Internal Practical – Compiler

Topic :- Implementation of a lexical analyzer for c language compiler .

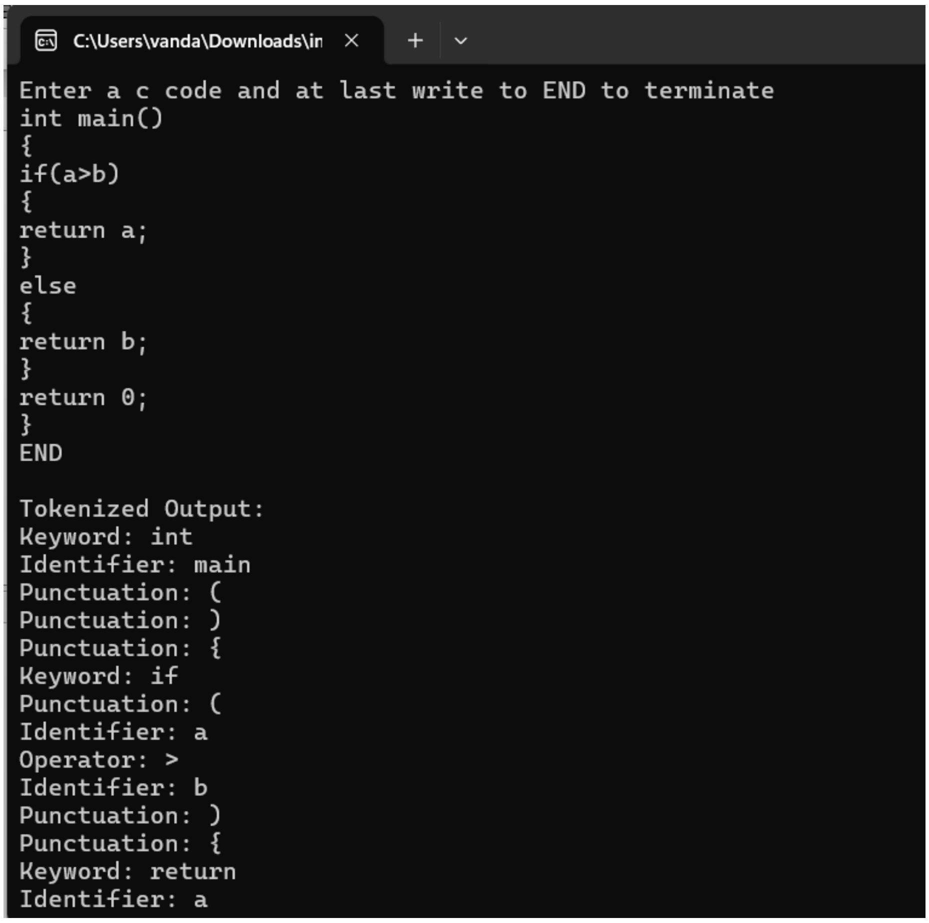
Code :-

```
#include<iostream>
#include<cctype>
#include<unordered_map>
#include<unordered_set>
#include<regex>

using namespace std;



unordered_set<string> keywords = {"int","return","if","else"};

unordered_set<string> symbolTable;
unordered_set<string> functions;

bool isKeyword(const string& str)
{
   return keywords.find(str) != keywords.end();
}

bool isOperator(char c)
{
   string operators = "+-*/%=<>!&|";
   return operators.find(c) != string::npos;
}

void tokenize(string code)
{
   string token;
```

