21CS8056
Sayan Paul

Compiler Design Lab-4

Imagine you are building a compiler for a simple programming language with the following token class specifications:

- *Identifiers consist of letters (a-z, A-Z) and digits (0-9) and are case-sensitive.*
- *Keywords: if, else, while, int.*
- *Operators: +, -, *, /.*
- *Parentheses: (,).*

Your task is to write a tokenizer that follows the maximal munch principle.

Write a shell script or a program in C that takes an input program as a string and outputs the sequence of tokens. For simplicity, you can assume that there are spaces between tokens.

If the Input is:
if (x = 1) {
y = y + 1;
} else {
y = y - 1;
}

The output should be:
1. *if, (, x, =, 1,), {, y, =, y, +, 1, ;, }, else, {, y, =, y, -, 1, ;, }*

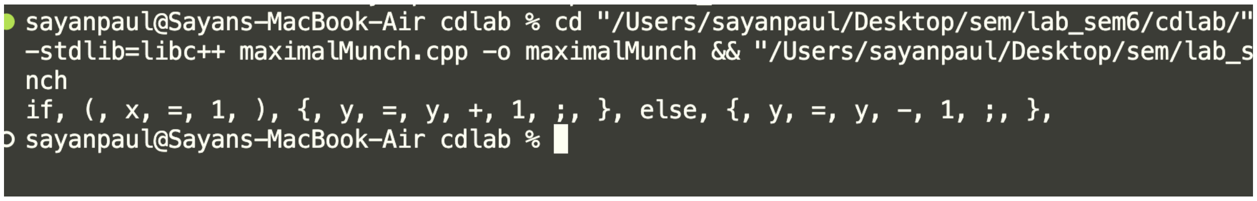
CODE:

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

using namespace std;

// prefix of kmp algo
vector<int> prefix(string &s)
{
    int n = s.length();
    vector<int> pi(n); // DFA
    for (int i = 1; i < n; i++)
    {
        int j = pi[i - 1];
        while ((j > 0) && (s[i] != s[j]))
```

```
            j = pi[j - 1];

        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }

    return pi;
}

// kmp algo to return starting indexes of a specific pattern, p
vector<int> index(string &s, string &p)
{
    string t = p;
    t += '\0';
    t += s;

    int n = p.size();
    vector<int> pi = prefix(t), ans;
    for (int i = n + 1; i < t.size(); i++)
    {
        if (pi[i] == n)
            ans.push_back(i - (2 * n));
    }

    return ans;
}

// appling maximal munch algo on the given string, prog
void maximalMunch(string &prog, vector<string> &patterns)
{
    int n = prog.size();
    // stores len of pattern, index of that pattern in vector
    vector<pair<int, int>> munch(n);

    for (int x = 0; x < patterns.size(); x++)
    {
        // stores starting indexes of pattern[x]
        vector<int> idx = index(prog, patterns[x]);
        for (int i : idx)
        {
            if (munch[i].first < patterns[x].length())
            {
                munch[i].first = patterns[x].length();
                munch[i].second = x;
            }
        }
```

```cpp
    }

    for (int i = 0; i < prog.size();)
    {
        // Delimiter
        if (prog[i] == '\n' || prog[i] == ' ')
            i++;
        // no pattern matches
        else if (munch[i].first == 0)
        {
            cout << "ERROR at index " << i << endl;
            return;
        }
        // pattern matches and printing it
        else
        {
            cout << patterns[munch[i].second] << ", ";
            i = i + munch[i].first;
        }
    }

    cout << endl;
    return;
}

int main()
{
    string prog = "if (x = 1) { y = y + 1; } else { y = y - 1;
}";
    vector<string> patterns = {"if", "(", ")", "x", "y", "+",
"-", ";", "1", "else", "{", "}", "="};
    // linear time algorithm
    maximalMunch(prog, patterns);

    // Maximal Munch will not work in this case
    // string prog = "abcdef";
    // vector<string> patterns = {"ab","abcd","cdef"};
    return 0;
}
```

OUTPUT:



```
sayanpaul@Sayans-MacBook-Air cdlab % cd "/Users/sayanpaul/Desktop/sem/lab_sem6/cdlab/"
-stdlib=libc++ maximalMunch.cpp -o maximalMunch && "/Users/sayanpaul/Desktop/sem/lab_s
nch
if, (, x, =, 1, ), {, y, =, y, +, 1, ;, }, else, {, y, =, y, -, 1, ;, },
sayanpaul@Sayans-MacBook-Air cdlab %
```