Aranich, Marcel - 251453
Quera, Clàudia - 231197
Villarino, Jorge - 231351
Solans, Arnau - 216530
García, David - 251587
Prat, Ariadna - 251281

System Overview

Firstly, our preprocessor ensures that there is an input program file to process. Then, if there are any flags, it reads and enables them for the different modules to function.
Once the program file is correctly processed as a string of characters, it is passed to the handle_includes module if enabled by the replace directives flag "-d" or the all directives flag "-a".
The preprocessor then utilizes the handle_backslash module, the handle_constants module, and the handle_macros module independently of the flags.
To prepare the remaining code, the preprocessor uses handle_ifdef_endif irrespective of the flags before processing the comments.
Lastly, if enabled by the replace all directives flag "-a" or the replace comments flag "-c", the preprocessor proceeds to utilize the handle.

Detailed System Design

The main function serves as the entry point for the program. It includes necessary header files and module header files. It handles command-line arguments, checking for flags and setting corresponding boolean variables. It opens the source file to be preprocessed and determines its size. It dynamically allocates memory to store the file contents. The program then preprocesses the file contents based on the flags set. Finally, it creates a new preprocessed file, writes the contents to it, and frees the allocated memory. The program terminates with a return value of 0 for successful execution and 1 for any errors encountered.

We have divided our code into the following modules:

- **handle_includes:** The handle includes module will have a handle_program_files function to process includes of the form *#include " "* that will be called first, and then a handle_compiler_files function to process includes of the form *#include* <>. It also will make use of an auxiliary function fetch_directory to search the directory in a

specified path (the common paths for compiler libraries) for a specified file in order to find the files which the preprocessor will substitute the include with.

- **handle_constants:** The handle_constants module includes the handle_constants function, which takes a character array ‘source_code’ and its size as arguments. A ‘line’ variable is defined to store each line of the source code. Then it will iterate through the characters of the source_code until a newline character (\n) is found or the end of the source code is reached. Once a newline character is encountered, it stores the characters of that line into a new variable ‘line’. If the line starts with a #define, it substitutes all occurrences of that identifier within the source code by the replacement value. If the line does not start with #define, it means it is not a constant definition, so the line is printed as it is.

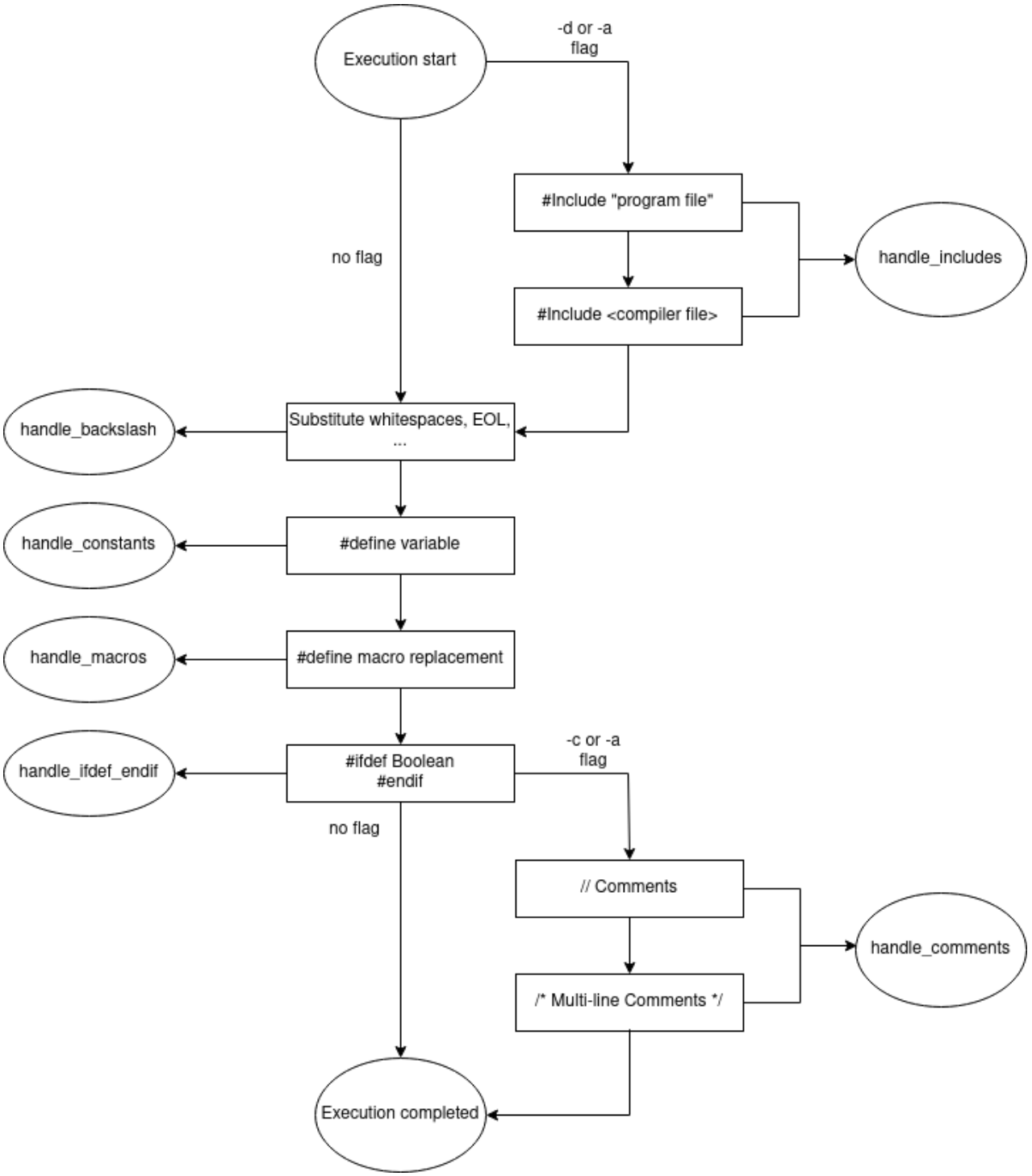
- **handle_macros:** The handle_macros module includes just the handle_macros function. This function will be similar to the handle_constants function as it will also look for lines starting with #define with the difference that in this case, if a #define directive is found, it searches for occurrences of the macro_name in the source_code and replaces them with the macro_body. After handling the macro definitions, the function prints the modified or original lines.

- **handle_comments:** The handle_comments module includes the remove_single_line_comments and the remove_multi_line_comments functions, which will detect and remove all comments in the code, independently of the flags to ensure the handling of comments. The remove_single_line_comments function will iterate through the lines of the code detecting single line comments and comments at the end of lines, and will delete them, and the remove_multi_line_comments function will do the same with multi line comments.

- **handle_ifdef_endif:** The handle_ifdef_endif includes the function “handle_ifdef_endif”. In the function, we begin with a while where we iterate every line. If the function detects the string #ifdef <str>, then we do other while for getting the string #define <str>. After, we compare the two <str>. If they are the same we continue the algorithm, searching the string #endif. If this procedure was correct, we put the structure ifdef-endif in the output version, otherwise not.

Module Interaction

We have included a workflow diagram to illustrate the interactions between the different modules in the program:



Individual Responsibilities

Breakdown of the tasks and roles assumed by each member of the team:

- **Handle includes:** Marcel Aranich and Arnau Solans
- **Handle ifdef and endif:** Ariadna Prat
- **Handle defines (macros and constants):** David Garcia and Clàudia Quera
- **Handle Comments:** Jorge Villarino