Compiler Design

3170701

Semester:-7
Lab Manual

Submitted By:

DISHEN MAKWANA
180470107035

Department of Computer Engineering
V.V.P. Engineering College

Vision of Institute

To be an exemplary institute, transforming students into competent professionals with human values.

Mission of Institute

- To provide a conducive academic environment for strengthening technical capabilities of the students.
- To strengthen linkage with industries, alumni and professional bodies.
- To organise various co-curricular and extra-curricular activities for overall development of the students.
- To practice good governance and conduct value- based activities for making students responsible citizens.

Vision of Department

Transforming students into globally efficient professionals with moral values.

Mission of Department

- To provide a strong foundation of computer engineering through effective teaching learning process.
- To enhance industry linkage & alumni network for better placement and real-world exposure.
- To provide various opportunities and platforms for all round development of students & encourage them for value-based practices.

Program Educational Objectives PEOs

Graduates will be able to :

- Apply computer engineering theories, principles and skills to meet the challenges of the society.
- Communicate effectively, work collaboratively and manifest professionalism with ethics.
- Exhibit life-long learning attitude and adapt to rapid technological changes in industry.
- Advance their career in industry, pursue higher education or become an entrepreneur.

Course Outcomes

Students will be able to

- Describe the basic concepts of automata theory, formal languages, and Compilation Process. (U)
- Apply suitable parsing strategies for compiling a program. (AP)
- Describe intermediate code generation, Code optimization and Error Recovery mechanisms. (U)
- Understand issues of run time environments and scheduling for instruction level parallelism. (U)

V.V.P. ENGINEERING COLLEGE

RAJKOT

Certificate

This is certify that

Mr. DISHEN MAKWANA, Enrollment No:- 180470107035 Branch:- COMPUTER ENGINEERING, Semester:- 7 has satisfactory completed the course in the subject:- Compiler Design (3170701) within the four walls of V.V.P. ENGINEERING COLLEGE, Rajkot.

Date of Submission :-

Dr. Kamal Sutaria,
Staff In-Charge

Head of Department,
Department of Computer Engineering,
V.V.P. Engineering College

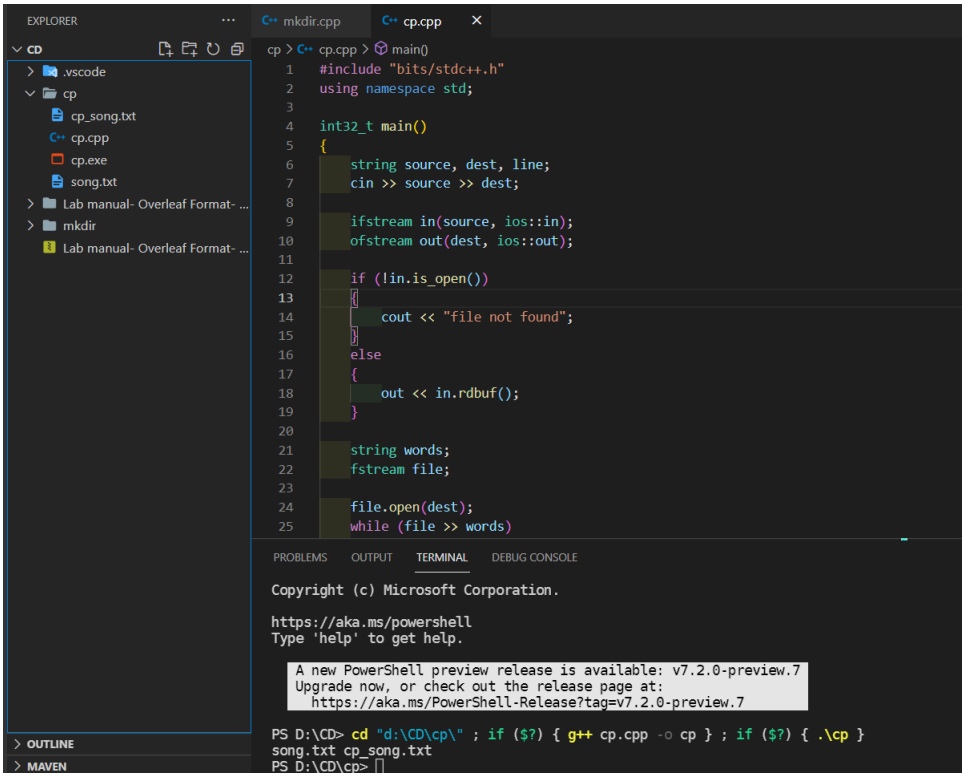
Index

1 Practical:-1 Implement following DOS commands. 1.COPY

Program:

```cpp
#include "bits/stdc++.h"
using namespace std;

int32_t main()
{
    string source, dest, line;
    cin >> source >> dest;

    ifstream in(source, ios::in);
    ofstream out(dest, ios::out);

    if (!in.is_open())
    {
        cout << "file not found";
    }
    else
    {
        out << in.rdbuf();
    }

    string words;
    fstream file;

    file.open(dest);
    while (file >> words)
    {
        cout << words << " ";
    }

    in.close();
    out.close();
    return 0;
}
```

OUTPUT:



Answer the following Questions based on Program.

1. How to pass run time arguments in program?
   Answer: The run time arguments can be passed while executing the program. The arguments need to be passed along with the name of the program.

2. Which error detects by your program if source file doesn't exist?
   Answer: If the source file does not exist, the program will raise "file file not exist!" error.

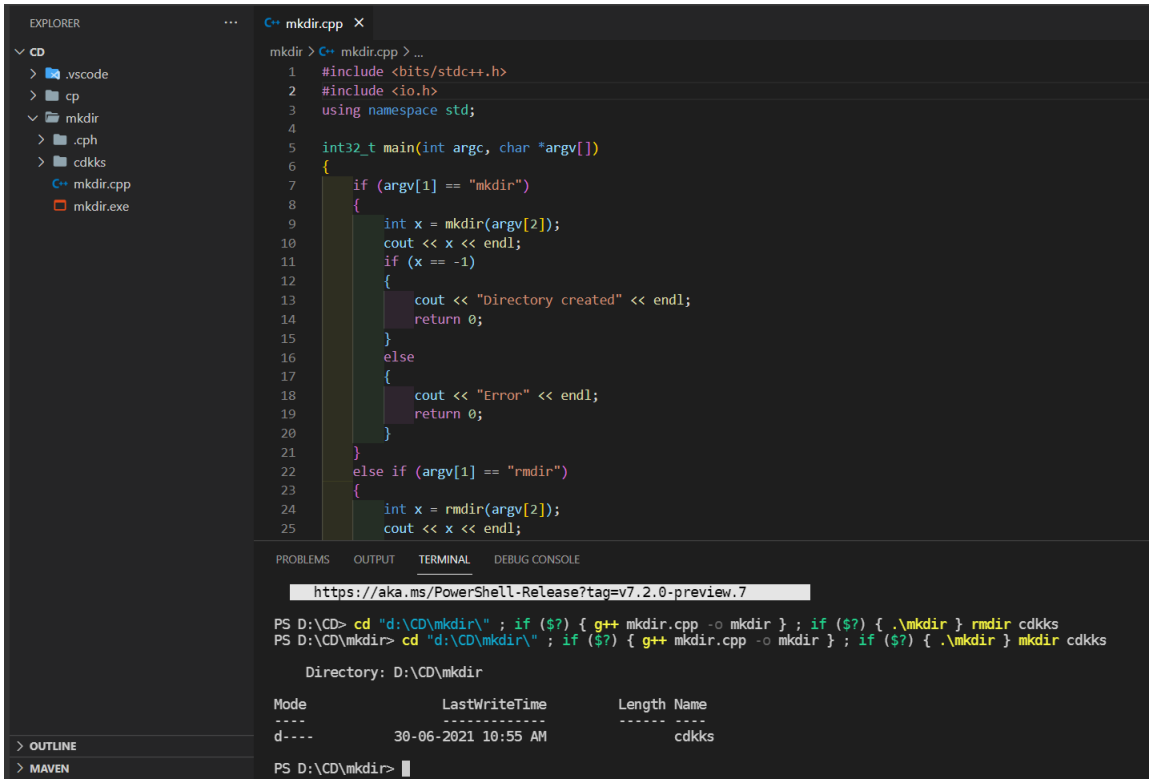
3. What if destination file is not available, is program execute successfully?
   Answer: If the destination file is not available, a new file will be created with the name provided and the program will execute successfully.

2 Practical:-2 Implement following DOS commands. 1.MKDIR and 2.RMDIR

Program:

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

int32_t main(int argc, char *argv[])
{
    if (argv[1] == "mkdir")
    {
        int x = mkdir(argv[2]);
        cout << x << endl;
        if (x == -1)
        {
            cout << "Directory created" << endl;
            return 0;
        }
        else
        {
            cout << "Error" << endl;
            return 0;
        }
    }
    else if (argv[1] == "rmdir")
    {
        int x = rmdir(argv[2]);
        cout << x << endl;
        if (x == -1)
        {
            cout << "Directory deleted" << endl;
            return 0;
        }
        else
        {
            cout << "Error" << endl;
            return 0;
        }
    }
    return 0;
}
```

OUTPUT:



Answer the following Questions based on Program.

1. What is the return type of MKDIR() and RMDIR()?
   Answer: The return type of MKDIR() and RMDIR() is integer. It returns 0utput successful completion and -1 in case of any error.

2. Which error detects by your program if source Directory exist?
   Answer: It display "error : error message" like this.

3. What will be the error for RMDIR(), if directory doesn't available ?
   Answer: If directory isn't available, the error for RMDIR() will be "Error : error message".

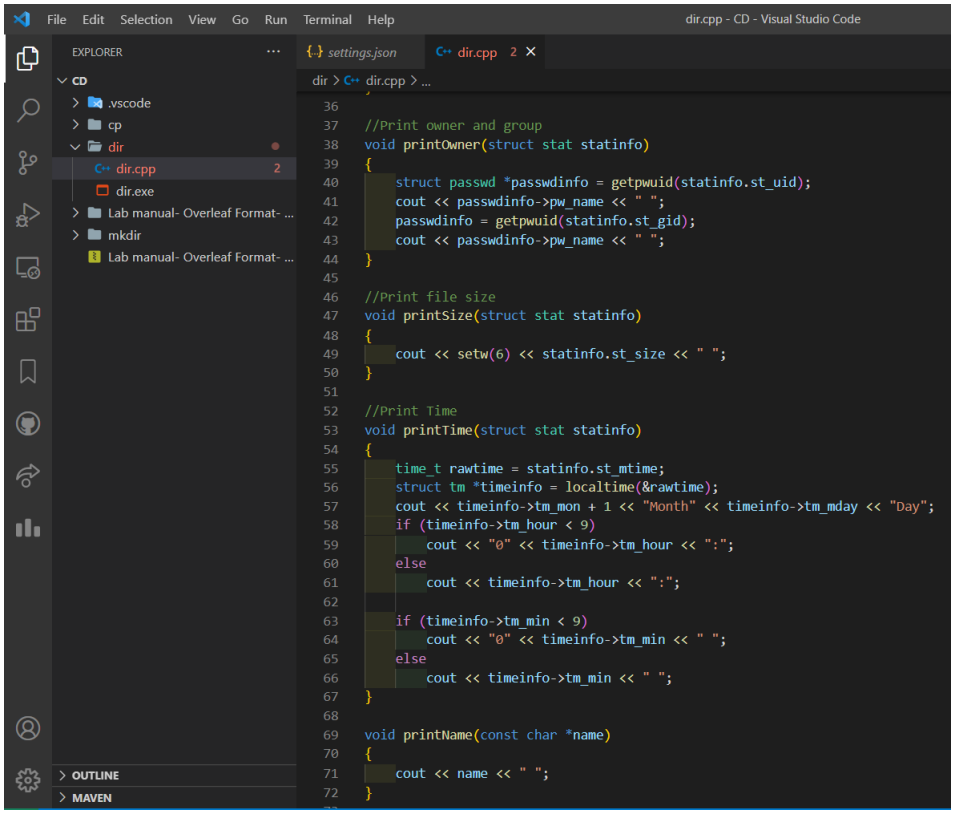
3 Practical:-3 Implement following DOS commands. 3. DIR

Program:

```
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>
#include <dirent.h>
#include <iostream>
#include <iomanip>
#include <pwd.h>
using namespace std;

//Print file permissions
void printPriority(struct stat statinfo)
{
    mode_t mode = statinfo.st_mode;
    //Judging the file type
    cout << (S_ISDIR(mode) ? 'd' : '-');

    //Determine USR permissions
    cout << (mode & S_IRUSR ? 'r' : '-');
    cout << (mode & S_IWUSR ? 'w' : '-');
    cout << (mode & S_IXUSR ? 'x' : '-');

    //Determine GRP permissions
    cout << (mode & S_IRGRP ? 'r' : '-');
    cout << (mode & S_IWGRP ? 'w' : '-');
    cout << (mode & S_IXGRP ? 'x' : '-');

    //Judgment of OTH authority
    cout << (mode & S_IROTH ? 'r' : '-');
    cout << (mode & S_IWOTH ? 'w' : '-');
    cout << (mode & S_IXOTH ? 'x' : '-');

    cout << " ";
}

//Print owner and group
void printOwner(struct stat statinfo)
{
    struct passwd *passwdinfo = getpwuid(statinfo.st_uid);
    cout << passwdinfo->pw_name << " ";
    passwdinfo = getpwuid(statinfo.st_gid);
    cout << passwdinfo->pw_name << " ";
}

//Print file size
void printSize(struct stat statinfo)
{
    cout << setw(6) << statinfo.st_size << " ";
}

//Print Time
void printTime(struct stat statinfo)
{
```

```
55      time_t rawtime = statinfo.st_mtime;
56      struct tm *timeinfo = localtime(&rawtime);
57      cout << timeinfo->tm_mon + 1 << "Month" << timeinfo->tm_mday << "Day";
58      if (timeinfo->tm_hour < 9)
59          cout << "0" << timeinfo->tm_hour << ":";
60      else
61          cout << timeinfo->tm_hour << ":";
62
63      if (timeinfo->tm_min < 9)
64          cout << "0" << timeinfo->tm_min << " ";
65      else
66          cout << timeinfo->tm_min << " ";
67  }
68
69  void printName(const char *name)
70  {
71      cout << name << " ";
72  }
73
74  void ls_l(const char *path)
75  {
76      chdir(path);
77      DIR *dir = opendir(".");
78      struct dirent *dirinfo;
79      while (dirinfo = readdir(dir))
80      {
81          if (!strcmp(dirinfo->d_name, ".") || !strcmp(dirinfo->d_name, ".."))
82              continue;
83          struct stat statinfo;
84          stat(dirinfo->d_name, &statinfo);
85          //begin
86          printPriority(statinfo);
87          printOwner(statinfo);
88          printSize(statinfo);
89          printTime(statinfo);
90          printName(dirinfo->d_name);
91          //end
92          cout << endl;
93      }
94  }
95
96  int main()
97  {
98      ls_l("/home/wwx/Linux/Day04/fileSystem");
99      return 0;
100 }
```

OUTPUT:




```
wwx@VM-0-7-ubuntu:~/Linux/Day04/homework5$ cd "/home/wwx/Linux/Day04/homework5/" && g++ main.cpp -o m
-rw-rw-r-- wwx wwx 3178 May 18th 13:45 main1.cpp
 -rwxr-xr-x wwx wwx 1035 May 14 20:01 file3
 -rw-rw-r-- wwx wwx 2917 May 18th 13:53 main2.cpp
 -rw-rw-r-- wwx wwx 1000 May 14 14:47 file1
 -rw-rw-r-- wwx wwx 11 May 14 20:01 file4
 -rwxr-xr-x wwx wwx 17 May 14 20:19 file6
 -rwxr-xr-x wwx wwx 4192 May 14 14:43 file2
 -rwxrwxr-x wwx wwx 14272 May 18th 13:45 main1
 drwxrwxr-x wwx wwx 4096 May 14 10:06 newdir
```

