Program 7:

WAP to implement LL(1) string checking process where a string, given by the user through the console, is checked against an LL1 table, given through a file.

```cpp
#include <iostream>
#include <stdio.h>
#include <vector>
#include <map>
#include <set>
#include <fstream>
#include <algorithm>

using namespace std;

int Read(char *str)
{
 int i = 0;
 while (1)
 {
  str[i] = getchar();
  if (str[i] == '\n' || str[i] == '\r')
  {
   str[i] = '\0';
   return i;
  }
  i++;
 }
}

template <typename T>
void printSet(set<T> &set1)
{
 for (auto i = set1.begin(); i != set1.end(); ++i)
  cout << *i << " ";
 cout << endl;
}

template <typename T>
void printVector(vector<T> &set1)
{
 for (auto i = set1.begin(); i != set1.end(); ++i)
  cout << *i << " ";
}

template <typename T>
int findInVector(vector<T> &vec, T elementToFind)
{
 int i = 0;
 for (auto &element : vec)
 {
  // printf("%c", element);
  if (element == elementToFind)
   return i;
  i++;
 }
 return -1;
```

```
}

void printRule(vector<vector<char>> CFG2D, int ruleNumber)
{
 auto rule = CFG2D[ruleNumber].begin();
 printf("%2d : ", ruleNumber);
 cout << *rule << " --> ";
 ++rule;
 for (; rule != CFG2D[ruleNumber].end(); ++rule)
  cout << *rule;
}
int main()
{
 FILE *filePointer = NULL;
 char ch1, ch2;
 int tempInt;

 filePointer = fopen("LL1.txt", "r");
 if (filePointer == NULL)
 {
  printf("Unable to open LL1.txt\n");
  return 1;
 }

 char startSymbol;
 vector<vector<char>> CFG2D;
 vector<char> tempVector;
 set<char> nonTerminals;
 set<char> terminals;
 map<char, set<char>> firstOf;
 map<char, set<char>> followOf;

 vector<char> LLParseTableColumn;
 vector<char> LLParseTableRow;
 vector<vector<int>> LLParseTable;
 vector<int> tempIntVector;

 fscanf(filePointer, "%c%c", &startSymbol, &ch1);
 printf("Start symbol : %c\n", startSymbol);

 while (fscanf(filePointer, "%c", &ch1) != EOF)
 {
  if (ch1 == '@')
  {
   do
   {
    fscanf(filePointer, "%c", &ch1);
   } while (ch1 != '\n');
   break;
  }
  if (ch1 == '\n')
```

```
 {
  CFG2D.push_back(tempVector);
  tempVector.clear();
  continue;
 }

 tempVector.push_back(ch1);
 if (ch1 >= 'A' && ch1 <= 'Z')
  nonTerminals.insert(ch1);
 else if (ch1 == '#')
  continue;
 else
  terminals.insert(ch1);
}
// CFG2D.push_back(tempVector);

// Setting first of map
char firstChar;
while (fscanf(filePointer, "%c", &ch1) != EOF)
{
 if (ch1 == '@')
 {
  do
  {
   fscanf(filePointer, "%c", &ch1);
  } while (ch1 != '\n');
  break;
 }
 firstChar = ch1;
 while (fscanf(filePointer, "%c", &ch1) != EOF)
 {
  if (ch1 == '\n')
   break;
  firstOf[firstChar].insert(ch1);
 }
}

// Setting follow of map
char followChar;
while (fscanf(filePointer, "%c", &ch1) != EOF)
{
 if (ch1 == '@')
 {
  do
  {
   fscanf(filePointer, "%c", &ch1);
  } while (ch1 != '\n');
  break;
 }
 followChar = ch1;
 while (fscanf(filePointer, "%c", &ch1) != EOF)
```

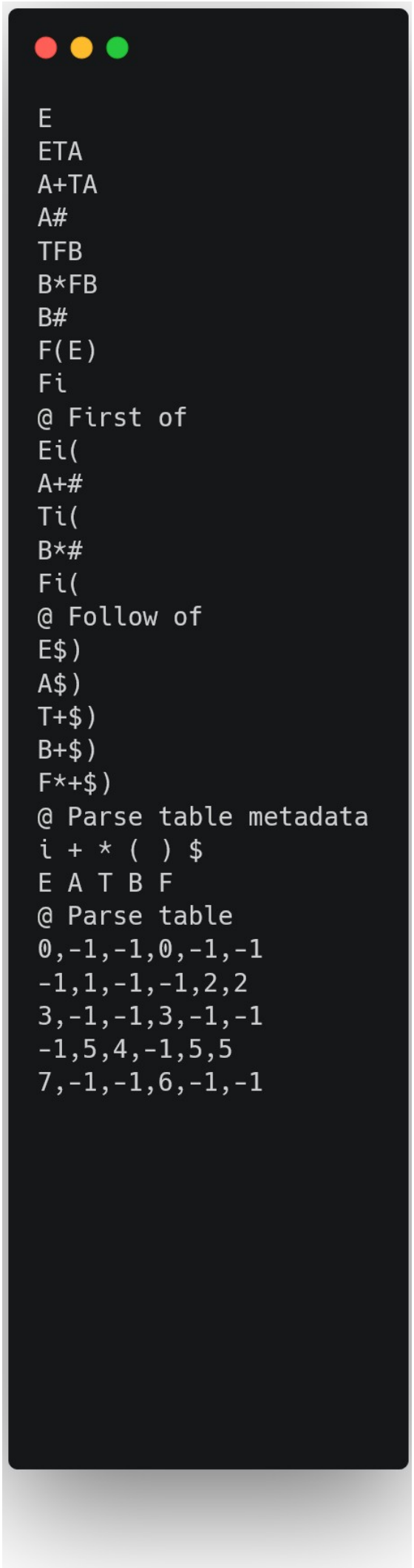
```
 {
  if (ch1 == '\n')
   break;
  followOf[followChar].insert(ch1);
 }
}

// Setting parse table column meta data
while (fscanf(filePointer, "%c%c", &ch1, &ch2) != EOF)
{
 LLParseTableColumn.push_back(ch1);
 if (ch2 == '\n')
  break;
}
// Setting parse table row meta data
while (fscanf(filePointer, "%c%c", &ch1, &ch2) != EOF)
{
 LLParseTableRow.push_back(ch1);
 if (ch2 == '\n')
  break;
}

