**REMOVAL OF REDUNDANT PARENTHESIS**

**-411746**

**-N C S Jagannath**

* I have constructed a translation schema for the conversion of the expression with parenthesis to removal of the not required parenthesis.
* Firstly it takes the expression and converts it into the stream of lexems which will be fed to the parser for constructing a parse tree.
* The grammar used for accomplishing the above task is as follows:

S->idl = {print(‘=’)} E

E->TR

T->FR1

R->+{print(‘+’)}TR|-{print(‘-‘)}TR

R1->\*{print(‘\*’)FR1|/{print(‘/’)}FR1

F-> ( { if (!checkredu(expr){print(‘(‘);value=value+1;} E) { if (value>0) print(‘}’)

F->id

Here chechredu is a function that checks whether the ‘(‘ is redundant or not.

Value is a global variable which keeps tracks the no.of open parenthesis to maintain homogenicity.

I have taken the fact that whether a parenthesis to be kept or depends upon the next two operators and the parenthesis between them

Ex: (a+b)\*c here the next two operators and parenthesis of ( is +,),\* which gives us the parenthesis are required.

While for (a+b)+c here the next two operators of ( is +,),+ where the parenthesis are not required.

I have enumerated all the ways that required. They are:

+)\* , -)\* , +)/ , -)/ ,-)- ,/)/ ,\*)/ ,/)\*.

And it also depends the previous operator and immediate next operator.

They are:

\*(+ , \*(- , /(+ , /(- , -(- , /(/ .

The codes are as follows:

**Header.h**

#include<iostream>

#include<cstdio>

#include<string.h>

using namespace std;

//creating a token.....

struct token{

int id;

char var;

string type;

token \*next;

};

//functions in lexer....

char firsttoken(token\*\*,string);

char gettoken(int);

void maketoken(string);

//functions in parser....

void s(char,string);

void idl(string);

void E(string);

void T(string);

void F(string);

void R(string);

void R1(string);

void match(char a);

**lexer.cpp**

*#include "header.h"*

*token \*temp1=NULL;*

*token \*temp2=NULL;*

*//cerates the first token returns the value and calls the maketoken..........*

*char firsttoken(token\*\* head,string buffer){*

*token \*new\_token=new token();*

*new\_token->id=0;*

*new\_token->next=NULL;*

*if (isalpha(buffer[0])){*

*new\_token->type="var";*

*new\_token->var=buffer[0];*

*}*

*else{*

*cout<<"lexical error expected a variable\n";*

*}*

*\*head=new\_token;*

*temp1=\*head;*

*temp2=\*head;*

*maketoken(buffer);*

*return buffer[0];*

*}*

*//it is a function used by parser for acquiring the next required token....*

*char gettoken(int j){*

*token \*temp3=temp1;*

*int i;*

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

*temp3=temp3->next;*

*}*

*return temp3->var;*

*}*

*//it creates the stream of tokens(linkedlist).......*

*void maketoken(string buffer){*

*int i=1;*

*for(i=1;i<=buffer.length();i++){*

*token\* new\_token=new token();*

*new\_token->id=i;*

*new\_token->next=NULL;*

*if(isalpha(buffer[i])){*

*new\_token->type="var";*

*new\_token->var=buffer[i];*

*}*

*else{*

*new\_token->type="op";*

*new\_token->var=buffer[i];*

*}*

*temp2->next=new\_token;*

*temp2=temp2->next;*

*}*

*return;*

*}*

**Parser.cpp**

#include "header.h"

int i=0;

int value=0;

char lad;

void s(char lat,string buffer){

lad=lat;

idl(buffer);

match('=');

cout<<"=";

E(buffer);

return;

}

void idl(string buffer){

cout<<lad;

i=i+1;

lad=gettoken(i);

return;

}

void E(string buffer){

T(buffer);

R(buffer);

return;

}

void T(string buffer){

F(buffer);

R1(buffer);

return;

}

void R(string buffer){

if(lad=='+'){

match('+');

cout<<"+";

T(buffer);

R(buffer);

}

else if(lad=='-'){

match('-');

cout<<"-";

T(buffer);

R(buffer);

}

return;

}

void R1(string buffer){

if(lad=='\*'){

match('\*');

cout<<"\*";

F(buffer);

R1(buffer);

