

The Preprocessor

Chapter 14

You will be able to

- Use preprocessor directives correctly
- Define simple macros for the preprocessor.

- Any line beginning with `#` is a preprocessor directive.
- The preprocessor acts on these directives and removes the lines before compilation begins for real.
 - Once a separate program that ran before the compiler.
 - Now typically integrated with the compiler, but still a distinct step prior to compilation.

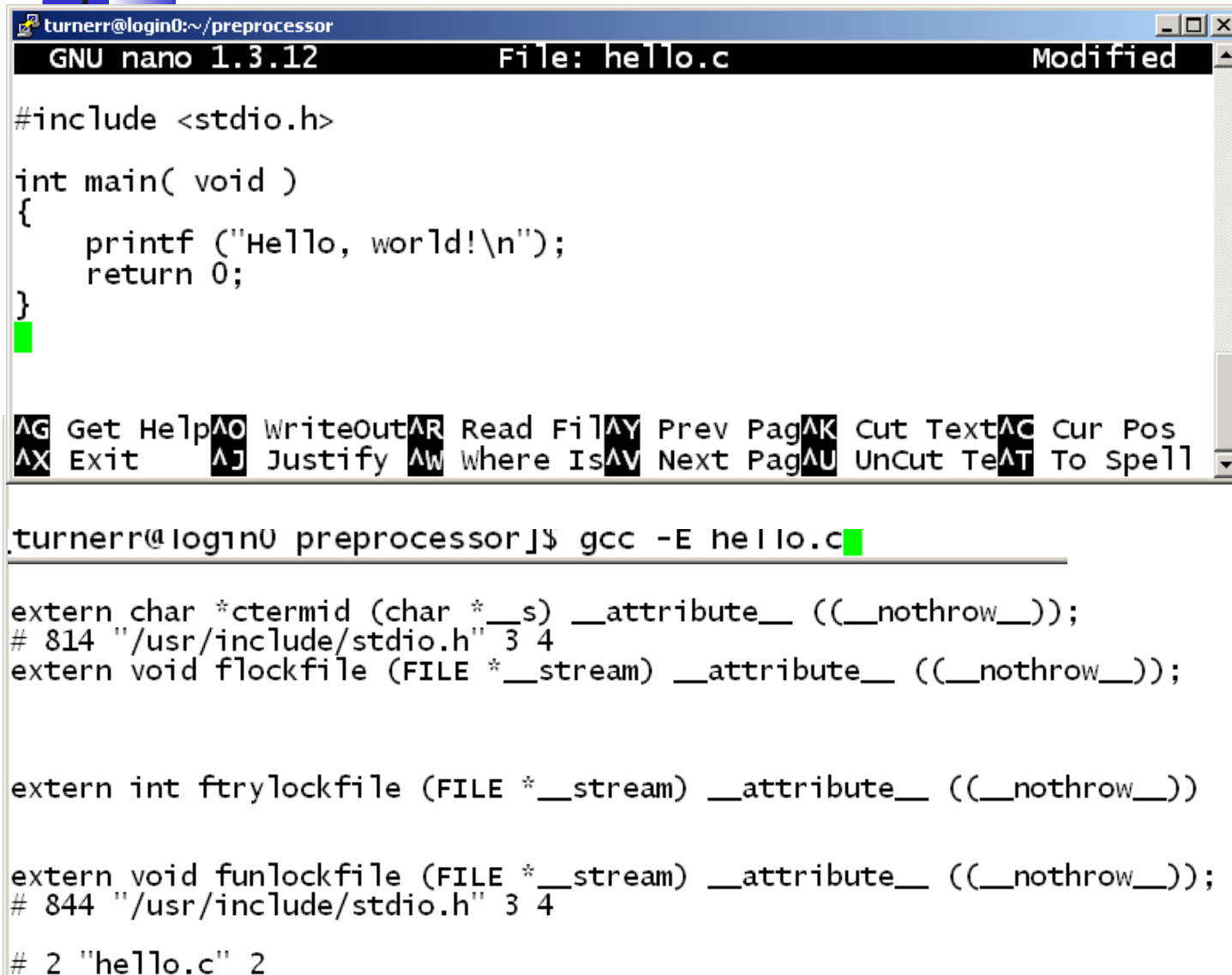
- All preprocessor commands modify the text of the source file
 - produces an intermediate file that the actual compiler processes.
- The effect is that the compiler sees a different file than the one you see.
 - Can lead to errors that are very difficult to find.



Caution

- To see the intermediate file with gcc on Unix use `-E` in the command line.
 - File will not be compiled.
 - Only the preprocessor will run.
 - Output goes to stdout.