```
wwx@VM-0-7-ubuntu:~/Linux/Day04/homework5$ ls -l /home/wwx/Linux/Day04/fileSystem
total 52
-rw-rw-r-- 1 wwx wwx  1000 May 14 14:47 file1
-rwxr-xr-x 1 wwx wwx  4192 May 14 14:43 file2
-rwxr-xr-x 1 wwx wwx  1035 May 14 20:01 file3
-rw-rw-r-- 1 wwx wwx    11 May 14 20:01 file4
-rwxr-xr-x 1 wwx wwx    17 May 14 20:19 file6
-rwxrwxr-x 1 wwx wwx 14272 May 18 13:45 main1
-rw-rw-r-- 1 wwx wwx  3178 May 18 13:45 main1.cpp
-rw-rw-r-- 1 wwx wwx  2917 May 18 13:53 main2.cpp
drwxrwxr-x 4 wwx wwx  4096 May 14 10:06 newdir
```

Answer the following Questions based on Program.

1. Why the number of files are displayed more as compare to original?
   Answer: Even if we think that those two commands would give us the same output, it is not actually true. When running "DIR" with the "-l" option, you are also printing a line for the total disk allocation for all files in this directory. As a consequence, you are counting a line that should not be counted, incrementing the final result by one.

2. What is the meaning of **.** and **..** in output list?
   Answer: . means current directory,
   .. means parent directory.

3. What if destination file is not available, will program execute successfully?
   Answer: If destination file is not available then it will generate exception :- "source file is not found".

4 Practical:-4 Implement a program to implement simple lexical analyzer.

Program:

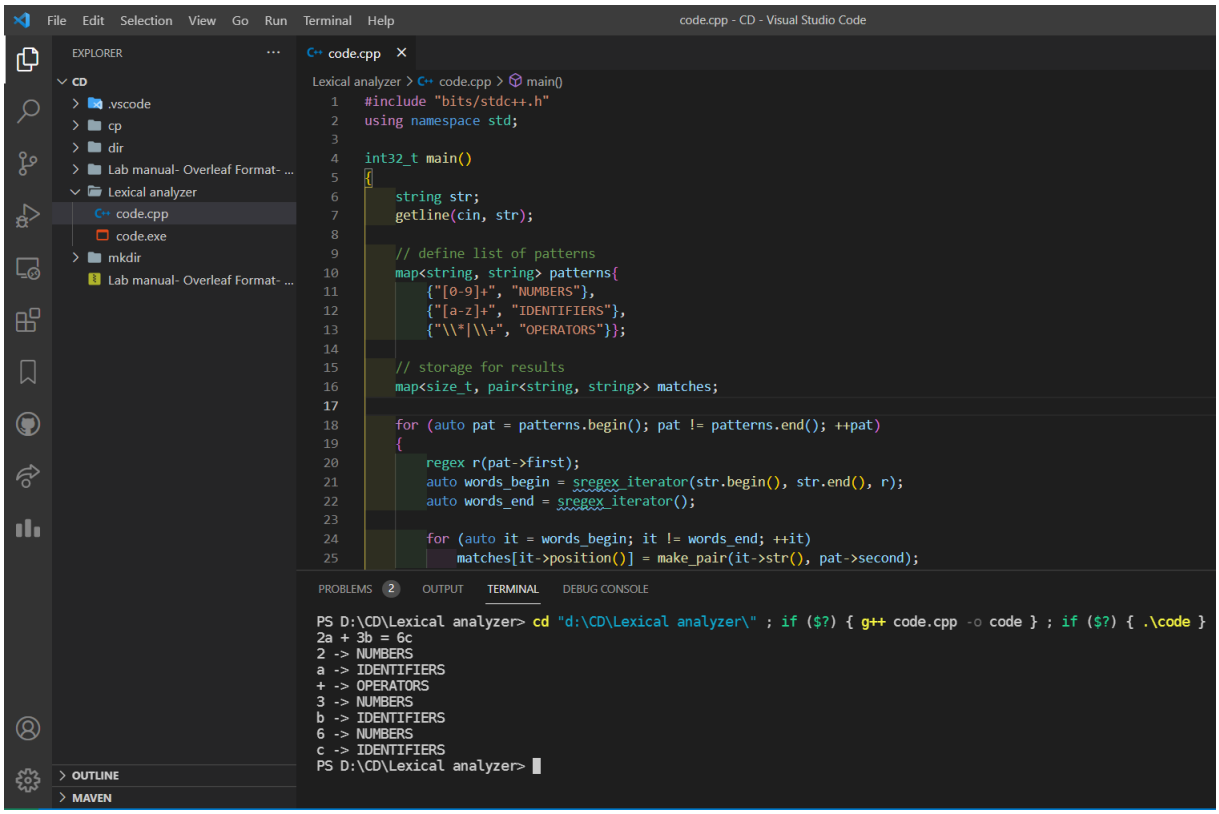
```
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

using namespace std;

bool isPunctuator(char ch) //check if the given character is a
punctuator or not
{
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
        ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
        ch == '[' || ch == ']' || ch == '{' || ch == '}' ||
        ch == '&' || ch == '|')
    {
        return true;
    }
    return false;
}

bool validIdentifier(char *str) //check if the given identifier is valid or not
{
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isPunctuator(str[0]) == true)
    {
        return false;
    } //if first character of string is a digit or a special character, identifier is not valid
    int i, len = strlen(str);
    if (len == 1)
    {
        return true;
    } //if length is one, validation is already completed, hence return true
    else
    {
        for (i = 1; i < len; i++) //identifier cannot contain special characters
        {
            if (isPunctuator(str[i]) == true)
            {
                return false;
            }
        }
    }
    return true;
}

bool isOperator(char ch) //check if the given character is an operator or not
{
    if (ch == '+' || ch == '-' || ch == '*' ||
```

```
53          ch == '/' || ch == '>' || ch == '<' ||
54          ch == '=' || ch == '|' || ch == '&')
55      {
56          return true;
57      }
58      return false;
59  }
60
61  bool isKeyword(char *str) //check if the given substring is a keyword or not
62  {
63      if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") || !strcmp(str, "do") ||
64      !strcmp(str, "break") || !strcmp(str, "continue") || !strcmp(str, "int") || !strcmp(str, "double") ||
65      !strcmp(str, "float") || !strcmp(str, "return") || !strcmp(str, "char") || !strcmp(str, "case") ||
66      !strcmp(str, "long") || !strcmp(str, "short") || !strcmp(str, "typedef") || !strcmp(str, "switch") ||
67      !strcmp(str, "unsigned") || !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") ||
68      !strcmp(str, "sizeof") || !strcmp(str, "long") || !strcmp(str, "volatile") || !strcmp(str, "typedef") ||
69      !strcmp(str, "enum") || !strcmp(str, "const") || !strcmp(str, "union") || !strcmp(str, "extern") ||
70      !strcmp(str, "bool"))
71      {
72          return true;
73      }
74      else
75      {
76          return false;
77      }
78  }
79
80  bool isNumber(char *str) //check if the given substring is a number or not
81  {
82      int i, len = strlen(str), numOfDecimal = 0;
83      if (len == 0)
84      {
85          return false;
86      }
87      for (i = 0; i < len; i++)
88      {
89          if (numOfDecimal > 1 && str[i] == '.')
90          {
91              return false;
92          }
93          else if (numOfDecimal <= 1)
94          {
95              numOfDecimal++;
96          }
97          if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3' && str[i] != '4' && str[i] != '5' &&
98          str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9' || (str[i] == '-' && i > 0))
99          {
100             return false;
101         }
102     }
103     return true;
104 }
105
106 char *subString(string realStr, int l, int r) //extract the required substring from the main string
107 {
108     int i;
109
110     char *str = (char *)malloc(sizeof(char) * (r - l + 2));
111
112     for (i = l; i <= r; i++)
113     {
```

```
114             str[i - l] = realStr[i];
115             str[r - l + 1] = '\0';
116         }
117         return str;
118     }
119
120     void parse(string str) //parse the expression
121     {
122         int left = 0, right = 0;
123         int len = str.length();
124         while (right <= len && left <= right)
125         {
126             if (isPunctuator(str[right]) == false) //if character is a digit or an alphabet
127             {
128                 right++;
129             }
130
131             if (isPunctuator(str[right]) == true && left == right) //if character is a punctuator
132             {
133                 if (isOperator(str[right]) == true)
134                 {
135                     std::cout << str[right] << " -> OPERATOR\n";
136                 }
137                 right++;
138                 left = right;
139             }
140             else if (isPunctuator(str[right]) == true && left != right || (right == len && left != right))
141             //check if parsed substring is a keyword or identifier or number
142             {
143                 char *sub = subString(str, left, right - 1); //extract substring
144
145                 if (isKeyword(sub) == true)
146                 {
147                     cout << sub << " -> KEYWORD\n";
148                 }
149                 else if (isNumber(sub) == true)
150                 {
151                     cout << sub << " -> NUMBER\n";
152                 }
153                 else if (validIdentifier(sub) == true && isPunctuator(str[right - 1]) == false)
154                 {
155                     cout << sub << " -> VALID IDENTIFIER\n";
156                 }
157                 else if (validIdentifier(sub) == false && isPunctuator(str[right - 1]) == false)
158                 {
159                     cout << sub << " -> NOT A VALID IDENTIFIER\n";
160                 }
161
162                 left = right;
163             }
164         }
165         return;
166     }
167
168     int main()
169     {
170         char filename[20];
171         cin >> filename;
172         ifstream fin(filename);
173
174         string str;
```

