COMPILER LAB REPORT

Assignment V Project 9

Class:UG-III Section: A1

Shaswata Saha (10)

Radib Kar (11)

Anuran Chakraborty (20)

Souvik Saha (27)

1. Question

Consider a simple PASCAL-like language with the following structure:

program {name of the program}
uses {comma delimited names of libraries you use}
const {global constant declaration block}
var {global variable declaration block}

function {function declarations, if any}
{local variables}
begin
...
end

begin {main program block starts}
...
end. {end of main program block}

Type declaration can be done as:
type-identifier-1, type-identifier-2 = type-specifier
Data Types: integer and real
Input , output statements are in the form **get x** and **put x**
Conditional statement of the form **expression?expression:expression** is supported
Relational operators are supported **{>,<,>=,<=}**
Arithmetic operators supported are **{+,-,*}**

Part I – Construct a CFG for this language.
Part II – Write lexical analyzer to scan the stream of characters from a program written in the above language and generate stream of tokens.
Part III – Write a top-down parser for this language.

2. Context Free Grammar

```
start -> program id rest1
rest1 -> uses liblist rest2 | rest2
liblist -> id liblist'
liblist' -> , id liblist' | ε
rest2 -> const const_list rest3
const_list -> id=num const_list'
const_list' -> , id=num const_list' | ε
rest3 -> var varlist rest4
varlist -> liblist=type;varlist'
varlist' -> liblist=type;liblist' | ε
type -> integer | real
rest4 -> function id varlist rest_function rest4 | rest_main
rest_function -> begin statements end ;
rest_main -> begin statements end .
statements -> get id; statements | put id ; statements|
id=something ; statements | ε
something -> term exp' s3
s3 -> ε | ? exp : exp
exp -> term exp'
exp' -> op term exp' | ε
term -> ID | num
op -> + | - | * | < | > | z | y
ID -> id
U -> ;
```

3. Code

```
#include<bits/stdc++.h>
#include <unistd.h>
using namespace std;

map<char, set<char> > firstSet;
map<char, set<char> > followSet;
map< pair<char,char>, string > table;
int noofProd;
string* production;
vector<string> prod;
set<char> t,nt;

int conflict;

map<char,string> symbols;

// Function to populate the symbol mapping
void populateSym()
{
        fstream file2;
        string str;

        file2.open("mapping.txt", ios::in);
        while(getline(file2,str))
        {
                // string first="";
                // first+=str[0];
                symbols.insert(make_pair(str[0],str.substr(2)));
        }
        file2.close();
        // map<string,string>::iterator it;
        // for(it=symbols.begin();it!=symbols.end();it++)
        //      cout<<it->first<<"\t\t"<<it->second<<endl;
}

// Function to print a production
void printProd(string prod)
{
        int i;
        // cout<<prod<<"\t\t\t\t";
        if(prod=="pop" || prod=="scan")
        {
                cout<<prod;
                return;
        }

        string actual="";
        for(i=0;i<prod.length();i++)
        {
                string pr="";
                pr+=prod[i];
                if(i==1)
                        actual+=" -> ";
                else if(symbols.find(prod[i])==symbols.end())// Trivial
characters
                        actual+=pr+" ";
                else
                        actual+=symbols[prod[i]]+" ";
        }
        cout<<actual;
}

// Function to remove left recursion
void removeLeftRecur()
{
        int i,j;
        for(i=0;i<noofProd;i++)
        {
                int nextind=i;
                // If this produciton has a left recursion
                if(production[i][0]==production[i][2])
                {
                        // Then try removing it
                        string newprod="";
                        newprod+=production[i][0];
                        newprod+='\'';
                        // For every produciton having X in the 2 index
                        for(j=0;j<noofProd;j++)
                        {
                                if(production[j][0]==production[i][0] &&
production[j][0]!=production[j][2])
                                {
                                        // Remove the first part
                                        prod.push_back(production[j]+newprod);
                                        nextind=j;
                                }
                                else if(production[j][0]==production[i][0] &&
production[j][0]==production[j][2])
                                {
                                        string nstr=production[j].substr(3);
                                        string nstr2=newprod+'=';
                                        prod.push_back(nstr2+nstr+newprod);
                                        nextind=j;
                                }
                        }
                        // Push epsilon
                        string ns=newprod+'='+'#';
                        prod.push_back(ns);
                }
                else
                        prod.push_back(production[i]);
                i=nextind;
        }
}

// Function to calculate first for a symbol
void first(char c, int rule_no)
{
        int j,k;

        // Case for terminal
        if(!isupper(c))
        {
                firstSet[c].insert(c);
        }
        // For all the productions
        for(j=rule_no;j<noofProd;j++)
        {
                if(production[j][0]==c)// If the production has c on LHS then
only calculate
                {
                        if(production[j][2]=='#') // If production is epsilon
then recur for the next symbol
                        {
                                firstSet[c].insert('#');
                        }
                        else if(!isupper(production[j][2])) // If start symbol
is a terminal the first is the start symbol
                        {
                                firstSet[c].insert(production[j][2]);
                        }
                        else // If it is a non-terminal then first calculate
its firstset
                        {
                                for(k=2;k<production[j].length();k++)
                                {
                                        // If it is a terminal simply add the
terminal
                                        if(!isupper(production[j][k]))
                                        {

        firstSet[c].insert(production[j][k]);
                                              break;
                                        }
                                        else
                                        {
                                              if(production[j][k]!=c)
                                              {
                                                      // If it is a
nonterminal calculate its first

        first(production[j][k],0);
                                                      // Add the first set to
it

        firstSet[c].insert(firstSet[production[j][k]].begin(),firstSet[product
ion[j][k]].end());
                                                      // If epsilon not in
this then break

        if(firstSet[production[j][k]].find('#')==firstSet[production[j][k]].en
d())
                                                              break;
                                                      else
                                                              // remove #

        firstSet[c].erase('#');
                                              }
                                              else
                                              {
                                                      // Check if present
symbol first has epsilon

        first(production[j][k],j+1);

        if(firstSet[production[j][k]].find('#')==firstSet[production[j][k]].en
d())
                                                              break;
                                              }
```

```
                                        }
                                }
                                // If last contains # add #
                                if(k==production[j].length())
                                        firstSet[c].insert('#');
                        }
                }
        }

}

// Function to calculate follow
void follow(char c)
{
        int i,j,k;
        // First add $ to follow set of start symbol
        if(production[0][0]==c)
                followSet[c].insert('$');
        // For every production
        for(i=0;i<noofProd;i++)
        {
                // Now traverse every production
                for(j=2;j<production[i].length();j++)
                {
                        if(production[i][j]==c) // If c found on RHS
                        {
                                if(j!=(production[i].length()-1))// It is not
the ending character
                                {
                                        // Insert the first of next non
terminal

        followSet[c].insert(firstSet[production[i][j+1]].begin(),firstSet[prod
uction[i][j+1]].end());
                                        for(k=j+1;k<production[i].length();)
                                        {

        if(firstSet[production[i][k]].find('#')==firstSet[production[i][k]].en
d())// If epsilon does not exist then break
                                                        break;
                                                k++;
                                                if(k==production[i].length())
                                                        break;
```

```
       followSet[c].insert(firstSet[production[i][k]].begin(),firstSet[produc
tion[i][k]].end());
                                   }
                                   // If even the last symbol has epsilon
in its first then compute follow of LHS
                                   if(k==production[i].length())
                                   {
                                          if(c!=production[i][0])
                             {
                                // Calculate the follow of the Non-
Terminal
                                // in the L.H.S. of the production
                                follow(production[i][0]);
                                // Insert into set

followSet[c].insert(followSet[production[i][0]].begin(),followSet[production[
i][0]].end());
                             }
                                   }
                            }
                            else
                            // For ending character add follow of LHS
                            if(j==(production[i].length()-1) &&
c!=production[i][0])
               {
                  // Calculate the follow of the Non-Terminal
                  // in the L.H.S. of the production
                  follow(production[i][0]);
                  // Insert into set

followSet[c].insert(followSet[production[i][0]].begin(),followSet[production[
i][0]].end());
               }

                     }

              }
       }
}

void fill_t_nt(){
       vector<string>::iterator i;
       string s;
       for(i=prod.begin();i!=prod.end();i++){
```

```
                nt.insert((*i)[0]);
        }

        t.insert('#');
        t.insert('$');

        for(i=prod.begin();i!=prod.end();i++){
                s=(*i);
                for(int j=2;j<s.length();j++){
                        if(nt.find(s[j])==nt.end()){
                                t.insert(s[j]);
                        }
                }
        }
}

void make_table(){

        vector<string>::iterator prodit;
        string rule;

        conflict=0;

        for(prodit = prod.begin(); prodit!= prod.end(); prodit++){
                rule = *(prodit);
                int i;
                for(i=2;i<rule.length();i++){
                        set<char> first = firstSet[rule[i]];
                        set<char>::iterator it;
                        int flag = 1; //check whether current character in rhs
of rule has epsilon

                        for(it = first.begin(); it!=first.end(); it++){
                                if((*it)=='#'){
                                        flag = 0;
                                }
                                else{

        if(table.find(make_pair(rule[0],(*it)))!=table.end() &&
table[make_pair(rule[0],(*it))]!=rule){
                                                cout<<"Error 1 at
"<<rule[0]<<","<<(*it)<<","<<rule<<","<<table[make_pair(rule[0],*it)]<<endl;
                                                conflict=1;
                                                return;
                                        }
                                        table[make_pair(rule[0],(*it))] = rule;
                                }
                        }
                        if(flag){   //if epsilon is not present, this rule is
not needed any more.
                                break;
                        }
                }

                if(i == rule.length()){  //the entire rhs has epsilon in
first. so followSet of lhs is used.
                        set<char> fol = followSet[rule[0]];
                        set<char>::iterator it;

                        for(it = fol.begin(); it!=fol.end(); it++){

        if(table.find(make_pair(rule[0],(*it)))!=table.end() &&
table[make_pair(rule[0],(*it))]!=rule){
                                                cout<<"Error 2 at
"<<rule[0]<<","<<(*it)<<","<<rule<<","<<table[make_pair(rule[0],*it)]<<endl;
                                                conflict=1;
                                                return;
                                }
                                table[make_pair(rule[0],(*it))] = rule;
                        }
                }
        }

        set<char>::iterator itt,itnt;
        // cout<<"Non-terminal\tTerminal\tRule\n";
        // for(itnt = nt.begin(); itnt!=nt.end();itnt++){  //non terminal loop
        //      for(itt = t.begin(); itt!=t.end();itt++){   //terminal loop
        //
        cout<<(*itnt)<<"\t"<<(*itt)<<"\t"<<table[make_pair(*itnt,*itt)]<<"\n";
        //      }
        //      cout<<endl;
        // }

        //set<char>::iterator itt, itnt;
        for(itnt = nt.begin(); itnt!=nt.end();itnt++){  //non terminal loop
                for(itt = t.begin(); itt!=t.end();itt++){   //terminal loop
                        if(table.count(make_pair(*itnt,*itt))==0){
                        //if(table.find( make_pair( (*itnt),(*itt) ) ) ==
table.end()){
```

```
                                if((*itt)=='$' ||
followSet[(*itnt)].find((*itt))!=followSet[(*itnt)].end()){
                                        table[make_pair((*itnt),(*itt))] =
"pop";
                                }
                                else
if(firstSet[(*itnt)].find((*itt))==firstSet[(*itnt)].end() &&
followSet[(*itnt)].find((*itt))==followSet[(*itnt)].end()){
                                        table[make_pair((*itnt),(*itt))] =
"scan";
                                }
                        }
                }
        }

}


int main(int argc, char const *argv[])
{
        int i,j;
        printf("Enter number of productions\n");
        cin>>noofProd;
        printf("Enter the productions individually\n");

        populateSym();

        production=new string[noofProd];
        for(i=0;i<noofProd;i++)
                cin>>production[i];

        removeLeftRecur();
        fill_t_nt();
        // for(i=0;i<prod.size();i++)
        //      cout<<prod[i]<<endl;

        // Insert first of terminals
        for(i=0;i<noofProd;i++)
                for(j=0;j<production[i].length();j++)
                        if(!isupper(production[i][j]))// Terminal

        firstSet[production[i][j]].insert(production[i][j]);

        for(i=0;i<noofProd;i++)
                first(production[i][0],0);
        map<char, set<char> >::iterator it;
        set<char>::iterator its;

        cout<<"printing terminals"<<endl;
        for(its=t.begin();its!=t.end();its++)
        {
                if(symbols.find(*its)==symbols.end())
                        cout<<*its<<endl;
                else
                        cout<<symbols[*its]<<endl;
        }

        cout<<"printing non terminals"<<endl;
        for(its=nt.begin();its!=nt.end();its++)
        {
                if(symbols.find(*its)==symbols.end())
                        cout<<*its<<endl;
                else
                        cout<<symbols[*its]<<endl;
        }

        // Printing first set
        for(it=firstSet.begin();it!=firstSet.end();it++)
        {
                cout<<"first(";
                if(symbols.find(it->first)==symbols.end())
                        cout<<it->first<<"  ";
                else
                        cout<<symbols[it->first];
                cout<<") : {";

                for(its=it->second.begin();its!=it->second.end();its++)
                {
                        if(symbols.find(*its)==symbols.end())
                                cout<<*its<<"  ";
                        else
                                cout<<symbols[*its]<<"  ";
                }
                cout<<"}\n";
        }
        cout<<"====================================\n";

        for(i=0;i<noofProd;i++)
                follow(production[i][0]);
        // Printing follow set
        for(it=followSet.begin();it!=followSet.end();it++)
        {
                cout<<"follow(";
                if(symbols.find(it->first)==symbols.end())
                        cout<<it->first;
                else
                        cout<<symbols[it->first];
                cout<<") : {";

                it->second.erase('#');
                for(its=it->second.begin();its!=it->second.end();its++)
                {
                        if(symbols.find(*its)==symbols.end())
                                cout<<*its<<"  ";
                        else
                                cout<<symbols[*its]<<"  ";
                }
                cout<<"}\n";
        }

        cout<<"=====================================\n";

        make_table();

        if(conflict==0)
        {

                cout<<"Table making Done\nPrinting table\n\n";

                set<char>::iterator itt,itnt;
                cout<<"Non-terminal\tTerminal\tRule\n";
                for(itnt = nt.begin(); itnt!=nt.end();itnt++){  //non terminal
loop
                        for(itt = t.begin(); itt!=t.end();itt++)
                        {   //terminal loop

                                if(symbols.find(*itnt)==symbols.end())
                                        cout<<*itnt<<"\t";
                                else
                                        cout<<symbols[*itnt]<<"\t";

                                if(symbols.find(*itt)==symbols.end())
                                        cout<<*itt<<"\t";
                                else
                                        cout<<symbols[*itt]<<"\t";

                                printProd(table[make_pair(*itnt,*itt)]);

                                cout<<"\n";
                        }
                        cout<<endl;
                }

                // Save parsing table to file
                fstream fout;
                fout.open("parsing_table.txt",ios::trunc | ios::out);
                int total=t.size()*nt.size();

                fout<<total<<endl;

                for(itnt = nt.begin(); itnt!=nt.end();itnt++){  //non terminal
loop
                        for(itt = t.begin(); itt!=t.end();itt++){   //terminal
loop
                                fout<<(*itnt)<<endl;
                                fout<<(*itt)<<endl;
                                fout<<table[make_pair(*itnt,*itt)]<<"\n";
                        }
                }
                fout<<"S"<<endl;
                fout.close();
                cout<<"Parsing table written to file\n";
        }

        return 0;
}
```