Looking at Preprocessor Output



```
turnerr@login0:~/preprocessor
GNU nano 1.3.12 File: hello.c Modified
#include <stdio.h>

int main( void )
{
    printf ("Hello, world!\n");
    return 0;
}

AG Get Help AO WriteOut AR Read Fil AY Prev Pag AK Cut Text AC Cur Pos
AX Exit AJ Justify AW Where Is AV Next Pag AU UnCut Te AT To Spell

turnerr@login0 preprocessor]$ gcc -E hello.c

extern char *ctermid (char *__s) __attribute__ ((__nothrow__));
# 814 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__));

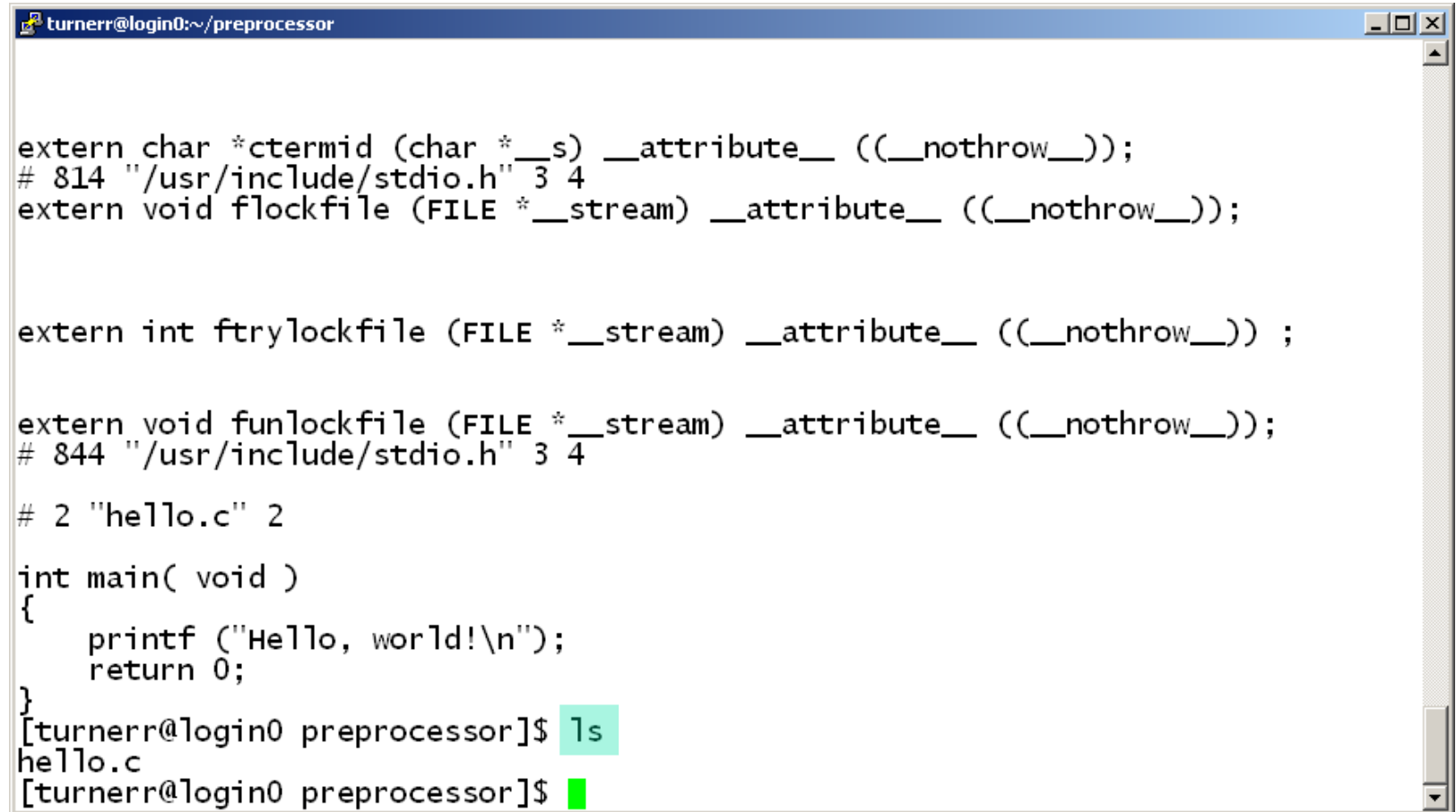
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__));

extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));
# 844 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2
```

This is the end
of the output
On your computer
you would have
to scroll up to
see the rest

After Scrolling Down



```
turnerr@login0:~/preprocessor

extern char *ctermid (char *__s) __attribute__ ((__nothrow__));
# 814 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__));

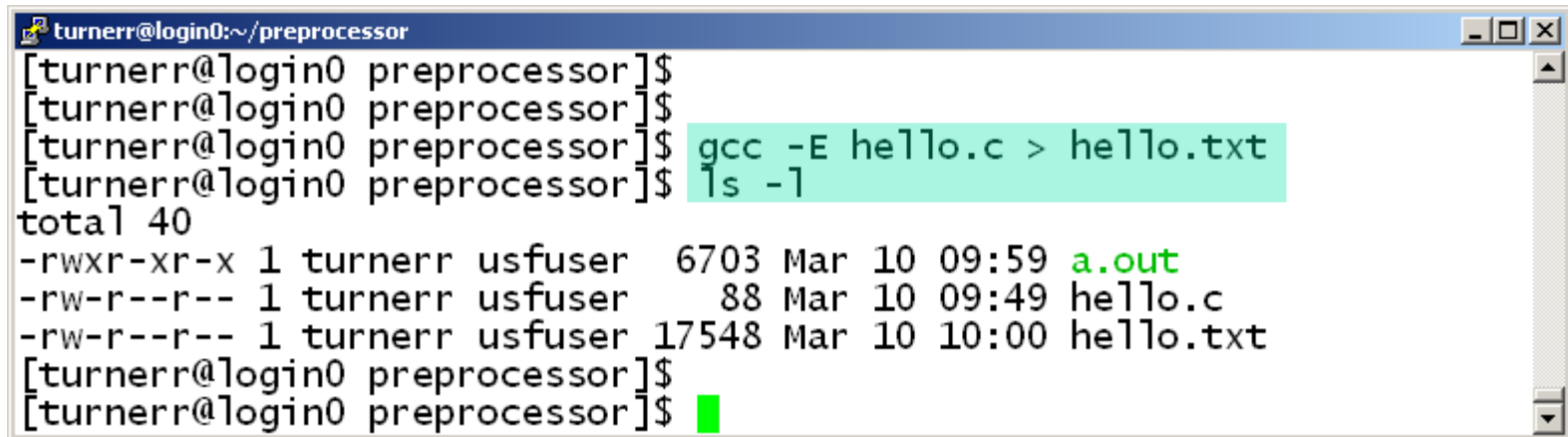
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__)) ;

extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));
# 844 "/usr/include/stdio.h" 3 4
# 2 "hello.c" 2
int main( void )
{
    printf ("Hello, world!\n");
    return 0;
}
[turnerr@login0 preprocessor]$ ls
hello.c
[turnerr@login0 preprocessor]$
```

