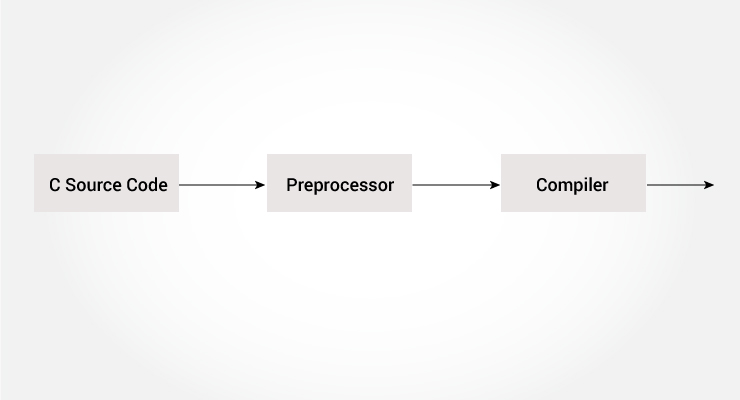
**C Preprocessor and Macros**



The C preprocessor is a macro preprocessor (allows you to define macros) that transforms your program before it is compiled. These transformations can be the inclusion of header file, macro expansions etc.

All preprocessing directives begin with a # symbol. For example,

#define PI 3.14

Some of the common uses of C preprocessor are:

**Including Header Files: #include**

The #include preprocessor is used to include header files to C programs. For example,

#include <stdio.h>

Here, stdio.h is a header file. The #include preprocessor directive replaces the above line with the contents of stdio.h header file.

That's the reason why you need to use #include <stdio.h> before you can use functions like scanf() and printf().

You can also create your own header file containing function declaration and include it in your program using this preprocessor directive.

#include "my\_header.h"

# How to write your own header file in C?

As we all know that files with .h extension are called **header files**in C.

These header files generally contain function declarations which we can be used in our main C program, like for e.g. there is need to include stdio.h in our C program to use function printf() in the program.

Header files are simply files in which you can declare your own functions that you can use in your main program or these can be used while writing large C programs.

**NOTE:** Header files generally contain definitions of data types, function prototypes and C preprocessor commands.

**Below is the short example of creating your own header file and using it accordingly.**

1. **Creating myhead.h :**Write the below code and then save the file as **myhead.h** or you can give any name but the extension should be .h indicating its a header file.

|  |
| --- |
| void add(int a, int b)  {      printf("Added value=%d\n", a + b);  }  void multiply(int a, int b)  {      printf("Multiplied value=%d\n", a \* b);  } |

1. **Including the .h file in other program :**

Now as we need to include stdio.h as #include in order to use printf() function.

We will also need to include the above header file myhead.h as **#include ”myhead.h”**.

The ” ” here are used to instructs the [preprocessor](https://www.geeksforgeeks.org/cc-preprocessors/)to look into the present folder and into the standard folder of all header files if not found in present folder. So, if you wish to use angular brackets instead of ” ” to include your header file you can save it in the standard folder of header files otherwise. If you are using ” ” you need to ensure that the header file you created is saved in the same folder in which you will save the C file using this header file.

1. **Using the created header file :**

|  |
| --- |
| // C program to use the above created header file  #include <stdio.h>  #include "myhead.h"  int main()  {      add(4, 6);        /\*This calls add function written in myhead.h        and therefore no compilation error.\*/      multiply(5, 5);        // Same for the multiply function in myhead.h      printf("BYE!See you Soon");      return 0;  } |

Output:

Added value:10

Multiplied value:25

BYE!See you Soon

**NOTE :**The above code compiles successfully and prints the above output only if you have created the header file and saved it in the same folder the above c file is saved.

**Important Points:**

The creation of header files are needed generally while writing large C programs so that the modules can share the function definitions, prototypes etc.

* Function and type declarations, global variables, structure declarations and in some cases, inline functions; definitions which need to be centralized in one file.
* In a header file, do not use redundant or other header files; only minimal set of statements.
* Don’t put function definitions in a header. Put these things in a separate .c file.
* Include Declarations for functions and variables whose definitions will be visible to the linker. Also, definitions of data structures and enumerations that are shared among multiple source files.
* In short, Put only what is necessary and keep the header file concised.

