DEPARTMENT OF COMPUTER ENGINEERING

Semester	T.E. Semester VI– SPCC
Subject	Software Engineering
Subject Professor In-charge	Prof. Pankaj Vanvari
Assisting Teachers	Prof. Pankaj Vanvari
Laboratory	M310B

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title:

Parser

Approach:

1. **Parser Function (Parser):**
    - Initialize the parsing index **pin** to 0.
    - Call the start symbol function **S** with the tokenized input and the parsing index.
    - Return the result of the start symbol function.
2. **Start Symbol Function (S):**
    - Check if the current token represents a valid starting symbol.
    - If the condition is met:
        - Move to the next token.
        - Check if the next token indicates the beginning of an expression.
        - If yes, call the expression function **E**.
        - Return the result of the expression function.

- If the conditions are not met, return false.

3. **Expression Function (E):**

    - Call the term function **T**.
    - If the term function returns true:
        - Call the expression prime function **E_**.
        - Return the result of the expression prime function.
    - If the term function fails, return false.

4. **Expression Prime Function (E_):**

    - Check if the current token represents an addition or subtraction operator.
    - If yes:
        - Move to the next token.
        - Call the term function **T**.
        - If the term function returns true, call the expression prime function recursively.
        - Return the result of the recursive call.
    - If the conditions are not met, return true.

5. **Term Function (T):**

    - Call the factor function **F**.
    - If the factor function returns true:
        - Call the term prime function **T_**.
        - Return the result of the term prime function.
    - If the factor function fails, return false.

6. **Term Prime Function (T_):**

    - Check if the current token represents a multiplication or division operator.
    - If yes:
        - Move to the next token.
        - Call the factor function **F**.

        - If the factor function returns true, call the term prime function recursively.
        - Return the result of the recursive call.
    - If the conditions are not met, return true.

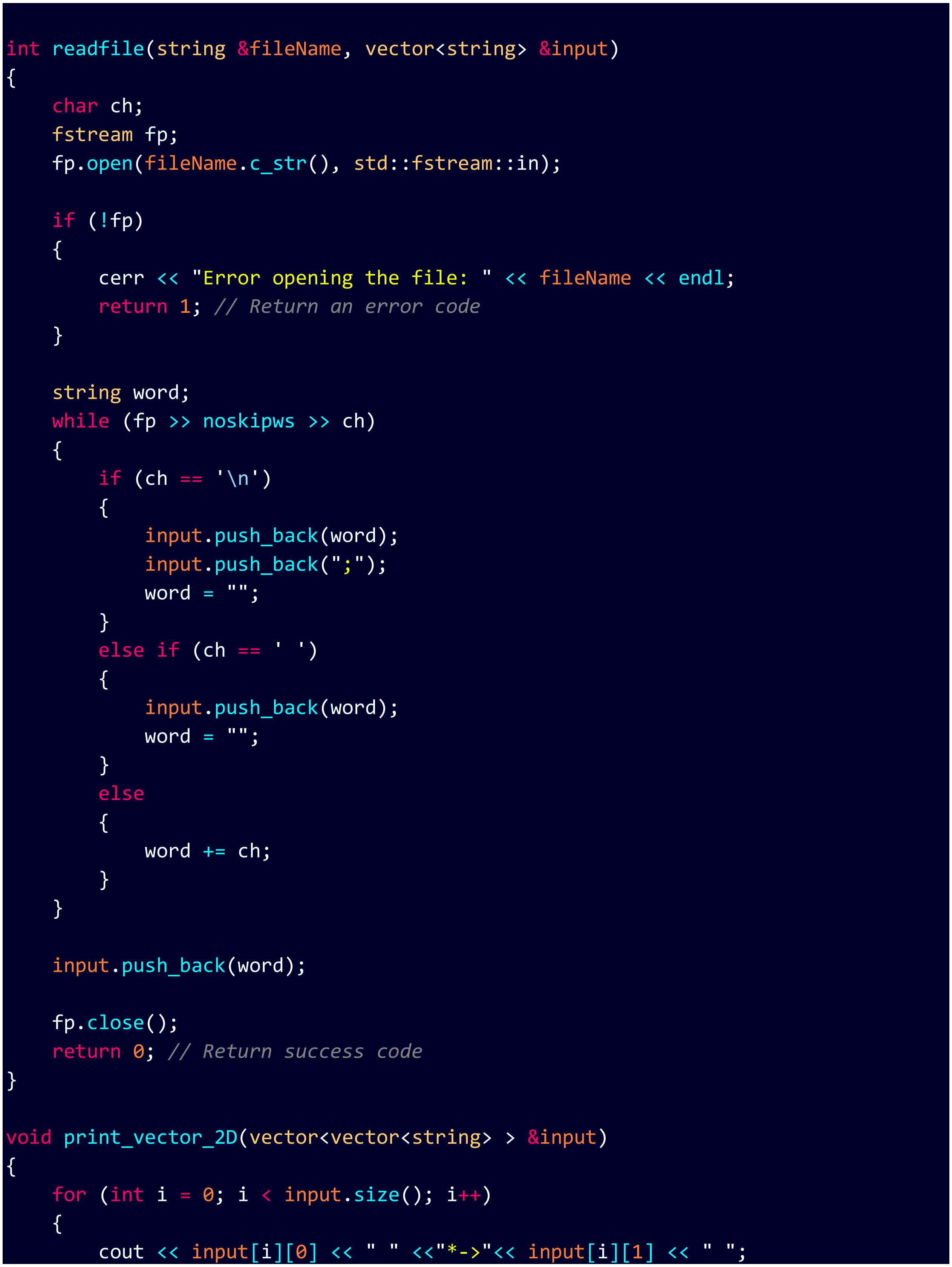
7. **Factor Function (F):**
    - Check the type of the current token:
        - If it represents an open parenthesis:
            - Move to the next token.
            - Call the expression function **E**.
            - If the expression function returns true and the next token is a closing parenthesis, move to the next token and return true.
        - If it represents an integer, float constant, or identifier, move to the next token and return true.
    - If none of the conditions are met, return false.

8. **P Function (P):**
    - Call the factor function **F**.
    - If the factor function returns true, call the P prime function **P_**.
    - Return the result of the P prime function.

