Name: Ali Hassaan Mughal

Reg: 173627

BSCS6-B

Language: C++

**Task 1:**

**Code:**

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <iterator>

#include <map>

using namespace std;

int main()

{

vector<string> variables;

vector<string> notvariables;

string tmpvar = "";

map<char,char> schars;

schars.insert(pair<char,char> ('+','+'));

//Returns end of map if element does not exist, else iterator to the element;

if (schars.end() == schars.find('+'))

cout << "Not Found";

int begin=0,forward=0,lCount=0, i,j;

string buffer;

ifstream file("inputexpressions.txt");

if (file.is\_open())

{

while (getline(file,buffer))

{

begin=0;

forward=0;

for (i=0;i<buffer.length();i++ )

{

if (buffer[i] == ' ')

continue;

if (buffer[i] == '=' || buffer[i] == '+' || buffer[i] == '\*' || buffer[i] == '/' || buffer[i] == '-' || i == buffer.length()-1)

{

tmpvar="";

schars.insert(pair<char,char>(buffer[i],buffer[i]));

if (i>0)

{

forward = i-1;

for (int j=begin; j<=forward;j++)

{

if (buffer[j] != ' ')

{

tmpvar=tmpvar+buffer[j];

if (! ((buffer[j] >= 'a' && buffer[j] <= 'z' ) || (buffer[j] >= 'A' &&buffer[j] <= 'Z' ) || (buffer[j] >= '0' &&buffer[j] <= '9' )))

schars.insert(pair<char,char> (buffer[j],buffer[j]));

}

}

variables.push\_back(tmpvar);

}

forward=i+1;

begin=i+1;

}

}

//cout << buffer << endl;

lCount++;

}

}

//cout << lCount;

cout << "The Variables or Identifiers are: " << endl;

for (i=0;i<variables.size();i++)

{

if (! ( (variables[i][0] >= 'a' && variables[i][0]<='z') || (variables[i][0]>='A' && variables[i][0]<='Z') || variables[i][0] == '\_' ))

{

notvariables.push\_back(variables[i]);

variables.erase(variables.begin() + i);

continue;

i--;

}

for (j=1;j<variables[i].length();j++)

{

if (! ( (variables[i][j]>='a' && variables[i][j] <= 'z') || (variables[i][j]>='A' && variables[i][j]<='Z') || (variables[i][j]>='0' && variables[i][j]<='9') || variables[i][j] == '\_' ) )

{

notvariables.push\_back(variables[i]);

variables.erase(variables.begin() + i);

i--;

break;

}

}

}

for (int i=0;i<variables.size();i++)

{

cout << variables[i] << endl;

}

cout << "Not variables list: " << endl;

for (int i=0;i<notvariables.size();i++)

{

cout << notvariables[i] << endl;

}

map<char, char>::iterator itr;

cout << endl << endl << "Special Characters and opertors are: " ;

for (itr = schars.begin(); itr != schars.end(); ++itr)

{

cout << itr->first << " ";

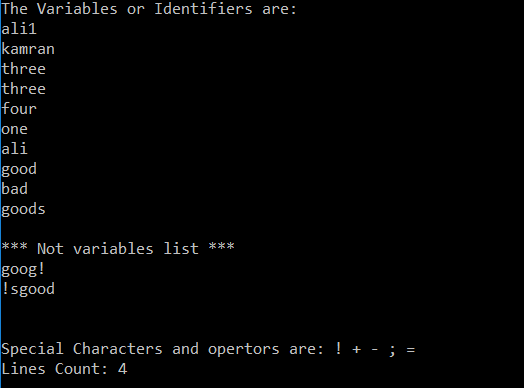
}

cout << endl << "Lines Count: " << lCount;

return 0;

}

**Output:**



**Task 2:**

**Code:**

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <iterator>

#include <map>

using namespace std;

int main()

{

vector<string> variables;

vector<string> notvariables;

string tmpvar = "";

map<char,char> schars;

schars.insert(pair<char,char> ('+','+'));

//Returns end of map if element does not exist, else iterator to the element;

if (schars.end() == schars.find('+'))

cout << "Not Found";

int begin=0,forward=0,lCount=0, i,j;

string buffer,buffer1;

ifstream file("inputexpressions.txt");

if (file.is\_open())

{

while (getline(file,buffer))

{

getline(file,buffer1);

begin=0;

forward=0;

for (i=0;i<buffer.length();i++ )

{

if (buffer[i] == ' ')

continue;

if (buffer[i] == '=' || buffer[i] == '+' || buffer[i] == '\*' || buffer[i] == '/' || buffer[i] == '-' || i == buffer.length()-1)

{

tmpvar="";

schars.insert(pair<char,char>(buffer[i],buffer[i]));

if (i>0)

{

forward = i-1;

for (int j=begin; j<=forward;j++)

{

if (buffer[j] != ' ')