// Clearing @ line
do
{
 fscanf(filePointer, "%c", &ch1);
} while (ch1 != '\n');

while (fscanf(filePointer, "%d%c", &tempInt, &ch2) != EOF)
{
 tempIntVector.push_back(tempInt);
 if (ch2 == '\n')
 {
  LLParseTable.push_back(tempIntVector);
  tempIntVector.clear();
 }
}
fclose(filePointer);

cout << "Terminals : ";
for (auto terminal : terminals)
 cout << terminal << " ";
cout << endl;
cout << "Non Terminals : ";
for (auto nonTerminal : nonTerminals)
 cout << nonTerminal << " ";
cout << endl;

cout << "CFG : \n";
int index = 0;
for (int i = 0; i < CFG2D.size(); ++i)
```

```
{
 printRule(CFG2D, i);
 cout << endl;
}
cout << endl;

//* Print first of
cout << "First of : \n";
for (auto i = firstOf.begin(); i != firstOf.end(); ++i)
{
 ch1 = (*i).first;
 printf("%c : ", ch1);
 auto &firstOfCh = firstOf[ch1];
 printSet(firstOfCh);
}
cout << endl;

//* Print first of
cout << "Follow of : \n";
for (auto i = followOf.begin(); i != followOf.end(); ++i)
{
 ch1 = (*i).first;
 printf("%c : ", ch1);
 auto &followOfCh = followOf[ch1];
 printSet(followOfCh);
}
cout << endl;

//* Print LL Parsing table
int numberOfColumns = LLParseTableColumn.size();
int numberOfRows = LLParseTableRow.size();

cout << "LL(1) Parsing table :\n";
cout << "    ";
for (int i = 0; i < numberOfColumns; i++)
{
 printf(" %c  ", LLParseTableColumn[i]);
}
cout << endl;
for (int i = 0; i < numberOfRows; i++)
{
 printf(" %c  ", LLParseTableRow[i]);
 for (int j = 0; j < numberOfColumns; j++)
 {
  tempInt = LLParseTable[i][j];
  if (tempInt == -1)
   printf("--  ", tempInt);
  else
   printf("%2d  ", tempInt);
 }
 cout << endl;
```

```
 }
 cout << endl;

 int flag = 0;
 char inputString[100] = {0};
 int inputStringLen = 0;
 int lookAheadIndex = 0;
 int ruleNumber = -1;
 string parseStack;
 while (1)
 {
  printf("\n------------------------\nEnter input string ('#' to
exit) : ");
  inputStringLen = Read(inputString);

  if (inputString[0] == '#')
   break;
  inputString[inputStringLen++] = '$';
  inputString[inputStringLen] = '\0';
  printf("Input string : %s, Input string len : %d\n\n", inputString,
inputStringLen);

  lookAheadIndex = 0;
  parseStack.clear();
  parseStack.push_back('$');
  parseStack.push_back(startSymbol);
  char top;
  int m, n;
  flag = 0;
  printf("     Stack          Input         Action\n");
  while (parseStack.size() > 0)
  {
   if (inputString[lookAheadIndex] == '$')
   {
    if (flag == 2)
    {
     printf("%10s    %10s    -----\n", parseStack.c_str(), inputString +
lookAheadIndex);
     cout << "Given string is accepted.\n";
     break;
    }
    flag = 1;
   }
   flag = 2;
   if (terminals.find(parseStack.back()) != terminals.end())
   {
    if (parseStack.back() != inputString[lookAheadIndex])
    {
     flag = 1;
     break;
    }
```

```
    printf("%10s    %10s    pop(%c)\n", parseStack.c_str(), inputString
+ lookAheadIndex, inputString[lookAheadIndex]);

    parseStack.pop_back();
    lookAheadIndex++;
    flag = 0;
   }
   else if (nonTerminals.find(parseStack.back()) != nonTerminals.end())
   {
    top = parseStack.back();
    m = findInVector(LLParseTableRow, top);
    n = findInVector(LLParseTableColumn, inputString[lookAheadIndex]);
    if (m == -1 || n == -1)
    {
     flag = 1;
     break;
    }
    ruleNumber = LLParseTable[m][n];
    if (ruleNumber == -1)
