55. Pre-Processing In C Programming:

The C preprocessor is a microprocessor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros. It is a program that processes our source program before it is passed to the compiler. Preprocessor commands (often known as directives) form what can almost be considered a language within C language.

The c preprocessor offers following operators to help in creating macros, macros continuation(\), stringize(#), token pasting(##). one of the powerful functions of the C preprocessor is the ability to simulate functions using parameterized macros.

The preprocessor defined operator is used in constant expression to determine if an identifier is defined using #define if the specified identifier is defined, the value is true (non-zero). if the symbol is not defined the value is false(zero).

Feature Of C Preprocessor:

before a C program is compiled it is passed through another program called 'Preprocessor'. The C program is often known as 'Source Code'. The Preprocessor works on the source code and creates 'Expanded Source Code'. If the source code is stored in a file PR1.C, then the expanded source code gets stored in a file PR1.I. It is this expanded source code that is sent to the compiler for compilation.

Different Properties Of Preprocessor:

The directives can be placed anywhere in a program but are most often placed at the beginning of a program, before the first function definition. We would learn the following preprocessor directives here:

1. Macro Expansion	3. Conditional Compilation
2. File Inclusion	4. Miscellaneous Directives

1. Macro Expansion:

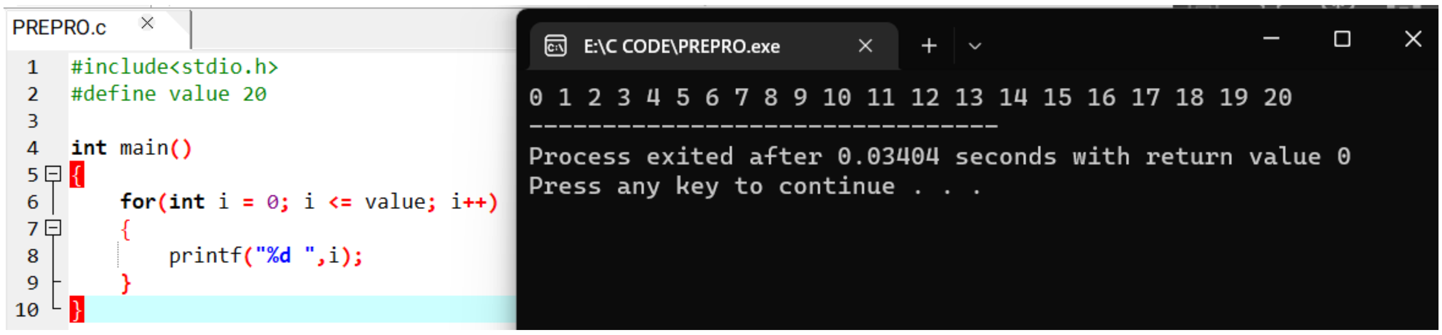
A macro is a segment of code which is replaced by the value of macro. Macro is defined by "#define" directive. There are two types of macros. a macro is a text substitution definition which mean whenever a macros is called the respective text for which macro is defined gets expanded at the line of call.

1.1 Object-Like Macros:

The Object-Like Macros is an identifier that is replaced by value. It is widely used to represent numeric constants.

1.2 Function-Like Macros:

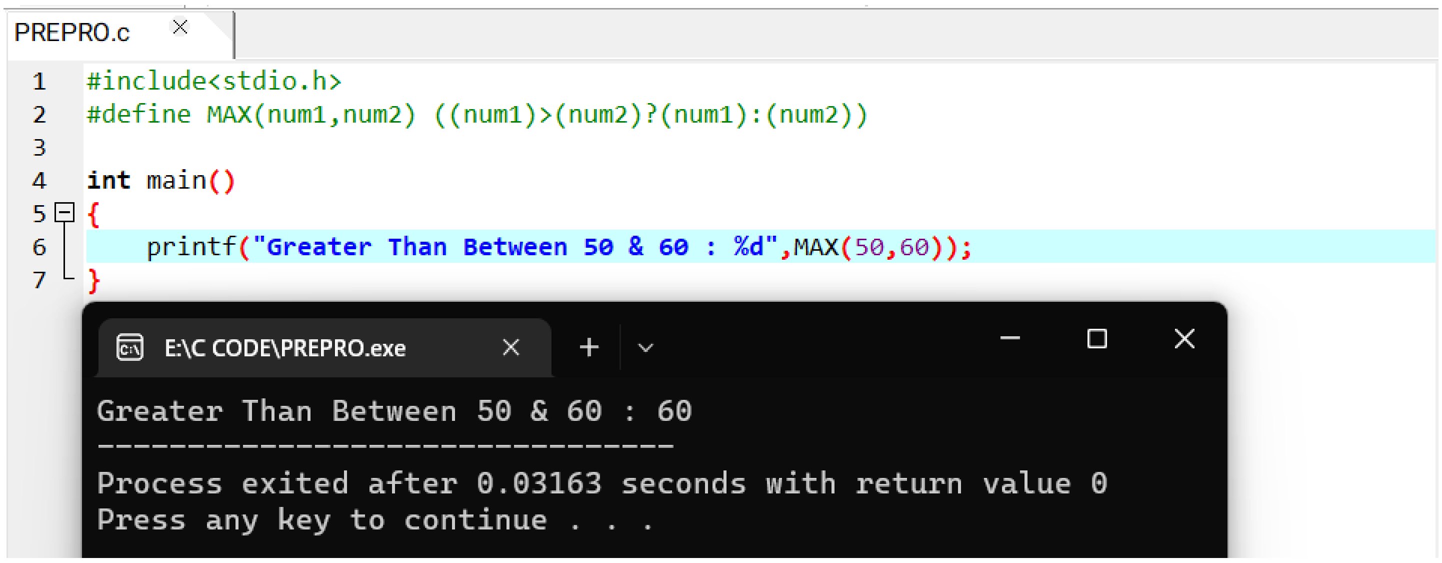
The function-like macro looks like function call.


```
#include<stdio.h>
#define value 20

int main()
{
    for(int i = 0; i <= value; i++)
    {
        printf("%d ",i);
    }
}
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
--------------------------------
Process exited after 0.03404 seconds with return value 0
Press any key to continue . . .
```

