $1 + $3;}

| E'-'E

{$$=$1-$3;}

|E'\*'E

{$$=$1\*$3;}

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

| '-' E %prec UMINUS {$$ = -$2;}

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

|NUM

|id

{$$=$1->val;}

|SIN'('E')'

{$$=sin(($3\*3.14)/180);}

|COS'('E')'

{$$=cos(($3\*3.14)/180);}

|TAN'('E')' {$$=tan(($3\*3.14)/180);}

;

%%

int main()

{

yyparse();

disp();

}

int yyerror()

{

return 1;

}

void disp()

{

struct symtab \*k;

for(k=SYM;k<&SYM[20];k++)

{

if(k->name)

{

printf("%s\t%f\n",k->name,k->val);

} } }

struct symtab \* install\_id(char \*s)

{

struct symtab \*k;

for(k=SYM;k<&SYM[20];k++)

{

if(k->name&&!strcmp(k->name,s))

return k;

else

if(!k->name)

{

k->name=strdup(s);

return k;

}

}

}

1. ICG

--icg.l

%{

#include<stdio.h>

#include<string.h>

#include "y.tab.h"

struct symtab \*install\_id(char \*s);

%}

l [A-Za-z]

d [0-9]

id {l}({l}|{d})\*

num {d}+(\.{d}+)?

%%

{id} {yylval.p=install\_id(yytext); return id;}

{num} {strcpy(yylval.v,yytext); return num;}

[-+\*/^=;] {return yytext[0];}

.|\n {}

%%

int yywrap()

{

return 1;

}

--out

yacc -d -v icg.y

lex icg.l

gcc y.tab.c lex.yy.c -lm

./a.out

a=b+c\*d;

--icg.y

%{

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void itoa1(int a,int k);

char\* insert\_quad(char op[10],char arg1[10],char arg2[10],int j);

struct symtab{

char \*name;

double value;

}SYM[20];

struct symtab \*install\_id(char \*s);

void display\_sym();

typedef struct quadruple{

char op[10];

char arg1[10];

char arg2[10];

char res[10];

}QUAD;

QUAD Q[30];

void display\_quad();

int i=0;

int tempvar=1;

char temp[10],st[10];

%}

%union{

struct symtab \*p;

char v[10];

}

%token<p> id

%token <v>num

%right '='

%left '\*''/'

%right '^'

%nonassoc UMINUS

%type<v>E

%%

S: S OS

|OS

;

OS:AS

;

AS:id'='E';'{strcpy(Q[i].op,"=");

strcpy(Q[i].arg1,$3);

strcpy(Q[i].arg2,"");

strcpy(Q[i].res,$1->name);

i++;}

;

E:E'+'E{strcpy($$,insert\_quad("+",$1,$3,i));i++;}

|E'-'E{strcpy($$,insert\_quad("-",$1,$3,i));i++;}

|E'\*'E{strcpy($$,insert\_quad("\*",$1,$3,i));i++;}

|E'/'E{strcpy($$,insert\_quad("/",$1,$3,i));i++;}

|E'^'E{strcpy($$,insert\_quad("^",$1,$3,i));i++;}

|'-'E{strcpy($$,insert\_quad("UMINUS",$2,"",i));i++;}

|id{strcpy($$,$1->name);}

|num{strcpy($$,$1);}

;

%%

char\* insert\_quad(char op[10],char arg1[10],char arg2[10],int j)

{

strcpy(Q[j].op,op);

strcpy(Q[j].arg1,arg1);

strcpy(Q[j].arg2,arg2);

strcpy(temp,"t");

itoa1(tempvar++,10);

strcat(temp,st);

strcpy(Q[j].res,temp);

return temp;

}

int yyerror(char \*s)

{

printf("error=%s\n",s);

return 1;

}

main()

{

yyparse();

display\_quad();

display\_sym();

}

struct symtab \*install\_id(char \*s)

{

struct symtab \*p;

for(p=SYM;p<&SYM[20];p++)

{

if(p->name&&!strcmp(p->name,s))

return p;

else

if(!p->name)

{

p->name=strdup(s);

return p;

}

}

}

void display\_sym()

{

struct symtab \*p;

printf("symbol name value\n");

for(p=SYM;p<&SYM[20];p++)

{

if(p->name)

printf("%s\t%lf\n",p->name,p->value);

}

}

void itoa1(int t,int b)

{

int j=0,k;

char m[10];

while(t!=0)