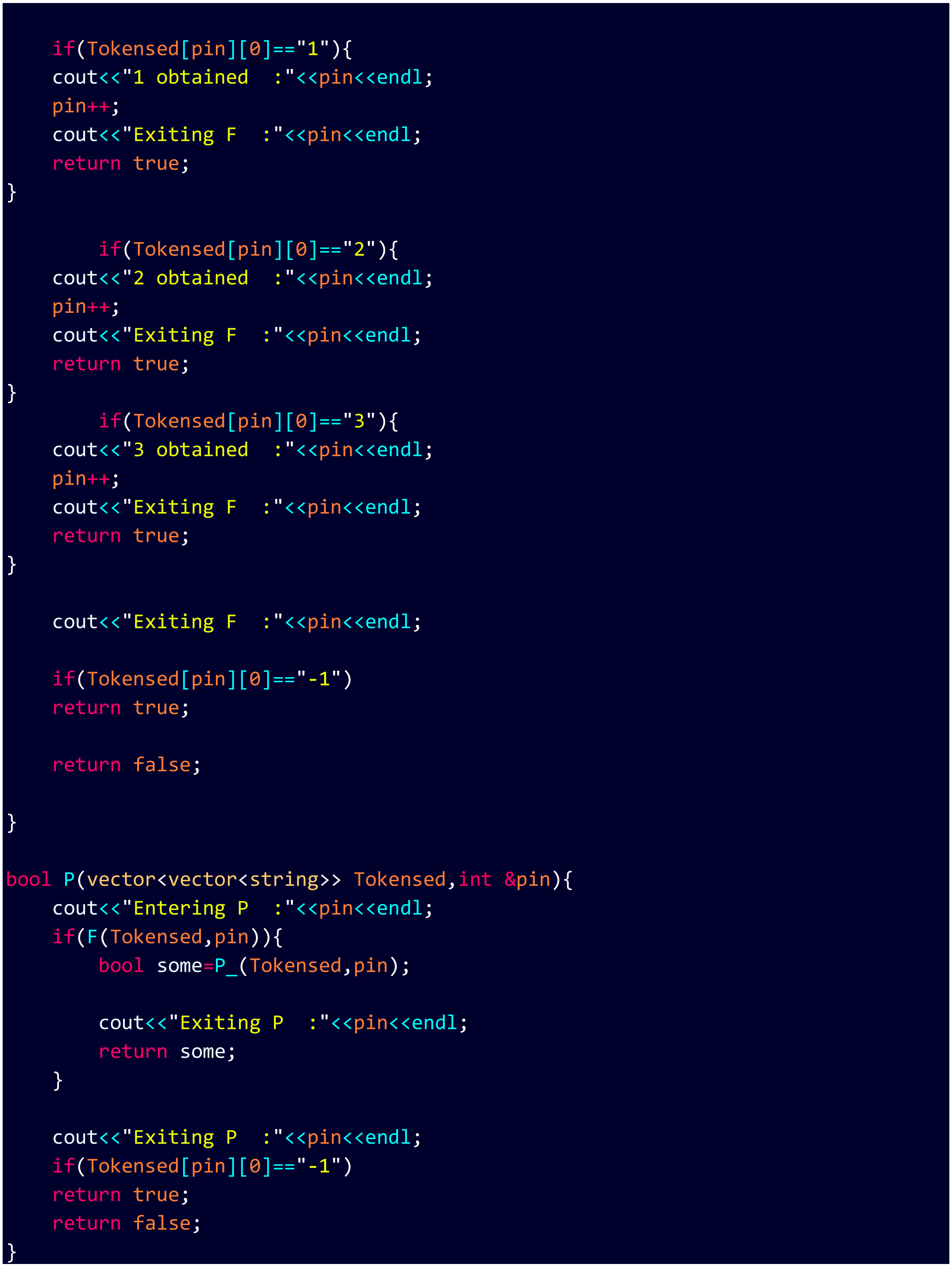
9. **P Prime Function (P_):**
    - Check if the current token represents the exponentiation operator.
    - If yes, move to the next token and call the factor function **F**.
    - Return true if the factor function returns true, otherwise return false.

Implementation:

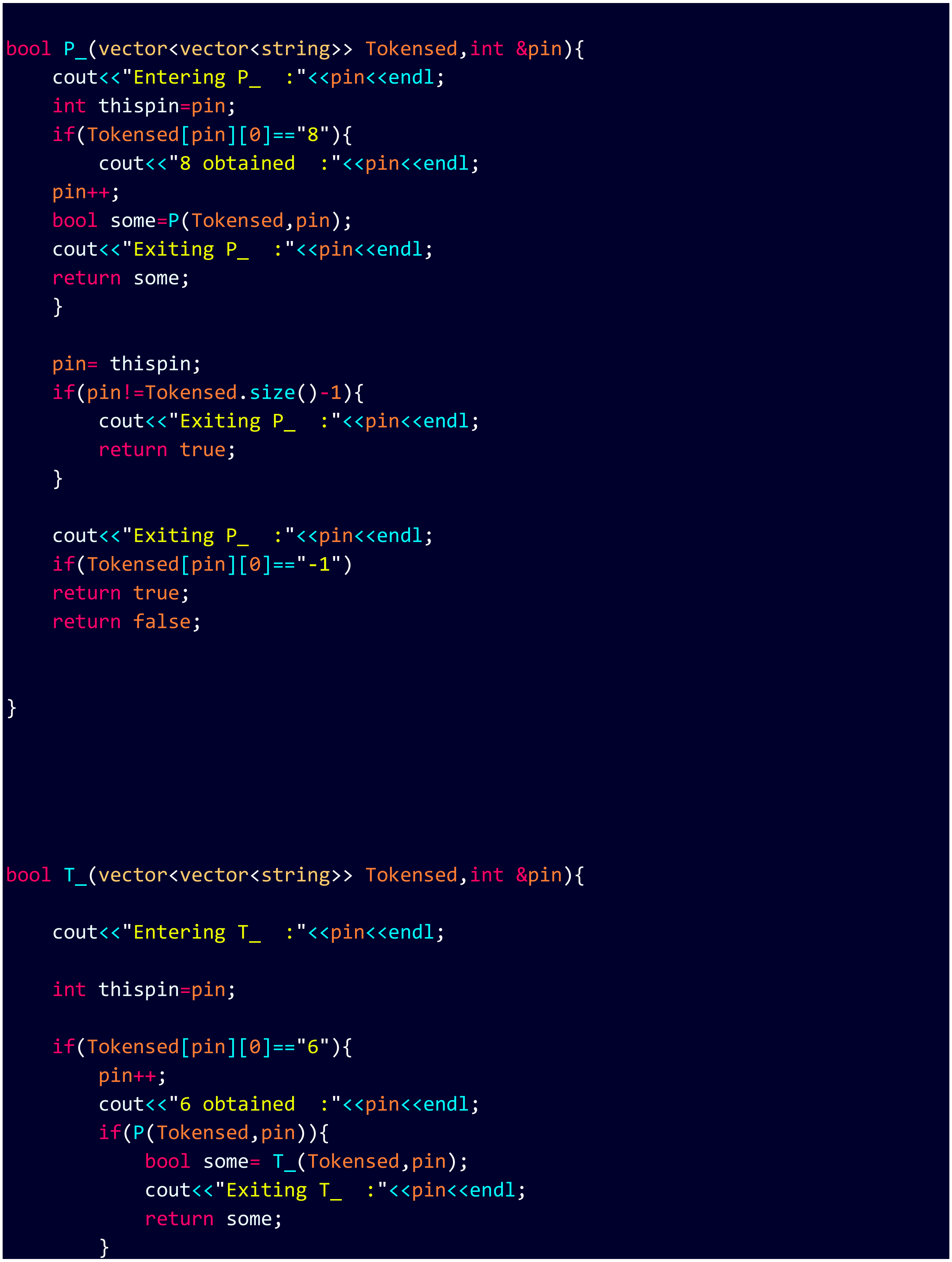
```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
using namespace std;
```



