Evaluation of the code

1. **Global Definitions and Utility Functions**

```
KEYWORDS = {'int', 'float', 'if', 'else', 'while', 'return'}
OPERATORS = {'+', '-', '*', '/', '=', '==', '!=', '&&', '||'}
SEPARATORS = {';', ',', '(', ')', '{', '}'}
```

Purpose: Defines the categories (keywords, operators, separators) to help classify tokens.

Effectiveness:
- These sets enable quick identification of tokens based on predefined categories.

2. **`is_identifier` Function**

```
def is_identifier(token):
    if token and (token[0].isalpha() or token[0] == '_'):
        return all(char.isalnum() or char == '_' for char in token[1:])
    return False
```

Purpose: Checks if a token is a valid identifier.

Effectiveness:
- Effectively distinguishes valid identifiers that start with an alphabetic character or underscore and contain only alphanumeric characters or underscores.

3. **`is_integer_literal` Function**

```
def is_integer_literal(token):
    return token.isdigit()
```

Purpose: Verifies if a token is an integer literal.

Effectiveness:
- A simple and efficient check for integer literals by confirming that all characters in the token are digits.

4. **`is_float_literal` Function**

```
def is_float_literal(token):
    if '.' in token:
        parts = token.split('.')
        return len(parts) == 2 and all(part.isdigit() for part in parts)
    return False
```

Purpose: Determines if a token represents a floating-point literal.

Effectiveness:
- Verifies the presence of a single decimal point and checks that both parts of the split token are numeric.

5. **`is_string_literal` Function**

```
def is_string_literal(token):
    return token.startswith('"') and token.endswith('"')
```

Purpose: Identifies tokens that are string literals.

Effectiveness:
- Ensures that the token begins and ends with double quotes, handling only single-line strings.

6. **`categorize_token` Function**

```
def categorize_token(token):
    if token in KEYWORDS:
        return f"{token} (keyword)"
    elif token in OPERATORS:
        return f"{token} (operator)"
    elif token in SEPARATORS:
        return f"{token} (separator)"
    elif is_identifier(token):
        return f"{token} (identifier)"
    elif is_integer_literal(token):
        return f"{token} (integer literal)"
    elif is_float_literal(token):
        return f"{token} (floating-point literal)"
    elif is_string_literal(token):
        return f"{token} (string literal)"
    return f"{token} (unknown)"
```

Purpose: Categorizes tokens based on predefined types or utility functions.

Effectiveness:
- The function follows a structured flow from specific types (keywords, operators) to general types (identifiers, literals), ending with `"unknown"` for unclassified tokens.

7. **`tokenize_and_categorize` Function**

```
def tokenize_and_categorize(input_string):
    tokens = []
    current_token = ""
```

```
    for char in input_string:
        if char.isspace():
            if current_token:
                tokens.append(current_token)
                current_token = ""
        elif char in OPERATORS or char in SEPARATORS:
            if current_token:
                tokens.append(current_token)
                current_token = ""
            tokens.append(char)
        else:
            current_token += char

    if current_token:
        tokens.append(current_token)

    categorized_tokens = [categorize_token(token) for token in tokens]
    return categorized_tokens
```

Purpose: Splits the input string into tokens and categorizes each.

Effectiveness:
- Handles tokenization and categorization efficiently, allowing for separation based on spaces, operators, and separators.

8. **`process_file` Function**

```
def process_file(filename):
    with open(filename, 'r') as file:
        input_code = file.read()

    categorized_tokens = tokenize_and_categorize(input_code)
    for token in categorized_tokens:
        print(token)
```

Purpose: Reads the input file and processes each line to extract and categorize tokens.

Effectiveness:
- Effective for reading, tokenizing, and printing categorized tokens, with `with open` ensuring safe file handling.

Output: