CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 5.2: Advanced looping

Toronto Metropolitan University

Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

Today

Looping #2

Nested loops, EOF-controlled loops, loop examples

3

Loop Structures

1) `for` loop
2) `while` loop
3) `do/while` loop

How do I decide?

© Alex Ufkes, 2020, 2023

for loop

Use when the number of iterations is known in advance.

```
int i;
for (i = 1; i <= 5; i++) {
    printf(“Hello, World!\n”);
}
```

A for loop conveniently integrates the initialization, condition, and update steps into a single line.

do/while loop

Use when the loop body must execute at least once.

```
int value;
do {
    printf(“Enter an integer: ”);
    scanf(“%d”, &value);
} while (value >=0);
```

A do/while loop is ideal for input validation.

while loop

Use in all other situations.

```
int n = 1;
while (n <= 2) {
    printf(“%d\n”, n);
    n = n + 1;
}
```

A while loop can be adapted to any looping task.

A Loop is as Loop is a Loop

If the task can be solved with a `while` loop, it can also be solved with a `for` loop, and so on.

By adjusting initializations, conditions, and updates, every loop can be tweaked to be the functional equivalent of another.

This is FANTASTIC practice! Try solving a looping problem with each of three loop styles.

Write a program that reads in five numbers from the keyboard and prints out the largest to the screen.

Write a program that reads in five numbers from the keyboard and prints out the largest to the screen.

```c
#include <stdio.h>

int main(void)
{
    double maximum, input;
    int i;

    scanf(“%lf”, &maximum);

    for (i = 0; i < 4; i++)
    {
        scanf(“%lf”, &input);

        if (input > maximum)
            maximum = input;
    }
    printf(“Max: %lf”, maximum);
}
```

Console

```
2.2
19
0.003
1.000
6.0

Max: 19
```

```
#include <stdio.h>

int main(void)
{
    double maximum, input;
    int i;

    for (i = 0; i < 5; i++)
    {
        scanf(“%lf”, &input);

        if (input > maximum)
            maximum = input;
    }
    printf(“Max: %lf”, maximum);
}
```

What’s wrong with this?

What’s STILL wrong with this?

```
#include <stdio.h>

int main(void)
{
     double maximum = 0, input;
     int i;

     for (i = 0; i < 5; i++)
     {
          scanf(“%lf”, &input);

          if (input > maximum)
               maximum = input;
     }
     printf(“Max: %lf”, maximum);
}
```

```
#include <stdio.h>

int main(void)
{
    double maximum, input;
    int i;

    scanf(“%lf”, &maximum);

    for (i = 0; i < 4; i++)
    {
        scanf(“%lf”, &input);

        if (input > maximum)
            maximum = input;
    }
    printf(“Max: %lf”, maximum);
}
```

Modify to compute the minimum?

Replace with <

Write a code snippet that reads in an integer from the keyboard and adds up its digits. I.e. 1234 = 1 + 2 + 3 + 4.

Write a code snippet that reads in an integer from the keyboard and adds up its digits. I.e. **1234 = 1 + 2 + 3 + 4**.

```
int n, sum = 0, rem;

scanf(“%d”, &n);

while (n != 0)
{                      /* n = 1234        */
    rem = n % 10;      /* 1234 % 10 = 4   */
    sum = sum + rem;   /* sum = 0 + 4     */
    n = n / 10;        /* 1234 / 10 = 123 */
}

printf(“Sum of digits = %d”, sum);
```

Nested Loops

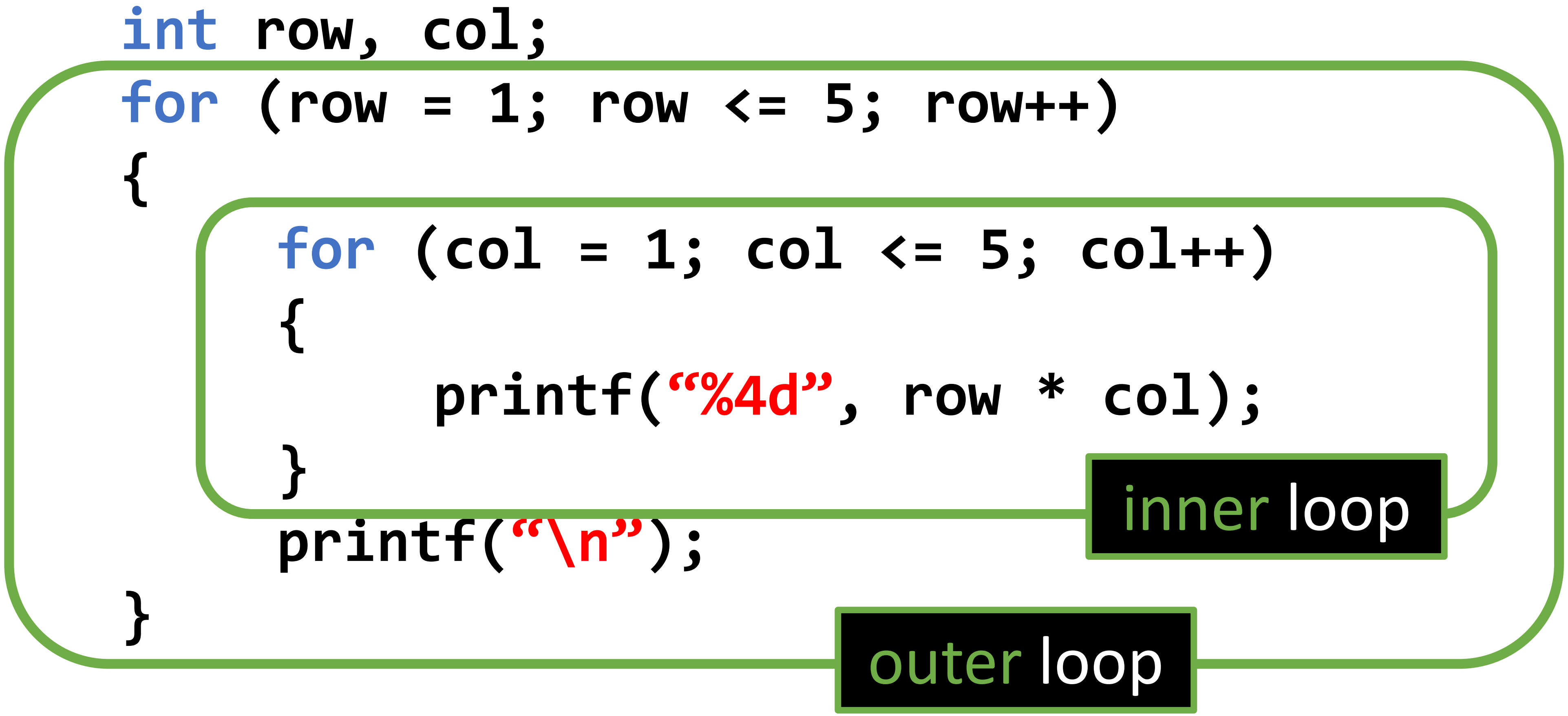
Nested Loops

A loop within a loop

The inner loop must be completely inside the outer loop

Inner and outer loops can be different loop styles.