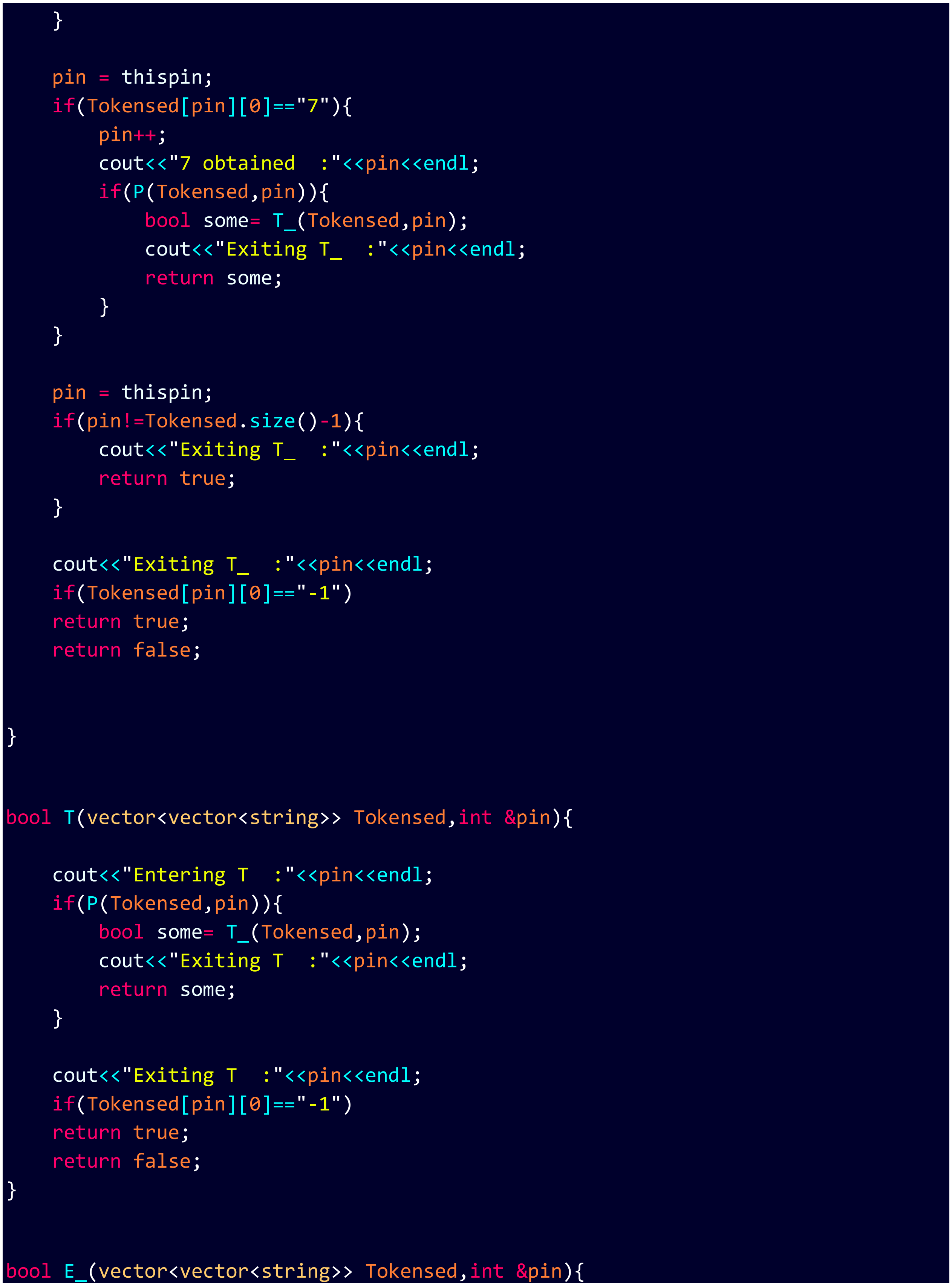
```
int readfile(string &fileName, vector<string> &input)
{
    char ch;
    fstream fp;
    fp.open(fileName.c_str(), std::fstream::in);

    if (!fp)
    {
        cerr << "Error opening the file: " << fileName << endl;
        return 1; // Return an error code
    }

    string word;
    while (fp >> noskipws >> ch)
    {
        if (ch == '\n')
        {
            input.push_back(word);
            input.push_back(";");
            word = "";
        }
        else if (ch == ' ')
        {
            input.push_back(word);
            word = "";
        }
        else
        {
            word += ch;
        }
    }

    input.push_back(word);

    fp.close();
    return 0; // Return success code
}

void print_vector_2D(vector<vector<string> > &input)
{
    for (int i = 0; i < input.size(); i++)
    {
        cout << input[i][0] << " " <<"*->"<< input[i][1] << " ";
        cout << endl;
    }
    cout << endl;
}

void print_vector(vector<string> &input)
{
    for (int i = 0; i < input.size(); i++)
    {
        cout << input[i] << " ";
    }
    cout << endl;
}

void Tokenization(vector<string> &input, vector<vector<string> > &Tokensed,
map<string, string> &keywords, map<string, int> &intcp, map<string, float>
&floatcp, map<string, string> &idp)
{
    int idc = 0;
    int intcc = 0;
    int floatcc = 0;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == ";")
            continue;

        if (keywords.find(input[i]) != keywords.end())
        {

            Tokensed.push_back({keywords[input[i]], "NA"});
        }
        else
        {
            // if the value in not in keyword db it can be eithre identifier or
constant

            string curr = input[i];
            char first_of_curr = curr[0]; // foc
            int val_of_foc = first_of_curr - '0';
            // cout << val_of_foc << endl;
            if (val_of_foc >= 0 && val_of_foc <= 9)
            {
                bool isfloat = false;
                for (auto x : curr)
                {
                    if (x == '.')
                        isfloat = true;
                }
                if (isfloat)
                {
                    float v = atof(curr.c_str());
                    string p = to_string(floatcc);
                    Tokensed.push_back({"3", p});
                    floatcp[p] = v;
                    floatcc++;
                }
                else
                {
                    int v = stoi(curr);
                    string p = to_string(intcc);
                    Tokensed.push_back({"2", p});
                    intcp[p] = v;
                    intcc++;
                }
            }
            else
            {
                string p = to_string(idc);

                Tokensed.push_back({"1", p});
                idp[p] = curr;
                idc++;
            }
        }
    }
}

void print_all_Symtabs(map<string, int> &intcp, map<string, float> &floatcp,
map<string, string> &idp)
{
    cout << "The integer constant pointer is: " << endl;
    for (auto x : intcp)
    {
        cout << x.first << "->" << x.second << endl;
    }
    cout << endl;
    cout << "The float constant pointer is: " << endl;
    for (auto x : floatcp)
    {
        cout << x.first << "->" << x.second << endl;
    }
    cout << endl;

    cout << "The identifier pointer is: " << endl;
    for (auto x : idp)
    {
        cout << x.first << "->" << x.second << endl;
    }
    cout << endl;
}


bool S(vector<vector<string>> Tokensed,int &pin);
bool E(vector<vector<string>> Tokensed,int &pin);
bool E_(vector<vector<string>> Tokensed,int &pin);
bool T(vector<vector<string>> Tokensed,int &pin);
bool T_(vector<vector<string>> Tokensed,int &pin);
bool P(vector<vector<string>> Tokensed,int &pin);
bool P_(vector<vector<string>> Tokensed,int &pin);
bool F(vector<vector<string>> Tokensed,int &pin);



bool F(vector<vector<string>> Tokensed,int &pin){
    cout<<"Entering F  :"<<pin<<endl;

    int thispin=pin;
    if(Tokensed[pin][0]=="10"){
            cout<<"10 obtained  :"<<pin<<endl;
        pin++;
        if(E(Tokensed,pin)){
            if(Tokensed[pin][0]=="11"){
                cout<<"11 obtained  :"<<pin<<endl;
                pin++;
                cout<<"Exiting F  :"<<pin<<endl;
                return true;
            }
        }
    }

    pin = thispin;
```



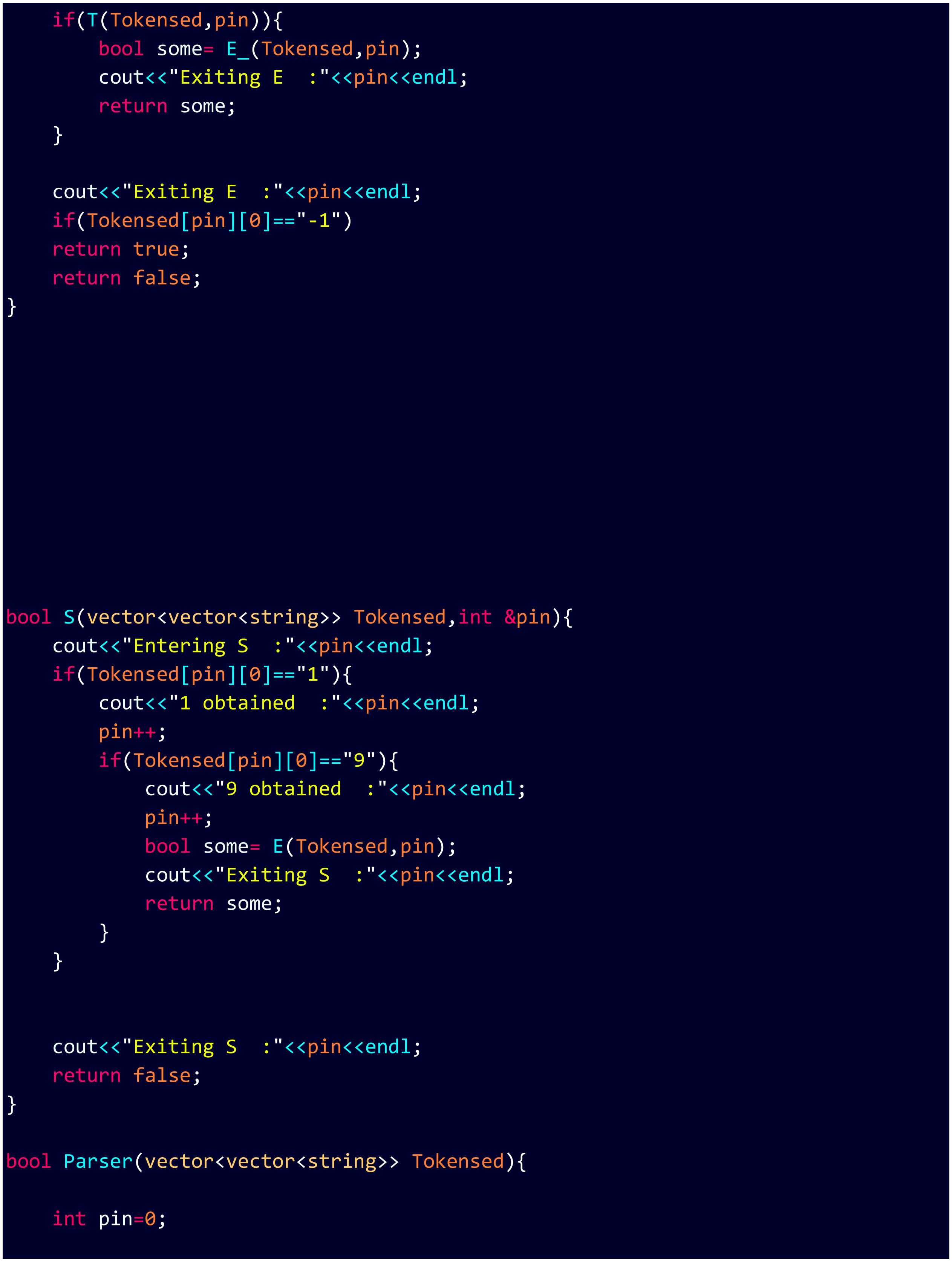
```
    if(Tokensed[pin][0]=="1"){
    cout<<"1 obtained  :"<<pin<<endl;
    pin++;
    cout<<"Exiting F  :"<<pin<<endl;
    return true;
}

        if(Tokensed[pin][0]=="2"){
    cout<<"2 obtained  :"<<pin<<endl;
    pin++;
    cout<<"Exiting F  :"<<pin<<endl;
    return true;
}
        if(Tokensed[pin][0]=="3"){
    cout<<"3 obtained  :"<<pin<<endl;
    pin++;
    cout<<"Exiting F  :"<<pin<<endl;
    return true;
}

    cout<<"Exiting F  :"<<pin<<endl;

    if(Tokensed[pin][0]=="-1")
    return true;

    return false;

}

bool P(vector<vector<string>> Tokensed,int &pin){
    cout<<"Entering P  :"<<pin<<endl;
    if(F(Tokensed,pin)){
        bool some=P_(Tokensed,pin);

        cout<<"Exiting P  :"<<pin<<endl;
        return some;
    }

    cout<<"Exiting P  :"<<pin<<endl;
    if(Tokensed[pin][0]=="-1")
    return true;
    return false;
}
```



