

C - HEADER FILES

A header file is a file with extension **.h** which contains C function declarations and macro definitions and to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that come with your compiler.

You request the use of a header file in your program by including it, with the C preprocessing directive **#include** like you have seen inclusion of **stdio.h** header file, which comes along with your compiler.

Including a header file is equal to copying the content of the header file but we do not do it because it will be very much error-prone and it is not a good idea to copy the content of header file in the source files, specially if we have multiple source file comprising our program.

A simple practice in C or C++ programs is that we keep all the constants, macros, system wide global variables, and function prototypes in header files and include that header file wherever it is required.

Include Syntax

Both user and system header files are included using the preprocessing directive **#include**. It has following two forms:

```
#include <file>
```

This form is used for system header files. It searches for a file named file in a standard list of system directories. You can prepend directories to this list with the -I option while compiling your source code.

```
#include "file"
```

This form is used for header files of your own program. It searches for a file named file in the directory containing the current file. You can prepend directories to this list with the -I option while compiling your source code.

Include Operation

The **#include** directive works by directing the C preprocessor to scan the specified file as input before continuing with the rest of the current source file. The output from the preprocessor contains the output already generated, followed by the output resulting from the included file, followed by the output that comes from the text after the **#include** directive. For example, if you have a header file header.h as follows:

```
char *test (void);
```

and a main program called *program.c* that uses the header file, like this:

```
int x;
#include "header.h"

int main (void)
{
    puts (test ());
}
```

the compiler will see the same token stream as it would if program.c read

```
int x;
char *test (void);

int main (void)
{
    puts (test ());
}
```

Once-Only Headers

If a header file happens to be included twice, the compiler will process its contents twice and will result an error. The standard way to prevent this is to enclose the entire real contents of the file in a conditional, like this:

```
#ifndef HEADER_FILE
#define HEADER_FILE

the entire header file file

#endif
```

This construct is commonly known as a wrapper **#ifndef**. When the header is included again, the conditional will be false, because `HEADER_FILE` is defined. The preprocessor will skip over the entire contents of the file, and the compiler will not see it twice.

Computed Includes

Sometimes it is necessary to select one of several different header files to be included into your program. They might specify configuration parameters to be used on different sorts of operating systems, for instance. You could do this with a series of conditionals as follows:

```
#if SYSTEM_1
    # include "system_1.h"
#elif SYSTEM_2
    # include "system_2.h"
#elif SYSTEM_3
    ...
#endif
```

But as it grows, it becomes tedious, instead the preprocessor offers the ability to use a macro for the header name. This is called a **computed include**. Instead of writing a header name as the direct argument of **#include**, you simply put a macro name there instead:

```
#define SYSTEM_H "system_1.h"
...
#include SYSTEM_H
```

`SYSTEM_H` will be expanded, and the preprocessor will look for `system_1.h` as if the **#include** had been written that way originally. `SYSTEM_H` could be defined by your Make file with a `-D` option.