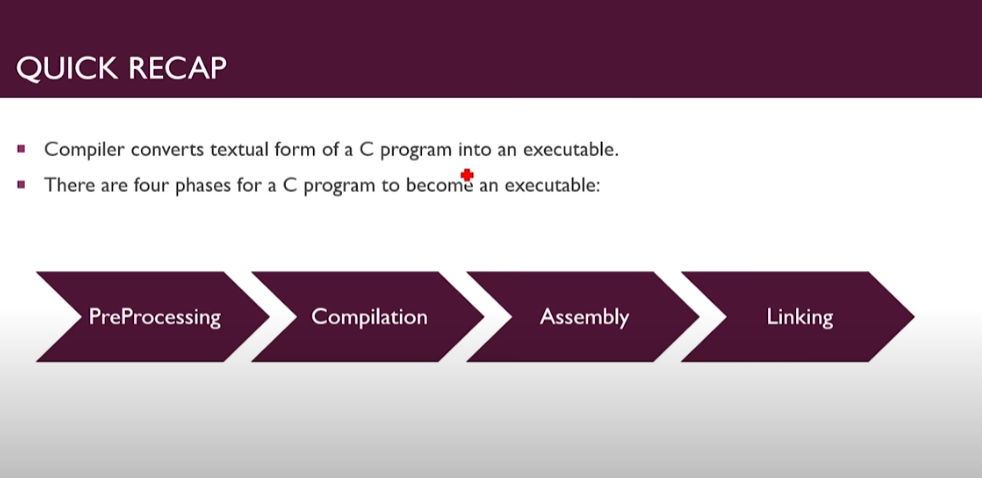
Content 42

C Pre-processor Introduction & Working

As we all know, what happens behind the scene in C programming. But before learning about pre-processors in C, it is quite import to revise the concept. Let us have a quick recap of what’s happening behind C program. The compiler converts the textual form of the C program into an executable file. There are four phases for the C program to become executable. These are **Pre-processing, compilation, assembly, and linking**. In pre-processing, it includes removing comments, expansion of macro, and expansion of include files. In the compilation phase, assembly level instructions are generated. Whereas in the assembly phase, the assembly level instructions are converted into machine code, and in linking, it resolves the function calls. In short, it links the function implementation. Now let us move to the main topic of this Content , **"Pre-processors in C."**



What Happens in **PREPROCESSING**

1. Removal of Comments.
2. Expansion of Macros. **preprocessing**
3. Expansion include files.

What Happens in **Compilation**

Only Machine Level Instructions

are generated.

What Happens in **Assembly**

1. Creates .o and .exe files but

function Calls are not resolved yet.