}

else if (lad=='/'){

match('/');

cout<<"/";

F(buffer);

R1(buffer);

}

return;

}

void F(string buffer){

if(lad=='('){

int j=i-1;

match('(');

string sa;

for(int r=j;r<buffer.length();r++){

if(buffer[r]=='+' || buffer[r]=='-' ||buffer[r]=='\*' ||buffer[r]=='/' ||buffer[r]=='(' ||buffer[r]==')' || buffer[r]=='=' ){

sa+=buffer[r];

}

}

char str2[3];

if(sa.length()>=5)

{

str2[0]=sa[2];

str2[1]=sa[3];

for(int k=4;k<100;k++){

if(sa[k]=='+'||sa[k]=='-'||sa[k]=='\*'||sa[k]=='/'){

str2[2]=sa[k];

break;

}

}

if(!strcmp(str2,"+)\*") || !strcmp(str2,"-)\*") || !strcmp(str2,"+)/") || !strcmp(str2,"-)/") || !strcmp(str2,"-)-") || !strcmp(str2,"/)/")||!strcmp(str2,"\*)/") || !strcmp(str2,"/)\*")){

cout<<"(";

value=value+1;

}}

char str1[3];

str1[0]=sa[0];

str1[1]=sa[1];

str1[2]=sa[2];

if(!strcmp(str1,"\*(+") || !strcmp(str1,"\*(-") || !strcmp(str1,"/(+") || !strcmp(str1,"/(-") || !strcmp(str1,"-(-") || !strcmp(str1,"/(/")){

cout<<"(";

value=value+1;

}

E(buffer);

match(')');

if(value>0){

cout<<")";

value=value-1;

}

}

else{

cout<<lad;

i=i+1;

lad=gettoken(i);

}

return;

}

void match(char a){

if(lad==a){

i=i+1;

lad=gettoken(i);

}

else{

cout<<"\n error expected\t"<<a<<"but got"<<lad<<"\n";

}

return;

}

**Main.cpp**

#include"header.h"

int main(){

string buffer;

char lat;

cout<<"enter the expression in the for var=expr";

getline(cin,buffer);

token \*head=NULL;

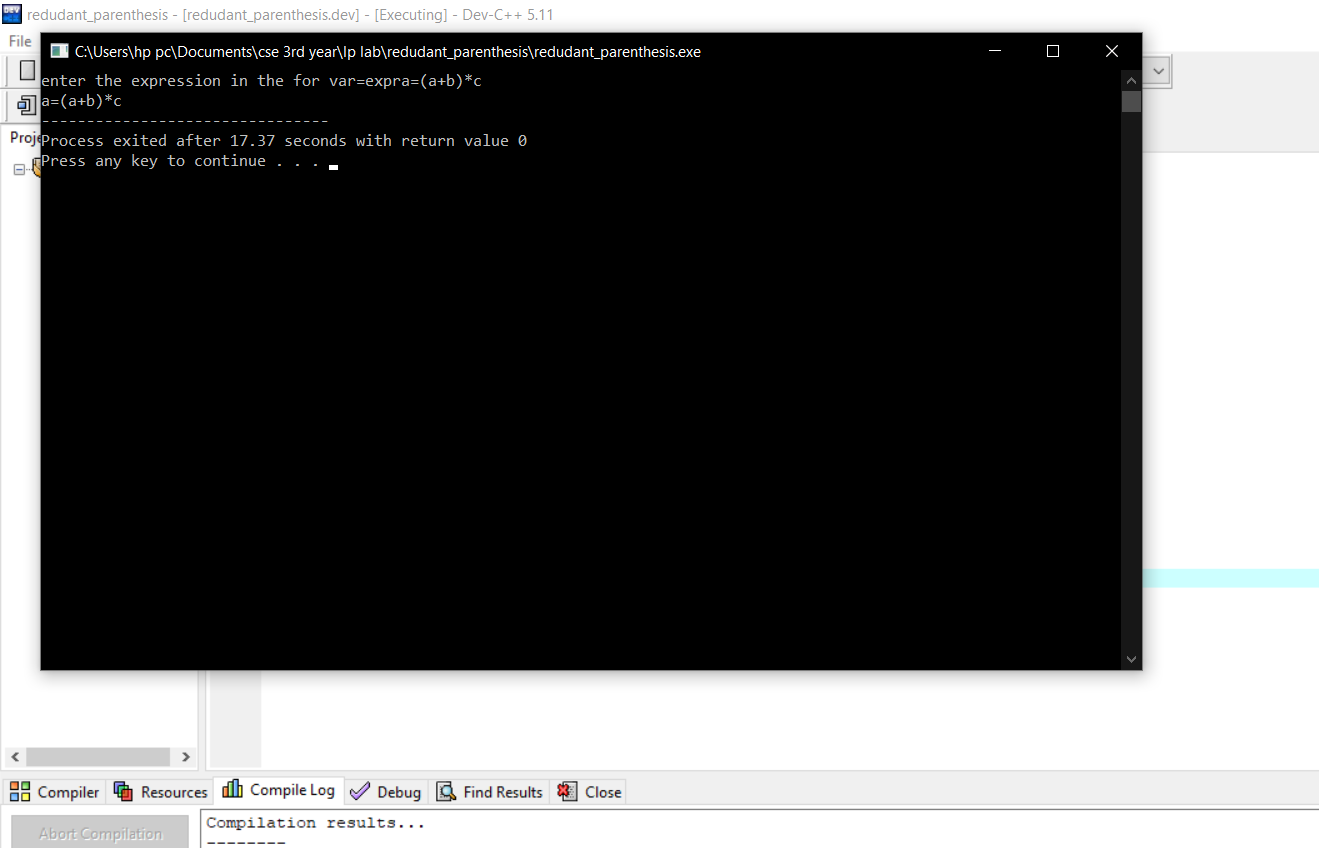
lat=firsttoken(&head,buffer);

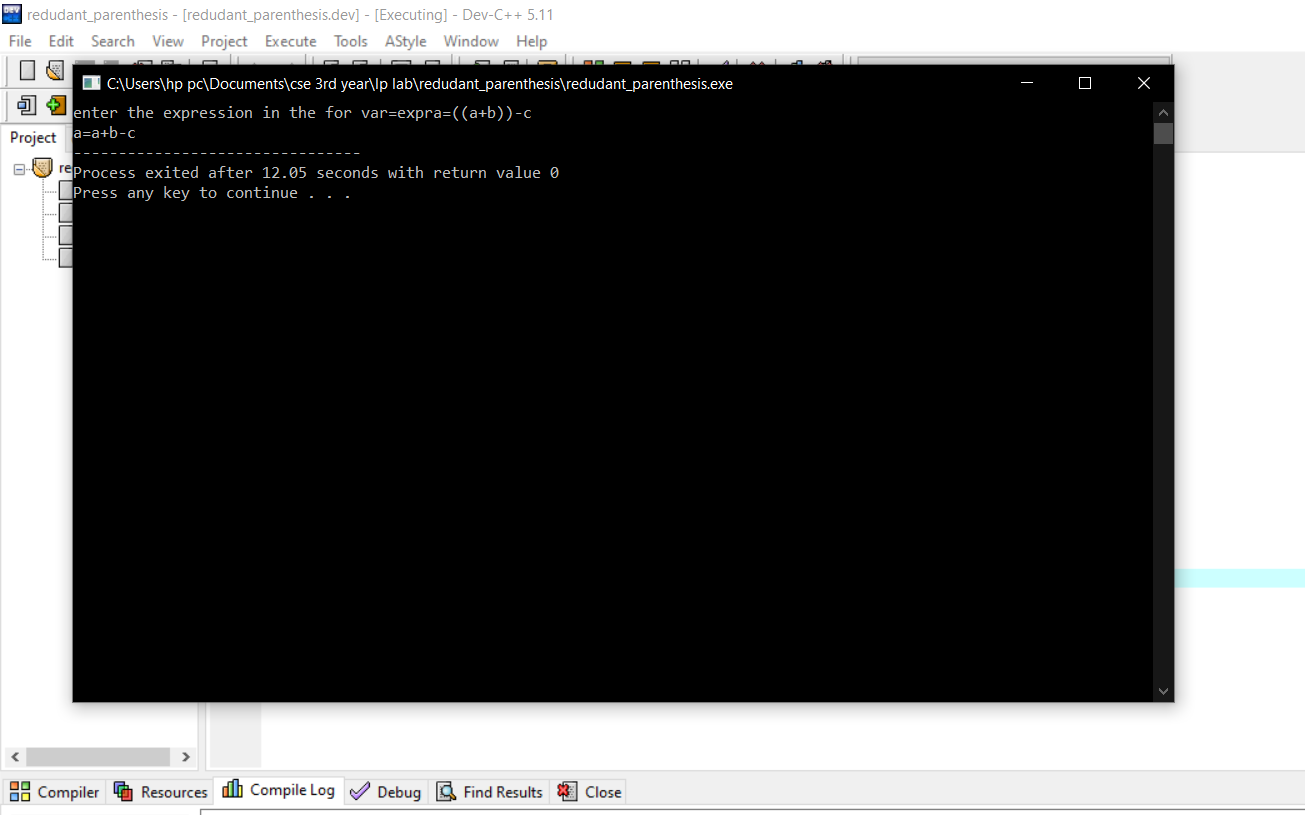
s(lat,buffer);