"#define value 20" is called macro definition. During preprocessing the preprocessor replaces every occurrence of "value" in the program with 20. "#define value 20" the "value" often called macro template. The "20" called its corresponding macro expansion.

why use #define in the above programs? What have we gained by substituting "value" for 20 in our program? Probably, we have made the program easier to read. There is perhaps a more important reason for using macro definition than mere readability.


```
#include<stdio.h>
#define MAX(num1,num2) ((num1)>(num2)?(num1):(num2))

int main()
{
    printf("Greater Than Between 50 & 60 : %d",MAX(50,60));
}
```

```
Greater Than Between 50 & 60 : 60
--------------------------------
Process exited after 0.03163 seconds with return value 0
Press any key to continue . . .
```

Predefines Macros:

DATE = it represent current date in "MMM DD YYYY" format.

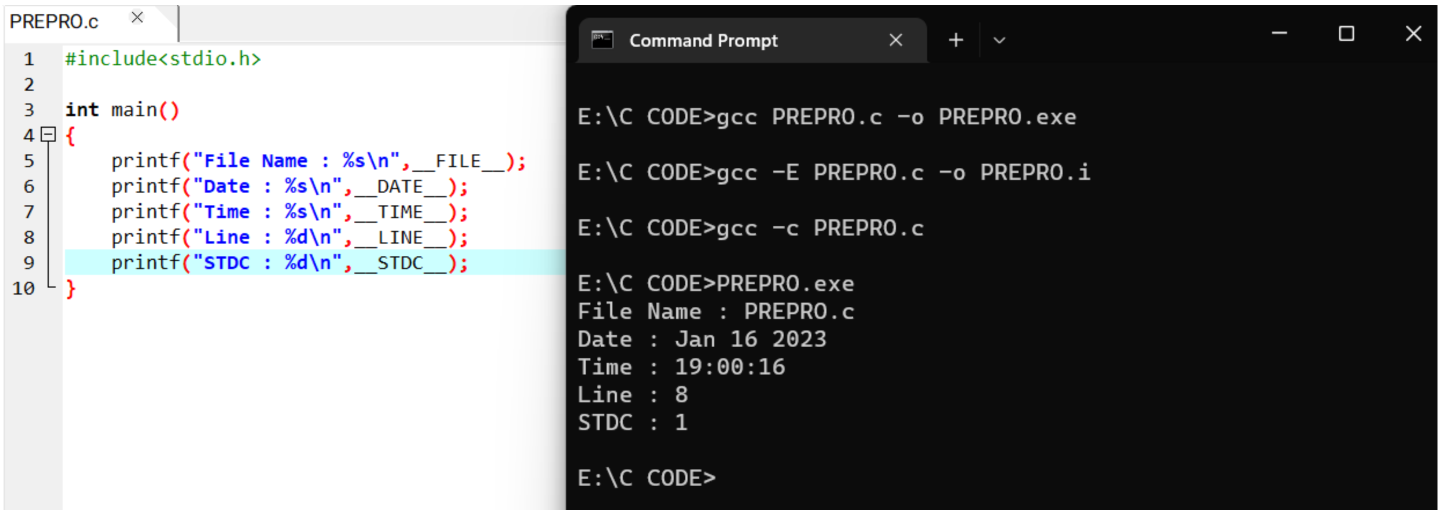
TIME = it represent current time in "HH:MM:SS" format.

FILE = it represent the current file name.

LINE = it represents current line number

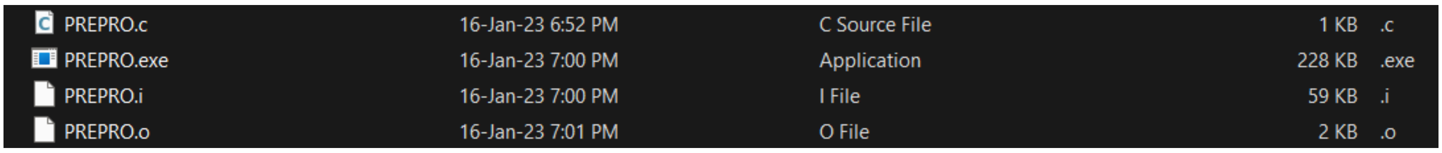
STDC = it defined as 1 when compiler compilers with the ANSI standard.

