College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

September 23, 2020

CS431 - Compiler Design Lab · 2020 ·

Exp 8

1 Operator Precedence Parsing

1.1 Aim

Develop an operator precedence parser for a given language

1.2 Theory

Operator Precedence Parser An operator precedence parser is a bottom-up parser that interprets an operator grammar. This parser is only used for operator grammars. Ambiguous grammars are not allowed in any parser except operator precedence parser. There are two methods for determining what precedence relations should hold between a pair of terminals:

1. Use the conventional associativity and precedence of operator.
2. The second method of selecting operator-precedence relations is first to construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse trees.

- This parser relies on the following three precedence relations: $\lessdot, \doteq, \gtrdot$
- a $\lessdot$ b This means a "yields precedence to" b.
- a $\gtrdot$ b This means a "takes precedence over" b.
- a $\doteq$ b This means a "has same precedence as" b.

1.3 Algorithm

Algorithm 1: Algorithm for precedence parsing

```
if ( a is $ and b is $ )
    return
else
    if a . > b or a =. b then
        push a onto the stack
        advance ip to the next input symbol
    else if a <. b then
        repeat
        c <- pop the stack
        until ( c . > stack - top )
    else error
end
```

1.4 Code

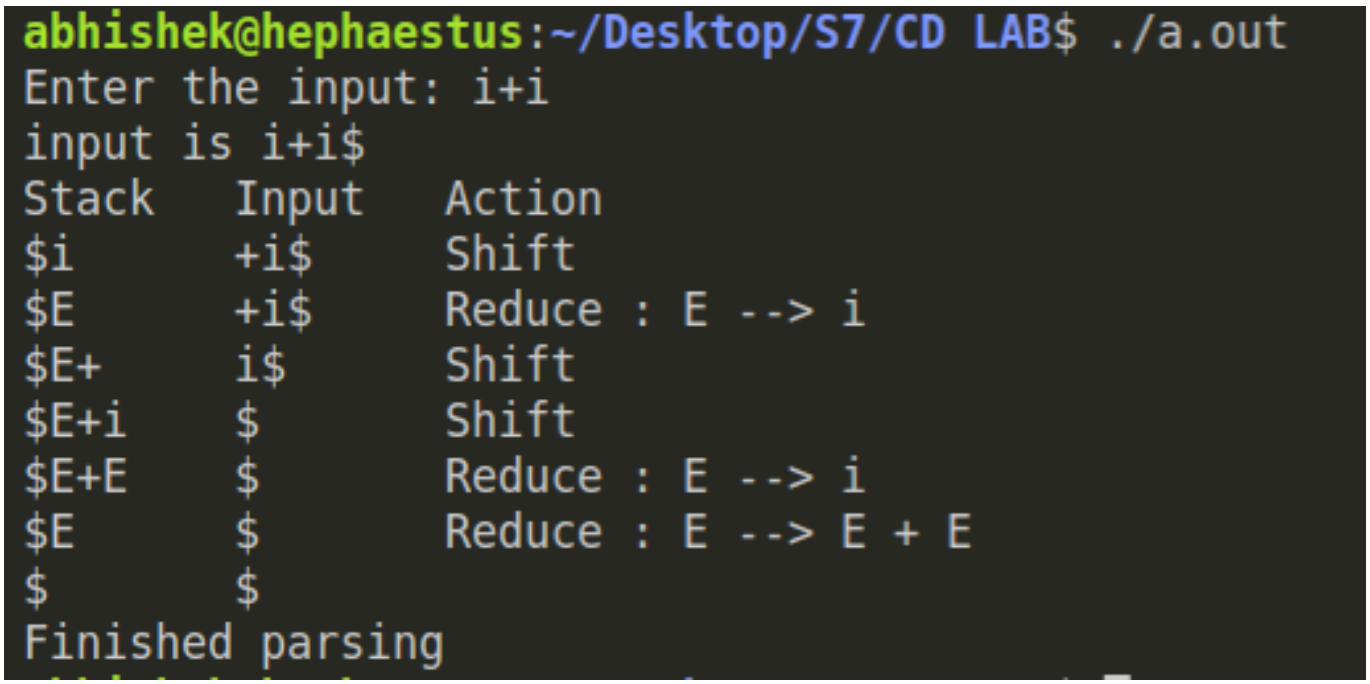
```
#include <bits/stdc++.h>
using namespace std;
void set_precedence(unordered_map<char, int> &precedence)
{
    precedence['$'] = 0;
    precedence['('] = 0;
    precedence['E'] = 1;
    precedence['+'] = 3;
    precedence['*'] = 4;
    precedence[')'] = 5;
    precedence['i'] = 5;
}
void print_stack(stack<char> check)
{
```

```
    string s = "";
    while (!check.empty())
    {
        s = check.top() + s;
        check.pop();
    }
    cout << s;
}
void print_string(string s, int n)
{
    int size = s.size();
    for (int i = n; i < size; ++i)
    {
        cout << s[i];
    }
}
int main()
{
    unordered_map<char, int> precedence;
    set_precedence(precedence);
    cout << "Enter the input: ";
    string s;
    stack<char> check;
    int ip = 0;
    check.push('$');
    cin >> s;
    s += "$";
    cout << "input is " << s << endl;
    cout << "Stack\tInput\tAction" << endl;
    while (true)
    {
        //cout << "in while loop" << endl;
        string action;
        if (s[ip] == '$' && check.top() == '$')
        {
            cout << "Finished parsing" << endl;
            break;
        }
        if (check.empty() || ip >= s.size())
        {
            cout << "Parsing Completed" << endl;
            break;
        }
        if (s[ip] == '(' || precedence[s[ip]] >= precedence[check.top()]) //Push into stack
        {
            //cout << "inside the shifft part" << endl;
            check.push(s[ip]);
            ip++;
            action = "Shift";
        }
        else
        {
            string temp = "";
            while (precedence[s[ip]] < precedence[check.top()])
            {
                char top = check.top();
                temp = top + temp;
                check.pop();
                if (top == 'i')
                {
                    break;
                }
            }
            if (temp == "i")
            {
                action = "Reduce : E --> i";
                check.push('E');
            }
            else if (temp == "E+E")
            {
                action = "Reduce : E --> E + E ";
                check.push('E');
            }
            else if (temp == "E*E")
            {
                action = "Reduce : E --> E * E ";
```

```
                check.push('E');
            }
            else if (temp == "(E)")
            {
                action = "Reduce : E --> ( E ) ";
                check.push('E');
            }
            else if (temp == "E")
            {
                //nothing
            }
            else
            {
                cout << "unexpected  condition " << temp << endl;
            }
        }
        //cout << "endl of loop" << endl;
        print_stack(check);
        //cout << "stack printing finished " << endl;
        cout << "\t";
        print_string(s, ip);
        cout << "\t";
        cout << action << endl;
    }

    return 0;
}
```

1.5 Output




```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the input: i+i
input is i+i$
Stack   Input   Action
$i      +i$     Shift
$E      +i$     Reduce : E --> i
$E+     i$      Shift
$E+i    $       Shift
$E+E    $       Reduce : E --> i
$E      $       Reduce : E --> E + E
$       $
Finished parsing
```

1.6 Result

Implemented the program to do precedence parsing. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.