Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Standard Streams
Redirections & Pipes

Standard streams

Standard streams

- Standard Streams

Standard streams

- Standard Streams
    - **stdin**: an input stream associated with a device - your keyboard; **scanf(..)**

Standard streams

- Standard Streams
  - **stdin**: an input stream associated with a device - your keyboard; **scanf(..)**
  - **stdout**: an output stream associated with a device - your terminal; **printf(..)**

Standard streams

- Standard Streams
    - **stdin**: an input stream associated with a device - your keyboard; **scanf(..)**
    - **stdout**: an output stream associated with a device - your terminal; **printf(..)**
    - **stderr**: an output stream associated with your terminal just like stdout; **fprintf(stderr, "string\n")**

Standard streams

- Standard Streams
  - **stdin**: an input stream associated with a device - your keyboard; **scanf(..)**
  - **stdout**: an output stream associated with a device - your terminal; **printf(..)**
  - **stderr**: an output stream associated with your terminal just like stdout; **fprintf(stderr, "string\n")**
  - **dev/null**: a output stream associated with no device. This stream is generally used to make textual output disappear and not show up anywhere.

Redirections (1)

Redirections (1)

- Redirecting output
  - use the “**>**” symbol to redirect the output of a command

Redirections (1)

- Redirecting output
  - use the “**>**” symbol to redirect the output of a command
- Appending to a file
  - use the “**>>**” to append standard output to a file

Redirections (1)

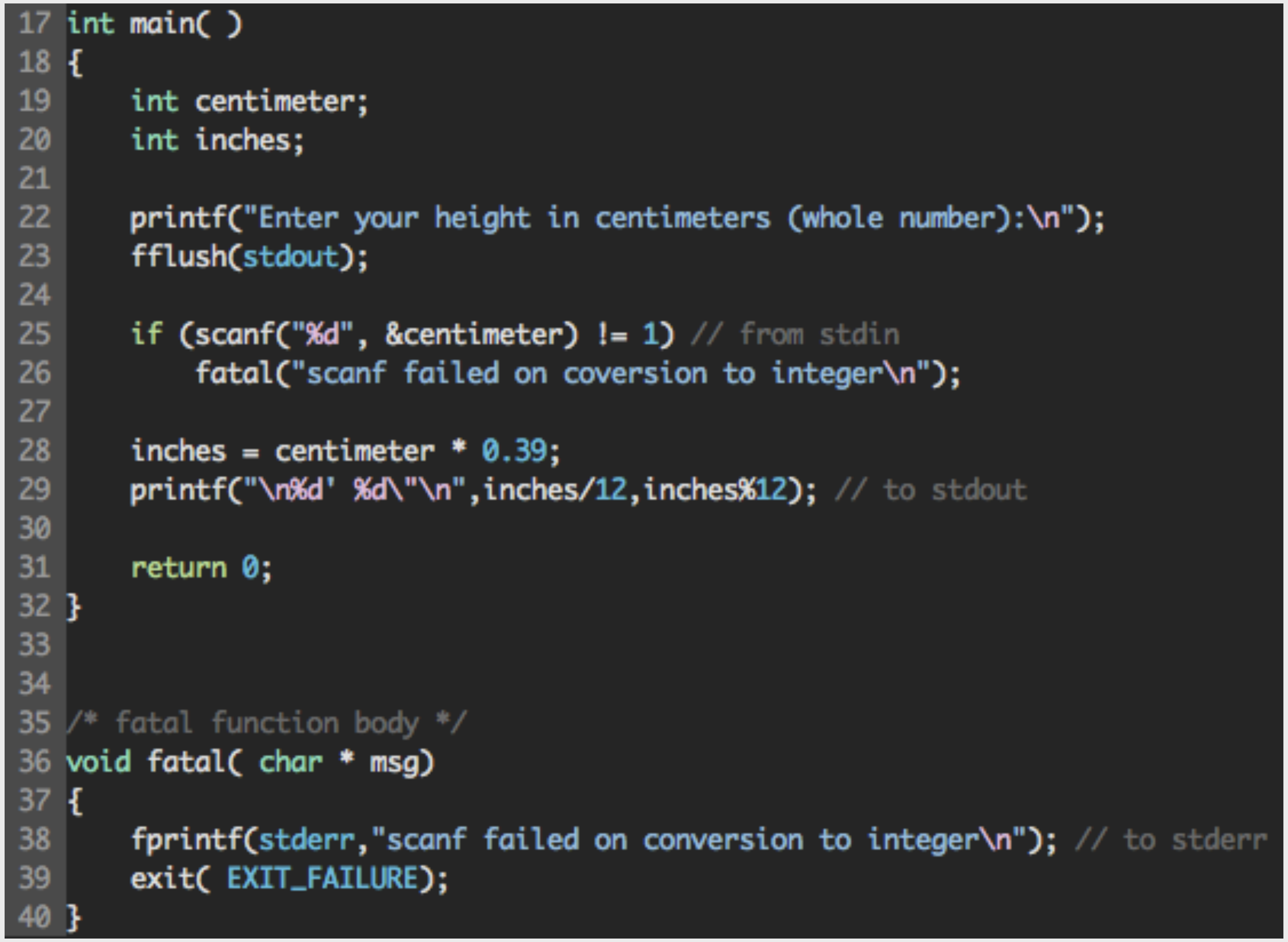
- Redirecting output
  - use the “**>**” symbol to redirect the output of a command
- Appending to a file
  - use the “**>>**” to append standard output to a file
- Redirecting input
  - use the “**<**” symbol to redirect the input of a command

Redirections (1)

- Redirecting output
  - use the "**>**" symbol to redirect the output of a command
- Appending to a file
  - use the "**>>**" to append standard output to a file
- Redirecting input
  - use the "**<**" symbol to redirect the input of a command
- Redirect output and error message
  - use teh "**>&**" to redirect both stdout and stderr to a file
  - redirection operations can be parenthesized
    - e.g., **(./a.out > out.txt) >& err.txt**

Redirections (2)

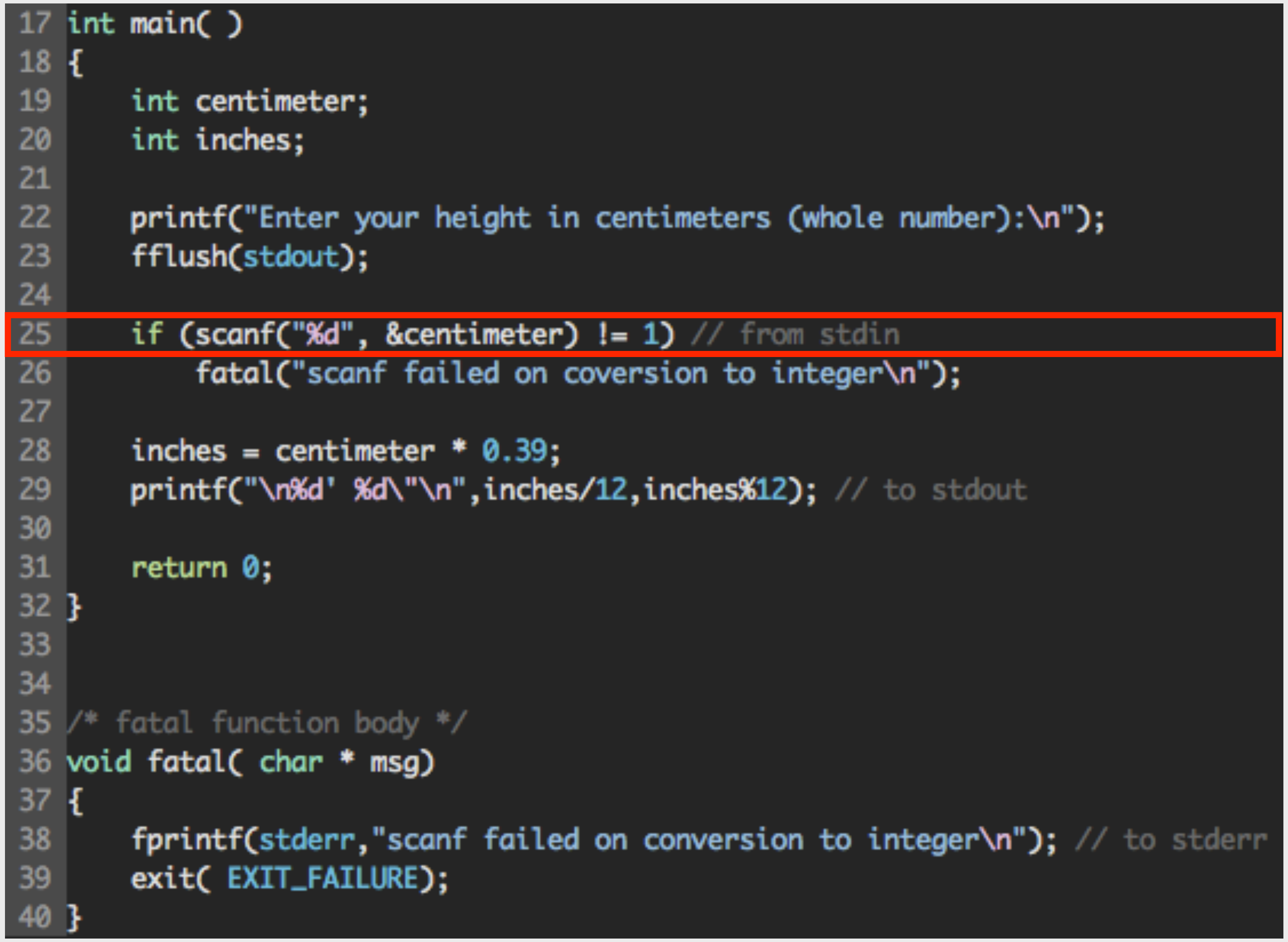
- Example: redir/example.c



```
int main( )
{
    int centimeter;
    int inches;

    printf("Enter your height in centimeters (whole number):\n");
    fflush(stdout);

    if (scanf("%d", &centimeter) != 1) // from stdin
        fatal("scanf failed on coversion to integer\n");

    inches = centimeter * 0.39;
    printf("\n%d' %d\"\n",inches/12,inches%12); // to stdout

    return 0;
}


/* fatal function body */
void fatal( char * msg)
{
    fprintf(stderr,"scanf failed on conversion to integer\n"); // to stderr
    exit( EXIT_FAILURE);
}
```

Redirections (2)

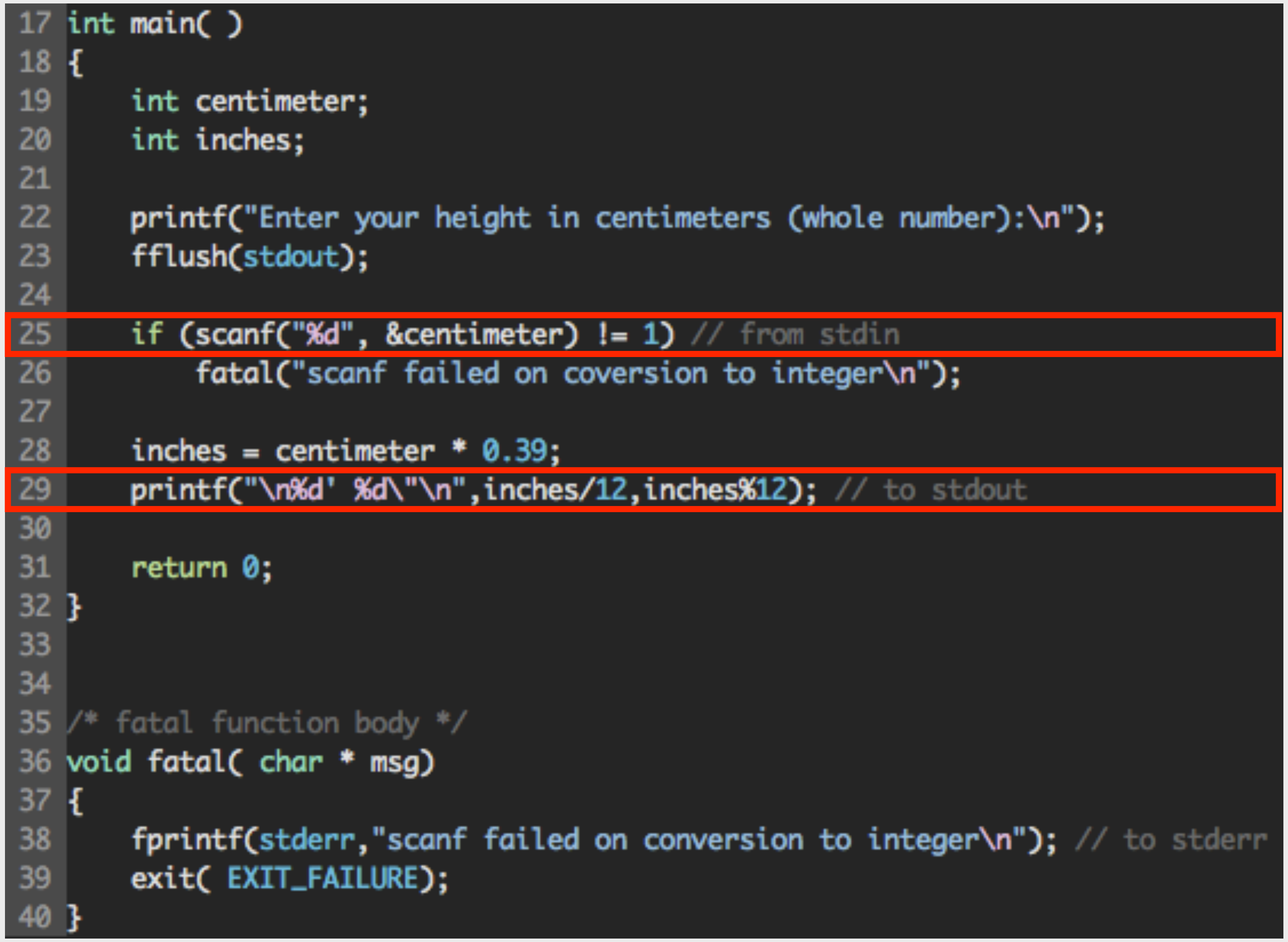
- Example: redir/example.c



```c
int main( )
{
    int centimeter;
    int inches;

    printf("Enter your height in centimeters (whole number):\n");
    fflush(stdout);

    if (scanf("%d", &centimeter) != 1) // from stdin
        fatal("scanf failed on coversion to integer\n");

    inches = centimeter * 0.39;
    printf("\n%d' %d\"\n",inches/12,inches%12); // to stdout

    return 0;
}


/* fatal function body */
void fatal( char * msg)
{
    fprintf(stderr,"scanf failed on conversion to integer\n"); // to stderr
    exit( EXIT_FAILURE);
}
```

Redirections (2)

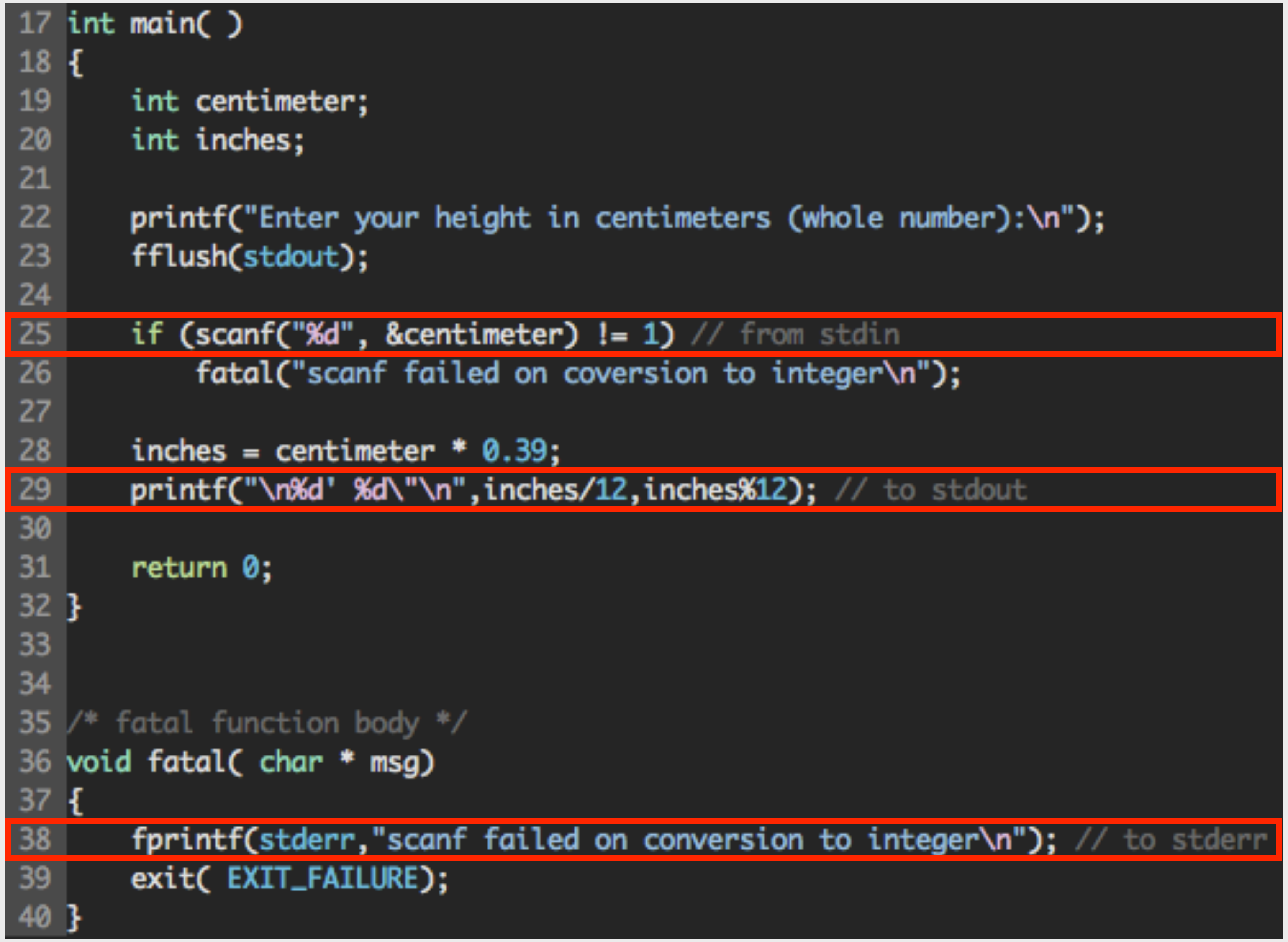
- Example: redir/example.c



```
int main( )
{
    int centimeter;
    int inches;

    printf("Enter your height in centimeters (whole number):\n");
    fflush(stdout);

    if (scanf("%d", &centimeter) != 1) // from stdin
        fatal("scanf failed on coversion to integer\n");

    inches = centimeter * 0.39;
    printf("\n%d' %d\"\n",inches/12,inches%12); // to stdout

    return 0;
}


/* fatal function body */
void fatal( char * msg)
{
    fprintf(stderr,"scanf failed on conversion to integer\n"); // to stderr
    exit( EXIT_FAILURE);
}
```

Redirections (2)

- Example: redir/example.c



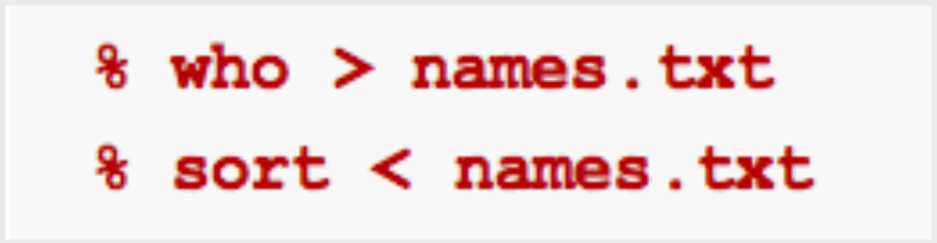
```c
int main( )
{
    int centimeter;
    int inches;

    printf("Enter your height in centimeters (whole number):\n");
    fflush(stdout);

    if (scanf("%d", &centimeter) != 1) // from stdin
        fatal("scanf failed on coversion to integer\n");

    inches = centimeter * 0.39;
    printf("\n%d' %d\"\n",inches/12,inches%12); // to stdout

    return 0;
}


/* fatal function body */
void fatal( char * msg)
{
    fprintf(stderr,"scanf failed on conversion to integer\n"); // to stderr
    exit( EXIT_FAILURE);
}
```

Pipes (1)

- Use the vertical bar “|” to pipe outputs to another command
    - **command1 | command2**
- use pipes to generate complex commands
    - e.g., **who | wc -l**

Pipes (1)

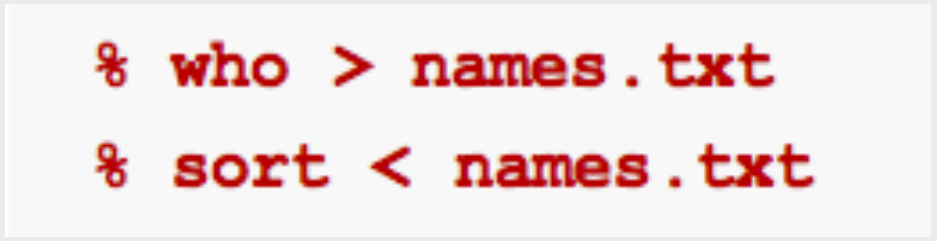
- Use the vertical bar “|” to pipe outputs to another command
    - **command1 | command2**
- use pipes to generate complex commands
    - e.g., **who | wc -l**



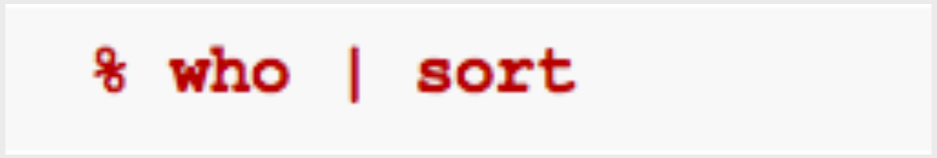
```
% who > names.txt
% sort < names.txt
```

Pipes (1)

- Use the vertical bar "|" to pipe outputs to another command
    - **command1 | command2**
- use pipes to generate complex commands
    - e.g., **who | wc -l**



```
% who > names.txt
% sort < names.txt
```



```
% who | sort
```

Pipes (1)

- Use the vertical bar "|" to pipe outputs to another command
  - **command1 | command2**
- use pipes to generate complex commands
  - e.g., **who | wc -l**

```
% who > names.txt
% sort < names.txt
```

equivalent

```
% who | sort
```

Pipes (2)

Pipes (2)

- Exercises (in pipes/list*.txt)

Pipes (2)

- Exercises (in pipes/list*.txt)
  - **wc -l list1.txt** // how many lines in list1.txt

Pipes (2)

- Exercises (in pipes/list*.txt)
  - **wc -l list1.txt** // how many lines in list1.txt
  - **find . -name ‘*.txt’ | wc -l** // how many txt files in current folder

Pipes (2)

- Exercises (in pipes/list*.txt)
  - **wc -l list1.txt** // how many lines in list1.txt
  - **find . -name ‘*.txt’ | wc -l** // how many txt files in current folder
  - **cat list*.txt | wc -l** // how many lines in all lists

Pipes (2)

- Exercises (in pipes/list*.txt)
  - **wc -l list1.txt** // how many lines in list1.txt
  - **find . -name ‘*.txt’ | wc -l** // how many txt files in current folder
  - **cat list*.txt | wc -l** // how many lines in all lists
  - **cat list*.txt | grep -w ‘the’ | wc -l** // how many words containing ‘the’ in all lists

Pipes (2)

- Exercises (in pipes/list*.txt)
  - **wc -l list1.txt** // how many lines in list1.txt
  - **find . -name '*.txt' | wc -l** // how many txt files in current folder
  - **cat list*.txt | wc -l** // how many lines in all lists
  - **cat list*.txt | grep -w 'the' | wc -l** // how many words containing 'the' in all lists
  - **cat list*.txt | sort | less** // sort all the words in all lists and display in less command

Pipes (2)

- Exercises (in pipes/list*.txt)
  - **wc -l list1.txt** // how many lines in list1.txt
  - **find . -name '*.txt' | wc -l** // how many txt files in current folder
  - **cat list*.txt | wc -l** // how many lines in all lists
  - **cat list*.txt | grep -w 'the' | wc -l** // how many words containing 'the' in all lists
  - **cat list*.txt | sort | less** // sort all the words in all lists and display in less command
  - **cat list*.txt | sort | uniq -c | less** // sort all the words in all lists and display the corresponding frequency in less command

Process Control

Suspend, Resume, Kill (1)

Suspend, Resume, Kill (1)

- control-C (^C)
    - kill process

Suspend, Resume, Kill (1)

- control-C (^C)
  - kill process
- control-Z (^Z)
  - suspend process
  - If you are stuck in vim or some other interactive program that won't let you get out of it, a quick ^Z will cause that program to go into suspended animation and let you back out to the prompt again.

Suspend, Resume, Kill (2)

Suspend, Resume, Kill (2)

- kill processID
    - how to find out the processID of suspend programs

Suspend, Resume, Kill (2)

- kill processID
  - how to find out the processID of suspend programs
- ps or jobs -l
  - ps command shows all your processes
  - jobs command shows the suspend and background processes

Suspend, Resume, Kill (2)

- kill processID
  - how to find out the processID of suspend programs
- ps or jobs -l
  - ps command shows all your processes
  - jobs command shows the suspend and background processes
- fg
  - resume (in foreground) the most recently suspend process

Suspend, Resume, Kill (2)

- kill processID
  - how to find out the processID of suspend programs
- ps or jobs -l
  - ps command shows all your processes
  - jobs command shows the suspend and background processes
- fg
  - resume (in foreground) the most recently suspend process
- bg
  - resume (in background) the most recently suspend process

Suspend, Resume, Kill (3)

Suspend, Resume, Kill (3)

- Practice

```
ls -R / > /dev/null &
jobs -l
fg  // resume in foreground
control-Z  // to suspend again
kill [PID]  // to terminate the process
```

Background process (1)

Background process (1)

- Background process

Background process (1)

- Background process
  - use & to run a program in the background

Background process (1)

- Background process
    - use & to run a program in the background
    - use fg to bring the process to the foreground

Background process (1)

- Background process
  - use & to run a program in the background
  - use fg to bring the process to the foreground
  - ^Z to suspend the process

Background process (2)

Background process (2)

- Practice

```
grep -r “a” * > /dev/null &
jobs -l
fg
^Z
kill [PID]
```

C Revisited

Elements of Program

Elements of Program

- Basic elements of a program

Elements of Program

- Basic elements of a program
  - data declarations (variables)

Elements of Program

- Basic elements of a program
  - data declarations (variables)
  - instructions (functions)

Elements of Program

- Basic elements of a program
  - data declarations (variables)
  - instructions (functions)
  - comments

Basic Program Structure

```
/************************************
**  Comments
*************************************/
... Data declarations ...

int main(){
  ... Executable statements ...
  return (0);
}
```

Variables

Variables

- Invalid variable names

Variables

- Invalid variable names
    - `3rd_entity /* Begins with a number */`

Variables

- Invalid variable names
    - `3rd_entity /* Begins with a number */`
    - `all$done /* Contains a “$” */`

Variables

- Invalid variable names
  - `3rd_entity /* Begins with a number */`
  - `all$done /* Contains a “$” */`
  - `the end /* Contains a space */`

Variables

- Invalid variable names
  - `3rd_entity /* Begins with a number */`
  - `all$done /* Contains a “$” */`
  - `the end /* Contains a space */`
  - `int /* Reserved word */`

Assignment Statements

Assignment Statements

- `answer = (1+2) * 4;`

Assignment Statements

- `answer = (1+2) * 4;`
    - “=” is not the meaning of equal

Assignment Statements

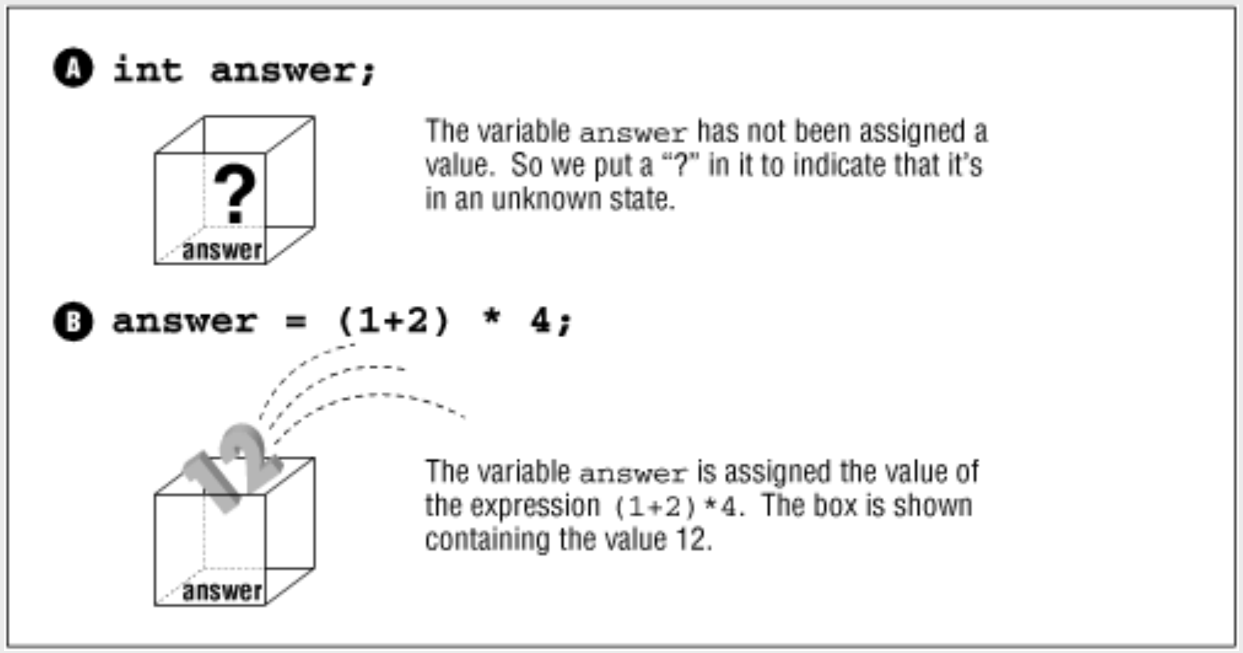
- `answer = (1+2) * 4;`
    - “=” is not the meaning of equal
    - “=” is an assignment operator

Assignment Statements

- `answer = (1+2) * 4;`
  - "=" is not the meaning of equal
  - "=" is an assignment operator
  - The variable "answer" on the left side of the equal sign (=) is assigned the value on the right side.

Assignment Statements

- `answer = (1+2) * 4;`
    - "=" is not the meaning of equal
    - "=" is an assignment operator
    - The variable "answer" on the left side of the equal sign (=) is assigned the value on the right side.



A int answer;
?
answer
The variable answer has not been assigned a value. So we put a "?" in it to indicate that it's in an unknown state.
B answer = (1+2) * 4;
12
answer
The variable answer is assigned the value of the expression (1+2)*4. The box is shown containing the value 12.

`printf` function

`printf` function

- `printf(format, expression-1, expression-2, ...)`

`printf` function

- `printf(format, expression-1, expression-2, ...)`
    - format: the string describing what to print

`printf` function

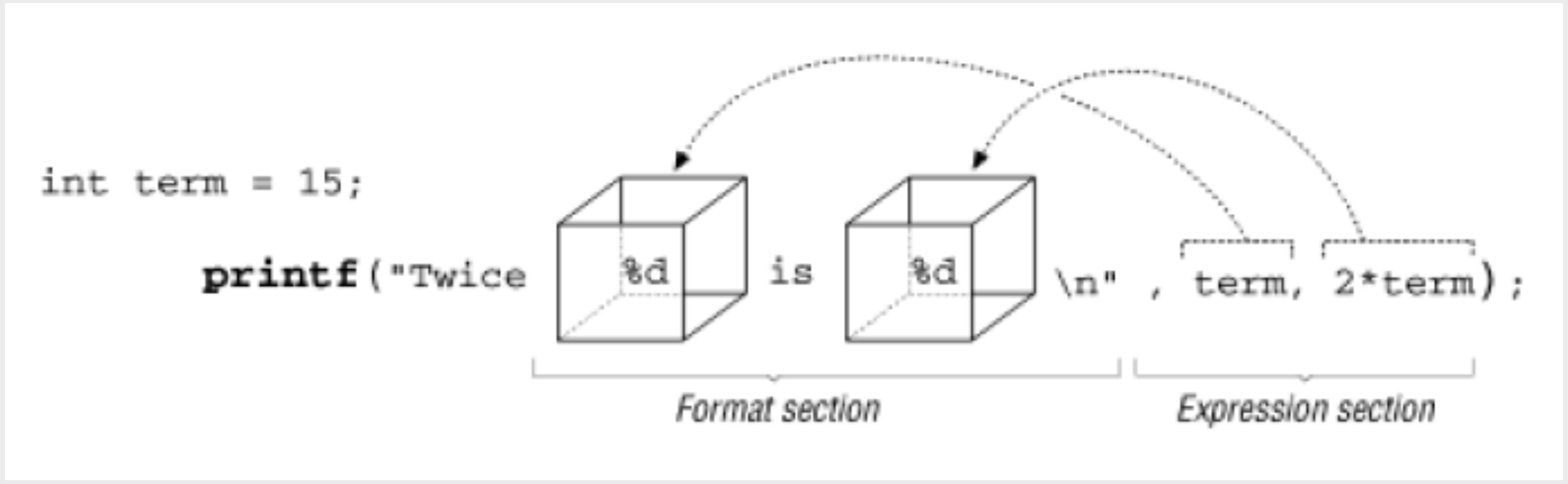
- `printf(format, expression-1, expression-2, ...)`
  - format: the string describing what to print
  - the value of expression-1 is printed in place of the first “%d” in the format string

`printf` function

- `printf(format, expression-1, expression-2, ...)`
    - format: the string describing what to print
    - the value of expression-1 is printed in place of the first "%d" in the format string
    - expression-2 is printed in place of the second, and so on

`printf` function

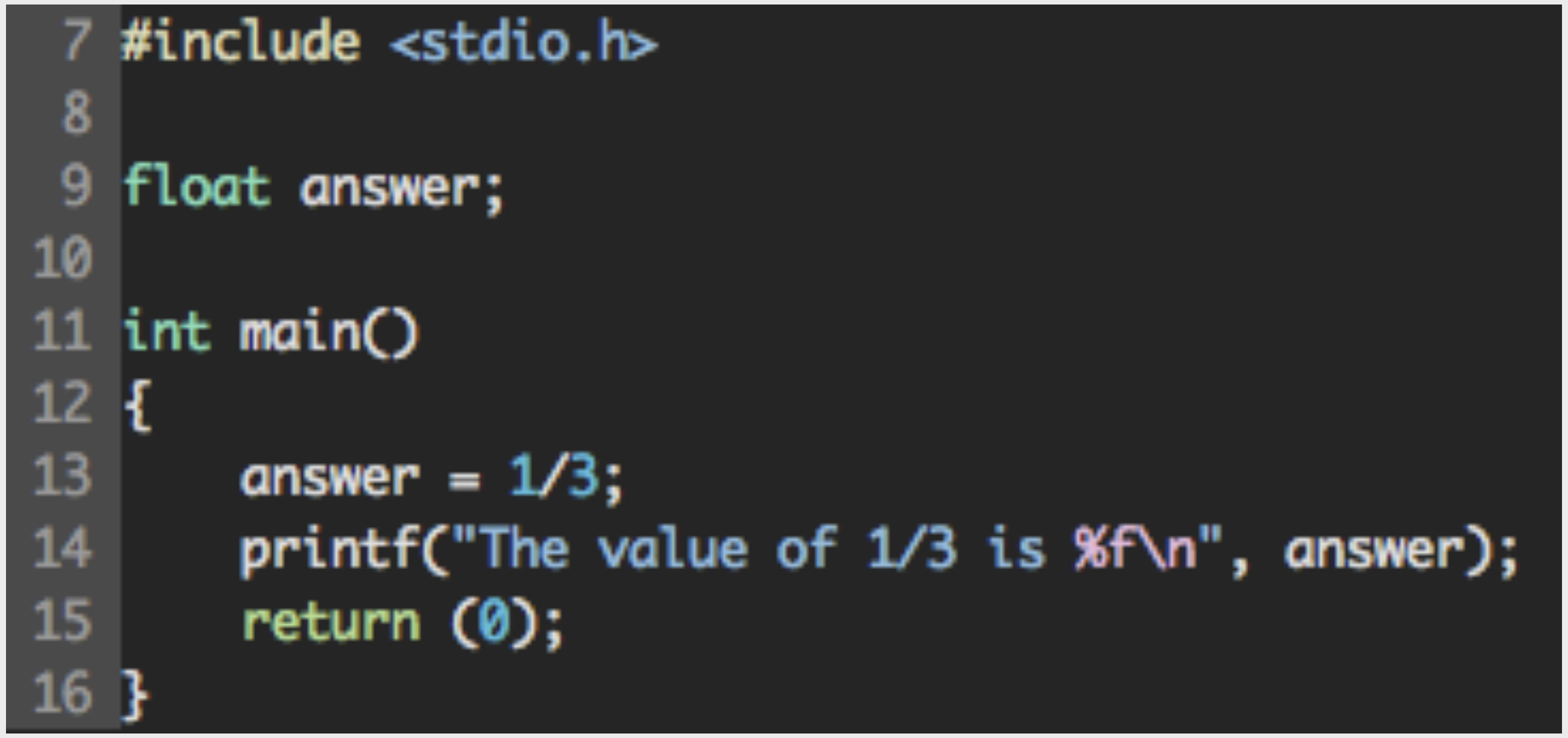
- `printf(format, expression-1, expression-2, ...)`
  - format: the string describing what to print
  - the value of expression-1 is printed in place of the first "%d" in the format string
  - expression-2 is printed in place of the second, and so on



```
int term = 15;
      printf("Twice %d is %d\n" , term, 2*term);
```

Floating Point vs. Integer Divide

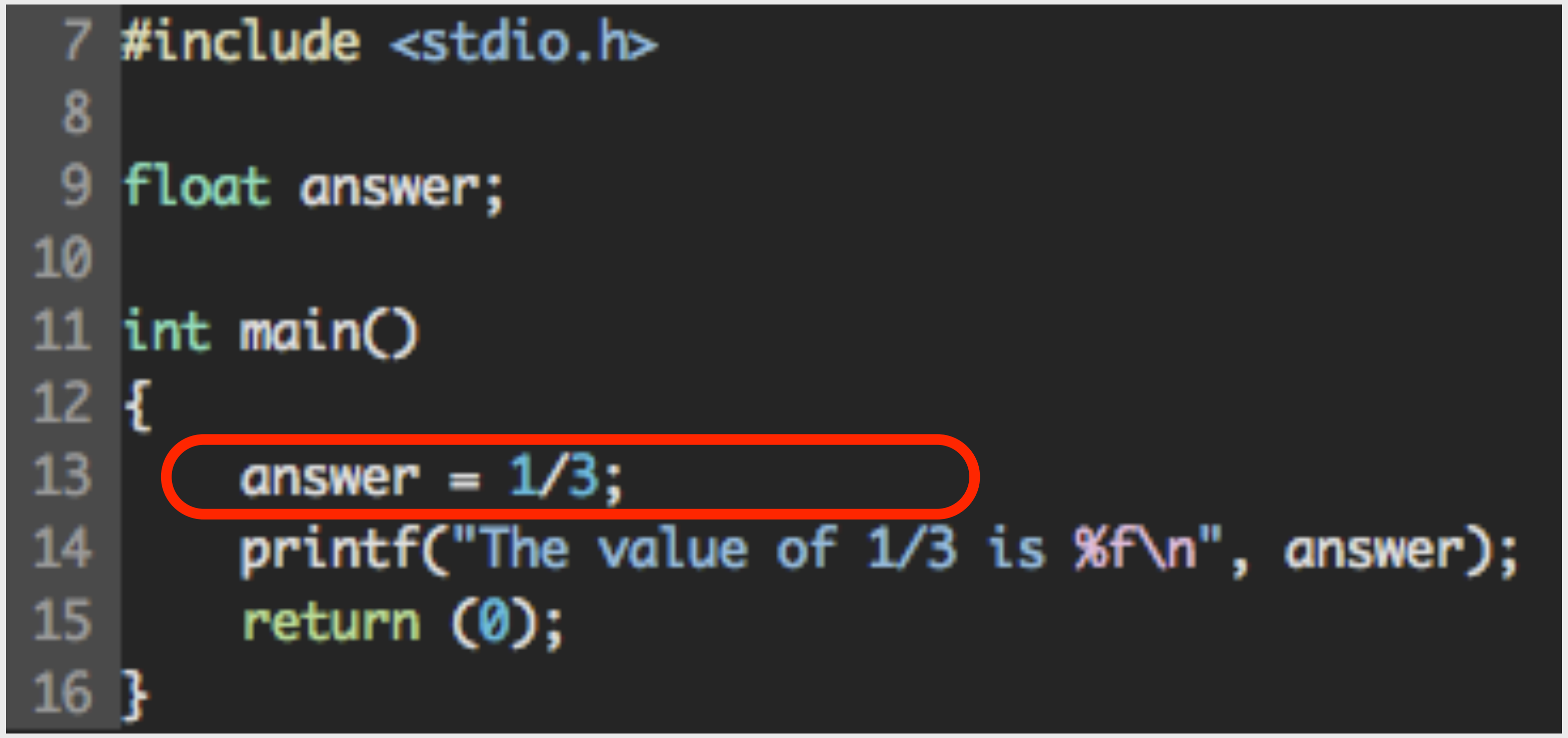
- Why is the result of the code 0.0?



```c
#include <stdio.h>

float answer;

int main()
{
    answer = 1/3;
    printf("The value of 1/3 is %f\n", answer);
    return (0);
}
```

Floating Point vs. Integer Divide

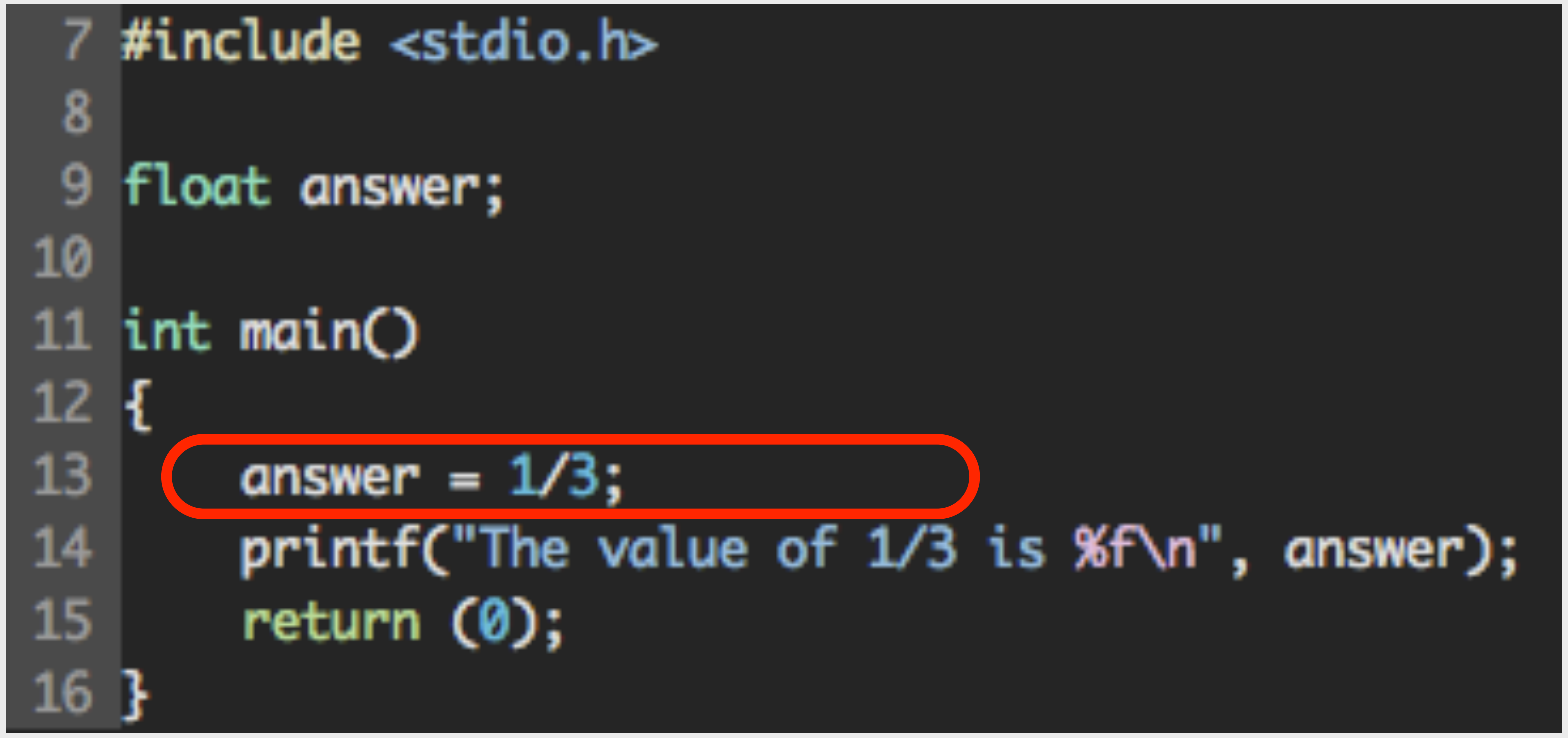
- Why is the result of the code 0.0?



```
#include <stdio.h>

float answer;

int main()
{
    answer = 1/3;
    printf("The value of 1/3 is %f\n", answer);
    return (0);
}
```

Floating Point vs. Integer Divide

- Why is the result of the code 0.0?

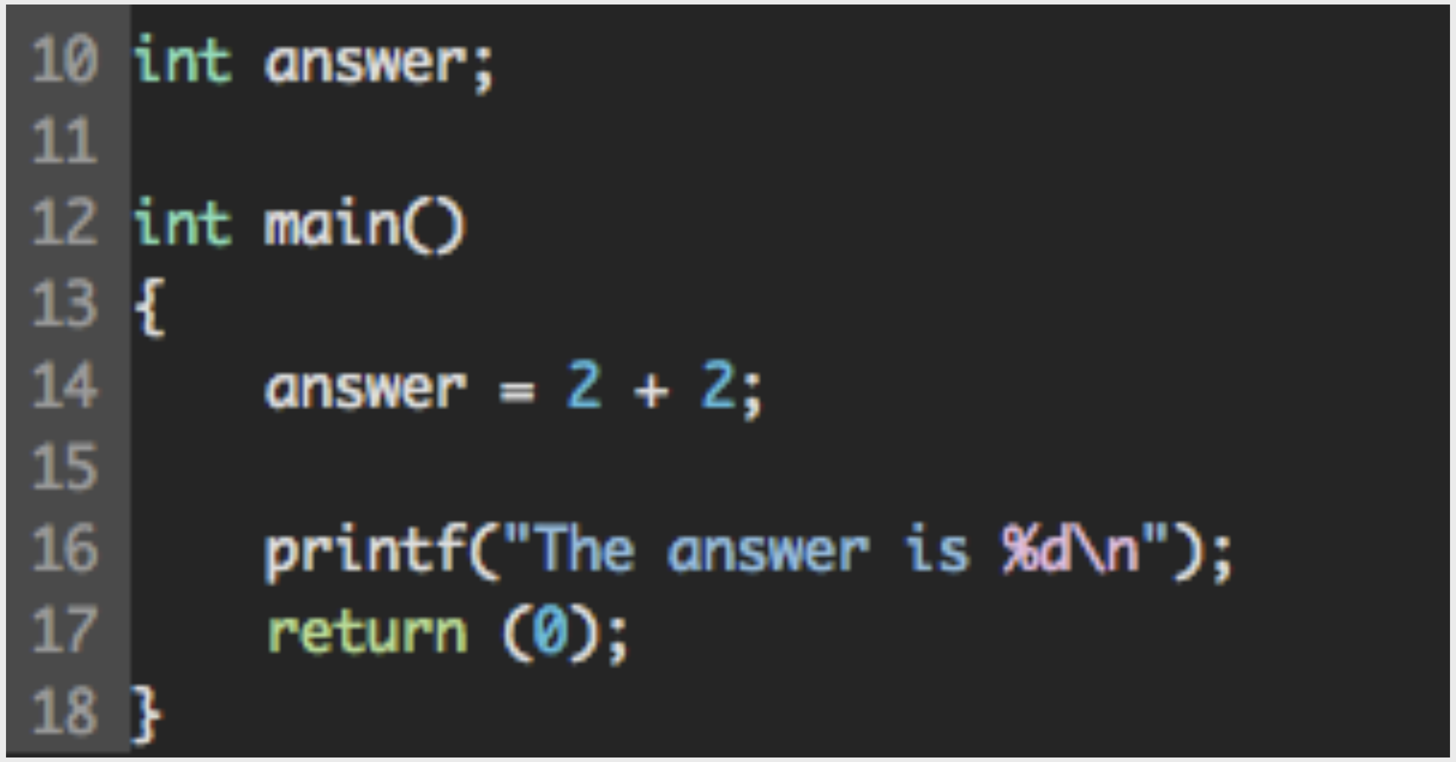


```
#include <stdio.h>

float answer;

int main()
{
    answer = 1/3;
    printf("The value of 1/3 is %f\n", answer);
    return (0);
}
```

```
answer = 1.0 / 3.0;
```

Floating Point vs. Integer Divide

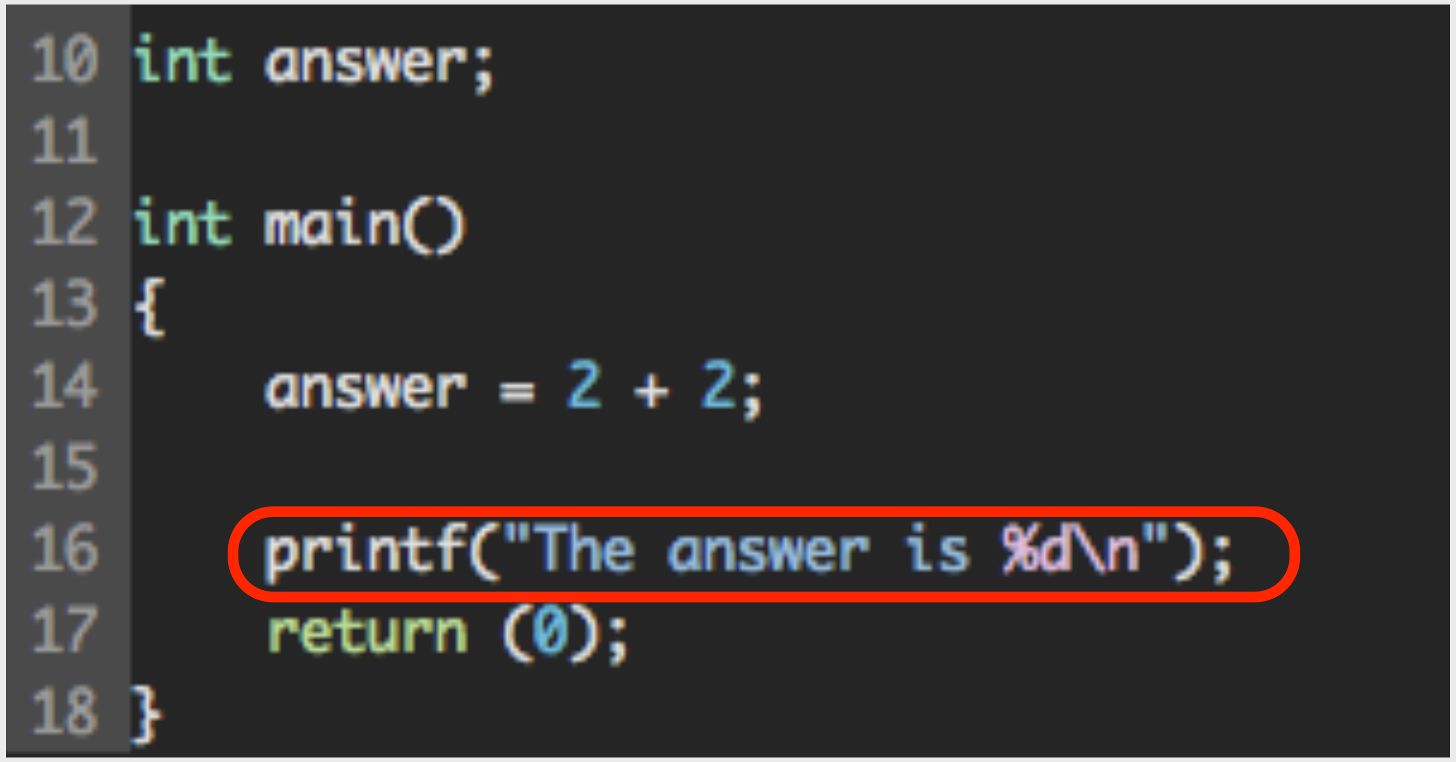
- Why does 2+2 = 5928?



```c
int answer;

int main()
{
    answer = 2 + 2;

    printf("The answer is %d\n");
    return (0);
}
```

Floating Point vs. Integer Divide

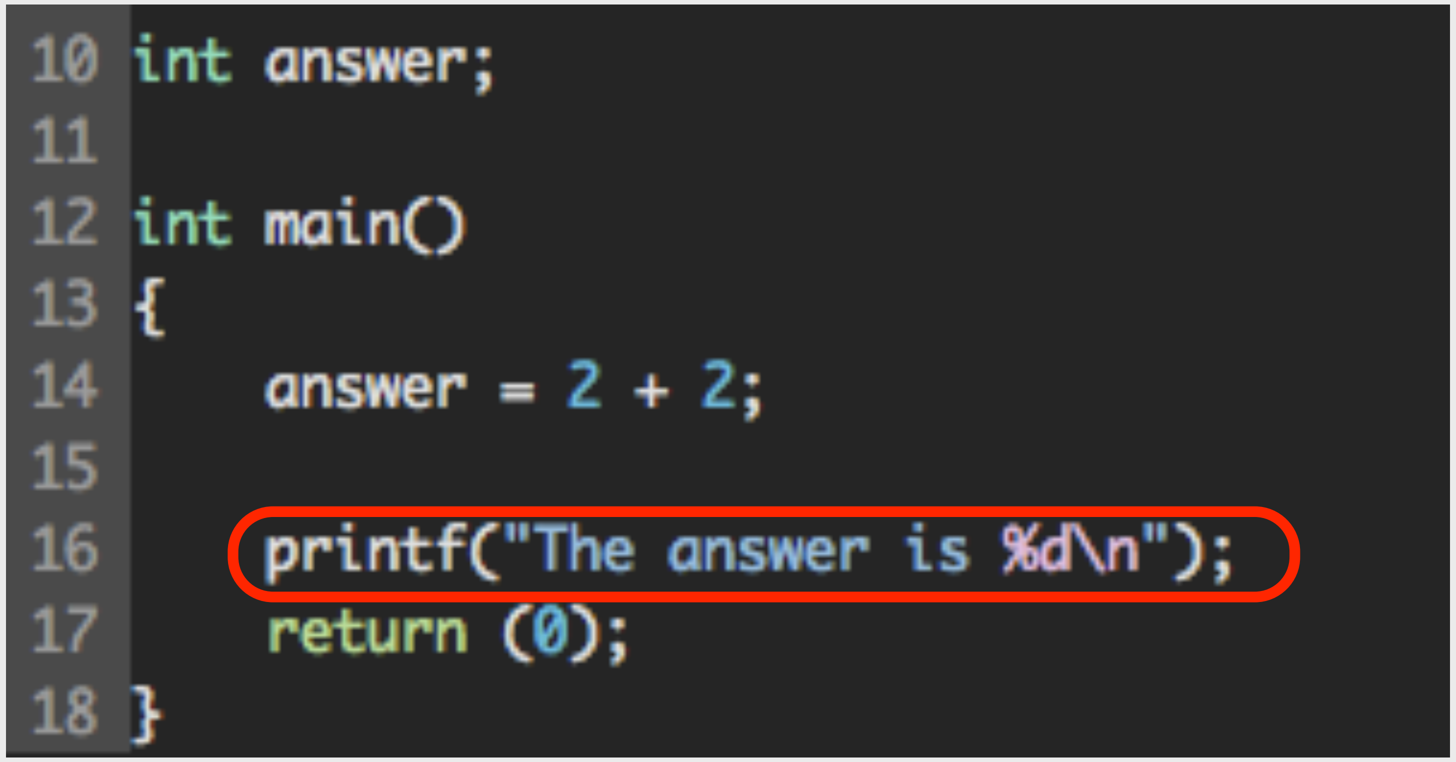
- Why does 2+2 = 5928?



```
int answer;

int main()
{
    answer = 2 + 2;

    printf("The answer is %d\n");
    return (0);
}
```

Floating Point vs. Integer Divide

- Why does 2+2 = 5928?

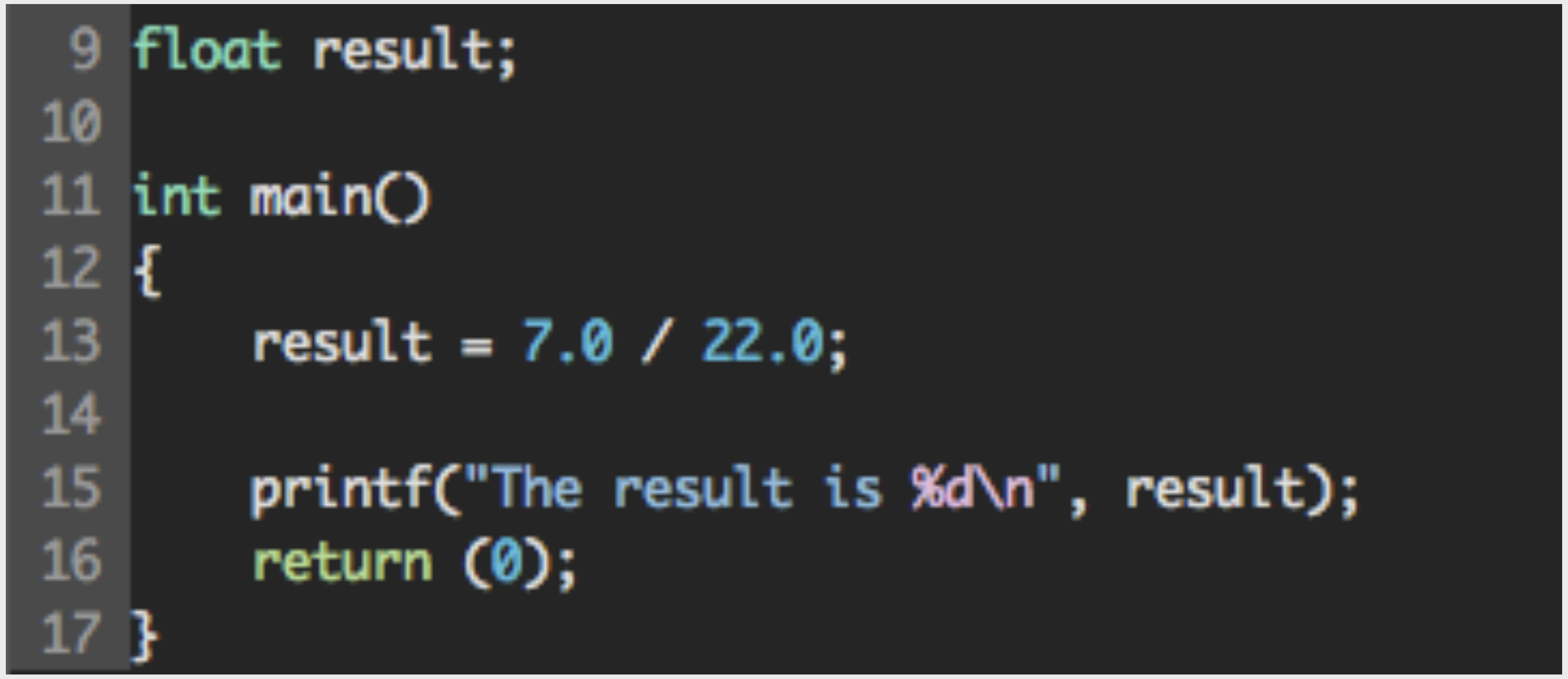


```
int answer;

int main()
{
      answer = 2 + 2;

      printf("The answer is %d\n");
      return (0);
}
```

```
printf("The answer is %d\n", answer);
```

Floating Point vs. Integer Divide

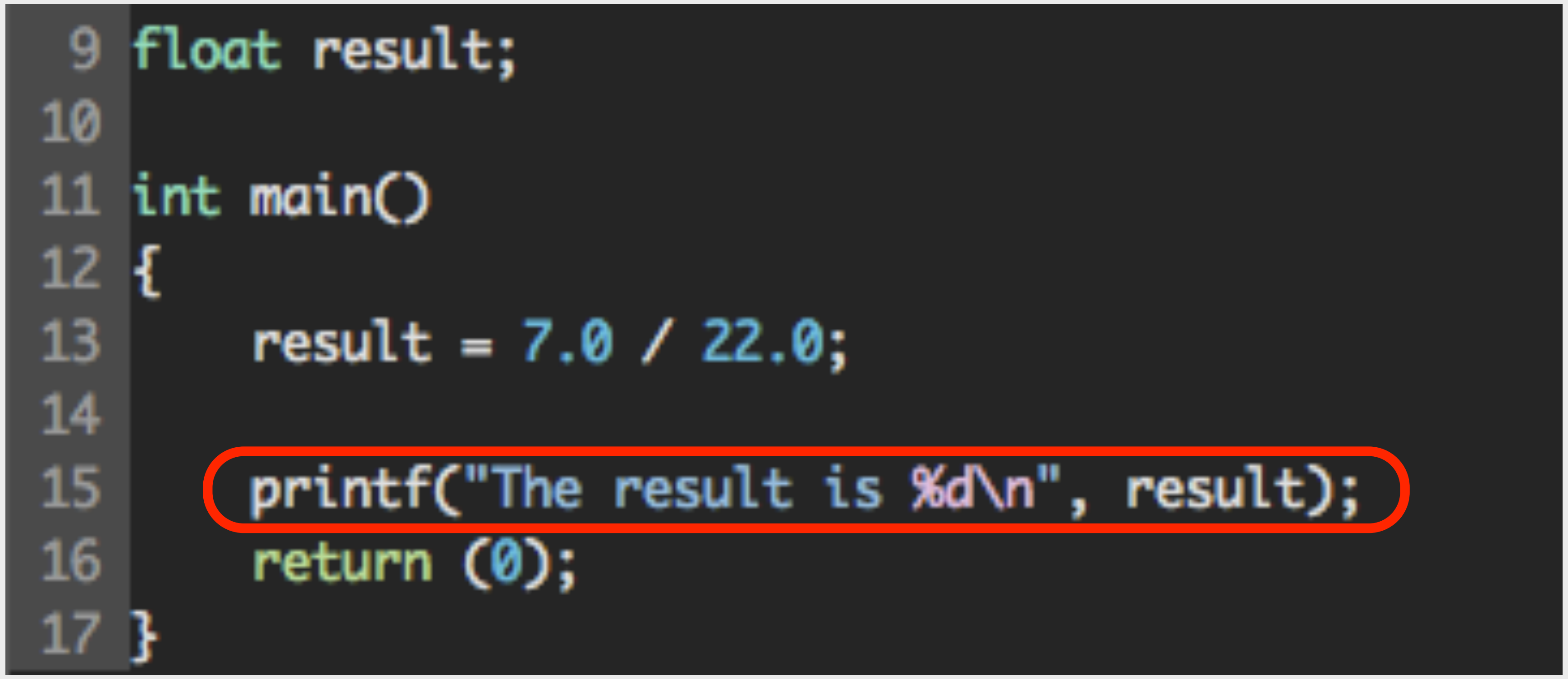
- Why does 7.0/22.0 = 1606412144?



```
float result;

int main()
{
    result = 7.0 / 22.0;

    printf("The result is %d\n", result);
    return (0);
}
```

Floating Point vs. Integer Divide

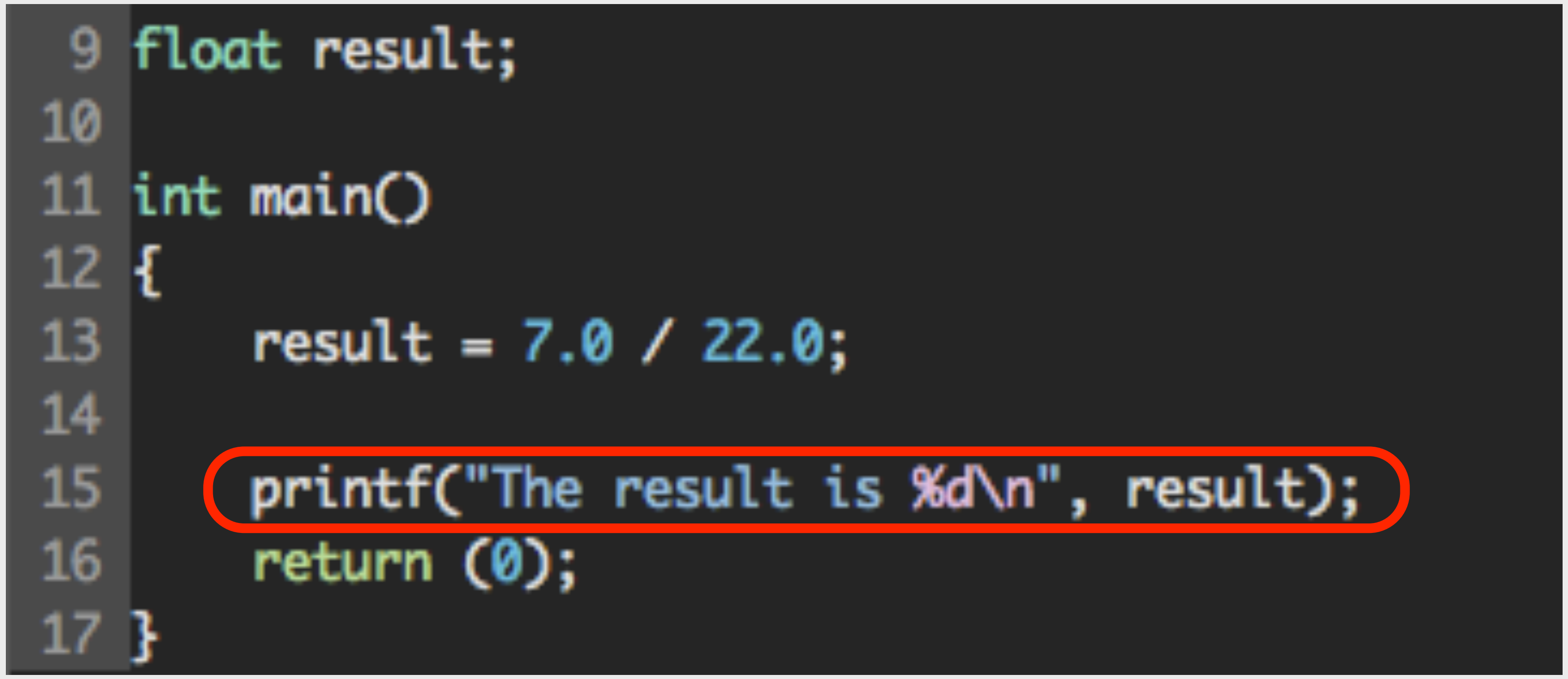
- Why does 7.0/22.0 = 1606412144?



```
float result;

int main()
{
    result = 7.0 / 22.0;

    printf("The result is %d\n", result);
    return (0);
}
```

Floating Point vs. Integer Divide

- Why does 7.0/22.0 = 1606412144?



```
float result;

int main()
{
    result = 7.0 / 22.0;

    printf("The result is %d\n", result);
    return (0);
}
```

```
printf("The result is %f\n", result);
```

Characters

Characters

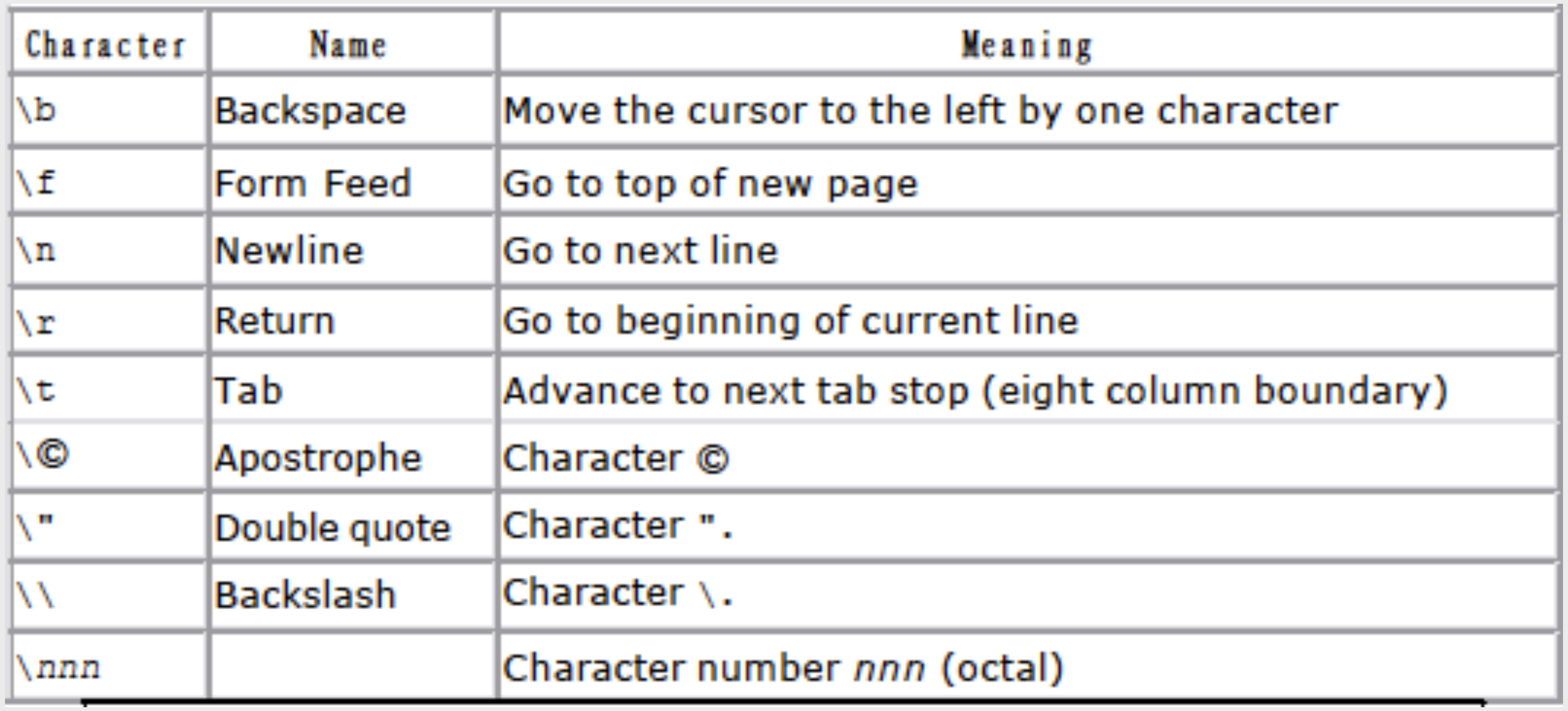
- Declaration

```
char variable /* comment */
```

Characters

- Declaration

```
char variable /* comment */
```



Character	Name	Meaning
\b	Backspace	Move the cursor to the left by one character
\f	Form Feed	Go to top of new page
\n	Newline	Go to next line
\r	Return	Go to beginning of current line
\t	Tab	Advance to next tab stop (eight column boundary)
\©	Apostrophe	Character ©
\"	Double quote	Character ".
\\	Backslash	Character \.
\nnn		Character number *nnn* (octal)

Arrays

Arrays

- `int data_list[10];`

Arrays

- `int data_list[10];`
  - An array is a set of consecutive memory locations used to store data

Arrays

- `int data_list[10];`
  - An array is a set of consecutive memory locations used to store data
  - Each item in the array is called an element

Strings

Strings

- Strings are sequences of characters

Strings

- Strings are sequences of characters
- In C, strings are carried out by character arrays

Strings

- Strings are sequences of characters
- In C, strings are carried out by character arrays
- `'\0'` is used to indicate the end of a string

Strings

- Strings are sequences of characters
- In C, strings are carried out by character arrays
- **`‘\0’`** is used to indicate the end of a string

```
char name[4];

name[0] = ‘S’;
name[1] = ‘a’;
name[2] = ‘m’;
name[3] = ‘\0’;
```

Copy a String

Copy a String

- `name = “Sam”;`   // illegal

Copy a String

- `name = “Sam”;`   // illegal
  - C does not allow one array to be assigned to another

Copy a String

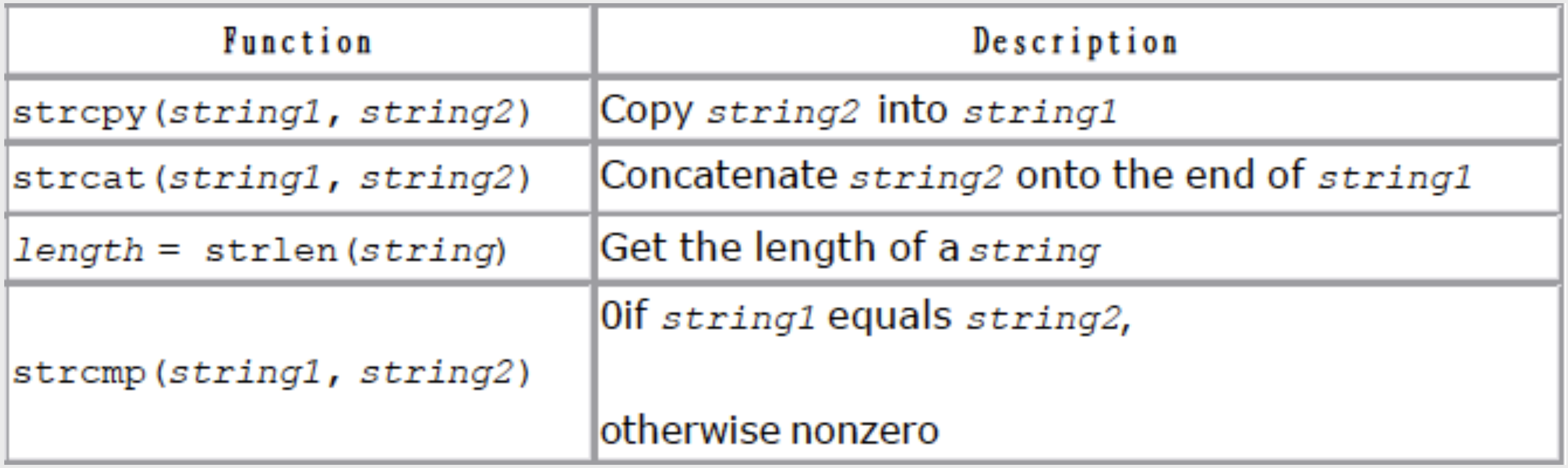
- `name = “Sam”;`   // illegal
  - C does not allow one array to be assigned to another
- Use **`strcpy()`** to copy a string

Copy a String

- `name = “Sam”;` // illegal
  - C does not allow one array to be assigned to another
- Use **`strcpy()`** to copy a string

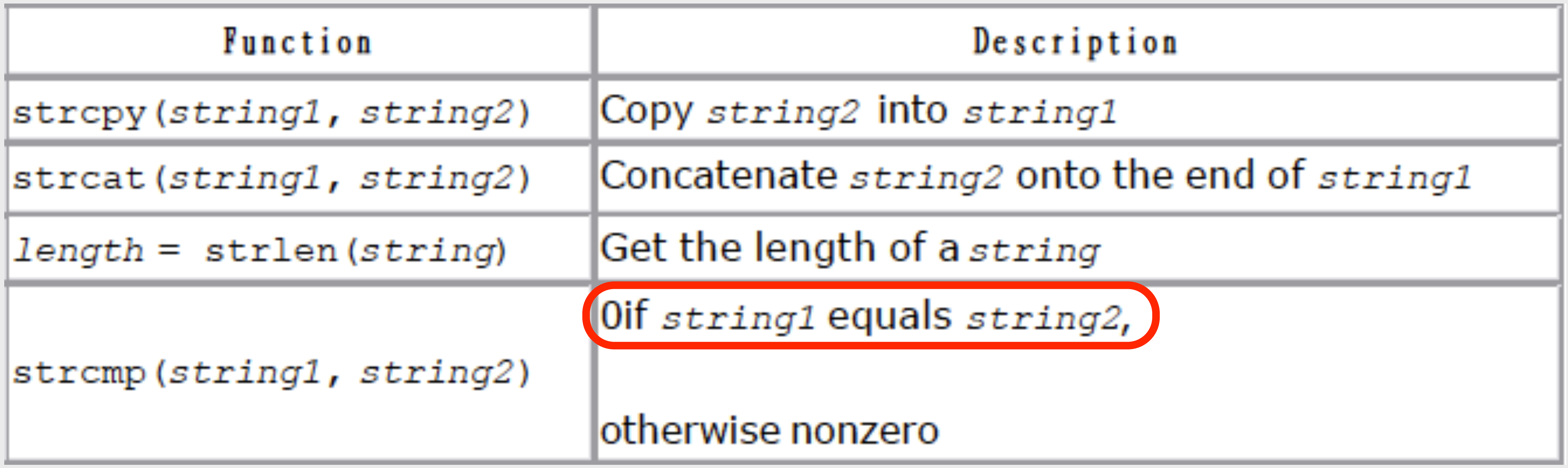
```
char name[4];

strcpy(name, “Sam”);
```

Common String Functions



Function	Description
strcpy(*string1*, *string2*)	Copy *string2* into *string1*
strcat(*string1*, *string2*)	Concatenate *string2* onto the end of *string1*
length = strlen(*string*)	Get the length of a *string*
strcmp(*string1*, *string2*)	0if *string1* equals *string2*, otherwise nonzero

Common String Functions



Function	Description
`strcpy(`*string1*`,` *string2*`)`	Copy *string2* into *string1*
`strcat(`*string1*`,` *string2*`)`	Concatenate *string2* onto the end of *string1*
length `=` `strlen(`*string*`)`	Get the length of a *string*
`strcmp(`*string1*`,` *string2*`)`	0if *string1* equals *string2*, otherwise nonzero

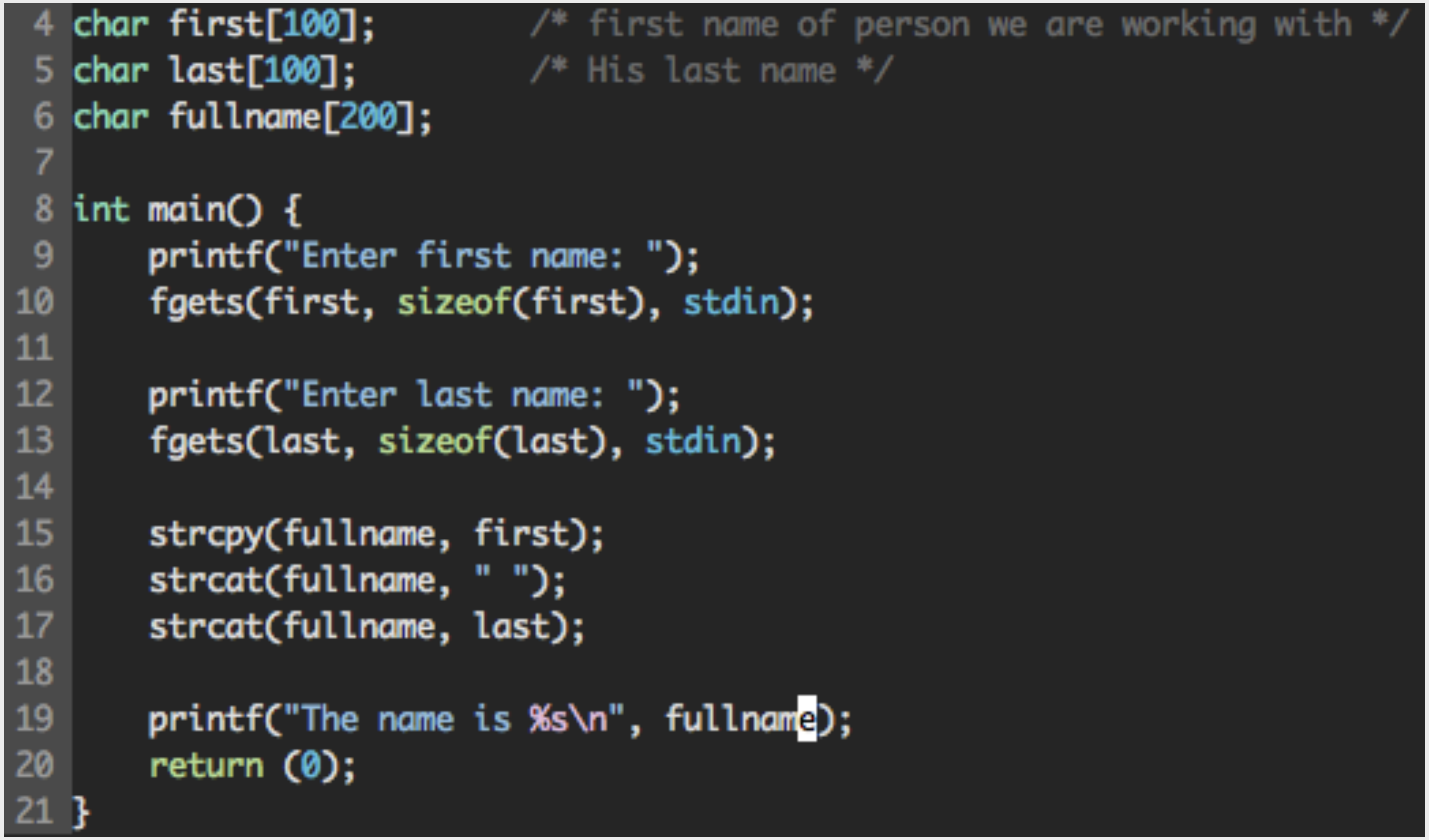
Reading Strings

- The standard functions **`fgets`** can be used to read a string from the keyboard

```
fgets(name, sizeof(name), stdin);
```

Reading Strings

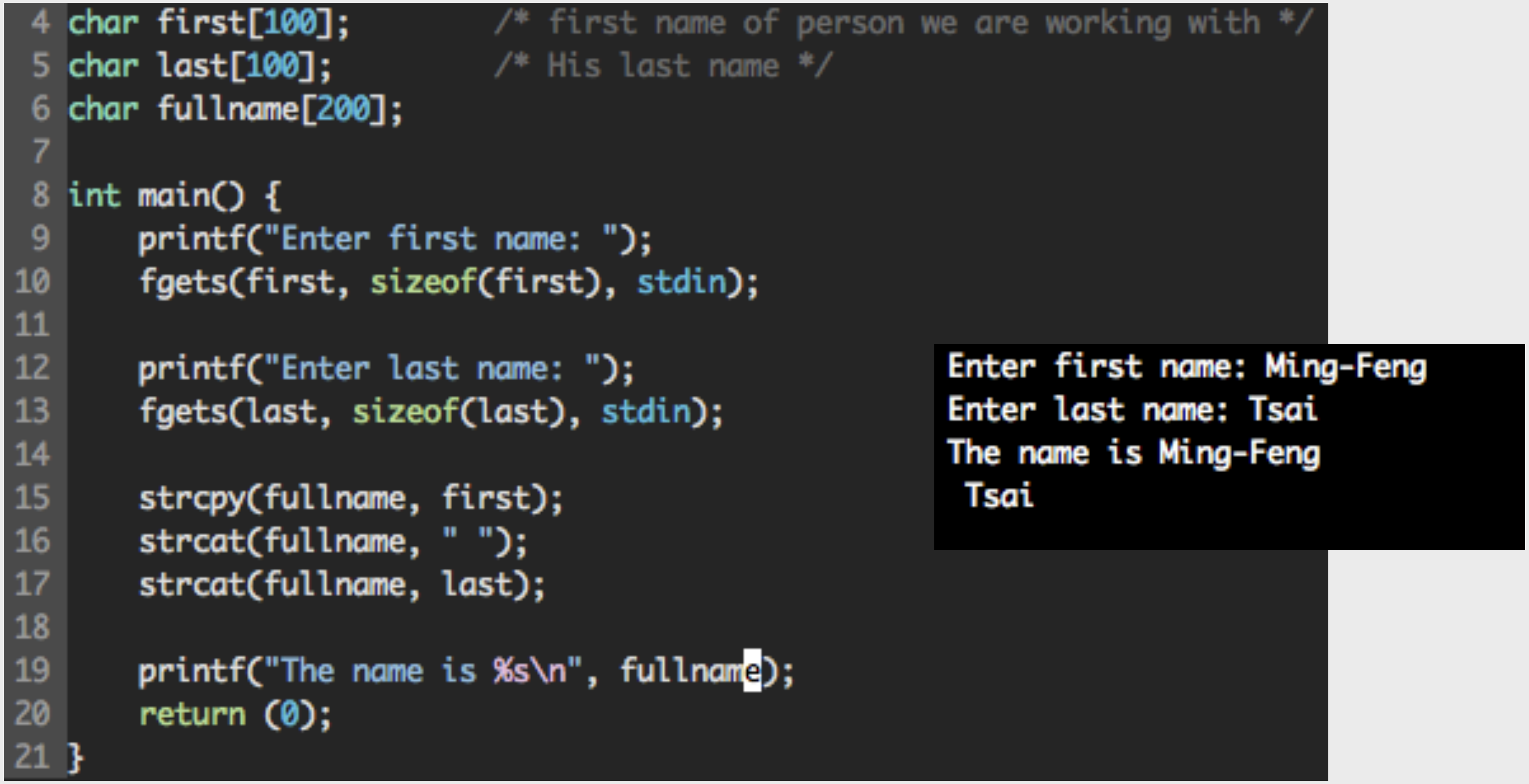
- Example: fullname.c



```
char first[100];        /* first name of person we are working with */
char last[100];         /* His last name */
char fullname[200];

int main() {
    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);

    strcpy(fullname, first);
    strcat(fullname, " ");
    strcat(fullname, last);

    printf("The name is %s\n", fullname);
    return (0);
}
```

Reading Strings

- Example: fullname.c

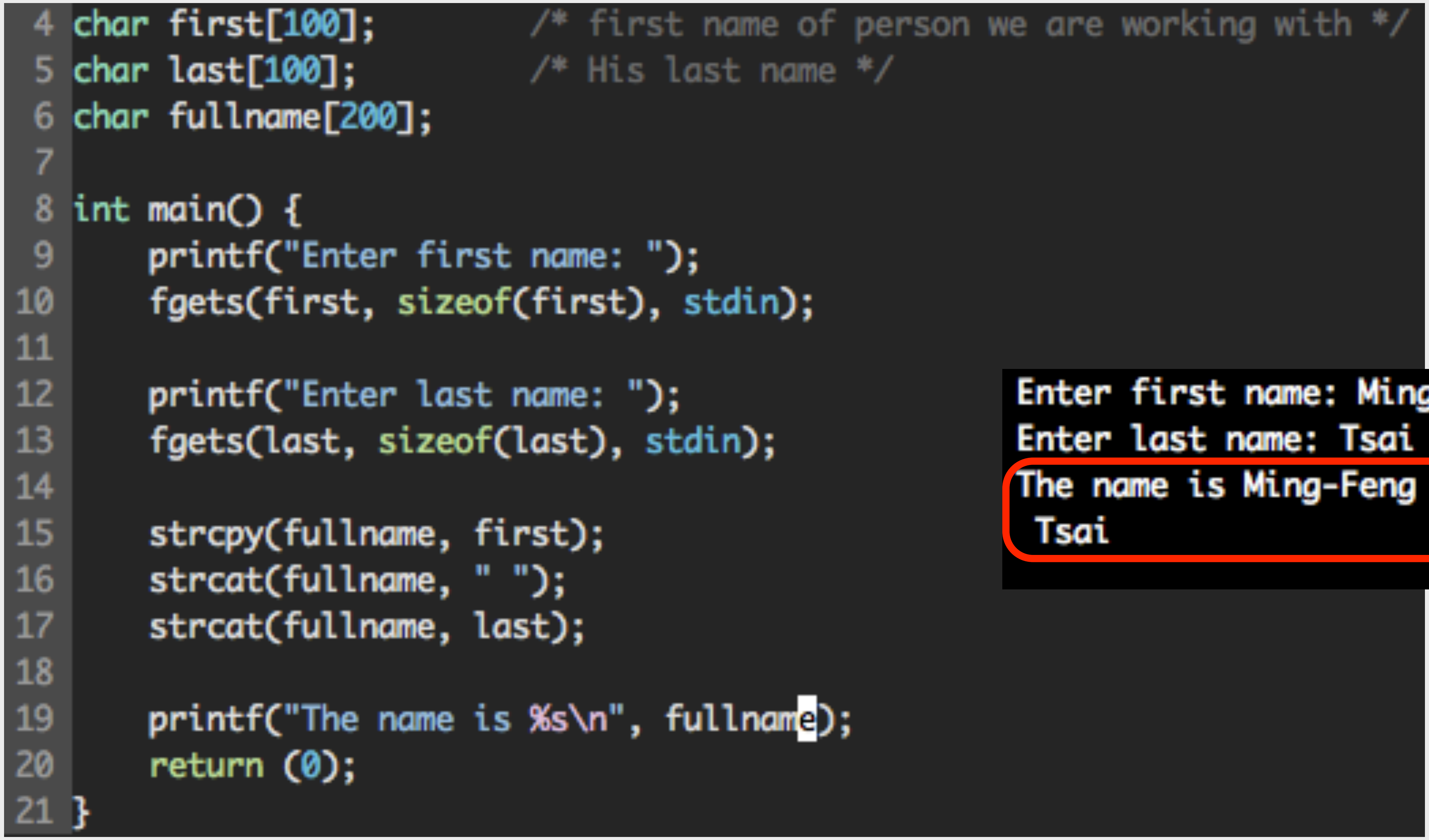


```c
char first[100];        /* first name of person we are working with */
char last[100];         /* His last name */
char fullname[200];

int main() {
    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);

    strcpy(fullname, first);
    strcat(fullname, " ");
    strcat(fullname, last);

    printf("The name is %s\n", fullname);
    return (0);
}
```

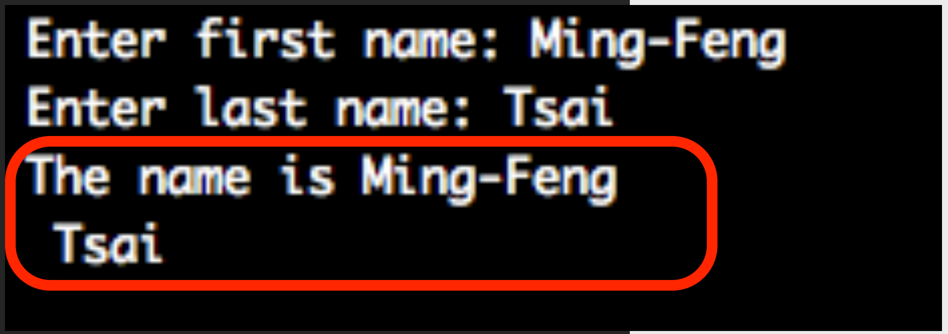
```
Enter first name: Ming-Feng
Enter last name: Tsai
The name is Ming-Feng
 Tsai
```

Reading Strings

- Example: fullname.c



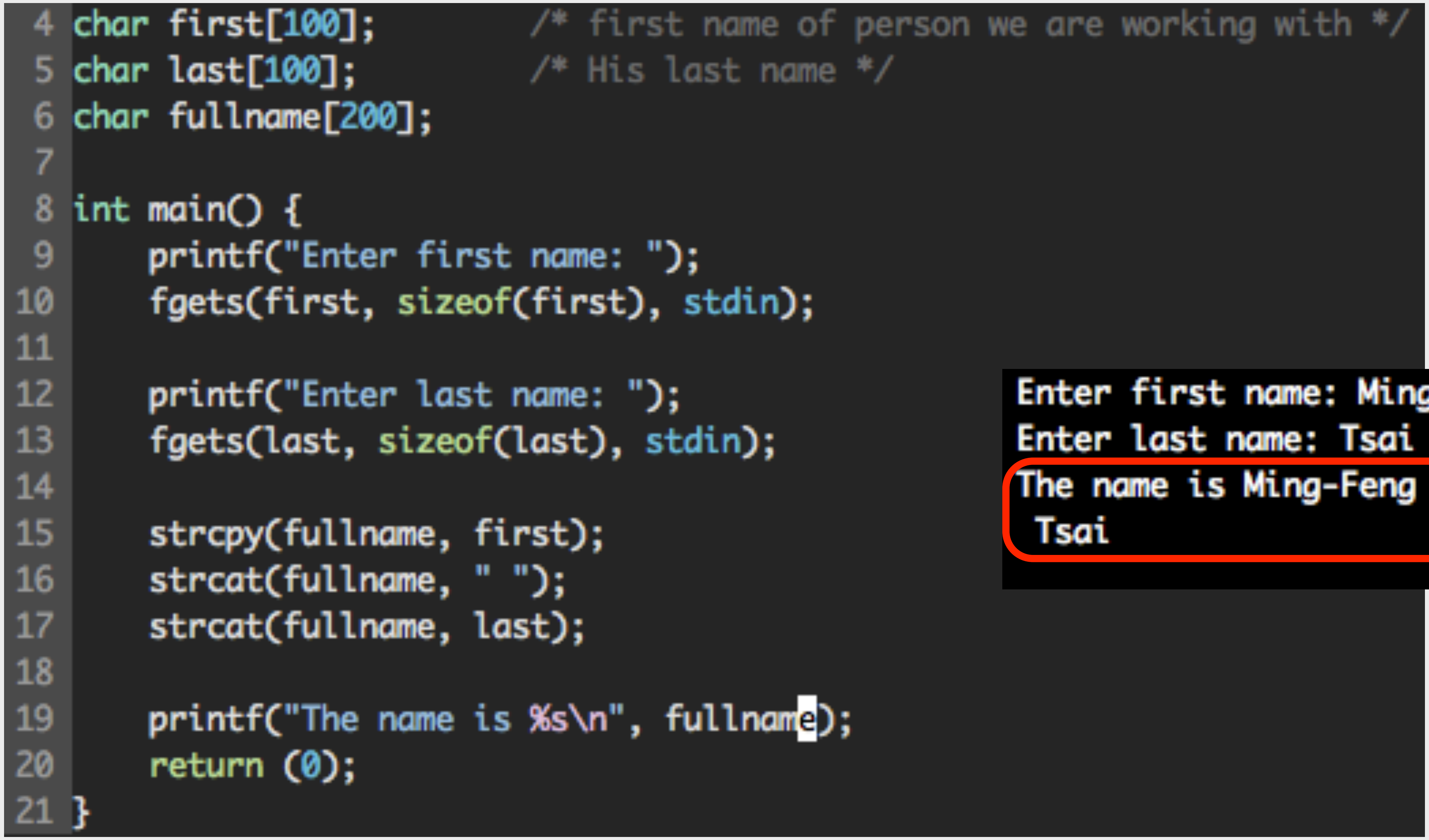
```c
char first[100];        /* first name of person we are working with */
char last[100];         /* His last name */
char fullname[200];

int main() {
    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);

    strcpy(fullname, first);
    strcat(fullname, " ");
    strcat(fullname, last);

    printf("The name is %s\n", fullname);
    return (0);
}
```



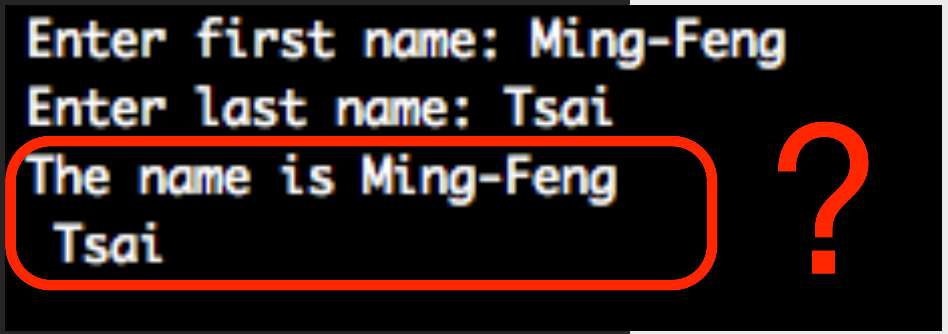
```
Enter first name: Ming-Feng
Enter last name: Tsai
The name is Ming-Feng
 Tsai
```

Reading Strings

- Example: fullname.c



```c
char first[100];        /* first name of person we are working with */
char last[100];         /* His last name */
char fullname[200];

int main() {
    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);

    strcpy(fullname, first);
    strcat(fullname, " ");
    strcat(fullname, last);

    printf("The name is %s\n", fullname);
    return (0);
}
```



```
Enter first name: Ming-Feng
Enter last name: Tsai
The name is Ming-Feng
 Tsai
```

?

Reading Strings

Reading Strings

- The explanation

Reading Strings

- The explanation
  - The **`fgets()`** function gets the entire line, including the end-of-line. We have to get rid of the character before printing.

```
first[ strlen(first) - 1 ] = '\0';
last[ strlen(last) - 1 ] = '\0';
```

Multidimensional Arrays

Multidimensional Arrays

- `type variable[size1][size2]`

Multidimensional Arrays

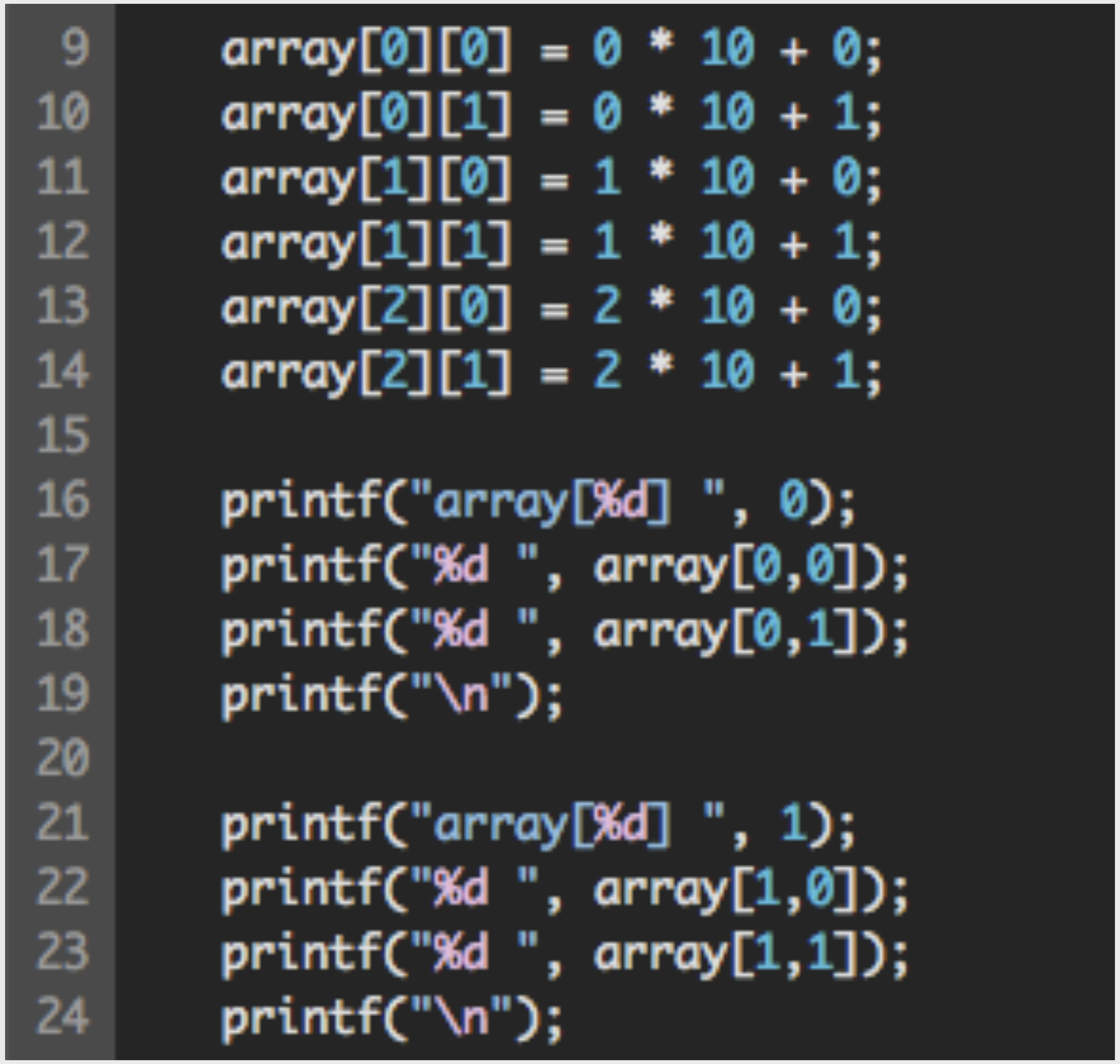
- `type variable[size1][size2]`
  - `int matrix[2][4];`
    `/* declare a 2*4 int array */`

Multidimensional Arrays

- `type variable[size1][size2]`
    - `int matrix[2][4];`
      `/* declare a 2*4 int array */`
    - `matrix[1][2] = 10; /* assign 10 */`

Multidimensional Arrays

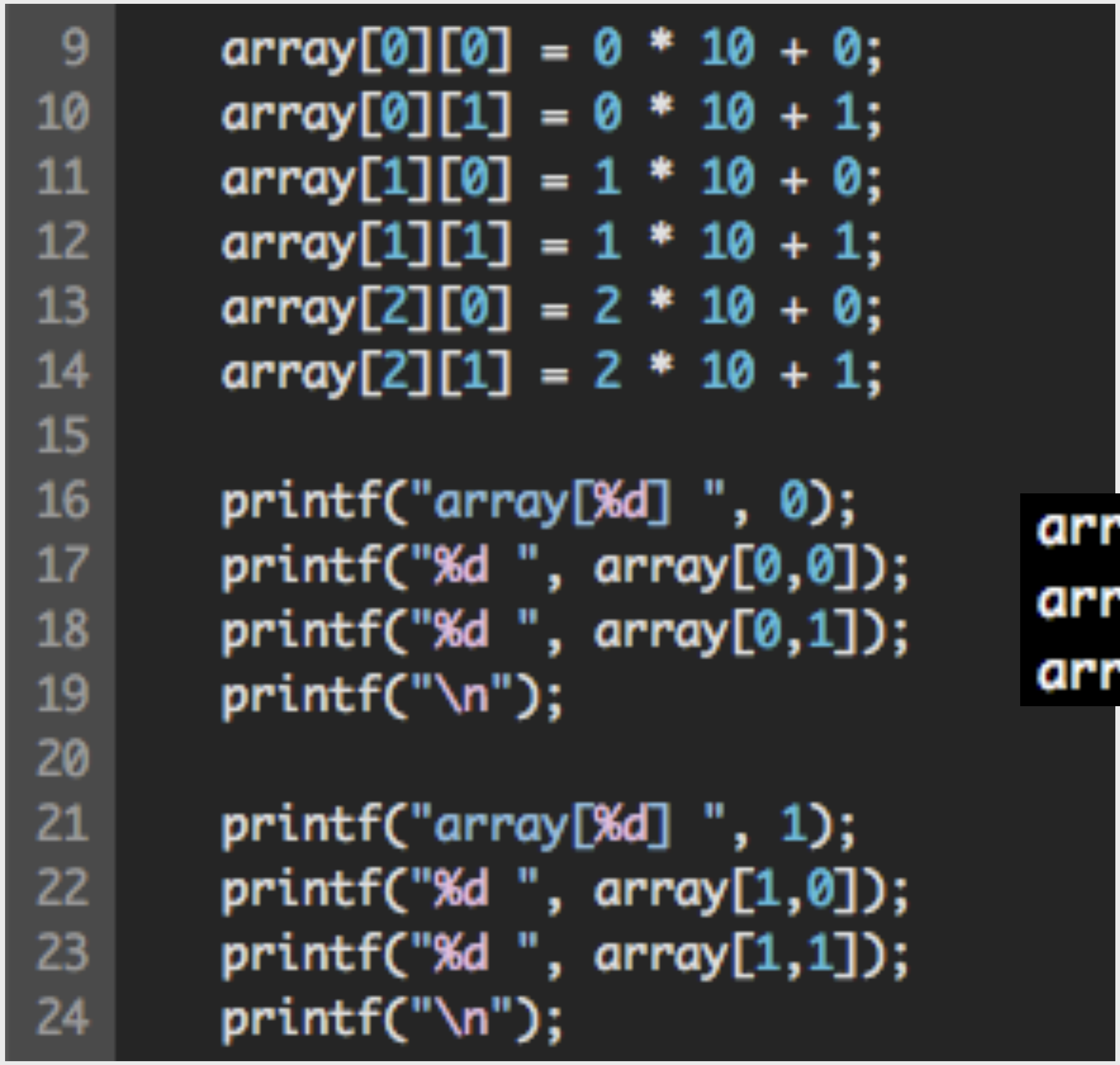
- Example: multiarray.c



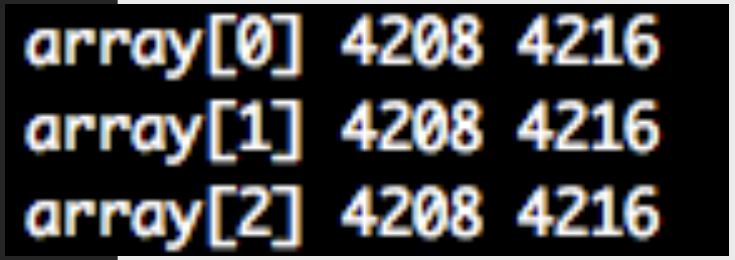
```
    array[0][0] = 0 * 10 + 0;
    array[0][1] = 0 * 10 + 1;
    array[1][0] = 1 * 10 + 0;
    array[1][1] = 1 * 10 + 1;
    array[2][0] = 2 * 10 + 0;
    array[2][1] = 2 * 10 + 1;

    printf("array[%d] ", 0);
    printf("%d ", array[0,0]);
    printf("%d ", array[0,1]);
    printf("\n");

    printf("array[%d] ", 1);
    printf("%d ", array[1,0]);
    printf("%d ", array[1,1]);
    printf("\n");
```

Multidimensional Arrays

- Example: multiarray.c



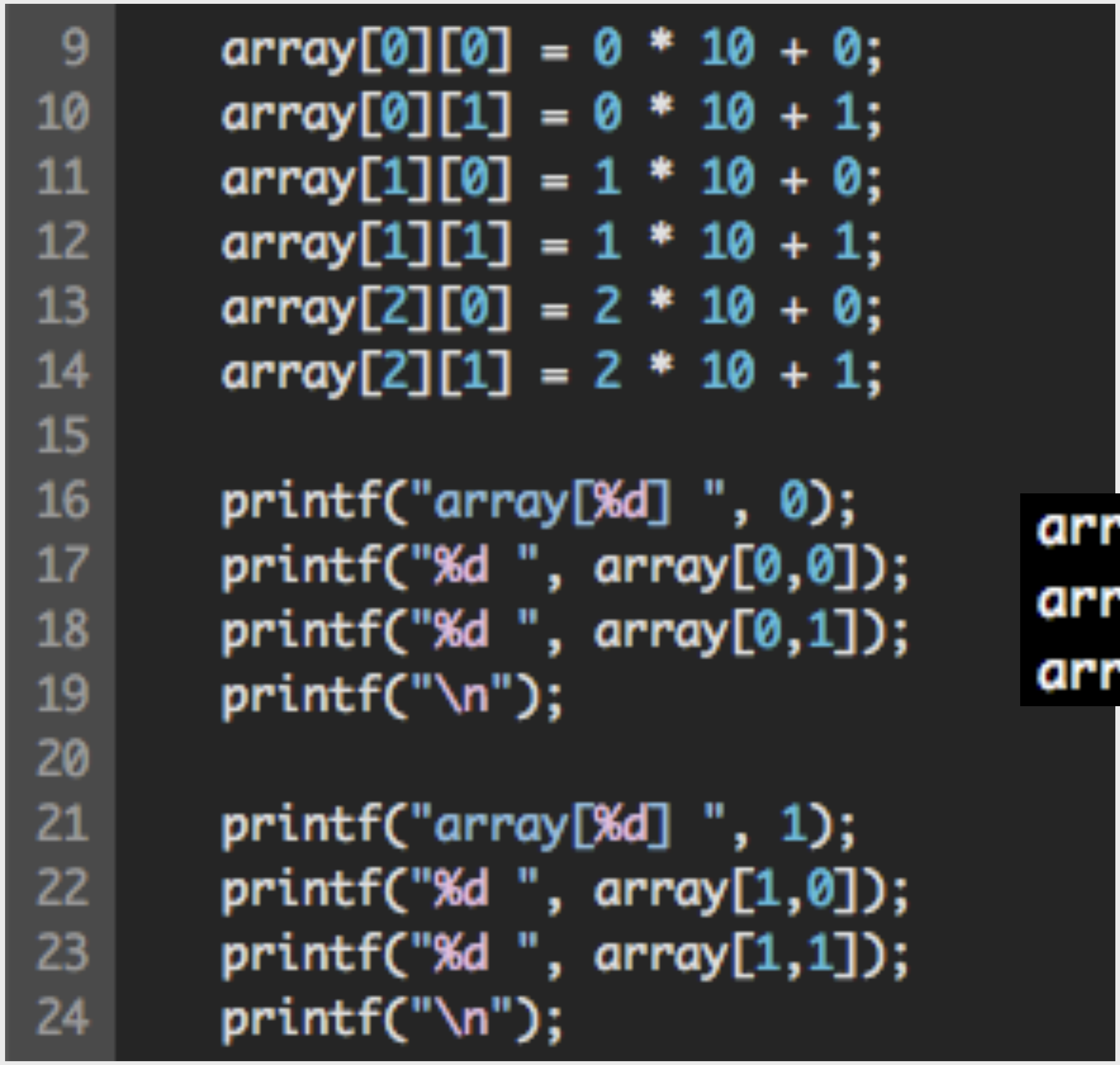
```
    array[0][0] = 0 * 10 + 0;
    array[0][1] = 0 * 10 + 1;
    array[1][0] = 1 * 10 + 0;
    array[1][1] = 1 * 10 + 1;
    array[2][0] = 2 * 10 + 0;
    array[2][1] = 2 * 10 + 1;

    printf("array[%d] ", 0);
    printf("%d ", array[0,0]);
    printf("%d ", array[0,1]);
    printf("\n");

    printf("array[%d] ", 1);
    printf("%d ", array[1,0]);
    printf("%d ", array[1,1]);
    printf("\n");
```



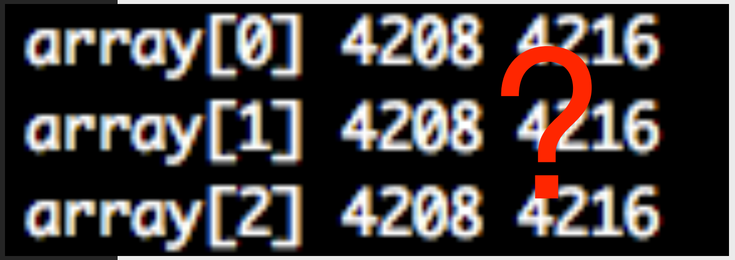
```
array[0] 4208 4216
array[1] 4208 4216
array[2] 4208 4216
```

Multidimensional Arrays

- Example: multiarray.c



```
    array[0][0] = 0 * 10 + 0;
    array[0][1] = 0 * 10 + 1;
    array[1][0] = 1 * 10 + 0;
    array[1][1] = 1 * 10 + 1;
    array[2][0] = 2 * 10 + 0;
    array[2][1] = 2 * 10 + 1;

    printf("array[%d] ", 0);
    printf("%d ", array[0,0]);
    printf("%d ", array[0,1]);
    printf("\n");

    printf("array[%d] ", 1);
    printf("%d ", array[1,0]);
    printf("%d ", array[1,1]);
    printf("\n");
```

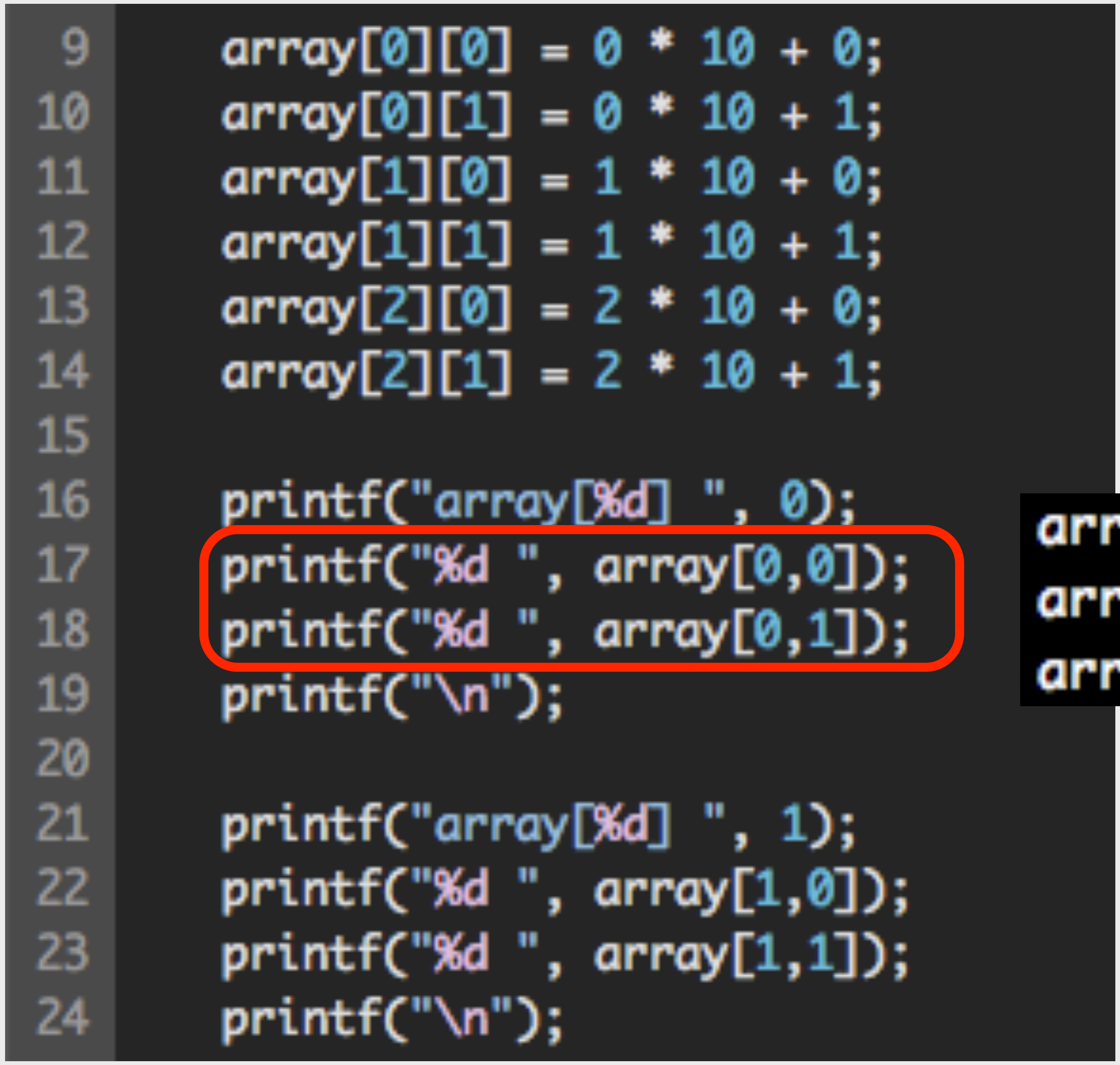


```
array[0] 4208 4216
array[1] 4208 4216
array[2] 4208 4216
```

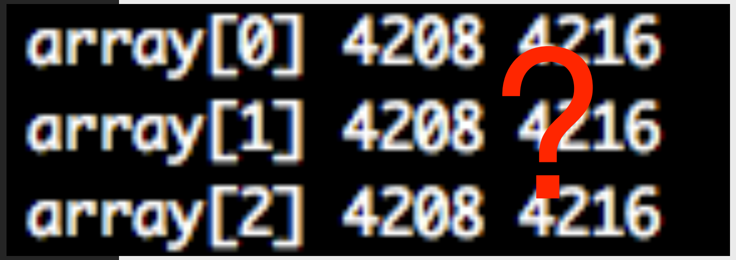
?

Multidimensional Arrays

- Example: multiarray.c



```
    array[0][0] = 0 * 10 + 0;
    array[0][1] = 0 * 10 + 1;
    array[1][0] = 1 * 10 + 0;
    array[1][1] = 1 * 10 + 1;
    array[2][0] = 2 * 10 + 0;
    array[2][1] = 2 * 10 + 1;

    printf("array[%d] ", 0);
    printf("%d ", array[0,0]);
    printf("%d ", array[0,1]);
    printf("\n");

    printf("array[%d] ", 1);
    printf("%d ", array[1,0]);
    printf("%d ", array[1,1]);
    printf("\n");
```

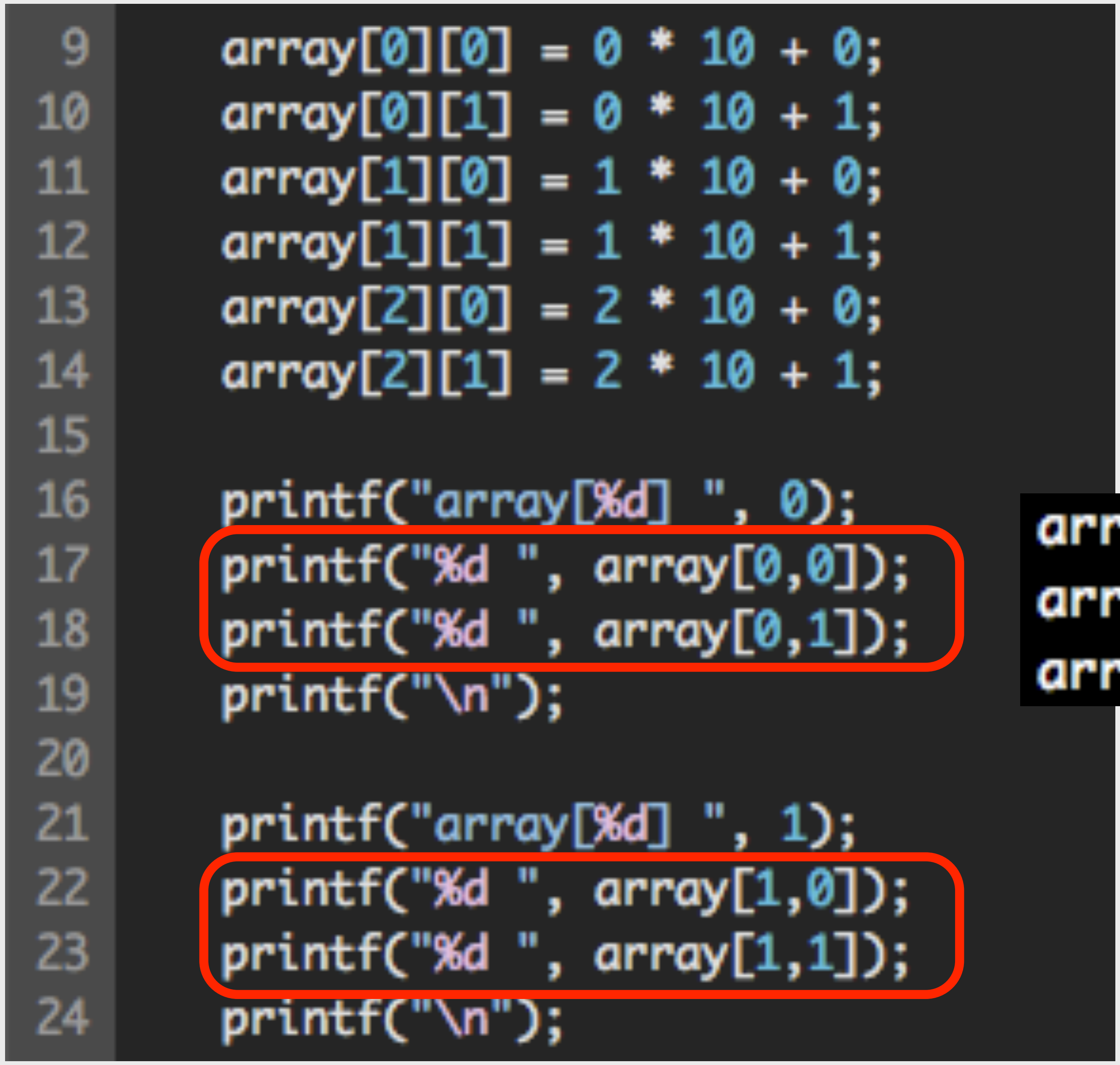


```
array[0] 4208 4216
array[1] 4208 4216
array[2] 4208 4216
```

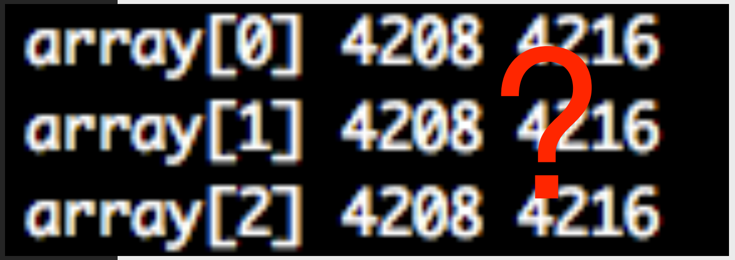
?

Multidimensional Arrays

- Example: multiarray.c



```
    array[0][0] = 0 * 10 + 0;
    array[0][1] = 0 * 10 + 1;
    array[1][0] = 1 * 10 + 0;
    array[1][1] = 1 * 10 + 1;
    array[2][0] = 2 * 10 + 0;
    array[2][1] = 2 * 10 + 1;

    printf("array[%d] ", 0);
    printf("%d ", array[0,0]);
    printf("%d ", array[0,1]);
    printf("\n");

    printf("array[%d] ", 1);
    printf("%d ", array[1,0]);
    printf("%d ", array[1,1]);
    printf("\n");
```



```
array[0] 4208 4216
array[1] 4208 4216
array[2] 4208 4216
```

?

Multidimensional Arrays

- C does not allow the notation used in other language of **`matrix[10,12]`**

```
printf("%d", array[0][0]);
printf("%d", array[0][1]);
...
```

Reading Numbers

Reading Numbers

- The function **`scanf`** is notorious

Reading Numbers

- The function **`scanf`** is notorious
    - because of its poor end-of-line handling

Reading Numbers

- The function **`scanf`** is notorious
    - because of its poor end-of-line handling
- In stead, use **`fgets`** to read a line of input and **`sscanf`** to convert the text into numbers

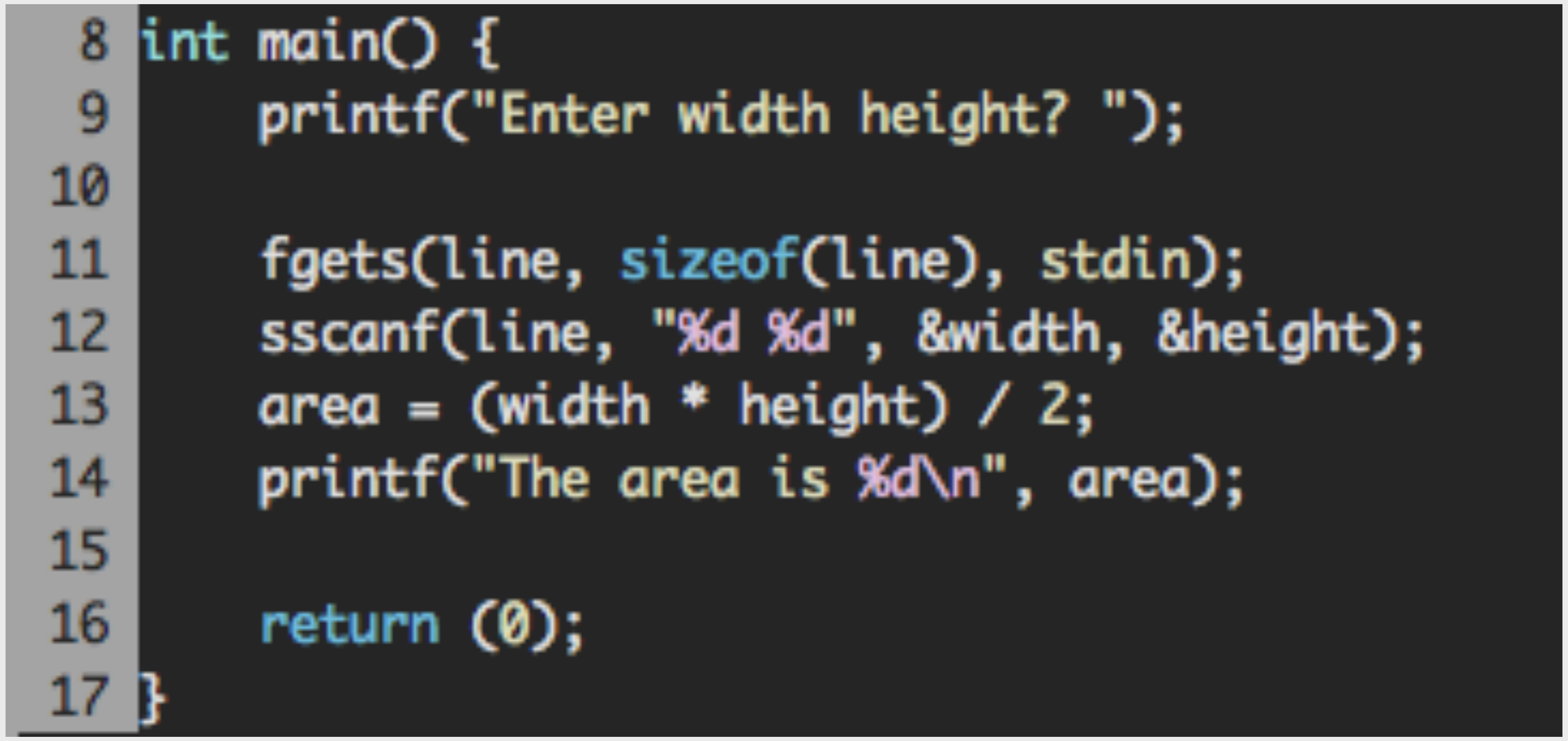
Reading Numbers

- The function **`scanf`** is notorious
  - because of its poor end-of-line handling
- In stead, use **`fgets`** to read a line of input and **`sscanf`** to convert the text into numbers

```
char line[100];
fgets(line, sizeof(line), stdin);
sscanf(line, format, &variable1, &variable2);
```

Reading Numbers

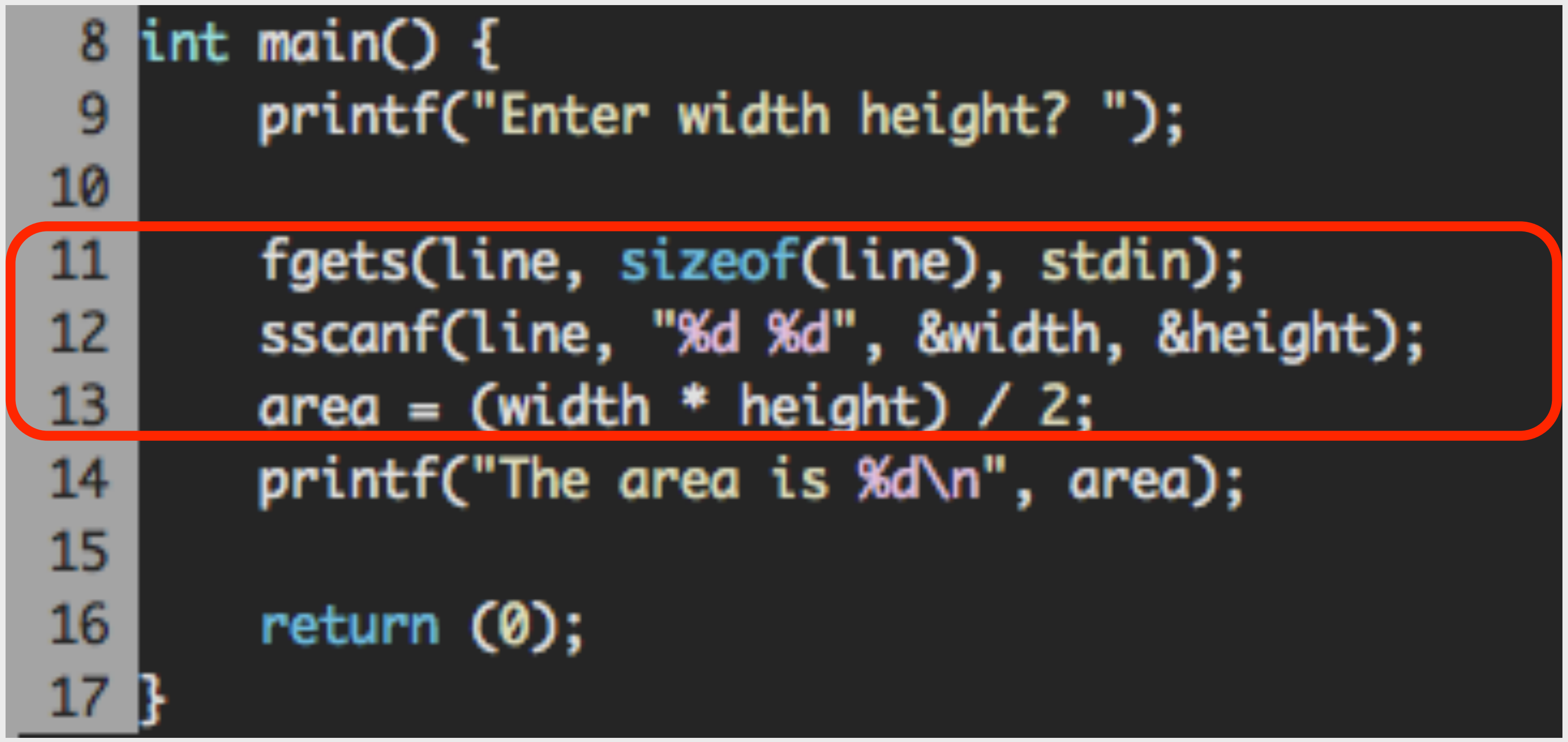
- Example: triangle.c



```c
int main() {
    printf("Enter width height? ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d", &width, &height);
    area = (width * height) / 2;
    printf("The area is %d\n", area);

    return (0);
}
```

Reading Numbers

- Example: triangle.c



```
int main() {
    printf("Enter width height? ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d", &width, &height);
    area = (width * height) / 2;
    printf("The area is %d\n", area);

    return (0);
}
```

Initialize Variables into Array

Initialize Variables into Array

```
int product_codes[3] = {10,972,45};
```

Initialize Variables into Array

```
int product_codes[3] = {10,972,45};

int matrix[2][4] =
{

  {1,2,3,4},

  {10,20,30,40}

};
```

Initialize Variables into Array

```
int product_codes[3] = {10,972,45};
```

```
int matrix[2][4] =
{
   {1,2,3,4},
   {10,20,30,40}
};
```

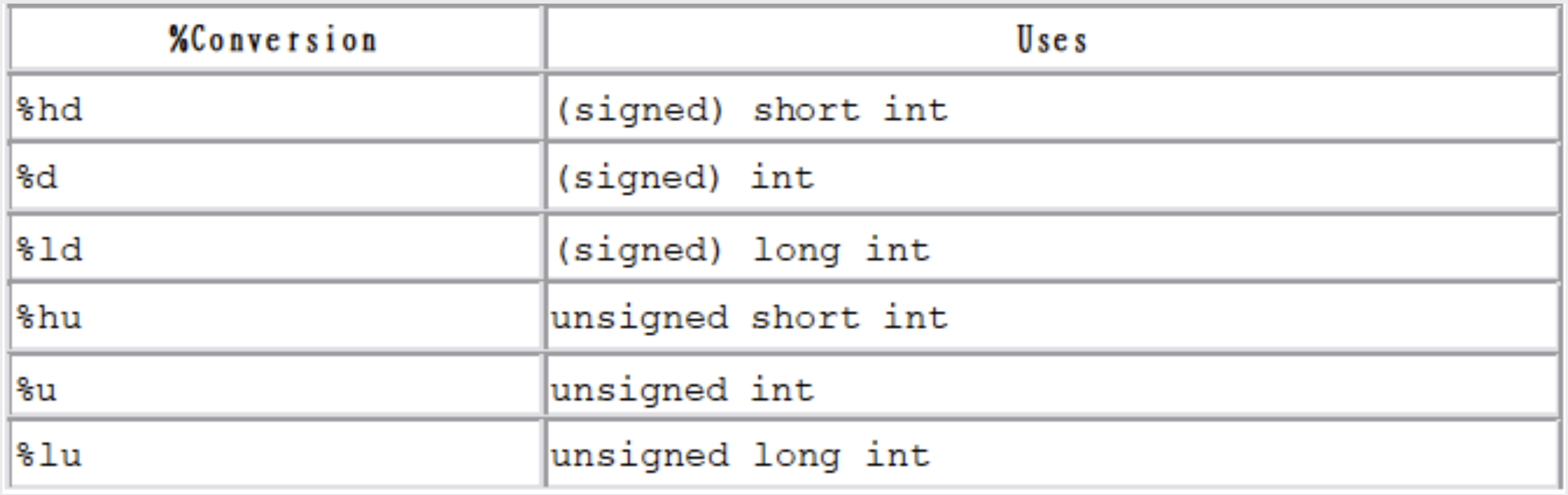
```
char name[50] = “Sam”;
```

is equivalent to

```
char name[50];
....
strcpy(name, “Sam”);
```

Types of Integers

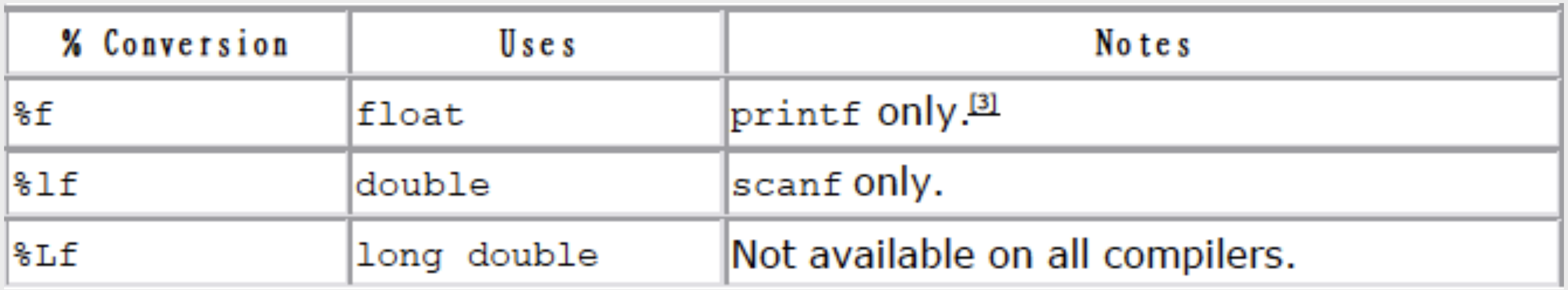
- Integer **`printf`**/**`scanf`** Conversions



%Conversion	Uses
`%hd`	`(signed) short int`
`%d`	`(signed) int`
`%ld`	`(signed) long int`
`%hu`	`unsigned short int`
`%u`	`unsigned int`
`%lu`	`unsigned long int`

Type of Floats

- Float **`printf`**/**`scanf`** Conversions



% Conversion	Uses	Notes
`%f`	`float`	`printf` only.[3]
`%lf`	`double`	`scanf` only.
`%Lf`	`long double`	Not available on all compilers.

++x or x++

++x or x++

- Which form should you use?

++x or x++

- Which form should you use?
  - In C, the choice doesn’t matter

++x or x++

- Which form should you use?
  - In C, the choice doesn’t matter
  - However, in C++, the prefix version (++x) is more efficient

++x or x++

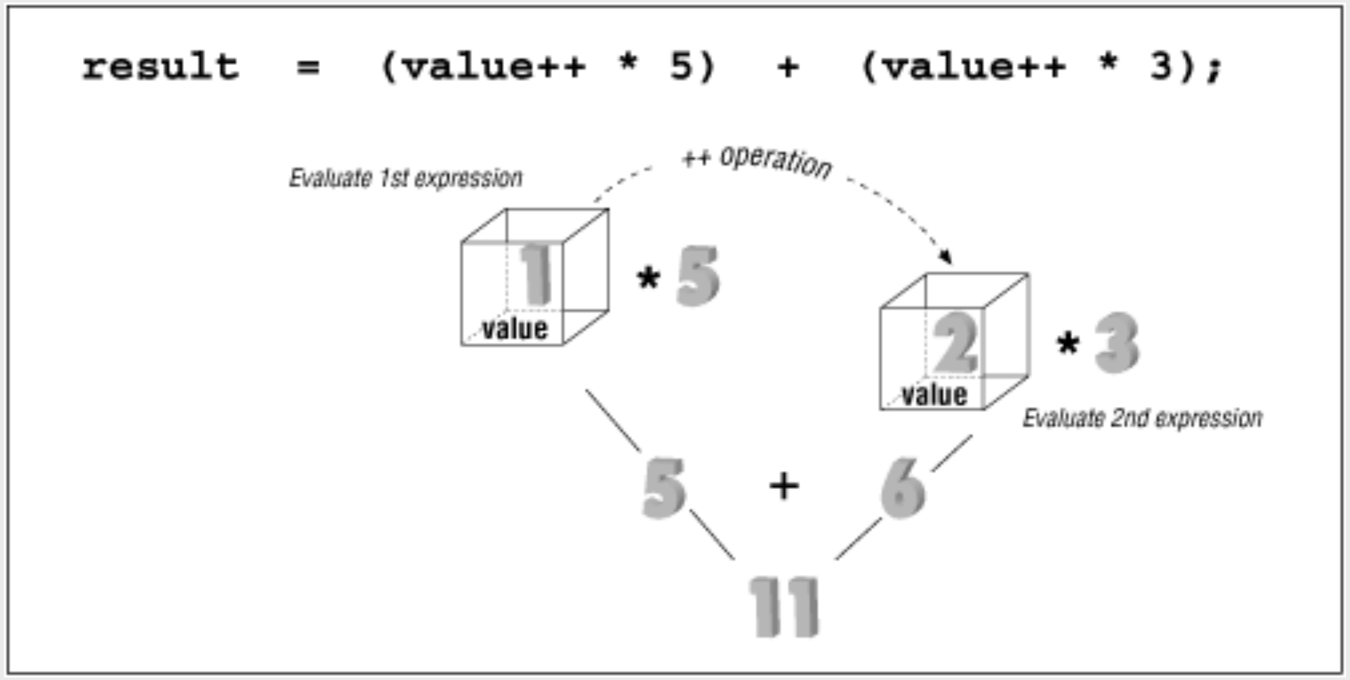
- Which form should you use?
  - In C, the choice doesn't matter
  - However, in C++, the prefix version (++x) is more efficient
  - In order to develop good habits for learning C++, use the prefix form

Side-Effect Problems

```
value = 1;

result = (value++ * 5) + (value++ * 3);
```

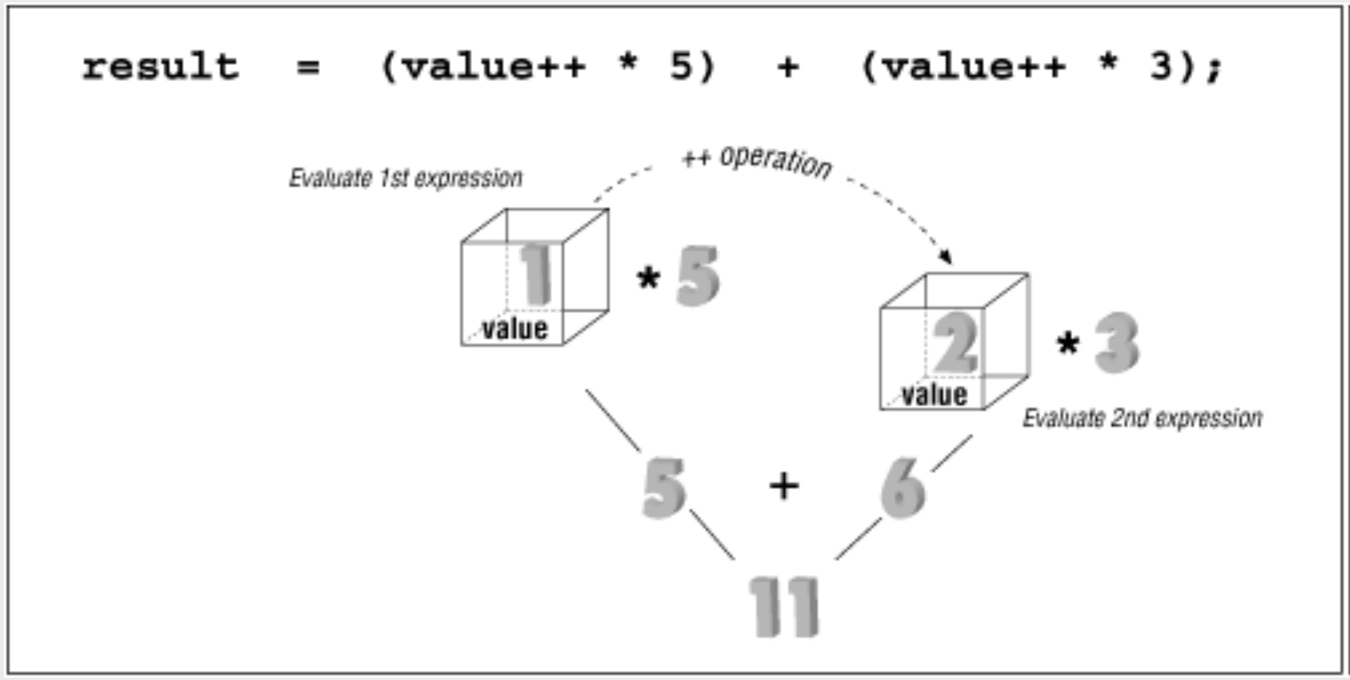
Side-Effect Problems

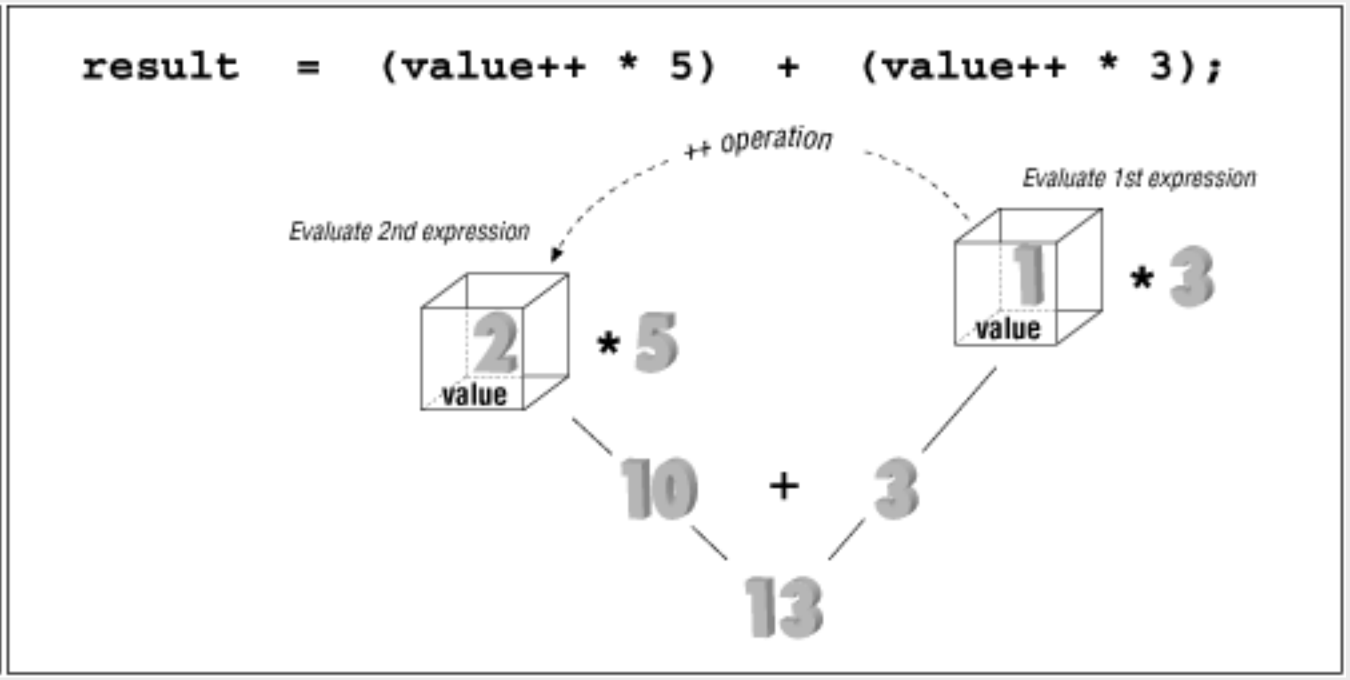
```
value = 1;

result = (value++ * 5) + (value++ * 3);
```



Side-Effect Problems

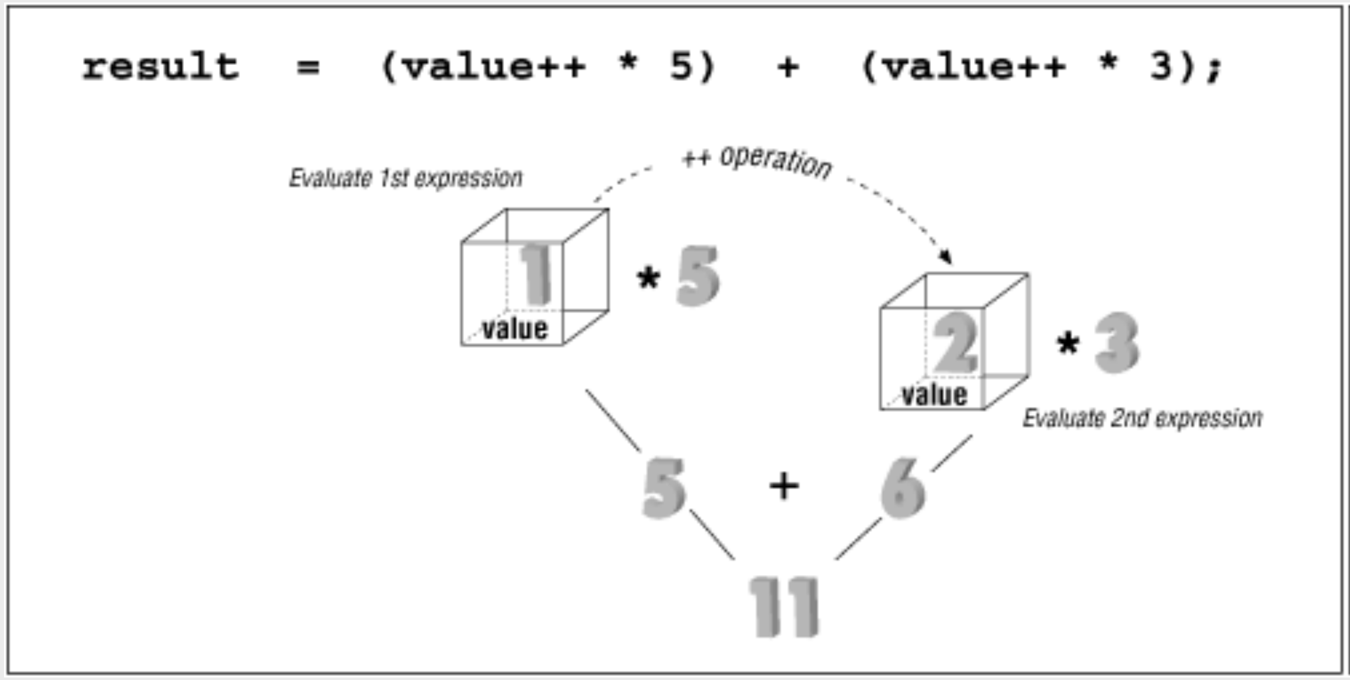
```
value = 1;

result = (value++ * 5) + (value++ * 3);
```

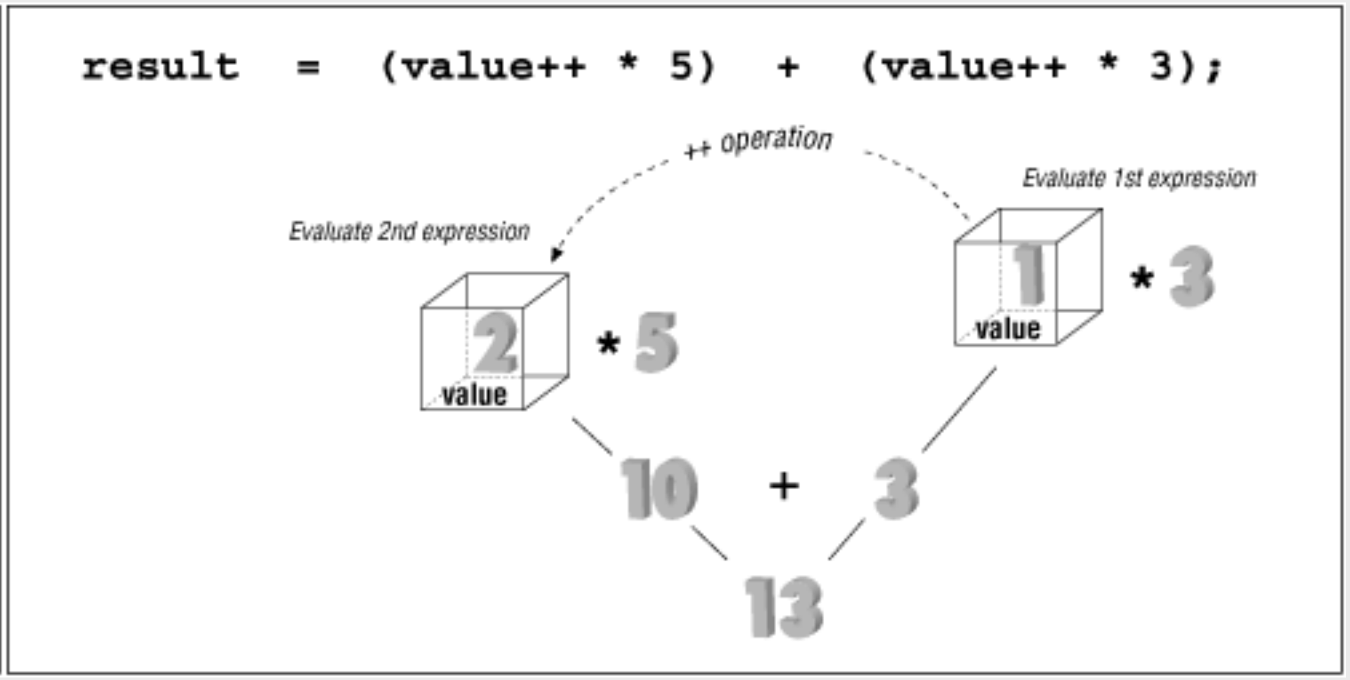




Side-Effect Problems

```
value = 1;

result = (value++ * 5) + (value++ * 3);
```

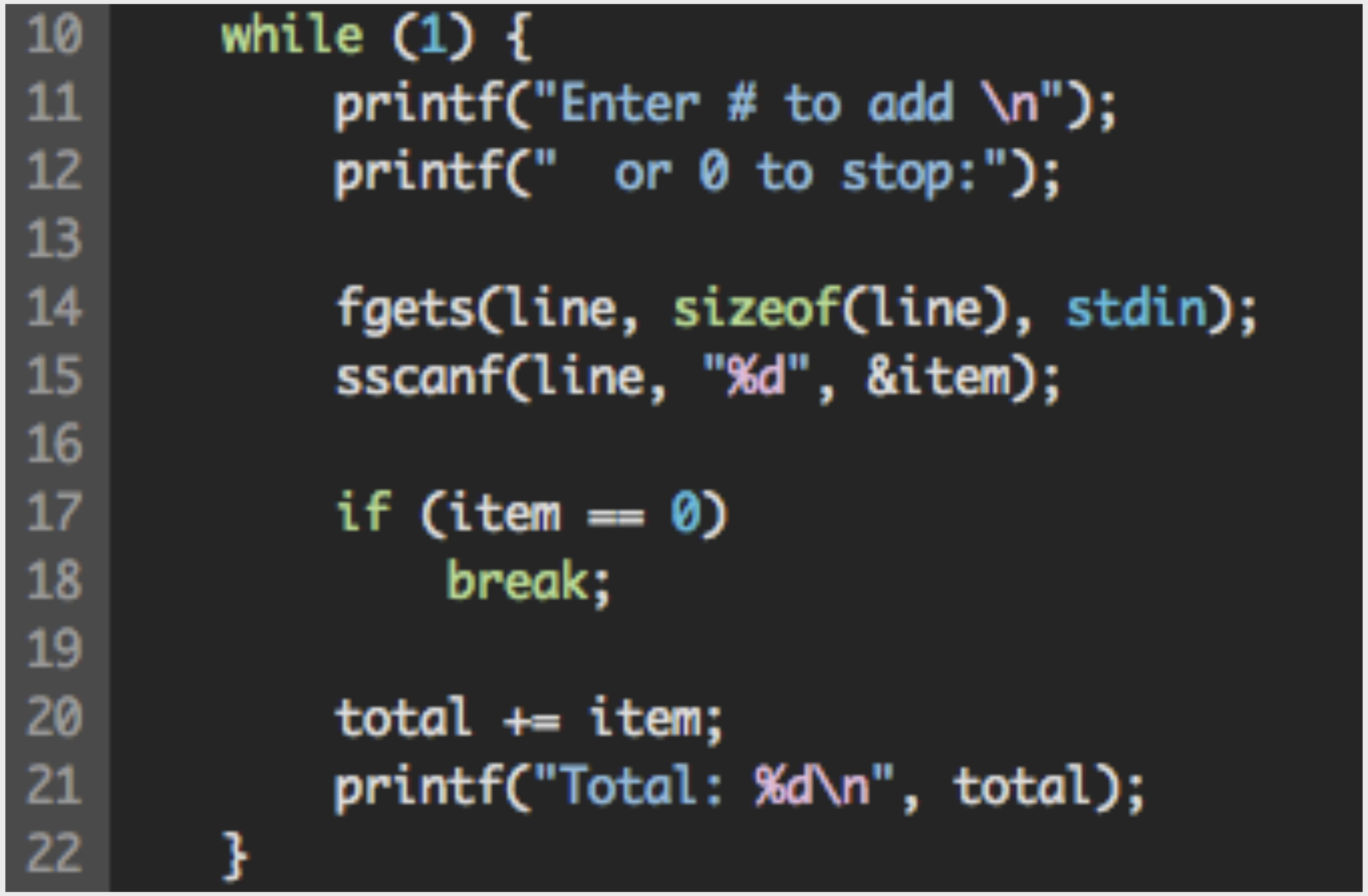




In order to avoid the trouble, always put ++ and -- on a line by themselves

`break` statement

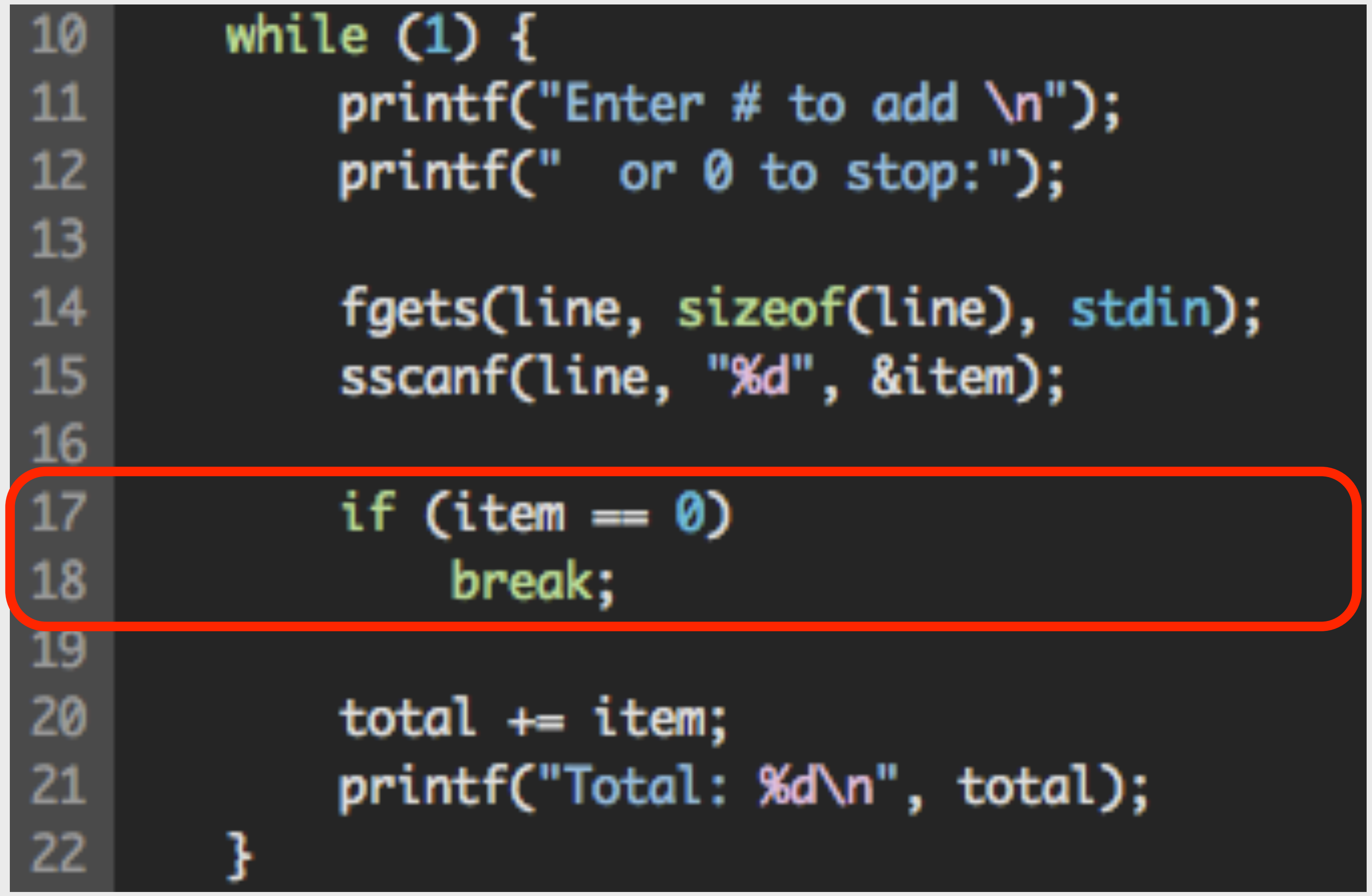
- Example: total_break.c



```
    while (1) {
        printf("Enter # to add \n");
        printf("  or 0 to stop:");

        fgets(line, sizeof(line), stdin);
        sscanf(line, "%d", &item);

        if (item == 0)
            break;

        total += item;
        printf("Total: %d\n", total);
    }
```

break statement

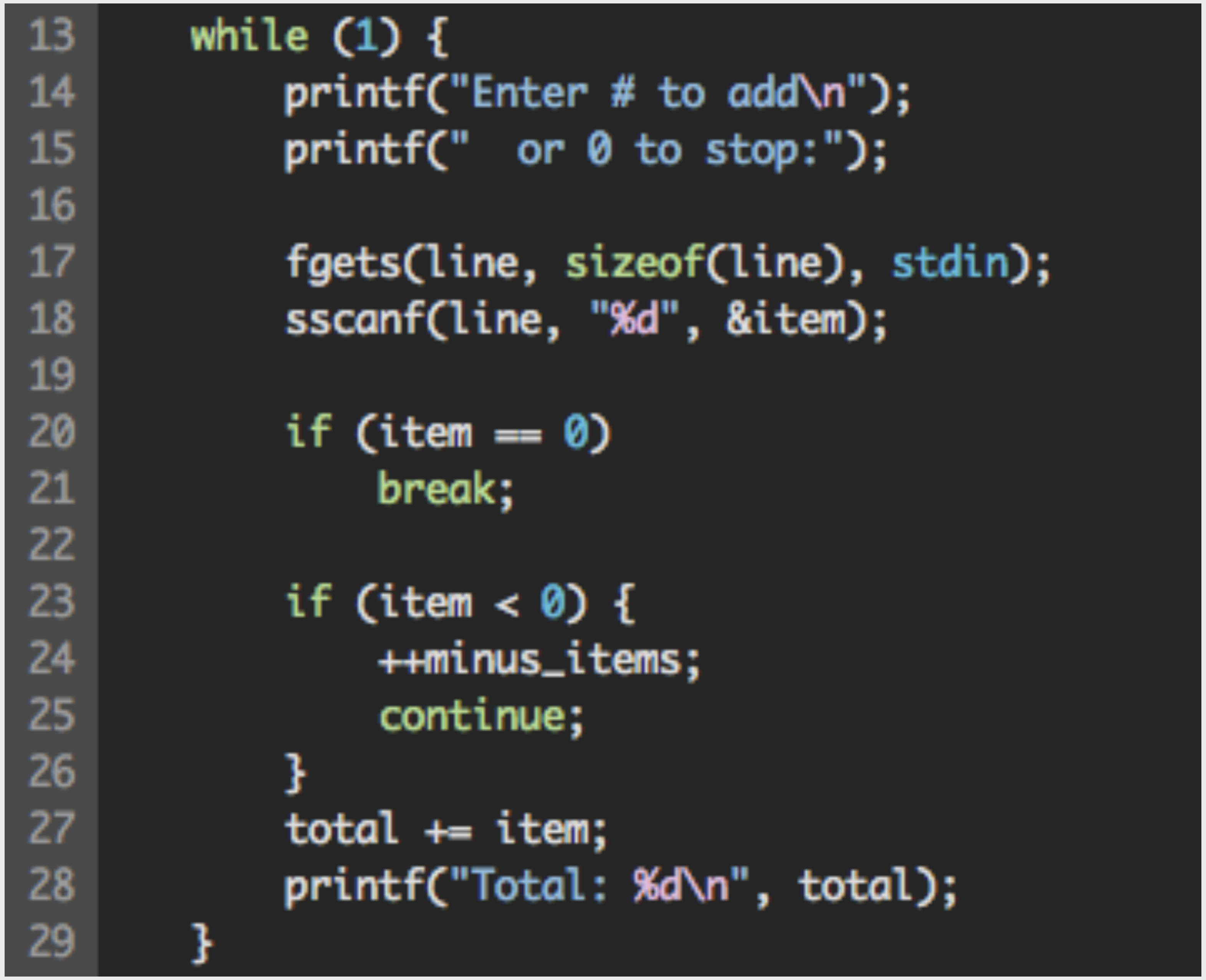
- Example: total_break.c



```
    while (1) {
        printf("Enter # to add \n");
        printf("  or 0 to stop:");

        fgets(line, sizeof(line), stdin);
        sscanf(line, "%d", &item);

        if (item == 0)
            break;

        total += item;
        printf("Total: %d\n", total);
    }
```

`continue` statement

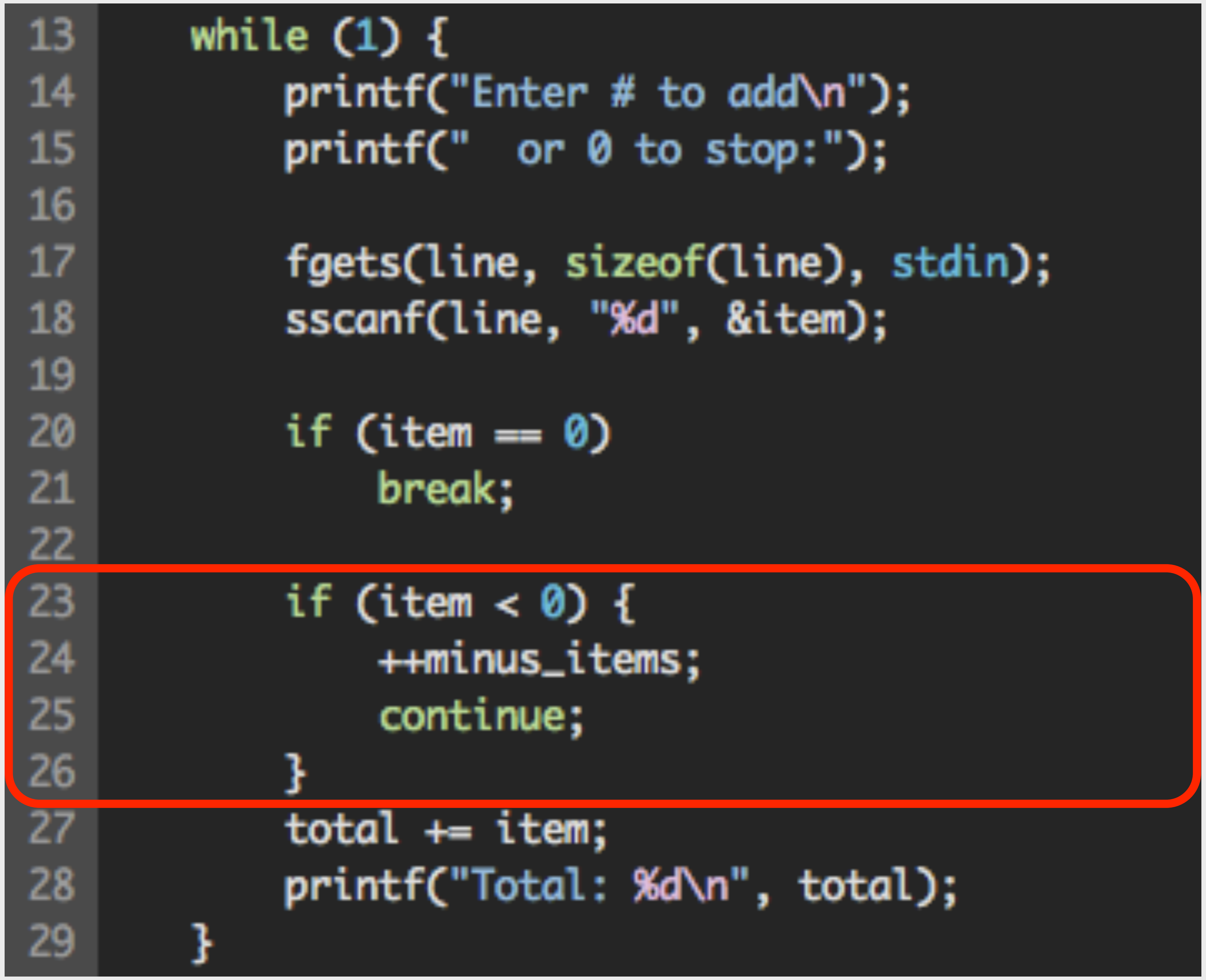
- Example: total_continue.c



```c
    while (1) {
        printf("Enter # to add\n");
        printf("   or 0 to stop:");

        fgets(line, sizeof(line), stdin);
        sscanf(line, "%d", &item);

        if (item == 0)
            break;

        if (item < 0) {
            ++minus_items;
            continue;
        }
        total += item;
        printf("Total: %d\n", total);
    }
```

`continue` statement

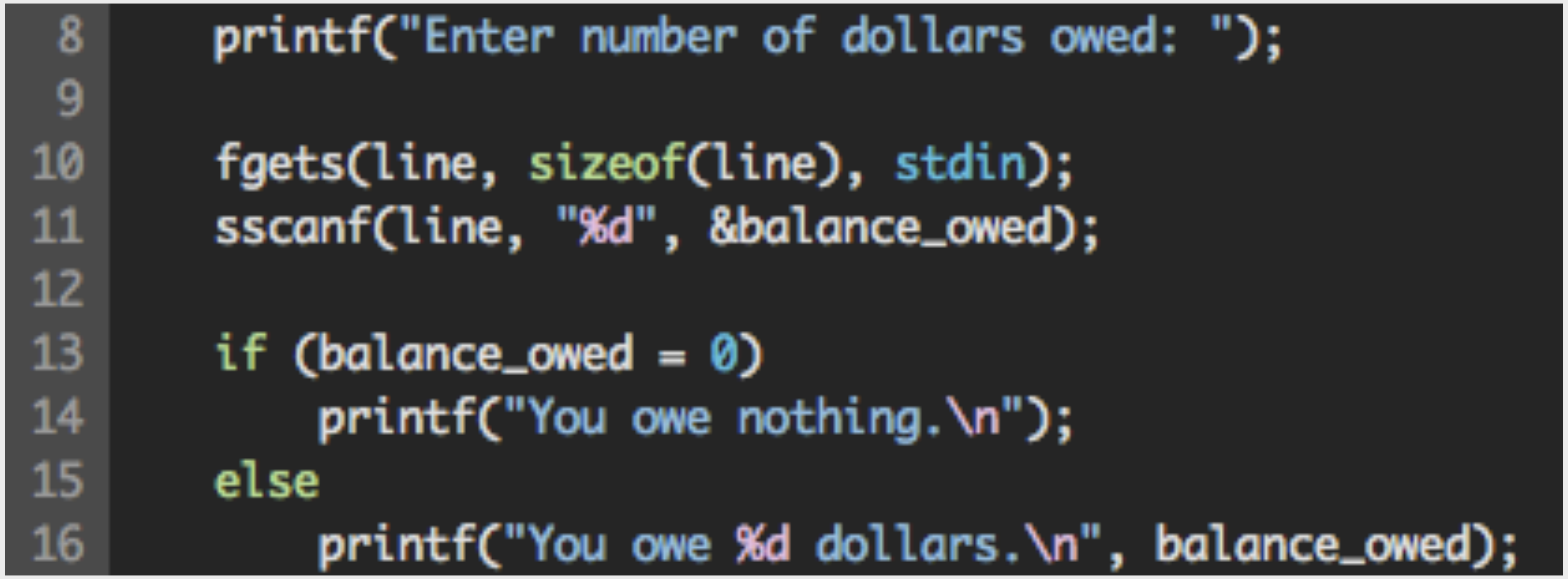
- Example: total_continue.c



```
    while (1) {
        printf("Enter # to add\n");
        printf("   or 0 to stop:");

        fgets(line, sizeof(line), stdin);
        sscanf(line, "%d", &item);

        if (item == 0)
            break;

        if (item < 0) {
            ++minus_items;
            continue;
        }
        total += item;
        printf("Total: %d\n", total);
    }
```

Assignment Anywhere Side Effect

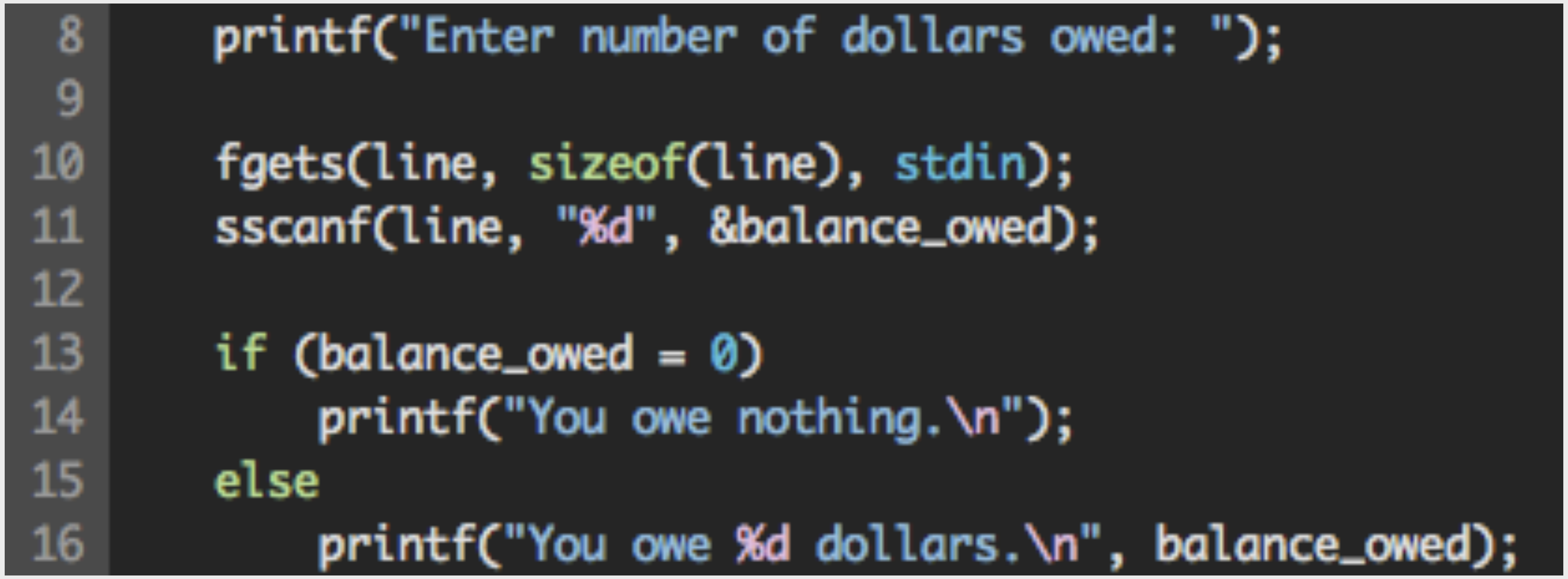
- Example: owe0.c



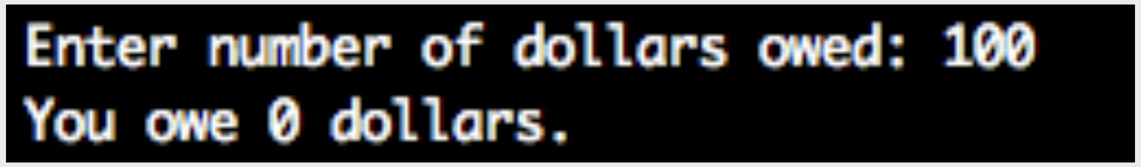
```c
    printf("Enter number of dollars owed: ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d", &balance_owed);

    if (balance_owed = 0)
        printf("You owe nothing.\n");
    else
        printf("You owe %d dollars.\n", balance_owed);
```

Assignment Anywhere Side Effect

- Example: owe0.c



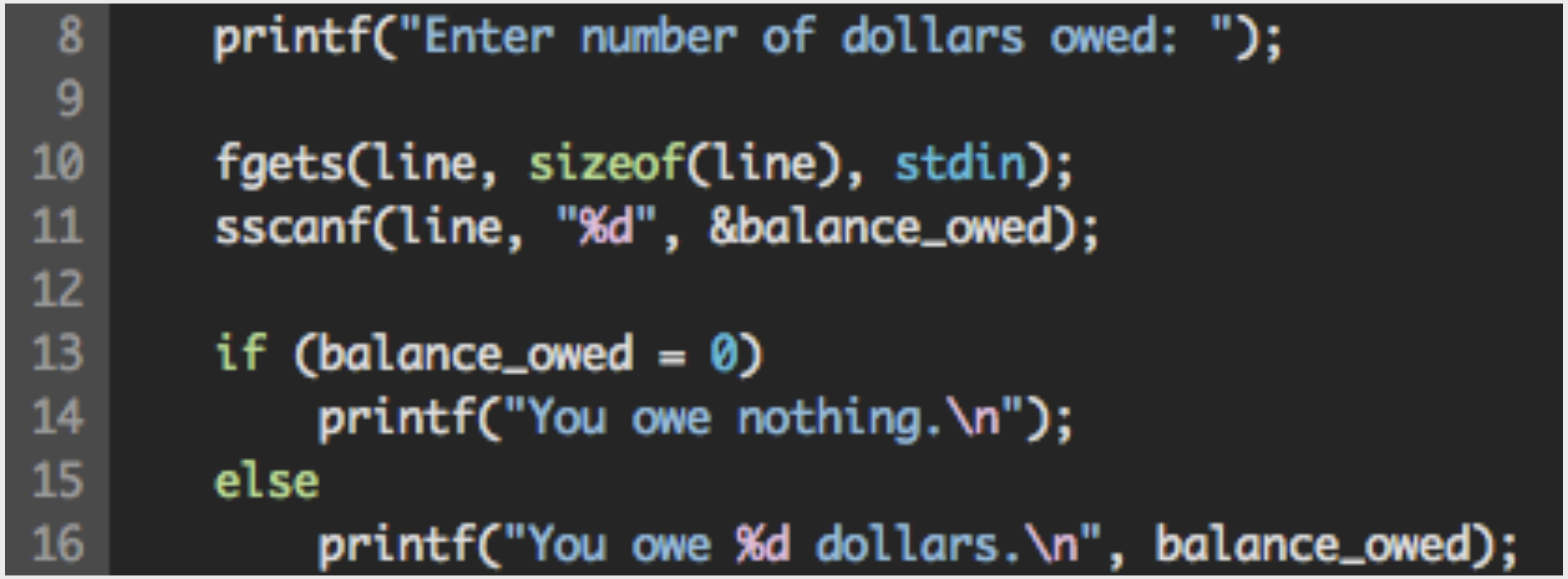
```
    printf("Enter number of dollars owed: ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d", &balance_owed);

    if (balance_owed = 0)
        printf("You owe nothing.\n");
    else
        printf("You owe %d dollars.\n", balance_owed);
```



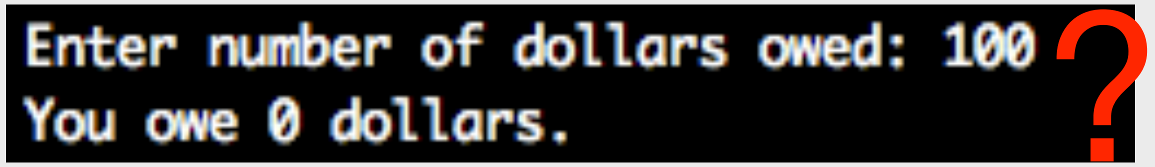
```
Enter number of dollars owed: 100
You owe 0 dollars.
```

Assignment Anywhere Side Effect

- Example: owe0.c



```
    printf("Enter number of dollars owed: ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d", &balance_owed);

    if (balance_owed = 0)
        printf("You owe nothing.\n");
    else
        printf("You owe %d dollars.\n", balance_owed);
```

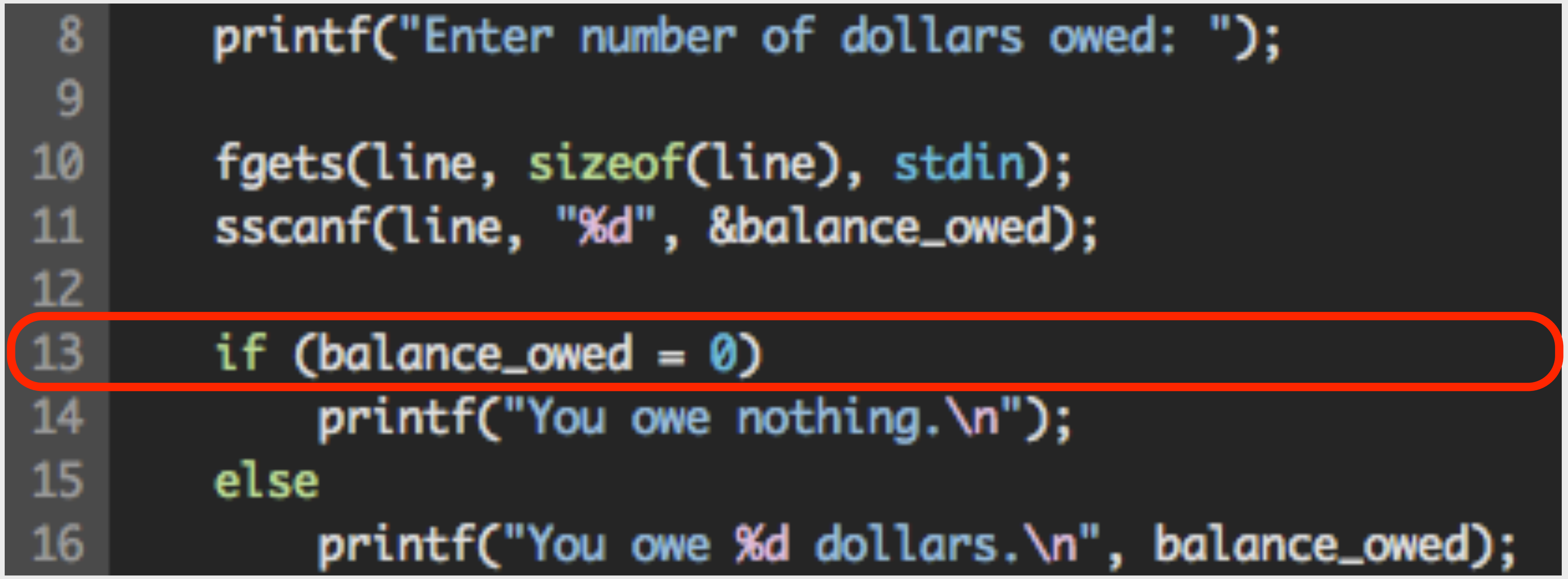


```
Enter number of dollars owed: 100
You owe 0 dollars.
```

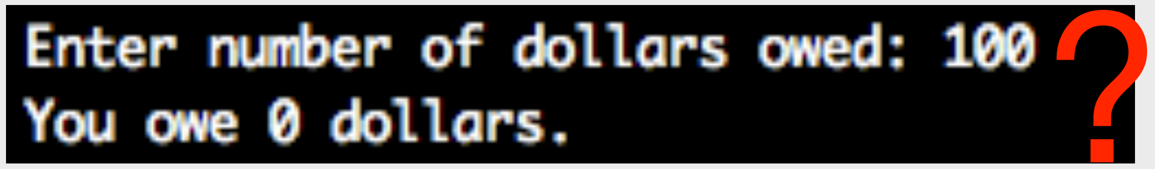
?

Assignment Anywhere Side Effect

- Example: owe0.c



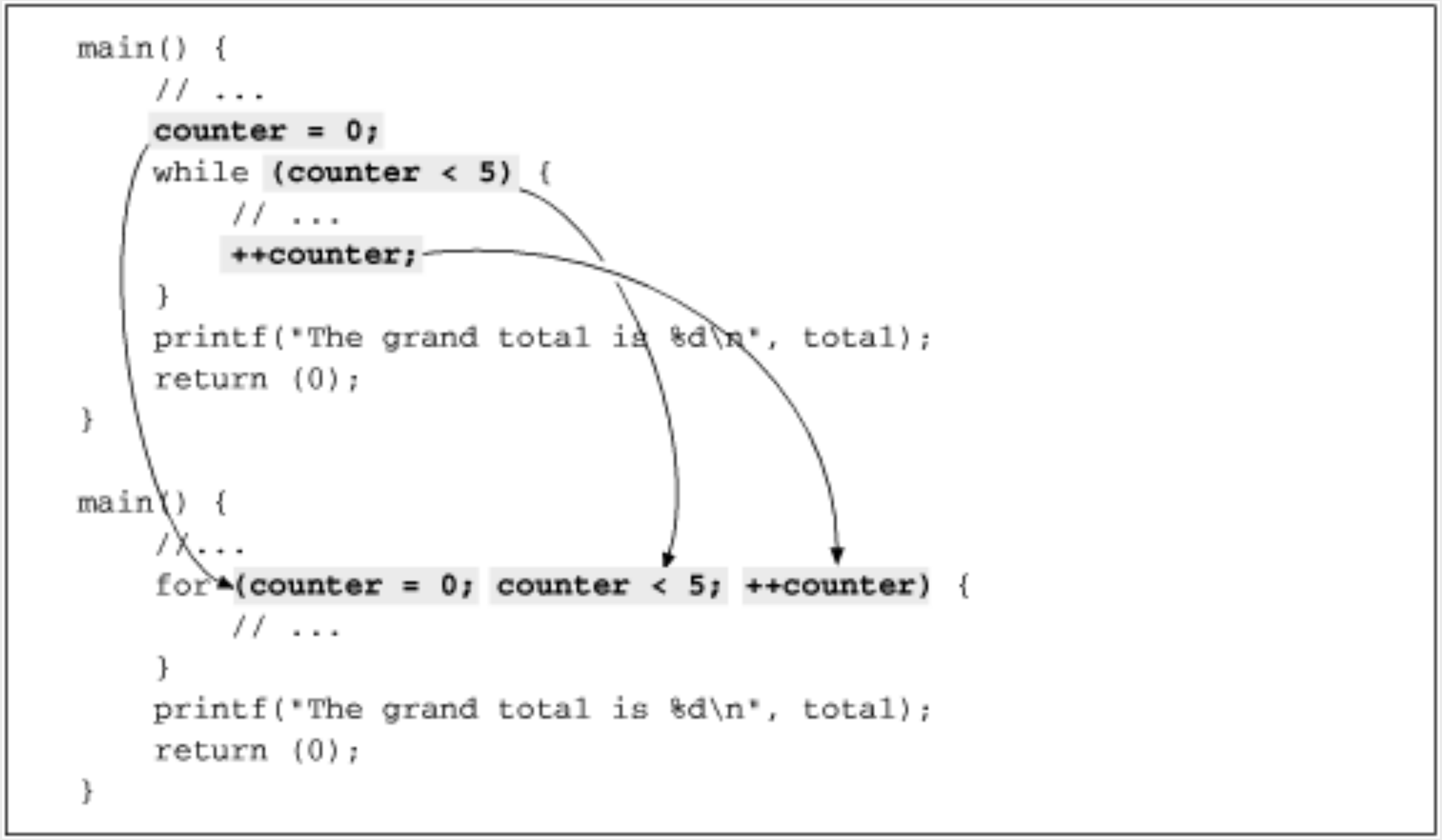
```
    printf("Enter number of dollars owed: ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d", &balance_owed);

    if (balance_owed = 0)
        printf("You owe nothing.\n");
    else
        printf("You owe %d dollars.\n", balance_owed);
```



```
Enter number of dollars owed: 100
You owe 0 dollars.
```

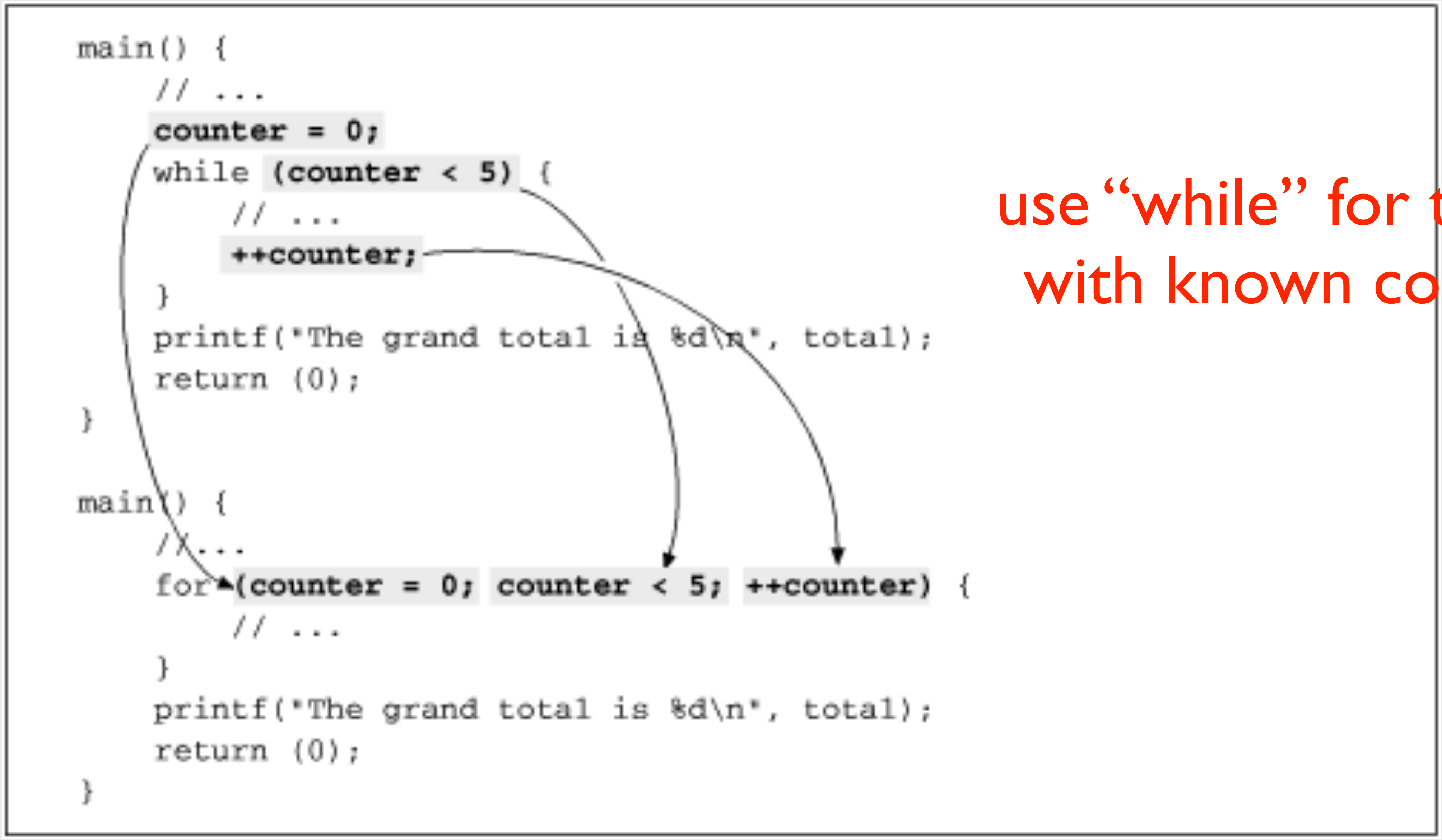
?

Similarities between “while” and “for”



```
main() {
    // ...
    counter = 0;
    while (counter < 5) {
        // ...
        ++counter;
    }
    printf("The grand total is %d\n", total);
    return (0);
}

main() {
    // ...
    for (counter = 0; counter < 5; ++counter) {
        // ...
    }
    printf("The grand total is %d\n", total);
    return (0);
}
```

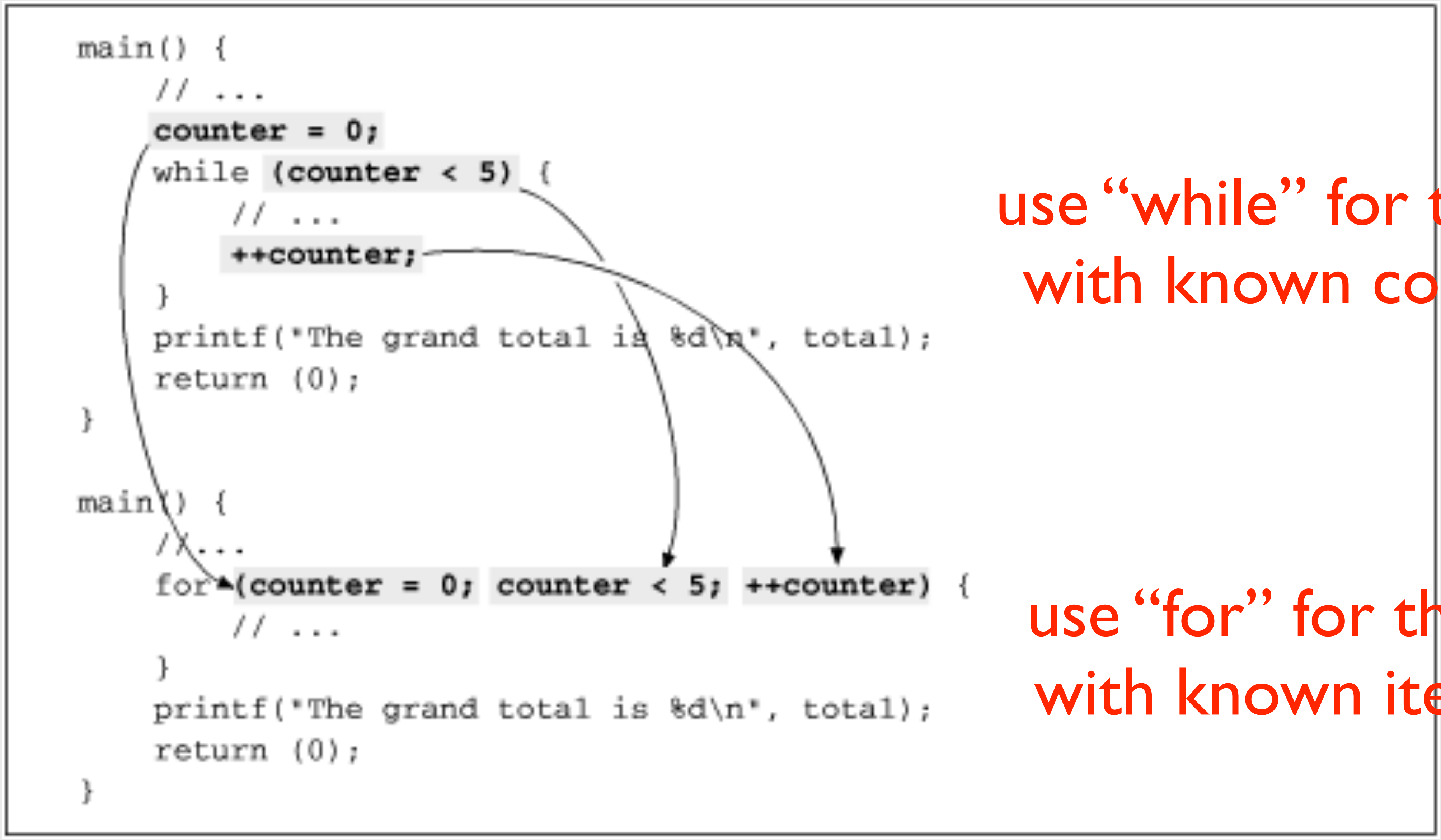
Similarities between “while” and “for”



```
main() {
    // ...
    counter = 0;
    while (counter < 5) {
        // ...
        ++counter;
    }
    printf("The grand total is %d\n", total);
    return (0);
}

main() {
    //...
    for (counter = 0; counter < 5; ++counter) {
        // ...
    }
    printf("The grand total is %d\n", total);
    return (0);
}
```

use “while” for the loops with known conditions

Similarities between “while” and “for”



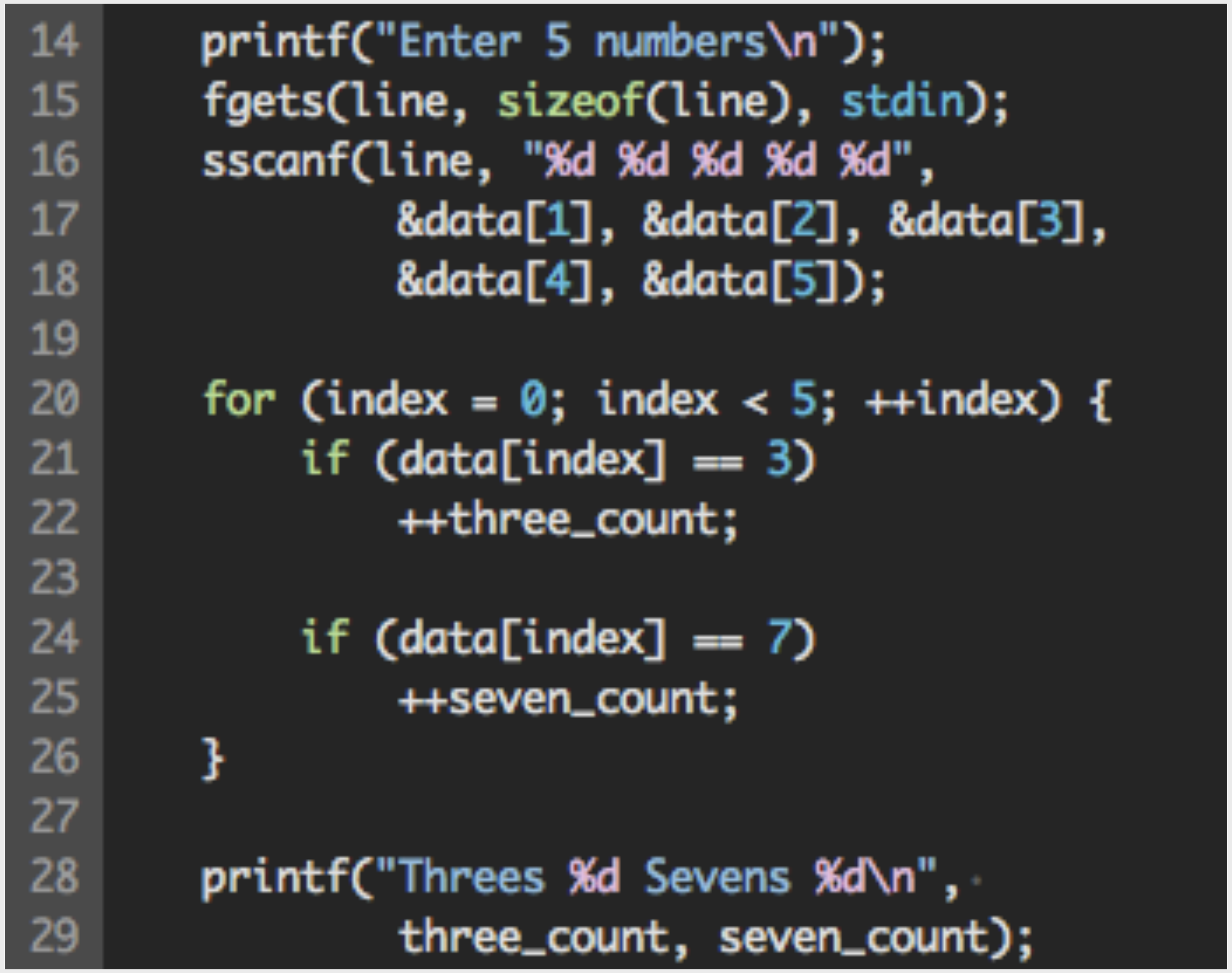
```
main() {
    // ...
    counter = 0;
    while (counter < 5) {
        // ...
        ++counter;
    }
    printf("The grand total is %d\n", total);
    return (0);
}

main() {
    // ...
    for (counter = 0; counter < 5; ++counter) {
        // ...
    }
    printf("The grand total is %d\n", total);
    return (0);
}
```

use “while” for the loops with known conditions

use “for” for the loops with known iterations

for statement

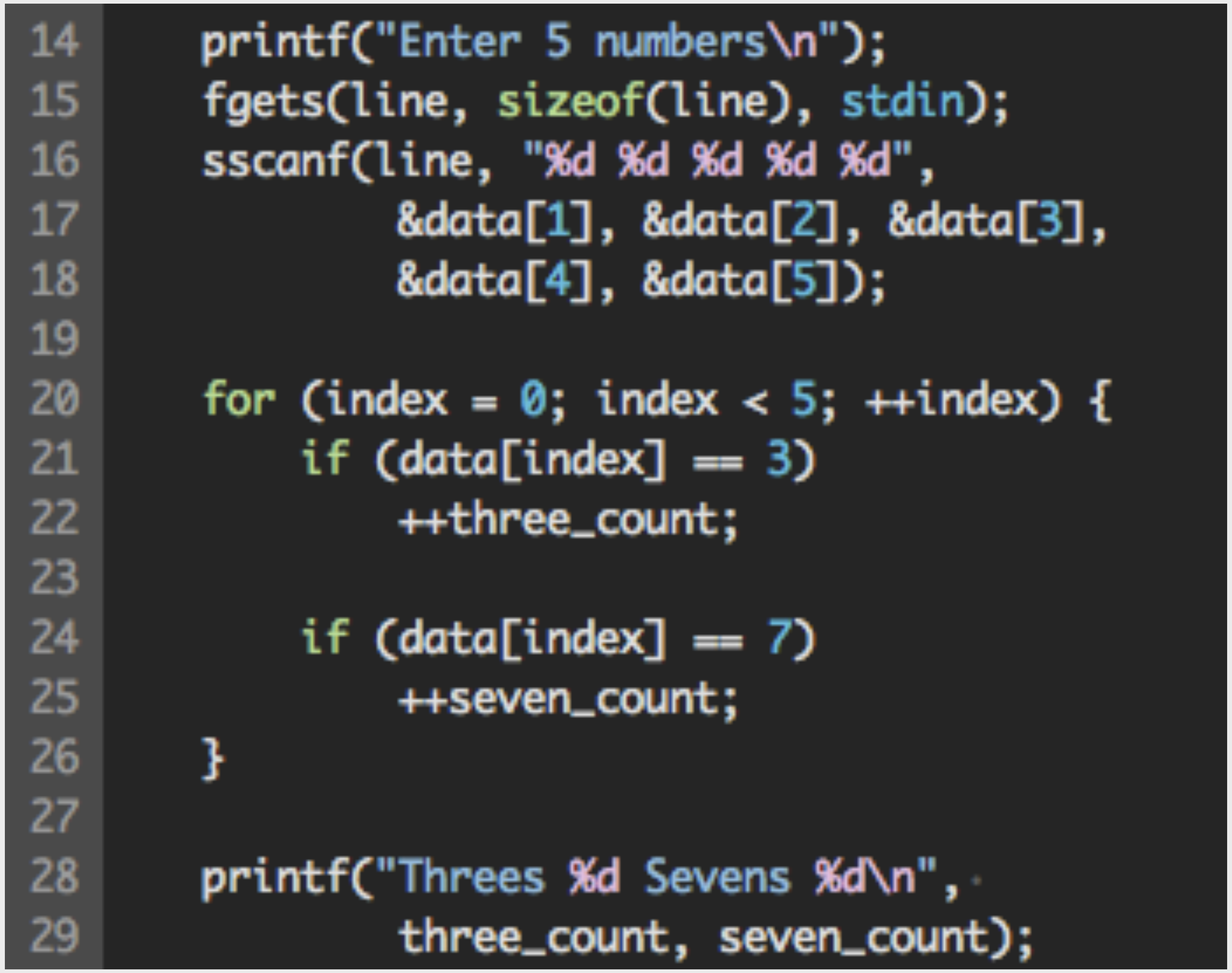
- Example: count_number.c



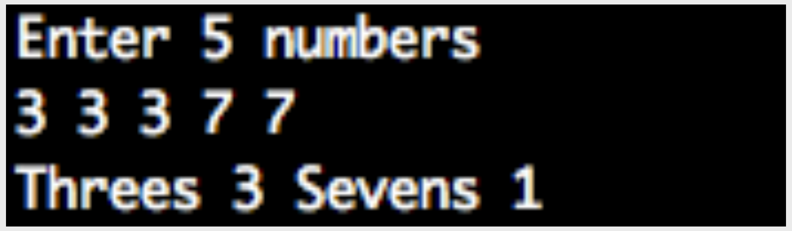
```c
    printf("Enter 5 numbers\n");
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d %d %d %d",
            &data[1], &data[2], &data[3],
            &data[4], &data[5]);

    for (index = 0; index < 5; ++index) {
        if (data[index] == 3)
            ++three_count;

        if (data[index] == 7)
            ++seven_count;
    }

    printf("Threes %d Sevens %d\n",
            three_count, seven_count);
```

for statement

- Example: count_number.c



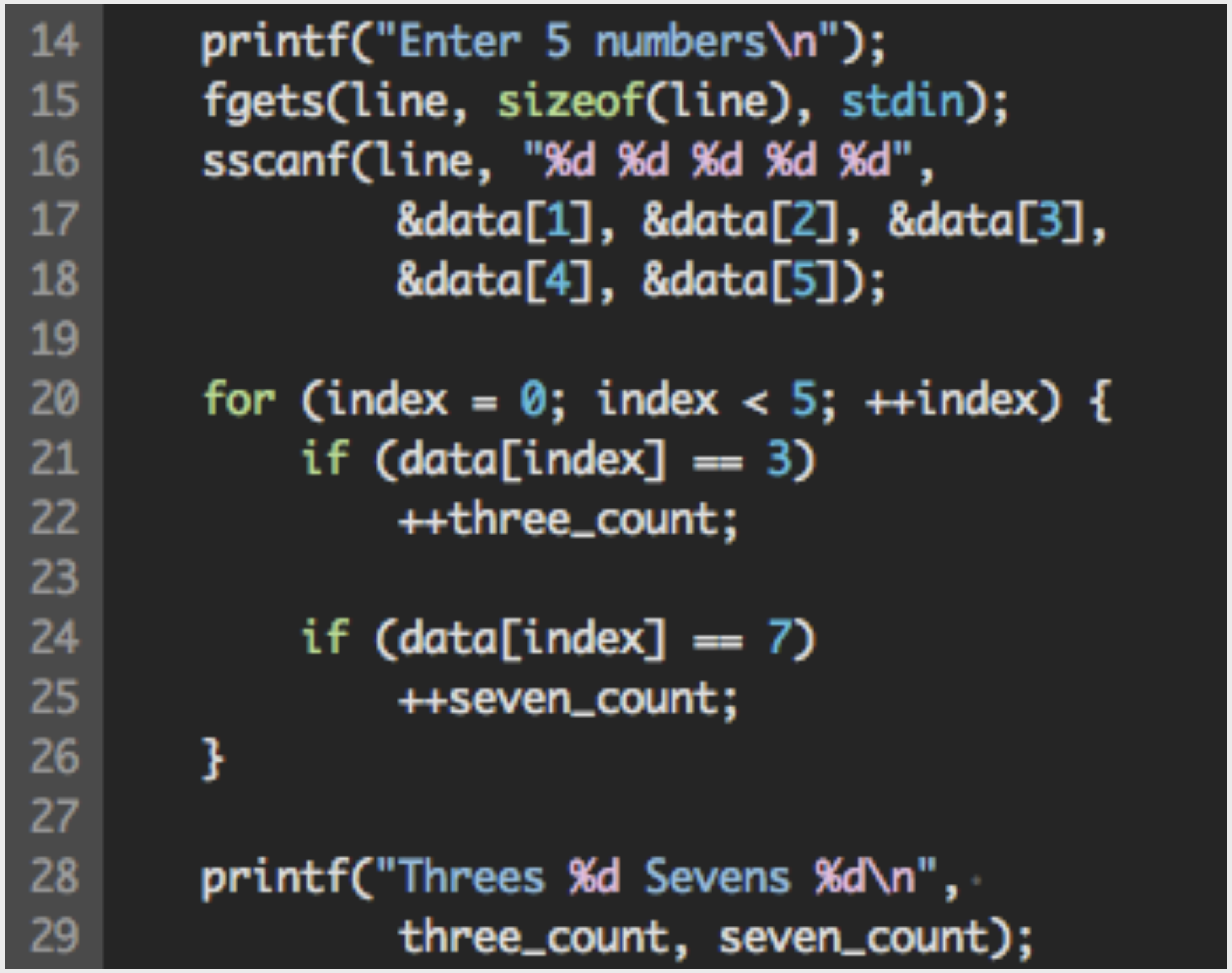
```
    printf("Enter 5 numbers\n");
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d %d %d %d",
            &data[1], &data[2], &data[3],
            &data[4], &data[5]);

    for (index = 0; index < 5; ++index) {
        if (data[index] == 3)
            ++three_count;

        if (data[index] == 7)
            ++seven_count;
    }

    printf("Threes %d Sevens %d\n",
            three_count, seven_count);
```



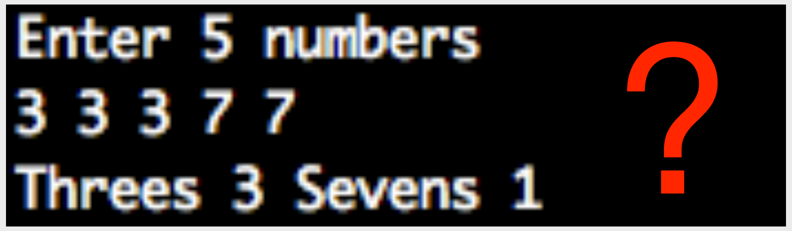
```
Enter 5 numbers
3 3 3 7 7
Threes 3 Sevens 1
```

for statement

- Example: count_number.c



```
    printf("Enter 5 numbers\n");
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d %d %d %d",
            &data[1], &data[2], &data[3],
            &data[4], &data[5]);

    for (index = 0; index < 5; ++index) {
        if (data[index] == 3)
            ++three_count;

        if (data[index] == 7)
            ++seven_count;
    }

    printf("Threes %d Sevens %d\n",
            three_count, seven_count);
```

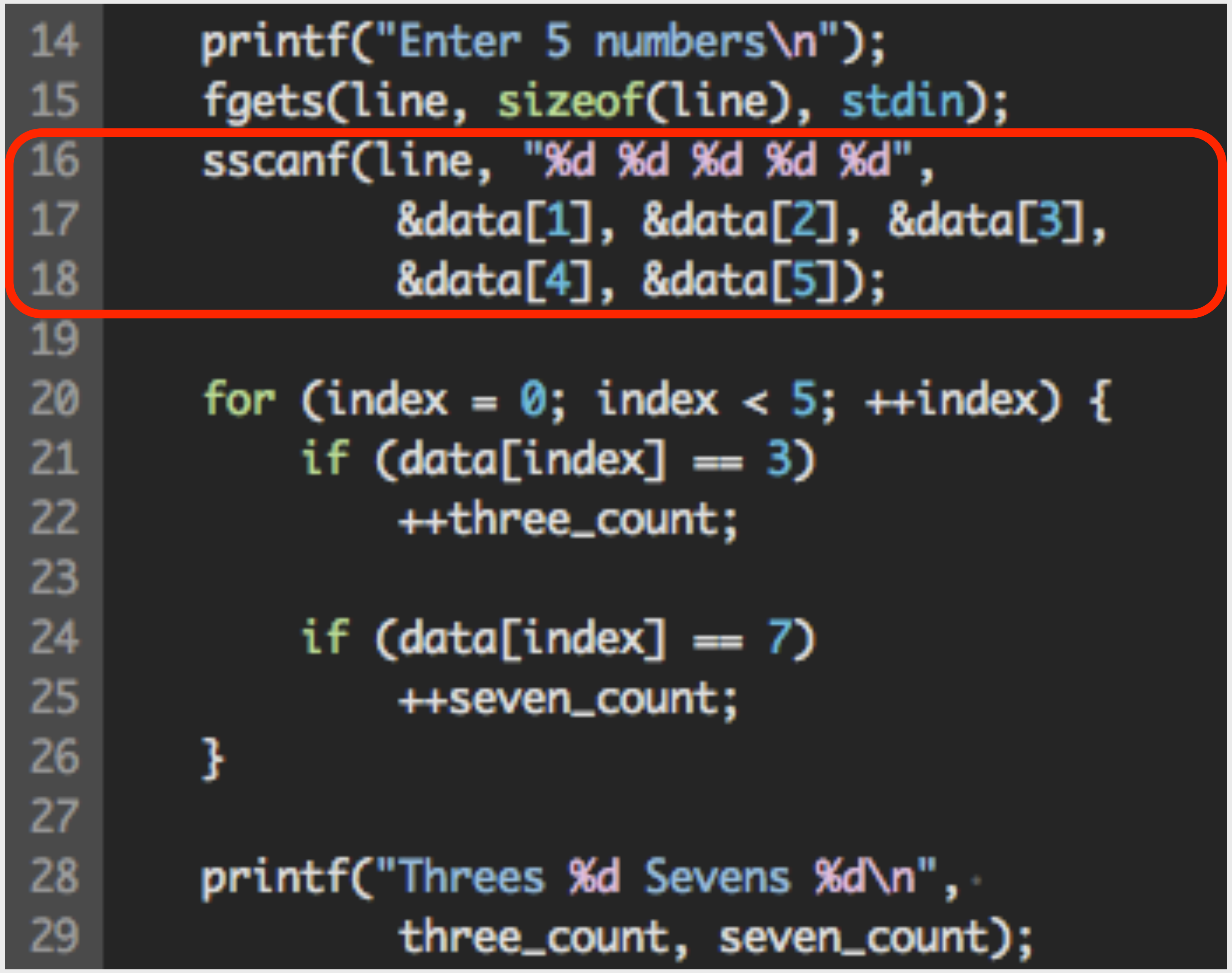


```
Enter 5 numbers
3 3 3 7 7
Threes 3 Sevens 1
```

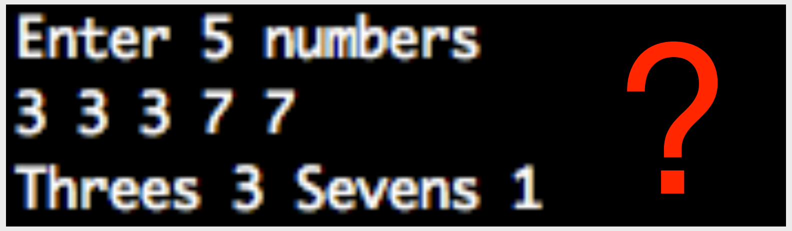
?

for statement

- Example: count_number.c



```
    printf("Enter 5 numbers\n");
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d %d %d %d",
           &data[1], &data[2], &data[3],
           &data[4], &data[5]);

    for (index = 0; index < 5; ++index) {
        if (data[index] == 3)
            ++three_count;

        if (data[index] == 7)
            ++seven_count;
    }

    printf("Threes %d Sevens %d\n",
           three_count, seven_count);
```



```
Enter 5 numbers
3 3 3 7 7
Threes 3 Sevens 1
```

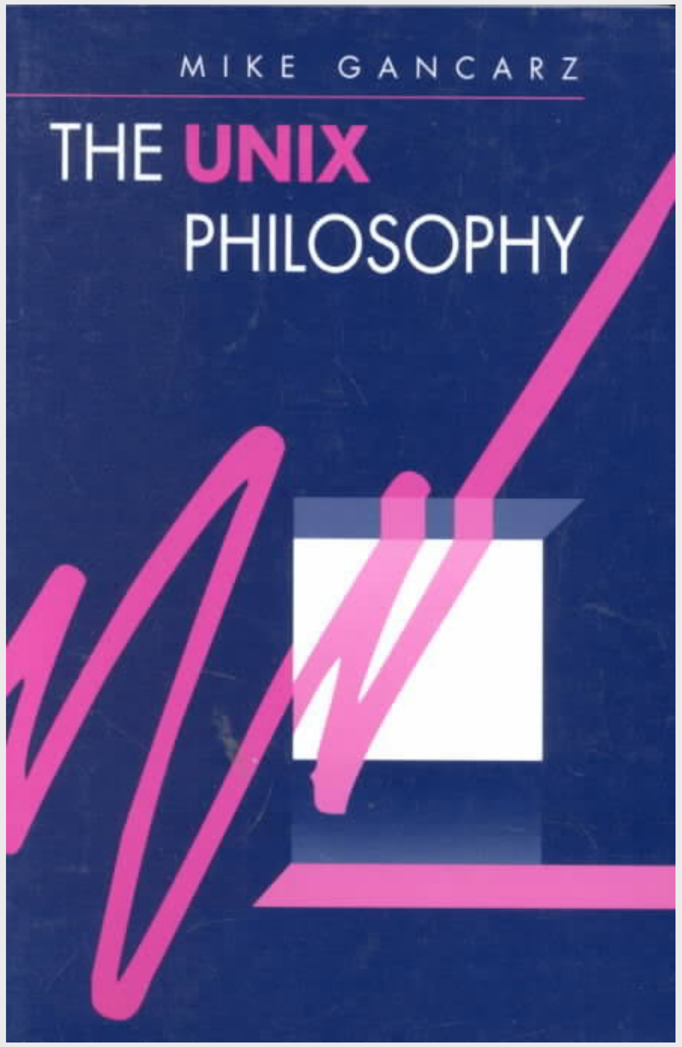
?

Dennis Ritchie

- An American computer scientist and winner, with Kenneth Thompson, of the 1983 Turing Award.
- He created the C programming language and, with Thompson, the UNIX operating system
- “UNIX is very simple, it just needs a genius to understand its simplicity.”
- “C is quirky, flawed, and an enormous success.”

UNIX Philosophy

- Small is beautiful.
- Make each program do one thing well.
- Build a prototype as soon as possible.
- Choose portability over efficiency.
- Store data in flat text files.
- Use software leverage to your advantage.
- Use shell scripts to increase leverage and portability.
- Avoid captive user interfaces.
- Make every program a filter.



Unix Tips

- Terminal Multiplexer (tmux)
  - A software to multiplex several virtual consoles, allowing a user to access multiple separate terminal sessions inside a single terminal window or remote terminal session
  - tmux 基本教學
  - tmux Tutorial

Browser Tips

Vimium - The Hacker's Browser

- Vimium
  - The Hacker’s Browser
  - Vimium is a Google Chrome extension which provides keyboard shortcuts for navigation and control in the spirit of the Vim editor.
  - http://vimium.github.io/