COMPILER LAB

REPORT ASSIGNMENT-1

NAME: SOHAM LAHIRI

CLASS: BCSE UG-III 6TH SEMESTER

ROLL NO: 002210501107

GROUP: A3

SUBMISSION DATE: 30/01/2025

Problem Statement: Write a program that will accept a 'C' code as input, and output a stream of tokens with tokens along with their classes (for example operator, identifier, constant etc) and the position (row and column) of each token in the 'C' code.

This C++ program performs lexical analysis on a C source file (targetc.c) by identifying and categorizing tokens such as keywords, operators, punctuation marks, and identifiers. Additionally, it generates a symbol table file (symboltable.txt). The program utilizes external files (keywords.txt, punctuations.txt, and operators.txt) to recognize language-specific elements.

Components of the Program
The program consists of several functions:

1. **File Processing Functions**
    - buildKeywordSet(bool keywordSet[256])
    - buildPunctuationSet(bool punctuationSet[256])
    - buildOperatorSet(bool operatorSet[256])
2. **Tokenization and Lexical Analysis Function**
    - computeTokens()
3. **Symbol Table Generation Function**
    - generateSymbolTableFile()
4. **Main Function**
    - Calls the helper functions and initiates lexical analysis.

Working:

Step 1: Building Keyword, Operator, and Punctuation Sets

The program starts by reading predefined lists of keywords, operators, and punctuation symbols from external text files and stores them in boolean arrays of size 256.

- buildKeywordSet() reads keywords.txt and marks the first character of each keyword in keywordSet as true.
- buildPunctuationSet() reads punctuations.txt and marks the corresponding character in punctuationSet.
- buildOperatorSet() reads operators.txt and marks the corresponding character in operatorSet.

This approach enables quick lookup operations to determine whether a character belongs to a specific category.

Step 2: Processing the Input File (targetc.c)

The computeTokens() function reads targetc.c line by line and processes it to extract tokens. The extracted tokens are classified into the following categories:

1. **Keywords**: Recognized using isKeyword(token, keywordSet), which checks if the first character of the token is present in keywordSet.
2. **Identifiers**: If a token is not recognized as a keyword, it is classified as an identifier.
3. **Operators**: Operators are detected using operatorSet and include both single-character (+, -, *, /, =) and compound operators (+=, -=, *=, /=, ++, --).
4. **Punctuation**: Characters like ;, {, }, ,, etc., are identified using punctuationSet.

For each identified token, the program prints relevant information:

- **Token value**
- **Token type (keyword, identifier, operator, punctuation)**
- **Row number**
- **Column number**

Step 3: Handling Compound Operators

The program checks for compound operators by examining pairs of consecutive characters. If a compound operator is found, it is printed as a single token, and the index is incremented to avoid double processing.

Step 4: Generating the Symbol Table

The generateSymbolTableFile() function creates an output file named symboltable.txt. While the current implementation does not store symbols, it provides a placeholder for writing symbol-related information extracted from the tokenization process.

Step 5: Execution Flow in main()

The main() function performs the following operations:

1. Initializes boolean arrays for keyword, punctuation, and operator sets.
2. Calls buildKeywordSet(), buildPunctuationSet(), and buildOperatorSet() to populate these sets.
3. Invokes computeTokens() to analyze targetc.c and output tokens.
4. Calls generateSymbolTableFile() to create the symbol table file.