A **`while`** loop can be inside a **`for`** loop, a **`do/while`** loop can be inside a **`while`** loop, etc.



```
int row, col;
for (row = 1; row <= 5; row++)
{
    for (col = 1; col <= 5; col++)
    {
        printf(“%4d”, row * col);
    }
    printf(“\n”);
}
```

Outer loop:

1) Execute entire inner loop
2) Print a newline
3) Repeat if row <= 5

Inner loop:

1) Print row * col
2) Repeat if col <= 5

```
int row, col;
for (row = 1; row <= 5; row++) {
    for (col = 1; col <= 5; col++) {
        printf(“%4d”, row * col);
    }
    printf(“\n”);
}
```

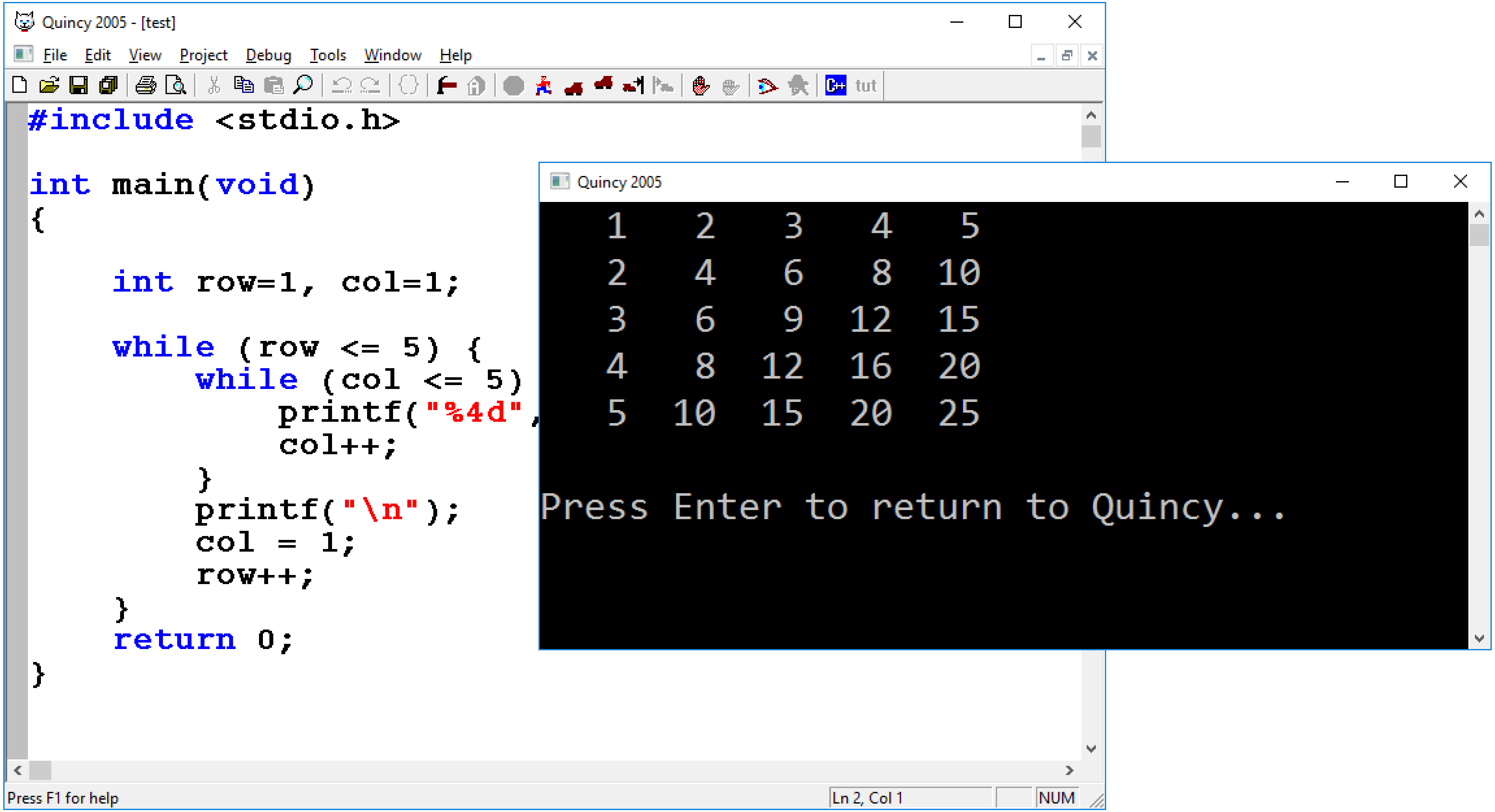
Output:

```
1    2    3    4    5      /* row = 1 */
2    4    6    8    10     /* row = 2 */
3    6    9    12   15     /* row = 3 */
4    8    12   16   20     /* row = 4 */
5    10   15   20   25     /* row = 5 */
```

Using while?

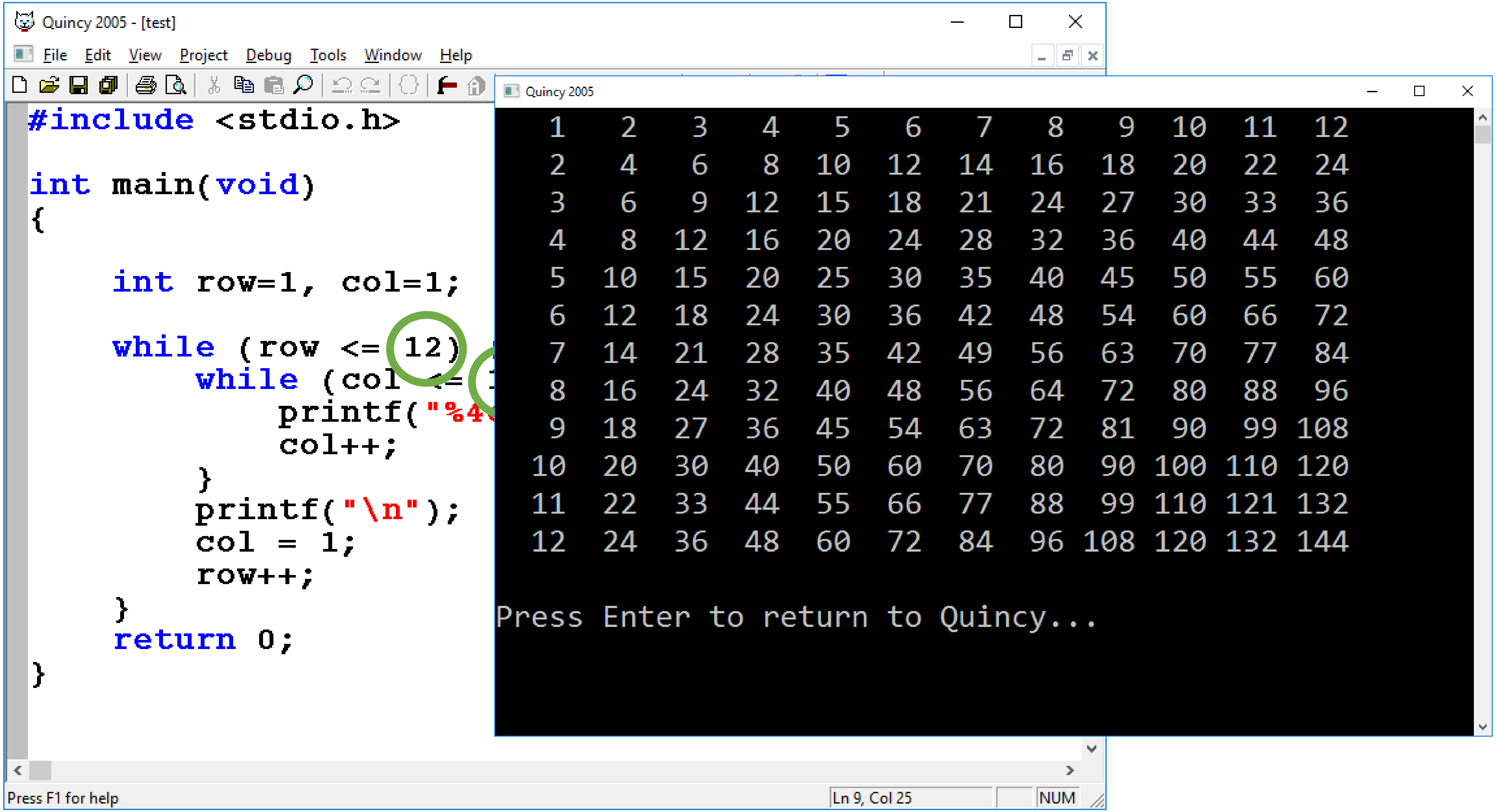
```
int row = 1, col = 1;     /* initialize      */
while (row <= 5) {        /* outer condition */
    while (col <= 5)  {   /* inner condition */
        printf(“%4d”, row * col);
        col++;            /* inner increment */
    }
    printf(“\n”);
    row++;          /* outer increment  */
    col = 1;        /* inner initialize */
}
```

