**SETHI ULLMAN ALGORITHM**...

N C S JAGANNATH

411746.

* I have only considered the two phases one is parsing(which is helpful in constructing the expression tree) and the code generation using the sethi ullman algorithm.
* The grammar used for building the expression tree is as follows

E->TR

T->FR1

R->+{makenode(‘+’)}TR | -{makenode('-‘)TR

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

F->(E) | id{makenode(‘id’)

Here the makenode(char\*) is a function which makes them node and joins them to the thing where is to be...

* After building an expression tree in the above format then it is fed to the gencode function which churns out the code
* The gencode function is as same as the discussed

The code is as follows:

Sethi\_ullman.cpp

//required header files,...

#include<iostream>

#include<cstdio>

#include<string.h>

#include<bits/stdc++.h>

using namespace std;

//structure for the node in the expression tree...

struct node{

node \*lft;

char var;

int reg;

node \*rght;

};

//stacks of permanent and temporary registers....

stack <string> rstack;

stack <string> tstack;

//node pointers that req. to built expression tree

node \*left\_t=NULL;

node \*right\_t=NULL;

node \*root=NULL;

node \*center\_t=NULL;

string buffer;

char lah,lat;

int i=0;

int value=0;//helpful to distinct expression with or without braces....

//SDT funcitons....

void s();

void idl();

void E();

void T();

void F();

void R();

void R1();

void match(char a);

//function for constructing the terminal nodes....only operators....

void makenode(char a){

node \*newnode=new node();

newnode->lft=left\_t;

newnode->rght=right\_t;

newnode->var=a;

if(newnode->lft!=NULL && newnode->rght!=NULL){

if(newnode->lft->reg==newnode->rght->reg)

newnode->reg=newnode->lft->reg+1;

else if(newnode->lft->reg>newnode->rght->reg)

newnode->reg=newnode->lft->reg;

else

newnode->reg=newnode->rght->reg;

}

else if(newnode->lft==NULL){

newnode->reg=newnode->rght->reg;

}

else

newnode->reg=newnode->lft->reg;

if(value==0){

if(center\_t==NULL)

root=newnode;

else if(center\_t->lft==NULL)

center\_t->lft=newnode;

else

center\_t->rght=newnode;

center\_t=newnode;

left\_t=NULL;

right\_t=NULL;

}

else{

left\_t=NULL;

right\_t=NULL;

if(center\_t->lft==NULL){

center\_t->lft=newnode;

}

else{

center\_t->rght=newnode;

}

}

}

void E(){

T();

R();

return;

}

void T(){

F();

R1();

return;

}

void R(){

if(lat=='+'){

match('+');

makenode('+');

T();

R();

}

else if(lat=='-'){

match('-');

makenode('-');

T();

R();

}

return;

}

void R1(){

if(lat=='\*'){

match('\*');

makenode('\*');

F();

R1();

}

else if (lat=='/'){

match('/');

makenode('/');

F();

R1();

}

return;

}

void F(){

if(lat=='('){

match('(');

value=value+1;

E();

match(')');

value=value-1;

}

else{

if(value>0){

//node for id inside a parenthesis

node \*newnode=new node();

if(left\_t==NULL){

newnode->lft=NULL;

newnode->reg=1;

newnode->var=lat;

newnode->rght=NULL;

left\_t=newnode;

}

else if(right\_t==NULL){

newnode->lft=NULL;

newnode->reg=0;

newnode->var=lat;

newnode->rght=NULL;

right\_t=newnode;

}

if(center\_t!=NULL){

center\_t->rght=newnode;

left\_t=center\_t;

right\_t=center\_t;

center\_t=NULL;

}

}

else{

node \*newnode=new node();

//node for id not in parenthesis.....

if(left\_t==NULL){

newnode->lft=NULL;

newnode->reg=1;

newnode->var=lat;

newnode->rght=NULL;

left\_t=newnode;

if(center\_t!=NULL){

if(center\_t->lft==NULL){

center\_t->lft=newnode;

}

else

center\_t->rght=newnode;

}

}

else if(right\_t==NULL){

newnode->lft=NULL;

newnode->reg=0;

newnode->var=lat;

newnode->rght=NULL;

right\_t=newnode;

if(center\_t!=NULL){

if(center\_t->lft==NULL){

center\_t->lft=newnode;

}

else

center\_t->rght=newnode;

}

}

}

i++;

lat=buffer[i];

}

return;

}

void match(char a){

if(lat==a){

i=i+1;

lat=buffer[i];

}

else{

cout<<"error";

}

return;

}

//helper function in gencode.....

void swap(stack <string> s){

string X,Y;

X=s.top();

s.pop();

