College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

October 15, 2020

Exp 10

1 recursive descent parser

1.1 Aim

Construct a recursive descent parser for an expression.

1.2 Theory

Recursive Descent Parser

It is a kind of Top-Down Parser. A top-down parser builds the parse tree from the top to down, starting with the start non-terminal. A Predictive Parser is a special case of Recursive Descent Parser, where no Back Tracking is required. By carefully writing a grammar means eliminating left recursion and left factoring from it, the resulting grammar will be a grammar that can be parsed by a recursive descent parser.

1.3 Algorithm

Algorithm 1: Algorithm for Recursive descent parser

```
One parse method per non - terminal symbol
A non - terminal symbol on the right - hand side of a rewrite rule leads to a call to the
    parse method for that non - terminal
A terminal symbol on the right - hand side of a rewrite rule leads to " consuming " that token
     from the input token string
| in the CFG leads to " if - else " in the parser
```

1.4 Code

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

vector<vector<string>> get_production(unordered_map<char, int> &non_term, int *num)
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    getline(cin, s);
    while (s != "")
    {
        int non_index;
        char left = s[0];
        if (non_term.find(s[0]) == non_term.end())
        {
            non_term[s[0]] = ++non;
            non_index = non;
        }
        else
        {
            non_index = non_term[s[0]];
        }
        string right = s.substr(4, s.size() - 4);
        production[non].push_back(right);
        //cout << "right side " << right << endl;
        getline(cin, s);
    }
    *num = non;
    return production;
}

unordered_set<char> split_string(string s)
```

```
{
    int n = s.size();
    unordered_set<char> result;
    for (int i = 0; i < n; ++i)
    {
        if (s[i] != ' ')
        {
            result.insert(s[i]);
        }
    }
    return result;
}

int method(vector<vector<string>> production, unordered_map<char, int> non_term, string input, int count, string crnt_prod, char E)
{
    int success = 0;
    cout << "using production " << E << "-->" << crnt_prod << endl;
    cout << "inspecting " << count << "th char in input" << endl;
    int size = crnt_prod.size();
    for (int i = 0; i < size; ++i)
    {
        if (non_term.find(crnt_prod[i]) == non_term.end())
        {
            if (crnt_prod[i] != input[count])
            {
                if (crnt_prod[i] == '#')
                {
                    cout << "epsilon found" << endl;
                    continue;
                }

                return -1;
            }
            else
            {
                cout << "matching index " << count << " of input =" << input[count] << " with " << crnt_prod[i] << " in " << crnt_prod << endl;
                success++;
            }
        }
        else
        {
            int fount = 0;
            char temp = crnt_prod[i];
            int non_term_num = non_term[crnt_prod[i]];
            for (int j = 0; j < production[non_term_num].size(); ++j)
            {
                int res = method(production, non_term, input, count + success, production[non_term_num][j], temp);
                if (res == 0)
                {
                    fount = 1;
                    continue;
                }
                if (res != -1)
                {
                    fount = 1;
                    success += res;
                    break;
                }
                else
                {
                    //contines loop
                }
            }
            if (fount == 0)
            {
                return -1;
            }
        }
    }

    return success;
}

```

```
106 bool recursive_descent(vector<vector<string>> production, unordered_map<char, int> non_term,
        char E, string input, int count)
107 {
108     int size = input.size();
109     bool res = false;
110     int non_term_num = non_term[E];
111     for (int i = 0; i < production[non_term_num].size(); ++i)
112     {
113         int ans = method(production, non_term, input, count, production[non_term_num][i], E);
114         if (ans != -1)
115         {
116             if (ans >= size)
117             {
118                 cout << "----------------------------------------" << endl;
119                 cout << "Valid and parsing finished successfully" << endl;
120                 return true;
121             }
122         }
123     }
124     cout << "----------------------------------------" << endl;
125     cout << "Invalid input" << endl;
126     return false;
127 }
128
129 int main()
130 {
131     int non = -1;
132     string s;
133     vector<vector<string>> production(100);
134     cout << "Enter the productions in the form \"S : r\" " << endl;
135     unordered_map<char, int> non_term;
136     production = get_production(non_term, &non);
137     unordered_set<char> terminals;
138     unordered_set<char> non_terminals;
139     cout << "Non-terminals: ";
140     getline(cin, s);
141     non_terminals = split_string(s);
142
143     cout << "Terminals: ";
144     getline(cin, s);
145     terminals = split_string(s);
146
147     char start;
148     cout << "Enter the start symobl: ";
149     cin >> start;
150     cout << "Enter the Expression: ";
151     cin >> s;
152     bool val = recursive_descent(production, non_term, start, s, 0);
153
154     return 0;
155 }
```

1.5 Output

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i+i*i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i+i*i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 2 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
matching index 3 of input =*in i+i*i with * in Y-->*FY
using production F-->(E)
using production F-->i
matching index 4 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
---------------------------------------
Valid and parsing finished successfully
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i++i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i++i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i++i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
----------------------------------------
Invalid input
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i+i*i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
```

```
using production R-->#
using production R-->+TR
matching index 1 of input =+in i+i*i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 2 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
matching index 3 of input =*in i+i*i with * in Y-->*FY
using production F-->(E)
using production F-->i
matching index 4 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
----------------------------------------
Valid and parsing finished successfully
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i++i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i++i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i++i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
----------------------------------------
Invalid input
```

1.6 Result

Implemented the program to construct a recursive descent parser. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.