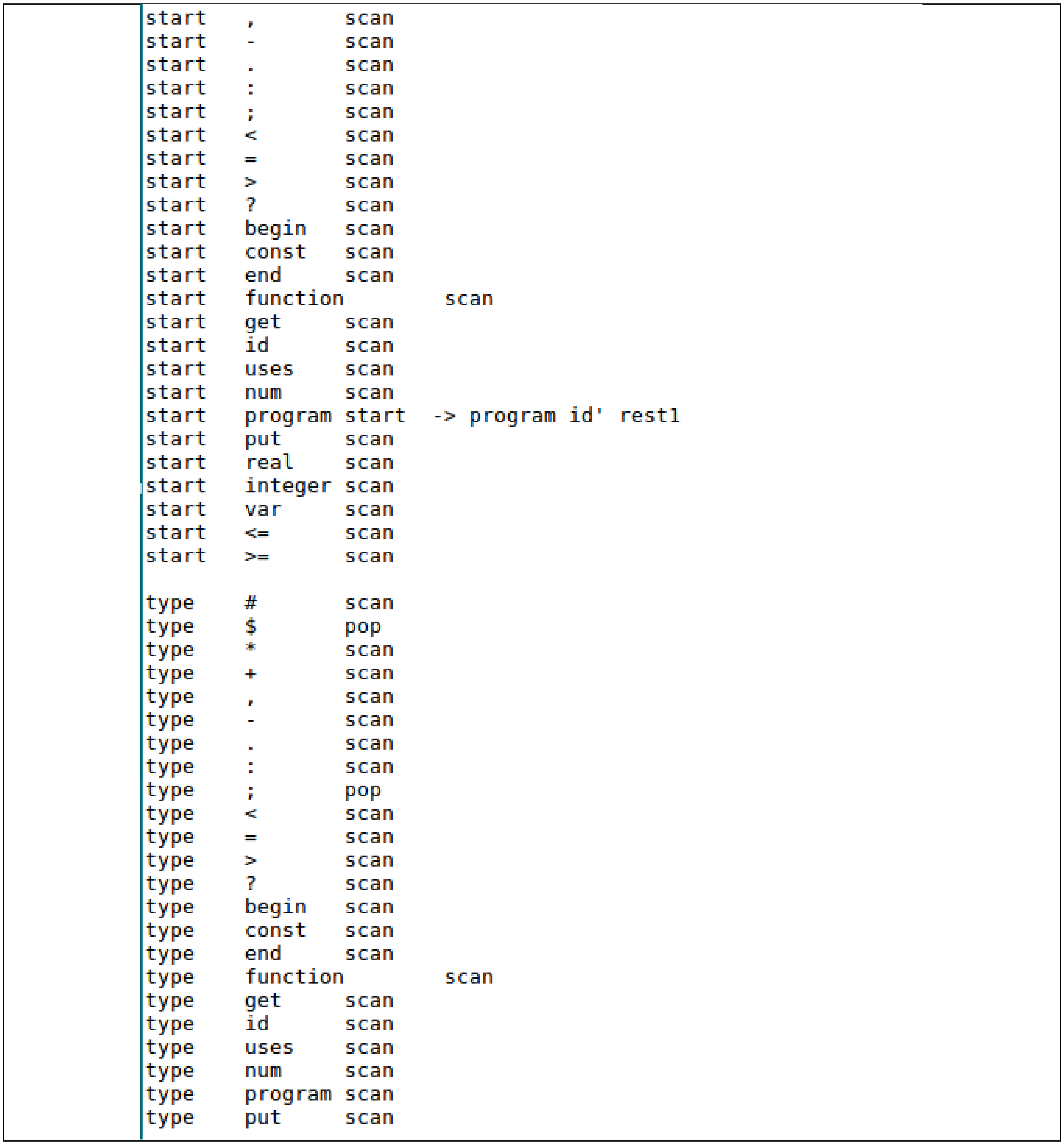
```
#include<bits/stdc++.h>
using namespace std;

map<string,string> symbols;
vector<pair<int,int> >rc;

// Function to populate the symbol mapping
void populateSym()
{
        fstream file2;
        string str;
        file2.open("mapping.txt", ios::in);
        while(getline(file2,str))
        {
                string first="";
                first+=str[0];
                symbols.insert(make_pair(first,str.substr(2)));
        }
        file2.close();
        // map<string,string>::iterator it;
        // for(it=symbols.begin();it!=symbols.end();it++)
        //      cout<<it->first<<"\t\t"<<it->second<<endl;
}

// Function to print a production
void printProd(string prod)
{
        int i;
        // cout<<prod<<"\t\t\t\t";
        if(prod=="pop" || prod=="scan")
                return;

        string actual="";
        for(i=0;i<prod.length();i++)
        {
                string pr="";
                pr+=prod[i];
                if(i==1)
                        actual+=" -> ";
                else if(symbols.find(pr)==symbols.end())// Trivial characters
                        actual+=pr+" ";
                else
                        actual+=symbols[pr]+" ";
        }
        cout<<actual;
}

void print(vector<pair<string,string> > v,vector<pair<int,int> >rcl){

        for(int i=0;i<v.size();i++){
                cout<<v[i].first<<"\t\t";
                if(symbols.find(v[i].second)==symbols.end())
                                cout<<v[i].second<<"\t\t";
                        else
                                cout<<symbols[v[i].second]<<"\t\t";
                cout<<rcl[i].first<<"\t\t"<<rcl[i].second<<endl;
        }
}
void printvector(vector<string> a){
        for(int i=0;i<a.size();i++){
                cout<<a[i]<<endl;
        }
}
vector<string> my(vector<string> v,vector<string> & vars,vector<pair<int,int>
>rc){
        vector<string> mylist;
        //vector<string> vars;
        vector<pair<string,string> > mp;
        for(int i=0;i<v.size();i++){
                if(v[i]=="program"){
                        mylist.push_back("p");
                        mp.push_back(make_pair(v[i],"p"));
                }
                else if(v[i]=="uses"){
                        mylist.push_back("l");
                        mp.push_back(make_pair(v[i],"l"));
                }
                else if(v[i]=="real"){
                        mylist.push_back("r");
                        mp.push_back(make_pair(v[i],"r"));
                }
                else if(v[i]=="integer"){
                        mylist.push_back("u");
                        mp.push_back(make_pair(v[i],"u"));
                }
                else if(v[i]=="var"){
                        mylist.push_back("v");
                        mp.push_back(make_pair(v[i],"v"));
                }

                else if(v[i]=="function"){
                        mylist.push_back("f");
                        mp.push_back(make_pair(v[i],"f"));
                }
                else if(v[i]=="begin"){
                        mylist.push_back("b");
                        mp.push_back(make_pair(v[i],"b"));
                }
                else if(v[i]=="end"){
                        mylist.push_back("e");
```

```
            mp.push_back(make_pair(v[i],"e"));
        }
        else if(v[i]=="get"){
            mylist.push_back("g");
            mp.push_back(make_pair(v[i],"g"));
        }
        else if(v[i]=="put"){
            mylist.push_back("q");
            mp.push_back(make_pair(v[i],"q"));
        }
        else if(v[i][0]=='('){
            mylist.push_back("x");
            mp.push_back(make_pair(v[i],"x"));
        }
        else if(v[i]=="const"){
            mylist.push_back("c");
            mp.push_back(make_pair(v[i],"c"));
        }
        else if(v[i]=="?" || v[i]==":"){
            mylist.push_back(v[i]);
            mp.push_back(make_pair(v[i],v[i]));
        }
        else if(v[i]=="." || v[i]==";" || v[i]==","){
            mylist.push_back(v[i]);
            mp.push_back(make_pair(v[i],v[i]));
        }
        else if(v[i]=="==" || v[i]=="=" || v[i]=="<" || v[i]==">" ||
v[i]=="<=" || v[i]==">="){
            if(v[i]=="<="){
                mylist.push_back("y");
                mp.push_back(make_pair(v[i],"y"));
            }
            else if(v[i]==">="){
                mylist.push_back("z");
                mp.push_back(make_pair(v[i],"z"));
            }
            else{
                mylist.push_back(v[i]);
                mp.push_back(make_pair(v[i],v[i]));
            }
        }
        else if(v[i]=="+" || v[i]=="-" || v[i]=="*"){
            mylist.push_back(v[i]);
            mp.push_back(make_pair(v[i],v[i]));
        }
```

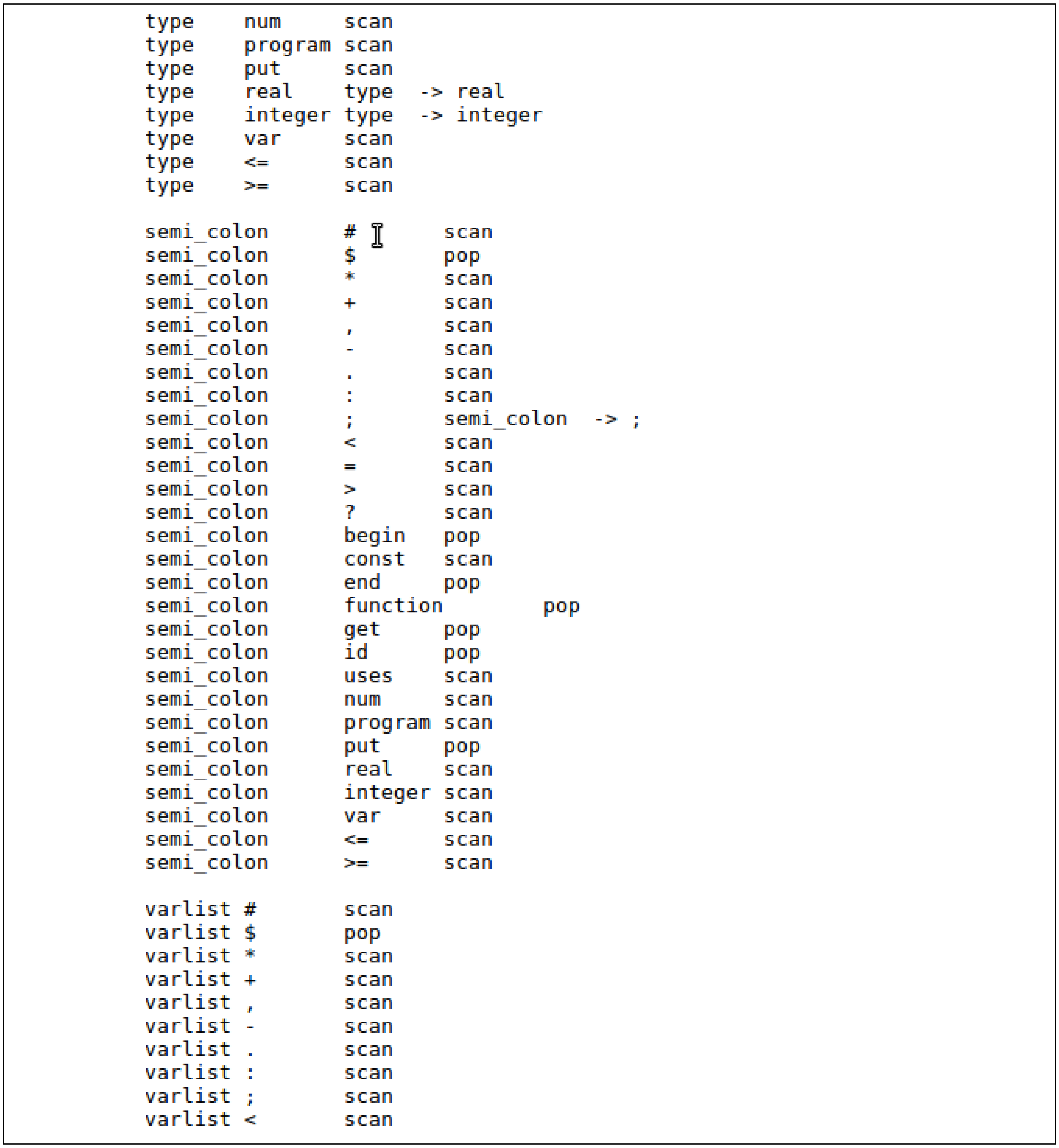
```
                else if((v[i][0]>='0' && v[i][0]<='9') || (v[i][0]=='-' &&
v[i][1]>='0' && v[i][1]<='9')){
                        mylist.push_back("n");
                        mp.push_back(make_pair(v[i],"n"));
                }
                else{
                        if(v[i]!=""){
                                mylist.push_back("i");
                                mp.push_back(make_pair(v[i],"i"));
                                vars.push_back(v[i]);
                        }

                }


        }
        //cout<<"tokens\t converted\n";

        cout<<"token\tconverted token\trow\tcolumn\n";

        print(mp,rc);
        return mylist;


}

bool isdelim(char c){
        if(c==',' || c==';' || c==' ' || c=='\t' || c=='\n' || c=='?' ||
c==':' || c=='=' || c=='>' || c=='<' || c=='+' || c=='*')
        return true;
        return false;
}
vector<string> extract(vector<string> s, vector<pair<int,int> >&rc){

        vector<string> store;

        for(int j=0;j<s.size();j++){
                string p=s[j];

                string temp="";
                int tab=0;
                int tag=0;

                for(int i=0;i<p.length();i++){
                        //cout<<p[i]<<" ";
                        int store_i=i;
                        if(tag){
                                store_i=tab+i;
                        }
                        if(isdelim(p[i])){
                                if(temp.size()!=0){
                                        store.push_back(temp);
                                        int len123=temp.size();
                                        rc.push_back(make_pair(j+1,store_i+1-
len123));
                        }
                                //cout<<temp<<endl;
                                //while(i<p.length() && p[i]==' ')
                                //      i++;
                                if(p[i]==',' || p[i]==';' || p[i]=='?' ||
p[i]==':' || p[i]=='.' || p[i]=='=' || p[i]=='+' || p[i]=='-' || p[i]=='*'){
                                        if(p[i]=='.'){
                                                if((p[i-1]>='0' && p[i-1]<='9')
&& (p[i+1]>='0' && p[i+1]<='9'))

        temp=temp+string(1,p[i]);
                                                        if(i==p.length()-1){
                                                        store.push_back(temp);
                                                        int len123=temp.size();

        rc.push_back(make_pair(j+1,store_i+2-len123));
                                                }
                                                else{

        store.push_back(string(1,p[i]));

        rc.push_back(make_pair(j+1,store_i+1));
                                                }
                                        }
                                        else if(p[i]=='-'){
                                                if((p[i-1]=='=' || p[i-1]=='<'
|| p[i-1]=='>') && (p[i+1]>='0' && p[i+1]<='9'))

        temp=temp+string(1,p[i]);
                                                        if(i==p.length()-1){
                                                        store.push_back(temp);
                                                        int len123=temp.size();

        rc.push_back(make_pair(j+1,store_i+2-len123));
                                                }
                                                else{

        store.push_back(string(1,p[i]));

        rc.push_back(make_pair(j+1,store_i+1));
                                                }
                                        }
                                        else{
                                                store.push_back(string(1,p[i]));

        rc.push_back(make_pair(j+1,store_i+1));
                                        }
                                }
                                else if(p[i]=='\t'){
                                        tab+=3;
                                        tag=1;
                                }
                                else if(p[i]=='<' && p[i+1]=='='){
                                        store.push_back("<=");
                                        rc.push_back(make_pair(j+1,store_i+1));
                                        i++;
                                }
                                else if(p[i]=='>' && p[i+1]=='='){
                                        store.push_back(">=");
                                        rc.push_back(make_pair(j+1,store_i+1));
                                        i++;

                                }
                                else if(p[i]=='>'){
                                        store.push_back(">");
                                        rc.push_back(make_pair(j+1,store_i+1));
                                }

                                else if(p[i]=='<'){
                                        store.push_back("<");
                                        rc.push_back(make_pair(j+1,store_i+1));
                                }

                                temp="";
                        }

                        else{

                                if(p[i]=='('){
                                        rc.push_back(make_pair(j+1,store_i+1));
                                        string w="";
                                        while(p[i]!=')'){
                                                w=w+string(1,p[i]);
                                                i++;
                                        }
                                        w=w+string(1,p[i]);
                                        store.push_back(w);


                                }
                                else{
                                        temp=temp+string(1,p[i]);
                                        if(i==p.length()-1){
                                                store.push_back(temp);
                                                int len123=temp.size();

        rc.push_back(make_pair(j+1,store_i+2-len123));

        //rc.push_back(make_pair(j+1,i+1));
                                                //cout<<temp<<endl;
                                        }
                                }
                        }

                }
        }

        return store;
}


// Function to print stack
void printStack(stack<char> st)
{
        stack<char> temp;
        string stack="";
        while(!st.empty())
        {
                temp.push(st.top());
                st.pop();
        }
        while(!temp.empty())
        {
                st.push(temp.top());
                stack+=temp.top();
                temp.pop();
        }
        cout<<"Stack: "<<stack<<endl;
}

// Function to parse a string
void parse(map< pair<char, char>, string > table, vector<string> expr, char
startSym)
{
        // Create the stack and push $
        stack<char> st;
        st.push('$');
        // Push start symbol onto stack
        st.push(startSym);

        int i=0,j;
        while(!st.empty() && i<expr.size())
        {
                // First check if appropriate production exists
                pair<char,char> temp;
                char ch=expr[i][0];

                temp=make_pair(st.top(),ch);
                // cout<<temp.first<<", "<<temp.second;

                // Check if there is a match
                if(st.top()==ch)
                {
                        cout<<"Action: match, Popping
"<<st.top()<<"\t\t\t\t\t\t\t\t\t";
                        if(symbols.find(expr[i])==symbols.end())
                                cout<<expr[i]<<"\t\t\t\t\t";
                        else
                                cout<<symbols[expr[i]]<<"\t\t\t\t";
                        printStack(st);

                        i++;
                        st.pop();
                        continue;
                }
                if(table.find(temp)==table.end())
                {
                        cout<<"Parse Error"<<endl;
                        break;
                }
                else
                if(table[temp]=="scan" || table[temp]=="pop")// If valid
production not found then error
                {
                        cout<<"Parse error at:
"<<rc[i].first<<":"<<rc[i].second<<endl;
                        if(table[temp]=="scan")
                        {
                                cout<<"Scan"<<endl;
                                i++;
                                continue;
                        }
                        else if(table[temp]=="pop")
                        {
                                cout<<"Pop"<<endl;
                                if(st.top()=='$')
                                        st.push('S');
                                else
                                        st.pop();
                                printStack(st);
                                continue;
                        }
                }


                // If valid production exists
                string pr=table[temp];
                cout<<"Action: Applying \t\t\t";
                printProd(pr);
                // cout<<pr;
                cout<<"Popping "<<st.top()<<"\t\t";
                // if(symbols.find(expr[i])==symbols.end())
                //              cout<<expr[i]<<"\t\t\t\t\t";
                //      else
                //              cout<<symbols[expr[i]]<<"\t\t\t\t\t";
                cout<<expr[i]<<"\t\t\t\t\t";
                printStack(st);
                st.pop();

                if(pr[2]!='#')
                        // push string onto stack
                        for(j=pr.length()-1;j>=2;j--)
                                st.push(pr[j]);
        }
```

```
}

int main(int argc, char const *argv[])
{
        populateSym();

        fstream file;
        string word, t, q, filename;


        filename = "test.pas";

        file.open(filename.c_str());

        vector<string> store;

        string str;
        while(getline(file,str)){
                store.push_back(str.c_str());
        }
        printvector(store);

        store=extract(store,rc);

        vector<string> tokens;
        vector<string> vars;
        tokens=my(store,vars,rc); //tokens are stored as per converted rules

        // ============= Parsing =======================

        cout<<"Parsing\n";

        // Create parsing table
        int i,j,num;
        char start;
        // cout<<"Enter number of entries in table"<<endl;
        cin>>num;
        cout<<num<<endl;
        map< pair<char, char>, string > parsingTab;
        // Take input
        // cout<<"For every entry first line is the non terminal second
terminal third the production"<<endl;
        for(i=0;i<num;i++)
        {
                char nonter,ter;
                string prod;

                cin>>nonter;
                cin>>ter;
                cin>>prod;

                parsingTab[make_pair(nonter,ter)]=prod;

        }


        cin>>start;

        tokens.push_back("$");

        parse(parsingTab,tokens,start);

        return 0;
}
```