To create preprocessor file(.i) file for a C programming file then use this code in command prompt.   "gcc -E filename.c -o filename.o" and press enter the command will create the (.i) preprocessor file.



```c
#include<stdio.h>

int main()
{
    printf("File Name : %s\n",__FILE__);
    printf("Date : %s\n",__DATE__);
    printf("Time : %s\n",__TIME__);
    printf("Line : %d\n",__LINE__);
    printf("STDC : %d\n",__STDC__);
}
```

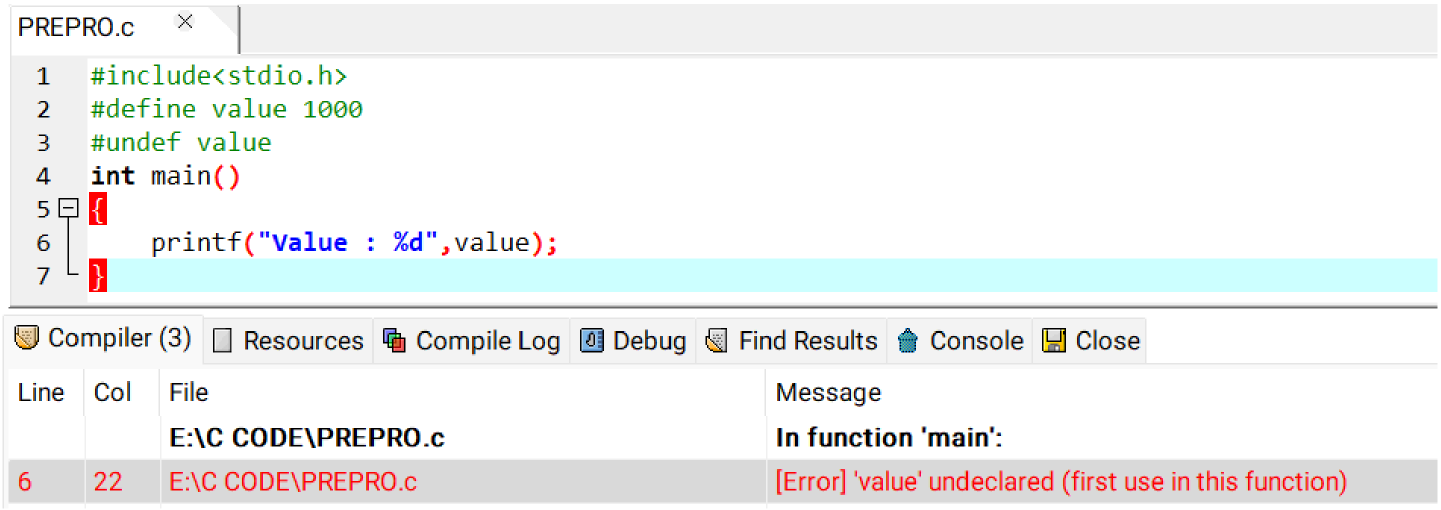
```
E:\C CODE>gcc PREPRO.c -o PREPRO.exe

E:\C CODE>gcc -E PREPRO.c -o PREPRO.i

E:\C CODE>gcc -c PREPRO.c

E:\C CODE>PREPRO.exe
File Name : PREPRO.c
Date : Jan 16 2023
Time : 19:00:16
Line : 8
STDC : 1

E:\C CODE>
```



PREPRO.c	16-Jan-23 6:52 PM	C Source File	1 KB	.c
PREPRO.exe	16-Jan-23 7:00 PM	Application	228 KB	.exe
PREPRO.i	16-Jan-23 7:00 PM	I File	59 KB	.i
PREPRO.o	16-Jan-23 7:01 PM	O File	2 KB	.o

#undef In C Preprocessor:

The #undef preprocessor directive is used to undefine the constant or macro defined by #define.



```c
#include<stdio.h>
#define value 1000
#undef value
int main()
{
    printf("Value : %d",value);
}
```

Line	Col	File	Message
		E:\C CODE\PREPRO.c	In function 'main':
6	22	E:\C CODE\PREPRO.c	[Error] 'value' undeclared (first use in this function)

2. File Inclusion:

The #include preprocessor directive is used to paste code of given file into current file. It is used include system-defined and user-defined header files. If included file is not found, compiler renders error. By the use of #include directive, we provide information to the preprocessor where to look for the header files. There are two variants to use #include directive.