Y=s.top();

s.push(Y);

s.push(X);

return;

}

void gencode(node \*root){

if(root->lft!=NULL){

if((root->lft)->lft==NULL&&root->lft->rght==NULL){

//left child is leaf node

cout<<rstack.top()<<"<-"<<root->lft->var<<"\n";

}}

if(root->rght!=NULL){

if(root->rght->lft==NULL&&root->rght->lft==NULL){

//right child is leaf node......

gencode(root->lft);

cout<<rstack.top()<<"<-"<<rstack.top()<<root->var<<root->rght->var<<"\n";

}}

if(root->lft!=NULL&&root->rght!=NULL){

if(root->lft->reg<root->rght->reg && root->rght->reg<3 ){

//right child requires more registers and less than total

string X;

swap(rstack);

gencode(root->rght);

X=rstack.top();

rstack.pop();

gencode(root->lft);

cout<<rstack.top()<<"<-"<<rstack.top()<<root->var<<"\t"<<X<<"\n";

rstack.push(X);

swap(rstack);

}

else if(root->lft->reg>=root->rght->reg && root->lft->reg<3 ){

//left child requires more registers and less than total

string X;

gencode(root->lft);

X=rstack.top();

rstack.pop();

gencode(root->rght);

cout<<rstack.top()<<"<-"<<rstack.top()<<root->var<<"\t"<<X<<"\n";

rstack.push(X);;

}

else if(root->lft->reg>=3&&root->rght->reg>=3){

//both require more registers....

string X;

gencode(root->rght);

X=tstack.top();

tstack.pop();

cout<<X<<"<-"<<rstack.top();

gencode(root->lft);

tstack.push(X);

cout<<rstack.top()<<"<-"<<rstack.top()<<root->var<<X<<"\n";

}}

return;

}

int main(){

cout<<"enter the expression";

getline(cin,buffer);

lat=buffer[i];

E();

//no. of permanent registers 3

rstack.push("r2");

rstack.push("r1");

rstack.push("r0");

//no .of temporary registers 2

tstack.push("t1");

tstack.push("t0");

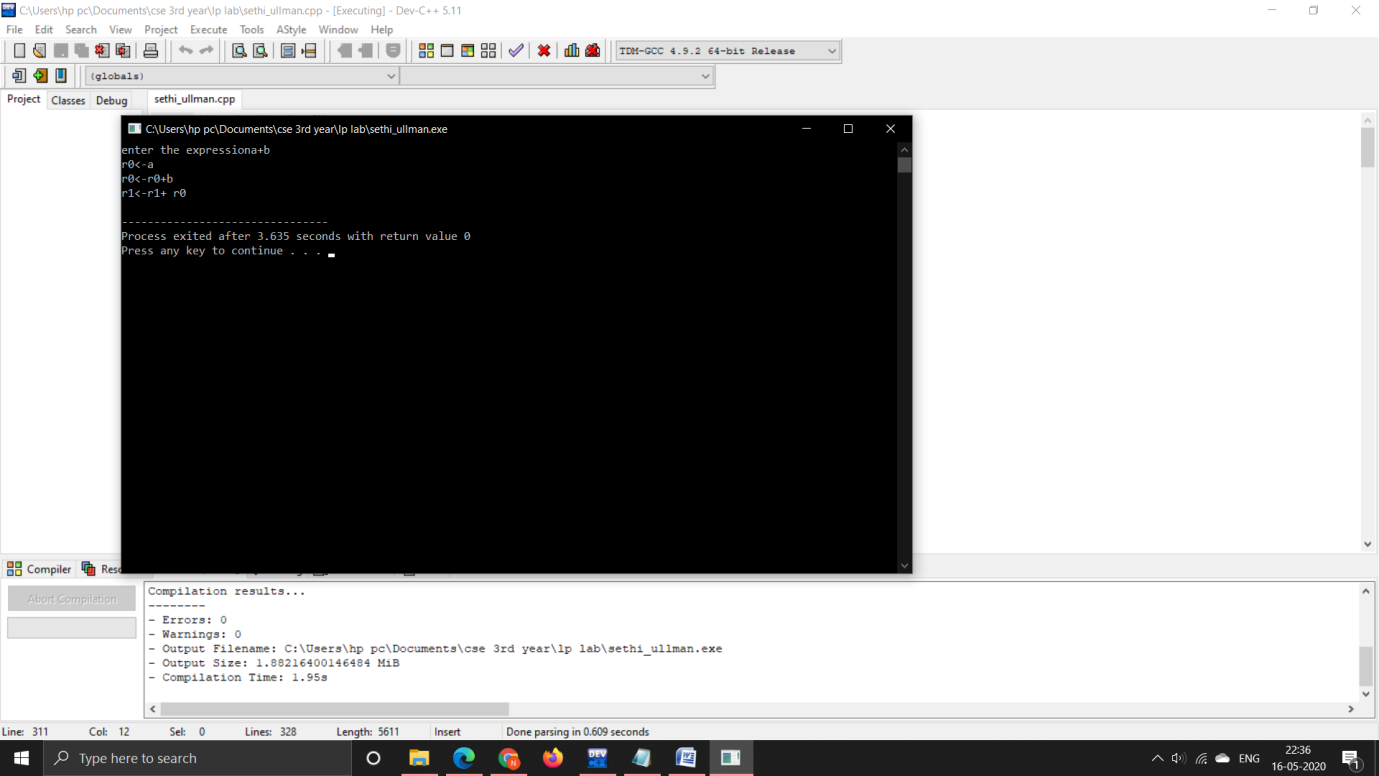
//generates code....

