Open Ended Experiment

Compiler Construction

(CSE 304)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted to:

Dr. Misha Kakkar

Assistant Professor

CSE Department, ASE

Submitted by:

Shaina Mehta

A2305219268

B.tech. C.S.E.

6CSE 4Y

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

NOIDA -201301

Experiment

Aim: To construct a mini compiler.

Theory:

Proposed Language:

The language that I have proposed here is of operator grammar of 3 arithmetic expressions '+', '*', and '=' along with two statements 'read' and 'print' statements. Alphabets and underscores are used as an identifier. Here, we are only assuming integer datatypes. The production rules of the grammar are given below:

S->A;

A->V=E|R|O

V->id

C->num

E->E+T

E->T

T->T*F

T->F

F->(E)|C|V

R->read(V)

O->print(C)|print(V)

Implementation:

Here, I have made three parts:

- **Lexical Part –** Here I have done tokenization and then categorization of tokens and removed the comments and extra spaces from there which is further used for syntax analysis and intermediate code generation.
- **Syntax Part –** Here, I have used SLR parser for it for parsing the string and checking the syntax errors. If the parser finds any syntax error in it then, it will terminate the program and then, then, it will raise the error.
- **Intermediate Code Generator -** Here, I have converted the infix expression into a postfix expression and then used the logic of evaluation of the postfix expression to convert it into 3 address code.

Note –

- If you don't place the semicolon after every line, then it will throw an error.
- Please leave the space after writing every token in the program.
- In the intermediate code generation phase, intermediate code for print and read statements has not been created. This will be taken as a future work

Syntax:

```
// This is a single line comment
/* This is a multi line comment */
read ( a ) ;
read ( b ) ;
c = a + b ;
print ( c ) ;
```