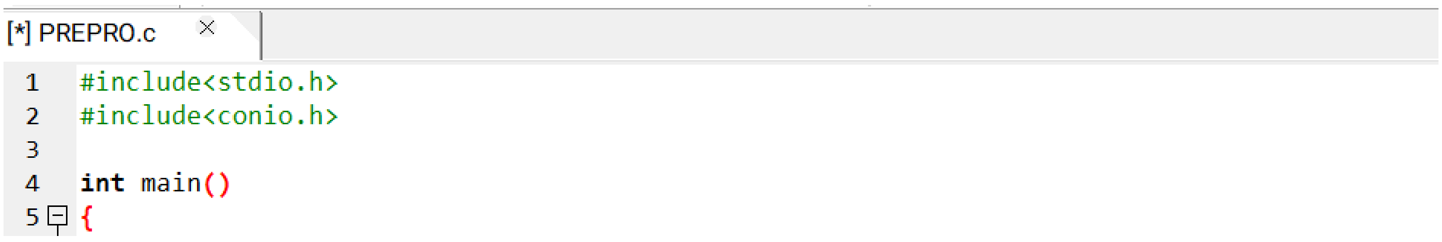
#include<filename.h> **#include "filename.extension"**

The #include <filename> tells the compiler to look for the directory where system header files are held.

The #include "filename" tells the compiler to look in the current directory from where program is running.

In #include directive, comments are not recognized. So in case of #include <a//b>, a//b is treated as filename.

In #include directive, backslash is considered as normal text not escape sequence. So in case of #include <a\nb>, a\nb is treated as filename. You can use only comment after filename otherwise it will give error.



[*] PREPRO.c

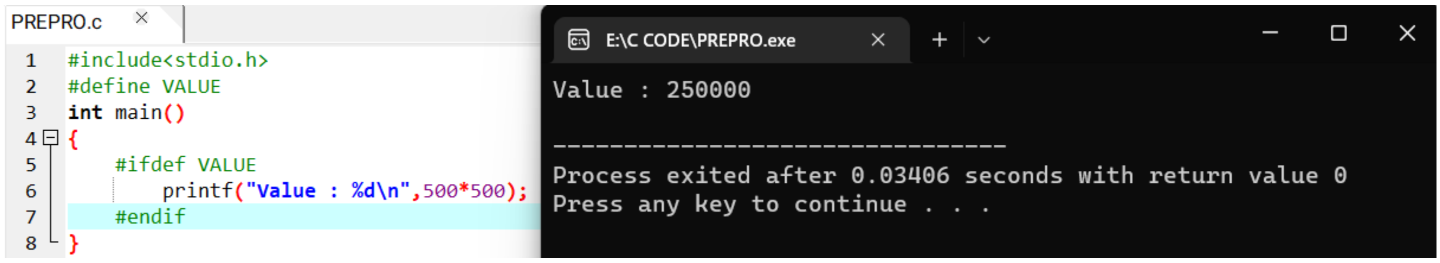
```c
#include<stdio.h>
#include<conio.h>

int main()
{
```

#include<stdio.h> & #include<conio.h> both are the #include file inclusion.

#include “mylib.h” this command would look for the file mylib.h in the current directory as well as the specified list of directories as mentioned in the search path. #include<mylib.h> would look for the file mylib.h in the specified list of directories only.

3. Conditional Compilation:

We can, if we want, have the compiler skip over part of a source code by inserting the preprocessing commands #ifdef and #endif, If macroname has been #defined, the block of code will be processed as usual; otherwise not. The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

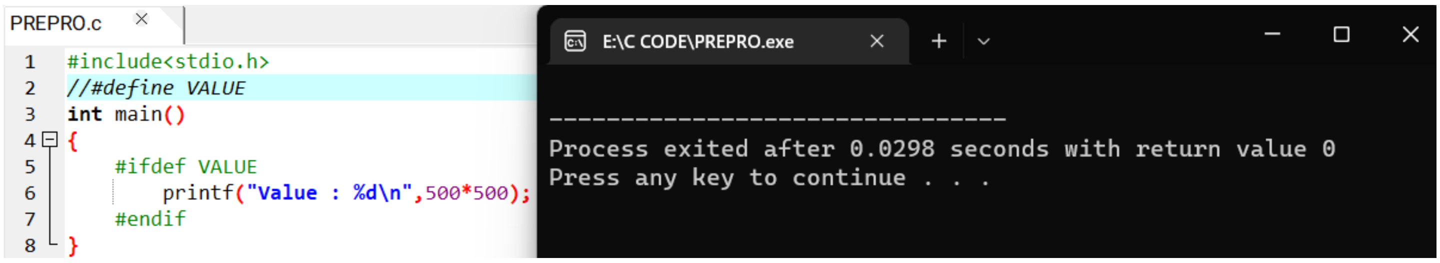


PREPRO.c

```c
#include<stdio.h>
#define VALUE
int main()
{
    #ifdef VALUE
        printf("Value : %d\n",500*500);
    #endif
}
```

E:\C CODE\PREPRO.exe

```
Value : 250000

--------------------------------
Process exited after 0.03406 seconds with return value 0
Press any key to continue . . .
```