Why do we re-initialize inner loop?



```c
#include <stdio.h>

int main(void)
{

    int row=1, col=1;

    while (row <= 5) {
        while (col <= 5)
            printf("%4d",
            col++;
        }
        printf("\n");
        col = 1;
        row++;
    }
    return 0;
}
```

```
   1   2   3   4   5
   2   4   6   8  10
   3   6   9  12  15
   4   8  12  16  20
   5  10  15  20  25

Press Enter to return to Quincy...
```



```
#include <stdio.h>

int main(void)
{

    int row=1, col=1;

    while (row <= 12)
        while (col <= 1
            printf("%4
            col++;
        }
        printf("\n");
        col = 1;
        row++;
    }
    return 0;
}
```

```
   1   2   3   4   5   6   7   8   9  10  11  12
   2   4   6   8  10  12  14  16  18  20  22  24
   3   6   9  12  15  18  21  24  27  30  33  36
   4   8  12  16  20  24  28  32  36  40  44  48
   5  10  15  20  25  30  35  40  45  50  55  60
   6  12  18  24  30  36  42  48  54  60  66  72
   7  14  21  28  35  42  49  56  63  70  77  84
   8  16  24  32  40  48  56  64  72  80  88  96
   9  18  27  36  45  54  63  72  81  90  99 108
  10  20  30  40  50  60  70  80  90 100 110 120
  11  22  33  44  55  66  77  88  99 110 121 132
  12  24  36  48  60  72  84  96 108 120 132 144

Press Enter to return to Quincy...
```

Write a code snippet to do the following:

Print the multiplication table starting at 5 and ending at 15.
Use nested **for** loops.

```
int row, col;
for (row = 5; row <= 15; row++)
{
    for (col = 5; col <= 15; col++)
    {
        printf(“%4d”, row * col);
    }
    printf(“\n”);
}
```

Inner and outer loops can affect each other!

```
int i, j;
for (i = 1; i <= 10; i++)
{
    for (j = 1; j <= 10; j++)
    {
        printf(“%d ”, i*j);
        i++; !!!
    }
    printf(“\ni = %d”, i);
}
```

inner loop

outer loop

Output:

```
1 4 9 16 25 36 49 64 81 100
i = 11
```

```
int i, j;
for (i = 1; i <= 4; i++)
{
    for (j = 1; j <= i; j++)
    {
        printf(“*”);
    }
    printf(“\n”);
}
```

inner loop

outer loop

Output:

```
*         /* i = 1, j = 1 to 1 */
**        /* i = 2, j = 1 to 2 */
***       /* i = 3, j = 1 to 3 */
****      /* i = 4, j = 1 to 4 */
```

break
vs.
continue

Remember break?

```
int n = 1;
while (1) {      /* Always true! */
    printf(“%d ”, n++);
    if (n > 5)
        break;   /* break exits the loop */
}
```

Console:

```
1 2 3 4 5
```

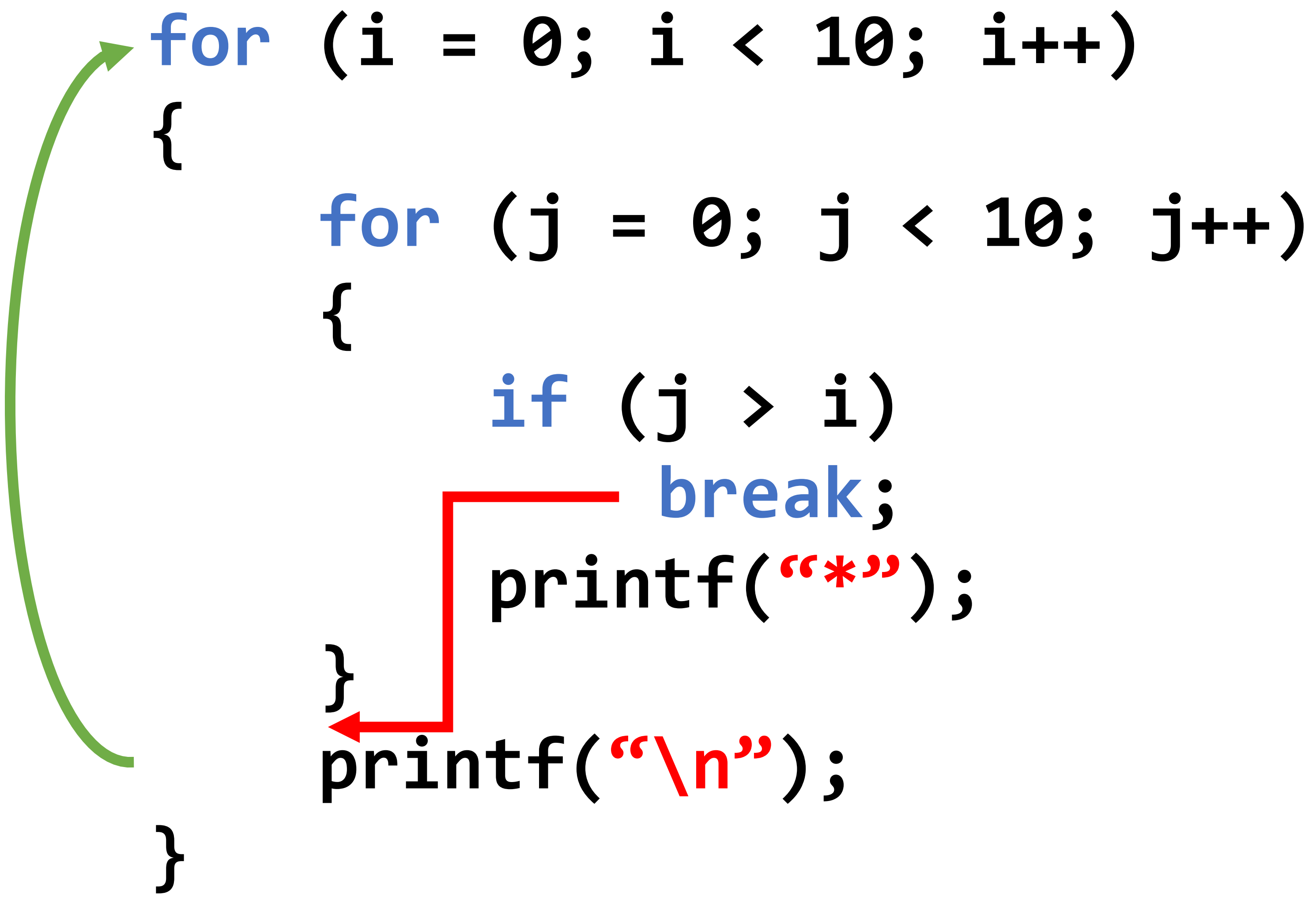
A break statement jumps out of the *innermost* enclosing loop of the statement

```
int i = 1;
while (1)   /* Condition is always true */
{
    if (i >= 6)     /* if + break is used
        break;         to exit the loop  */
    printf(“%d ”, i);
    i = i + 1;
}
```