    {
     flag = 1;
     break;
    }
    printf("%10s    %10s    ", parseStack.c_str(), inputString +
lookAheadIndex);
    // printVector(CFG2D[ruleNumber]);
    printRule(CFG2D, ruleNumber);
    cout << endl;
    parseStack.pop_back();
    for (auto reverseItr = CFG2D[ruleNumber].rbegin(); reverseItr !=
CFG2D[ruleNumber].rend(); reverseItr++)
    {
     parseStack.push_back(*reverseItr);
    }
    parseStack.pop_back();

    if (parseStack.back() == '#')
     parseStack.pop_back();
    flag = 0;
   }
  }
  if (flag == 1)
  {
   cout << "Given string is not accepted.\n";
  }
 }

 printf("\nExiting...\n");

 return 0;
}
```

LL1.txt



```
E
ETA
A+TA
A#
TFB
B*FB
B#
F(E)
Fi
@ First of
Ei(
A+#
Ti(
B*#
Fi(
@ Follow of
E$)
A$)
T+$)
B+$)
F*+$)
@ Parse table metadata
i + * ( ) $
E A T B F
@ Parse table
0,-1,-1,0,-1,-1
-1,1,-1,-1,2,2
3,-1,-1,3,-1,-1
-1,5,4,-1,5,5
7,-1,-1,6,-1,-1
```

Output:

```
Start symbol : E
Terminals : ( ) * + i
Non Terminals : A B E F T
CFG :
 0 : E --> TA
 1 : A --> +TA
 2 : A --> #
 3 : T --> FB
 4 : B --> *FB
 5 : B --> #
 6 : F --> (E)
 7 : F --> i

First of :
A : # +
B : # *
E : ( i
F : ( i
T : ( i

Follow of :
A : $ )
B : $ ) +
E : $ )
F : $ ) * +
T : $ ) +

LL(1) Parsing table :
     i   +   *   (   )   $
 E   0  --  --   0  --  --
 A  --   1  --  --   2   2
 T   3  --  --   3  --  --
 B  --   5   4  --   5   5
 F   7  --  --   6  --  --
```

```
-----------------------
Enter input string ('#' to exit) : (i+i)*i
Input string : (i+i)*i$, Input string len : 8

     Stack          Input         Action
       $E      (i+i)*i$     0 : E --> TA
      $AT      (i+i)*i$     3 : T --> FB
     $ABF      (i+i)*i$     6 : F --> (E)
    $AB)E(     (i+i)*i$    pop(()
     $AB)E      i+i)*i$     0 : E --> TA
    $AB)AT      i+i)*i$     3 : T --> FB
   $AB)ABF      i+i)*i$     7 : F --> i
   $AB)ABi      i+i)*i$    pop(i)
    $AB)AB       +i)*i$     5 : B --> #
     $AB)A       +i)*i$     1 : A --> +TA
   $AB)AT+       +i)*i$    pop(+)
    $AB)AT        i)*i$     3 : T --> FB
   $AB)ABF        i)*i$     7 : F --> i
   $AB)ABi        i)*i$    pop(i)
    $AB)AB         )*i$     5 : B --> #
     $AB)A         )*i$     2 : A --> #
      $AB)         )*i$    pop())
       $AB          *i$     4 : B --> *FB
     $ABF*          *i$    pop(*)
      $ABF           i$     7 : F --> i
      $ABi           i$    pop(i)
       $AB            $     5 : B --> #
        $A            $     2 : A --> #
         $            $     -----
Given string is accepted.

-----------------------
```

```
-----------------------
Enter input string ('#' to exit) : i(i-i)
Input string : i(i-i)$, Input string len : 7

     Stack          Input         Action
        $E       i(i-i)$     0 : E --> TA
       $AT       i(i-i)$     3 : T --> FB
      $ABF       i(i-i)$     7 : F --> i
      $ABi       i(i-i)$    pop(i)
Given string is not accepted.

-----------------------
Enter input string ('#' to exit) : abc
Input string : abc$, Input string len : 4

     Stack          Input         Action
Given string is not accepted.

-----------------------
Enter input string ('#' to exit) :
Input string : $, Input string len : 1

     Stack          Input         Action
Given string is not accepted.

-----------------------
Enter input string ('#' to exit) : #

Exiting...
```