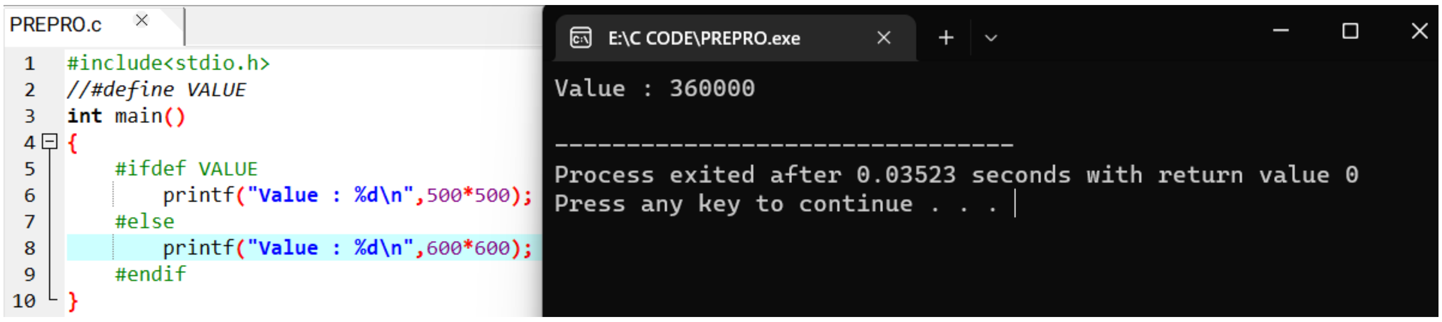


PREPRO.c

```c
#include<stdio.h>
//#define VALUE
int main()
{
    #ifdef VALUE
        printf("Value : %d\n",500*500);
    #endif
}
```

E:\C CODE\PREPRO.exe

```

--------------------------------
Process exited after 0.0298 seconds with return value 0
Press any key to continue . . .
```


PREPRO.c

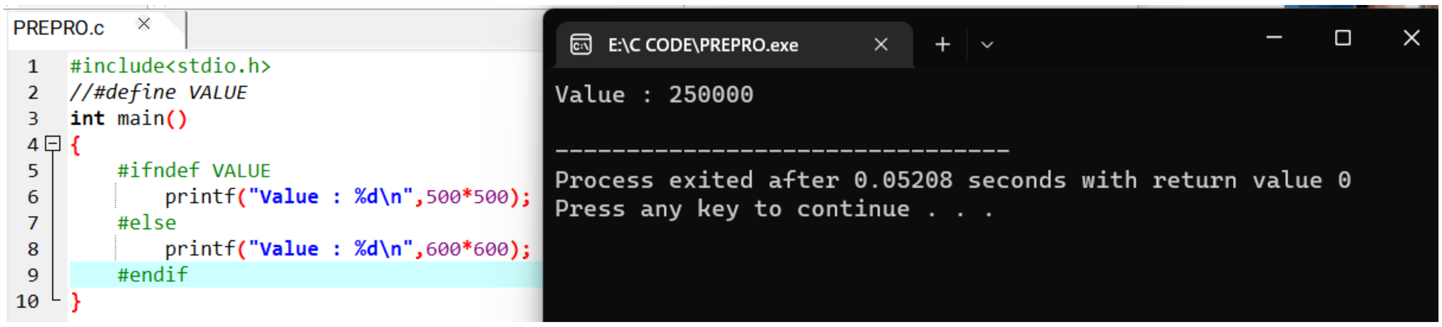
```c
#include<stdio.h>
//#define VALUE
int main()
{
    #ifdef VALUE
        printf("Value : %d\n",500*500);
    #else
        printf("Value : %d\n",600*600);
    #endif
}
```

E:\C CODE\PREPRO.exe

```
Value : 360000

--------------------------------
Process exited after 0.03523 seconds with return value 0
Press any key to continue . . .
```

#ifndef In C Preprocessor:

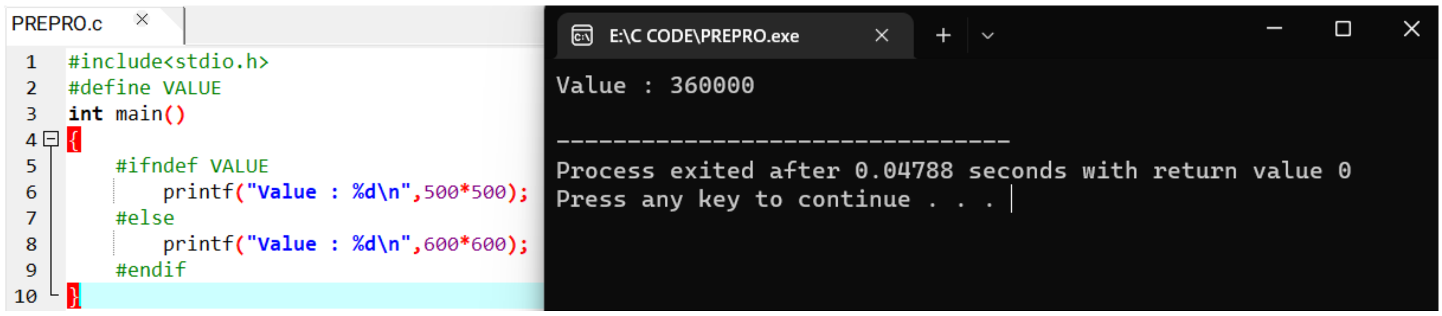
The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it executes the code otherwise #else code is executed, if present.


