AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Choose an item.

Assignment Cover Sheet

Assignment Title:	Create Token		
Assignment No:	Final term final task	Date of Submission:	5 April 2024
Course Title:	Compiler design		
Course Code:	Click here to enter text.	Section:	A
Semester:	Spring 2023-24	Course Teacher:	**NAZMUS SAKIB SHAN**

Group Name/No.:

No	Name	ID	Program	Signature
1			Choose an item.	
2			Choose an item.	
3			Choose an item.	
4			Choose an item.	
5			Choose an item.	
6			Choose an item.	
7			Choose an item.	
8			Choose an item.	
9			Choose an item.	
10			Choose an item.	

Faculty use only		
FACULTYCOMMENTS	Marks Obtained	
	Total Marks	

CODE:

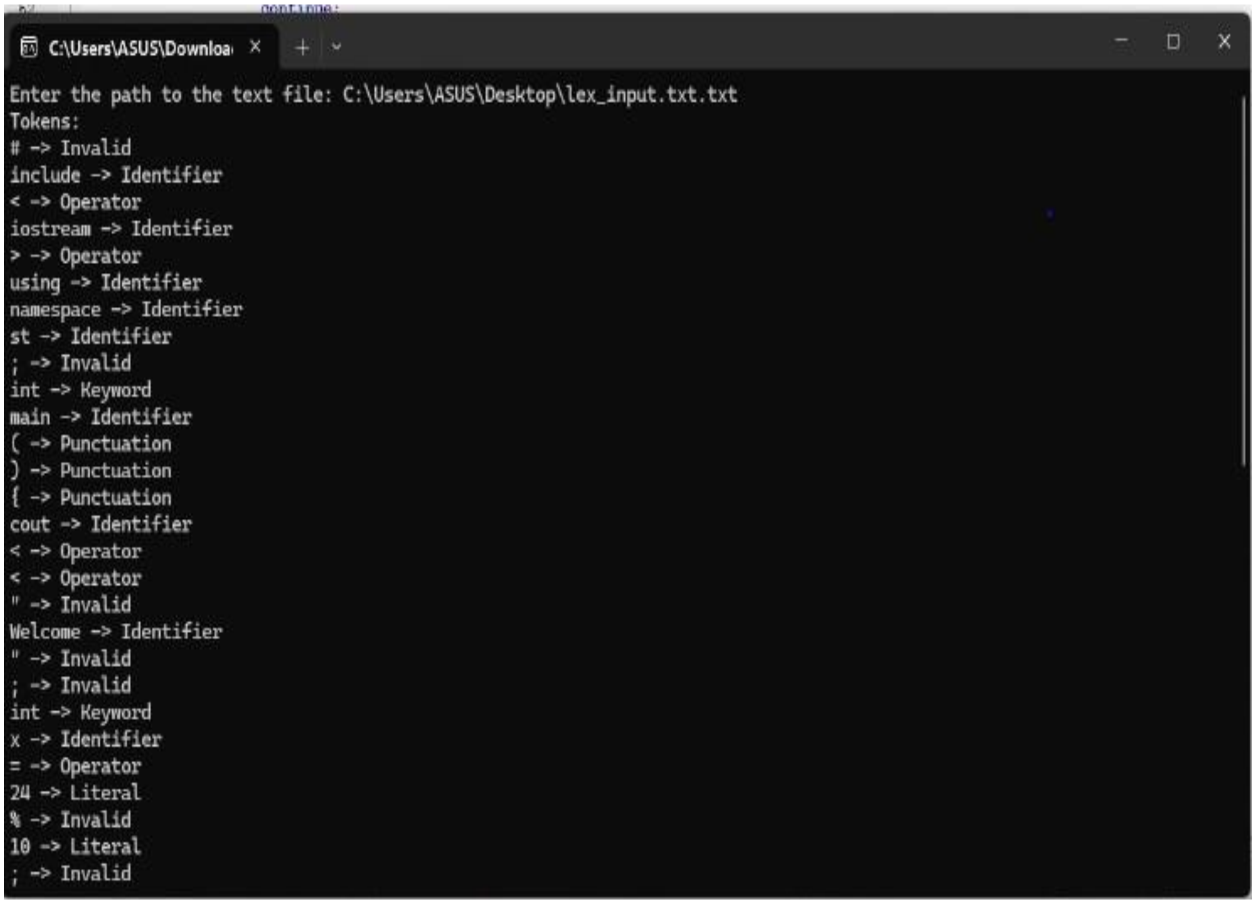
```
 #include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <unordered_set>

#include <cctype>


using namespace std;


enum class TokenType {

   IDENTIFIER,

   KEYWORD,

   OPERATOR,

   LITERAL,

   PUNCTUATION,

   COMMENT,

   INVALID

};
```

```
unordered_set<string> keywords = {"int", "float", "if", "else", "for", "while", "return"};
unordered_set<string> operators = {"+", "-", "*", "/", "=", "==", "<", ">", "<=", ">=", "!="};

TokenType getTokenType(const string& token) {
   if (keywords.find(token) != keywords.end()) {
      return TokenType::KEYWORD;
   } else if (operators.find(token) != operators.end()) {
      return TokenType::OPERATOR;
   } else if (token == "(" || token == ")" || token == "{" || token == "}") {
      return TokenType::PUNCTUATION;
   } else if (isdigit(token[0])) {
      return TokenType::LITERAL;
   } else if (isalpha(token[0]) || token[0] == '_') {
      return TokenType::IDENTIFIER;
   } else {
      return TokenType::INVALID;
   }
}

vector<string> tokenize(const string& code) {
   vector<string> tokens;
   string token;
   bool inComment = false;

   for (char c : code) {
        }

        if (inComment) {
            if (c == '\n') {
                inComment = false;
            }
            continue;
        }

        if (isspace(c) || ispunct(c)) {
            if (!token.empty()) {
                tokens.push_back(token);
                token.clear();
