1.   The details will be filled in due course. The interface of this module is included first. It is also used by
the main programs.

2.   Substitution

Ritchie writes in "The Development of the C Language", see `c_development.html`: "... but the most
important was the introduction of the preprocessor ... The preprocessor performs macro substitution, using
conventions distinct from the rest of the language. ..."

Here's what Wittgenstein says in the **TLP**, see `tlp.pdf`: "6.24 The method by which mathematics arrives
at its equations is the method of substitution. For equations express the substitutability of two expressions,
and we proceed from a number of equations to new equations, replacing expressions by others in accordance
with the equations."

Stallman defines *header file* in "The C Preprocessor", see `cpp.pdf`: A *header file* is a file containing C
declarations and macro definitions (see Chapter 3 [Macros], page 13) to be shared between several source
files. You request the use of a header file in your program by including it, with the C preprocessing directive
#include.

3.   System Header

The system header `<stdio.h>` declares three types, several macros, and many functions for performing input
and output, see **ISO/IEC 9899:TC3**.

To use the *printf()* function we must include `<stdio.h>`. *printf* is the name of one of the main C output
functions, and stands for "print formatted". The *printf* format string is a control parameter used by a
class of functions in the input/output libraries of C and many other programming languages. The string
is written in a simple template language: characters are usually copied literally into the function's output,
but format specifiers, which start with a translate a piece of data (such as a number) to characters, see
`https://en.wikipedia.org/wiki/Printf_format_string`.

The output function *printf* translates internal values to characters:

```
int printf(char *format, arg1, arg2, ...);
```

printf converts, formats, and prints its arguments on the format. It returns the number of characters printed,
see **KR**.

#include <stdio.h>

4.   External variables

Because external variables are globally accessible, they can be used instead of argument lists to communicate
data between functions. Furthermore, because external variables remain in existence permanently, rather
than appearing and disappearing as functions are called and exited, they retain their values even ater the
functions that set them ve returned, see **KR**.

   Declaration of the global variables or function simply declares that the variable or function exists, but the
memory is not allocated for them.

   argc: copy of *ac* parameter to *main*.
   argv: copy of *av* parameter to *main*

   extern int *argc*;
   extern char $**argv$;

5.   Coming to the definition of the global variables, when we define a variable or function, in addition to
everything that a declaration does, it also allocates memory for that variable or function.

   int *argc*;
   char $**argv$;

6.   Index.

LEXTERN