Universitat Pompeu Fabra Barcelona

# P2 - Scanner: User Manual

## Introduction

---

Welcome to the C Scanner User Manual! This document will guide you through using and testing the C Scanner program effectively. The C Scanner is designed to efficiently recognize various token categories within C code, including identifiers, keywords, numbers, literals, special characters, and operands. This manual will provide you with step-by-step instructions on how to interact with the program and conduct thorough testing.

## System Requirements

---

- Operating System: Linux, macOS, or Windows (with a compatible terminal emulator)
- C Compiler: GCC (GNU Compiler Collection)

## Installation

---

1. Ensure that you have GCC installed on your system. If not, you can install it through your package manager or by downloading it from the official GCC website.
2. Download the source code for the C Scanner program.
3. Open a terminal window and navigate to the directory containing the source code.
4. Compile the program using the following command:

```
gcc main.c errors.c utils.c datastructures.c debug.c  -o parser
```

## Usage

---

Once the program has been successfully compiled, you can use it to scan C code files and test its tokenization capabilities. Follow these steps to run the program:

1. Open a terminal window.
2. Navigate to the directory containing the compiled `parser` executable.
3. Run the program with the following command:

```
./parser [path_to_test_file]
```

Replace `[path_to_test_file]` with the path to the C code file you want to scan. For example, if you want to test file test1.c located in the tests folder, the command would be:

```
./parser tests/test1.c
```

**Testing**

---

To ensure that the C Scanner program functions correctly and accurately tokenizes C code, thorough testing is essential. Here's how you can conduct effective testing:

Begin by preparing your test cases, either using the provided files in the tests folder or creating your own. Once you have your test cases ready, run them by executing the parser program with each test case, specifying the path to the test file as described in the "Usage" section of the manual. After running each test case, carefully compare the tokenized output produced by the program against the expected output format. Ensure that each token is correctly identified and categorized according to the predefined patterns and rules. If the program fails to tokenize input correctly, utilize the debug functionality to identify and resolve any issues. Pay close attention to the debug information output by the program to understand its internal processes and pinpoint errors or discrepancies. Iterate the testing process with different test cases and input variations to thoroughly evaluate the program's functionality and reliability. Based on insights gained from testing, make any necessary modifications to the program to improve its performance and accuracy. By following these steps and comparing the program's output against the expected format, you can ensure that the C Scanner program correctly identifies and categorizes tokens within C code files.

Below is an example of the expected output format for the provided C code snippet:

- Input:

```
if (x > 3)
   printf("true");
else
   printf("false");
```

- Expected output:

<if, CAT_KEYWORD><(, CAT_SPECIAL_CHAR><x, CAT_IDENTIFIER><>, CAT_OPERATOR><3, CAT_NUMBER><), CAT_SPECIAL_CHAR><printf, CAT_IDENTIFIER><(, CAT_SPECIAL_CHAR><"true", CAT_LITERAL><), CAT_SPECIAL_CHAR><else, CAT_KEYWORD><printf, CAT_IDENTIFIER><(, CAT_SPECIAL_CHAR><"false", CAT_LITERAL><), CAT_SPECIAL_CHAR>

When we apply the output with the debug mode, the output is the same but the tokens are tidy in their respective line. Moreover, every line is listed with a number. If you want to remove the debug mode, you can change the variable debug_mode by inserting “DEBUG_OFF” in the file debug.c.

- Expected output with debug mode enabled:

1 <int, CAT_TYPE> <n, CAT_IDENTIFIER> <=, CAT_OPERAND> <3, CAT_NUMBER> <;, CAT_SPECIALCHAR>
2

<char, CAT_TYPE> <vect, CAT_IDENTIFIER> <[, CAT_SPECIALCHAR> <2, CAT_NUMBER> <], CAT_SPECIALCHAR> <;, CAT_SPECIALCHAR>
3

<double, CAT_IDENTIFIER> <*, CAT_OPERAND> <vect, CAT_IDENTIFIER> <;, CAT_SPECIALCHAR>
4

<int, CAT_TYPE> <vect, CAT_IDENTIFIER> <[, CAT_SPECIALCHAR> <], CAT_SPECIALCHAR> <=, CAT_OPERAND> <{, CAT_SPECIALCHAR> <1, CAT_NUMBER> <,, CAT_SPECIALCHAR> <2, CAT_NUMBER> <}, CAT_SPECIALCHAR> <;, CAT_SPECIALCHAR>
5

**Conclusion**

---

Congratulations! You have successfully learned how to use and test the C Scanner program. By following the instructions outlined in this manual and conducting thorough testing, you can ensure that the program meets its objectives of providing reliable tokenization capabilities for C code. If you encounter any issues or have questions, refer to the documentation or seek assistance from the project team.