            }
            if (c == '/') {
                if (!tokens.empty() && tokens.back() == "/") {
                    tokens.pop_back();
                    inComment = true;
                    continue;
                }
            }
            if (!isspace(c)) {
                tokens.push_back(string(1, c));
            }
        } else {
            token += c;
        }
    }
    if (!token.empty()) {
        tokens.push_back(token);
    }

    return tokens;
}

int main() {
    string filePath;
    cout << "Enter the path to the text file: ";
    getline(cin, filePath);

    ifstream file(filePath);
    if (!file.is_open()) {
        cerr << "Error opening file." << endl;
        return 1;
    }

    string code;
    string line;
    while (getline(file, line)) {
        code += line + '\n';
    }

    vector<string> tokens = tokenize(code);
```

```
cout << "Tokens:" << endl;
for (const auto& token : tokens) {
  cout << token << " -> ";
  switch (getTokenType(token)) {
    case TokenType::IDENTIFIER:
      cout << "Identifier";
      break;
    case TokenType::KEYWORD:
      cout << "Keyword";
      break;
    case TokenType::OPERATOR:
      cout << "Operator";
      break;
    case TokenType::LITERAL:
      cout << "Literal";
      break;
    case TokenType::PUNCTUATION:
      cout << "Punctuation";
      break;
    case TokenType::COMMENT:
      cout << "Comment";
      break;
    case TokenType::INVALID:
      cout << "Invalid";
      break;
```

```
        }
        cout << endl;
    }

    file.close();
    return 0;
}
```

OUTPUT:



C:\Users\ASUS\Downloa

```
Enter the path to the text file: C:\Users\ASUS\Desktop\lex_input.txt.txt
Tokens:
# -> Invalid
include -> Identifier
< -> Operator
iostream -> Identifier
> -> Operator
using -> Identifier
namespace -> Identifier
st -> Identifier
; -> Invalid
int -> Keyword
main -> Identifier
( -> Punctuation
) -> Punctuation
{ -> Punctuation
cout -> Identifier
< -> Operator
< -> Operator
" -> Invalid
Welcome -> Identifier
" -> Invalid
; -> Invalid
int -> Keyword
x -> Identifier
= -> Operator
24 -> Literal
% -> Invalid
10 -> Literal
; -> Invalid
```