Code:

```
#include <iostream>
#include <string>
#include <stack>
#include <vector>
#include <fstream>
#include <regex>
#include <algorithm>
#include <cctype>
using namespace std;
vector<vector<string>> Action = {{"S3","S4","@","@","@","@","@","@","S8","S7","@"},
    {"@","@","@","@","@","@","S9","@","@","@","@"},
     {"@","@","@","@","@","@","@","S10","@","@","@"},
    {"@","@","R4","R4","@","R4","R4","R4","@","@","@"},
    {"@","@","R5","R5","@","R5","R5","@","@","@","@"},
    {"@","@","@","@","@","@","R2","@","@","@","@"},
    {"@","@","@","@","@","@","R3","@","@","@","@"},
    {"@","@","@","@","S11","@","@","@","@","@","@"},
    {"@","@","@","@","S12","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@","@","Accept"},
    {"S3","S4","@","@","S16","@","@","@","@","@","@"},
    {"S3","@","@","@","@","@","@","@","@","@","@"},
    {"S3","S4","@","@","@","@","@","@","@","@","@"},
    {"@","@","S21","@","@","@","R1","@","@","@","@"},
```

```
    {"@","@","R7","S22","@","R7","R7","@","@","@","@"},
    {"@","@","R9","R9","@","R9","R9","@","@","@","@"},
    {"S3","S4","@","@","S16","@","@","@","@","@","@"},
    {"@","@","R11","R11","@","R11","R11","@","@","@","@"},
    {"@","@","@","@","@","S24","@","@","@","@","@"},
    {"@","@","@","@","@","S25","@","@","@","@","@"},
    {"@","@","@","@","@","S26","@","@","@","@","@"},
    {"S3","S4","@","@","S16","@","@","@","@","@","@"},
    {"S3","S4","@","@","S16","@","@","@","@","@","@"},
    {"@","@","S21","@","@","S29","@","@","@","@","@"},
    {"@","@","@","@","@","@","R12","@","@","@","@"},
    {"@","@","@","@","@","@","R13","@","@","@","@"},
    {"@","@","@","@","@","@","R14","@","@","@","@"},
    {"@","@","R6","S22","@","R6","R6","@","@","@","@"},
    {"@","@","R8","R8","@","R8","R8","@","@","@","@"},
    {"@","@","R10","R10","@","R10","R10","@","@","@","@"},
    {"@","@","R15","R15","@","R15","R15","@","@","@","@"}};
vector<vector<string>> Goto = {{"@","1","2","@","@","@","@","5","6"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","17","30","13","14","15","@","@"},
    {"@","@","18","@","@","@","@","@","@"},
    {"@","@","19","20","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
```

```
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","17","30","23","14","15","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","17","30","@","27","15","@","@"},
    {"@","@","17","30","@","@","28","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"},
    {"@","@","@","@","@","@","@","@","@"}};
vector <string> term = {"id","num","+","*","(",")",";","=","print","read","$"};
vector <string> nonterm = {"S","A","V","C","E","T","F","R","O"};
vector <vector<string>> prod = {{"S","->","A",";"},
                                {"A","->","V","=","E"},
                                {"A","->","R"},
                                {"A","->","O"},
                                {"V","->","id"},
                                {"C","->","num"},
                                {"E","->","E","+","T"},
                                {"E","->","T"},
                                {"T","->","T","*","F"},
                                {"T","->","F"},
                                {"F","->","(","E",")"},
                                {"F","->","V"},
```

```
                        {"R","->","read","(","V",")"},
                        {"O","->","print","(","V",")"},
                        {"O","->","print","(","C",")"},
                        {"F","->","C"}};
int it = 0;
void infixToPostfix(vector<string> inExp,vector<string> &postExp){
    stack <string> s;
    for(int i=0;i<inExp.size();i++){
        if(inExp[i].compare("(")==0){
            s.push(inExp[i]);
        }
        else if(inExp[i].compare(")")==0){
            while(!s.empty() && s.top().compare("(")!=0){
                postExp.push_back(s.top());
                s.pop();
            }
            s.pop();
        }
        else if(inExp[i].compare("^")==0){
            if(s.empty()){
                s.push(inExp[i]);
            }
            else
            {
                string a = s.top();
                if(a.compare("^")==0){
                    postExp.push_back(a);
                    s.pop();
                    while(!s.empty()){
                        string b = s.top();
                        if(b.compare("^")==0||b.compare("*")==0||b.compare("/")==0){
```

```
                postExp.push_back(b);
                s.pop();
            }
            else{
                break;
            }
        }
    s.push(inExp[i]);
    }
    else{
        s.push(inExp[i]);
    }
  }
}
else if(inExp[i].compare("*")==0){
  if(s.empty()){
    s.push(inExp[i]);
  }
  else{
    string a = s.top();
    if(a.compare("^")==0||a .compare("*")==0||a.compare("/")==0){
    while(!s.empty()){
        string b = s.top();
        if (b.compare("^")==0||b.compare("*")==0||b.compare("/")==0){
            postExp.push_back(b);
            s.pop();
        }
        else{
            break;
        }
    }
```

```
        s.push(inExp[i]);
        }
        else{
            s.push(inExp[i]);
        }
    }
}
else if(inExp[i].compare("/")==0){
    if(s.empty()){
        s.push(inExp[i]);
    }
    else{
        string a = s.top();
        if(a.compare("^")==0||a.compare("*")==0||a.compare("/")==0){
        while(!s.empty()){
            string b = s.top();
            if(b.compare("^")==0||b.compare("*")==0||b.compare("/")==0){
                string b = s.top();
                postExp.push_back(b);
                s.pop();
            }
            else{
                break;
            }
        }
         s.push(inExp[i]);
        }
        else{
             s.push(inExp[i]);
        }
    }
```

```
    }
    else if(inExp[i].compare("+")==0){
        if(s.empty()){
            s.push(inExp[i]);
        }
        else{
            string a = s.top();

if(a.compare("^")==0||a.compare("*")==0||a.compare("/")==0||a.compare("+")==0||a.compare("-")==0){
                while(!s.empty()){
                    string b = s.top();

if(b.compare("^")==0||b.compare("*")==0||b.compare("/")==0||b.compare("+")==0||b.compare("-")==0){
                        string b = s.top();
                        postExp.push_back(b);
                        s.pop();
                    }
                    else{
                        break;
                    }
                }
                 s.push(inExp[i]);
                }
                else{
                    s.push(inExp[i]);
                }
