American International University-Bangladesh

Compiler Design

(ASSIGNMENT)

Submitted By: **Kamruzzaman Sony**

ID: 22-46797-1

Section: [H]

Course Instructor Name: **Nazmus Sakib Shan**

Date of Submission: 4 May 2024

1. GitHub Repository Link:

   https://github.com/hisony/Compiler-Design/tree/main/Lexical_analyzer_assignment

2. Input:

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

int main() {
cout << "Welcome";
int x = 24 % 10;
if (x === 4) {
x = 40;
}
int y = 50;
int #z = 60;

return 0;
}
```

3. Programme code:

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

using namespace std;

vector<pair<int, string>> ans;

bool isValidVariableName(const string &name)
{
```

```
    if (name.empty() || !isalpha(name[0]) && name[0] != '_')
    {
        return false;
    }
    for (char c : name)
    {
        if (!isalnum(c) && c != '_')
        {
            return false;
        }
    }
    return true;
}

bool isValidFunctionName(const string &name)
{
    if (name.empty() || !isalpha(name[0]) && name[0] != '_')
    {
        return false;
    }
    for (char c : name)
    {
        if (!isalnum(c) && c != '_' && c != '(' && c != ')')
        {
            return false;
        }
    }
    return true;
}

bool isAllDigits(const string &str)
{
    return all_of(str.begin(), str.end(), ::isdigit);
}

bool containsArithmeticOperator(const string &str)
{
    string operators = "+-*/%";

    for (char c : str)
    {
        if (operators.find(c) != string::npos)
        {
            return true;
        }
```

```
    }

    return false;
}

void process(int lineNo, string s)
{
    int left_idx = 0;
    int right_idx = 0;

    s += ' ';

    string temp;

    bool isKeyword = false;

    while (right_idx < s.size() && s[right_idx] == ' ')
    {
        right_idx++;
    }

    while (right_idx < s.size())
    {
        if (s[right_idx] != ' ')
        {
            temp += s[right_idx];
            right_idx++;
        }
        else
        {
            string current_temp = temp;
            temp = "";

            while (right_idx < s.size() && s[right_idx] == ' ')
            {
                right_idx++;
            }

            if (current_temp == "#include")
            {
                ans.push_back({lineNo, "Header File Declaration."});
                return;
            }

            if (current_temp == "using")
```

```
            {
                ans.push_back({lineNo, "Namespace Declaration."});
                return;
            }

            if (current_temp == "return")
            {
                ans.push_back({lineNo, "Return Called."});
                return;
            }

            if (current_temp == "int" || current_temp == "char" || current_temp
== "string" || current_temp == "float" || current_temp == "double")
            {
                current_temp += " = Keyword.";
                ans.push_back({lineNo, current_temp});
                isKeyword = true;
                continue;
            }

            if (current_temp == "cin" || current_temp == "cout" || current_temp
== "<<" || current_temp == ">>")
            {
                current_temp += " = built in name.";
                ans.push_back({lineNo, current_temp});
                continue;
            }

            if (current_temp == "if" || current_temp == "else")
            {
                ans.push_back({lineNo, "Condition Declaration."});
                continue;
            }

            if (current_temp == "{" || current_temp == "}")
            {
                ans.push_back({lineNo, '"' + current_temp + '"' + " = Bracket
Sequence."});
                continue;
            }

            if (current_temp == "=" || current_temp == "==" || current_temp ==
"!=")
            {
```