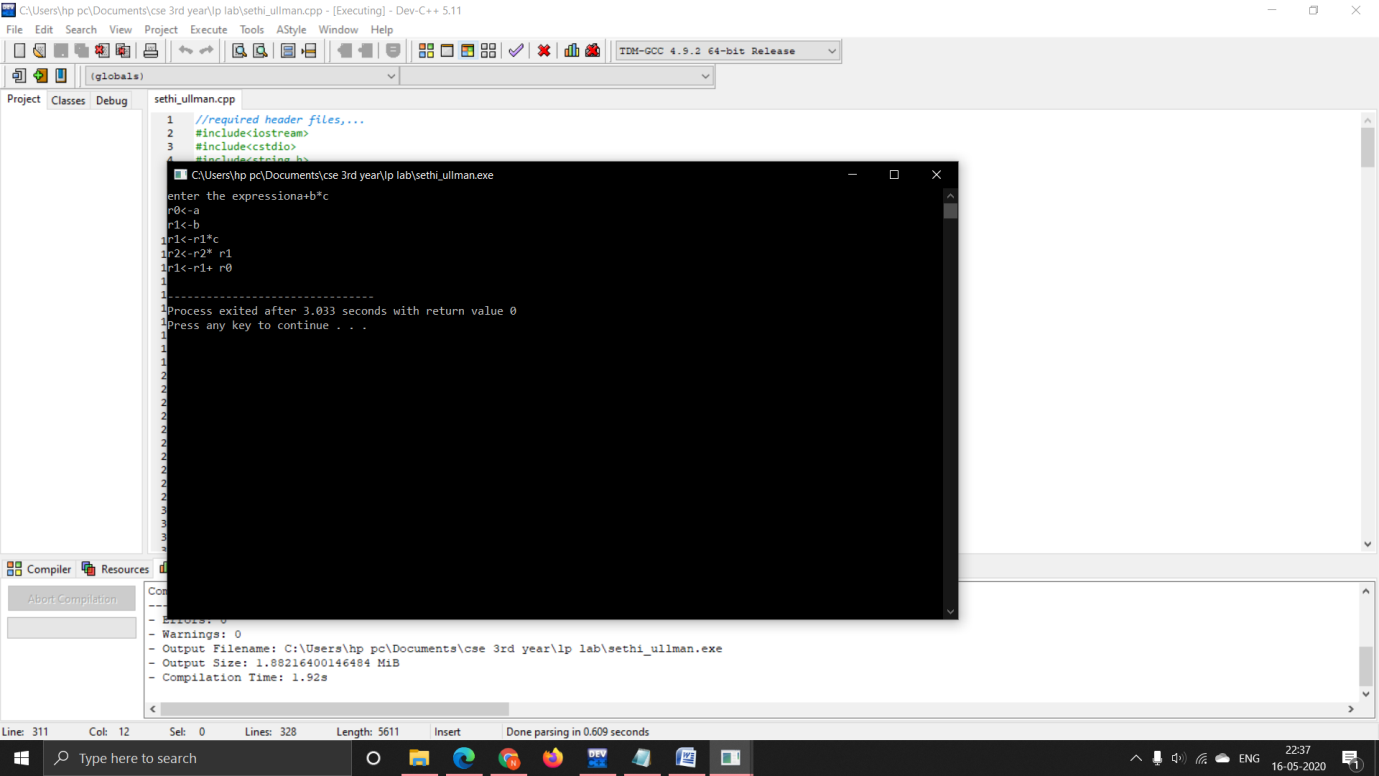
gencode(root);

return 0;

}

Some of the outputs are:





I have done parsing and code generation in a single program...

THANK YOU.........