Implementation:

ASSIGNMENT1.cpp:

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;

// Constants for file names
const string keywordsFile = "keywords.txt";
const string punctuationsFile = "punctuations.txt";
const string operatorsFile = "operators.txt";
const string inputFile = "targetc.c";
const string symbolTableFile = "symboltable.txt";

// Helper functions to retrieve data from files and store them as arrays or hash maps

void buildKeywordSet(bool keywordSet[256]) {
    ifstream fi(keywordsFile);
    string word;
    while (fi >> word) {
        keywordSet[word[0]] = true;
    }
    fi.close();
}

void buildPunctuationSet(bool punctuationSet[256]) {
    ifstream fi(punctuationsFile);
    char ch;
    while (fi >> ch) {
        punctuationSet[ch] = true;
    }
    fi.close();
}

void buildOperatorSet(bool operatorSet[256]) {
    ifstream fi(operatorsFile);
    char ch;
    while (fi >> ch) {
        operatorSet[ch] = true;
    }
    fi.close();
}

// Helper function to check if the string is a valid keyword or identifier
```

```
bool isKeyword(const string &token, bool keywordSet[256]) {
   return keywordSet[token[0]];
}

void computeTokens() {
   ifstream fi(inputFile);
   string str;
   int row = 1;
   bool keywordSet[256] = {false}; // Array to track keyword characters
   bool punctuationSet[256] = {false}; // Array to track punctuation characters
   bool operatorSet[256] = {false}; // Array to track operator characters

   // Build the sets from files
   buildKeywordSet(keywordSet);
   buildPunctuationSet(punctuationSet);
   buildOperatorSet(operatorSet);

   // Output header for token table
   cout << "--------------------------------------------------------------------\n";
   printf("\tToken \t\tType \t\tRow \t\tColumn \n");
   cout << "--------------------------------------------------------------------\n";

   // Process each line from the input file
   while (getline(fi, str)) {
      string token;
      int startidx = -1, endidx = -1;
      for (int i = 0; i <= str.size(); ++i) {
         // Tokenize the string based on spaces or end of line
         if (i == str.size() || str[i] == ' ') {
            if (startidx != -1) {
               token = str.substr(startidx, endidx - startidx + 1);
               int col = startidx + 1;
               if (isKeyword(token, keywordSet)) {
                  cout << "\t" << token << "\t\tkeyword \t" << row << "\t\t" << col << "\n";
               } else {
                  cout << "\t" << token << "\t\tidentifier \t" << row << "\t\t" << col << "\n";
               }
            }
            startidx = -1;
         }
         // Check for compound operators
         else if (i + 1 < str.size() &&
                 ((str[i] == '+' && str[i + 1] == '+') ||
                  (str[i] == '-' && str[i + 1] == '-') ||
                  (str[i] == '+' && str[i + 1] == '=') ||
                  (str[i] == '-' && str[i + 1] == '=') ||
                (str[i] == '*' && str[i + 1] == '=') ||
                (str[i] == '/' && str[i + 1] == '='))) {
            if (startidx != -1) {
                token = str.substr(startidx, endidx - startidx + 1);
                int col = startidx + 1;
                cout << "\t" << token << "\t\tidentifier \t" << row << "\t\t" << col << "\n";
            }
            token = str.substr(i, 2);
            int col = i + 1;
            cout << "\t" << token << "\t\toperator \t" << row << "\t\t" << col << "\n";
            i++; // Skip the next character since it's part of the compound operator
            startidx = -1;
        }
        else if (operatorSet[str[i]]) {
            if (startidx != -1) {
                token = str.substr(startidx, endidx - startidx + 1);
                int col = startidx + 1;
                cout << "\t" << token << "\t\tidentifier \t" << row << "\t\t" << col << "\n";
            }
            token = str[i];
            int col = i + 1;
            cout << "\t" << token << "\t\toperator \t" << row << "\t\t" << col << "\n";
            startidx = -1;
        }
        else if (punctuationSet[str[i]]) {
            if (startidx != -1) {
                token = str.substr(startidx, endidx - startidx + 1);
                int col = startidx + 1;
                cout << "\t" << token << "\t\tidentifier \t" << row << "\t\t" << col << "\n";
            }
            token = str[i];
            int col = i + 1;
            cout << "\t" << token << "\t\tpunctuation \t" << row << "\t\t" << col << "\n";
            startidx = -1;
        } else if (startidx == -1) {
            startidx = endidx = i;
        } else {
            endidx = i;
        }
    }
    row++;
  }
  cout << "--------------------------------------------------------------------\n";
  fi.close();
}
```