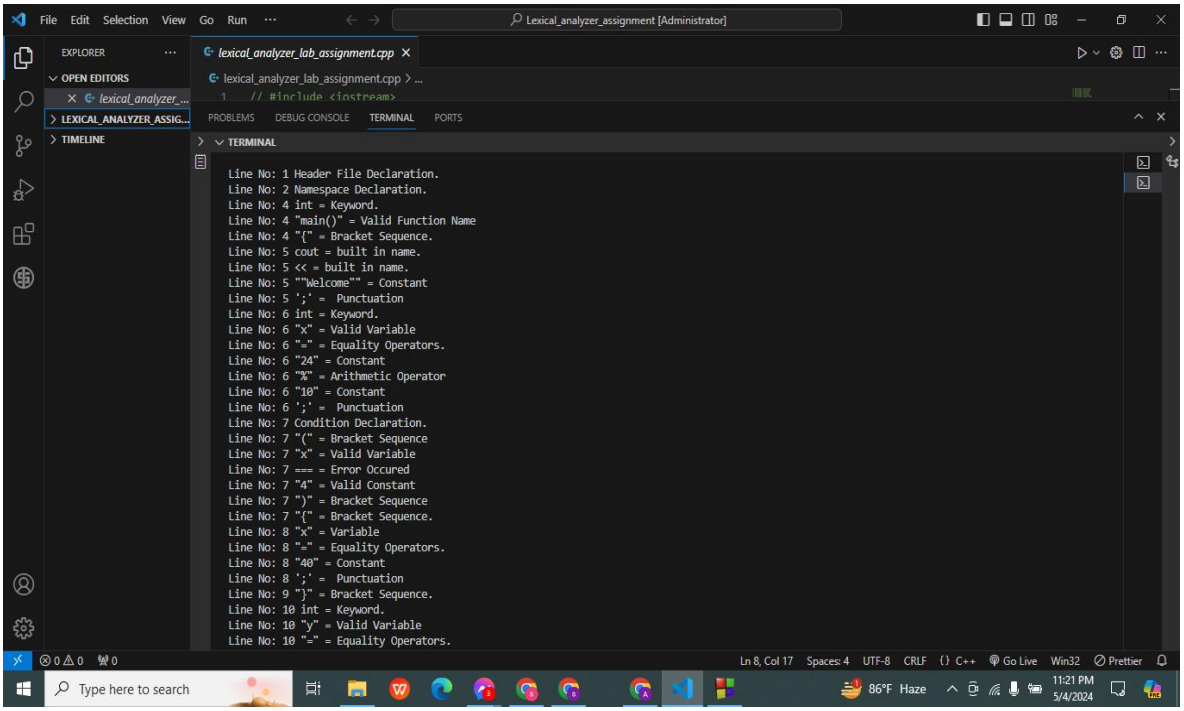
```
                ans.push_back({lineNo, '"' + current_temp + '"' + " = Equality
Operators."});
                continue;
            }

            if (isKeyword)
            {
                isKeyword = false;
                if (isValidVariableName(current_temp))
                {
                    ans.push_back({lineNo, '"' + current_temp + '"' + " = Valid
Variable"});
                    continue;
                }
                else
                {
                    if (isValidFunctionName(current_temp))
                    {
                        ans.push_back({lineNo, '"' + current_temp + '"' + " =
Valid Function Name"});
                        continue;
                    }
                    else
                    {
                        ans.push_back({lineNo, '"' + current_temp + '"' + " =
Error inValid Variable/Function Name"});
                        continue;
                    }
                }
            }

            if (current_temp.size() && current_temp[current_temp.size() - 1] ==
';')
            {
                if (right_idx < s.size())
                {
                    ans.push_back({lineNo, '"' + current_temp + '"' + " = Error
inValid Punctuation."});
                    continue;
                }

                string varCheck = current_temp;
                varCheck.pop_back();
                if (isValidVariableName(varCheck))
                {
```