{

tmpvar=tmpvar+buffer[j];

if (! ((buffer[j] >= 'a' && buffer[j] <= 'z' ) || (buffer[j] >= 'A' &&buffer[j] <= 'Z' ) || (buffer[j] >= '0' &&buffer[j] <= '9' )))

schars.insert(pair<char,char> (buffer[j],buffer[j]));

}

}

variables.push\_back(tmpvar);

}

forward=i+1;

begin=i+1;

}

}

begin=0;

forward=0;

for (i=0;i<buffer1.length();i++ )

{

if (buffer1[i] == ' ')

continue;

if (buffer1[i] == '=' || buffer1[i] == '+' || buffer1[i] == '\*' || buffer1[i] == '/' || buffer1[i] == '-' || i == buffer1.length()-1)

{

tmpvar="";

schars.insert(pair<char,char>(buffer1[i],buffer1[i]));

if (i>0)

{

forward = i-1;

for (int j=begin; j<=forward;j++)

{

if (buffer1[j] != ' ')

{

tmpvar=tmpvar+buffer1[j];

if (! ((buffer1[j] >= 'a' && buffer1[j] <= 'z' ) || (buffer1[j] >= 'A' &&buffer[j] <= 'Z' ) || (buffer[j] >= '0' &&buffer[j] <= '9' )))

schars.insert(pair<char,char> (buffer1[j],buffer1[j]));

}

}

variables.push\_back(tmpvar);

}

forward=i+1;

begin=i+1;

}

}

//cout << buffer << endl;

lCount++;

}

}

//cout << lCount;

cout << "The Variables or Identifiers are: " << endl;

for (i=0;i<variables.size();i++)

{

if (! ( (variables[i][0] >= 'a' && variables[i][0]<='z') || (variables[i][0]>='A' && variables[i][0]<='Z') || variables[i][0] == '\_' ))

{

notvariables.push\_back(variables[i]);

variables.erase(variables.begin() + i);

continue;

i--;

}

for (j=1;j<variables[i].length();j++)

{

if (! ( (variables[i][j]>='a' && variables[i][j] <= 'z') || (variables[i][j]>='A' && variables[i][j]<='Z') || (variables[i][j]>='0' && variables[i][j]<='9') || variables[i][j] == '\_' ) )

{

notvariables.push\_back(variables[i]);

variables.erase(variables.begin() + i);

i--;

break;

}

}

}

for (int i=0;i<variables.size();i++)

{

cout << variables[i] << endl;

}

cout << endl << "\*\*\*Not variables list\*\*\*: " << endl;

for (int i=0;i<notvariables.size();i++)

{

cout << notvariables[i] << endl;

}

map<char, char>::iterator itr;

cout << endl << "Special Characters and opertors are: " ;

for (itr = schars.begin(); itr != schars.end(); ++itr)

{

cout << itr->first << " ";

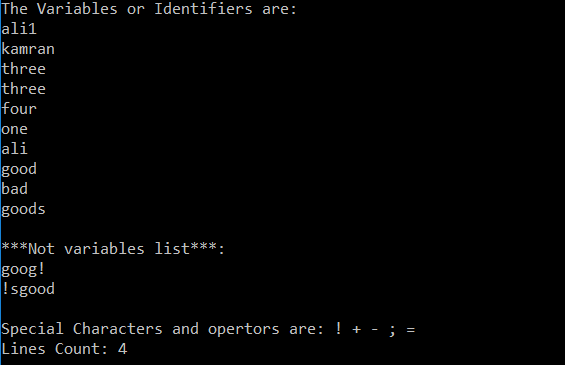
}

cout << endl << "Lines Count: " << lCount;

return 0;

}

**Output:**



**Task 3:**

**Code:**

#include <iostream>

#include <string>

using namespace std;

int main()

{

cout << "Words starting with T/t or M/m are:" << endl;

string text = "Once the next lexeme is determined, forward is set to character at its right end. Then, after the lexeme is recorded as an attribute value of a token returned to the parser, Lexeme Begin is set to the character immediately after the lexeme just found. If we use the scheme of Buffer pairs we must check, each time we advance forward, that we have not moved off one of the buffers; if we do, then we must also reload the other buffer. Thus, for each character read, we make two tests: one for the end of the buffer, and one to determine what character is read (the latter may be a multiway branch). We can combine the buffer-end test with the test for the current character if we extend each buffer to hold a sentinel character at the end. The sentinel is a special character that cannot be part of the source program, and a natural choice is the character EOF.";

int begin = 0, j;

int forward = 0;

for (int i=0;i<text.length();i++){

if (text[i] == ' ')

{

forward=i-1;

if (text[begin] == 't' || text[begin] == 'm' || text[begin] == 'T' || text[begin] == 'M' )

{

for (j=begin;j<=forward;j++)

{

if (((text[j] >= 'a' && text[j] <= 'z') || (text[j] >= 'A' && text[j] <= 'Z')))

cout << text[j];

}

cout << "\t";

}

begin=i+1;

forward=i+1;

}

}

return 0;

}

**Output:**