Output:

```
1 2 3 4 5
```

A break statement jumps out of the *innermost* enclosing loop of the statement



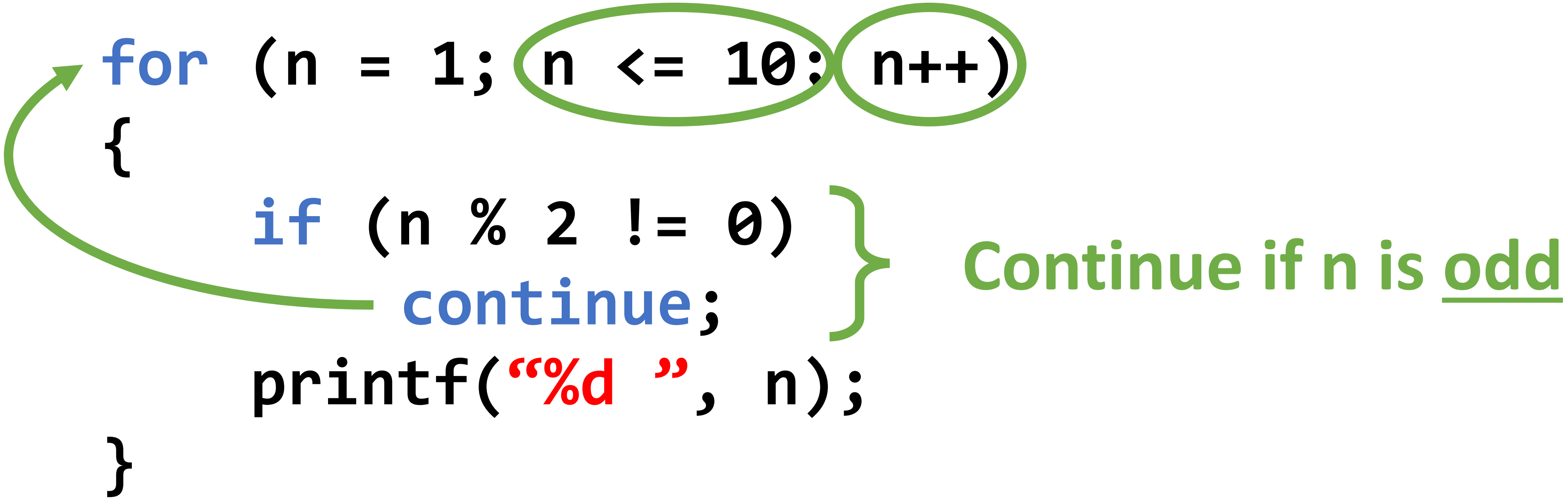
```
for (i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        if (j > i)
            break;
        printf(“*”);
    }
    printf(“\n”);
}
```

`break` VS `continue`

The `continue` statement ends the *current iteration* of a loop

Contrast with the `break` statement, which exits the loop entirely.

The **continue** statement skips the *current iteration* of a loop



```
for (n = 1; n <= 10; n++)
{
    if (n % 2 != 0)
      continue;
    printf("%d ", n);
}
```

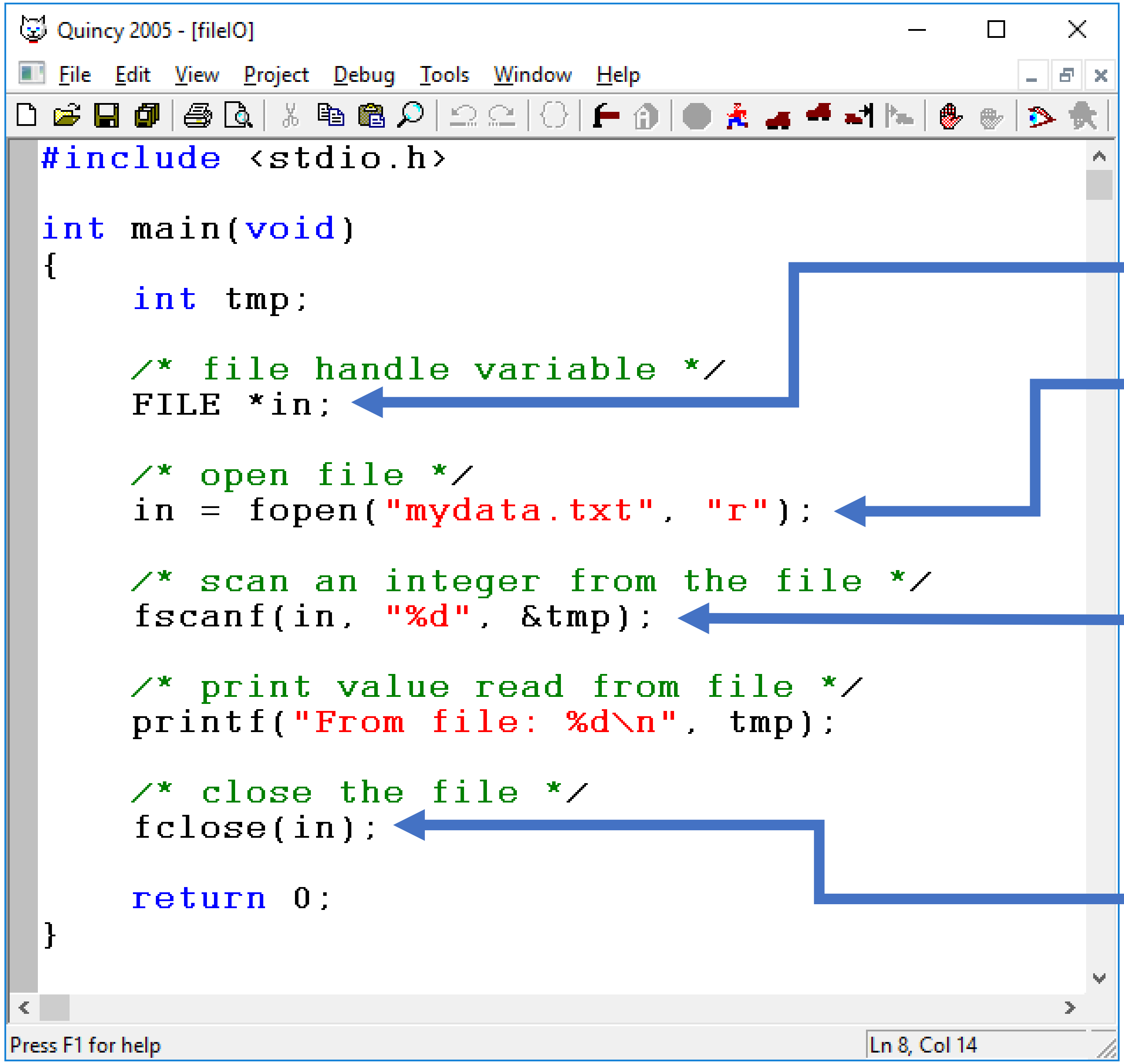
Continue if n is odd

Output:

```
2 4 6 8 10
```

Loops + File I/O

fscanf



```
#include <stdio.h>

int main(void)
{
    int tmp;

    /* file handle variable */
    FILE *in;

    /* open file */
    in = fopen("mydata.txt", "r");

    /* scan an integer from the file */
    fscanf(in, "%d", &tmp);

    /* print value read from file */
    printf("From file: %d\n", tmp);

    /* close the file */
    fclose(in);

    return 0;
}
```