**Macros using #define**

A macro is a fragment of code that is given a name. You can define a macro in C using the #define preprocessor directive.

Here's an example.

#define c 299792458 // speed of light

Here, when we use c in our program, it is replaced with 299792458.

**Example 1: #define preprocessor**

#include <stdio.h>

#define PI 3.1415

int main()

{

float radius, area;

printf("Enter the radius: ");

scanf("%f", &radius);

// Notice, the use of PI

area = PI\*radius\*radius;

printf("Area=%.2f",area);

return 0;

}

**Function like Macros**

You can also define macros that work in a similar way like a function call. This is known as function-like macros. For example,

#define circleArea(r) (3.1415\*(r)\*(r))

Every time the program encounters circleArea(argument), it is replaced by (3.1415\*(argument)\*(argument)).

Suppose, we passed 5 as an argument then, it expands as below:

circleArea(5) expands to (3.1415\*5\*5)

**Example 2: Using #define preprocessor**

#include <stdio.h>

#define PI 3.1415

#define circleArea(r) (PI\*r\*r)

int main() {

float radius, area;

printf("Enter the radius: ");

scanf("%f", &radius);

area = circleArea(radius);

printf("Area = %.2f", area);

return 0;

}

**Uses of Conditional**

* use different code depending on the machine, operating system
* compile same source file in two different programs
* to exclude certain code from the program but to keep it as reference for future purpose

### 2. C Conditional Macros

Conditional macros are very useful to apply conditions. Code snippets are guarded with a condition checking if a certain macro is defined or not. They are very helpful in large project having code segregated as per releases of project. If some part of code needs to be executed for release 1 of project and some other part of code needs to be executed for release 2, then it can be easily achieved through conditional macros.

**Uses of Conditional**

* use different code depending on the machine, operating system
* compile same source file in two different programs
* to exclude certain code from the program but to keep it as reference for future purpose

**How to use conditional?**

To use conditional, #ifdef, #if, #defined, #else and #elseif directives are used.

**#ifdef Directive**

#ifdef MACRO

// conditional codes

#endif

Here, the conditional codes are included in the program only if MACRO is defined.

**#if, #elif and #else Directive**

#if expression

// conditional codes

#endif

Here, expression is an expression of integer type (can be integers, characters, arithmetic expression, macros and so on).

The conditional codes are included in the program only if the expression is evaluated to a non-zero value.

The optional #else directive can be used with #if directive.

#if expression

conditional codes if expression is non-zero

#else

conditional if expression is 0

#endif

You can also add nested conditional to your #if...#else using #elif

#if expression

// conditional codes if expression is non-zero

#elif expression1

// conditional codes if expression is non-zero

#elif expression2

// conditional codes if expression is non-zero

#else

// conditional if all expressions are 0

#endif

Here is the syntax :

#ifdef PRJ\_REL\_01

..

.. code of REL 01 ..

..

#else

..

.. code of REL 02 ..

..

#endif

To comment multiples lines of code, macro is used commonly in way given below :

#if 0

..

.. code to be commented ..

..

#endif

Here, we will understand above features of macro through working program that is given below.

#include <stdio.h>

int main() {

#if 0

printf("commented code 1");

printf("commented code 2");

#endif

#define TEST1 1

#ifdef TEST1

printf("MACRO TEST1 is defined\n");

#endif

#ifdef TEST3

printf("MACRO TEST3 is defined\n");

#else

printf("MACRO TEST3 is NOT defined\n");

#endif

return 0;

}

Output:

$ ./macro

MACRO TEST1 is defined

MACRO TEST3 is NOT defined

Here, we can see that “commented code 1”, “commented code 2” are not printed because these lines of code are commented under #if 0 macro. And, TEST1 macro is defined so, string “MACRO TEST1 is defined” is printed and since macro TEST3 is not defined, so “MACRO TEST3 is defined” is not printed.