hello.c is still the only file in the working directory.

Writing to a File

- Redirect stdout to a file.
 - `gcc -E hello.c > hello.txt`

A terminal window titled 'turnerr@login0:~/preprocessor' showing a series of commands and their outputs. The commands entered are '[turnerr@login0 preprocessor]\$' (repeated three times), 'gcc -E hello.c > hello.txt', and 'ls -l'. The output shows the execution of the gcc command, followed by the output of the ls command listing files 'a.out', 'hello.c', and 'hello.txt' with their respective permissions, owners, sizes, and timestamps. The command prompt is currently at the end of the last line.

```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -E hello.c > hello.txt
[turnerr@login0 preprocessor]$ ls -l
total 40
-rwxr-xr-x 1 turnerr usfuser 6703 Mar 10 09:59 a.out
-rw-r--r-- 1 turnerr usfuser 88 Mar 10 09:49 hello.c
-rw-r--r-- 1 turnerr usfuser 17548 Mar 10 10:00 hello.txt
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
```

Examine hello.txt with your editor.

`#define identifier replacement-list`

Space No = No ;

Where *identifier* appears in the code,
it will be replaced by *replacement-
list*.

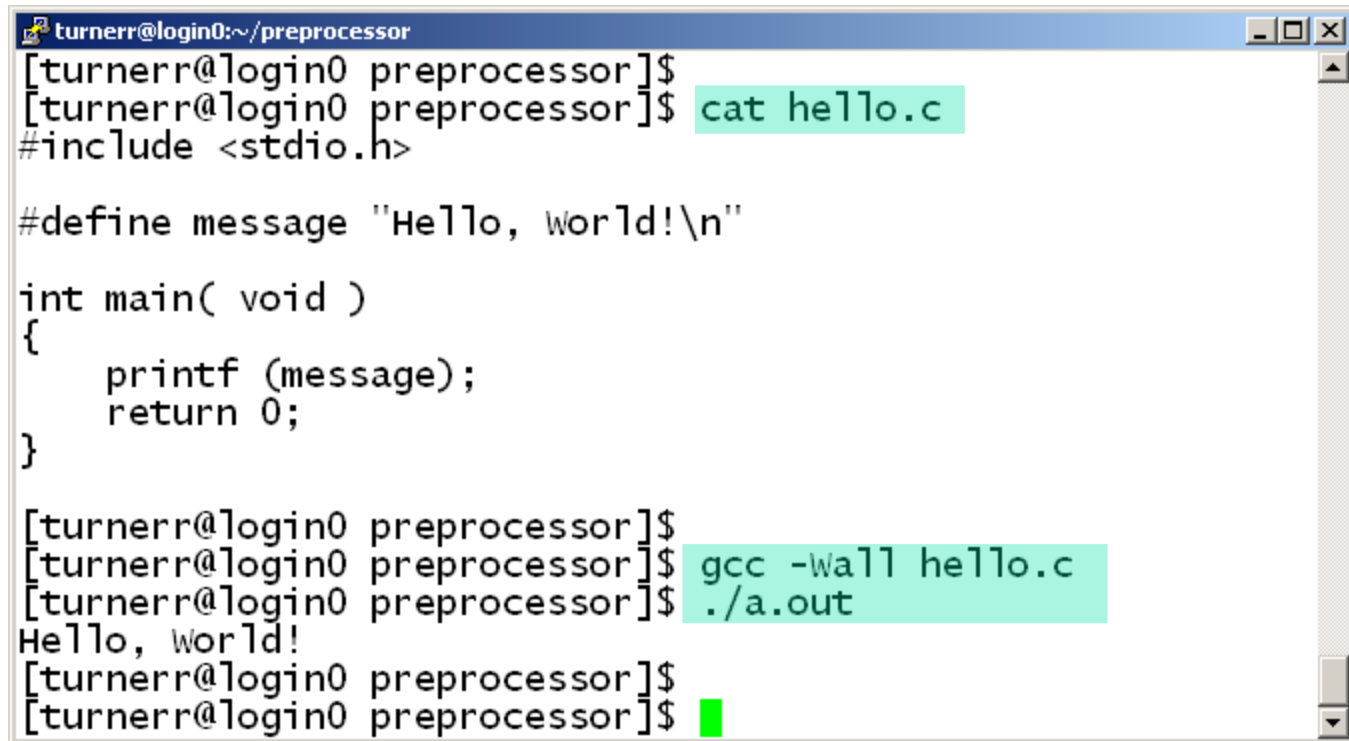
replacement-list can have multiple words including identifiers,
keywords, numeric constants, characters, operators, and
punctuation.

Macro Definition: Example



```
turnerr@login0:~/preprocessor
GNU nano 1.3.12      File: hello.c      Modified
#include <stdio.h>
#define message "Hello, World!\n"
int main( void )
{
    printf (message);
    return 0;
}
^G Get Help ^O Write Out ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit    ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Run the Program



```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ cat hello.c
#include <stdio.h>

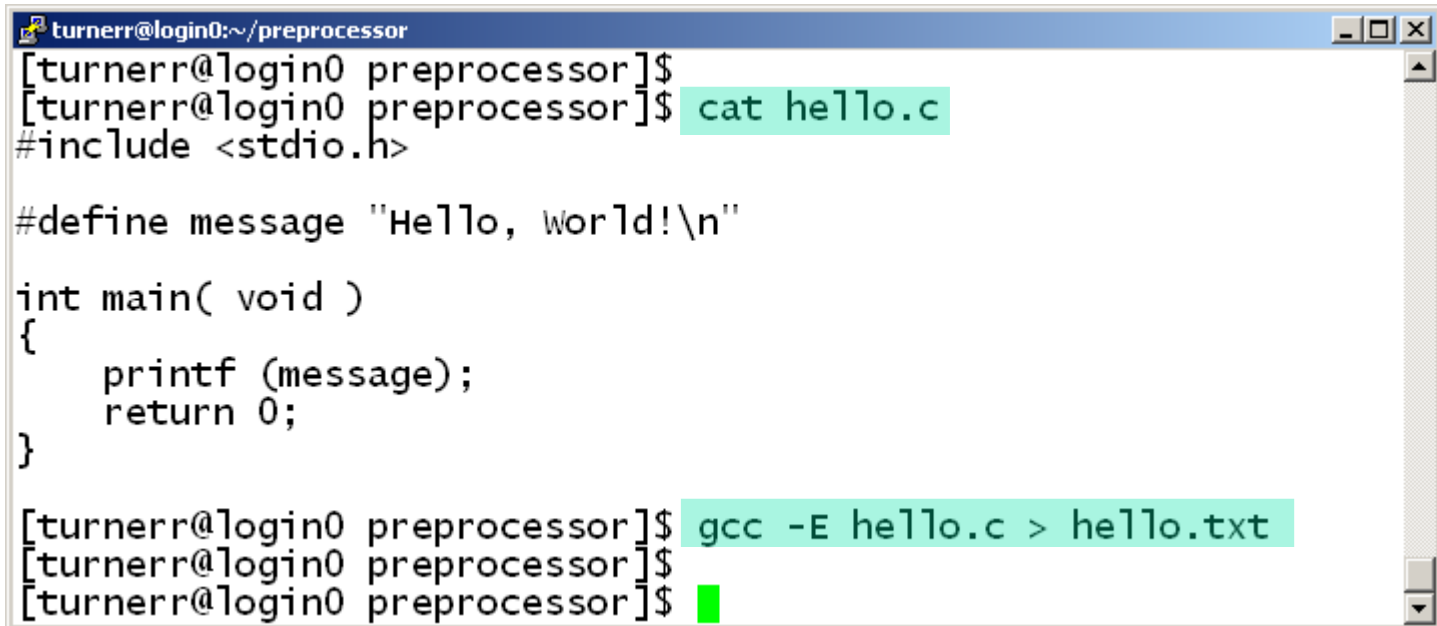
#define message "Hello, World!\n"

int main( void )
{
    printf (message);
    return 0;
}

[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -Wall hello.c
[turnerr@login0 preprocessor]$ ./a.out
Hello, World!
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
```

Works the same.

Look at the preprocessor output

A terminal window titled 'turnerr@login0:~/preprocessor' showing the output of the C preprocessor. The user has entered 'cat hello.c' and 'gcc -E hello.c > hello.txt'. The output shows the preprocessed code for a program that prints 'Hello, World!'.

```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ cat hello.c
#include <stdio.h>

#define message "Hello, World!\n"

int main( void )
{
    printf (message);
    return 0;
}

[turnerr@login0 preprocessor]$ gcc -E hello.c > hello.txt
[turnerr@login0 preprocessor]$
```

Preprocessor Output

```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ cat hello.c
#include <stdio.h>

#define message "Hello, world!\n"

int main( void )
{
    printf (message);
    return 0;
}

[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ tail hello.txt

# 2 "hello.c" 2

int main( void )
{
    printf ("Hello, world!\n");
    return 0;
}
[turnerr@login0 preprocessor]$
```

What we wrote.

What the compiler saw.

Try This



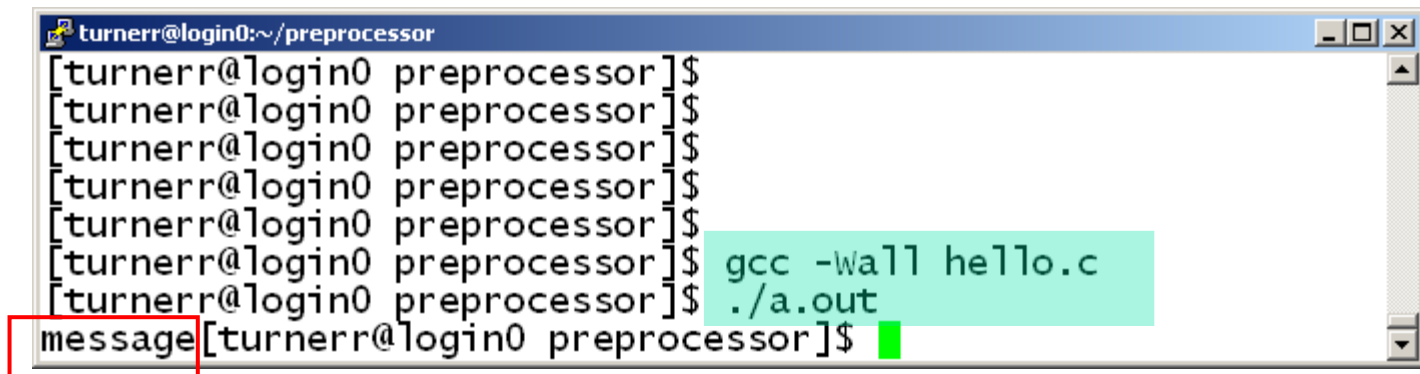
```
turnerr@login0:~/preprocessor
GNU nano 1.3.12      File: hello.c      Modified
#include <stdio.h>

#define message "Hello, World!\n"

int main( void )
{
    printf ("message");
    return 0;
}

^G Get Help ^O Write Out ^R Read From File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit    ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

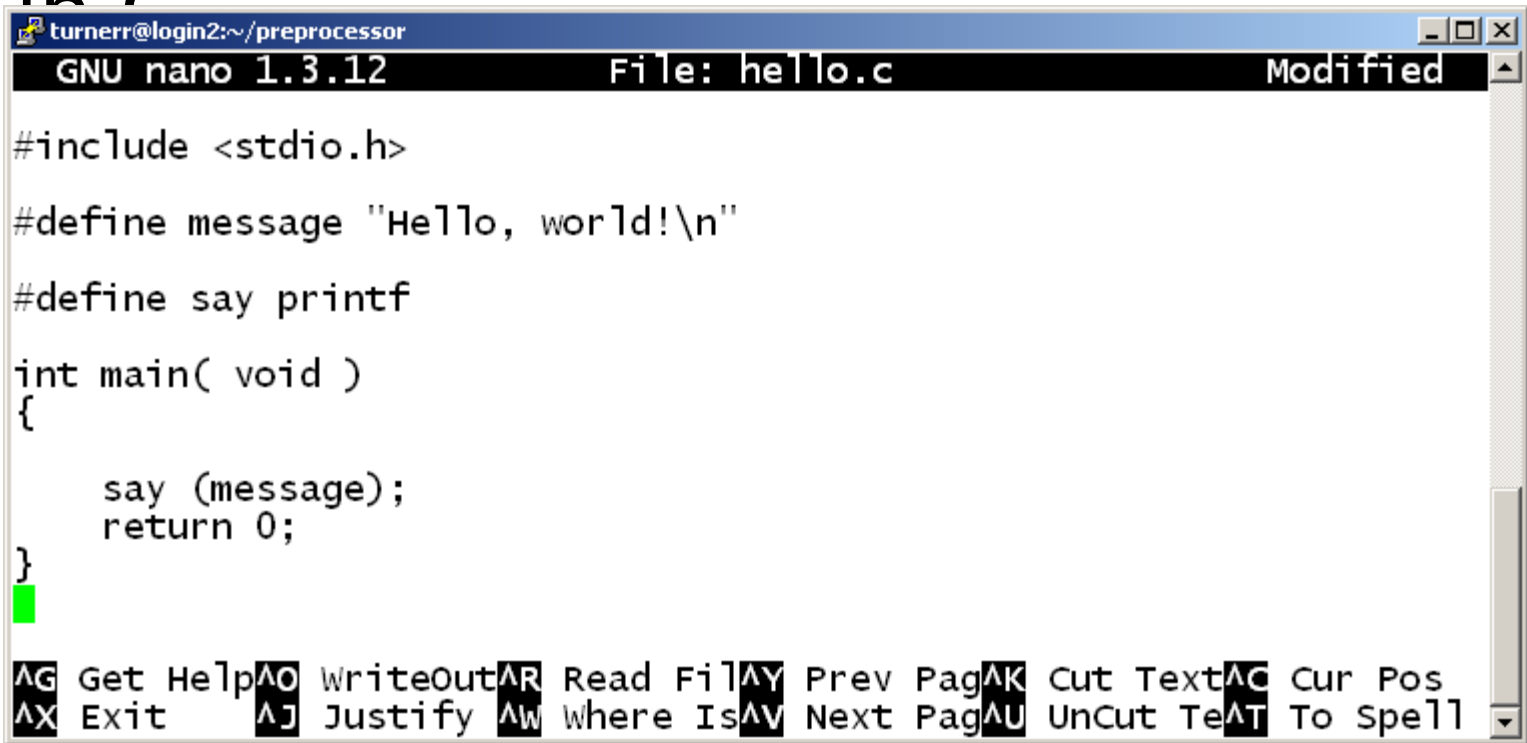
Program Running



```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -Wall hello.c
[turnerr@login0 preprocessor]$ ./a.out
message[turnerr@login0 preprocessor]$
```