```
175
176     while (fin)
177     {
178         // Read a Line from File
179         getline(fin, str);
180
181         parse(str);
182         // Print line in Console
183         // cout << str << endl;
184     }
185     return 0;
186 }
```

OUTPUT:



Answer the following Questions based on Program.

1. What is the main function of Lexical Analyzer?
   Answer: The main task of lexical analysis is to read input characters in the code and produce tokens. "Get next token" is a command which is sent from the parser to the lexical analyzer. On receiving this command, the lexical analyzer scans the input until it finds the next token.

2. How many tockens are there in following code?
   printf ("i = %d, &i = %x", i, &i);
   Answer: identifier: 2, character: 4, alphabet: 2,
   total = 8 tokens.

3. List the secondary duties of Lexical Analyzer?
   Answer: The lexical analyzer is the first phase of compiler. Its main task is to read the input characters and produce as output a sequence of tokens that the parser uses for syntax analysis.
   • It is implemented by making lexical analyzer be a subroutine.
   • Upon receiving a "get next token" command from parser, the lexical analyzer reads the input character until it can identify the next token.
   • It may also perform secondary task at user interface.
   • One such task is stripping out from the source program comments and white space in the form of blanks, tabs, and newline characters.
   • Some lexical analyzer are divided into cascade of two phases, the first called scanning and second is "lexical analysis".
   • The scanner is responsible for doing simple task while lexical analysis does the more complex task.

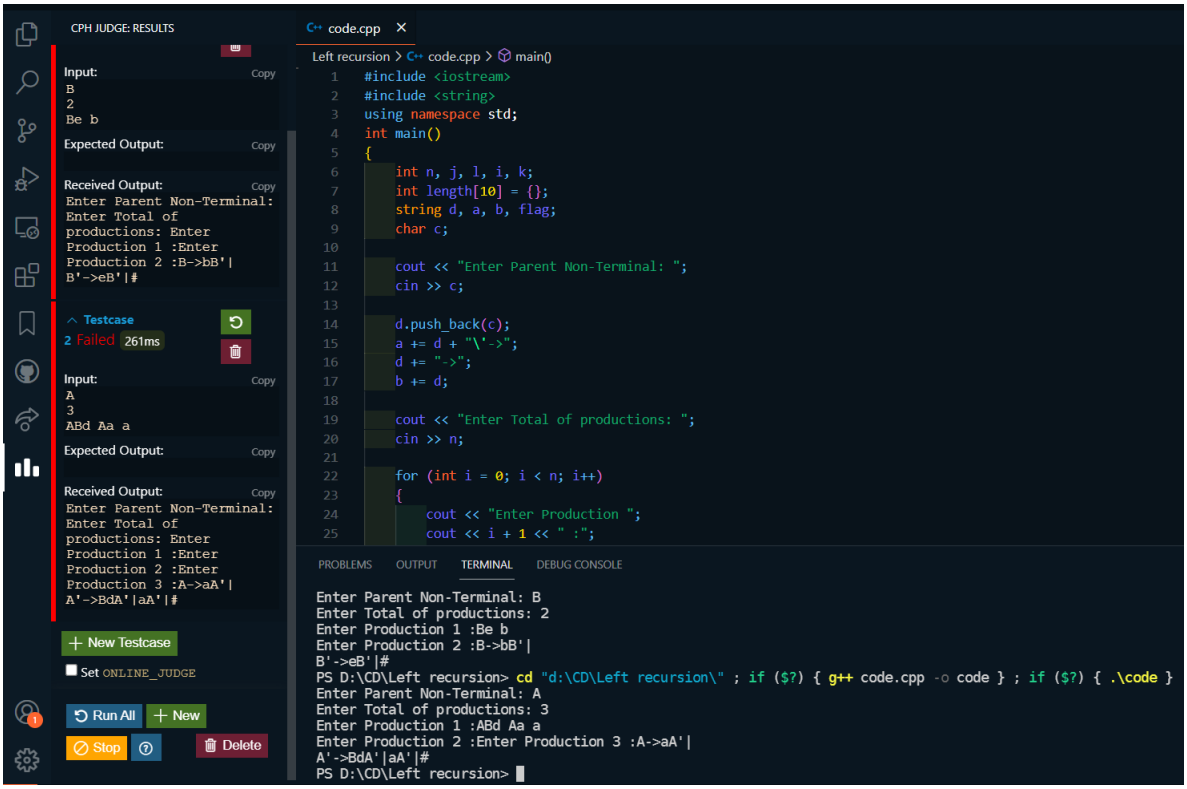
5 Program:-5 Write a program to eliminate left recursion from a grammar.

Program:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int n, j, l, i, k;
    int length[10] = {};
    string d, a, b, flag;
    char c;

    cout << "Enter Parent Non-Terminal: ";
    cin >> c;
    d.push_back(c);
    a += d + "\'->";
    d += "->";
    b += d;
    cout << "Enter Total of productions: ";
    cin >> n;

    for (int i = 0; i < n; i++)
    {
        cout << "Enter Production ";
        cout << i + 1 << " :";
        cin >> flag;
        length[i] = flag.size();
        d += flag;
        if (i != n - 1)
        {
            d += "|";
        }
    }

    for (i = 0, k = 3; i < n; i++)
    {
        if (d[0] != d[k])
        {
            if (d[k] == '#')
            {
                b.push_back(d[0]);
                b += "\'";
            }
            else
            {
                for (j = k; j < k + length[i]; j++)
                {
                    b.push_back(d[j]);
                }
                k = j + 1;
                b.push_back(d[0]);
                b += "\'|";
            }
```

```
        }
        else
        {
            if (d[k] != '#')
            {
                for (l = k + 1; l < k + length[i]; l++)
                {
                    a.push_back(d[l]);
                }
                k = l + 1;
                a.push_back(d[0]);
                a += "\'|";
            }
        }
    }
    a += "#";
    cout << b << endl;
    cout << a << endl;
    return 0;
}
```

OUTPUT:



Answer the following Questions based on Program.

1. Why left Recursion creates a problem in Top Down Parsing?
   Answer: Left recursion often poses problems for parsers, either because it leads them into infinite recursion (as in the case of most top-down parsers) or because they expect rules in a normal form that forbids it.Therefore, a grammar is often preprocessed to eliminate the left recursion.

2. How many types of Left Recursion do you know? List their Names.
   Answer:
   1) Left Recursion
   2) Right Recursion
   3) Indirect Recursion

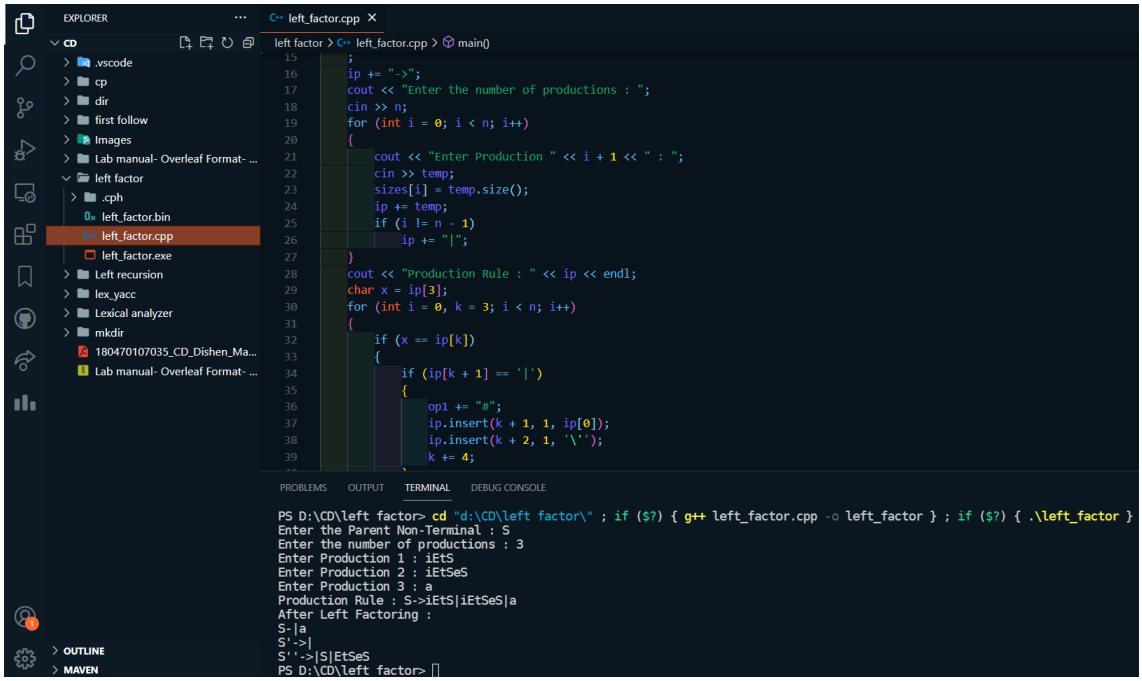
6 Program:-6 Write a program to left factor a grammar.