```
void generateSymbolTableFile() {
    ofstream fo(symbolTableFile, ios::out | ios::trunc);
    if (fo.is_open()) {
        fo << "Symbol Table Content: \n";
        // Write symbols to the file here...
        // This section assumes symbols are stored in an array/list
        // and symbols are collected during the tokenization process.
    }
    fo.close();
}

int main() {
    // Initialize arrays for sets
    bool keywordSet[256] = {false};
    bool punctuationSet[256] = {false};
    bool operatorSet[256] = {false};

    // Build sets for keywords, punctuation, and operators
    buildKeywordSet(keywordSet);
    buildPunctuationSet(punctuationSet);
    buildOperatorSet(operatorSet);

    // Process the input file and generate tokens
    computeTokens();

    // Generate symbol table
    generateSymbolTableFile();

    return 0;
}
```

Keywords.txt:

```
auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
```

for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while

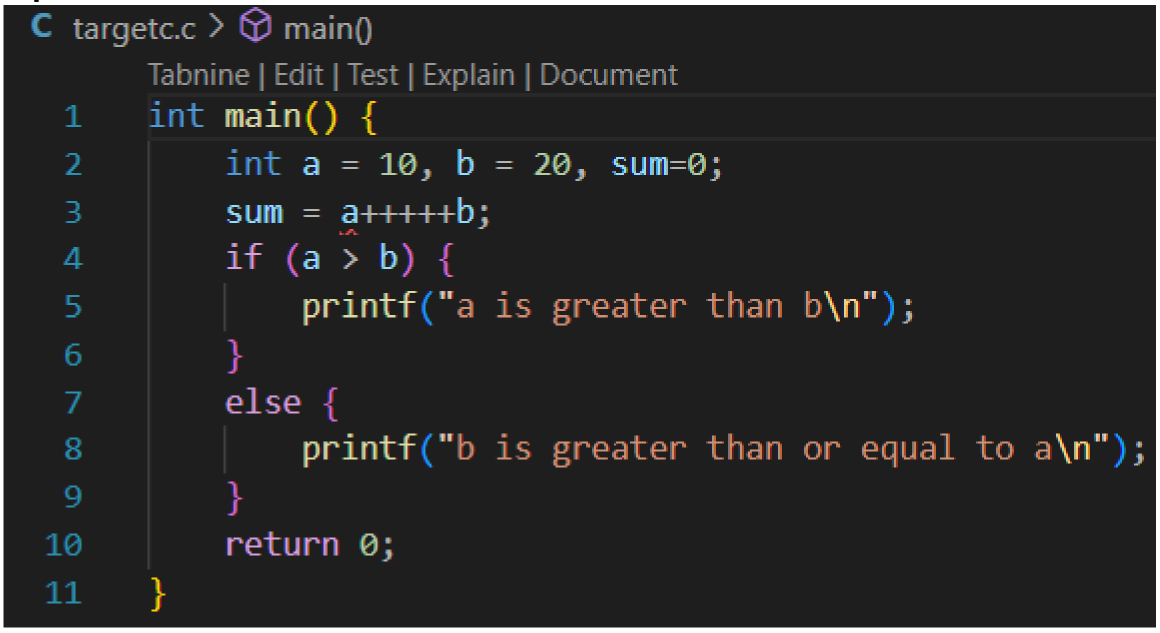
Operators.txt