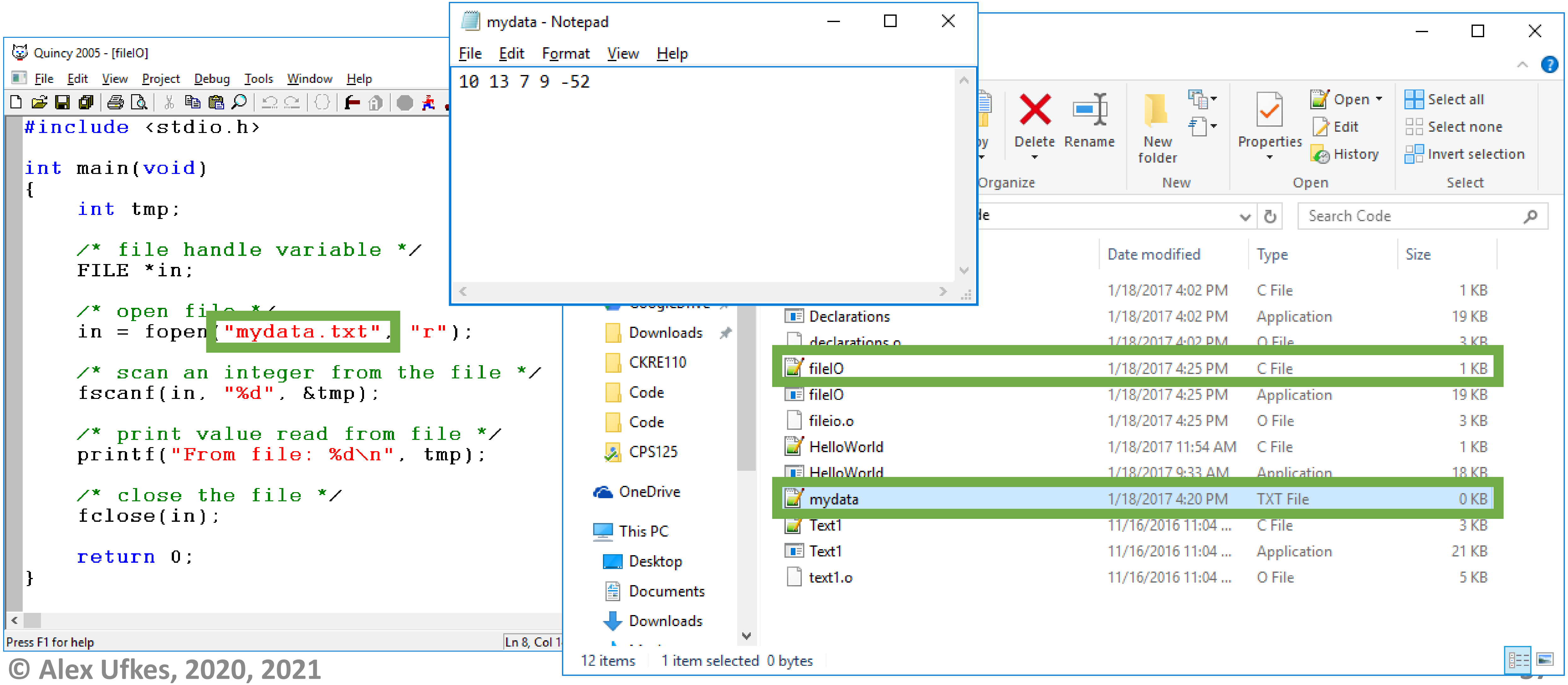
New variable type: FILE *

fopen to open the file. First argument is the filename, second is the mode. **"r"** means ***read***.

fscanf - Like scanf but takes an argument before the string: the file handle variable.

fclose – File should be closed once we're done with it.

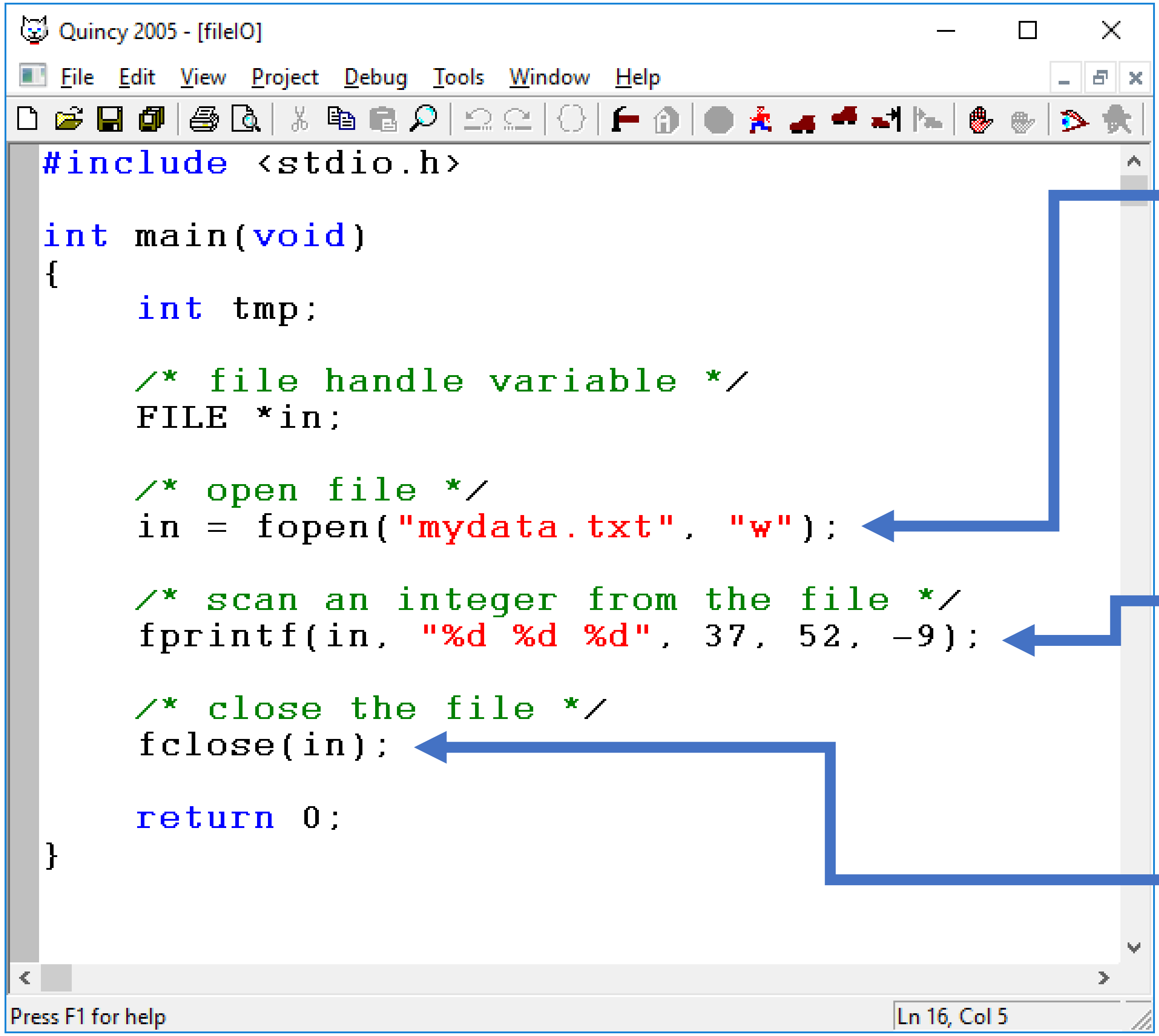
Practicalities



```c
#include <stdio.h>

int main(void)
{
    int tmp;

    /* file handle variable */
    FILE *in;

    /* open file */
    in = fopen("mydata.txt", "r");

    /* scan an integer from the file */
    fscanf(in, "%d", &tmp);

    /* print value read from file */
    printf("From file: %d\n", tmp);

    /* close the file */
    fclose(in);

    return 0;
}
```

```
10 13 7 9 -52
```