Program:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string ip, op1, op2, temp;
    int sizes[10] = {};
    char c;
    int n, j, l;
    cout << "Enter the Parent Non-Terminal : ";
    cin >> c;
    ip.push_back(c);
    op1 += ip + "\'->";
    op2 += ip + "\'\'->";
    ;
    ip += "->";
    cout << "Enter the number of productions : ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cout << "Enter Production " << i + 1 << " : ";
        cin >> temp;
        sizes[i] = temp.size();
        ip += temp;
        if (i != n - 1)
            ip += "|";
    }
    cout << "Production Rule : " << ip << endl;
    char x = ip[3];
    for (int i = 0, k = 3; i < n; i++)
    {
        if (x == ip[k])
        {
            if (ip[k + 1] == '|')
            {
                op1 += "#";
                ip.insert(k + 1, 1, ip[0]);
                ip.insert(k + 2, 1, '\'');
                k += 4;
            }
            else
            {
                op1 += "|" + ip.substr(k + 1, sizes[i] - 1);
                ip.erase(k - 1, sizes[i] + 1);
            }
        }
        else
        {
            while (ip[k++] != '|')
                ;
        }
    }
    char y = op1[6];
    for (int i = 0, k = 6; i < n - 1; i++)
```

```
    {
        if (y == op1[k])
        {
            if (op1[k + 1] == '|')
            {
                op2 += "#";
                op1.insert(k + 1, 1, op1[0]);
                op1.insert(k + 2, 2, '\'');
                k += 5;
            }
            else
            {
                temp.clear();
                for (int s = k + 1; s < op1.length(); s++)
                    temp.push_back(op1[s]);
                op2 += "|" + temp;
                op1.erase(k - 1, temp.length() + 2);
            }
        }
    }
    op2.erase(op2.size() - 1);
    cout << "After Left Factoring : " << endl;
    cout << ip << endl;
    cout << op1 << endl;
    cout << op2 << endl;
    return 0;
}
```

OUTPUT:



Answer the following Questions based on Program.

1. Why left Factoring creates a problem in Top Down Parsing?
   Answer: A top down parser generated from the grammar is not efficient as it requires backtracking.

2. Remove Left Recursion from A→ Xyz —Xz
   Answer: A → XA′
   A′ → yz / z

7 Program:-7 Write a program that will find the FIRST SET of the grammar.

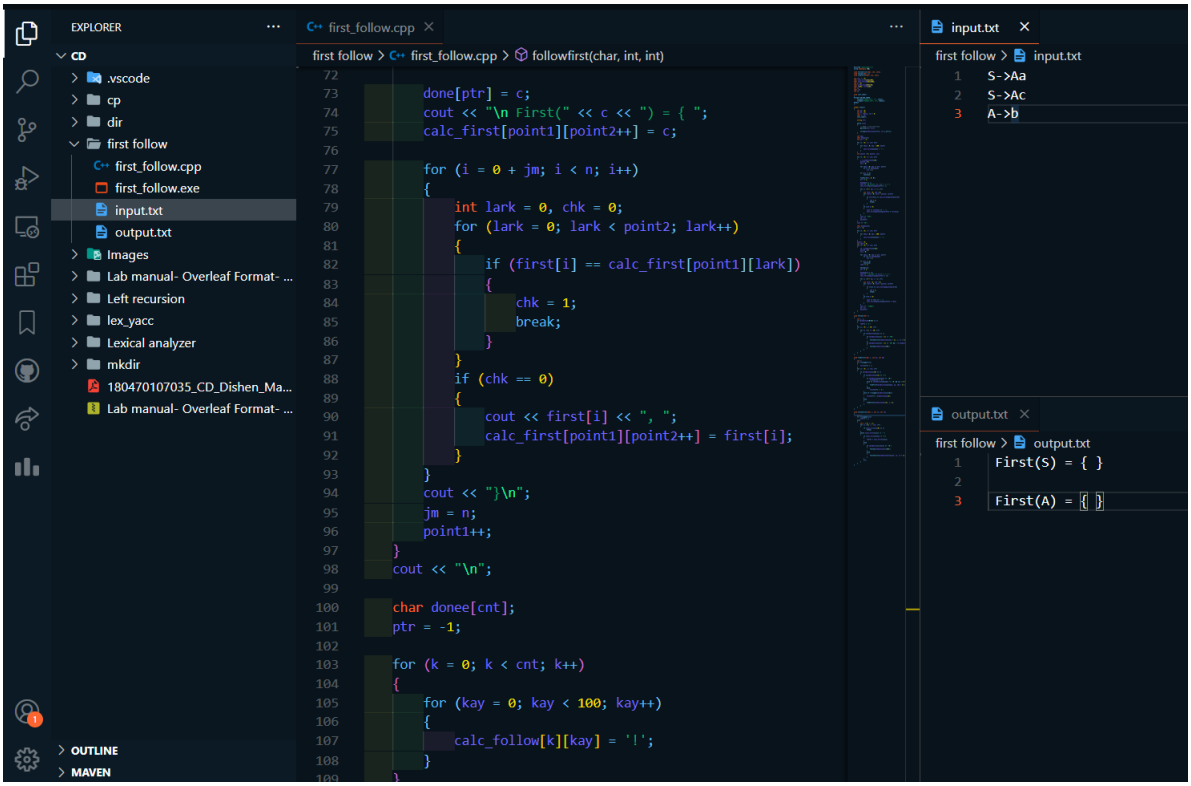
Program:

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

void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);

int cnt, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

void init_code()
{
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
}

int32_t main()
{
    int jm = 0;
    int km = 0;
    int i, choice, cnt = 0;
    char c, ch;
    init_code();
    string str;

    while (cin)
    {
        // Read a Line from File
        getline(cin, str);

        strcpy(production[cnt++], str.c_str());
    }

    int kay;
    char done[cnt];
    int ptr = -1;
    for (k = 0; k < cnt; k++)
    {
        for (kay = 0; kay < 100; kay++)
        {
            calc_first[k][kay] = '!';
        }
    }
```

```
    int point1 = 0, point2, xxx;

    for (k = 0; k < cnt; k++)
    {
        c = production[k][0];
        point2 = 0;
        xxx = 0;
        for (kay = 0; kay <= ptr; kay++)
            if (c == done[kay])
                xxx = 1;

        if (xxx == 1)
            continue;
        findfirst(c, 0, 0);
        ptr += 1;
        done[ptr] = c;
        cout << "\n First(" << c << ") = { ";
        calc_first[point1][point2++] = c;

        for (i = 0 + jm; i < n; i++)
        {
            int lark = 0, chk = 0;
            for (lark = 0; lark < point2; lark++)
            {
                if (first[i] == calc_first[point1][lark])
                {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0)
            {
                cout << first[i] << ", ";
                calc_first[point1][point2++] = first[i];
            }
        }
        cout << "}\n";
        jm = n;
        point1++;
    }
}

void findfirst(char c, int q1, int q2)
{
    int j;
    if (!(isupper(c)))
    {
        first[n++] = c;
    }
    for (j = 0; j < cnt; j++)
    {
        if (production[j][0] == c)
        {
            if (production[j][2] == '#')
            {
                if (production[q1][q2] == '\0')
                    first[n++] = '#';
                else if (production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                {
                    findfirst(production[q1][q2], q1, (q2 + 1));
                }
```

```
                else
                    first[n++] = '#';
            }
            else if (!isupper(production[j][2]))
            {
                first[n++] = production[j][2];
            }
            else
            {
                findfirst(production[j][2], j, 3);
            }
        }
    }
}
```

OUTPUT:



Answer the following Questions based on Program.

