Experiment – 7

AIM- Write a program to implement LL(1) predictive parser

Description - LL parser (Left-to-right, Leftmost derivation) is a top-down parser for a subset of context-free languages. It parses the input from Left to right, performing Leftmost derivation of the sentence.

LL parser consists of:

1. input buffer (source code)
2. stack
3. parsing table (state machine)

Parsing table: Table is used to get the next production number to apply, based on current symbol from the buffer, and the symbol (terminal or non-terminal) on top of the stack.

- Rows in the table are non-terminals
- Columns are terminals

Parsing algorithm:

- if the top of the stack is *terminal*, and matches current symbol in buffer, then just discard it from the stack, and move cursor further. (if doesn't match -- parse error).
- Else (it must be a non-terminal), replace it with an alternative production, corresponding to the production number.(if no production -- parse error).

$ - is a special symbol used to mark bottom of the stack and end of the buffer.

S - is a start symbol.

Code -

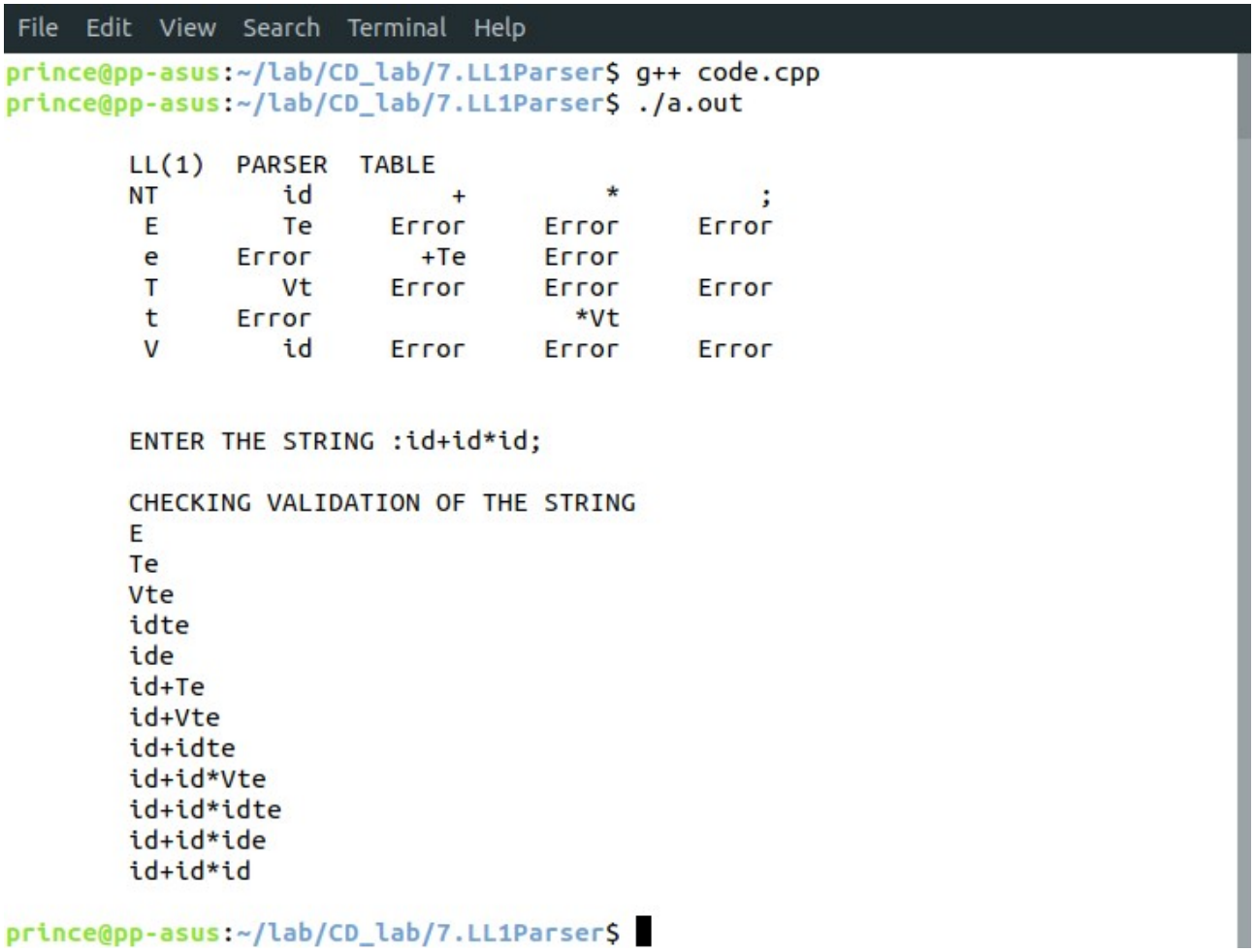
```
#include <bits/stdc++.h>

using namespace std;

int main(){
    int i = 0, j = 0, k = 0, m = 0, n = 0, o = 0, o1 = 0,
        var = 0, l = 0, f = 0, c = 0, f1 = 0;
    char str[30], str1[40] = "E", temp[20],
                temp1[20], temp2[20], tt[20], t3[20];
    temp1[0] = '\0';
    temp2[0] = '\0';
    char t[10];
    char array[6][5][10] = {
        "NT", "id", "+", "*", ";",
        "E", "Te", "Error", "Error", "Error",
        "e", "Error", "+Te", "Error", "\0",
        "T", "Vt", "Error", "Error", "Error",
        "t", "Error", "\0", "*Vt", "\0",
        "V", "id", "Error", "Error", "Error"};
    cout << "\n\tLL(1)  PARSER  TABLE \n";
    for (i = 0; i < 6; i++){
        for (j = 0; j < 5; j++){
            cout.setf(ios::right);
            cout.width(10);
            cout << array[i][j];
        }
        cout << endl;
    }
    cout << endl;
    cout << "\n\tENTER THE STRING :";
    cin >> str;
    if (str[strlen(str) - 1] != ';'){
        cout << "END OF STRING MARKER SHOULD BE ';'";
        exit(1);
    }
    cout << "\n\tCHECKING VALIDATION OF THE STRING ";
    cout << "\n\t" << str1;
    i = 0;

    while (i < strlen(str)){
    again:
```

```
    if (str[i] == ' ' && i < strlen(str)){
        cout << "\n\tSPACES IS NOT ALLOWED IN SOURSE STRING ";
        exit(1);
    }
    temp[k] = str[i];
    temp[k + 1] = '\0';