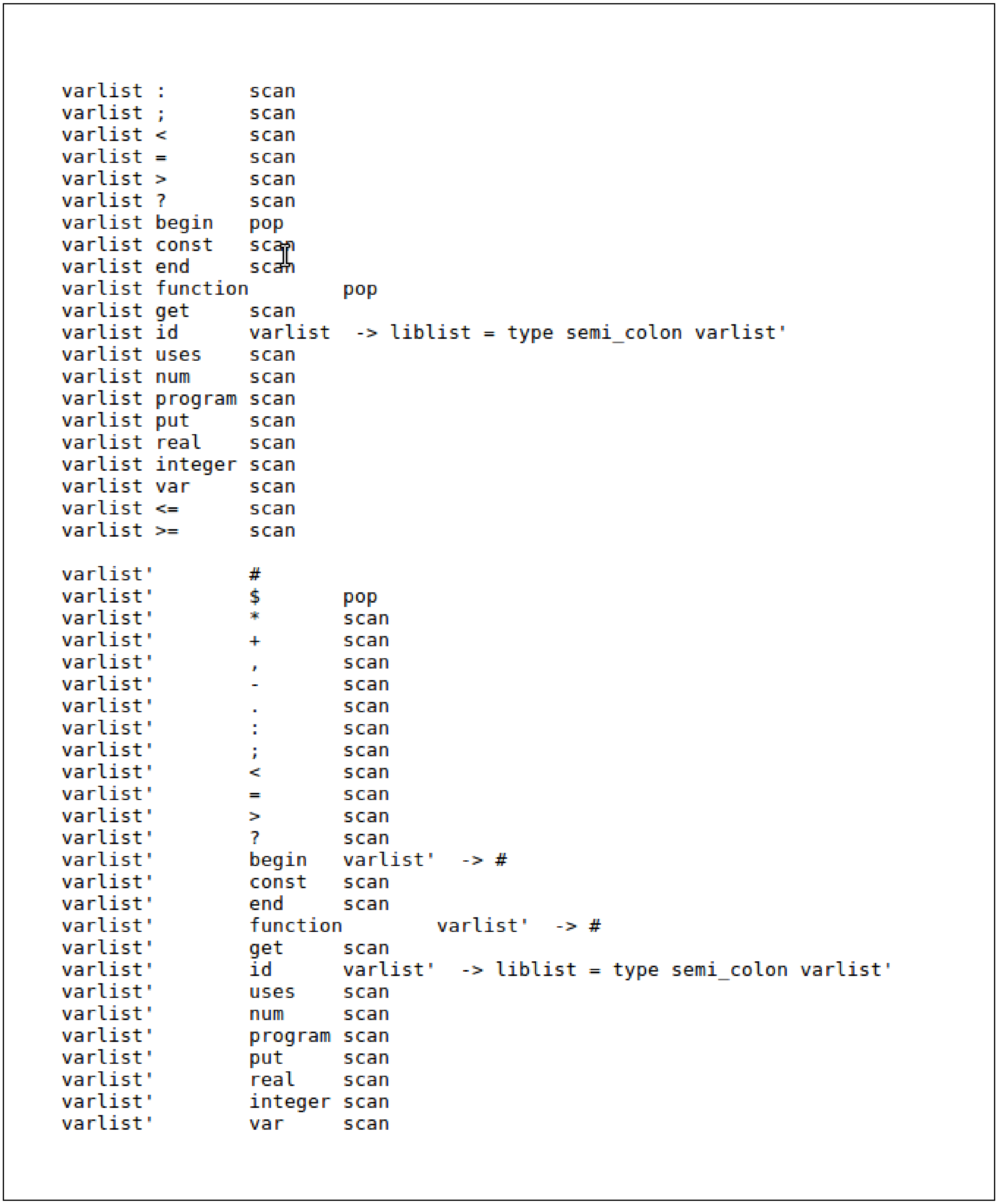
4. Output



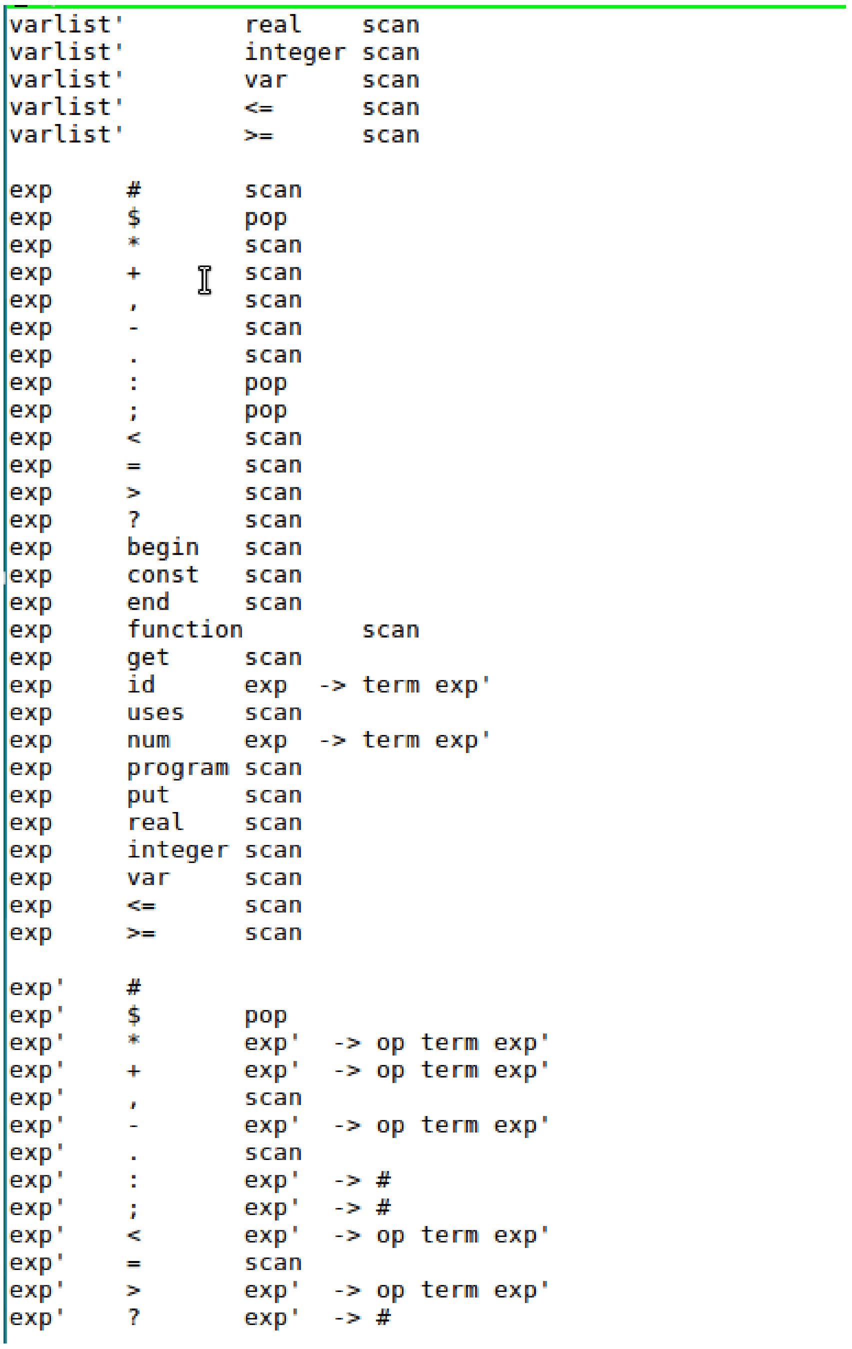
```
start   ,       scan
start   -       scan
start   .       scan
start   :       scan
start   ;       scan
start   <       scan
start   =       scan
start   >       scan
start   ?       scan
start   begin   scan
start   const   scan
start   end     scan
start   function        scan
start   get     scan
start   id      scan
start   uses    scan
start   num     scan
start   program start  -> program id' rest1
start   put     scan
start   real    scan
start   integer scan
start   var     scan
start   <=      scan
start   >=      scan

type    #       scan
type    $       pop
type    *       scan
type    +       scan
type    ,       scan
type    -       scan
type    .       scan
type    :       scan
type    ;       pop
type    <       scan
type    =       scan
type    >       scan
type    ?       scan
type    begin   scan
type    const   scan
type    end     scan
type    function        scan
type    get     scan
type    id      scan
type    uses    scan
type    num     scan
type    program scan
type    put     scan
```



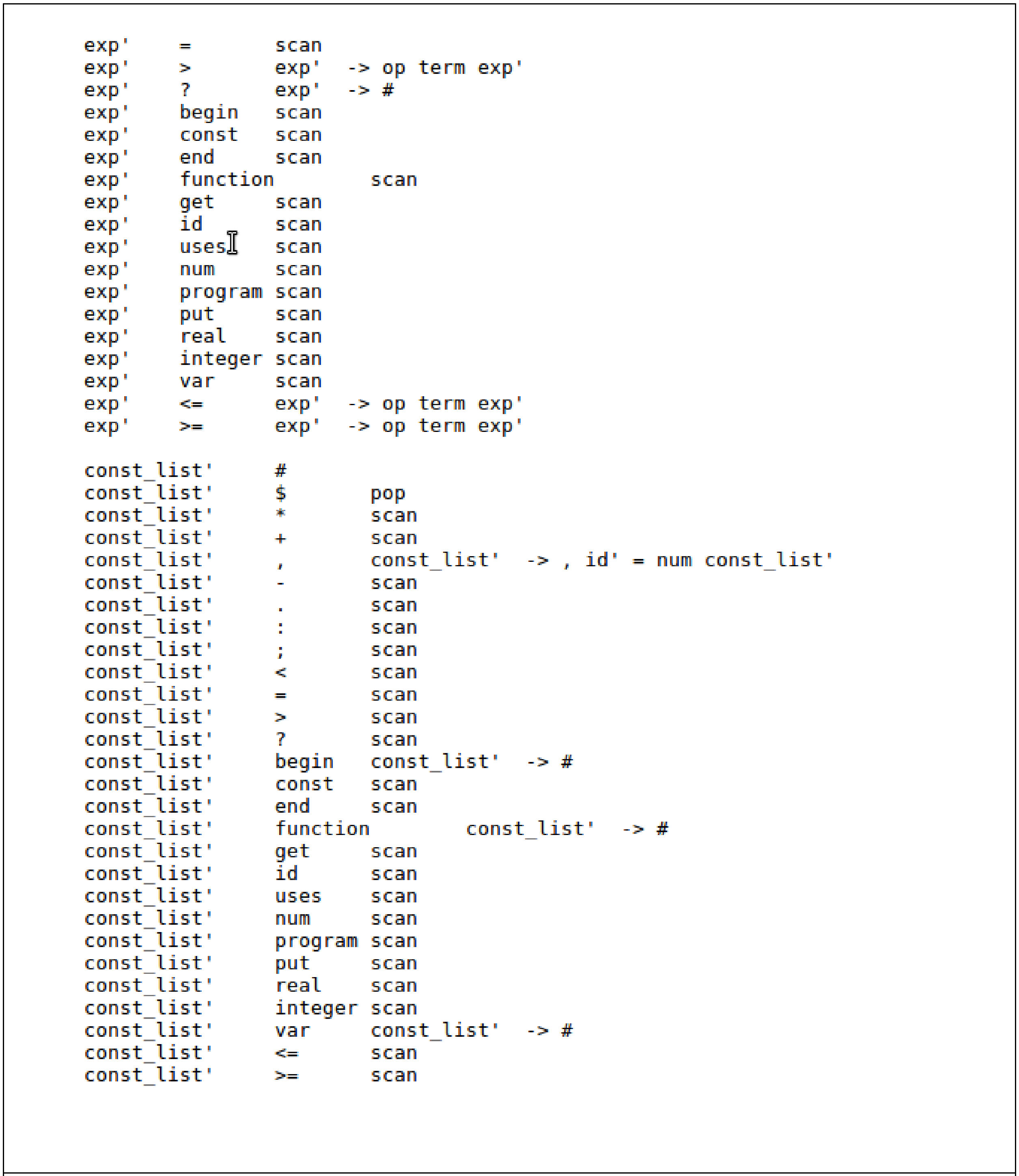
```
type    num     scan
type    program scan
type    put     scan
type    real    type  -> real
type    integer type  -> integer
type    var     scan
type    <=      scan
type    >=      scan

semi_colon      #       scan
semi_colon      $       pop
semi_colon      *       scan
semi_colon      +       scan
semi_colon      ,       scan
semi_colon      -       scan
semi_colon      .       scan
semi_colon      :       scan
semi_colon      ;       semi_colon  -> ;
semi_colon      <       scan
semi_colon      =       scan
semi_colon      >       scan
semi_colon      ?       scan
semi_colon      begin   pop
semi_colon      const   scan
semi_colon      end     pop
semi_colon      function        pop
semi_colon      get     pop
semi_colon      id      pop
semi_colon      uses    scan
semi_colon      num     scan
semi_colon      program scan
semi_colon      put     pop
semi_colon      real    scan
semi_colon      integer scan
semi_colon      var     scan
semi_colon      <=      scan
semi_colon      >=      scan

varlist #       scan
varlist $       pop
varlist *       scan
varlist +       scan
varlist ,       scan
varlist -       scan
varlist .       scan
varlist :       scan
varlist ;       scan
varlist <       scan
```



```
varlist :       scan
varlist ;       scan
varlist <       scan
varlist =       scan
varlist >       scan
varlist ?       scan
varlist begin   pop
varlist const   scan
varlist end     scan
varlist function        pop
varlist get     scan
varlist id      varlist  -> liblist = type semi_colon varlist'
varlist uses    scan
varlist num     scan
varlist program scan
varlist put     scan
varlist real    scan
varlist integer scan
varlist var     scan
varlist <=      scan
varlist >=      scan

varlist'        #
varlist'        $       pop
varlist'        *       scan
varlist'        +       scan
varlist'        ,       scan
varlist'        -       scan
varlist'        .       scan
varlist'        :       scan
varlist'        ;       scan
varlist'        <       scan
varlist'        =       scan
varlist'        >       scan
varlist'        ?       scan
varlist'        begin   varlist'  -> #
varlist'        const   scan
varlist'        end     scan
varlist'        function        varlist'  -> #
varlist'        get     scan
varlist'        id      varlist'  -> liblist = type semi_colon varlist'
varlist'        uses    scan
varlist'        num     scan
varlist'        program scan
varlist'        put     scan
varlist'        real    scan
varlist'        integer scan
varlist'        var     scan
```



```
varlist'        real    scan
varlist'        integer scan
varlist'        var     scan
varlist'        <=      scan
varlist'        >=      scan

exp     #       scan
exp     $       pop
exp     *       scan
exp     +       scan
exp     ,       scan
exp     -       scan
exp     .       scan
exp     :       pop
exp     ;       pop
exp     <       scan
exp     =       scan
exp     >       scan
exp     ?       scan
exp     begin   scan
exp     const   scan
exp     end     scan
exp     function        scan
exp     get     scan
exp     id      exp  -> term exp'
exp     uses    scan
exp     num     exp  -> term exp'
exp     program scan
exp     put     scan
exp     real    scan
exp     integer scan
exp     var     scan
exp     <=      scan
exp     >=      scan

exp'    #
exp'    $       pop
exp'    *       exp'  -> op term exp'
exp'    +       exp'  -> op term exp'
exp'    ,       scan
exp'    -       exp'  -> op term exp'
exp'    .       scan
exp'    :       exp'  -> #
exp'    ;       exp'  -> #
exp'    <       exp'  -> op term exp'
exp'    =       scan
exp'    >       exp'  -> op term exp'
exp'    ?       exp'  -> #
```