        }
    }
    else if(inExp[i].compare("-")==0){
        if(s.empty()){
```

```
            s.push(inExp[i]);
        }
        else{
            string a = s.top();

if(a.compare("^")==0||a.compare("*")==0||a.compare("/")==0||a.compare("+")==0||a.compare("-")==0){
            while(!s.empty()){
                string b = s.top();

if(b.compare("^")==0||b.compare("*")==0||b.compare("/")==0||b.compare("+")==0||b.compare("-")==0){
                    string b = s.top();
                    postExp.push_back(b);
                    s.pop();
                }
                else{
                    break;
                }
            }
             s.push(inExp[i]);
            }
            else{
                s.push(inExp[i]);
            }
        }
      }
      else{
        postExp.push_back(inExp[i]);
      }
   }
   while(!s.empty()){
```

```
        postExp.push_back(s.top());
        s.pop();
    }
}
void TACGenerate(vector <string> input, vector <string> &code)
{
    //vector <string> v;
    regex r("[0-9]+");
    stack <string> s;
    //int len=strlen(output);
    //puts(output);
    string e = input[0]+" "+input[1];
    vector <string> inExp;
    vector <string> output;
    for(int i=2;i<input.size();i++){
        inExp.push_back(input[i]);
    }
    infixToPostfix(inExp,output);
    for(int i=0;i<output.size();i++)
    {
        if(regex_match(output[i],r) || output[i].at(0) == '_' || (output[i].at(0)>='A' && output[i].at(0) <='Z') || (output[i].at(0)>='a' && output[i].at(0)<='z'))
        {
            s.push(output[i]);
        }
        else
        {
            string a= s.top();
            //printf("%d ",a);
            s.pop();
            string b= s.top();
```

```
            //printf("%d ",b);
            s.pop();
            string c;
            if(output[i].compare("+")==0)
            {
                code.push_back("temp" + to_string(it) + " = " + b + " + " + a);
            }
            else
            {
                code.push_back("temp" + to_string(it) + " = " + b + " * " + a);
            }
            s.push("temp"+to_string(it));
            it++;
        }
    }
    string an = s.top();
    s.pop();
    code.push_back(e+" "+an);
}
int parse(vector <string> input){
    int flag = 0;
    input.push_back("$");
    vector<string> stack;
    stack.push_back("0");
    int itr = 0;
    /*
    for(int i=0;i<input.size();i++){
        cout<<input[i]<<" ";
    }*/
    while(1){
        int state = stoi(stack[stack.size()-1]);
```

```
string a = input[itr];
//cout<<itr<<" "<<input[itr]<<" ";
int sym = find(term.begin(),term.end(),a) - term.begin();
//cout<<state<<" "<<sym<<" "<<a<<" "<<Action[state][sym]<<endl;
/*
if(itr==4){
   break;
}*/
if(Action[state][sym].at(0) == 'S'){
   stack.push_back(a);
   stack.push_back(Action[state][sym].substr(1));
   itr++;
   /*
   if(itr==5){
      break;
   }*/
}
else if(Action[state][sym].at(0) == 'R'){
   stack.pop_back();
   int pro = stoi(Action[state][sym].substr(1));
   //cout<<pro<<endl;
   string lhs = prod[pro].at(0);
   //string rhs = prod[pro].substr(3);
   /*
   for(int i=0;i<=rhs.length()-1;i++){
      //cout<<rhs;
      stack.pop_back();
   }*/
   int i=0;
   int n = prod[pro].size()-2;
   //cout<<n<<endl;
```

```
    if(n==1){
        stack.pop_back();
    }
    else{
        while(i<((2*n)-1)){
            stack.pop_back();
            i++;
        }
    }
    /*
    if(itr==4){
        break;
    }*/
    //for(int i=0;i<rhs.length()-1)
    //cout<<lhs<<" "<<rhs<<endl;
    //cout<<lhs;
    stack.push_back(lhs);
    //cout<<typeid().name()<<endl;
    string b = stack.at(stack.size()-2);
    //cout<<b<<endl;
    //cout<<lhs;
    int s = stoi(b);
    int sy = find(nonterm.begin(),nonterm.end(),lhs) - nonterm.begin();
    stack.push_back(Goto[s][sy]);
    //cout<<stack.at(stack.size()-1);
    //break;
}
else if(Action[state][sym].compare("Accept")==0){
    cout<<"The string has been parsed successfully."<<endl;
    break;
}
```

```
        else{
            cout<<"Error!!!"<<endl;
            cout<<"The string could not be parsed successfully,"<<endl;
            flag = 1;
            break;
        }
    }
    return flag;
}
string category(string token){
    string objects[] = {"print","read"};        //predefined identifiers included in header files
    string operators[] = {"+","*","="};
    string delim[] = {"(",")",";"};
    //regex b("[0-9]+");
    for(int i=0; i< sizeof(operators)/sizeof(operators[0]); i++){
        if(token == operators[i]){
            return token;
        }
    }
    for(int i=0; i< sizeof(delim)/sizeof(delim[0]); i++){
        if(token == delim[i]){
            return token;
        }
    }
    for(int i=0; i< sizeof(objects)/sizeof(objects[0]); i++){
        if(token == objects[i]){
            return token;
        }
    }
    if(token[0] == '_' || (token[0]>='A' && token[0] <='Z') || (token[0]>='a' && token[0]<='z')){
        string t = "id";
```

```
        return t;
    }
    for(int i=0;i<token.size();i++){
        if(!isdigit(token.at(i))){
            string t = "num";
            break;
        }
    }
    string st = "error";
    return st;
}
int main(){
    // Lexical Analysis
    ofstream fout;
    ifstream fin;
    string text,txt,remove;
    vector <string> inp;
    vector <string> inp1;
    int tokens=0;
    fout.open("parser.txt", ios::trunc | ios::out );
    if(!fout) cout<<"File creation failed";
    fin.open("code.txt", ios::in);
    int flag = 1;
    while(!fin.eof()){
        fin>>text;
        if((text[0]=='/' && text[1]=='*') || regex_match(text,regex("(//)(.*)"))){
            if(text[1]=='*') getline(fin,remove,'/');
            else getline(fin,remove);
            continue;
        }
        if(text=="") continue;         //empty lines
```

```
    txt = category(text);
    if(txt.compare("error")==0){
        flag=0;
    }
    else{
        inp.push_back(txt);
        inp1.push_back(text);
    }
    fout<<text<<" ";
    tokens++;
    text="";
}
fin.close();
fout.close();
//int flag1 = 1;
if(flag!=0){
     // Syntax Analysis
vector <string> input;
for(int i=0;i<inp.size();i++){
    if(inp[i].compare(";")==0){
        input.push_back(inp[i]);
        int  error = parse(input);
        if(error==1){
            cout<<"Error!!!"<<endl;
            cout<<"The string could not be parsed successfully,"<<endl;
            flag = 0;
            break;
        }
        /*
        for(int i=0;i<input.size();i++){
            cout<<input[i];
```

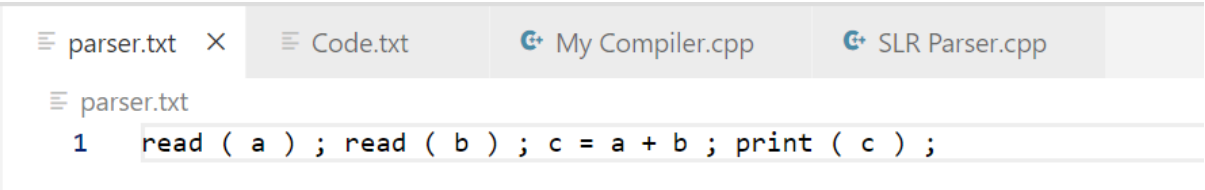
```
        }
        cout<<endl;*/
        input.clear();
    }
    else{
        input.push_back(inp[i]);
    }
}
if(flag==1){
    if(inp[inp.size()-1].compare(";")!=0){
        flag=0;
    }
}
}
//cout<<flag1<<endl;
if(flag!=0){
    // Intermediate Code Generator
vector <string> input1;
vector <string> code;
for(int i=0;i<inp1.size();i++){
    if(inp1[i].compare(";")==0){
        if(input1[0].compare("print")!=0 && input1[0].compare("read")!=0){
            if(input1.size()==3){
                code.push_back(input1[0]+" "+input1[1]+" "+input1[2]);
            }
            else{
                /*for(int j=0;j<input1.size();j++){
                        cout<<input1[j]<<" ";
                }
                cout<<endl;*/
                TACGenerate(input1,code);
```

```
            }
        }
        input1.clear();
    }
    else{
        input1.push_back(inp1[i]);
    }
}
cout<<"The intermediate code is: "<<endl;
for(int i=0;i<code.size();i++){
    cout<<code[i]<<endl;
}
cout<<endl;
}
if(flag==0){
    cout<<"Error!!!"<<endl;
    cout<<"Please Correct it!!!"<<endl;
}
```

Data:

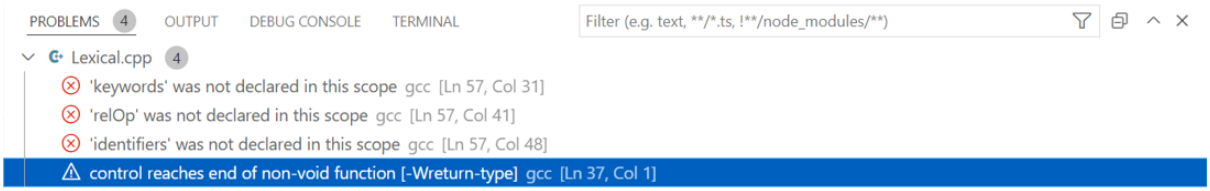
Output:

After Lexical Analyser:



After Syntax Analyser and Intermediate Code Generation:

Errors:



Results and Conclusion: Mini compiler has been constructed successfully.