Assignment - 1
2023IMG-040
CD LAB

Question 1: Design and implement a lexical analyzer for given language using C/C++/Python and the lexical analyzer should ignore redundant spaces, tabs and new line.

```
Prog
   Integer a, b
   Begin
      read n;
      if a < 10
      then
         b :=1;
         else;
      endif
      while a < 10
      do
         b := 5*a;
         a := a+1;
      endwhile;

      write a;
      write b;

end
```

Code:

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

// operators, keywords, numbers, variables

static const unordered_set<string> KEYWORDS = {
    "INTEGER","BEGIN","READ","IF","ELSE","THEN","ENDIF",
    "WHILE","DO","ENDWHILE","WRITE","PROG","END"
};

bool isKeyword( string &s) {
    return KEYWORDS.find(s) != KEYWORDS.end();
}

// This is was of O(N)
// bool isKeyword(char buffer[])
// {
//     // vector<string> keywords(13) = {
//     //     "INTEGER", "BEGIN", "READ", "IF", "ELSE", "THEN","ENDIF", "WHILE",
"DO","ENDWHILE", "WRITE", "PROG", "END"
//     // };
//     char keywords[32][10] = {"INTEGER", "BEGIN", "READ", "IF", "ELSE", "THEN",
"ENDIF", "WHILE", "DO", "ENDWHILE", "WRITE", "Prog", "End"};
//     int i, flag = 0;
//     for (i = 0; i < 13; ++i)
//     {
//         if (strcmp(keywords[i], buffer) == 0)
//         {
//             return true;
//         }
//     }
//     return false;
// }

bool isOperator(char ch)
{
    if (
        ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=' || ch == ';')
    {
        return true;
    }

    return false;
}


//O(1) using string_view
bool isNumber(const string_view &sv) {
    if (sv.empty()) return false;
    int dots = 0;
    for (size_t i = 0; i < sv.size(); ++i) {
        char c = sv[i];
        if (c == '.') {
            if (++dots > 1) return false;
        }
        else if (!(isdigit(c) || (c=='-' && i==0))) {
            return false;
        }
    }
    return true;
}

// was in general O(N)
// bool isNumber(char *str)
// {
//     int len = strlen(str);
//     int numberOfDecimal = 0;
//     if (len == 0)
//     {
//         return false;
//     }
//     for (int i = 0; i < len; i++)
//     {
//         if (str[i] == '.')
//         {
//             numberOfDecimal++;
//             if (numberOfDecimal > 1)
//             {
//                 return false;
//             }
//         }
//         else if (!(str[i] >= '0' && str[i] <= '9') && !(str[i] == '-' && i == 0))
//         {
//             return false;
//         }
//     }
//     return true;
// }

char *substringExtraction(char *realString, int l, int r)
{
    int i;
    char *str = (char *)malloc(sizeof(char) * (r - l + 2));
    for (int i = l; i <= r; i++)
    {
        str[i - l] = realString[i];
        str[r - l + 1] = '\0';
    }
    return str;
}

void parser(string &input) {
    size_t left = 0, right = 0, N = input.size();
    while (right <= N && left <= right) {
        if (right < N && isOperator(input[right])) {
            cout << input[right] << " is an operator\n";
            ++right;
            left = right;
        }
        else if (right == N || isspace((unsigned char)input[right])) {
            if (left != right) {
                string_view raw{ input.data() + left, right - left };

                string up{ raw };
                transform(up.begin(), up.end(), up.begin(), ::toupper);

                if (isKeyword(up)) {
                    cout << raw << " is a keyword\n";
                }
                else if (isNumber(raw)) {
                    cout << raw << " is a number\n";
                }
                else {
                    cout << raw << " is an identifier\n";
                }
            }
            ++right;
            left = right;
        }
        else {
            ++right;
        }
    }
}

// void parser(char *str)
// {
//     int left = 0, right = 0;
//     int len = strlen(str);
//     while (right <= len && left <= right)
//     {
//         if (right < len && isOperator(str[right]))
//         {
//             cout << str[right] << " is an operator\n";
//             right++;
//             left = right;
//         }
//         else if (right == len || str[right] == ' ' || str[right] == '\t' || str[right]
== '\n')
//         {
//             if (left != right)
//             {
//                 char *sub = substringExtraction(str, left, right - 1);
//                 if (isKeyword(sub))
//                 {
//                     cout << sub << " is a keyword\n";
//                 }
//                 else if (isNumber(sub))
//                 {
//                     cout << sub << " is a number\n";
//                 }
//                 else
//                 {
//                     cout << sub << " is an identifier\n";
//                 }
//                 free(sub);
//             }
//             right++;
//             left = right;
//         }
//         else
//         {
//             right++;
//         }
//     }
// }

int main() {
    cout << "Working dir: " << filesystem::current_path() << "\n";

    ifstream fin("program.txt");
    if (!fin.is_open()) {
        cerr << "❌ error opening program.txt\n";
        return 1;
    }


    string fileContent, line;
    while (getline(fin, line)) {
        fileContent += line;
        fileContent += ' ';
    }
    fin.close();

    cout << "Input program:\n" << fileContent << "\n\nTokens:\n";
    parser(fileContent);
    return 0;
}
```

Output:

```
Working dir: "/home/charizard-op/Desktop/Compiler Design/LAB 1/test_assignment1"
Input program:
Prog     INTEGER a, b     BEGIN         READ n;         IF a < 10         THEN
b :=1;             ELSE;         ENDIF         WHILE a < 10         DO             b :=
5*a;             a := a+1;         ENDWHILE;                  WRITE a;         WRITE b;
end

Tokens:
Prog is a keyword
INTEGER is a keyword
a, is an identifier
b is an identifier
BEGIN is a keyword
READ is a keyword
; is an operator
IF is a keyword
a is an identifier
< is an operator
10 is a number
THEN is a keyword
b is an identifier
= is an operator
; is an operator
; is an operator
ENDIF is a keyword
WHILE is a keyword
a is an identifier
< is an operator
10 is a number
DO is a keyword
b is an identifier
= is an operator
* is an operator
; is an operator
a is an identifier
= is an operator
+ is an operator
; is an operator
; is an operator
WRITE is a keyword
; is an operator
WRITE is a keyword
; is an operator
end is a keyword
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm}
0<"/tmp/Microsoft-MIEngine-In-tcga12s0.olo" 1>"/tmp/Microsoft-MIEngine-Out-5dv2pjd4.v5a"
```

Time Complexity :
O(N) overall

Earlier, it was O(N^2) due to substring comparisons and two pointer approach.