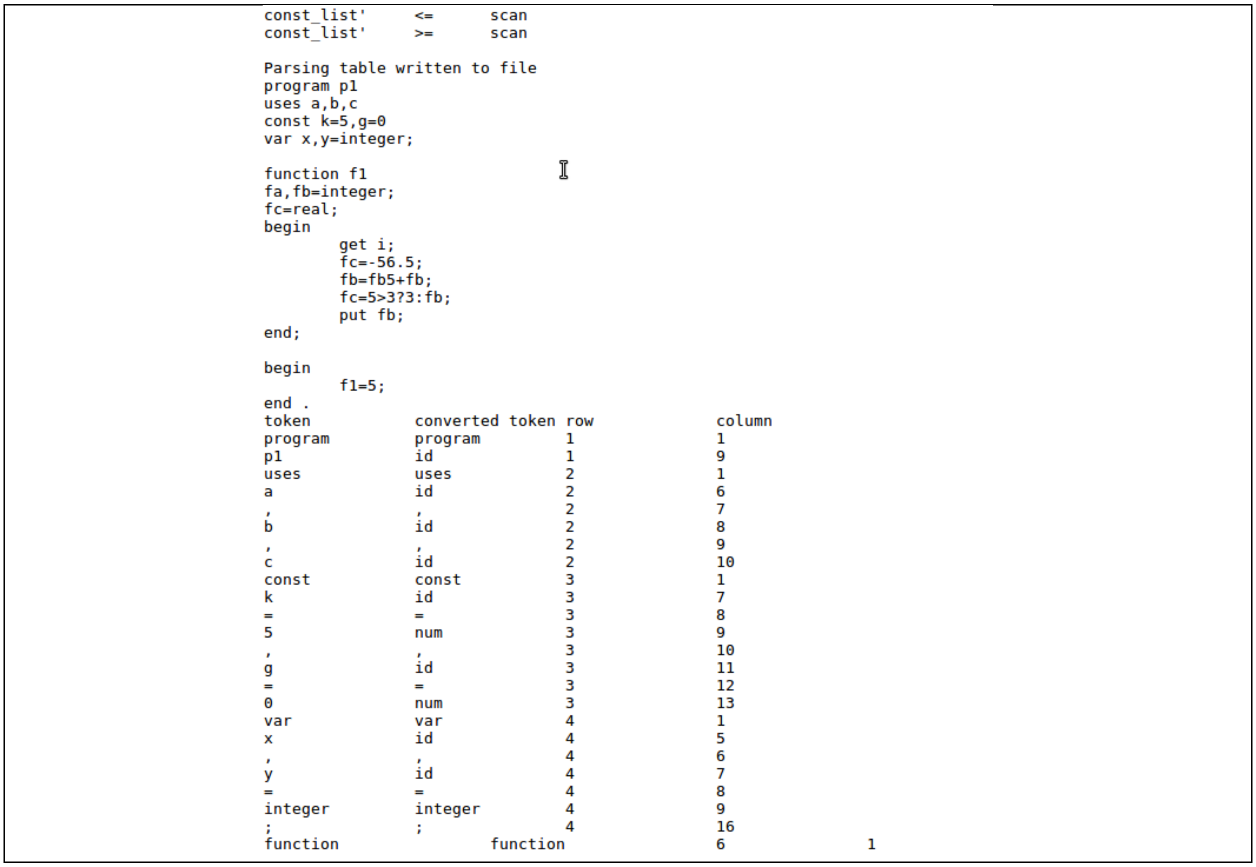


```
exp'    =       scan
exp'    >       exp'  -> op term exp'
exp'    ?       exp'  -> #
exp'    begin   scan
exp'    const   scan
exp'    end     scan
exp'    function        scan
exp'    get     scan
exp'    id      scan
exp'    uses    scan
exp'    num     scan
exp'    program scan
exp'    put     scan
exp'    real    scan
exp'    integer scan
exp'    var     scan
exp'    <=      exp'  -> op term exp'
exp'    >=      exp'  -> op term exp'

const_list'     #
const_list'     $       pop
const_list'     *       scan
const_list'     +       scan
const_list'     ,       const_list'  -> , id' = num const_list'
const_list'     -       scan
const_list'     .       scan
const_list'     :       scan
const_list'     ;       scan
const_list'     <       scan
const_list'     =       scan
const_list'     >       scan
const_list'     ?       scan
const_list'     begin   const_list'  -> #
const_list'     const   scan
const_list'     end     scan
const_list'     function        const_list'  -> #
const_list'     get     scan
const_list'     id      scan
const_list'     uses    scan
const_list'     num     scan
const_list'     program scan
const_list'     put     scan
const_list'     real    scan
const_list'     integer scan
const_list'     var     const_list'  -> #
const_list'     <=      scan
const_list'     >=      scan
```

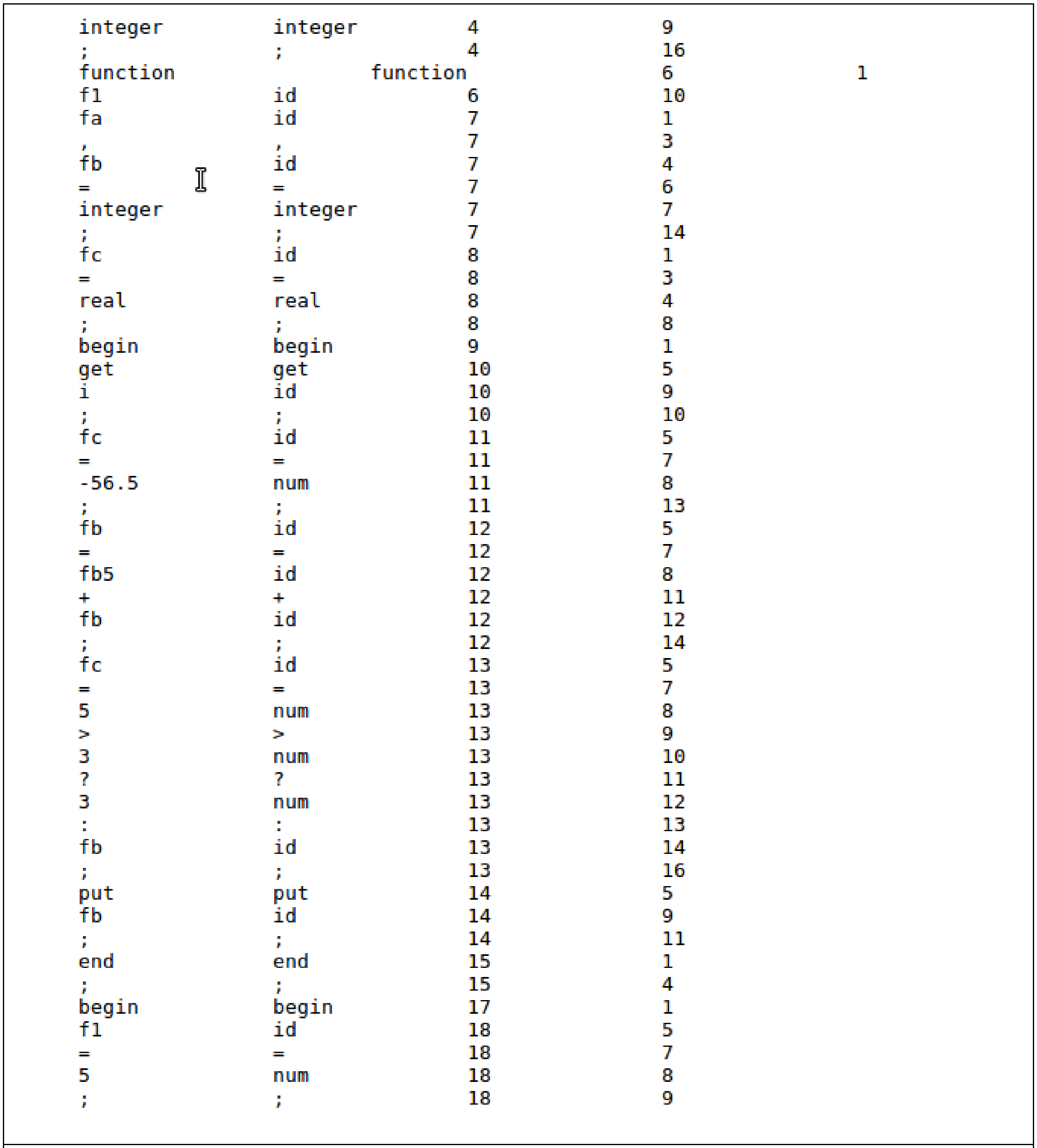
The above figures show the parsing table of the top down parser

Test Input file 1:

```
program p1
uses a,b,c
const k=5,g=0
var x,y=integer;

function f1
fa,fb=integer;
fc=real;
begin
        get i;
        fc=-56.5;
        fb=fb5+fb;
        fc=5>3?3:fb;
        put fb;
end;

begin
        f1=5;
end .
```

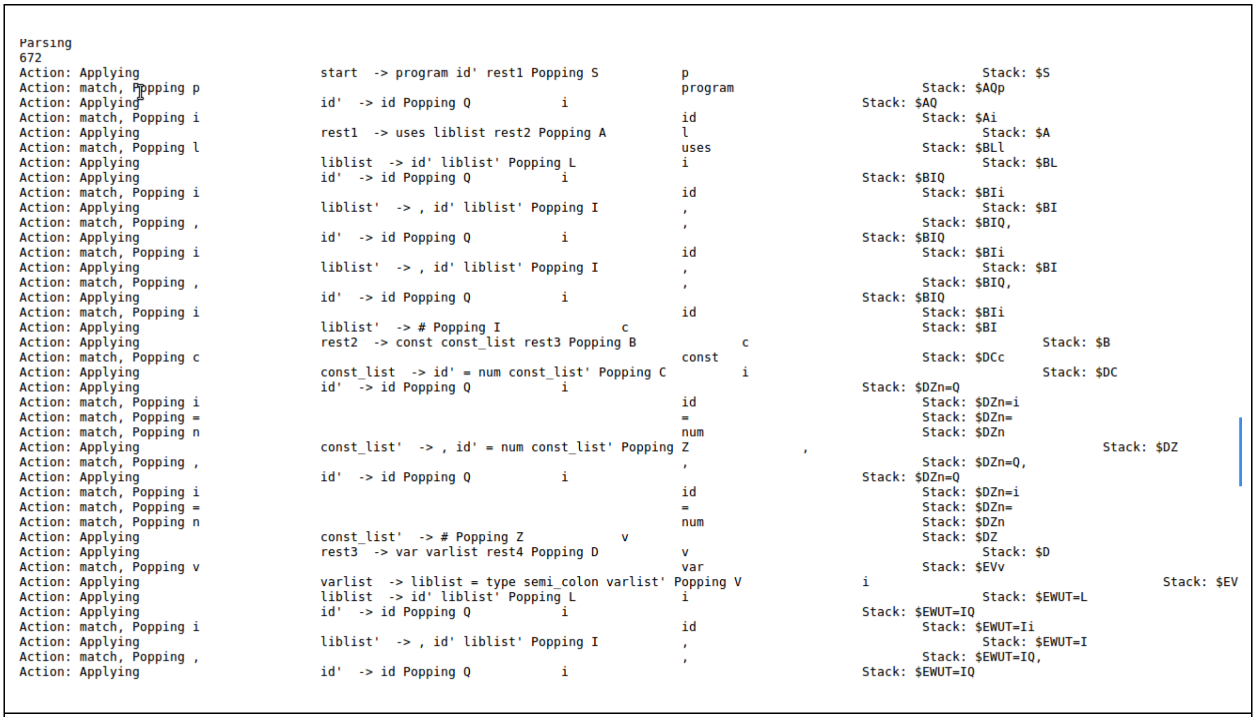


```
const_list'     <=      scan
const_list'     >=      scan

Parsing table written to file
program p1
uses a,b,c
const k=5,g=0
var x,y=integer;

function f1
fa,fb=integer;
fc=real;
begin
        get i;
        fc=-56.5;
        fb=fb5+fb;
        fc=5>3?3:fb;
        put fb;
end;

begin
        f1=5;
end .
token           converted token row             column
program         program         1               1
p1              id              1               9
uses            uses            2               1
a               id              2               6
,               ,               2               7
b               id              2               8
,               ,               2               9
c               id              2               10
const           const           3               1
k               id              3               7
=               =               3               8
5               num             3               9
,               ,               3               10
g               id              3               11
=               =               3               12
0               num             3               13
var             var             4               1
x               id              4               5
,               ,               4               6
y               id              4               7
=               =               4               8
integer         integer         4               9
;               ;               4               16
function                function                6               1
```