PREPRO.c

```c
#include<stdio.h>
//#define VALUE
int main()
{
    #ifndef VALUE
        printf("Value : %d\n",500*500);
    #else
        printf("Value : %d\n",600*600);
    #endif
}
```

E:\C CODE\PREPRO.exe

```
Value : 250000

--------------------------------
Process exited after 0.05208 seconds with return value 0
Press any key to continue . . .
```


PREPRO.c

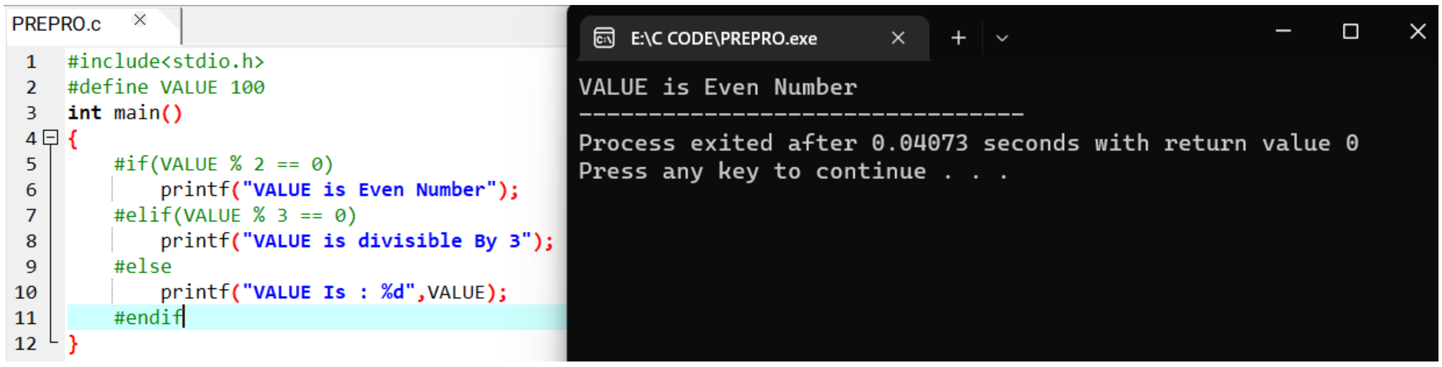
```c
#include<stdio.h>
#define VALUE
int main()
{
    #ifndef VALUE
        printf("Value : %d\n",500*500);
    #else
        printf("Value : %d\n",600*600);
    #endif
}
```

E:\C CODE\PREPRO.exe

```
Value : 360000

--------------------------------
Process exited after 0.04788 seconds with return value 0
Press any key to continue . . .
```

#if & #elif Directives:

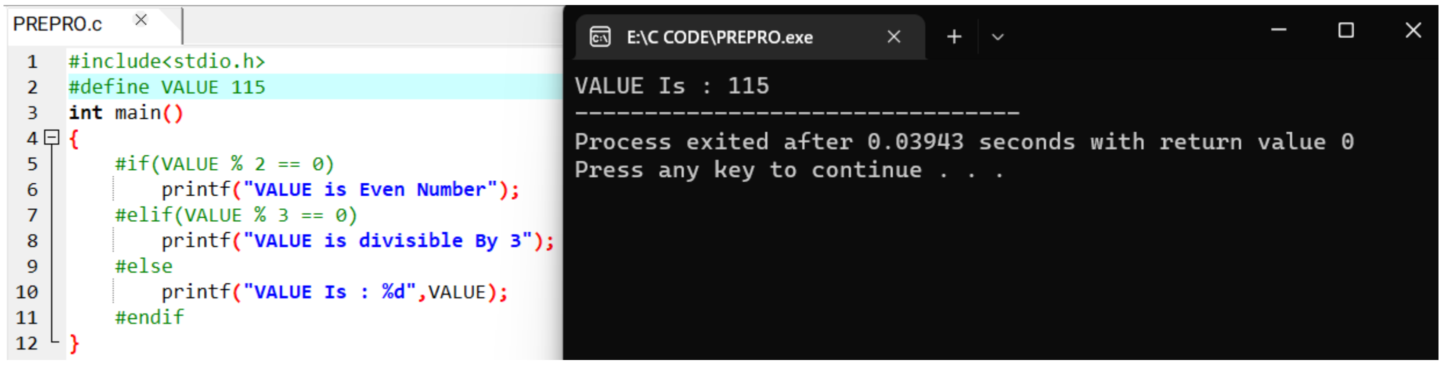
The #if directive can be used to test whether an expression evaluates to a non-zero value or not. If the result of the expression is non-zero, then subsequent lines upto a #else, #elif or #endif are compiled, otherwise they are skipped.