1. Write Rules for finding FIRST.
   Answer:

   (a) If X is a terminal then First(X) is just X!

   (b) If there is a Production X → then add to First(X)

   (c) If there is a Production X → Y1Y2..Yk then add first(Y1Y2..Yk) to first(X)

   (d) First(Y1Y2..Yk) is either

      i. First(Y1) (if First(Y1) doesn't contain)

      ii. OR (if First(Y1) does contain) then First (Y1Y2..Yk) is everything in First(Y1) (except for) as well as everything in First(Y2..Yk)

      iii. If First(Y1) First(Y2)..First(Yk) all contain then add to First(Y1Y2..Yk) as well.

8 Program:-8 Write a program that will find the FOLLOW SET of the grammar.

Program:

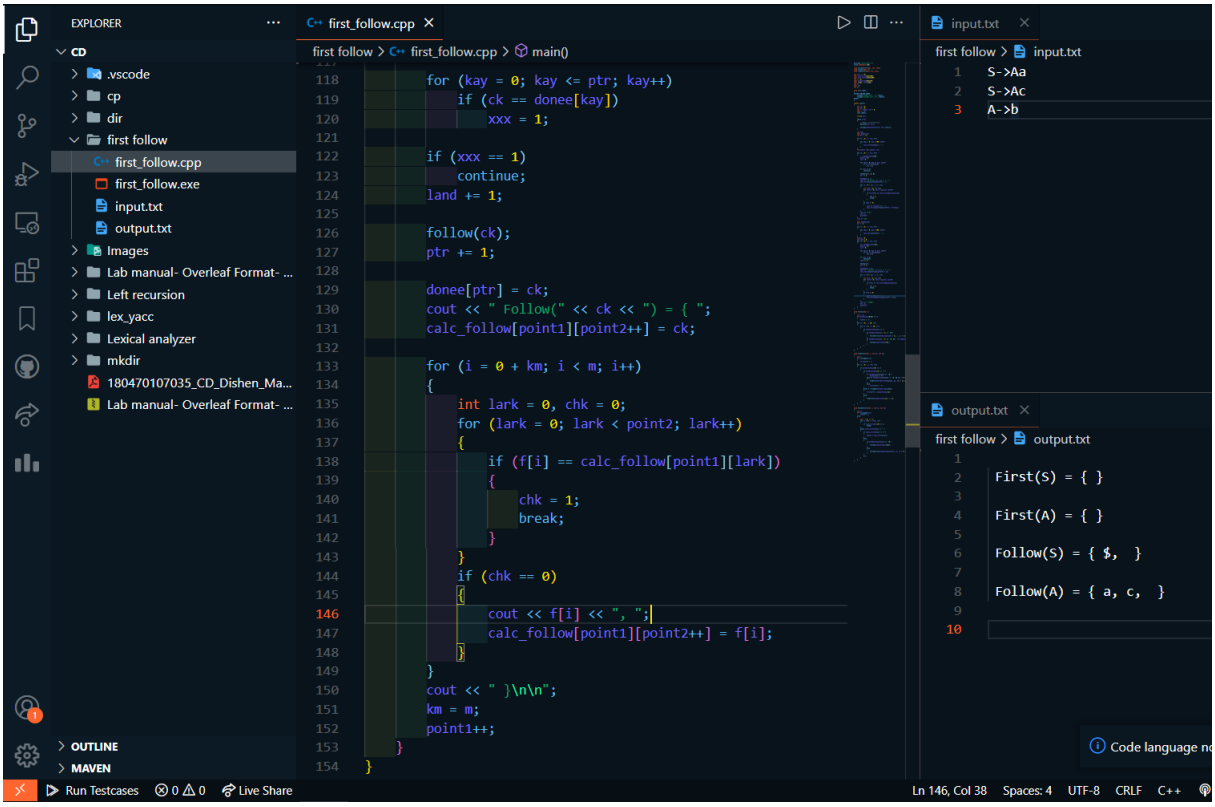
```
#include "bits/stdc++.h"
using namespace std;

void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);

int cnt, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

void init_code()
{
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
}

int32_t main()
{
    int jm = 0;
    int km = 0;
    int i, choice, cnt = 0;
    char c, ch;
    init_code();
    string str;

    while (cin)
    {
        // Read a Line from File
        getline(cin, str);

        strcpy(production[cnt++], str.c_str());
    }

    int kay;
    char done[cnt];
    int ptr = -1;
    for (k = 0; k < cnt; k++)
    {
        for (kay = 0; kay < 100; kay++)
        {
            calc_first[k][kay] = '!';
        }
    }
```

```
53      int point1 = 0, point2, xxx;
54
55      for (k = 0; k < cnt; k++)
56      {
57          c = production[k][0];
58          point2 = 0;
59          xxx = 0;
60          for (kay = 0; kay <= ptr; kay++)
61              if (c == done[kay])
62                  xxx = 1;
63
64          if (xxx == 1)
65              continue;
66          findfirst(c, 0, 0);
67          ptr += 1;
68          done[ptr] = c;
69          cout << "\n First(" << c << ") = { ";
70          calc_first[point1][point2++] = c;
71
72          for (i = 0 + jm; i < n; i++)
73          {
74              int lark = 0, chk = 0;
75              for (lark = 0; lark < point2; lark++)
76              {
77                  if (first[i] == calc_first[point1][lark])
78                  {
79                      chk = 1;
80                      break;
81                  }
82              }
83              if (chk == 0)
84              {
85                  cout << first[i] << ", ";
86                  calc_first[point1][point2++] = first[i];
87              }
88          }
89          cout << "}\n";
90          jm = n;
91          point1++;
92      }
93      cout << "\n";
94
95      char donee[cnt];
96      ptr = -1;
97      for (k = 0; k < cnt; k++)
98      {
99          for (kay = 0; kay < 100; kay++)
100         {
101             calc_follow[k][kay] = '!';
102         }
103     }
104     point1 = 0;
105     int land = 0;
106     for (e = 0; e < cnt; e++)
107     {
108         ck = production[e][0];
109         point2 = 0;
110         xxx = 0;
111         for (kay = 0; kay <= ptr; kay++)
112             if (ck == donee[kay])
113                 xxx = 1;
```

```
114
115         if (xxx == 1)
116             continue;
117         land += 1;
118         follow(ck);
119         ptr += 1;
120         donee[ptr] = ck;
121         cout << " Follow(" << ck << ") = { ";
122         calc_follow[point1][point2++] = ck;
123
124         for (i = 0 + km; i < m; i++)
125         {
126             int lark = 0, chk = 0;
127             for (lark = 0; lark < point2; lark++)
128             {
129                 if (f[i] == calc_follow[point1][lark])
130                 {
131                     chk = 1;
132                     break;
133                 }
134             }
135             if (chk == 0)
136             {
137                 cout << f[i] << ", ";
138                 calc_follow[point1][point2++] = f[i];
139             }
140         }
141         cout << " }\n\n";
142         km = m;
143         point1++;
144     }
145 }
146
147 void follow(char c)
148 {
149     int i, j;
150     if (production[0][0] == c)
151     {
152         f[m++] = '$';
153     }
154     for (i = 0; i < 10; i++)
155     {
156         for (j = 2; j < 10; j++)
157         {
158             if (production[i][j] == c)
159             {
160                 if (production[i][j + 1] != '\0')
161                 {
162                     followfirst(production[i][j + 1], i, (j + 2));
163                 }
164                 if (production[i][j + 1] == '\0' && c != production[i][0])
165                 {
166                     follow(production[i][0]);
167                 }
168             }
169         }
170     }
171 }
172
173 void followfirst(char c, int c1, int c2)
174 {
```

```
    int k;
    if (!(isupper(c)))
        f[m++] = c;
    else
    {
        int i = 0, j = 1;
        for (i = 0; i < cnt; i++)
        {
            if (calc_first[i][0] == c)
                break;
        }
        while (calc_first[i][j] != '!')
        {
            if (calc_first[i][j] != '#')
            {
                f[m++] = calc_first[i][j];
            }
            else
            {
                if (production[c1][c2] == '\0')
                {
                    follow(production[c1][0]);
                }
                else
                {
                    followfirst(production[c1][c2], c1, c2 + 1);
                }
            }
            j++;
        }
    }
}
```

OUTPUT:



Answer the following Questions based on Program.