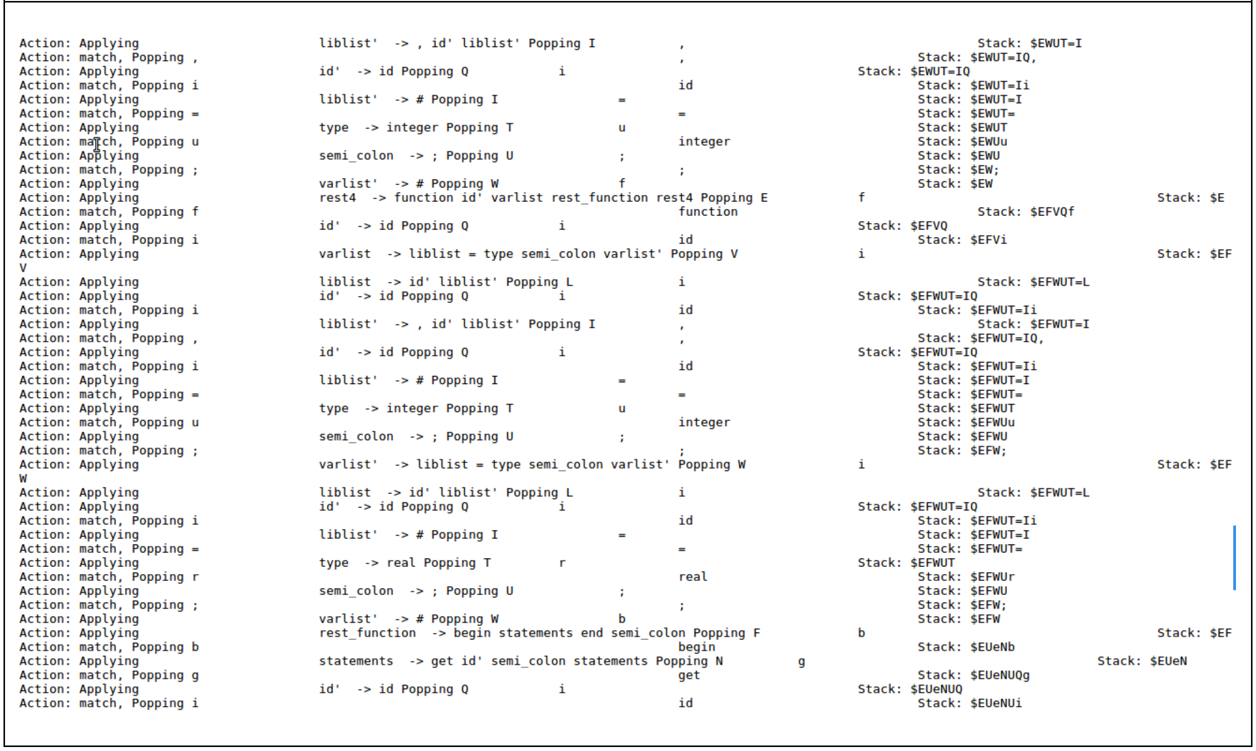


```
integer         integer         4               9
;               ;               4               16
function                function        6               1
f1              id              6               10
fa              id              7               1
,               ,               7               3
fb              id              7               4
=               =               7               6
integer         integer         7               7
;               ;               7               14
fc              id              8               1
=               =               8               3
real            real            8               4
;               ;               8               8
begin           begin           9               1
get             get             10              5
i               id              10              9
;               ;               10              10
fc              id              11              5
=               =               11              7
-56.5           num             11              8
;               ;               11              13
fb              id              12              5
=               =               12              7
fb5             id              12              8
+               +               12              11
fb              id              12              12
;               ;               12              14
fc              id              13              5
=               =               13              7
5               num             13              8
>               >               13              9
3               num             13              10
?               ?               13              11
3               num             13              12
:               :               13              13
fb              id              13              14
;               ;               13              16
put             put             14              5
fb              id              14              9
;               ;               14              11
end             end             15              1
;               ;               15              4
begin           begin           17              1
f1              id              18              5
=               =               18              7
5               num             18              8
;               ;               18              9
```

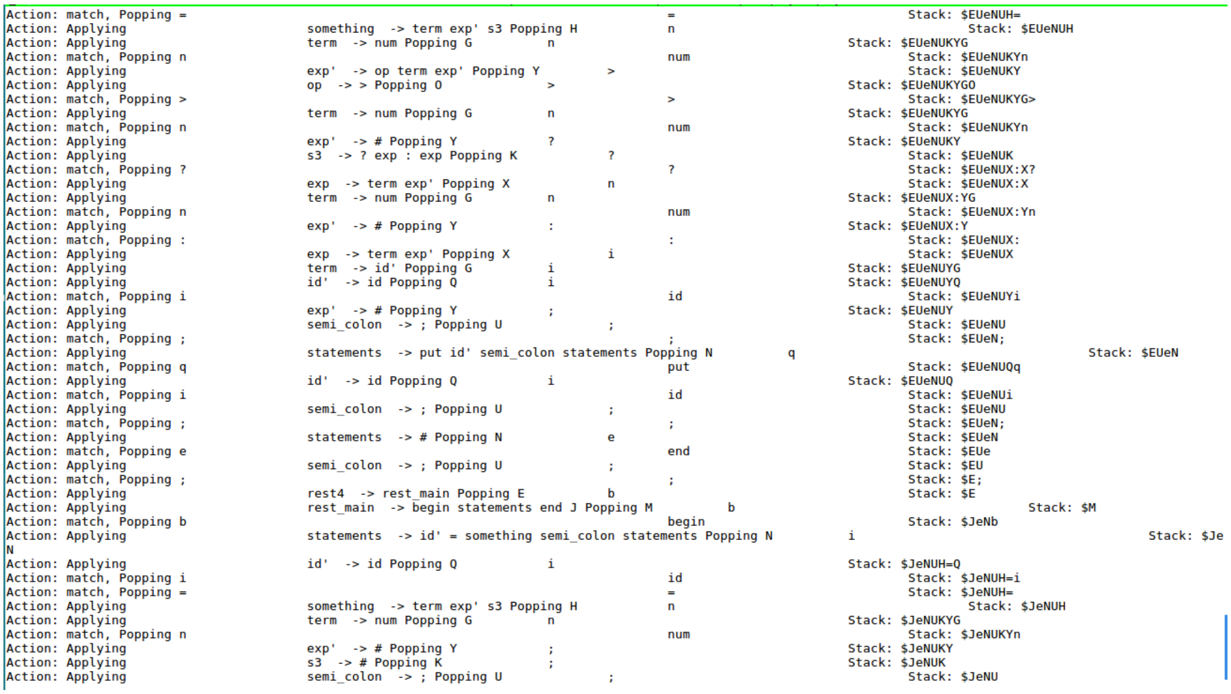
The above figures show the lexical tokenizing of the program



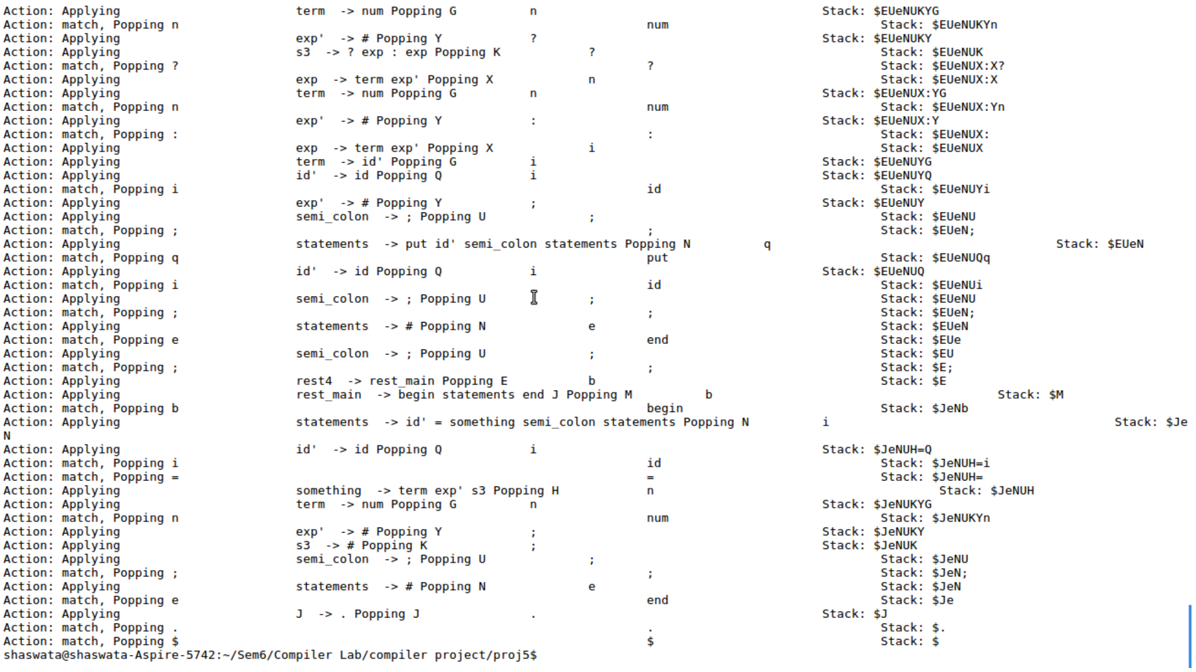
```
Parsing
672
Action: Applying                        start  -> program id' rest1 Popping S           p                                       Stack: $S
Action: match, Popping p                                                                program                         Stack: $AQp
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $AQ
Action: match, Popping i                                                                id                              Stack: $Ai
Action: Applying                        rest1  -> uses liblist rest2 Popping A          l                                       Stack: $A
Action: match, Popping l                                                                uses                            Stack: $BLl
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $BL
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $BIQ
Action: match, Popping i                                                                id                              Stack: $BIi
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $BI
Action: match, Popping ,                                                                ,                               Stack: $BIQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $BIQ
Action: match, Popping i                                                                id                              Stack: $BIi
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $BI
Action: match, Popping ,                                                                ,                               Stack: $BIQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $BIQ
Action: match, Popping i                                                                id                              Stack: $BIi
Action: Applying                        liblist'  -> # Popping I                c                               Stack: $BI
Action: Applying                        rest2  -> const const_list rest3 Popping B              c                                       Stack: $B
Action: match, Popping c                                                                const                           Stack: $DCc
Action: Applying                        const_list  -> id' = num const_list' Popping C          i                               Stack: $DC
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $DZn=Q
Action: match, Popping i                                                                id                              Stack: $DZn=i
Action: match, Popping =                                                                =                               Stack: $DZn=
Action: match, Popping n                                                                num                             Stack: $DZn
Action: Applying                        const_list'  -> , id' = num const_list' Popping Z               ,                               Stack: $DZ
Action: match, Popping ,                                                                ,                               Stack: $DZn=Q,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $DZn=Q
Action: match, Popping i                                                                id                              Stack: $DZn=i
Action: match, Popping =                                                                =                               Stack: $DZn=
Action: match, Popping n                                                                num                             Stack: $DZn
Action: Applying                        const_list'  -> # Popping Z             v                               Stack: $DZ
Action: Applying                        rest3  -> var varlist rest4 Popping D           v                                       Stack: $D
Action: match, Popping v                                                                var                             Stack: $EVv
Action: Applying                        varlist  -> liblist = type semi_colon varlist' Popping V                i                                               Stack: $EV
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $EWUT=L
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EWUT=Ii
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $EWUT=I
Action: match, Popping ,                                                                ,                               Stack: $EWUT=IQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EWUT=IQ
```



```
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $EWUT=I
Action: match, Popping ,                                                                ,                               Stack: $EWUT=IQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                               Stack: $EWUT=I
Action: match, Popping =                                                                =                               Stack: $EWUT=
Action: Applying                        type  -> integer Popping T              u                               Stack: $EWUT
Action: match, Popping u                                                                integer                         Stack: $EWUu
Action: Applying                        semi_colon  -> ; Popping U              ;                               Stack: $EWU
Action: match, Popping ;                                                                ;                               Stack: $EW;
Action: Applying                        varlist'  -> # Popping W                f                               Stack: $EW
Action: Applying                        rest4  -> function id' varlist rest_function rest4 Popping E            f                                               Stack: $E
Action: match, Popping f                                                                function                                Stack: $EFVQf
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFVQ
Action: match, Popping i                                                                id                              Stack: $EFVi
Action: Applying                        varlist  -> liblist = type semi_colon varlist' Popping V                i                                               Stack: $EF
V
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $EFWUT=L
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EFWUT=Ii
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $EFWUT=I
Action: match, Popping ,                                                                ,                               Stack: $EFWUT=IQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EFWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                               Stack: $EFWUT=I
Action: match, Popping =                                                                =                               Stack: $EFWUT=
Action: Applying                        type  -> integer Popping T              u                               Stack: $EFWUT
Action: match, Popping u                                                                integer                         Stack: $EFWUu
Action: Applying                        semi_colon  -> ; Popping U              ;                               Stack: $EFWU
Action: match, Popping ;                                                                ;                               Stack: $EFW;
Action: Applying                        varlist'  -> liblist = type semi_colon varlist' Popping W               i                                               Stack: $EF
W
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $EFWUT=L
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EFWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                               Stack: $EFWUT=I
Action: match, Popping =                                                                =                               Stack: $EFWUT=
Action: Applying                        type  -> real Popping T         r                                       Stack: $EFWUT
Action: match, Popping r                                                                real                            Stack: $EFWUr
Action: Applying                        semi_colon  -> ; Popping U              ;                               Stack: $EFWU
Action: match, Popping ;                                                                ;                               Stack: $EFW;
Action: Applying                        varlist'  -> # Popping W                b                               Stack: $EFW
Action: Applying                        rest_function  -> begin statements end semi_colon Popping F             b                                               Stack: $EF
Action: match, Popping b                                                                begin                           Stack: $EUeNb
Action: Applying                        statements  -> get id' semi_colon statements Popping N          g                               Stack: $EUeN
Action: match, Popping g                                                                get                             Stack: $EUeNUQg
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EUeNUQ
Action: match, Popping i                                                                id                              Stack: $EUeNUi
```



```
Action: match, Popping =                                                                =                               Stack: $EUeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            n                                       Stack: $EUeNUH
Action: Applying                        term  -> num Popping G          n                                       Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> op term exp' Popping Y         >                                       Stack: $EUeNUKY
Action: Applying                        op  -> > Popping O              >                                       Stack: $EUeNUKYGO
Action: match, Popping >                                                                >                               Stack: $EUeNUKYG>
Action: Applying                        term  -> num Popping G          n                                       Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> # Popping Y            ?                                       Stack: $EUeNUKY
Action: Applying                        s3  -> ? exp : exp Popping K            ?                                       Stack: $EUeNUK
Action: match, Popping ?                                                                ?                               Stack: $EUeNUX:X?
Action: Applying                        exp  -> term exp' Popping X             n                                       Stack: $EUeNUX:X
Action: Applying                        term  -> num Popping G          n                                       Stack: $EUeNUX:YG
Action: match, Popping n                                                                num                             Stack: $EUeNUX:Yn
Action: Applying                        exp'  -> # Popping Y            :                                       Stack: $EUeNUX:Y
Action: match, Popping :                                                                :                               Stack: $EUeNUX:
Action: Applying                        exp  -> term exp' Popping X             i                                       Stack: $EUeNUX
Action: Applying                        term  -> id' Popping G          i                                       Stack: $EUeNUYG
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EUeNUYQ
Action: match, Popping i                                                                id                              Stack: $EUeNUYi
Action: Applying                        exp'  -> # Popping Y            ;                                       Stack: $EUeNUY
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> put id' semi_colon statements Popping N          q                                       Stack: $EUeN
Action: match, Popping q                                                                put                             Stack: $EUeNUQq
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EUeNUQ
Action: match, Popping i                                                                id                              Stack: $EUeNUi
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> # Popping N              e                                       Stack: $EUeN
Action: match, Popping e                                                                end                             Stack: $EUe
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EU
Action: match, Popping ;                                                                ;                               Stack: $E;
Action: Applying                        rest4  -> rest_main Popping E           b                                       Stack: $E
Action: Applying                        rest_main  -> begin statements end J Popping M          b                                       Stack: $M
Action: match, Popping b                                                                begin                           Stack: $JeNb
Action: Applying                        statements  -> id' = something semi_colon statements Popping N          i                                       Stack: $Je
N
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $JeNUH=Q
Action: match, Popping i                                                                id                              Stack: $JeNUH=i
Action: match, Popping =                                                                =                               Stack: $JeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            n                                       Stack: $JeNUH
Action: Applying                        term  -> num Popping G          n                                       Stack: $JeNUKYG
Action: match, Popping n                                                                num                             Stack: $JeNUKYn
Action: Applying                        exp'  -> # Popping Y            ;                                       Stack: $JeNUKY
Action: Applying                        s3  -> # Popping K              ;                                       Stack: $JeNUK
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $JeNU
```