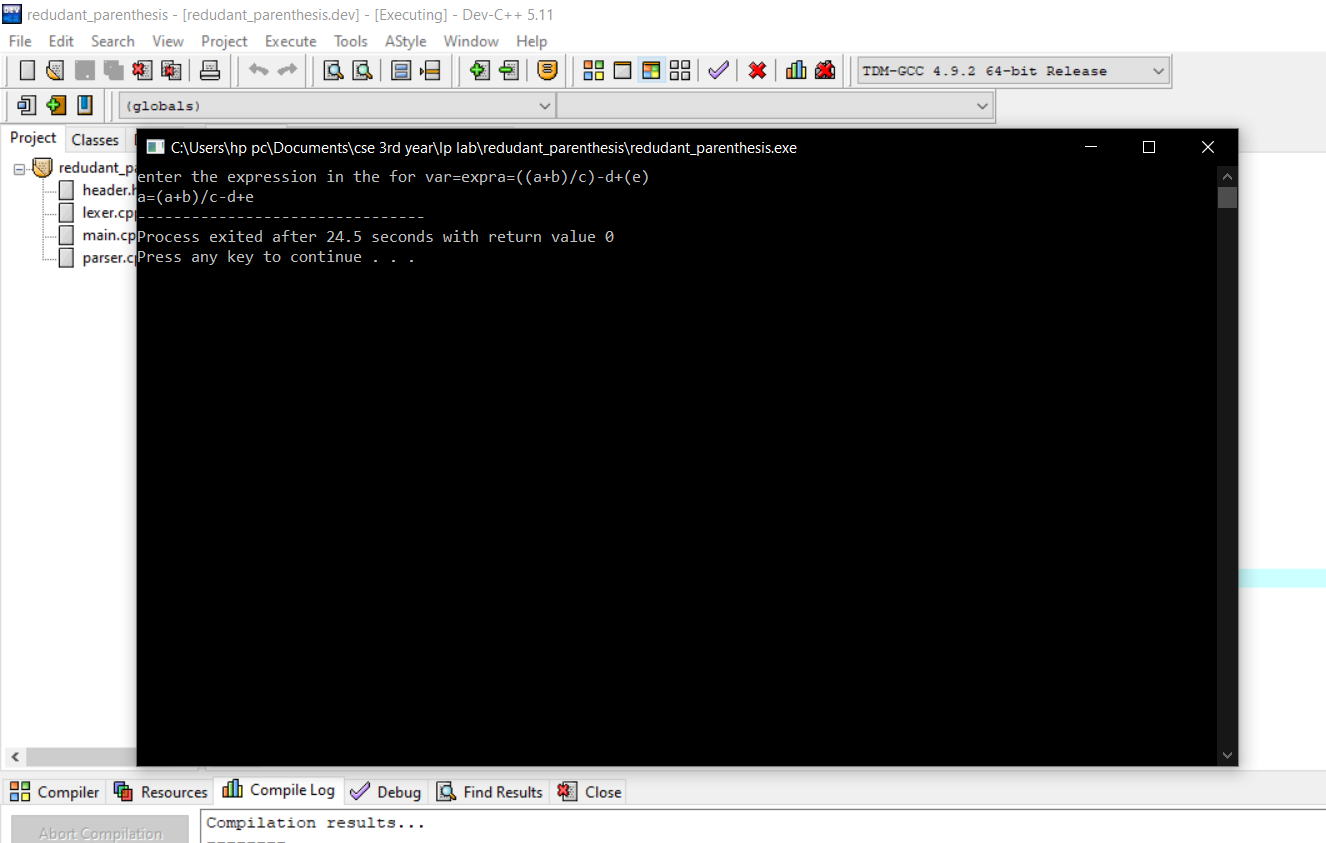
return 0;