	Date modified	Type	Size
	1/18/2017 4:02 PM	C File	1 KB
Declarations	1/18/2017 4:02 PM	Application	19 KB
declarations.o	1/18/2017 4:02 PM	O File	3 KB
fileIO	1/18/2017 4:25 PM	C File	1 KB
fileIO	1/18/2017 4:25 PM	Application	19 KB
fileio.o	1/18/2017 4:25 PM	O File	3 KB
HelloWorld	1/18/2017 11:54 AM	C File	1 KB
HelloWorld	1/18/2017 9:33 AM	Application	18 KB
mydata	1/18/2017 4:20 PM	TXT File	0 KB
Text1	11/16/2016 11:04 ...	C File	3 KB
Text1	11/16/2016 11:04 ...	Application	21 KB
text1.o	11/16/2016 11:04 ...	O File	5 KB



```
#include <stdio.h>

int main(void)
{
    int tmp;

    /* file handle variable */
    FILE *in;

    /* open file */
    in = fopen("mydata.txt", "w");

    /* scan an integer from the file */
    fprintf(in, "%d %d %d", 37, 52, -9);

    /* close the file */
    fclose(in);

    return 0;
}
```

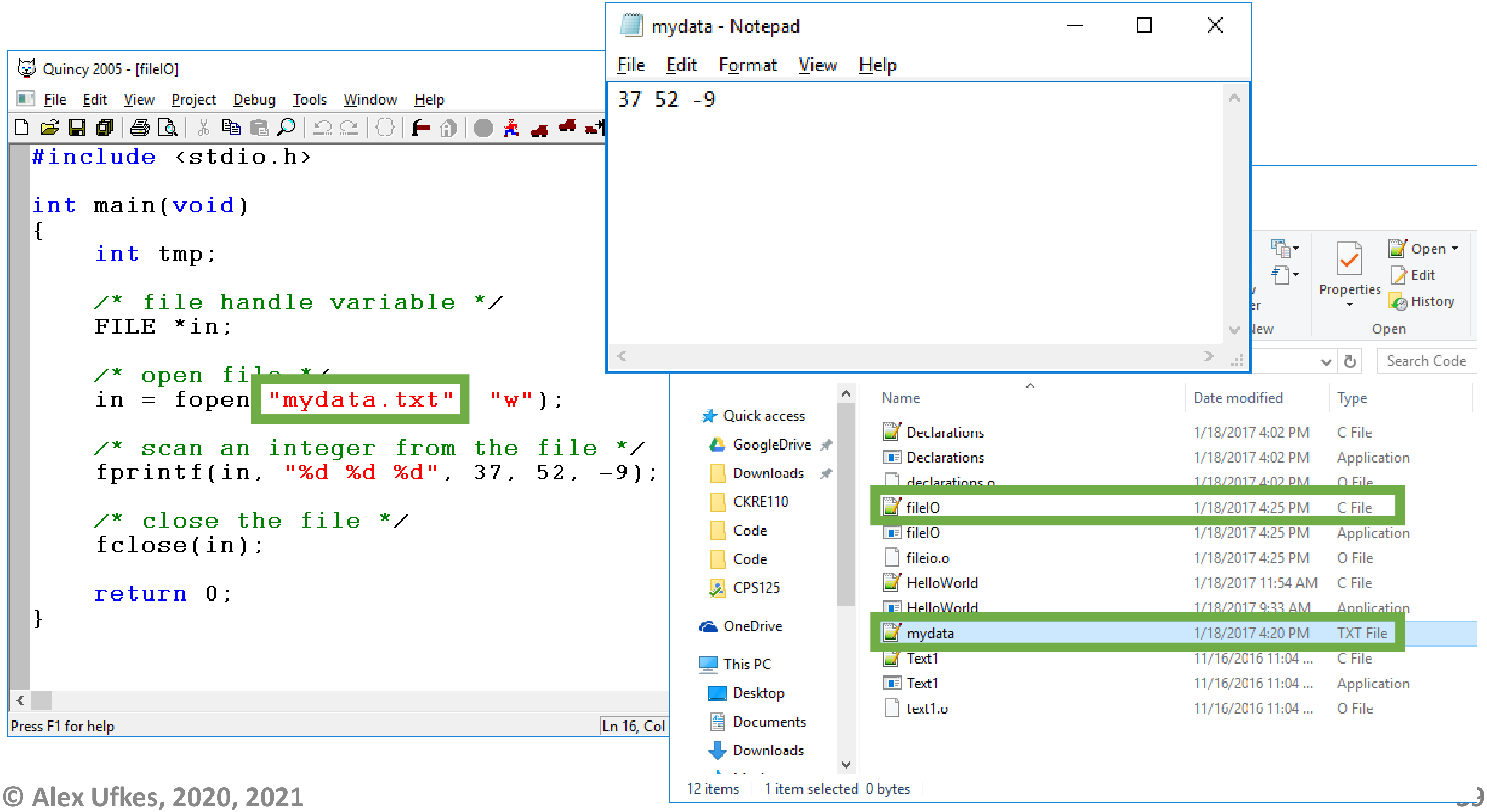
fprintf

Must first open the file:

- This time we're ***writing***. **"w"** means write.
- ***If the file already exists, it will be overwritten!***

fprintf - Like printf but takes an argument before the string: the file handle variable.

fclose – File should be closed once we're done with it.



```
#include <stdio.h>

int main(void)
{
    int tmp;

    /* file handle variable */
    FILE *in;

    /* open file */
    in = fopen("mydata.txt" "w");

    /* scan an integer from the file */
    fprintf(in, "%d %d %d", 37, 52, -9);

    /* close the file */
    fclose(in);

    return 0;
}
```

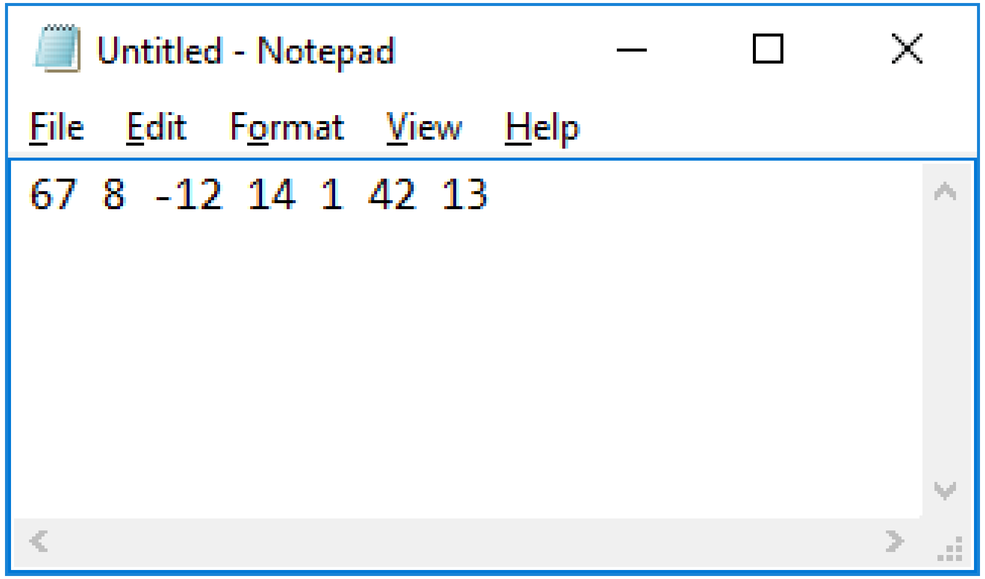
```
37 52 -9
```

Name	Date modified	Type
Declarations	1/18/2017 4:02 PM	C File
Declarations	1/18/2017 4:02 PM	Application
declarations.o	1/18/2017 4:02 PM	O File
fileIO	1/18/2017 4:25 PM	C File
fileIO	1/18/2017 4:25 PM	Application
fileio.o	1/18/2017 4:25 PM	O File
HelloWorld	1/18/2017 11:54 AM	C File
HelloWorld	1/18/2017 9:33 AM	Application
mydata	1/18/2017 4:20 PM	TXT File
Text1	11/16/2016 11:04 ...	C File
Text1	11/16/2016 11:04 ...	Application
text1.o	11/16/2016 11:04 ...	O File