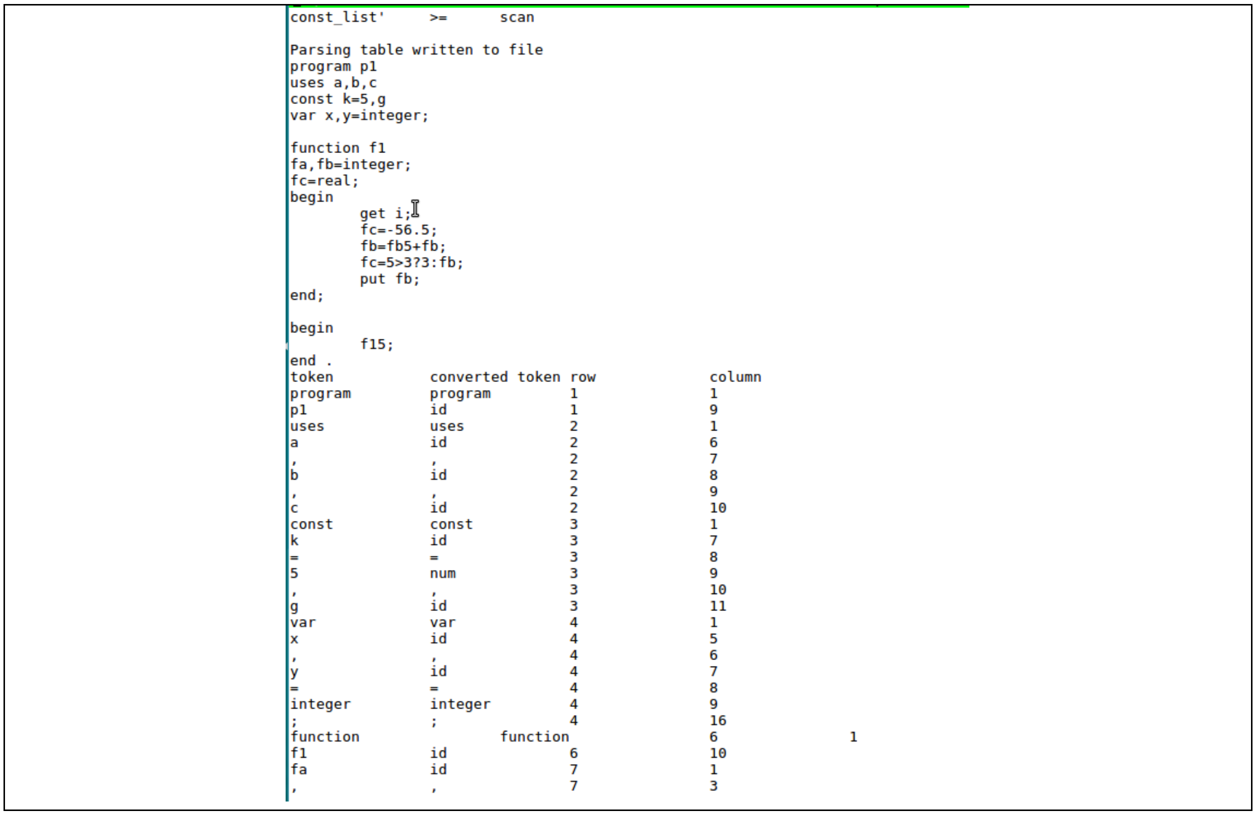


```
Action: Applying                        term  -> num Popping G          n                                       Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> # Popping Y            ?                                       Stack: $EUeNUKY
Action: Applying                        s3  -> ? exp : exp Popping K            ?                                       Stack: $EUeNUK
Action: match, Popping ?                                                                ?                               Stack: $EUeNUX:X?
Action: Applying                        exp  -> term exp' Popping X             n                                       Stack: $EUeNUX:X
Action: Applying                        term  -> num Popping G          n                                       Stack: $EUeNUX:YG
Action: match, Popping n                                                                num                             Stack: $EUeNUX:Yn
Action: Applying                        exp'  -> # Popping Y            :                                       Stack: $EUeNUX:Y
Action: match, Popping :                                                                :                               Stack: $EUeNUX:
Action: Applying                        exp  -> term exp' Popping X             i                                       Stack: $EUeNUX
Action: Applying                        term  -> id' Popping G          i                                       Stack: $EUeNUYG
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EUeNUYQ
Action: match, Popping i                                                                id                              Stack: $EUeNUYi
Action: Applying                        exp'  -> # Popping Y            ;                                       Stack: $EUeNUY
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> put id' semi_colon statements Popping N          q                                       Stack: $EUeN
Action: match, Popping q                                                                put                             Stack: $EUeNUQq
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EUeNUQ
Action: match, Popping i                                                                id                              Stack: $EUeNUi
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> # Popping N              e                                       Stack: $EUeN
Action: match, Popping e                                                                end                             Stack: $EUe
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EU
Action: match, Popping ;                                                                ;                               Stack: $E;
Action: Applying                        rest4  -> rest_main Popping E           b                                       Stack: $E
Action: Applying                        rest_main  -> begin statements end J Popping M          b                                       Stack: $M
Action: match, Popping b                                                                begin                           Stack: $JeNb
Action: Applying                        statements  -> id' = something semi_colon statements Popping N          i                                       Stack: $Je
N
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $JeNUH=Q
Action: match, Popping i                                                                id                              Stack: $JeNUH=i
Action: match, Popping =                                                                =                               Stack: $JeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            n                                       Stack: $JeNUH
Action: Applying                        term  -> num Popping G          n                                       Stack: $JeNUKYG
Action: match, Popping n                                                                num                             Stack: $JeNUKYn
Action: Applying                        exp'  -> # Popping Y            ;                                       Stack: $JeNUKY
Action: Applying                        s3  -> # Popping K              ;                                       Stack: $JeNUK
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $JeNU
Action: match, Popping ;                                                                ;                               Stack: $JeN;
Action: Applying                        statements  -> # Popping N              e                                       Stack: $JeN
Action: match, Popping e                                                                end                             Stack: $Je
Action: Applying                        J  -> . Popping J               .                                       Stack: $J
Action: match, Popping .                                                                .                               Stack: $.
Action: match, Popping $                                                                $                               Stack: $
shaswata@shaswata-Aspire-5742:~/Sem6/Compiler Lab/compiler project/proj5$
```

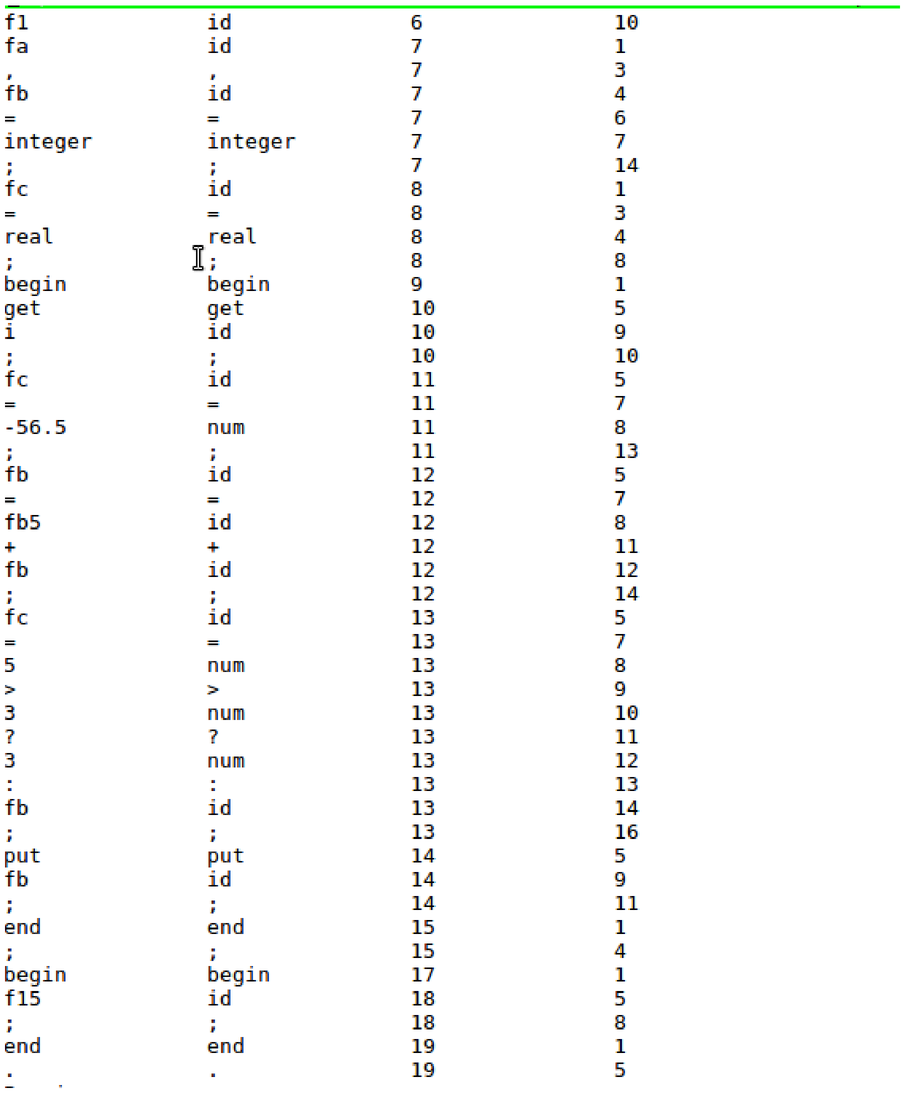
The above figures shows the parsing procedure

Test Input file 2:

```
program p1
uses a,b,c
const k=5,g
var x,y=integer;

function f1
fa,fb=integer;
fc=real;
begin
        get i;
        fc=-56.5;
        fb=fb5+fb;
        fc=5>3?3:fb;
        put fb;
end;

begin
        f15;
end .
```

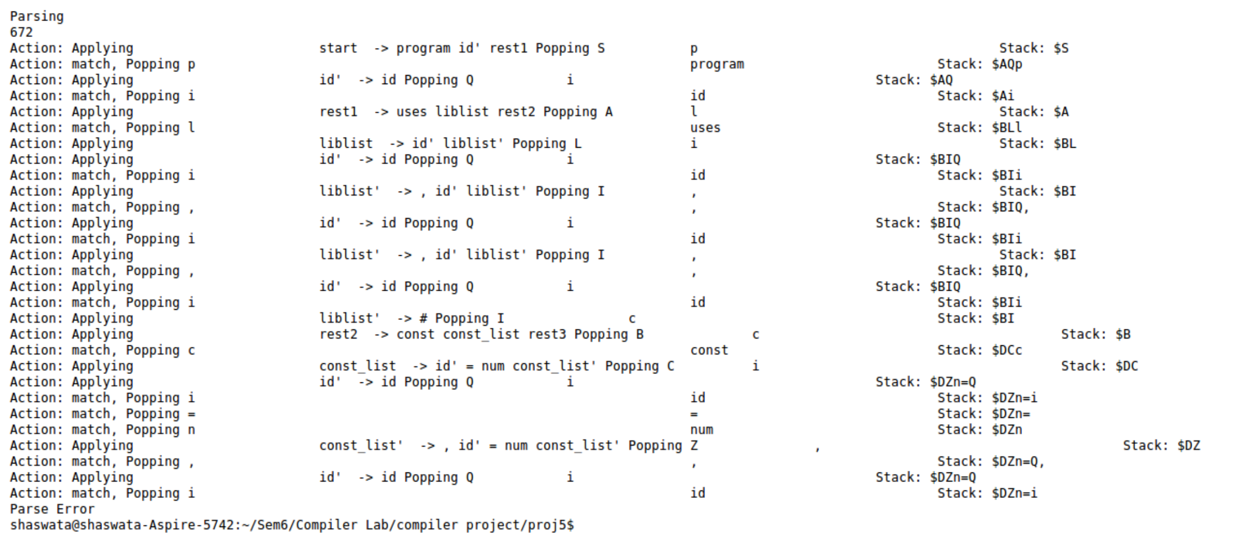


```
const_list'     >=      scan

Parsing table written to file
program p1
uses a,b,c
const k=5,g
var x,y=integer;

function f1
fa,fb=integer;
fc=real;
begin
        get i;
        fc=-56.5;
        fb=fb5+fb;
        fc=5>3?3:fb;
        put fb;
end;

begin
        f15;
end .
token           converted token row             column
program         program         1               1
p1              id              1               9
uses            uses            2               1
a               id              2               6
,               ,               2               7
b               id              2               8
,               ,               2               9
c               id              2               10
const           const           3               1
k               id              3               7
=               =               3               8
5               num             3               9
,               ,               3               10
g               id              3               11
var             var             4               1
x               id              4               5
,               ,               4               6
y               id              4               7
=               =               4               8
integer         integer         4               9
;               ;               4               16
function                function                6               1
f1              id              6               10
fa              id              7               1
,               ,               7               3
```



```
f1              id              6               10
fa              id              7               1
,               ,               7               3
fb              id              7               4
=               =               7               6
integer         integer         7               7
;               ;               7               14
fc              id              8               1
=               =               8               3
real            real            8               4
;               ;               8               8
begin           begin           9               1
get             get             10              5
i               id              10              9
;               ;               10              10
fc              id              11              5
=               =               11              7
-56.5           num             11              8
;               ;               11              13
fb              id              12              5
=               =               12              7
fb5             id              12              8
+               +               12              11
fb              id              12              12
;               ;               12              14
fc              id              13              5
=               =               13              7
5               num             13              8
>               >               13              9
3               num             13              10
?               ?               13              11
3               num             13              12
:               :               13              13
fb              id              13              14
;               ;               13              16
put             put             14              5
fb              id              14              9
;               ;               14              11
end             end             15              1
;               ;               15              4
begin           begin           17              1
f15             id              18              5
;               ;               18              8
end             end             19              1
.               .               19              5
```