```
+
-
*
/
%
=
!
>
<
&
|
^
~
?
:
++
--
-=
+=
*=
/=
```

Punctuation.txt

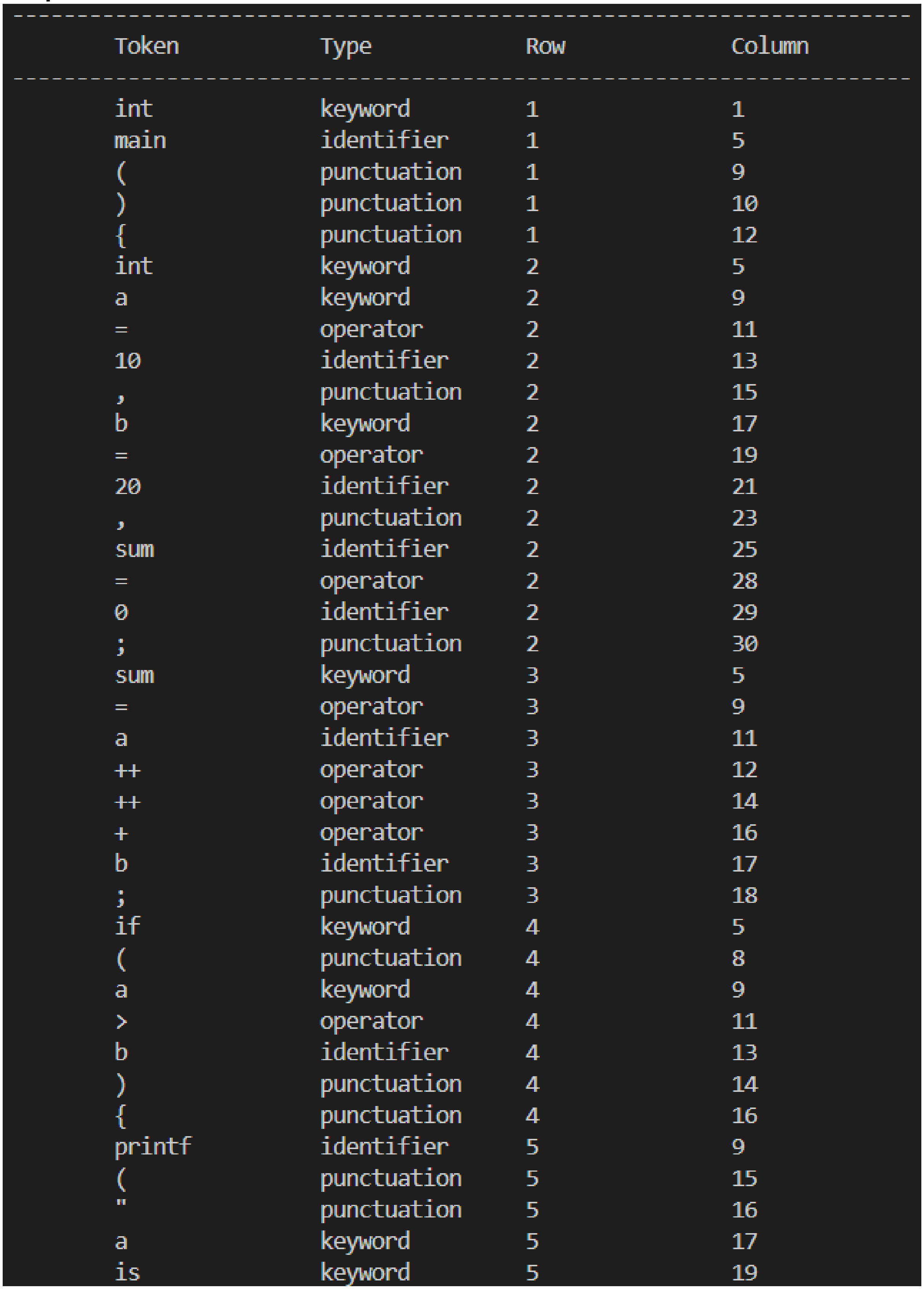
```