1. Write Rules for finding FOLLOW.
   Answer:

   (a) First put $ (the end of input marker) in Follow(S) (S is the start symbol)

   (b) If there is a production A → aBb, (where a can be a whole string) then everything in FIRST(b) except for is placed in FOLLOW(B).

   (c) If there is a production A → aB, then everything in FOLLOW(A) is in FOLLOW(B)

   (d) If there is a production A → aBb, where FIRST(b) contains , then everything in FOLLOW(A) is in FOLLOW(B)

9 Program:-9 Implement RDP for a grammar.

Program:

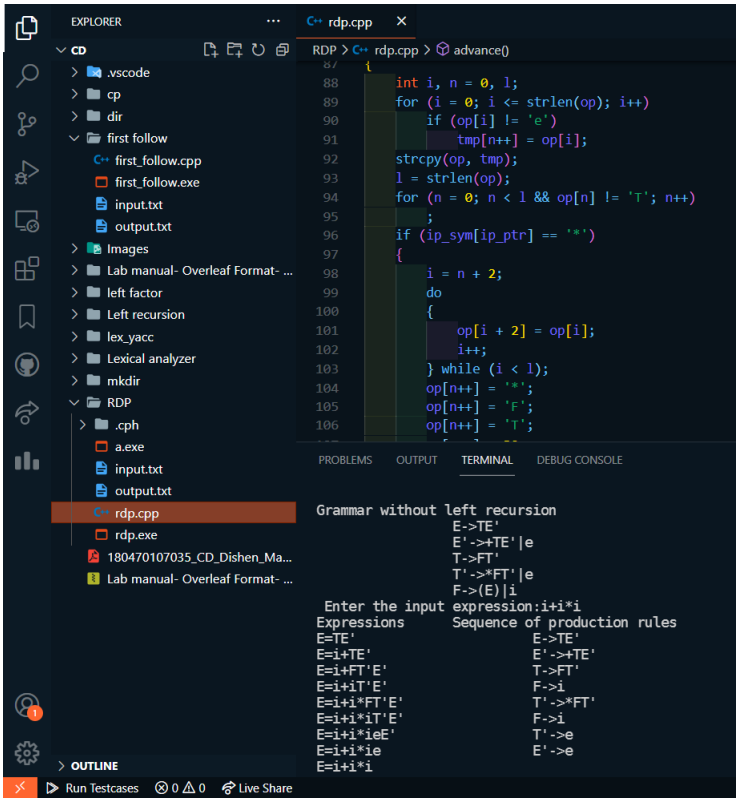
```
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "stdlib.h"
#include "ctype.h"

char ip_sym[15], ip_ptr = 0, op[50], tmp[50];
void e_prime();
void e();
void t_prime();
void t();
void f();
void advance();
int n = 0;
void e()
{
    strcpy(op, "TE'");
    printf("E=%-25s", op);
    printf("E->TE'\n");
    t();
    e_prime();
}

void e_prime()
{
    int i, n = 0, l;
    for (i = 0; i <= strlen(op); i++)
        if (op[i] != 'e')
            tmp[n++] = op[i];
    strcpy(op, tmp);
    l = strlen(op);
    for (n = 0; n < l && op[n] != 'E'; n++)
        ;
    if (ip_sym[ip_ptr] == '+')
    {
        i = n + 2;
        do
        {
            op[i + 2] = op[i];
            i++;
        } while (i <= l);
        op[n++] = '+';
        op[n++] = 'T';
        op[n++] = 'E';
        op[n++] = 39;
        printf("E=%-25s", op);
        printf("E'->+TE'\n");
        advance();
        t();
        e_prime();
    }
    else
    {
        op[n] = 'e';
```

```
55          for (i = n + 1; i <= strlen(op); i++)
56              op[i] = op[i + 1];
57          printf("E=%-25s", op);
58          printf("E'->e");
59      }
60  }
61  void t()
62  {
63      int i, n = 0, l;
64      for (i = 0; i <= strlen(op); i++)
65          if (op[i] != 'e')
66              tmp[n++] = op[i];
67      strcpy(op, tmp);
68      l = strlen(op);
69      for (n = 0; n < l && op[n] != 'T'; n++)
70          ;
71      i = n + 1;
72      do
73      {
74          op[i + 2] = op[i];
75          i++;
76      } while (i < l);
77      op[n++] = 'F';
78      op[n++] = 'T';
79      op[n++] = 39;
80      printf("E=%-25s", op);
81      printf("T->FT'\n");
82      f();
83      t_prime();
84  }
85
86  void t_prime()
87  {
88      int i, n = 0, l;
89      for (i = 0; i <= strlen(op); i++)
90          if (op[i] != 'e')
91              tmp[n++] = op[i];
92      strcpy(op, tmp);
93      l = strlen(op);
94      for (n = 0; n < l && op[n] != 'T'; n++)
95          ;
96      if (ip_sym[ip_ptr] == '*')
97      {
98          i = n + 2;
99          do
100         {
101             op[i + 2] = op[i];
102             i++;
103         } while (i < l);
104         op[n++] = '*';
105         op[n++] = 'F';
106         op[n++] = 'T';
107         op[n++] = 39;
108         printf("E=%-25s", op);
109         printf("T'->*FT'\n");
110         advance();
111         f();
112         t_prime();
113     }
114     else
115     {
```

```
116             op[n] = 'e';
117             for (i = n + 1; i <= strlen(op); i++)
118                 op[i] = op[i + 1];
119             printf("E=%-25s", op);
120             printf("T'->e\n");
121         }
122     }
123
124     void f()
125     {
126         int i, n = 0, l;
127         for (i = 0; i <= strlen(op); i++)
128             if (op[i] != 'e')
129                 tmp[n++] = op[i];
130         strcpy(op, tmp);
131         l = strlen(op);
132         for (n = 0; n < l && op[n] != 'F'; n++)
133             ;
134         if ((ip_sym[ip_ptr] == 'i') || (ip_sym[ip_ptr] == 'I'))
135         {
136             op[n] = 'i';
137             printf("E=%-25s", op);
138             printf("F->i\n");
139             advance();
140         }
141         else
142         {
143             if (ip_sym[ip_ptr] == '(')
144             {
145                 advance();
146                 e();
147                 if (ip_sym[ip_ptr] == ')')
148                 {
149                     advance();
150                     i = n + 2;
151                     do
152                     {
153                         op[i + 2] = op[i];
154                         i++;
155                     } while (i <= l);
156                     op[n++] = '(';
157                     op[n++] = 'E';
158                     op[n++] = ')';
159                     printf("E=%-25s", op);
160                     printf("F->(E)\n");
161                 }
162             }
163             else
164             {
165                 printf("\n\t syntax error");
166                 getch();
167                 exit(1);
168             }
169         }
170     }
171
172     void advance()
173     {
174         ip_ptr++;
175     }
176
```

```
177 int main()
178 {
179     int i;
180     printf("\nGrammar without left recursion");
181     printf("\n\t\t E->TE' \n\t\t E'->+TE'|e \n\t\t T->FT' ");
182     printf("\n\t\t T'->*FT'|e \n\t\t F->(E)|i");
183     printf("\n Enter the input expression:");
184     gets(ip_sym);
185     printf("Expressions");
186     printf("\t Sequence of production rules\n");
187     e();
188     for (i = 0; i < strlen(ip_sym); i++)
189     {
190         if (ip_sym[i] != '+' && ip_sym[i] != '*' && ip_sym[i] != '(' && ip_sym[i] != ')' && ip_sym[i] != 'i' && ip_sym[i
191         {
192             printf("\nSyntax error");
193             break;
194         }
195         for (i = 0; i <= strlen(op); i++)
196             if (op[i] != 'e')
197                 tmp[n++] = op[i];
198         strcpy(op, tmp);
199         printf("\nE=%-25s", op);
200     }
201     getch();
202     return 0;
203 }
```

OUTPUT:



Answer the following Questions based on Program.

1. For Top Down Parsing, When We Announce out result for Input String?
   Answer: If X = a = $, The parser halts and announces successful completion of parsing. So, when our stack contain $ as next symbol then we can announce out result for input string.