{

m[j]=t%b+48;

t=t/b;

j++;

}

m[j]='\0';

j=0;

for(k=strlen(m)-1;k>=0;k--)

st[j++]=m[k];

st[j]='\0';

}

void display\_quad()

{

int j;

printf("s\_no\top\targ1\targ2\tres\n");

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

printf("%d\t%s\t%s\t%s\t%s\n",j,Q[j].op,Q[j].arg1,Q[j].arg2,Q[j].res);

}

1. Code optimization

#include<stdio.h>

#include<math.h>

#include<string.h>

# include<ctype.h>

#include<stdlib.h>

struct quad

{

char ope[5];

char arg1[5];

char arg2[5];

char res[5];

}QUAD[5];

int i=0,n,c=0;

void get()

{

printf("\nEnter no of lines in a block");

scanf("%d",&n);

printf("enter ICG in form operator arg1 arg2 result:");

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

scanf("%s\n%s\n%s\n%s",&QUAD[i].ope,&QUAD[i].arg1,&QUAD[i].arg2,&QUAD[i].res);

}

void const\_folding()

{

int j,c1=0,d=0;

char ch[5],ch1[5],num[10];

int flag1 =1, flag2 =1;

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

{

flag1 =1;flag2 =1;

for (j=0;j<strlen(QUAD[i].arg1);j++)

{

if(!isdigit(QUAD[i].arg1[j]))

{ flag1 = 0;printf("Operand1 is not contstant, Constant folding can not applied to quadruple

%d\n",i);

break;

}

}

for (j=0;j<strlen(QUAD[i].arg2);j++)

{

if(!isdigit(QUAD[i].arg2[j]))

{ flag2 = 0; printf("Operand2 is not contstant, Constant folding can not applied to quadruple

%d\n",i);

break;

}

}

if(flag1 == 1 && flag2 ==1)

{

c=atoi(QUAD[i].arg1);

c1=atoi(QUAD[i].arg2);

if(strcmp(QUAD[i].ope,"\*")==0)

{

d=c\*c1;

//itoa(d,ch,10);

snprintf(ch, 10, "%d", d);

strcpy(QUAD[i].ope,"=");

strcpy(QUAD[i].arg1,ch);

strcpy(QUAD[i].arg2,"\0");

}

if(strcmp(QUAD[i].ope,"/")==0)

{

d=c/c1;

//itoa(d,ch,10);

snprintf(ch, 10, "%d", d);

strcpy(QUAD[i].ope,"=");

strcpy(QUAD[i].arg1,ch);

strcpy(QUAD[i].arg2,"\0");

}

if(strcmp(QUAD[i].ope,"+")==0)

{

d=c+c1;

//itoa(d,ch,10);

snprintf(ch, 10, "%d", d);

strcpy(QUAD[i].ope,"=");

strcpy(QUAD[i].arg1,ch);

strcpy(QUAD[i].arg2,"\0");

}

if(strcmp(QUAD[i].ope,"-")==0)

{

d=c-c1;

//itoa(d,ch,10);

snprintf(ch, 10, "%d", d);

strcpy(QUAD[i].ope,"=");

strcpy(QUAD[i].arg1,ch);

strcpy(QUAD[i].arg2,"\0");

}

}

}

}

void strength\_reduction()

{

int j=0,n1=0,m=0,c=0,tempo=0,t=0;

char ch[5],cc[5],ct[2],pres[5];

int flag;

strcpy(ct,"s");

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

c=0;

if(strcmp(QUAD[i].ope,"\*")==0||strcmp(QUAD[i].ope,"/")==0)

{ j = 1;

if(strcmp(QUAD[i].ope,"\*")==0)

flag =0;

else

flag =1;

if((atoi(QUAD[i].arg2))>0)

{

m=atoi(QUAD[i].arg2);

while(n1<=m)

{

n1=pow(2,j);

j++;

}

j=j-2;

n1=pow(2,j);

c=m-n1;

printf("number! is 2^%d + %d",j,c);

if(c==0)

{

//itoa(j,ch,10);

snprintf(ch, 10, "%d", j);

if(flag==0)

strcpy(QUAD[i].ope,"<<");

else

strcpy(QUAD[i].ope,">>");

// strcpy(QUAD[i].arg1,ch);

strcpy(QUAD[i].arg2,ch);

// strcpy(QUAD[i].res,"t2");

}

else

{

strcpy(pres,QUAD[i].res);

//itoa(j,ch,10);

snprintf(ch, 10, "%d", j);

if(flag==0)

strcpy(QUAD[i].ope,"<<");

else

strcpy(QUAD[i].ope,">>");

strcpy(QUAD[i].arg2,ch);

strcpy(QUAD[i].res,"t2");

i++;

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

{

for(j=n;j>=i;j--)

QUAD[j+1] = QUAD[j];

if(c==1)

{

//itoa(c,ch,10);

snprintf(ch, 10, "%d", j);

if(flag==0)

strcpy(QUAD[i].ope,"+");

else

strcpy(QUAD[i].ope,"-");

tempo=i-1;

strcpy(QUAD[i].arg1,QUAD[tempo].res);

strcpy(QUAD[i].arg2,ch);

//itoa(i,cc,10);

snprintf(cc, 10, "%d", i);

strcat(ct,cc);

printf("CT is %s",ct);

strcpy(QUAD[i].res,ct);

}

else

{

strcpy(ct,"s");

//itoa(c-(c-1),ch,10);

snprintf(ch, 10, "%d", c-(c-1));

if(flag==0)

strcpy(QUAD[i].ope,"+");

else

strcpy(QUAD[i].ope,"-");

tempo=i-1;

strcpy(QUAD[i].arg1,QUAD[tempo].res);

strcpy(QUAD[i].arg2,ch);

//strcat("t",i);

//itoa(i,cc,10);

snprintf(cc, 10, "%d", i);

strcat(ct,cc);

strcpy(QUAD[i].res,ct);

}

i++;

n=n+1;

}

/\* itoa(c,ch,10);

strcpy(QUAD[i].ope,"+");

tempo=i-2;

strcpy(QUAD[i].arg1,QUAD[tempo].res);

tempo=tempo+1;

strcpy(QUAD[i].arg2,QUAD[tempo].res);

strcpy(QUAD[i].res,"t2");

\*/

}

}

}

printf("n value =%d\n",n);

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

{

if(strcmp(QUAD[j].arg1, pres) ==0)

strcpy(QUAD[j].arg1,QUAD[i-1].res);

else if (strcmp(QUAD[j].arg2, pres) ==0)

strcpy(QUAD[j].arg2,QUAD[i-1].res);

}

if(c!=0)

i = i-1;

}

}

void disp()

{

printf("\nQuadraple\noperator\targ1\targ2\tresult\n");

printf("n value is %d\n",n);

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

printf("\t%s\t%s\t%s\t%s\n",QUAD[i].ope,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].res);

}

void main()

{

get();

disp();

const\_folding();

printf("Quadruples after constant folding\n");

disp();

strength\_reduction();

printf("Quadruples after strength reduction\n");

disp();

}

1. Tree generation

#include <iostream>

#include <stack>

#include <string>

struct bin\_tree

{

char data;

int label;

bin\_tree\* right;

bin\_tree\* left;

};

typedef bin\_tree node;

void insertNode(node\*\* tree, char val)

{

node\* temp = nullptr;

if (!(\*tree))

{

temp = new node;

temp->left = nullptr;

temp->right = nullptr;

temp->data = val;

temp->label = -1;

\*tree = temp;

}

}

void insert(node\*\* tree, char val)

{

char l, r;

int numofchildren;

insertNode(tree, val);

std::cout << "\nEnter number of children of " << val << ": ";

std::cin >> numofchildren;

if (numofchildren == 2)

{

std::cout << "\nEnter Left Child of " << val << ": ";

std::cin >> l;

insertNode(&(\*tree)->left, l);

std::cout << "\nEnter Right Child of " << val << ": ";

std::cin >> r;

insertNode(&(\*tree)->right, r);

insert(&(\*tree)->left, l);

insert(&(\*tree)->right, r);

}

}

void findLeafNodeLabel(node\* tree, int val)

{

if (tree->left != nullptr && tree->right != nullptr)

{

findLeafNodeLabel(tree->left, 1);

findLeafNodeLabel(tree->right, 0);

}

else

{

tree->label = val;

}

}

void findInteriorNodeLabel(node\* tree)

{

if (tree->left->label == -1)

{

findInteriorNodeLabel(tree->left);

}

else if (tree->right->label == -1)

{

findInteriorNodeLabel(tree->right);

}

else

{

if (tree->left != nullptr && tree->right != nullptr)

{

if (tree->left->label == tree->right->label)

{

tree->label = tree->left->label + 1;

}

else

{

if (tree->left->label > tree->right->label)

{

tree->label = tree->left->label;

}

else

{

tree->label = tree->right->label;

}

}

}

}

}

void printInorder(node\* tree)

{

if (tree)

{

printInorder(tree->left);

std::cout << tree->data << " with Label " << tree->label << std::endl;

printInorder(tree->right);

}

}

void swap(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

}

int pop(int\* R, int& top)

{

int temp = R[top];

top--;

return temp;

}

void push(int\* R, int& top, int temp, int numOfRegisters)

{

if (top == numOfRegisters - 1)

{

std::cout << "Stack overflow! Storing in temporary variable T." << std::endl;

top = numOfRegisters; // Reset top to indicate overflow

R[top] = temp; // Store value in T

}

else

{

top++;

R[top] = temp;

}

}

std::string operationName(char temp)

{

switch (temp)

{

case '+':

return "ADD";

case '-':

return "SUB";

case '\*':

return "MUL";

case '/':

return "DIV";

default:

return "";

}

}

void generateCode(node\* tree, int\* R, int& top, int numOfRegisters)

{

if (tree->left != nullptr && tree->right != nullptr)

{

if (tree->left->label == 1 && tree->right->label == 0 && tree->left->left == nullptr && tree->left->right == nullptr && tree->right->left == nullptr && tree->right->right == nullptr)

{

std::cout << "MOV " << tree->left->data << ", ";

if (top == numOfRegisters)

std::cout << "T";

else

std::cout << "R[" << top << "]";

std::cout << "\n";

std::string op = operationName(tree->data);

std::cout << op << " " << tree->right->data << ", ";

if (top == numOfRegisters)

std::cout << "T";

else

std::cout << "R[" << top << "]";

std::cout << "\n";

}

else if (tree->left->label >= 1 && tree->right->label == 0)

{

generateCode(tree->left, R, top, numOfRegisters);

std::string op = operationName(tree->data);

std::cout << op << " " << tree->right->data << ", ";

if (top == numOfRegisters)

std::cout << "T";

else

std::cout << "R[" << top << "]";

std::cout << "\n";

}

else if (tree->left->label < tree->right->label)

{

swap(R[top], R[top - 1]);

generateCode(tree->right, R, top, numOfRegisters);

int temp = pop(R, top);

generateCode(tree->left, R, top, numOfRegisters);

push(R, top, temp, numOfRegisters);

swap(R[top], R[top - 1]);

std::string op = operationName(tree->data);

std::cout << op << " R[" << top - 1 << "], ";

if (top == numOfRegisters)

std::cout << "T";

else

std::cout << "R[" << top << "]";

std::cout << "\n";

}

else if (tree->left->label >= tree->right->label)

{

int temp;

generateCode(tree->left, R, top, numOfRegisters);

temp = pop(R, top);

generateCode(tree->right, R, top, numOfRegisters);

push(R, top, temp, numOfRegisters);

std::string op = operationName(tree->data);

std::cout << op << " R[" << top - 1 << "], ";

if (top == numOfRegisters)

std::cout << "T";

else

std::cout << "R[" << top << "]";

std::cout << "\n";

}

}

else if (tree->left == nullptr && tree->right == nullptr && tree->label == 1)

{

std::cout << "MOV " << tree->data << ", ";

if (top == numOfRegisters)

std::cout << "T";

else

std::cout << "R[" << top << "]";

std::cout << "\n";

}

}

void deleteTree(node\* tree)

{

if (tree)

{

deleteTree(tree->left);

deleteTree(tree->right);

delete tree;

}

}

int main()

{

node\* root = nullptr;

node\* tmp;

char val;

int i, temp;

std::cout << "Enter root of tree: ";

std::cin >> val;

insert(&root, val);

findLeafNodeLabel(root, 1);

while (root->label == -1)

findInteriorNodeLabel(root);

int numOfRegisters;

std::cout << "Enter the number of registers available: ";

std::cin >> numOfRegisters;

int\* R = new int[numOfRegisters];

temp = numOfRegisters - 1;

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

{

R[i] = temp;

temp--;

}

std::cout << "\nInorder Display:\n";

printInorder(root);

std::cout << "\nAssembly Code:\n";

int top = root->label - 1;

generateCode(root, R, top, numOfRegisters);

deleteTree(root);

delete[] R;

return 0;

}