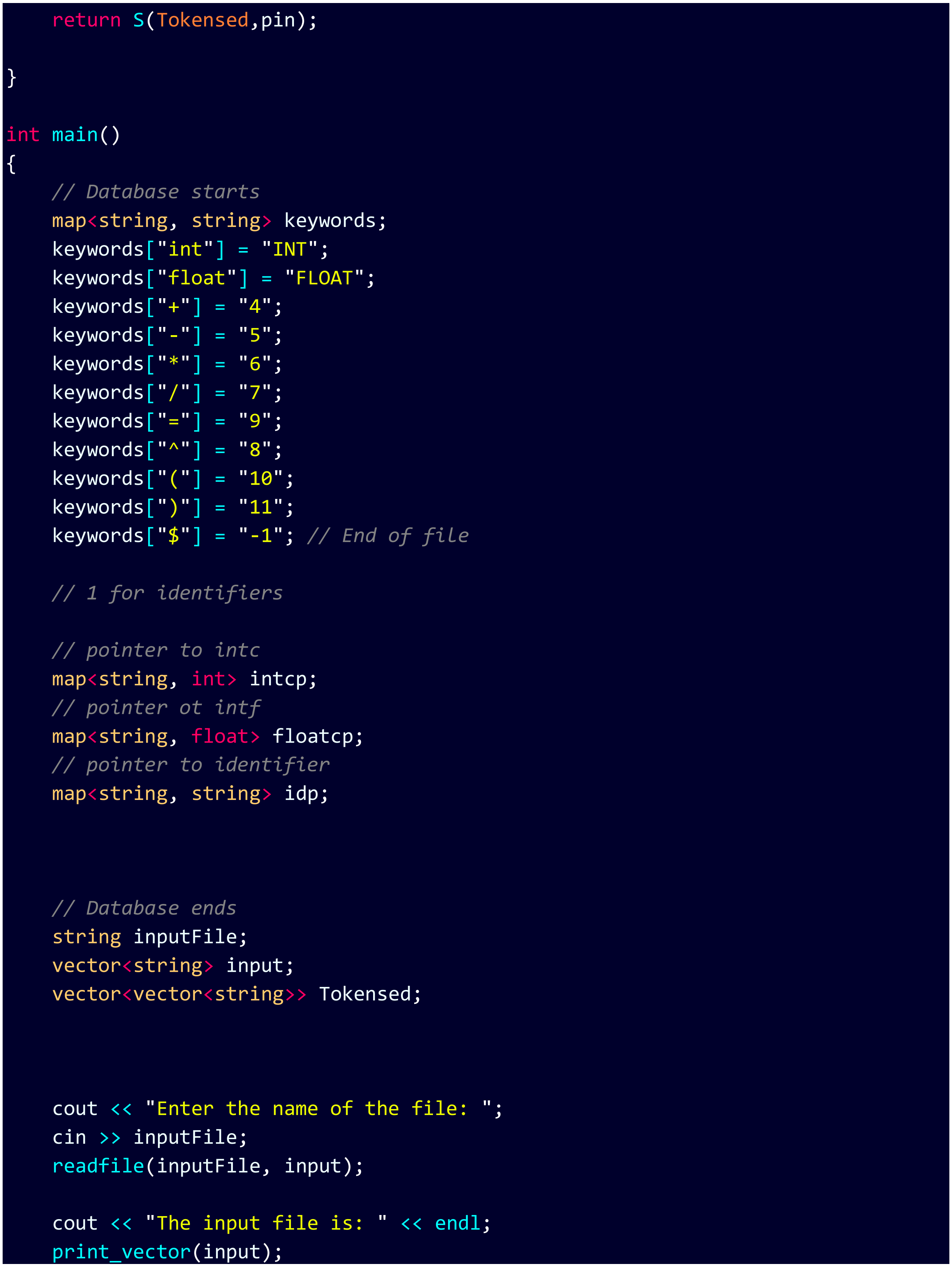
```
bool P_(vector<vector<string>> Tokensed,int &pin){
    cout<<"Entering P_  :"<<pin<<endl;
    int thispin=pin;
    if(Tokensed[pin][0]=="8"){
        cout<<"8 obtained  :"<<pin<<endl;
    pin++;
    bool some=P(Tokensed,pin);
    cout<<"Exiting P_  :"<<pin<<endl;
    return some;
    }

    pin= thispin;
    if(pin!=Tokensed.size()-1){
        cout<<"Exiting P_  :"<<pin<<endl;
        return true;
    }

    cout<<"Exiting P_  :"<<pin<<endl;
    if(Tokensed[pin][0]=="-1")
    return true;
    return false;


}




bool T_(vector<vector<string>> Tokensed,int &pin){

    cout<<"Entering T_  :"<<pin<<endl;

    int thispin=pin;

    if(Tokensed[pin][0]=="6"){
        pin++;
        cout<<"6 obtained  :"<<pin<<endl;
        if(P(Tokensed,pin)){
            bool some= T_(Tokensed,pin);
            cout<<"Exiting T_  :"<<pin<<endl;
            return some;
        }
```



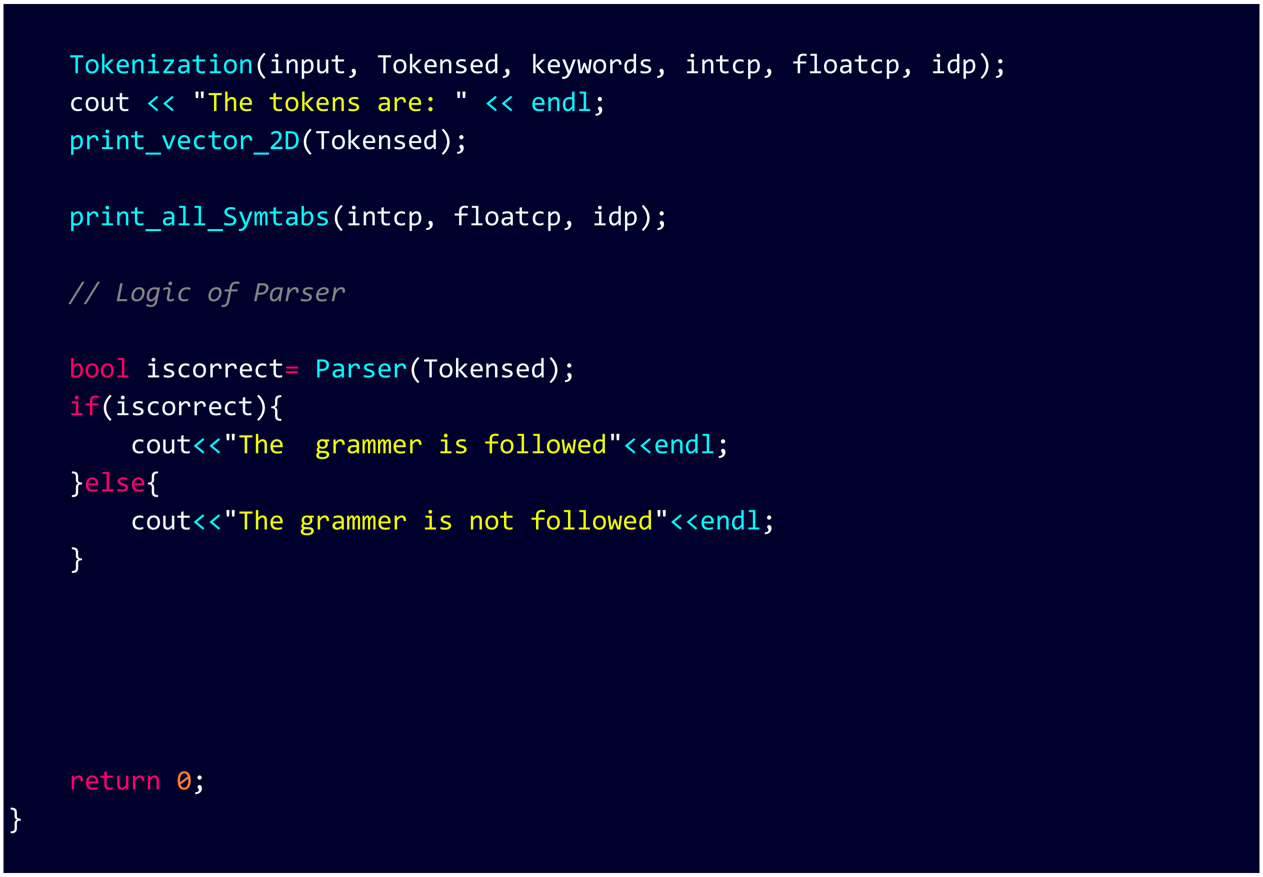
```
    }

    pin = thispin;
    if(Tokensed[pin][0]=="7"){
        pin++;
        cout<<"7 obtained  :"<<pin<<endl;
        if(P(Tokensed,pin)){
            bool some= T_(Tokensed,pin);
            cout<<"Exiting T_  :"<<pin<<endl;
            return some;
        }
    }

    pin = thispin;
    if(pin!=Tokensed.size()-1){
        cout<<"Exiting T_  :"<<pin<<endl;
        return true;
    }

    cout<<"Exiting T_  :"<<pin<<endl;
    if(Tokensed[pin][0]=="-1")
    return true;
    return false;


}


bool T(vector<vector<string>> Tokensed,int &pin){

    cout<<"Entering T  :"<<pin<<endl;
    if(P(Tokensed,pin)){
        bool some= T_(Tokensed,pin);
        cout<<"Exiting T  :"<<pin<<endl;
        return some;
    }

    cout<<"Exiting T  :"<<pin<<endl;
    if(Tokensed[pin][0]=="-1")
    return true;
    return false;
}


bool E_(vector<vector<string>> Tokensed,int &pin){
    cout<<"Entering E_  :"<<pin<<endl;

    int thispin=pin;

    if(Tokensed[pin][0]=="4"){
            cout<<"4 obtained  :"<<pin<<endl;
        pin++;
        if(T(Tokensed,pin)){
            bool some= E_(Tokensed,pin);
            cout<<"Exiting E_  :"<<pin<<endl;
            return some;
        }
    }

    pin=thispin;
    if(Tokensed[pin][0]=="5"){
        cout<<"5 obtained  :"<<pin<<endl;
        pin++;
        if(T(Tokensed,pin)){
            bool some= E_(Tokensed,pin);
            cout<<"Exiting E_  :"<<pin<<endl;
            return some;
        }
    }


    pin=thispin;
    if(pin!=Tokensed.size()-1){
        cout<<"Exiting E_  :"<<pin<<endl;
        return true;
    }

    cout<<"Exiting E_  :"<<pin<<endl;
    if(Tokensed[pin][0]=="-1")
    return true;
    return false;

}



bool E(vector<vector<string>> Tokensed,int &pin){
    cout<<"Entering E  :"<<pin<<endl;
```



```
    if(T(Tokensed,pin)){
        bool some= E_(Tokensed,pin);
        cout<<"Exiting E  :"<<pin<<endl;
        return some;
    }

    cout<<"Exiting E  :"<<pin<<endl;
    if(Tokensed[pin][0]=="-1")
    return true;
    return false;
}









bool S(vector<vector<string>> Tokensed,int &pin){
    cout<<"Entering S  :"<<pin<<endl;
    if(Tokensed[pin][0]=="1"){
        cout<<"1 obtained  :"<<pin<<endl;
        pin++;
        if(Tokensed[pin][0]=="9"){
            cout<<"9 obtained  :"<<pin<<endl;
            pin++;
            bool some= E(Tokensed,pin);
            cout<<"Exiting S  :"<<pin<<endl;
            return some;
        }
    }


    cout<<"Exiting S  :"<<pin<<endl;
    return false;
}

bool Parser(vector<vector<string>> Tokensed){

    int pin=0;
```