2. When String is accepted, What the Stack Contains?
   Answer: When String is accepted, Stack Contains $.

10 Program:-10 To Study about Lexical Analyzer Generator(LEX) and Flex(Fast Lexical Analyzer)

Lex:

```
- Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the
C program.

The function of Lex is as follows:
- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and
produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.

Lex file format
A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:

{ definitions }
%%
 { rules }
%%
{ user subroutines }

Definitions include declarations of constant, variable and regular definitions.

Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.

Where pi describes the regular expression and action1 describes the actions what action the lexical analyzer should
take when pattern pi matches a lexeme.

User subroutines are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer
and compiled separately.
```

Flex:

```
FLEX (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers.
Bison produces parser from the input file provided by the user. The function yylex() is automatically generated by the
flex when it is provided with a .l file and this yylex() function is expected by parser to call to retrieve tokens from
current/this token stream.

Program Structure:

In the input file, there are 3 sections:
1. Definition Section: The definition section contains the declaration of variables, regular definitions, manifest
constants. In the definition section, text is enclosed in
%{ %} brackets. Anything written in this brackets is copied directly to the file lex.yy.c

Syntax:
%{
```

```
    // Definitions
%}

2. Rules Section: The rules section contains a series of rules in the form: pattern action and pattern must be
unintended and action begin on the same line in {} brackets. The rule section is enclosed in %% %%.

Syntax:
%%
pattern  action
%%

3. User Code Section: This section contains C statements and additional functions.
We can also compile these functions separately and load with the lexical analyzer.

Basic Program Structure:
%{
// Definitions
%}
%%
Rules
%%

How to run the program:
To run the program, it should be first saved with the extension .l or .lex. Run the below commands on terminal in order
to run the program file.

Step 1: lex filename.l or lex filename.lex depending on the extension file is saved with
Step 2: gcc lex.yy.c
Step 3: ./a.out
Step 4: Provide the input to program in case it is required
```

Answer the following Questions based on Program.

1. Write Lex Rule for Number.
   Answer: digit[0-9]

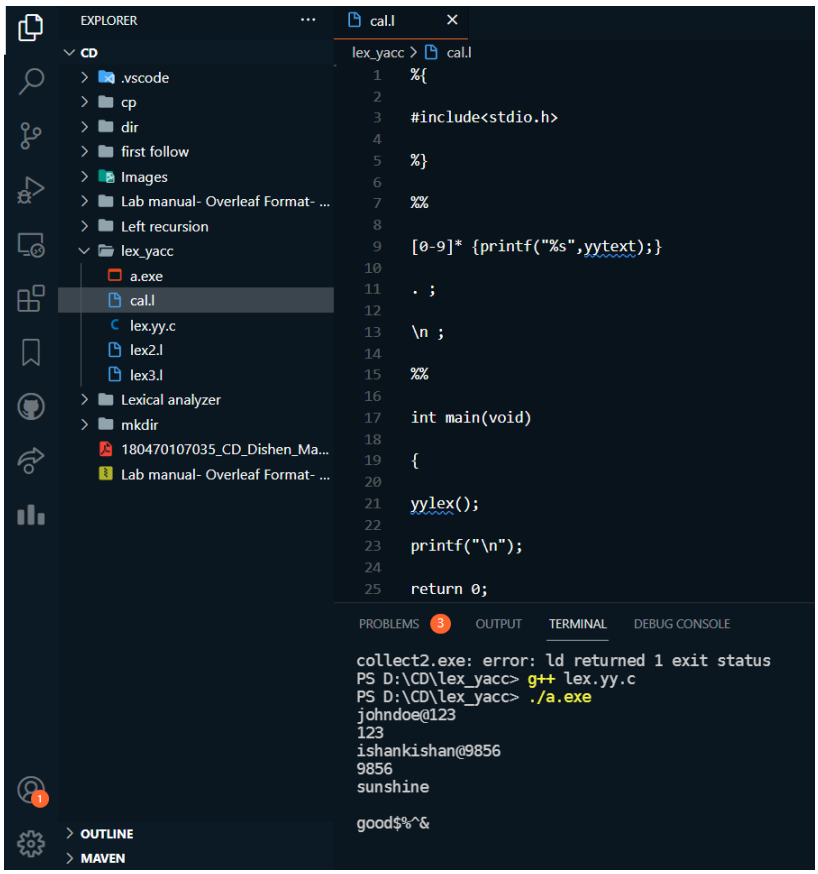
2. Write Lex Rule for Identifiers.
   Answer: {letter}({letter} ||$\{digit\}$) $*$

11 Program:-11 A lexer to print out all numbers from a given file. (Hint: By default lex reads from standard input).

Program:

```
%{
#include <stdio.h>
%}
%%
[0-9]* {printf("%s",yytext);}
. ;
/n ;
%%
int main(void)
{
yylex();
printf("\n");
return 0;
}
int yywrap()
{return(1);}
```

OUTPUT:



Answer the following Questions based on Program.

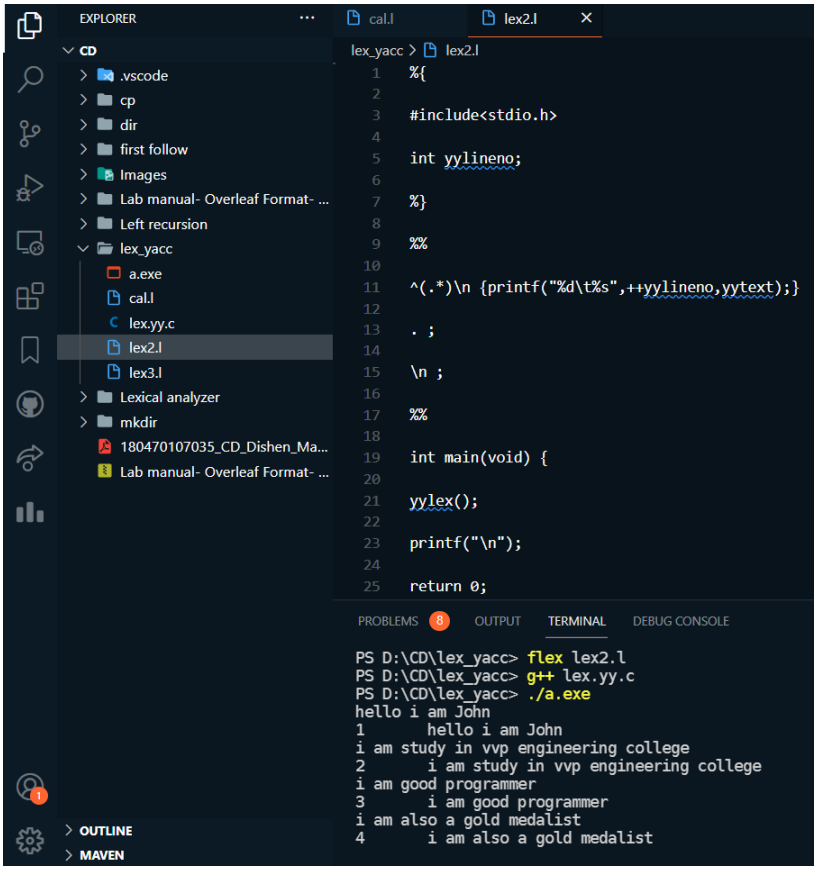
1. Write Lex Rule to print Alphabets from given String.
   Answer: [a - z] [A Z] +

12 Program:-12 A lexer which adds line numbers to the given file and display the same onto the standard output.

Program:

```
%{
#include <stdio.h>
int yylineno;
%}a
%%
^(.*)/n {printf("%d\t%s",++yylineno,yytext);}
. ;
/n ;
%%
int main(void) {
yylex();
printf("\n");
return 0;
}
int yywrap()
{return(1);}
```

OUTPUT:



13 Program:-13 A lexer, which classifies tokens as words, numbers or ”other”.

Program:

```
%{
#include<stdio.h>
%}
%%
[a-zA-Z]+ {printf("Word : %s\n",yytext);}
[0-9]+ {printf("Number : %s\n",yytext);}
[^ /t/na-zA-Z0-9]+|[a-zA-Z0-9]+ {printf("Others : %s\n",yytext);}
. ;
%%
int main(void)
{
yylex();
printf("\n");
return 0;
}

int yywrap()
{return(1);}
```

OUTPUT: