MACROS

The C Preprocessor – Introduction

The preprocessor deals strictly in text. Here is a list of the standard preprocessor directives and macros excluding #define.

- #include <*filename*>, #include "*filename*" – expands into contents of the given file into current position. The <> means to search the standard include path for the file while the "" means to search the current directory.
- #error *message*, #warning *message* – Causes the compiler to either halt or issue a warning if this line is reached. Useful for debugging.
- #pragma – Passes options to the compiler. Options change from compiler to compiler
- #if *condition*, #elsif *condition*, #endif – Includes or excludes a block of text dependent on the value of the condition. #if 0 is useful for removing a block of code from complication.
- __FILE__, __LINE__, __DATE__, __func__ – these macros expand into strings representing the current file, line, date, and in c99, the current function.

The C Preprocessor – #define basics

#define macros

```
#define SOME_LABEL  To some list of literals
#define MIN(x,y) ((x) < (y) ? (x) : (y))
#define printf(x,...) fprintf(stdout, x, __VA_ARGS__)
```

- Macros can be used for quick and dirty constants.

  Though is it often preferable nowadays to do:
  const T name = value;
  where T is a type. This is because this creates a variable with type info.

- Macros can be used to like functions. Think of them as a patterned search and replace.
  Some simple functions are often implemented as just a #define macro. Common examples are "min" and "printf." Many libraries implement them in a fashion similar to the examples above.

  You can even do variable argument macros by putting an elipse ("…") in the parameter list. The tag __VA_ARGS__ expands to all the extra arguments with the comma. (You may notice a problem with our definition of "printf" given our explanation of __VA_ARGS__. Most compilers extend the behavior of __VA_ARGS__ expansion to make up for this problem.)

The C Preprocessor – #define fun!

#define macros string manipulation operators

```
#define concat(x,y)  x##y
#define mkstr(x)  #x
```

- ## performs a concatenation of the two preprocessor arguments. This may be useful for autogenerating mangled names or some other sort of textual manipulation. Thus,

  concat(wordA,wordB)

  is equivalent to

  wordAwordB

- # makes the following macro argument a string (with quotes). It also chomps whitespace so everything is only 1 space. Thus:

  mkstr(bu   ha ha ha   me lo lo   weeeeeeee)

  becomes

  “bu ha ha ha me lo lo weeeeeeee”

Macros vs. Functions: Argument Evaluation

- Macros and functions may behave differently if an argument is referenced multiple times:
  - a function argument is evaluated once, before the call
  - a macro argument is evaluated each time it is encountered in the macro body.
- Example:

`int dbl(x) { return x + x;}`
`…`
`u = 10; v = dbl(u++);`
`printf("u = %d, v = %d", u, v);`
 	
prints: **u = 11, v = 20**
	`#define  Dbl(x)    x + x`
	`…`
	`u = 10; v = Dbl(u++);`
	`printf("u = %d, v = %d", u, v);`
	
	prints: **u = 12, v = 21**

Dbl(u++) expands to: u++ + u++

Properties of macros

- Macros may be nested
  - in definitions, e.g.:

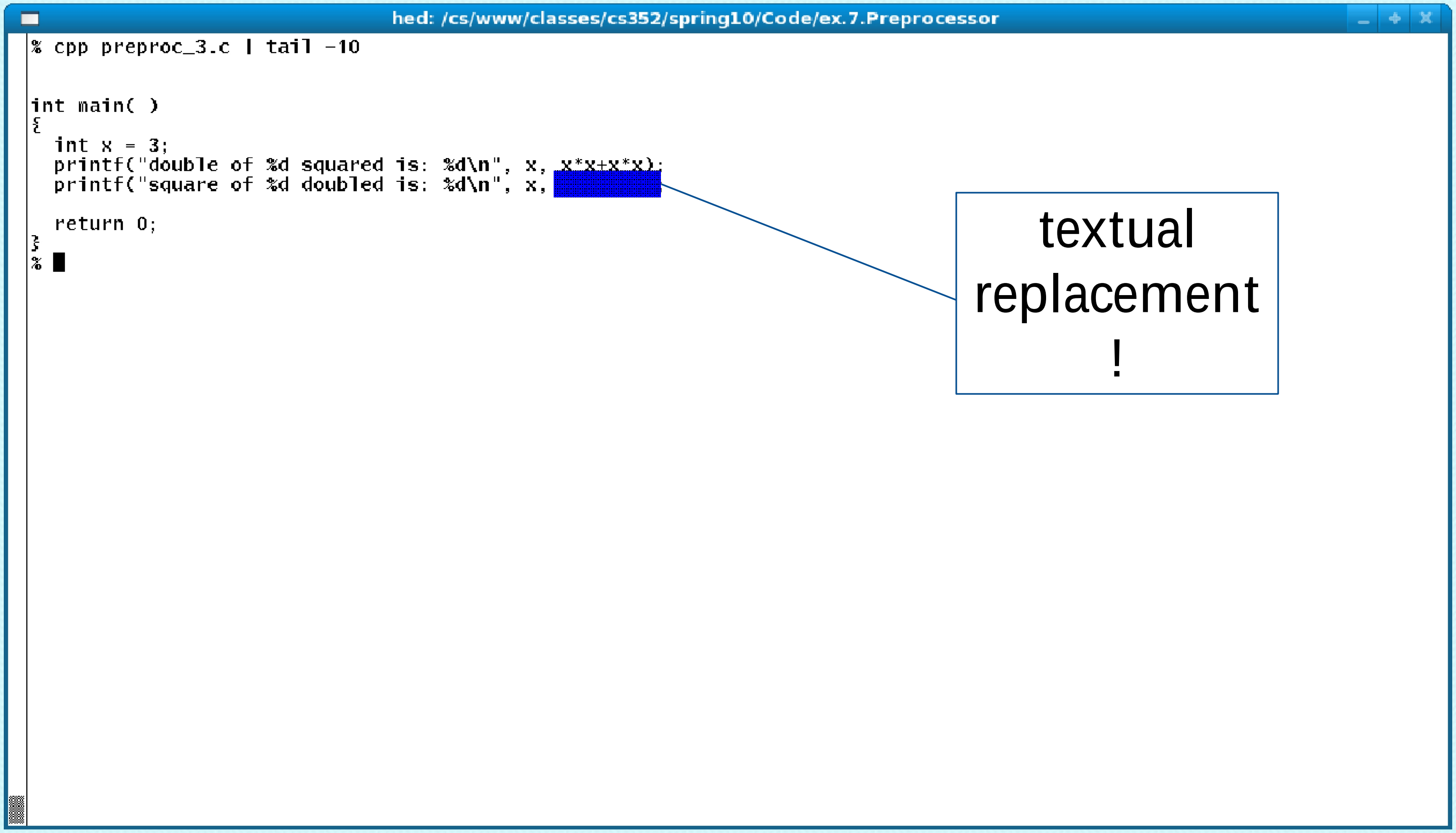
```
#define Pi          3.1416
#define Twice_Pi    2*Pi
```

  - in uses, e.g.:

```
#define double(x)   x+x
#define Pi  3.1416
...
if ( x > double(Pi) ) ...
```

- Nested macros are expanded recursively

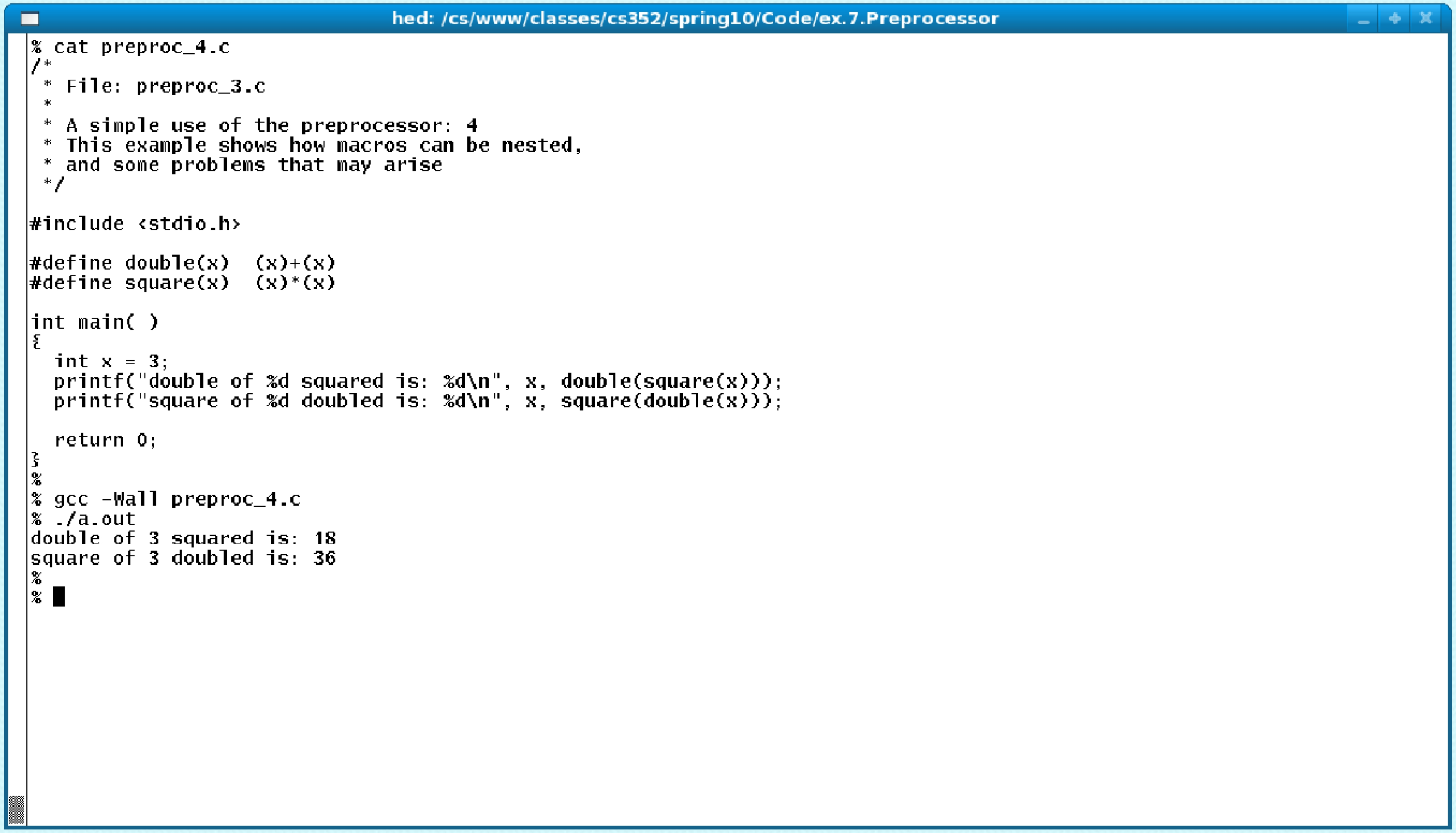
What happened?



```
% cpp preproc_3.c | tail -10


int main( )
{
  int x = 3;
  printf("double of %d squared is: %d\n", x, x*x+x*x);
  printf("square of %d doubled is: %d\n", x, [illegible]

  return 0;
}
% 
```

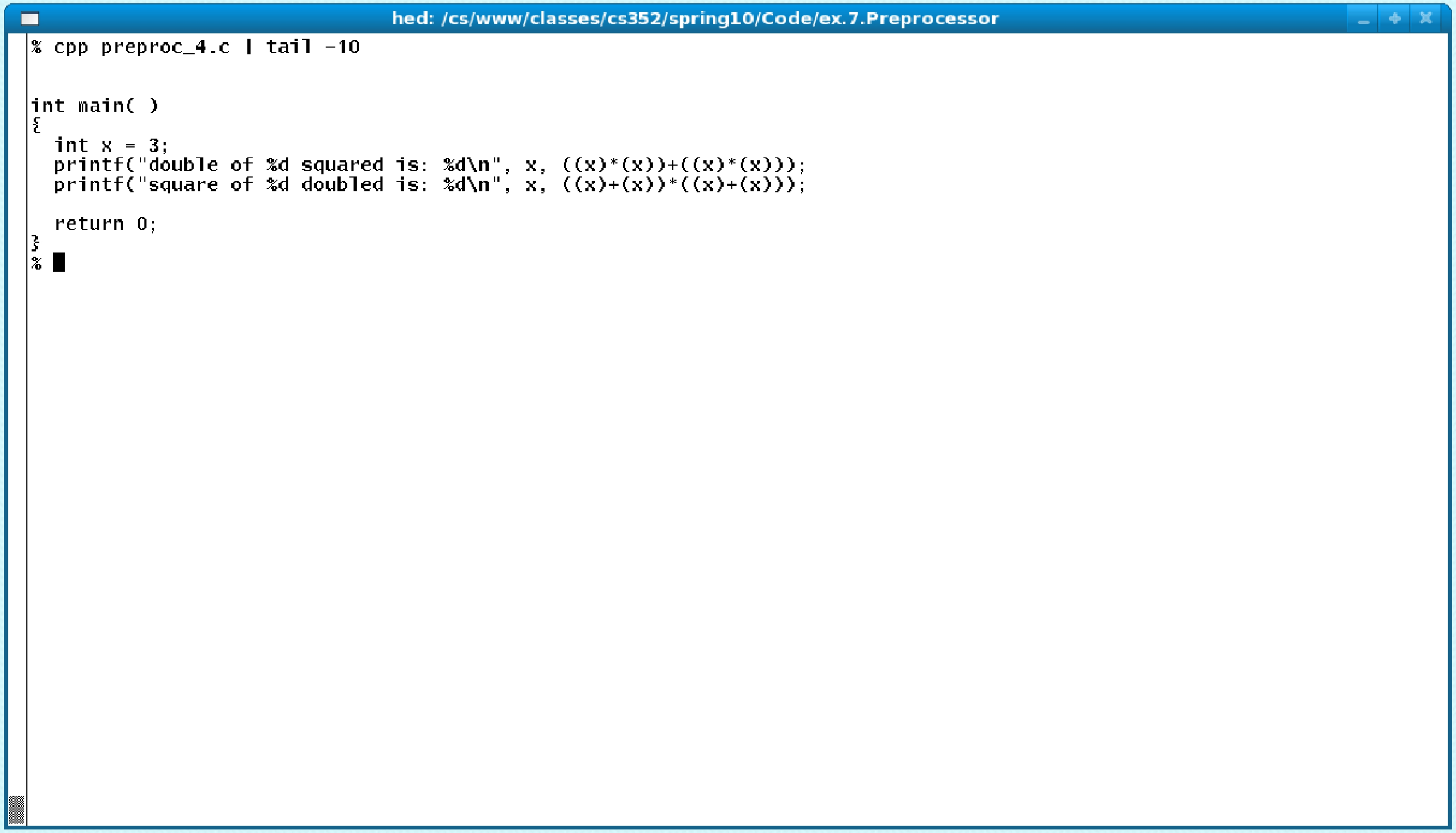
textual replacement !

Avoiding the problem


hed: /cs/www/classes/cs352/spring10/Code/ex.7.Preprocessor

```
% cat preproc_4.c
/*
 * File: preproc_3.c
 *
 * A simple use of the preprocessor: 4
 * This example shows how macros can be nested,
 * and some problems that may arise
 */

#include <stdio.h>

#define double(x)  (x)+(x)
#define square(x)  (x)*(x)

int main( )
{
  int x = 3;
  printf("double of %d squared is: %d\n", x, double(square(x)));
  printf("square of %d doubled is: %d\n", x, square(double(x)));

  return 0;
}
%
% gcc -Wall preproc_4.c
% ./a.out
double of 3 squared is: 18
square of 3 doubled is: 36
%
% █
```

What happened



hed: /cs/www/classes/cs352/spring10/Code/ex.7.Preprocessor

```
% cpp preproc_4.c | tail -10


int main( )
{
  int x = 3;
  printf("double of %d squared is: %d\n", x, ((x)*(x))+((x)*(x)));
  printf("square of %d doubled is: %d\n", x, ((x)+(x))*((x)+(x)));

  return 0;
}
% 
```