```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cctype>
#include <unordered_set>

using namespace std;

enum class TokenType {
    KEYWORD,
    IDENTIFIER,
    LITERAL,
    OPERATOR,
    SEPARATOR,
    UNKNOWN
};

struct Token {
    TokenType type;
    string value;

    Token(TokenType type, string value) : type(type), value(value) {}
};

unordered_set<string> keywords = {
    "int", "float", "double", "char", "if", "else", "for", "while", "return", "void", "main"
};
unordered_set<char> operators = {
    '+', '-', '*', '/', '=', '<', '>', '!', '&', '|'
};
unordered_set<char> separators = {
    '(', ')', '{', '}', ';', ',', '[', ']'
};

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

bool isOperator(char ch) {
    return operators.find(ch) != operators.end();
}

bool isSeparator(char ch) {
    return separators.find(ch) != separators.end();
}

vector<Token> tokenize(const string &code) {
    vector<Token> tokens;
    string buffer;
    int length = code.length();

    for (int i = 0; i < length; ++i) {
        char ch = code[i];

        if (isspace(ch)) {
            continue;
        }

        if (isSeparator(ch)) {
            tokens.push_back(Token(TokenType::SEPARATOR, string(1, ch)));
            continue;
        }
```

```
        if (isOperator(ch)) {
            tokens.push_back(Token(TokenType::OPERATOR, string(1, ch)));
            continue;
        }

        if (isalpha(ch) || ch == '_') {
            buffer.clear();
            while (isalnum(ch) || ch == '_') {
                buffer += ch;
                ch = code[++i];
            }
            --i;

            if (isKeyword(buffer)) {
                tokens.push_back(Token(TokenType::KEYWORD, buffer));
            } else {
                tokens.push_back(Token(TokenType::IDENTIFIER, buffer));
            }
            continue;
        }

        if (isdigit(ch)) {
            buffer.clear();
            while (isdigit(ch) || ch == '.') {
                buffer += ch;
                ch = code[++i];
            }
            --i;

            tokens.push_back(Token(TokenType::LITERAL, buffer));
            continue;
        }

        tokens.push_back(Token(TokenType::UNKNOWN, string(1, ch)));
    }

    return tokens;
}

void printTokens(const vector<Token> &tokens) {
    for (const auto &token : tokens) {
        string type;
        switch (token.type) {
            case TokenType::KEYWORD: type = "KEYWORD"; break;
            case TokenType::IDENTIFIER: type = "IDENTIFIER"; break;
            case TokenType::LITERAL: type = "LITERAL"; break;
            case TokenType::OPERATOR: type = "OPERATOR"; break;
            case TokenType::SEPARATOR: type = "SEPARATOR"; break;
            default: type = "UNKNOWN"; break;
        }
        cout << "<" << type << ", " << token.value << ">" << endl;
    }
}

int main() {
    string filename = "test.cpp";

    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Could not open the file " << filename << endl;
        return 1;
    }
```

```
    string code((istreambuf_iterator<char>(file)), istreambuf_iterator<char>());
    file.close();

    vector<Token> tokens = tokenize(code);
    printTokens(tokens);

    return 0;
}
```

OUTPUT:

```
<UNKNOWN, #>
<IDENTIFIER, include>
<OPERATOR, <>
<IDENTIFIER, iostream>
<OPERATOR, >>
<IDENTIFIER, using>
<IDENTIFIER, namespace>
<IDENTIFIER, std>
<SEPARATOR, ;>
<KEYWORD, int>
<KEYWORD, main>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, {>
<IDENTIFIER, cout>
<OPERATOR, <>
<OPERATOR, <>
<UNKNOWN, '>
<IDENTIFIER, Hello>
<IDENTIFIER, World>
<UNKNOWN, '>
<OPERATOR, <>
<OPERATOR, <>
<IDENTIFIER, endl>
<SEPARATOR, ;>
<KEYWORD, return>
<LITERAL, 0>
<SEPARATOR, ;>
<SEPARATOR, }>
```