**Preprocessors**

Table of Contents

[Macros 3](#_Toc65506286)

[Function-Like-Macros 5](#_Toc65506287)

[Predefined and Local Header Files 6](#_Toc65506288)

[Conditioned Compilation 7](#_Toc65506289)

[C Built-in Macros 10](#_Toc65506290)

A preprocessor is a language that takes as input a text file written using some programming language syntax and output another text file following the syntax of another programming language. The text file does not necessarily have to be a .txt file. It can be any file that contains text like a .c file or a .java file.

Purpose – Extend the syntax of the language by adding new instructions

The Preprocessor Language – not a part of the C language

Examples:

#define  
#elif  
#include  
#endif  
#if  
#ifdef

C

## Macros

* An instruction that can be expanded into another set of instructions.
* A name that can be substituted by another set of names.
* Can be considered as an abbreviation in the programming world.

The #define preprocessor is used to declare a macro.

#define **DEBUG** printf("I AM HERE\n");

C

Whenever **DEBUG** is used, it will mean printf("I AM HERE\n");

int main()  
{  
 **DEBUG**}

C

OUTPUT: I AM HERE

(A BUG is a mistake or an error in code. DEBUG means to remove the error or mistake.)

#include <stdio.h>  
#define **DEBUG** printf("I AM HERE\n ");  
int main()  
{  
 arr[100];  
 arr[-1] – 5;  
 **DEBUG**}

C

OUTPUT:

The phrase I AM HERE was not printed since an error exists and the program stopped before reaching the third line. This can be used to find errors in the code.

#include <stdio.h>  
#define **MAX\_SIZE** 1000000  
int main()  
{  
 arr[**MAX\_SIZE**];  
}

C

This code makes it easier to change the maximum size at once. A variable could have been used but a variable can be changed later in the code. In this way, the value cannot be changed.

Function-Like-Macros

#include <stdio.h>  
#define **SUM**(i, j) i+j  
int main()  
{  
 int s = **SUM**(10, 20);  
 printf("%d", s);  
}

C

OUTPUT: 30

#include <stdio.h>  
#define **RANGE**(i, min, max) (i < min) || (i > max) ? 1 : 0  
int main()  
{  
 **RANGE**(5, 2, 10)  
 *//if 5 < 2 or 5 > 10, return 1; else return 0*}

C

In function-like-macros,

* Instead of function calls, and inline code is generated, saving time for function calls and returns.
* As code is duplicated, might take longer than functions.

## Predefined and Local Header Files

#include <stdio.h> - finds file in predefined location

#include "stdio.h" - finds file in local folder first, then pre-defined location

## Conditioned Compilation

#if  
#else  
#elif  
#endif  
#ifdef  
#ifndef

C

In case of macros, if a statement is not true, after compilation the final code will not contain that line at all. This can be used to save memory, and make it easier to change the code inside.

Conditional macros compile conditionally, while conditional statements execute conditionally.

#include<stdio.h>  
#define **CHAR\_SET** 256 *//changing this line can change a lot of things inside*int main()  
{  
 int i;  
 #if **CHAR\_SET** == 256 *//if this is true the rest disappears* printf("Displaying 256");  
 #else printf("Displaying 127");

*//if this is true the rest disappears* #endif  
 for (i=0; i<**CHAR\_SET**; i++)  
 printf("%c", i);  
}

C

#ifdef checks if a macro is defined. #ifndef does the opposite.

#include <stdio.h>  
  
#define **DEBUG** printf("Error");  
  
int main()  
{  
 int a;  
 scanf("%d", &a);  
 #ifdef **DEBUG** printf("%d", a);  
 #endif  
 #ifndef **DEBUG** printf("FINAL VERSION");  
 #endif  
}

C

This code will print the integer after scanning it. It can be used to check if there was any error while scanning. Similarly, the #ifdef macro can be used in different parts of the code to check different things. After debugging is done, the **DEBUG** definition can simply be removed from the code and all the **DEBUG** lines inside the code will be removed from the final application.

#error gives an error message in the log and stops the compilation.

int main()  
{  
 *// code that needs to be checked* #error Error  
 *// code that is known has problems and will not work*}

C

#undefine removes a macro definition

#undefine macro\_name

C

#line changes the line number and file name

#line 34 "prog.c"

C

#pragma gives instructions to the compiler

#pragma instructions

C

## C Built-in Macros

\_\_LINE\_\_ int returns the line being compiled

\_\_FILE\_\_ string returns the file name

\_\_DATE\_\_ string returns the compilation date (MMM/DD/YYYY)

\_\_TIME\_\_ string returns the compilation time (HH:MM:SS)

\_\_STDC\_\_ value 1 (ANSI C standard is being followed)

#include<stdio.h>  
#include<stdio.h>  
int main()  
{  
 printf("Time: %s, Date: %s", \_\_TIME\_\_, \_\_DATE\_\_);  
}

C

OUTPUT: Time: 09:00:00, Date: April 23, 2019