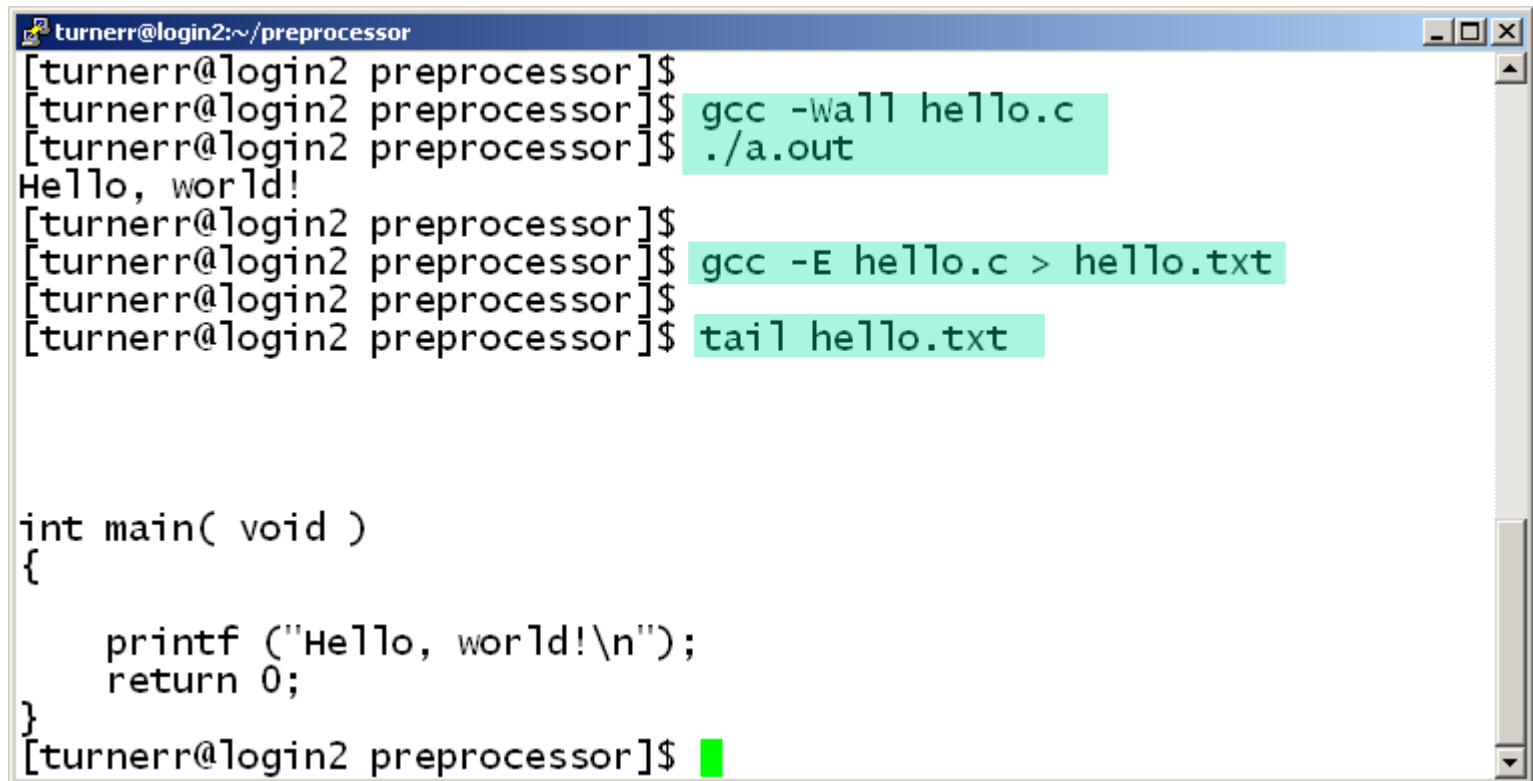
No substitution inside string literals.

- You can use macros to do cute things in C



```
turnerr@login2:~/preprocessor
GNU nano 1.3.12      File: hello.c      Modified
#include <stdio.h>
#define message "Hello, world!\n"
#define say printf
int main( void )
{
    say (message);
    return 0;
}
^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit    ^J Justify ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```


Macro Abuse

A terminal window titled 'turnerr@login2:~/preprocessor' showing a series of commands and their outputs. The commands are: 'gcc -Wall hello.c', './a.out', 'gcc -E hello.c > hello.txt', and 'tail hello.txt'. The output of './a.out' is 'Hello, world!'. Below the commands, the source code of 'hello.c' is displayed, showing a simple 'main' function that prints 'Hello, world!'. The terminal window has a blue title bar and standard window controls (minimize, maximize, close) in the top right corner.