12 items 1 item selected 0 bytes

EOF-Controlled Loop

Loop that reads values from a file and stops when there are no more values left



```
67 8 -12 14 1 42 13
```

<u>The Problem:</u>
We don’t know ahead of time how many values are in the file.

scanf Revisited

```
#include <stdio.h>
int main (void)
{
        int x, z1, z2;
        x = scanf(“%d%d”, &z1, &z2);
        printf(“%d”, x);    /* 2 */
}
```

Output of scanf is the number of values *successfully* read.

What About fscanf?

```
#include <stdio.h>
int main (void)
{
     int x, z1, z2;
     FILE *input = fopen(“data.txt”, “r”);
     x = fscanf(input, “%d%d”, &z1, &z2);
     printf(“%d”, x);
}
```

Output of fscanf is the number of values *successfully* read.

Detecting EOF (End of File)

File goes into a ***fail state*** when attempting to read data beyond the end of file (EOF)

```
int x, status;
FILE *input = fopen(“data.txt”, “r”);
status = fscanf(input, “%d”, &x);
```

What value of **status** corresponds to being at EOF?

EOF is a #define Macro

In stdio.h:

```
    #define EOF (-1)
```

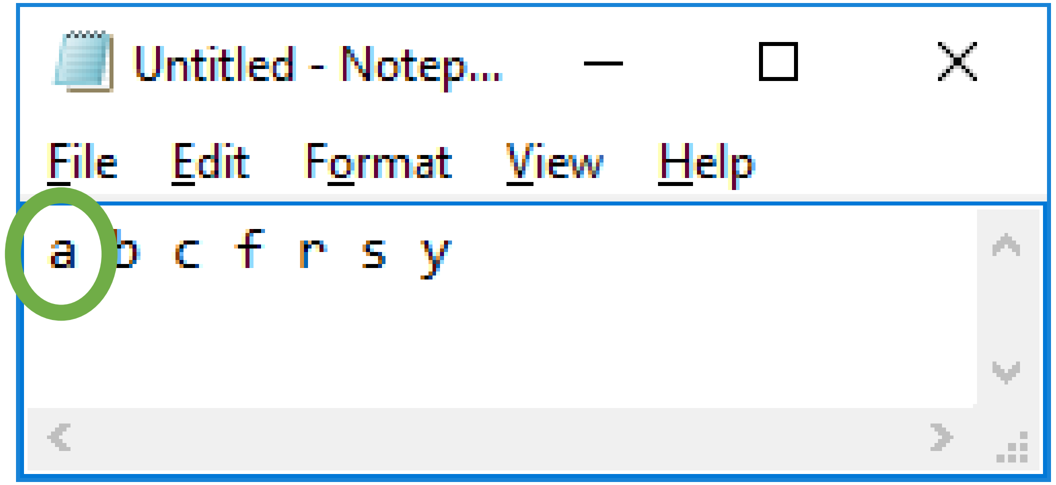
Why -1 and not 0?

Just because we fail to read any values successfully, doesn't necessarily mean we are at the end of the file.

Try it!

```
printf("EOF = %d\n", EOF);
```

```
int status;
double x;
FILE *f = fopen(“data.txt”, “r”);
status = fscanf(f, “%lf”, &x);
printf(“Status: %d\n”, status);
```



```
a b c f r s y
```

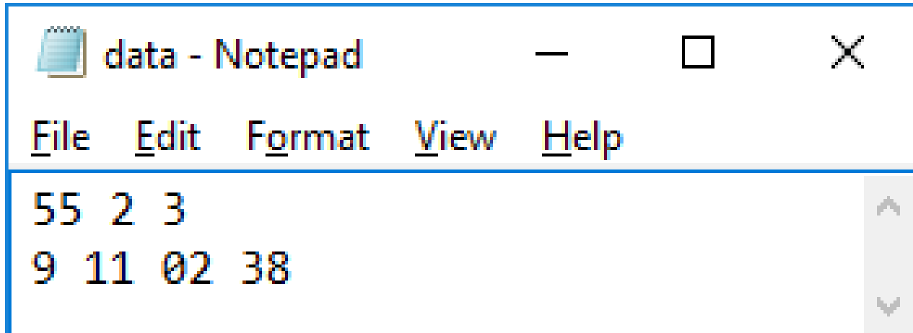
Output:

```
Status: 0
```

Write a code snippet to do the following:

Read all the integers from a file and compute their sum.

```
int number, sum = 0, status;

FILE *in = fopen(“data.txt”, “r”);      /* Open file         */
status = fscanf(in, “%d”, &number);     /* scan first number */

while (status != EOF) {                 /* Check for EOF     */
  sum = sum + number;                   /* Add number to sum */
  status = fscanf(in, “%d”, &number);   /* scan next number  */
}

printf(“Sum: %d\n”, sum);
```

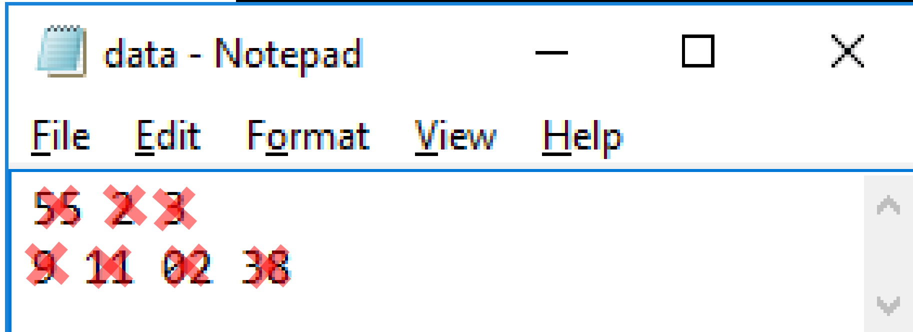


```
55 2 3
9 11 02 38
```

```
int number, sum = 0, status;

FILE *in = fopen(“data.txt”, “r”);
status = fscanf(in, “%d”, &number);
printf(“status: %d\n”, status);

while (status != EOF) {
   sum = sum + number;
   status = fscanf(in, “%d”, &number);
   printf(“status: %d\n”, status);
}
printf(“Sum: %d\n”, sum);
```

Console:

```
status: 1
status: 1
status: 1
status: 1
status: 1
status: 1
status: 1
status: -1
Sum: 120
```



~~55~~ ~~2~~ ~~3~~
~~9~~ ~~11~~ ~~02~~ ~~38~~

A Cleaner Way

```
int number, sum = 0;

FILE *in = fopen(“data.txt”, “r”);

while (fscanf(in, “%d”, &number) != EOF) {
  sum = sum + number;
}

printf(“Sum: %d\n”, sum);
```