!
"
#
$
%
&
'
(
)
*
+
,
-
.
/
:
;
<
=
>
?
@
[
\
]
^
_
`
{
|
}
-
```

Input:

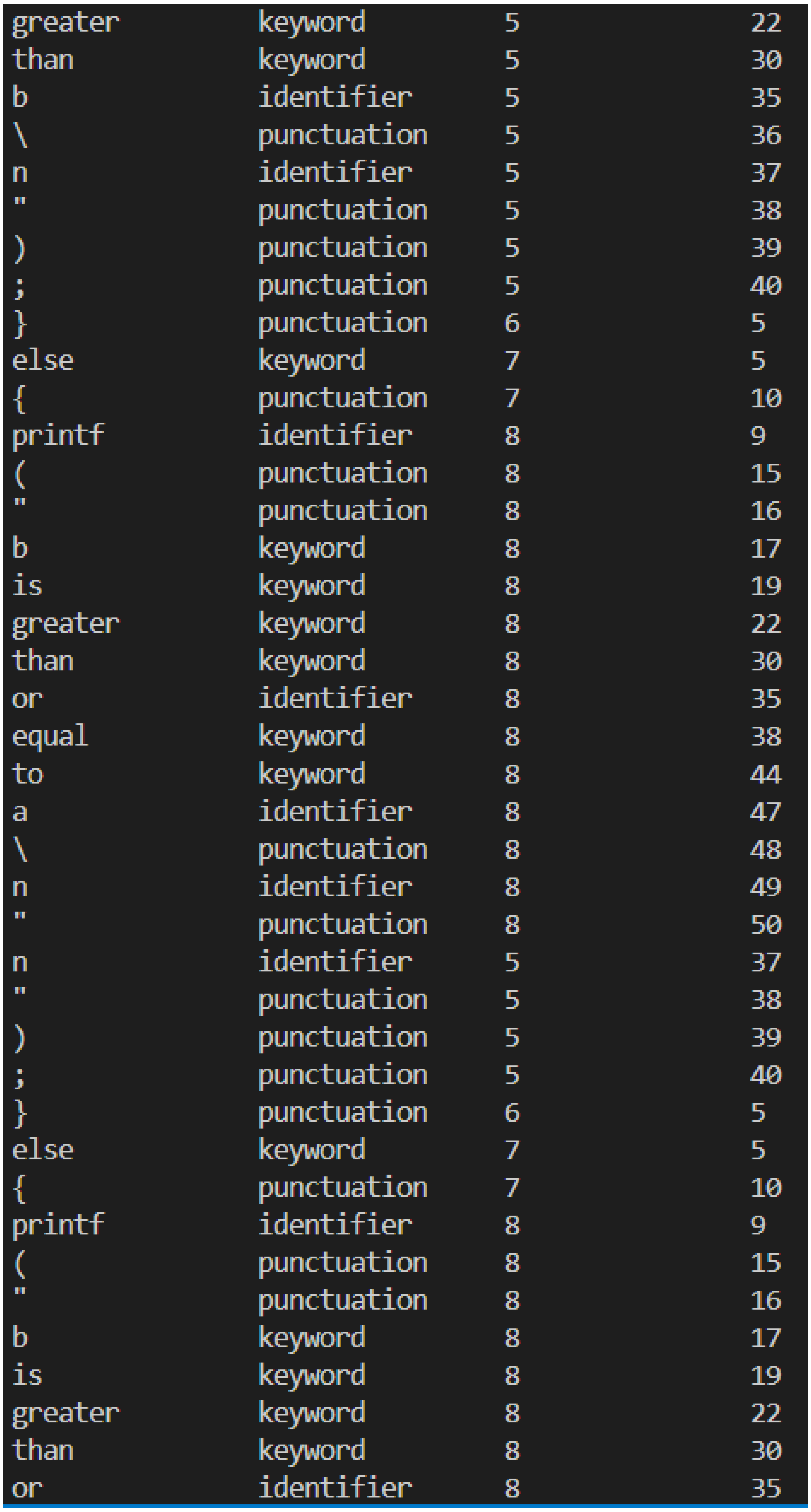