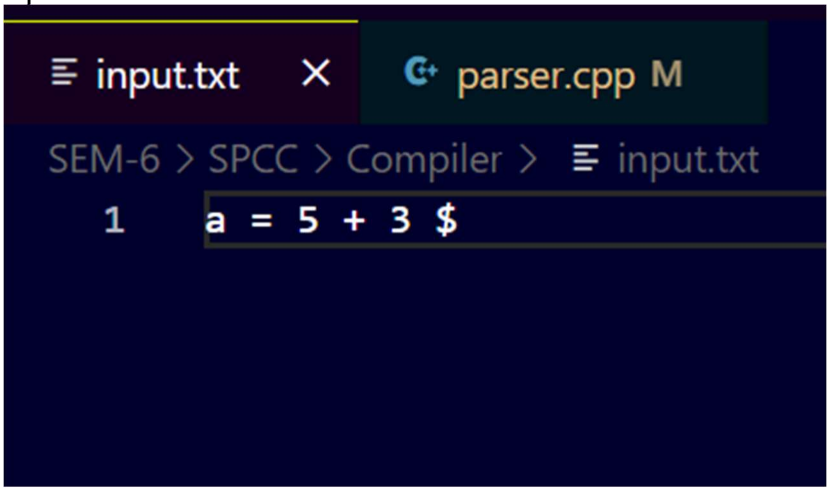
```
    return S(Tokensed,pin);

}

int main()
{
    // Database starts
    map<string, string> keywords;
    keywords["int"] = "INT";
    keywords["float"] = "FLOAT";
    keywords["+"] = "4";
    keywords["-"] = "5";
    keywords["*"] = "6";
    keywords["/"] = "7";
    keywords["="] = "9";
    keywords["^"] = "8";
    keywords["("] = "10";
    keywords[")"] = "11";
    keywords["$"] = "-1"; // End of file

    // 1 for identifiers

    // pointer to intc
    map<string, int> intcp;
    // pointer ot intf
    map<string, float> floatcp;
    // pointer to identifier
    map<string, string> idp;



    // Database ends
    string inputFile;
    vector<string> input;
    vector<vector<string>> Tokensed;



    cout << "Enter the name of the file: ";
    cin >> inputFile;
    readfile(inputFile, input);

    cout << "The input file is: " << endl;
    print_vector(input);
```



```
    Tokenization(input, Tokensed, keywords, intcp, floatcp, idp);
    cout << "The tokens are: " << endl;
    print_vector_2D(Tokensed);

    print_all_Symtabs(intcp, floatcp, idp);

    // Logic of Parser

    bool iscorrect= Parser(Tokensed);
    if(iscorrect){
        cout<<"The  grammer is followed"<<endl;
    }else{
        cout<<"The grammer is not followed"<<endl;
    }




    return 0;
}
```

End Result:

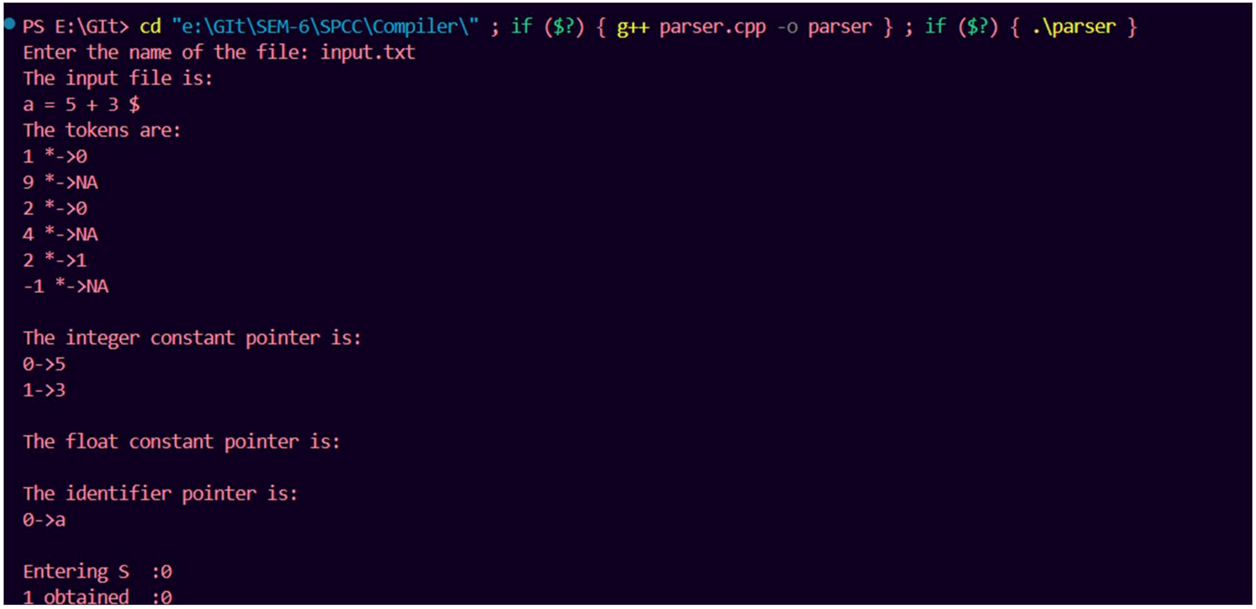
Input file:



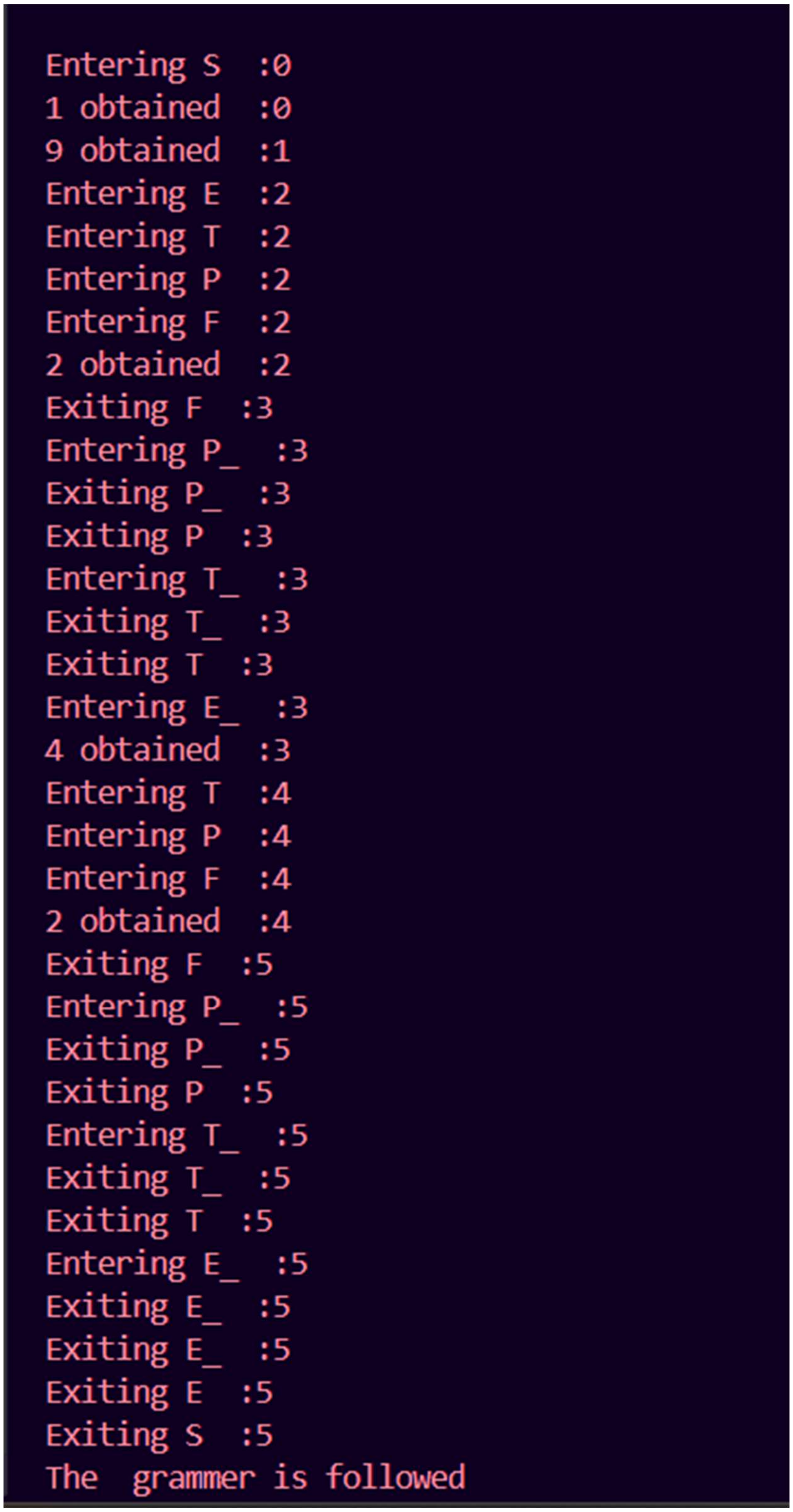
```
a = 5 + 3 $
```

Output :

Accepted=>

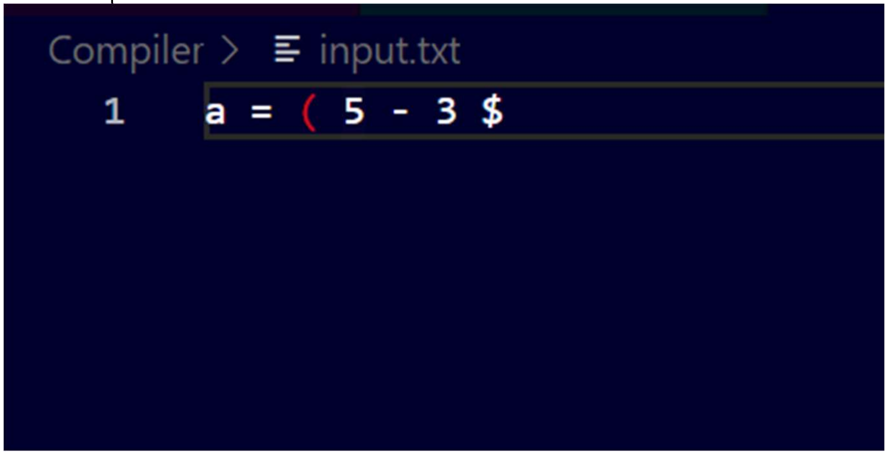


```
PS E:\GIt> cd "e:\GIt\SEM-6\SPCC\Compiler\" ; if ($?) { g++ parser.cpp -o parser } ; if ($?) { .\parser }
Enter the name of the file: input.txt
The input file is:
a = 5 + 3 $
The tokens are:
1 *->0
9 *->NA
2 *->0
4 *->NA
2 *->1
-1 *->NA

The integer constant pointer is:
0->5
1->3

The float constant pointer is:

The identifier pointer is:
0->a

Entering S  :0
1 obtained  :0
```

