Problem 4. GCC argument descriptor

Problem background

To compile C/C++ code, we need to execute a command that calls the compiler with some arguments. For example:

```
gcc hello.c -o hello.exe -std=c17 -Wall -Wpedantic -Wextra
```

This command compiles the C source code file `hello.c` with `gcc`, setting the output file to be `hello.exe`, setting the language standard to be ISO C17, and enabling some kinds of warnings and errors by `-Wall -Wpedantic -Wextra`.

Taking a look at GCC's manual, you will find an overwhelmingly long list of supported arguments. Is there a tool that can generate some explanations of the arguments, or at least for some of the most common ones? For example, suppose the name of that tool is `gcc_descriptor.exe` (without `.exe` on Mac OS X and Linux). It would be helpful if the following command (with `.\` replaced by `./` on Mac OS X and Linux)

```
.\gcc_descriptor hello.c -o hello.exe -std=c17 -Wall -Wpedantic -Wextra
```

could print

```
hello.c: C source code as input file.
-o hello.exe: Place the primary output in file hello.exe.
-std=c17: Set the language standard to ISO C17.
-Wall: Enable all the warnings about constructions that some users consider questionable, and
that are easy to avoid (or modify to prevent the warning).
-Wpedantic: Issue all the warnings demanded by strict ISO C and ISO C++ and reject all programs
that use forbidden extensions.
-Wextra: Enable some extra warning flags that are not enabled by -Wall.
```

Your task is to implement such a tool that generates explanations for some very common GCC arguments.

Problem description

Write a program that **accepts command-line arguments**, which are the arguments that we want to pass to GCC, and print the descriptions for them.

The supported kinds of arguments are as follows.

1. `-Wall`

Description:

```
-Wall: Enable all the warnings about constructions that some users consider questionable, and
that are easy to avoid (or modify to prevent the warning).
```

2. `-Wpedantic`

Description:

```
-Wpedantic: Issue all the warnings demanded by strict ISO C and ISO C++ and reject all programs
that use forbidden extensions.
```

3. `-Wextra`

Description:

```
-Wextra: Enable some extra warning flags that are not enabled by -Wall.
```

4. `-Werror`

Description:

```
-Werror: Make all warnings into errors.
```

5. `-o xxx`

Description:

```
-o xxx: Place the primary output in file xxx.
```

where `xxx` is replaced by the actual file name (see the example below). Note that this option consists of **two arguments**, where the first one is `-o` and the second one is the file name.

6. `-I xxx`

Description:

```
-I xxx: Add the directory xxx to the list of directories to be searched for header files during
preprocessing.
```

where `xxx` is replaced by the actual directory name (see the example below). Note that this option consists of **two arguments**, where the first one is `-I` and the second one is the directory name.

7. `-std=xxx`

Description:

```
-std=xxx: Set the language standard to yyy.
```

where `xxx` is replaced by the actual value (see the example below) and `yyy` is replaced by the name of the standard according to the following table.

`xxx`	`yyy`	example	example output

xxx	yyy	example	example output
cN	ISO CN	c17	ISO C17
c++N	ISO C++N	c++20	ISO C++20
gnuN	GNU dialect of CN	gnu11	GNU dialect of C11
gnu++N	GNU dialect of C++N	gnu++14	GNU dialect of C++14

`N` **consists of two digits.** For example, the 2003 ISO C++ standard is named `C++03`.

8. Other

Anything that does not match the patterns above is treated as an input file.

Description:

```
xxx: yyy as input file.
```

where `xxx` is replaced by the file name and `yyy` is replaced by the type of the file according to the following table.

xxx	yyy	example
*.c	C source code	hello.c
*.h	C/C++ header file	stdio.h
*.cpp, *.C, *.cc, *.cxx	C++ source code	checker.cc, game.cpp
*.hpp, *.hxx	C++ header file	utils.hpp

Notes

There is no input for this problem. The arguments to be explained are given as the **command line arguments** of your program.

It is guaranteed that the arguments are valid: Anything that does not match the patterns in 1 ~ 7 must be a valid C/C++ source or header file name as described in the table above. For the language standards (pattern 7), you don't need to check whether the standard that it refers to actually exists. Things like `-std=c++100000` won't appear.

The names of the files and directories involved in the arguments do not contain whitespaces or quotes.

The name following `-o` or `-I` is not a C/C++ source or header file.

Print the explanations in order of the arguments.

Please be extremely careful about the output contents, especially the spaces, newlines and punctuations.

Example

Suppose the name of your program (executable) is `my_program.exe` (without `.exe` on Mac OS X and Linux). If the following command (with `.\` replaced by `./` on Mac OS X and Linux) is executed,

```
.\my_program -std=c++20 -Wall -Wpedantic -Wextra a.c b.hxx c.cpp -Werror -o output_file -I
/usr/local/boost_1_80_0/
```

the output is

```
-std=c++20: Set the language standard to ISO C++20.
-Wall: Enable all the warnings about constructions that some users consider questionable, and
that are easy to avoid (or modify to prevent the warning).
-Wpedantic: Issue all the warnings demanded by strict ISO C and ISO C++ and reject all programs
that use forbidden extensions.
-Wextra: Enable some extra warning flags that are not enabled by -Wall.
a.c: C source code as input file.
b.hxx: C++ header file as input file.
c.cpp: C++ source code as input file.
-Werror: Make all warnings into errors.
-o output_file: Place the primary output in file output_file.
-I /usr/local/boost_1_80_0/: Add the directory /usr/local/boost_1_80_0/ to the list of
directories to be searched for header files during preprocessing.
```