The above figures show the lexical tokenizing of the program

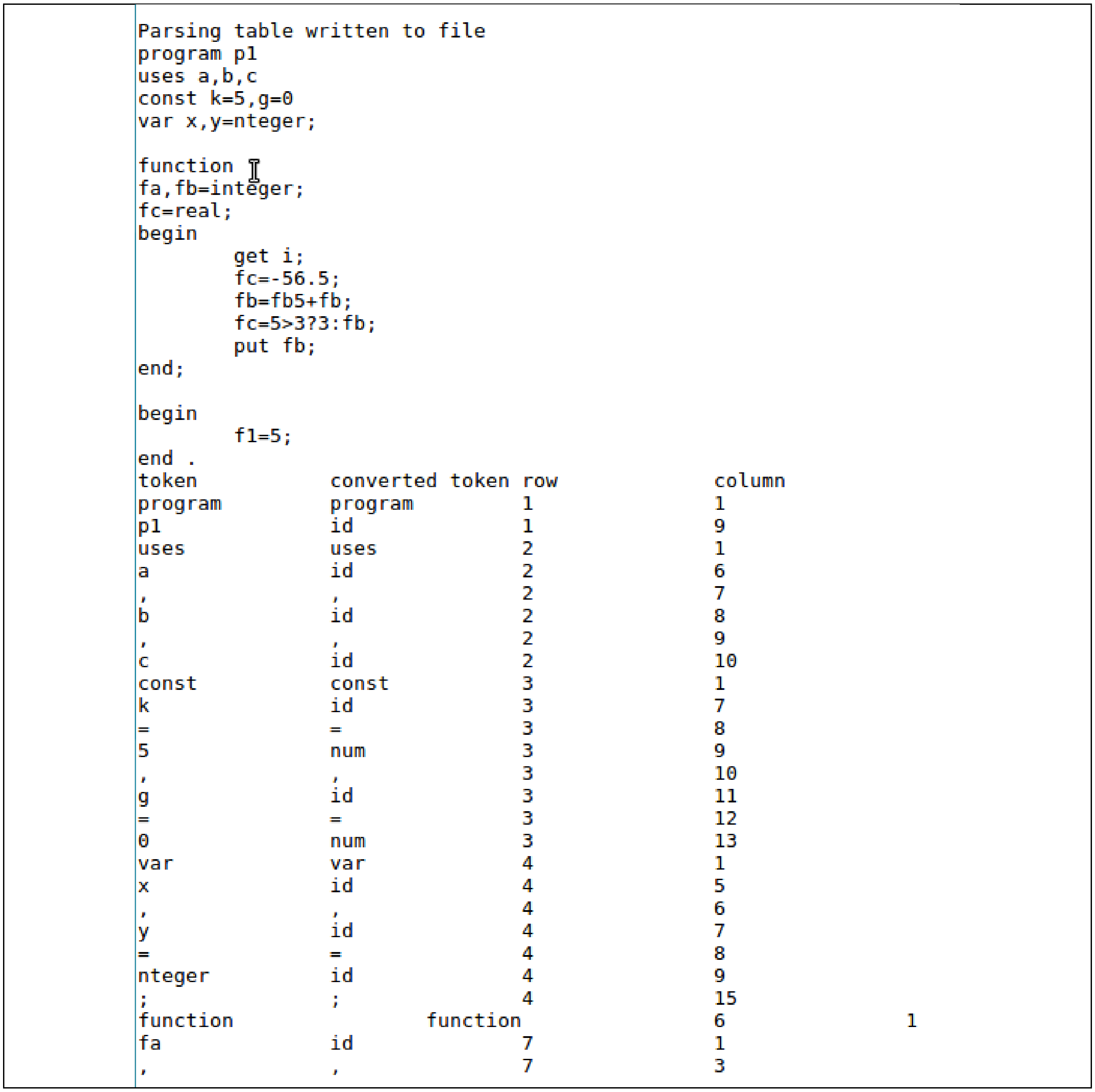


```
Parsing
672
Action: Applying                        start  -> program id' rest1 Popping S          p                                       Stack: $S
Action: match, Popping p                                                               program                         Stack: $AQp
Action: Applying                        id'  -> id Popping Q            i                                      Stack: $AQ
Action: match, Popping i                                                               id                              Stack: $Ai
Action: Applying                        rest1  -> uses liblist rest2 Popping A         l                                       Stack: $A
Action: match, Popping l                                                               uses                            Stack: $BLl
Action: Applying                        liblist  -> id' liblist' Popping L             i                                       Stack: $BL
Action: Applying                        id'  -> id Popping Q            i                                      Stack: $BIQ
Action: match, Popping i                                                               id                              Stack: $BIi
Action: Applying                        liblist'  -> , id' liblist' Popping I          ,                                       Stack: $BI
Action: match, Popping ,                                                               ,                               Stack: $BIQ,
Action: Applying                        id'  -> id Popping Q            i                                      Stack: $BIQ
Action: match, Popping i                                                               id                              Stack: $BIi
Action: Applying                        liblist'  -> , id' liblist' Popping I          ,                                       Stack: $BI
Action: match, Popping ,                                                               ,                               Stack: $BIQ,
Action: Applying                        id'  -> id Popping Q            i                                      Stack: $BIQ
Action: match, Popping i                                                               id                              Stack: $BIi
Action: Applying                        liblist'  -> # Popping I                c                                      Stack: $BI
Action: Applying                        rest2  -> const const_list rest3 Popping B              c                                       Stack: $B
Action: match, Popping c                                                               const                           Stack: $DCc
Action: Applying                        const_list  -> id' = num const_list' Popping C          i                                       Stack: $DC
Action: Applying                        id'  -> id Popping Q            i                                      Stack: $DZn=Q
Action: match, Popping i                                                               id                              Stack: $DZn=i
Action: match, Popping =                                                               =                               Stack: $DZn=
Action: match, Popping n                                                               num                             Stack: $DZn
Action: Applying                        const_list'  -> , id' = num const_list' Popping Z               ,                                       Stack: $DZ
Action: match, Popping ,                                                               ,                               Stack: $DZn=Q,
Action: Applying                        id'  -> id Popping Q            i                                      Stack: $DZn=Q
Action: match, Popping i                                                               id                              Stack: $DZn=i
Parse Error
shaswata@shaswata-Aspire-5742:~/Sem6/Compiler Lab/compiler project/proj5$
```

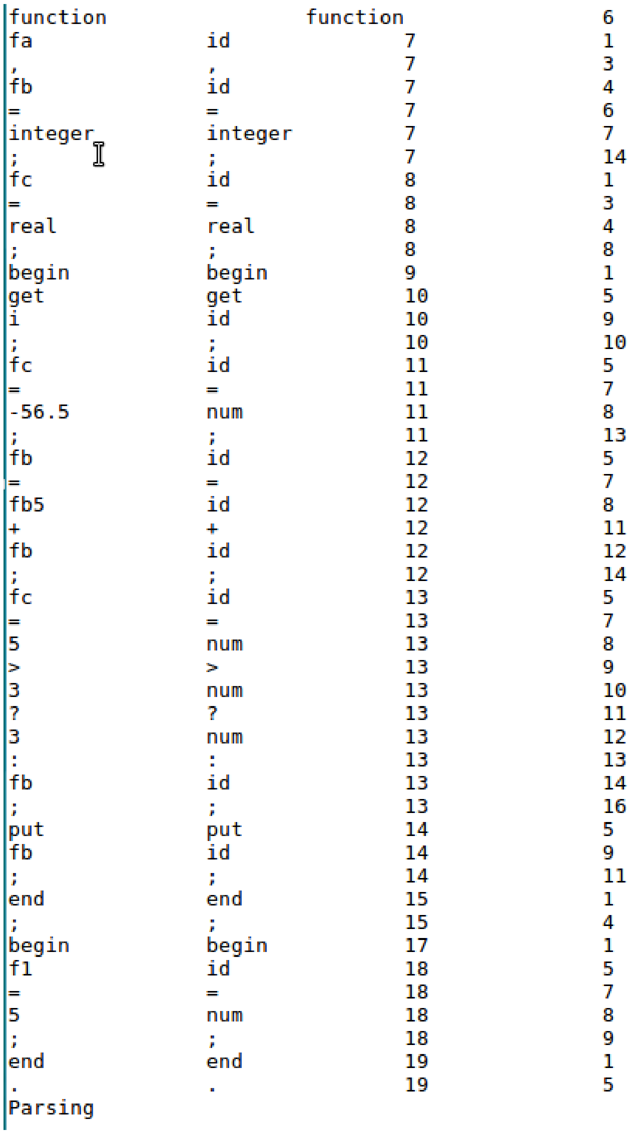
The above figures shows the parsing procedure

Test Input file 3:

```
program p1
uses a,b,c
const k=5,g=0
var x,y=nteger;

function
fa,fb=integer;
fc=real;
begin
        get i;
        fc=-56.5;
        fb=fb5+fb;
        fc=5>3?3:fb;
        put fb;
end;

begin
        f1=5;
end .
```

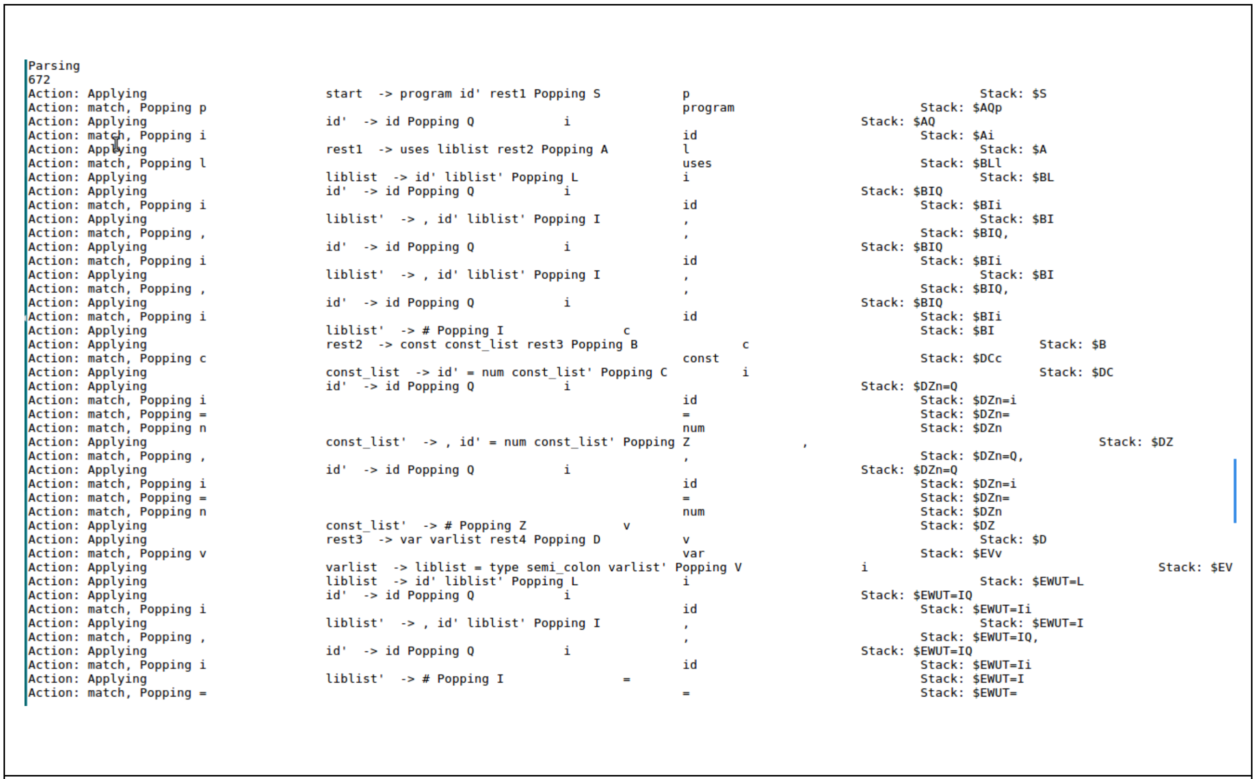


```
Parsing table written to file
program p1
uses a,b,c
const k=5,g=0
var x,y=nteger;

function 
fa,fb=integer;
fc=real;
begin
        get i;
        fc=-56.5;
        fb=fb5+fb;
        fc=5>3?3:fb;
        put fb;
end;

begin
        f1=5;
end .
token           converted token row              column
program         program         1                1
p1              id              1                9
uses            uses            2                1
a               id              2                6
,               ,               2                7
b               id              2                8
,               ,               2                9
c               id              2                10
const           const           3                1
k               id              3                7
=               =               3                8
5               num             3                9
,               ,               3                10
g               id              3                11
=               =               3                12
0               num             3                13
var             var             4                1
x               id              4                5
,               ,               4                6
y               id              4                7
=               =               4                8
nteger          id              4                9
;               ;               4                15
function                function                 6                1
fa              id              7                1
,               ,               7                3
```

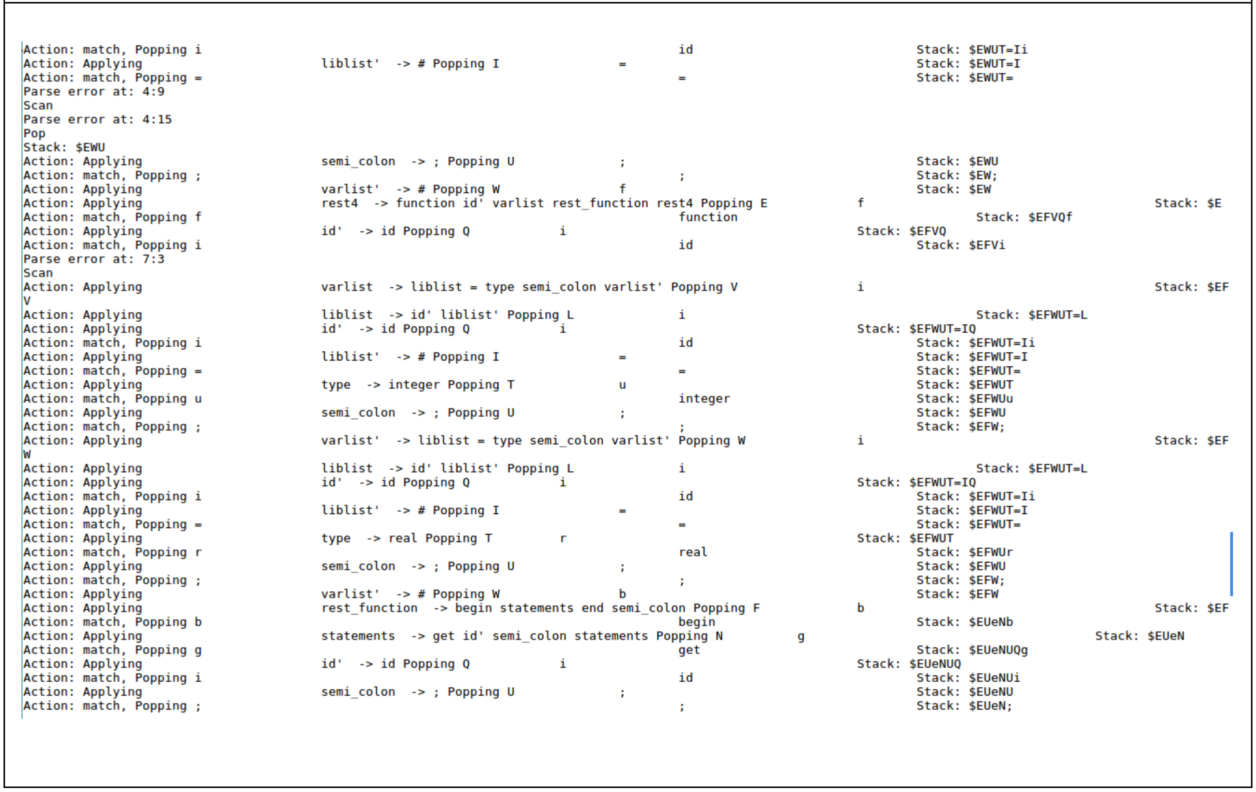


```
function                 function       6
fa              id               7      1
,               ,                7      3
fb              id               7      4
=               =                7      6
integer         integer          7      7
;               ;                7      14
fc              id               8      1
=               =                8      3
real            real             8      4
;               ;                8      8
begin           begin            9      1
get             get              10     5
i               id               10     9
;               ;                10     10
fc              id               11     5
=               =                11     7
-56.5           num              11     8
;               ;                11     13
fb              id               12     5
=               =                12     7
fb5             id               12     8
+               +                12     11
fb              id               12     12
;               ;                12     14
fc              id               13     5
=               =                13     7
5               num              13     8
>               >                13     9
3               num              13     10
?               ?                13     11
3               num              13     12
:               :                13     13
fb              id               13     14
;               ;                13     16
put             put              14     5
fb              id               14     9
;               ;                14     11
end             end              15     1
;               ;                15     4
begin           begin            17     1
f1              id               18     5
=               =                18     7
5               num              18     8
;               ;                18     9
end             end              19     1
.               .                19     5
Parsing
```