    f1 = 0;
again1:
    if (i >= strlen(str)){
        exit(1);
    }
    for (int l = 1; l <= 4; l++){
        if (strcmp(temp, array[0][l]) == 0){
            f1 = 1;
            m = 0, o = 0, var = 0, o1 = 0;
            temp1[0] = '\0';
            temp2[0] = '\0';
            int len = strlen(str1);
            while (m < strlen(str1) && m < strlen(str)){
                if (str1[m] == str[m]){
                    var = m + 1;
                    temp2[o1] = str1[m];
                    m++;
                    o1++;
                }else{
                    if ((m + 1) < strlen(str1)){
                        m++;
                        temp1[o] = str1[m];
                        o++;
                    }else{
                        m++;
                    }
                }
            }
            temp2[o1] = '\0';
            temp1[o] = '\0';
            t[0] = str1[var];
            t[1] = '\0';
            for (n = 1; n <= 5; n++){
                if (strcmp(array[n][0], t) == 0){
                    break;
                }
```

```
}
strcpy(str1, temp2);
strcat(str1, array[n][l]);
strcat(str1, temp1);
cout << "\n\t" << str1;

if (array[n][l][0] == '\0'){
   if (i == (strlen(str) - 1)){
      int len = strlen(str1);
      str1[len - 1] = '\0';
      cout << "\n\t" << str1;
      cout << "\n\n\tENTERED STRING IS VALID";
      exit(1);
   }
   temp1[0] = '\0';
   temp2[0] = '\0';
   t[0] = '\0';
   goto again1;
}
if (strcmp(array[n][l], "Error") == 0){
   cout << "\n\tERROR IN YOUR SOURCE STRING";
   exit(1);
}
tt[0] = '\0';
strcpy(tt, array[n][l]);
t3[0] = '\0';
f = 0;
for (c = 0; c < strlen(tt); c++){
   t3[c] = tt[c];
   t3[c + 1] = '\0';
   if (strcmp(t3, temp) == 0){
      f = 0;
      break;
   }
   else{
      f = 1;
   }
}

if (f == 0){
   temp[0] = '\0';
   temp1[0] = '\0';
```

```
                temp2[0] = '\0';
                t[0] = '\0';
                i++;
                k = 0;
                goto again;
             }else{
                temp1[0] = '\0';
                temp2[0] = '\0';
                t[0] = '\0';
                goto again1;
             }
          }
       }
       i++;
       k++;
    }
    if (f1 == 0)
       cout << "\nENTERED STRING IS INVALID\n";
    else
       cout << "\n\n\tENTERED STRING IS VALID\n";

    cout << endl << endl;
    return 0;
}
```

Output –



Learnings – Using LL1 parser we can analyse any code, given proper grammer and codes. It is widely used in Syntax Analysers.