PREPRO.c

```c
#include<stdio.h>
#define VALUE 100
int main()
{
    #if(VALUE % 2 == 0)
        printf("VALUE is Even Number");
    #elif(VALUE % 3 == 0)
        printf("VALUE is divisible By 3");
    #else
        printf("VALUE Is : %d",VALUE);
    #endif
}
```

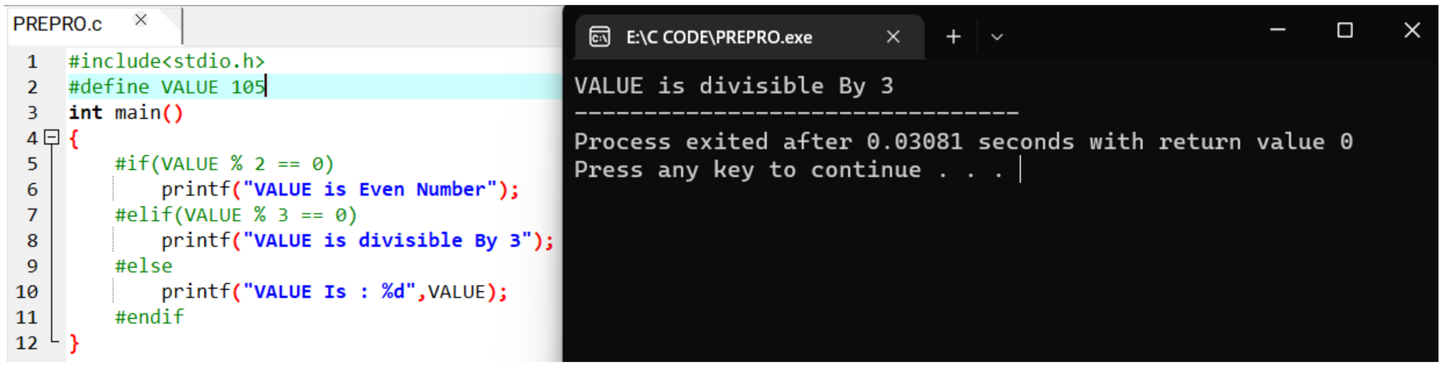
E:\C CODE\PREPRO.exe

```
VALUE is Even Number
--------------------------------
Process exited after 0.04073 seconds with return value 0
Press any key to continue . . .
```



```c
#include<stdio.h>
#define VALUE 115
int main()
{
    #if(VALUE % 2 == 0)
        printf("VALUE is Even Number");
    #elif(VALUE % 3 == 0)
        printf("VALUE is divisible By 3");
    #else
        printf("VALUE Is : %d",VALUE);
    #endif
}
```

```
VALUE Is : 115
--------------------------------
Process exited after 0.03943 seconds with return value 0
Press any key to continue . . .
```

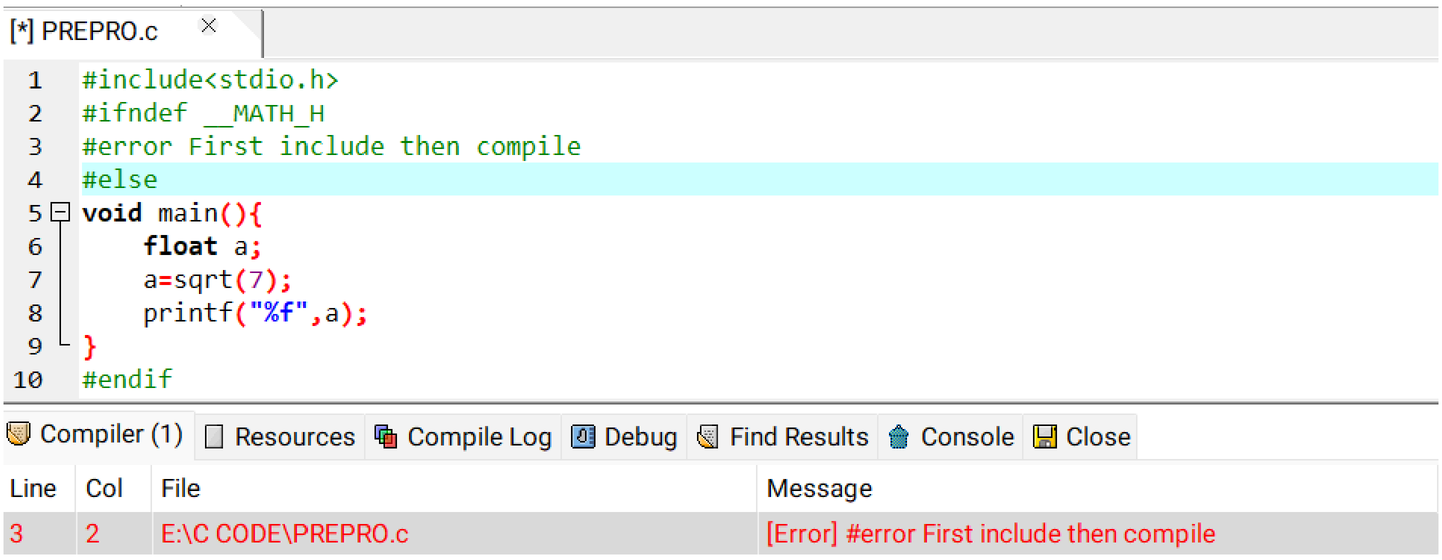


```c
#include<stdio.h>
#define VALUE 105
int main()
{
    #if(VALUE % 2 == 0)
        printf("VALUE is Even Number");
    #elif(VALUE % 3 == 0)
        printf("VALUE is divisible By 3");
    #else
        printf("VALUE Is : %d",VALUE);
    #endif
}
```

```
VALUE is divisible By 3
--------------------------------
Process exited after 0.03081 seconds with return value 0
Press any key to continue . . .
```