```c
int main() {
    int a = 10, b = 20, sum=0;
    sum = a+++++b;
    if (a > b) {
        printf("a is greater than b\n");
    }
    else {
        printf("b is greater than or equal to a\n");
    }
    return 0;
}
```

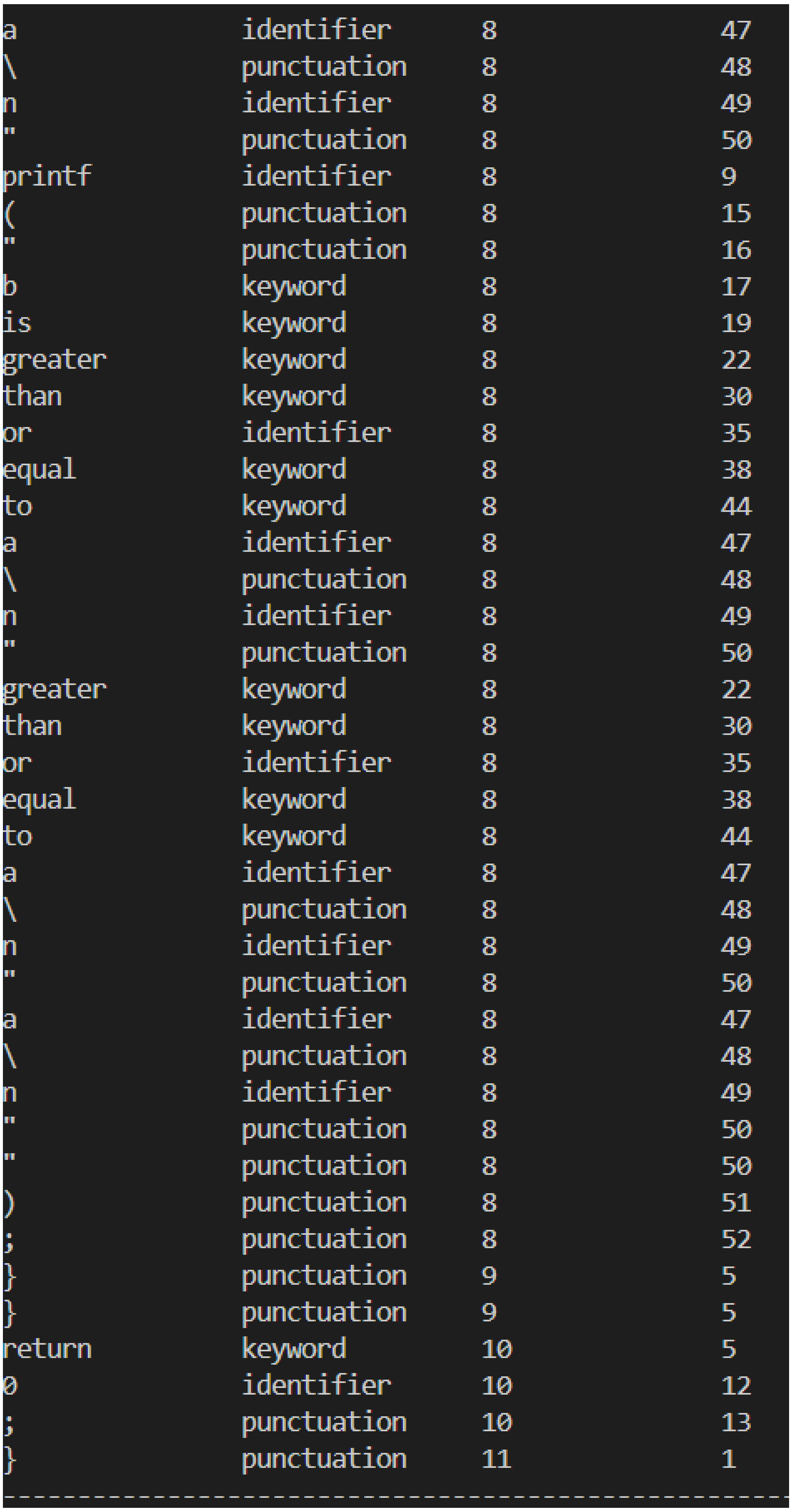
Output:



```
---------------------------------------------------------------------------
        Token           Type            Row             Column
---------------------------------------------------------------------------
        int             keyword         1               1
        main            identifier      1               5
        (               punctuation     1               9
        )               punctuation     1               10
        {               punctuation     1               12
        int             keyword         2               5
        a               keyword         2               9
        =               operator        2               11
        10              identifier      2               13
        ,               punctuation     2               15
        b               keyword         2               17
        =               operator        2               19
        20              identifier      2               21
        ,               punctuation     2               23
        sum             identifier      2               25
        =               operator        2               28
        0               identifier      2               29
        ;               punctuation     2               30
        sum             keyword         3               5
        =               operator        3               9
        a               identifier      3               11
        ++              operator        3               12
        ++              operator        3               14
        +               operator        3               16
        b               identifier      3               17
        ;               punctuation     3               18
        if              keyword         4               5
        (               punctuation     4               8
        a               keyword         4               9
        >               operator        4               11
        b               identifier      4               13
        )               punctuation     4               14
        {               punctuation     4               16
        printf          identifier      5               9
        (               punctuation     5               15
        "               punctuation     5               16
        a               keyword         5               17
        is              keyword         5               19
```



```
greater      keyword      5       22
than         keyword      5       30
b            identifier   5       35
\            punctuation  5       36
n            identifier   5       37
"            punctuation  5       38
)            punctuation  5       39
;            punctuation  5       40
}            punctuation  6       5
else         keyword      7       5
{            punctuation  7       10
printf       identifier   8       9
(            punctuation  8       15
"            punctuation  8       16
b            keyword      8       17
is           keyword      8       19
greater      keyword      8       22
than         keyword      8       30
or           identifier   8       35
equal        keyword      8       38
to           keyword      8       44
a            identifier   8       47
\            punctuation  8       48
n            identifier   8       49
"            punctuation  8       50
n            identifier   5       37
"            punctuation  5       38
)            punctuation  5       39
;            punctuation  5       40
}            punctuation  6       5
else         keyword      7       5
{            punctuation  7       10
printf       identifier   8       9
(            punctuation  8       15
"            punctuation  8       16
b            keyword      8       17
is           keyword      8       19
greater      keyword      8       22
than         keyword      8       30
or           identifier   8       35
```



```
a               identifier      8               47
\               punctuation     8               48
n               identifier      8               49
"               punctuation     8               50
printf          identifier      8               9
(               punctuation     8               15
"               punctuation     8               16
b               keyword         8               17
is              keyword         8               19
greater         keyword         8               22
than            keyword         8               30
or              identifier      8               35
equal           keyword         8               38
to              keyword         8               44
a               identifier      8               47
\               punctuation     8               48
n               identifier      8               49
"               punctuation     8               50
greater         keyword         8               22
than            keyword         8               30
or              identifier      8               35
equal           keyword         8               38
to              keyword         8               44
a               identifier      8               47
\               punctuation     8               48
n               identifier      8               49
"               punctuation     8               50
a               identifier      8               47
\               punctuation     8               48
n               identifier      8               49
"               punctuation     8               50
"               punctuation     8               50
)               punctuation     8               51
;               punctuation     8               52
}               punctuation     9               5
}               punctuation     9               5
return          keyword         10              5
0               identifier      10              12
;               punctuation     10              13
}               punctuation     11              1
-----------------------------------------------------
```