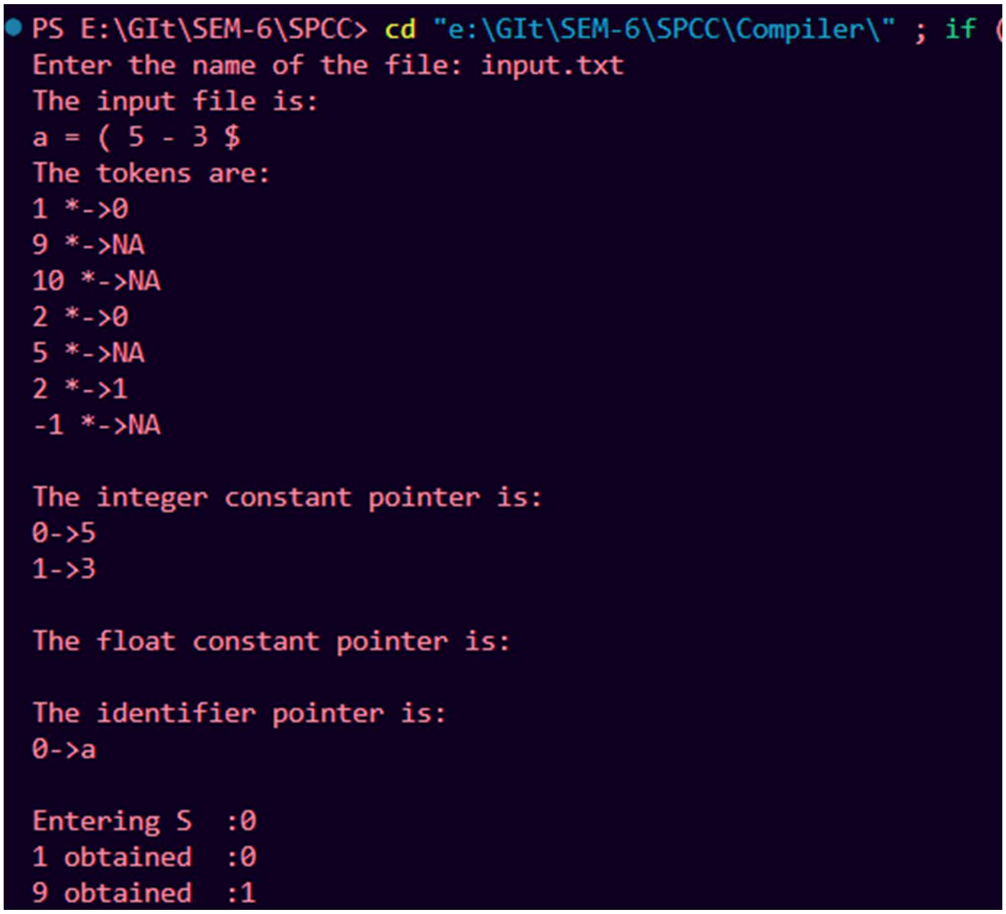


```
Entering S  :0
1 obtained  :0
9 obtained  :1
Entering E  :2
Entering T  :2
Entering P  :2
Entering F  :2
2 obtained  :2
Exiting F  :3
Entering P_  :3
Exiting P_  :3
Exiting P  :3
Entering T_  :3
Exiting T_  :3
Exiting T  :3
Entering E_  :3
4 obtained  :3
Entering T  :4
Entering P  :4
Entering F  :4
2 obtained  :4
Exiting F  :5
Entering P_  :5
Exiting P_  :5
Exiting P  :5
Entering T_  :5
Exiting T_  :5
Exiting T  :5
Entering E_  :5
Exiting E_  :5
Exiting E_  :5
Exiting E  :5
Exiting S  :5
The  grammer is followed
```

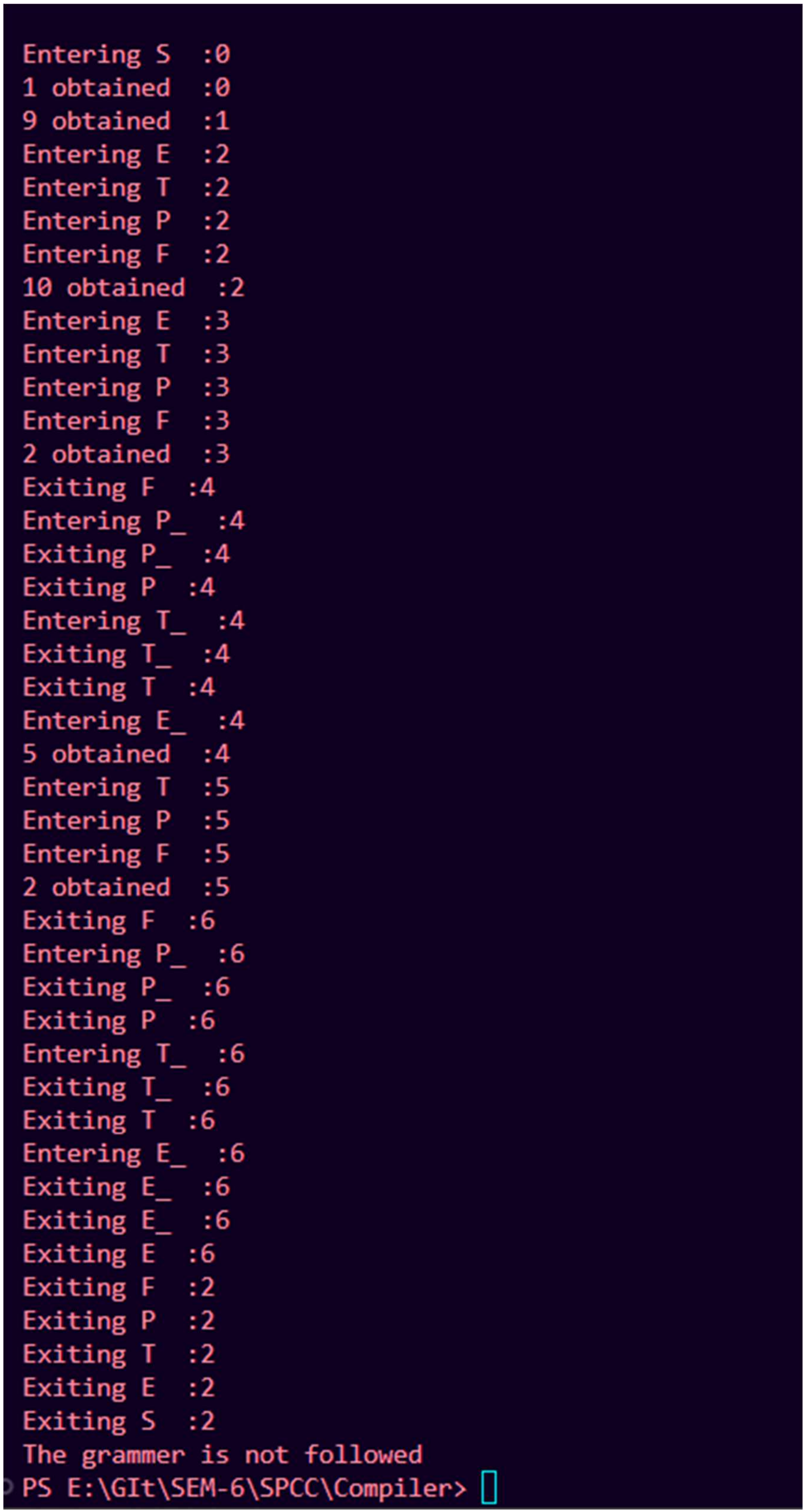
Not Accepted=>



```
a = ( 5 - 3 $
```



```
PS E:\GIt\SEM-6\SPCC> cd "e:\GIt\SEM-6\SPCC\Compiler\" ; if (
Enter the name of the file: input.txt
The input file is:
a = ( 5 - 3 $
The tokens are:
1 *->0
9 *->NA
10 *->NA
2 *->0
5 *->NA
2 *->1
-1 *->NA

The integer constant pointer is:
0->5
1->3

The float constant pointer is:

The identifier pointer is:
0->a

Entering S  :0
1 obtained  :0
9 obtained  :1
```



```
Entering S  :0
1 obtained  :0
9 obtained  :1
Entering E  :2
Entering T  :2
Entering P  :2
Entering F  :2
10 obtained  :2
Entering E  :3
Entering T  :3
Entering P  :3
Entering F  :3
2 obtained  :3
Exiting F  :4
Entering P_  :4
Exiting P_  :4
Exiting P  :4
Entering T_  :4
Exiting T_  :4
Exiting T  :4
Entering E_  :4
5 obtained  :4
Entering T  :5
Entering P  :5
Entering F  :5
2 obtained  :5
Exiting F  :6
Entering P_  :6
Exiting P_  :6
Exiting P  :6
Entering T_  :6
Exiting T_  :6
Exiting T  :6
Entering E_  :6
Exiting E_  :6
Exiting E_  :6
Exiting E  :6
Exiting F  :2
Exiting P  :2
Exiting T  :2
Exiting E  :2
Exiting S  :2
The grammer is not followed
PS E:\GIt\SEM-6\SPCC\Compiler> 
```