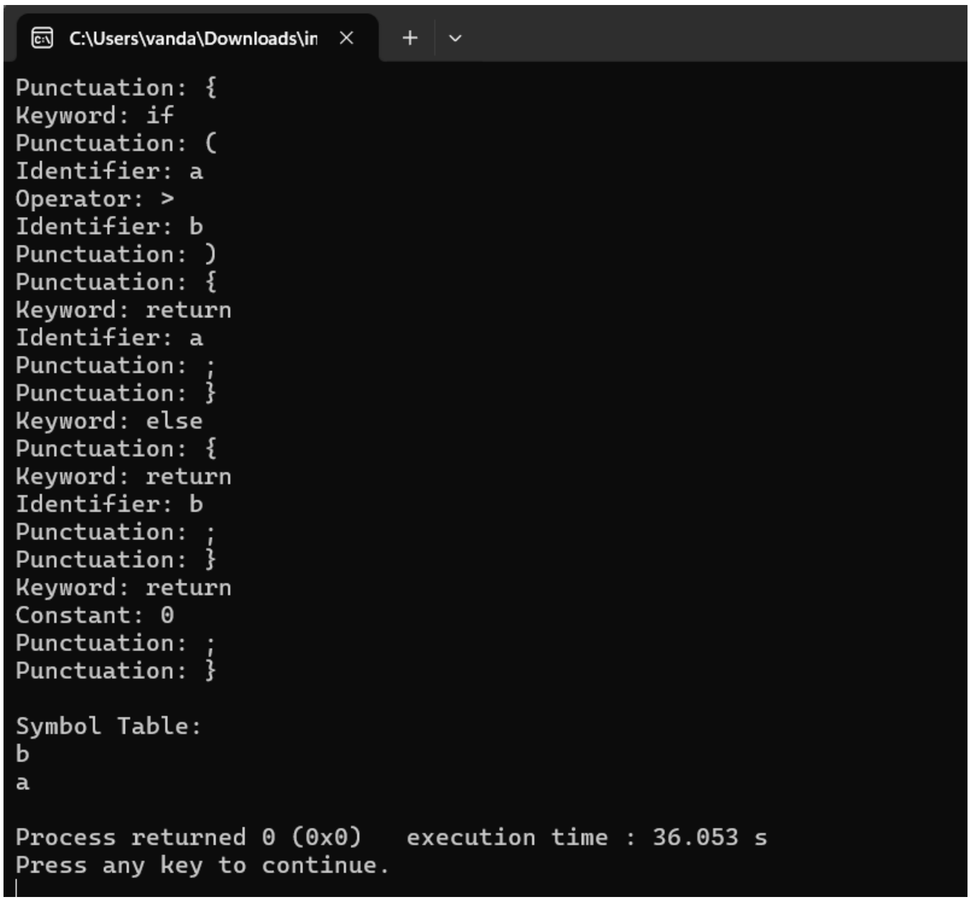
```
  for (size_t i = 0; i < code.length(); i++) {
     char c = code[i];
      if (isspace(c)) continue;
   if (isalpha(c) || c == '_') {
         token.clear();
         while (isalnum(code[i]) || code[i] == '_') {
            token += code[i];
            i++;
         }
         i--;
         if (isKeyword(token)) {
            cout << "Keyword: " << token << endl;
         }
         else if (token != "main") {
            symbolTable.insert(token);
            cout << "Identifier: " << token << endl;
         }
          else {
            cout << "Identifier: " << token << endl;
         }
      }
      else if (isdigit(c)) {
         token.clear();
         while (isalnum(code[i])) {
            token += code[i];
            i++;
         }
         i--;
         if (regex_match(token, regex("[0-9]+"))) {
            cout << "Constant: " << token << endl;
         }
         else {
            cout << "Lexical Error: " << token << " invalid lexeme" << endl;
         }
      }

     else if (c == '\'') {
         token.clear();
         token += c;
         i++;
```

```
            if (code[i] != '\'' || code[i + 1] != ';') {
                token += code[i];
                if (code[i + 1] == '\'') {
                    token += code[i + 1];
                    i++;
                    cout << "String: " << token << endl;
                }
                else {
                    cout << "Lexical Error: Invalid character literal" << endl;
                }
            }
            i++;
        }
        else if (isOperator(c)) {
            token.clear();
            token += c;
            if (isOperator(code[i + 1])) {
                token += code[i + 1];
                i++;
            }
            cout << "Operator: " << token << endl;
        }
        else if (ispunct(c)) {
            cout << "Punctuation: " << c << endl;
        }

        else {
            cout << "Lexical Error: " << c << " invalid lexeme" << endl;
        }
    }
}




int main()
{

    string code , line;
    cout<<"Enter a c code and at last write to END to terminate"<<endl;
    while (getline(cin, line)) {
        if (line == "END") {
            break;
        }
        code += line + "\n";
    }
    cout << "\nTokenized Output:\n";
    tokenize(code);

    cout << "\nSymbol Table:\n";
    for (const auto& entry : symbolTable) {
        cout << entry << endl;
    }

    return 0;
}
```

Output :-



```
Enter a c code and at last write to END to terminate
int main()
{
if(a>b)
{
return a;
}
else
{
return b;
}
return 0;
}
END

Tokenized Output:
Keyword: int
Identifier: main
Punctuation: (
Punctuation: )
Punctuation: {
Keyword: if
Punctuation: (
Identifier: a
Operator: >
Identifier: b
Punctuation: )
Punctuation: {
Keyword: return
Identifier: a
```



```
Punctuation: {
Keyword: if
Punctuation: (
Identifier: a
Operator: >
Identifier: b
Punctuation: )
Punctuation: {
Keyword: return
Identifier: a
Punctuation: ;
Punctuation: }
Keyword: else
Punctuation: {
Keyword: return
Identifier: b
Punctuation: ;
Punctuation: }
Keyword: return
Constant: 0
Punctuation: ;
Punctuation: }

Symbol Table:
b
a

Process returned 0 (0x0)   execution time : 36.053 s
Press any key to continue.
```