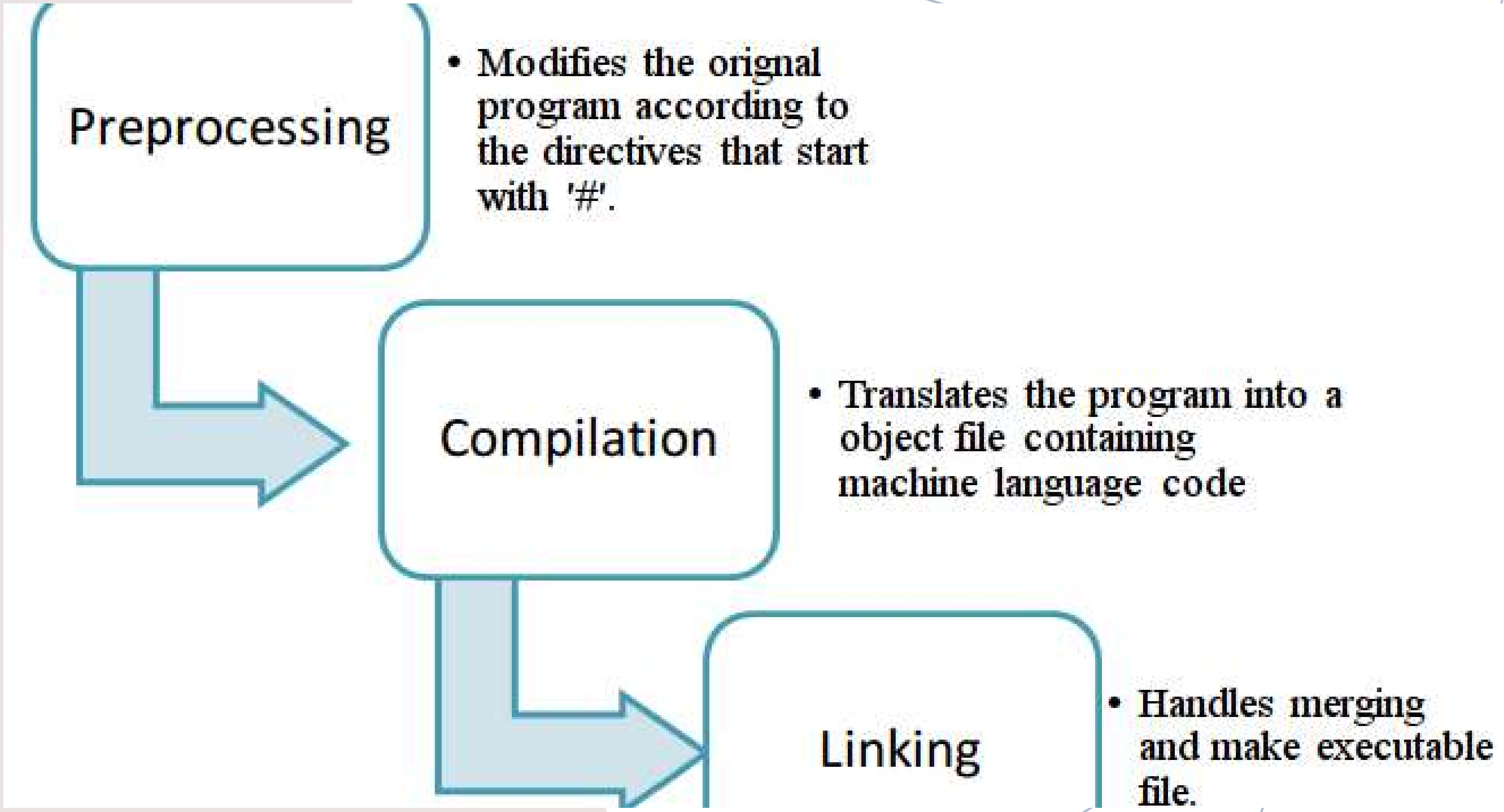

- Modifies the orignal program according to the directives that start with '#'.
- Translates the program into a object file containing machine language code
- Handles merging and make executable file.

PREPROCESSOR

- Preprocessors are programs that process our source code before compilation
- This file is then processed by preprocessors and an expanded source code file is generated named program
- This expanded file is compiled by the compiler and an object code file is generated named program .obj.
- Finally, the linker links this object code file to the object code of the library functions to generate the executable file program.exe.
- Preprocessor programs provide preprocessors directives which tell the compiler to preprocess the source code before compiling. All of these preprocessor directives begin with a '#' (hash) symbol. The '#' symbol indicates that, whatever statement starts with #, is going to the preprocessor program, and preprocessor program will execute this statement.

TYPES OF PREPROCESSOR DIRECTIVES

- Macros
- File Inclusion
- Conditional Compilation
- Other directives

MACROS

- Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The '#define' directive is used to define a macro.

```
// macro definition

#define LIMIT 5
```

```
#include <iostream>

// macro definition
#define LIMIT 5
int main()
{
        for (int i = 0; i < LIMIT; i++) {
                std::cout << i << "\n";
        }

        return 0;
}
```

FILE INCLUSION

- This type of preprocessor directive tells the compiler to include a file in the source code program. There are two types of files which can be included by the user in the program:
- Header File or Standard files: These files contains definition of pre-defined functions like printf(), scanf() etc. These files must be included for working with these functions. Different function are declared in different header files. For example standard I/O functions are in ‘iostream’ file whereas functions which perform string operations are in ‘string’ file
- #include< file_name >

- user defined files: When a program becomes very large, it is good practice to divide it into smaller files and include whenever needed. These types of files are user defined files. These files can be included as:
- #include"filename"

CONDITIONAL COMPILATION

- Conditional Compilation directives are type of directives which helps to compile a specific portion of the program or to skip compilation of some specific part of the program based on some conditions. This can be done with the help of two preprocessing commands ‘ifdef‘ and ‘endif‘.

```
#ifdef macro_name
    statement1;
    statement2;
    statement3;
    .
    .
    .
    statementN;
#endif
```

#UNDEF DIRECTIVE

- The #undef directive is used to undefine an existing macro. This directive works as:
- #undef LIMIT
- Using this statement will undefine the existing macro LIMIT. After this statement every “#ifdef LIMIT” statement will evaluate to false.

#PRAGMA DIRECTIVE

- This directive is a special purpose directive and is used to turn on or off some features. This type of directives are compiler-specific, i.e., they vary from compiler to compiler.

- **#pragma startup** and **#pragma exit**
- These directives helps us to specify the functions that are needed to run before program startup(before the control passes to main()) and just before program exit (just before the control returns from main()).

#PRAGMA WARN DIRECTIVE

- This directive is used to hide the warning message which are displayed during compilation.
- We can hide the warnings as shown below:
- #pragma warn -rvl: This directive hides those warning which are raised when a function which is supposed to return a value does not returns a value.
- #pragma warn -par: This directive hides those warning which are raised when a function does not uses the parameters passed to it.
- #pragma warn -rch: This directive hides those warning which are raised when a code is unreachable. For example: any code written after the return statement in a function is unreachable.