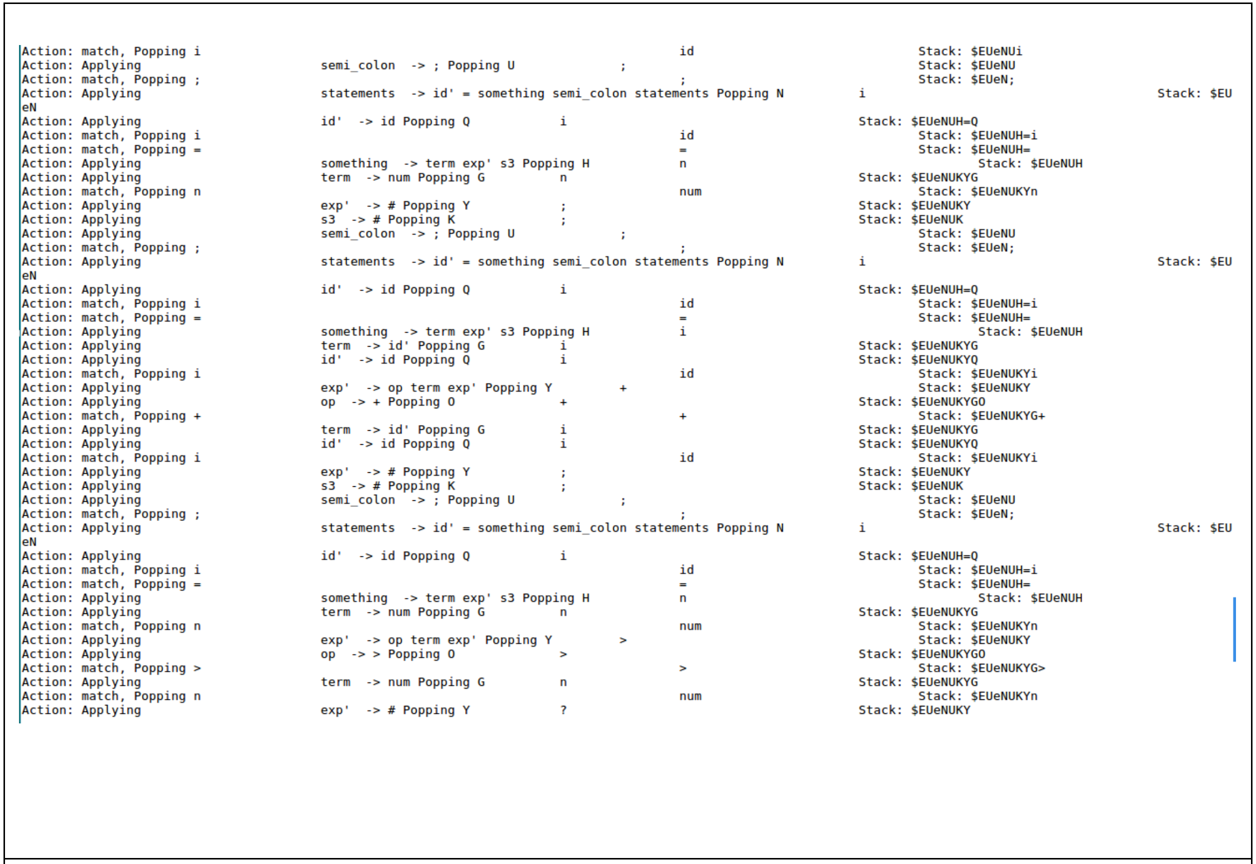
The above figures show the lexical tokenizing of the program



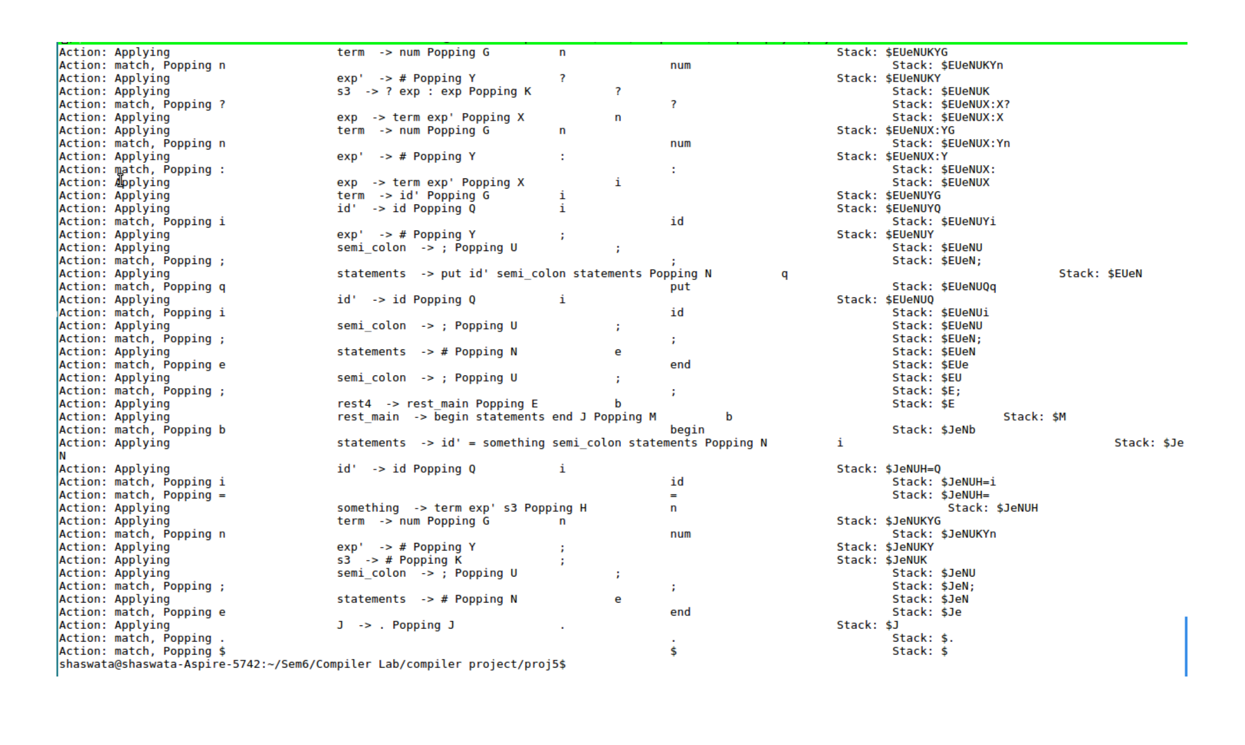
```
Parsing
672
Action: Applying                        start  -> program id' rest1 Popping S           p                                       Stack: $S
Action: match, Popping p                                                                program                         Stack: $AQp
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $AQ
Action: match, Popping i                                                                id                              Stack: $Ai
Action: Applying                        rest1  -> uses liblist rest2 Popping A          l                                       Stack: $A
Action: match, Popping l                                                                uses                            Stack: $BLl
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $BL
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $BIQ
Action: match, Popping i                                                                id                              Stack: $BIi
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $BI
Action: match, Popping ,                                                                ,                               Stack: $BIQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $BIQ
Action: match, Popping i                                                                id                              Stack: $BIi
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $BI
Action: match, Popping ,                                                                ,                               Stack: $BIQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $BIQ
Action: match, Popping i                                                                id                              Stack: $BIi
Action: Applying                        liblist'  -> # Popping I                c                                       Stack: $BI
Action: Applying                        rest2  -> const const_list rest3 Popping B              c                                       Stack: $B
Action: match, Popping c                                                                const                           Stack: $DCc
Action: Applying                        const_list  -> id' = num const_list' Popping C          i                                       Stack: $DC
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $DZn=Q
Action: match, Popping i                                                                id                              Stack: $DZn=i
Action: match, Popping =                                                                =                               Stack: $DZn=
Action: match, Popping n                                                                num                             Stack: $DZn
Action: Applying                        const_list'  -> , id' = num const_list' Popping Z               ,                                       Stack: $DZ
Action: match, Popping ,                                                                ,                               Stack: $DZn=Q,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $DZn=Q
Action: match, Popping i                                                                id                              Stack: $DZn=i
Action: match, Popping =                                                                =                               Stack: $DZn=
Action: match, Popping n                                                                num                             Stack: $DZn
Action: Applying                        const_list'  -> # Popping Z             v                                       Stack: $DZ
Action: Applying                        rest3  -> var varlist rest4 Popping D           v                                       Stack: $D
Action: match, Popping v                                                                var                             Stack: $EVv
Action: Applying                        varlist  -> liblist = type semi_colon varlist' Popping V                i                                       Stack: $EV
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $EWUT=L
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EWUT=Ii
Action: Applying                        liblist'  -> , id' liblist' Popping I           ,                                       Stack: $EWUT=I
Action: match, Popping ,                                                                ,                               Stack: $EWUT=IQ,
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                                       Stack: $EWUT=I
Action: match, Popping =                                                                =                               Stack: $EWUT=
```



```
Action: match, Popping i                                                                id                              Stack: $EWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                                       Stack: $EWUT=I
Action: match, Popping =                                                                =                               Stack: $EWUT=
Parse error at: 4:9
Scan
Parse error at: 4:15
Pop
Stack: $EWU
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EWU
Action: match, Popping ;                                                                ;                               Stack: $EW;
Action: Applying                        varlist'  -> # Popping W                f                                       Stack: $EW
Action: Applying                        rest4  -> function id' varlist rest_function rest4 Popping E            f                                       Stack: $E
Action: match, Popping f                                                                function                                Stack: $EFVQf
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFVQ
Action: match, Popping i                                                                id                              Stack: $EFVi
Parse error at: 7:3
Scan
Action: Applying                        varlist  -> liblist = type semi_colon varlist' Popping V                i                                       Stack: $EF
V
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $EFWUT=L
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EFWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                                       Stack: $EFWUT=I
Action: match, Popping =                                                                =                               Stack: $EFWUT=
Action: Applying                        type  -> integer Popping T              u                                       Stack: $EFWUT
Action: match, Popping u                                                                integer                         Stack: $EFWUu
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EFWU
Action: match, Popping ;                                                                ;                               Stack: $EFW;
Action: Applying                        varlist'  -> liblist = type semi_colon varlist' Popping W               i                                       Stack: $EF
W
Action: Applying                        liblist  -> id' liblist' Popping L              i                                       Stack: $EFWUT=L
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EFWUT=IQ
Action: match, Popping i                                                                id                              Stack: $EFWUT=Ii
Action: Applying                        liblist'  -> # Popping I                =                                       Stack: $EFWUT=I
Action: match, Popping =                                                                =                               Stack: $EFWUT=
Action: Applying                        type  -> real Popping T         r                                       Stack: $EFWUT
Action: match, Popping r                                                                real                            Stack: $EFWUr
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EFWU
Action: match, Popping ;                                                                ;                               Stack: $EFW;
Action: Applying                        varlist'  -> # Popping W                b                                       Stack: $EFW
Action: Applying                        rest_function  -> begin statements end semi_colon Popping F             b                                       Stack: $EF
Action: match, Popping b                                                                begin                           Stack: $EUeNb
Action: Applying                        statements  -> get id' semi_colon statements Popping N          g                                       Stack: $EUeN
Action: match, Popping g                                                                get                             Stack: $EUeNUQg
Action: Applying                        id'  -> id Popping Q            i                                       Stack: $EUeNUQ
Action: match, Popping i                                                                id                              Stack: $EUeNUi
Action: Applying                        semi_colon  -> ; Popping U              ;                                       Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
```



```
Action: match, Popping i                                                                id                              Stack: $EUeNUi
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> id' = something semi_colon statements Popping N           i                                      Stack: $EU
eN
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUH=Q
Action: match, Popping i                                                                id                              Stack: $EUeNUH=i
Action: match, Popping =                                                                =                               Stack: $EUeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            n                                       Stack: $EUeNUH
Action: Applying                        term  -> num Popping G           n                                      Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> # Popping Y             ;                                      Stack: $EUeNUKY
Action: Applying                        s3  -> # Popping K               ;                                      Stack: $EUeNUK
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> id' = something semi_colon statements Popping N           i                                      Stack: $EU
eN
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUH=Q
Action: match, Popping i                                                                id                              Stack: $EUeNUH=i
Action: match, Popping =                                                                =                               Stack: $EUeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            i                                       Stack: $EUeNUH
Action: Applying                        term  -> id' Popping G           i                                      Stack: $EUeNUKYG
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUKYQ
Action: match, Popping i                                                                id                              Stack: $EUeNUKYi
Action: Applying                        exp'  -> op term exp' Popping Y          +                                      Stack: $EUeNUKY
Action: Applying                        op  -> + Popping O               +                                      Stack: $EUeNUKYGO
Action: match, Popping +                                                                +                               Stack: $EUeNUKYG+
Action: Applying                        term  -> id' Popping G           i                                      Stack: $EUeNUKYG
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUKYQ
Action: match, Popping i                                                                id                              Stack: $EUeNUKYi
Action: Applying                        exp'  -> # Popping Y             ;                                      Stack: $EUeNUKY
Action: Applying                        s3  -> # Popping K               ;                                      Stack: $EUeNUK
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> id' = something semi_colon statements Popping N           i                                      Stack: $EU
eN
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUH=Q
Action: match, Popping i                                                                id                              Stack: $EUeNUH=i
Action: match, Popping =                                                                =                               Stack: $EUeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            n                                       Stack: $EUeNUH
Action: Applying                        term  -> num Popping G           n                                      Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> op term exp' Popping Y          >                                      Stack: $EUeNUKY
Action: Applying                        op  -> > Popping O               >                                      Stack: $EUeNUKYGO
Action: match, Popping >                                                                >                               Stack: $EUeNUKYG>
Action: Applying                        term  -> num Popping G           n                                      Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> # Popping Y             ?                                      Stack: $EUeNUKY
```



```
Action: Applying                        term  -> num Popping G           n                                      Stack: $EUeNUKYG
Action: match, Popping n                                                                num                             Stack: $EUeNUKYn
Action: Applying                        exp'  -> # Popping Y             ?                                      Stack: $EUeNUKY
Action: Applying                        s3  -> ? exp : exp Popping K             ?                                      Stack: $EUeNUK
Action: match, Popping ?                                                                ?                               Stack: $EUeNUX:X?
Action: Applying                        exp  -> term exp' Popping X              n                                      Stack: $EUeNUX:X
Action: Applying                        term  -> num Popping G           n                                      Stack: $EUeNUX:YG
Action: match, Popping n                                                                num                             Stack: $EUeNUX:Yn
Action: Applying                        exp'  -> # Popping Y             :                                      Stack: $EUeNUX:Y
Action: match, Popping :                                                                :                               Stack: $EUeNUX:
Action: Applying                        exp  -> term exp' Popping X              i                                      Stack: $EUeNUX
Action: Applying                        term  -> id' Popping G           i                                      Stack: $EUeNUYG
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUYQ
Action: match, Popping i                                                                id                              Stack: $EUeNUYi
Action: Applying                        exp'  -> # Popping Y             ;                                      Stack: $EUeNUY
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> put id' semi_colon statements Popping N           q                                      Stack: $EUeN
Action: match, Popping q                                                                put                             Stack: $EUeNUQq
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $EUeNUQ
Action: match, Popping i                                                                id                              Stack: $EUeNUi
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $EUeNU
Action: match, Popping ;                                                                ;                               Stack: $EUeN;
Action: Applying                        statements  -> # Popping N               e                                      Stack: $EUeN
Action: match, Popping e                                                                end                             Stack: $EUe
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $EU
Action: match, Popping ;                                                                ;                               Stack: $E;
Action: Applying                        rest4  -> rest_main Popping E            b                                      Stack: $E
Action: Applying                        rest_main  -> begin statements end J Popping M          b                                      Stack: $M
Action: match, Popping b                                                                begin                           Stack: $JeNb
Action: Applying                        statements  -> id' = something semi_colon statements Popping N           i                                      Stack: $Je
N
Action: Applying                        id'   -> id Popping Q            i                                      Stack: $JeNUH=Q
Action: match, Popping i                                                                id                              Stack: $JeNUH=i
Action: match, Popping =                                                                =                               Stack: $JeNUH=
Action: Applying                        something  -> term exp' s3 Popping H            n                                       Stack: $JeNUH
Action: Applying                        term  -> num Popping G           n                                      Stack: $JeNUKYG
Action: match, Popping n                                                                num                             Stack: $JeNUKYn
Action: Applying                        exp'  -> # Popping Y             ;                                      Stack: $JeNUKY
Action: Applying                        s3  -> # Popping K               ;                                      Stack: $JeNUK
Action: Applying                        semi_colon  -> ; Popping U               ;                                      Stack: $JeNU
Action: match, Popping ;                                                                ;                               Stack: $JeN;
Action: Applying                        statements  -> # Popping N               e                                      Stack: $JeN
Action: match, Popping e                                                                end                             Stack: $Je
Action: Applying                        J  -> . Popping J                .                                      Stack: $J
Action: match, Popping .                                                                .                               Stack: $.
Action: match, Popping $                                                                $                               Stack: $
shaswata@shaswata-Aspire-5742:~/Sem6/Compiler Lab/compiler project/proj5$
```

The above figures shows the parsing procedure