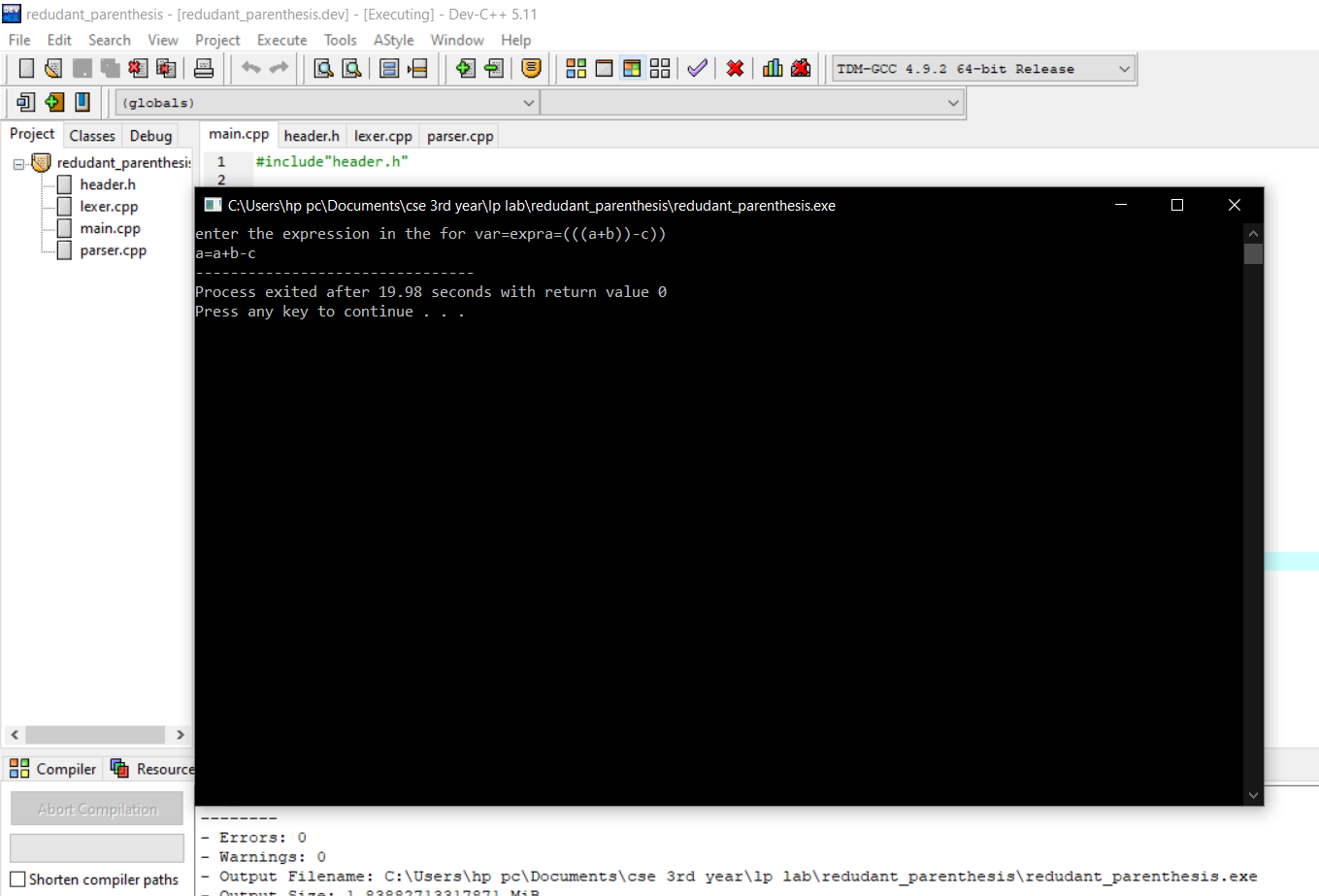
}

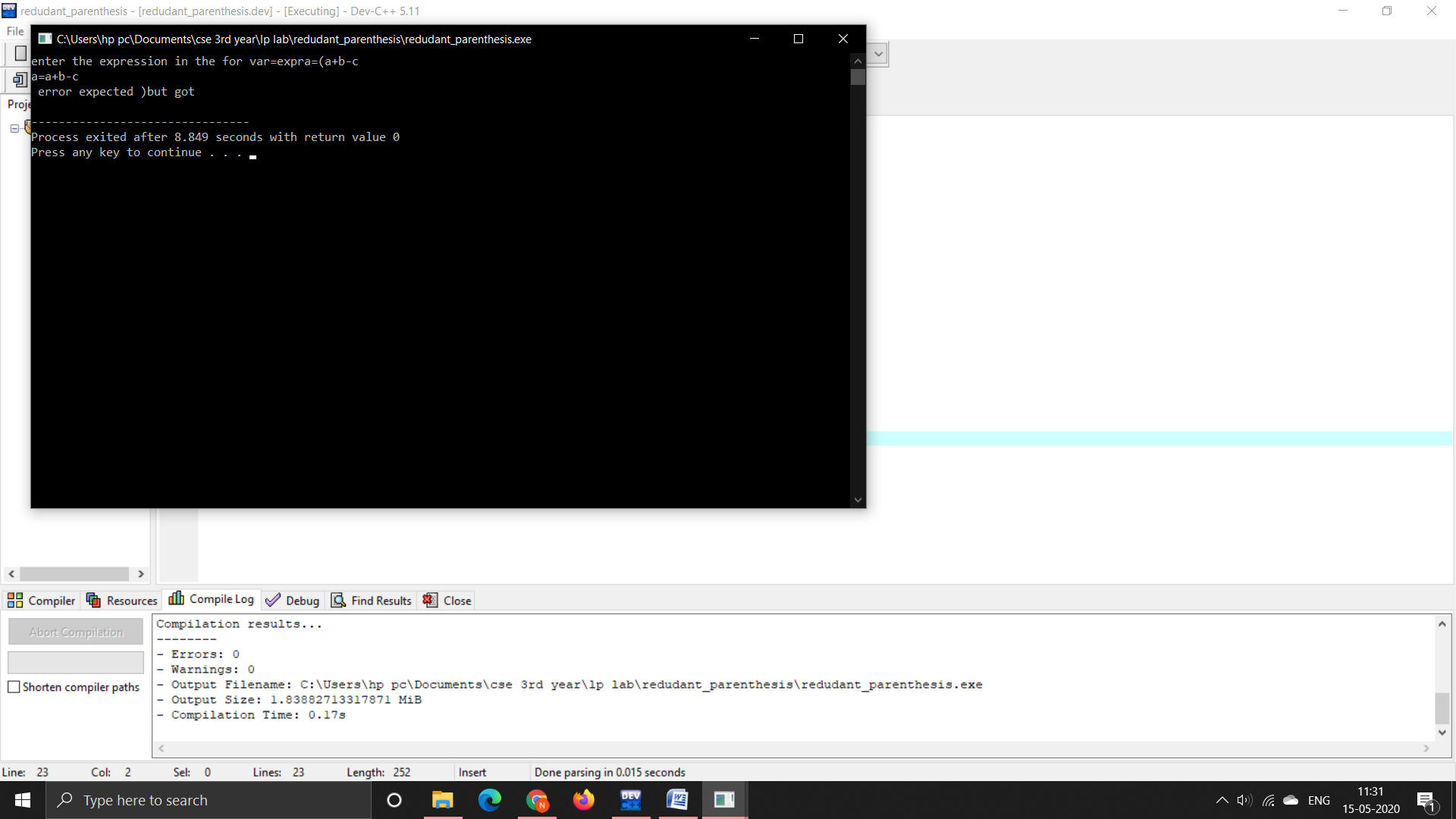
Some outputs are:











\*error routines are included in the parser and lexer which may be not great just incorporated basic ones.

\*only one thing the parser not handles is empty parenthesis(for this we can write another check function to handle it).

a=()+b;

it throws error but the required output may not come.

Here output is a=b;

But we get a=)+b

**THANK YOU**