```
turnerr@login2:~/preprocessor
[turnerr@login2 preprocessor]$ gcc -Wall hello.c
[turnerr@login2 preprocessor]$ ./a.out
Hello, world!
[turnerr@login2 preprocessor]$ gcc -E hello.c > hello.txt
[turnerr@login2 preprocessor]$ tail hello.txt

int main( void )
{
    printf ("Hello, world!\n");
    return 0;
}
[turnerr@login2 preprocessor]$
```

Don't do this!

Call a printf a printf.

- This rule applies to some uses shown in our textbook:

```
#define BOOL int
#define true 1
#define false 0
```

- Many C programmers like to use macros like these.
 - “Makes the program easier to understand.”
- My opinion: Better to see what the compiler sees.

- Define meaningful names for numerical values.
- `#define FREEZING_PT 32.0`



Example: celsius.c

```
#include <stdio.h>


#define FREEZING_PT 32.0
#define SCALE_FACTOR (5.0 / 9.0)

int main( void )
{
    double fahrenheit = 0.0;
    double celsius = 0.0;
    printf ("Enter Fahrenheit temperature: ");
    scanf("%lg", &fahrenheit);
    celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
    printf ("Celsius equivalent is: %.1f\n", celsius);
    return 0;
}
```

Example: celsius.c

```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -Wall celsius.c
[turnerr@login0 preprocessor]$ ./a.out
Enter Fahrenheit temperature: 85
Celsius equivalent is: 29.4
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -E celsius.c > celsius.txt
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ tail celsius.txt
int main( void )
{
    double fahrenheit = 0.0;
    double celsius = 0.0;
    printf ("Enter Fahrenheit temperature: ");
    scanf("%lg", &fahrenheit);
    celsius = (fahrenheit - 32.0) * (5.0 / 9.0);
    printf ("Celsius equivalent is: %.1f\n", celsius);
    return 0;
}
[turnerr@login0 preprocessor]$
```

A Simple Mistake



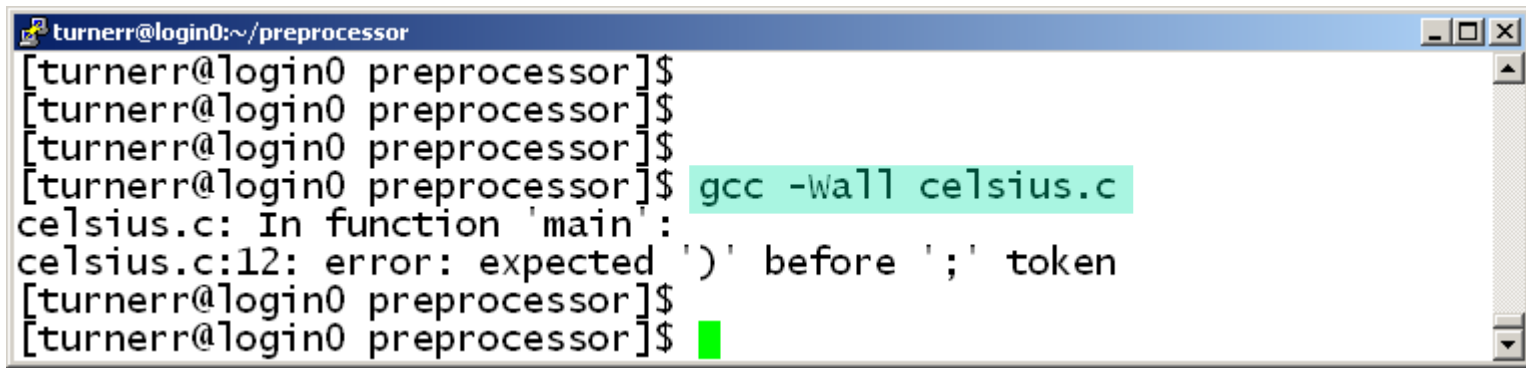
```
turnerr@login0:~/preprocessor
GNU nano 1.3.12      File: celsius.c      Modified
#include <stdio.h>

#define FREEZING_PT 32.0;
#define SCALE_FACTOR (5.0 / 9.0);

int main( void )
{
    double fahrenheit = 0.0;
    double celsius = 0.0;
    printf ("Enter Fahrenheit temperature: ");
    scanf ("%lg", &fahrenheit);
    celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
    printf ("Celsius equivalent is: %.1f\n", celsius);
    return 0;
}

^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit    ^J Justify ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```

A Simple Mistake




```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -Wall celsius.c
celsius.c: In function 'main':
celsius.c:12: error: expected ')' before ';' token
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
```

Look at line 12 of celsius.c.

What's wrong with it?

Look at Error Line



```
turnerr@login0:~/preprocessor
GNU nano 1.3.12      File: celsius.c

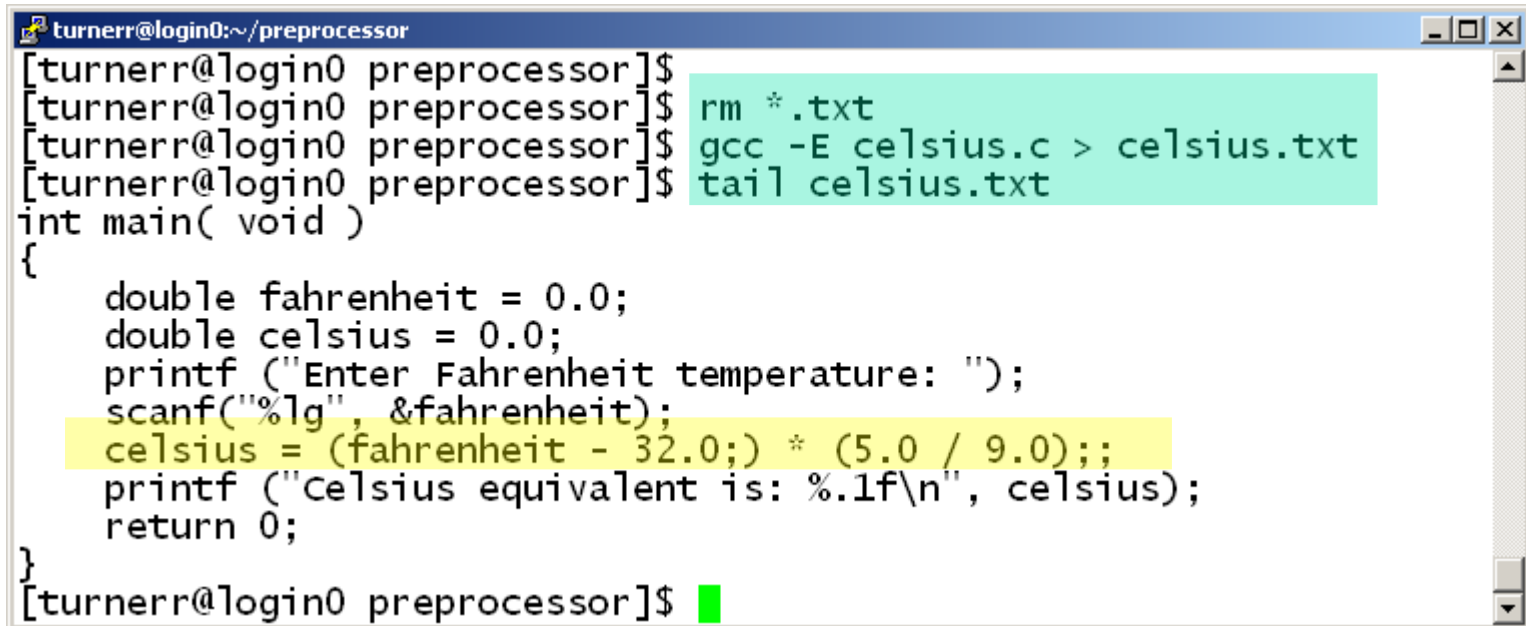
#include <stdio.h>

#define FREEZING_PT 32.0;
#define SCALE_FACTOR (5.0 / 9.0);

int main( void )
{
    double fahrenheit = 0.0;
    double celsius = 0.0;
    printf ("Enter Fahrenheit temperature: ");
    scanf("%lg", &fahrenheit);
    celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
    printf ("Celsius equivalent is: %.1f\n", celsius);
    return 0;
}

[ line 12/17 (70%), col 1/57 (1%), char 237/366 (64%) ]
^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit     ^J Justify ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```


What the Compiler Saw



```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ rm *.txt
[turnerr@login0 preprocessor]$ gcc -E celsius.c > celsius.txt
[turnerr@login0 preprocessor]$ tail celsius.txt
int main( void )
{
    double fahrenheit = 0.0;
    double celsius = 0.0;
    printf ("Enter Fahrenheit temperature: ");
    scanf("%lg", &fahrenheit);
    celsius = (fahrenheit - 32.0;) * (5.0 / 9.0);;
    printf ("Celsius equivalent is: %.1f\n", celsius);
    return 0;
}
[turnerr@login0 preprocessor]$
```

- What we have seen so far are *simple* macros.
 - Direct substitution of the replacement list for the macro identifier where the identifier appears in the code (except in string literals and comments.)

- Looks somewhat like a function definition.
- Definition includes parameters.
 - You must provide values to substitute for parameters when the macro is used.
- Example:

```
#define MAX(x,y) ((x)>(y)?(x):(y))
```



Parameterized Macros

```
#include <stdio.h>

#define MAX(x,y) ((x)>(y)?(x):(y))

int main ( void )
{
    int n1 = 0;
    int n2 = 0;

    printf ("Enter an integer: ");
    scanf ("%d", &n1);
    printf ("Enter another integer: ");
    scanf ("%d", &n2);

    printf ("The maximum value is %d\n", MAX(n1,n2));

    return 0;
}
```

macro_demo.c

```
turnerr@login0:~/preprocessor
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -Wall max.c
[turnerr@login0 preprocessor]$ ./a.out
Enter an integer: 17
Enter another integer: 33
The maximum value is 33
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$ gcc -E max.c > max.txt
[turnerr@login0 preprocessor]$ tail max.txt

    printf ("Enter an integer: ");
    scanf ("%d", &n1);
    printf ("Enter another integer: ");
    scanf ("%d", &n2);

    printf ("The maximum value is %d\n", ((n1)>(n2)?(n1):(n2)));

    return 0;
}
[turnerr@login0 preprocessor]$
[turnerr@login0 preprocessor]$
```

- My opinion:
 - Parameterized macros are bad programming practice.
 - Potential for problems far outweighs any potential benefits.
- Don't use them in programs for this course.
 - Programming Style Guidelines prohibit use.
 - Usage will result in point deductions for project.

- Read about details in the textbook.
 - Pages 321 – 333
- Don't worry about remembering.
 - Not important unless you are doing things with macros that you shouldn't do in this course.
 - Look up if you need in future work.
 - “Not on the test”