#error In C:

The #error preprocessor directive indicates error. The compiler gives fatal error if #error directive is found and skips further compilation process.



```c
#include<stdio.h>
#ifndef __MATH_H
#error First include then compile
#else
void main(){
    float a;
    a=sqrt(7);
    printf("%f",a);
}
#endif
```

Line	Col	File	Message
3	2	E:\C CODE\PREPRO.c	[Error] #error First include then compile

4. Miscellaneous Directives:

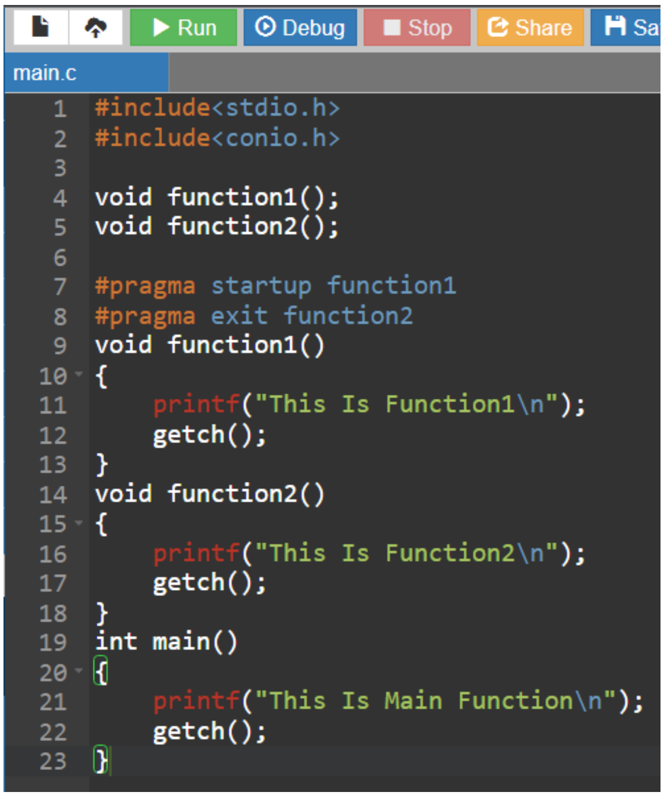
There are two types of directives are there #undef, #pragma

#undef Directive In C Preprocessing:

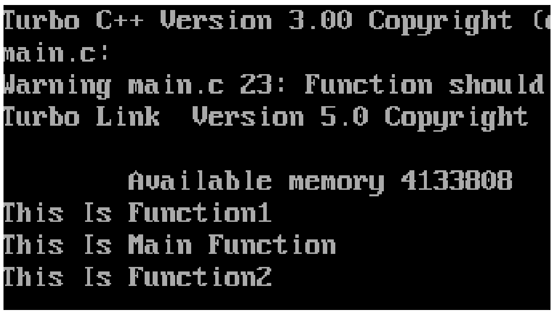
it may be desirable to cause a defined name to become ‘undefined’. This can be accomplished by means of the #undef directive. In order to undefine a macro that has been earlier #defined, the directive.

#pragma Directive In C Preprocessing:

The #pragma preprocessor directive is used to provide additional information to the compiler. The #pragma directive is used by the compiler to offer machine or operating-system feature. These are the following #prgama elements #pragma argsused, #pragma exit, #pragma hdrfile, #pragma hdrstop, #pragma inline, #pragma option, #pragma saveregs, #pragma startup, #pragma warn.



```
#include<stdio.h>
#include<conio.h>

void function1();
void function2();

#pragma startup function1
#pragma exit function2
void function1()
{
    printf("This Is Function1\n");
    getch();
}
void function2()
{
    printf("This Is Function2\n");
    getch();
}
int main()
{
    printf("This Is Main Function\n");
    getch();
}
```



```
Turbo C++ Version 3.00 Copyright (
main.c:
Warning main.c 23: Function should
Turbo Link  Version 5.0 Copyright

        Available memory 4133808
This Is Function1
This Is Main Function
This Is Function2
```

Note that the functions funtion1( ) and function2( ) should neither receive nor return any value. If we want two functions to get executed at startup then their pragmas should be defined in the reverse order in which you want to get them called.

<u>Header File:</u>

A header file is a file with extension (.h). a general practice in C or C++ program is that we keep all the constants, macros, system wide global variable, and function prototypes in header files and include that header file whatever it is required.

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

If a header file happens to be twice, the compiler will process its content twice and may result an error. the standard way to prevent this is to enclose the entire real contents of the header file in a conditional as follows.