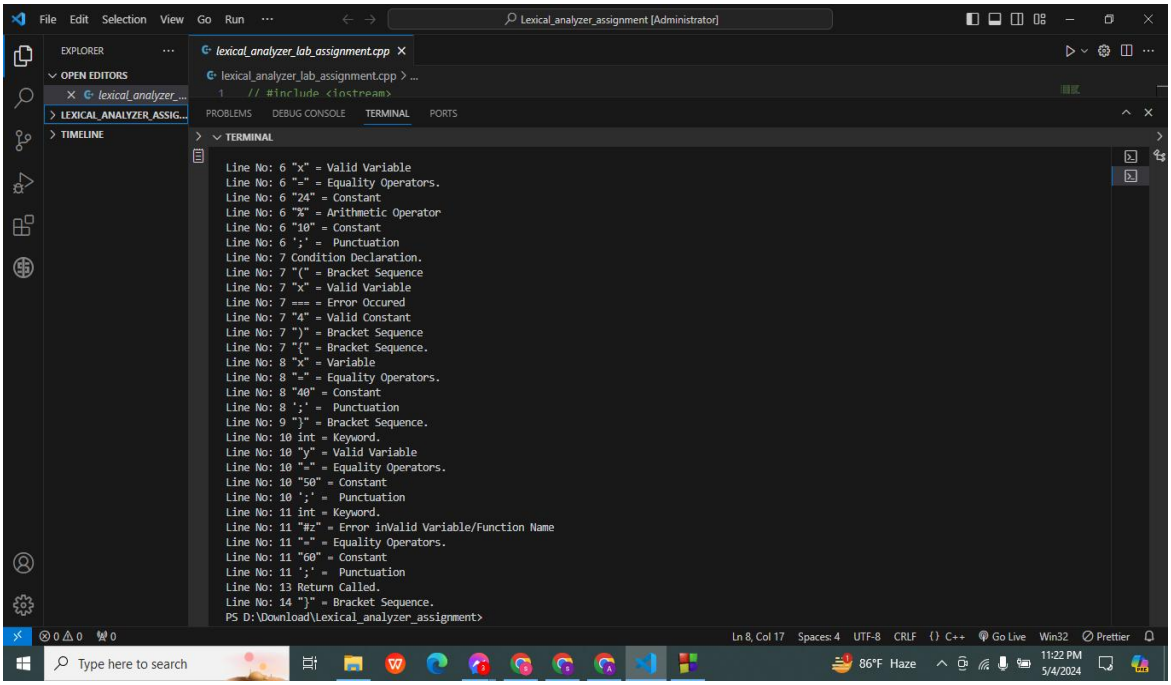
```
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Variable"});
                }
                else if (isAllDigits(varCheck))
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Constant"});
                }
                else if (varCheck[varCheck.size() - 1] == '"')
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Constant"});
                }
                else
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Error
inValid Variable/Constant"});
                }

                ans.push_back({lineNo, "';' =  Punctuation"});
                continue;
            }

            if (current_temp.size() && (current_temp[current_temp.size() - 1] ==
')' || current_temp[current_temp.size() - 1] == '}'))
            {

                string varCheck = current_temp;
                varCheck.pop_back();
                if (isValidVariableName(varCheck))
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Valid
Variable"});
                }
                else if (isAllDigits(varCheck))
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Valid
Constant"});
                }
                else
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Error
inValid Variable/Constant"});
                }

                string aa;
                aa += current_temp[current_temp.size() - 1];
```

```
                ans.push_back({lineNo, '"' + aa + '"' + " = Bracket Sequence"});
                continue;
            }

            if (current_temp.size() && (current_temp[0] == '(' || current_temp[0]
== '{'))
            {

                string varCheck;
                for (int i = 1; i < current_temp.size(); i++)
                    varCheck += current_temp[i];

                string aa;
                aa += current_temp[0];

                ans.push_back({lineNo, '"' + aa + '"' + " = Bracket Sequence"});

                if (isValidVariableName(varCheck))
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Valid
Variable"});
                }
                else if (isAllDigits(varCheck))
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Valid
Constant"});
                }
                else
                {
                    ans.push_back({lineNo, '"' + varCheck + '"' + " = Error
inValid Variable/Constant"});
                }
                continue;
            }

            if (isValidVariableName(current_temp))
            {
                ans.push_back({lineNo, '"' + current_temp + '"' + " = Variable"});
                continue;
            }
            else if (isAllDigits(current_temp))
            {
                ans.push_back({lineNo, '"' + current_temp + '"' + " = Constant"});
                continue;
            }
```

```
            else if (containsArithmeticOperator(current_temp))
            {
                ans.push_back({lineNo, '"' + current_temp + '"' + " = Arithmetic
Operator"});
                continue;
            }

            ans.push_back({lineNo, current_temp + " = Error Occured"});
            continue;
        }
    }
}

int main()
{
    ifstream inFile;
    inFile.open("input.txt");

    int cnt = 1;

    if (inFile.is_open())
    {
        string input;
        while (getline(inFile, input))
        {
            process(cnt, input);
            cnt++;
        }
    }

    for (auto it : ans)
    {
        cout << "Line No: " << it.first << ' ' << it.second << endl;
    }

    inFile.close();
    return 0;
}
```

