```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <regex>
#include <sstream>
#include <iomanip>

using namespace std;

// Definición de un token
struct Token {
    string type;
    string value;
    int line;
    int column;
};

// Función para tokenizar el código
vector<Token> tokenize(const string& code) {
    vector<Token> tokens;
    regex token_specification(
        R"((\bint\b)|(\b[a-zA-Z_][a-zA-Z_0-9]*\b)|(\b\d+\b)|(=)|(\+)|(;)|(\n)|(\s+)|(.) )"
    );

    int line_num = 0;
    int line_start = 0;

    auto words_begin = sregex_iterator(code.begin(), code.end(), token_specification);
    auto words_end = sregex_iterator();
```

```
    for (sregex_iterator i = words_begin; i != words_end; ++i) {
        smatch match = *i;
        string value = match.str(0);
        int column = match.position(0) - line_start;

        if (match[7].matched) { // Nueva línea
            line_start = match.position(0) + match.length(0);
            line_num++;
        } else if (!match[8].matched && !match[9].matched) { // Omitir espacios y caracteres no reconocidos
            string type;
            if (match[1].matched) type = "int";
            else if (match[2].matched) type = "IDENTIFIER";
            else if (match[3].matched) type = "NUMBER";
            else if (match[4].matched) type = "=";
            else if (match[5].matched) type = "+";
            else if (match[6].matched) type = ";";
            tokens.push_back({type, value, line_num, column});
        }
    }

    return tokens;
}

// Función para crear la tabla hash
unordered_map<string, string> create_hash_table(const vector<Token>& tokens) {
    unordered_map<string, string> hash_table;
    for (const auto& token : tokens) {
```

```
        stringstream key;
        key << setw(2) << setfill('0') << token.line << setw(2) << setfill('0') << token.column;
        hash_table[key.str()] = token.value;
    }
    return hash_table;
}

// Función para mostrar el menú
void show_menu() {
    cout << "Menu:" << endl;
    cout << "1. INGRESAR CODIGO" << endl;
    cout << "2. SALIR" << endl;
    cout << "SELECCIONE LA OPCION: ";
}

int main() {
    int choice;
    string code;

    do {
        show_menu();
        cin >> choice;
        cin.ignore(); // Ignorar el carácter de nueva línea después de la opción

        switch (choice) {
            case 1:
                cout << "INGRESE CODIGO:" << endl;
                code.clear();
                while (true) {
```

```
            string line;
            getline(cin, line);
            if (line.empty()) break;
            code += line + '\n';
        }

        // Tokenizar el código
        {
            vector<Token> tokens = tokenize(code);

            // Crear la tabla hash
            unordered_map<string, string> hash_table = create_hash_table(tokens);

            // Imprimir la tabla hash
            cout << left << setw(10) << "Clave" << "Token" << endl;
            for (const auto& pair : hash_table) {
                cout << left << setw(10) << pair.first << pair.second << endl;
            }
        }
        break;
    case 2:
        cout << "Saliendo..." << endl;
        break;
    default:
        cout << "Opción no válida. Intente nuevamente." << endl;
        break;
  }
} while (choice != 2);
```

```
    return 0;
}
```