1. Assembly Level Language get converted into Machine Code.

What Happens In **Linking**

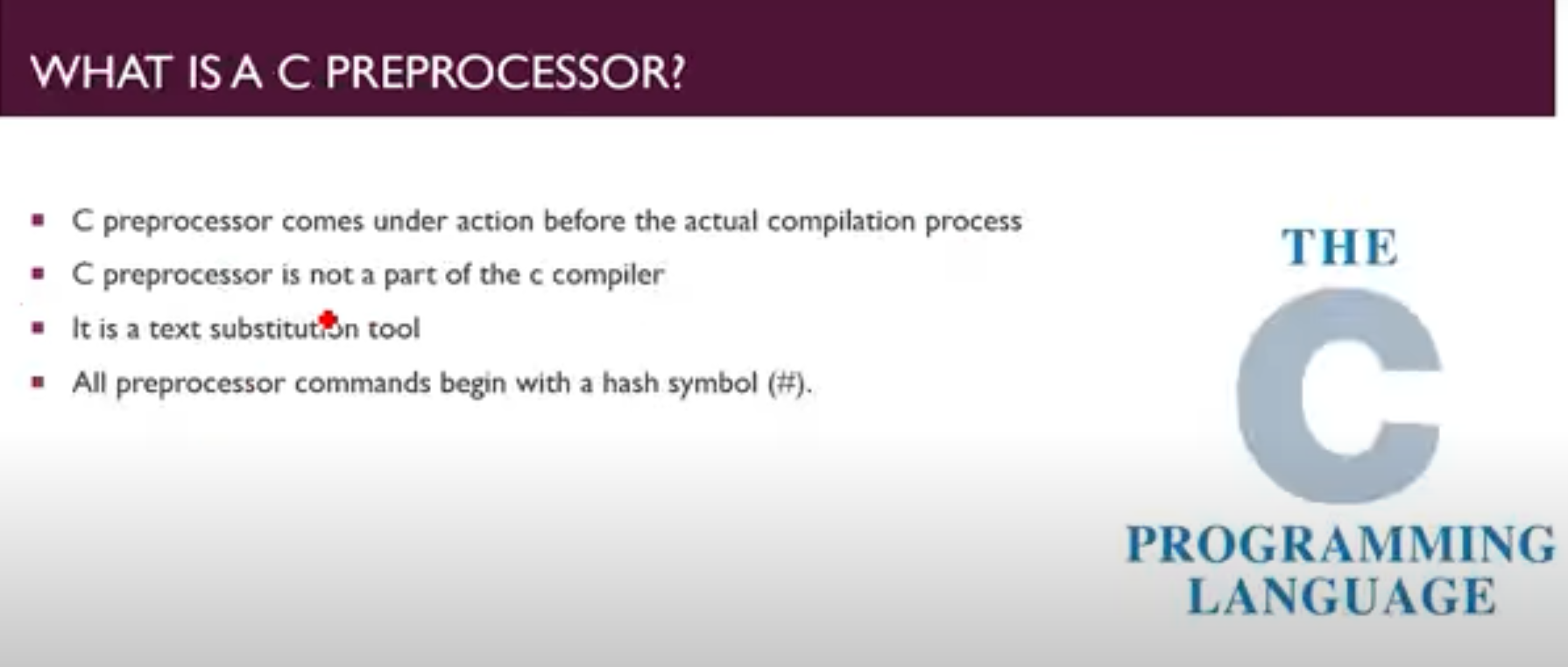
Links the Function

implementation.

#### What is a C Pre-processor?

The **C Pre-processor** is not a part of the compiler. The C Pre-processor is just a text substitution tool, and it instructs the compiler to do required pre-processing before the actual compilation process. We refer to the C Pre-processor as CPP.

Pre-processor directives like #include come into effect as soon as they are seen and keep working until the end of the file that contains them; the program's block structure is irrelevant. In C, Pre-processing directives are lines in the program that start with the hash symbol`#.' The hash symbol `#' is followed by an identifier that is the directive name. For example, `#include.' Whitespace is also allowed before and after the hash symbol`#.' The pre-processor in C does not know about the scope rules of C language.



**The following table has all the important pre-processor directives**:

|  |  |
| --- | --- |
| **Directive** | **Description** |
| #include | This directive will insert a particular header from another file. |
| #define | This will substitute a pre-processor macro. |
| #undef | Using #undef, we can undefine a pre-processor macro. |
| #ifdef | It will return true if this macro is defined. |
| #if | It will test if a compile time condition is true. |
| #elif | Using #elif, we can define #else and #if in one statement |
| #endif | It will end pre-processor conditional |
| #error | Using #error, we can print the error message on stderr. |
| #pragma | It will issue the special commands to the compiler, using a standardized method. |

#### Pre-processors Examples:-

The following are examples of pre-processors in C.

#define ARRAY\_LENGTH 46

This directive tells the CPP to set the ARRAY\_LENGTH with 46.

#include <stdio.h>

These directives will get stdio.h from **system libraries**.

#undef FSIZE

#define FSIZE 30

It will undefine existing FSIZE and define it as 30.

#### Summary:-

The first step in compiling the C program is the **pre-processor,** a sort of automated editor that modifies our source code before passing it on to the compiler to translate into machine language code. One of the tasks of the pre-processor is that it removes all comments, which the compiler ignores. The pre-processor also responds to **directives** in the code like #define, #include or #elif, which give the pre-processor the instructions on how to edit the source code before passing it on to the compiler.