4. Output:


File Edit Selection View Go Run
Lexical_analyzer_assignment [Administrator]
EXPLORER
OPEN EDITORS
lexical_analyzer_...
LEXICAL_ANALYZER_ASSIG...
TIMELINE
lexical_analyzer_lab_assignment.cpp
lexical_analyzer_lab_assignment.cpp > ...
// #include <iostream>
PROBLEMS DEBUG CONSOLE TERMINAL PORTS
TERMINAL
Line No: 1 Header File Declaration.
Line No: 2 Namespace Declaration.
Line No: 4 int = Keyword.
Line No: 4 "main()" = Valid Function Name
Line No: 4 "{" = Bracket Sequence.
Line No: 5 cout = built in name.
Line No: 5 << = built in name.
Line No: 5 ""Welcome"" = Constant
Line No: 5 ';' = Punctuation
Line No: 6 int = Keyword.
Line No: 6 "x" = Valid Variable
Line No: 6 "=" = Equality Operators.
Line No: 6 "24" = Constant
Line No: 6 "%" = Arithmetic Operator
Line No: 6 "10" = Constant
Line No: 6 ';' = Punctuation
Line No: 7 Condition Declaration.
Line No: 7 "(" = Bracket Sequence
Line No: 7 "x" = Valid Variable
Line No: 7 === = Error Occured
Line No: 7 "4" = Valid Constant
Line No: 7 ")" = Bracket Sequence
Line No: 7 "{" = Bracket Sequence.
Line No: 8 "x" = Variable
Line No: 8 "=" = Equality Operators.
Line No: 8 "40" = Constant
Line No: 8 ';' = Punctuation
Line No: 9 "}" = Bracket Sequence.
Line No: 10 int = Keyword.
Line No: 10 "y" = Valid Variable
Line No: 10 "=" = Equality Operators.
Ln 8, Col 17 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32 Prettier
Type here to search
86°F Haze
11:21 PM
5/4/2024


File Edit Selection View Go Run
Lexical_analyzer_assignment [Administrator]
EXPLORER
OPEN EDITORS
lexical_analyzer_...
LEXICAL_ANALYZER_ASSIG...
TIMELINE
lexical_analyzer_lab_assignment.cpp
lexical_analyzer_lab_assignment.cpp > ...
// #include <iostream>
PROBLEMS DEBUG CONSOLE TERMINAL PORTS
TERMINAL
Line No: 6 "x" = Valid Variable
Line No: 6 "=" = Equality Operators.
Line No: 6 "24" = Constant
Line No: 6 "%" = Arithmetic Operator
Line No: 6 "10" = Constant
Line No: 6 ';' = Punctuation
Line No: 7 Condition Declaration.
Line No: 7 "(" = Bracket Sequence
Line No: 7 "x" = Valid Variable
Line No: 7 === = Error Occured
Line No: 7 "4" = Valid Constant
Line No: 7 ")" = Bracket Sequence
Line No: 7 "{" = Bracket Sequence.
Line No: 8 "x" = Variable
Line No: 8 "=" = Equality Operators.
Line No: 8 "40" = Constant
Line No: 8 ';' = Punctuation
Line No: 9 "}" = Bracket Sequence.
Line No: 10 int = Keyword.
Line No: 10 "y" = Valid Variable
Line No: 10 "=" = Equality Operators.
Line No: 10 "50" = Constant
Line No: 10 ';' = Punctuation
Line No: 11 int = Keyword.
Line No: 11 "#z" = Error inValid Variable/Function Name
Line No: 11 "=" = Equality Operators.
Line No: 11 "60" = Constant
Line No: 11 ';' = Punctuation
Line No: 13 Return Called.
Line No: 14 "}" = Bracket Sequence.
PS D:\Download\Lexical_analyzer_assignment>
Ln 8, Col 17 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32 Prettier
Type here to search
86°F Haze
11:22 PM
5/4/2024