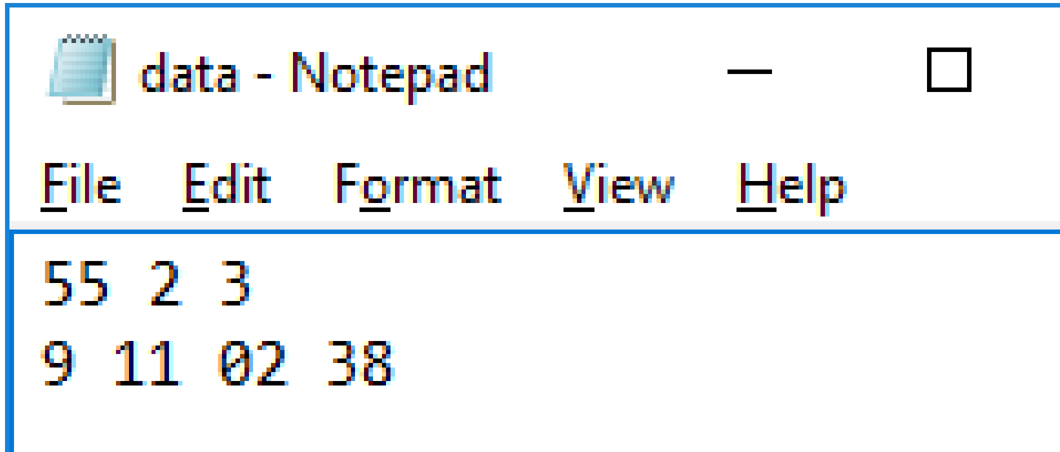
The `fscanf` call can go directly inside the `while` condition!

However! The output of `fscanf` is no longer being saved.

Write a code snippet to do the following:

Find the maximum number in a file.

```
int num, max = INT_MIN;   /*  #include <limits.h>  */

FILE *in = fopen("data.txt", "r");

while (fscanf(in, "%d", &num) != EOF) {
    if (num > max) {
        max = num;
    }
}
printf("Max: %d\n", max);
```



```
55 2 3
9 11 02 38
```

Common Mistake

```
int num, status;

FILE *in = fopen(“data.txt”, “r”);

status = fscanf(in, “%d”, &num);
while (num != EOF) {
    printf(“num: %d\n”, num);
    status = fscanf(in, “%d”, &num);
}
```

BAD!

The value of **num** has ***nothing*** to do with **EOF**.

It’s the value of **status** that we want to check.

`feof()`

Another way to check for **EOF**

`feof` is a function that checks the status of a file.

It returns a non-zero value (`true`) if EOF has been reached. Otherwise, it returns zero (`false`)

feof()

```
int number, sum = 0;

FILE *in = fopen(“data.txt”, “r”);

while (!feof(in)) {
    fscanf(in, “%d”, &number);
    sum = sum + number;
}

printf(“Sum: %d\n”, sum);
```

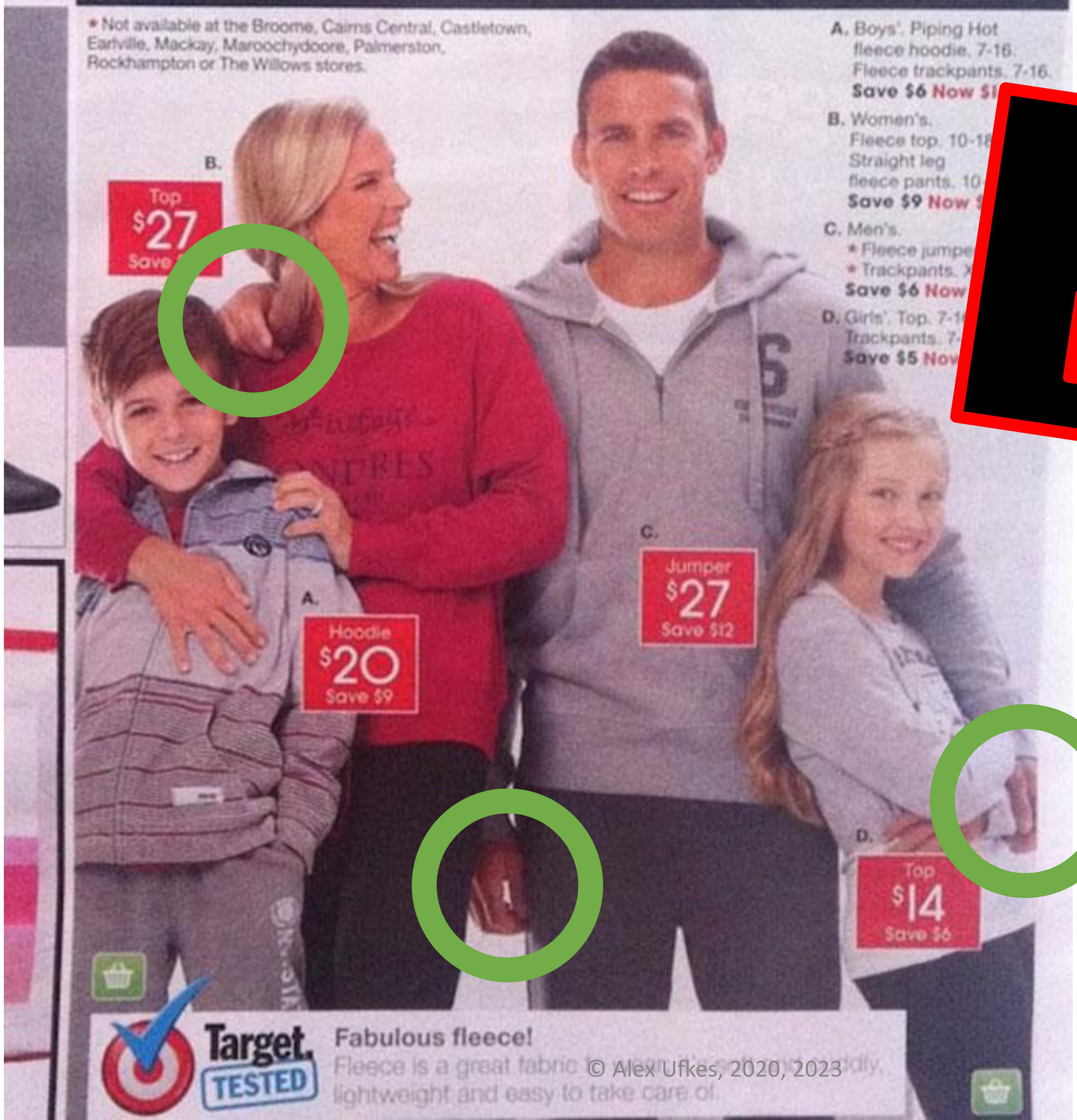
Keep looping as long as we are ***not*** at the end of file `in`

Don’t Forget!

```
FILE *in = fopen(“data.txt”, “r”);
…
fclose(in);
```

Always close your files!

Spot the ERROR?



* Not available at the Broome, Cairns Central, Castletown, Earlville, Mackay, Maroochydoore, Palmerston, Rockhampton or The Willows stores.

A. Boys'. Piping Hot fleece hoodie. 7-16. Fleece trackpants. 7-16.
Save $6 Now $1

B. Women's.
Fleece top. 10-18
Straight leg
fleece pants. 10-
Save $9 Now $

C. Men's.
* Fleece jumpe
* Trackpants. X
Save $6 Now

D. Girls'. Top. 7-1
Trackpants. 7-
Save $5 Now

Fabulous fleece!
Fleece is a great fabric
lightweight and easy to take care of.

```
#include <stdio.h>

int main(void)
{
    int n, x = -3;
    for (n = x, n <= 0, n += 1)
        printf(“%d\n”, n);

    return 0;
}
```

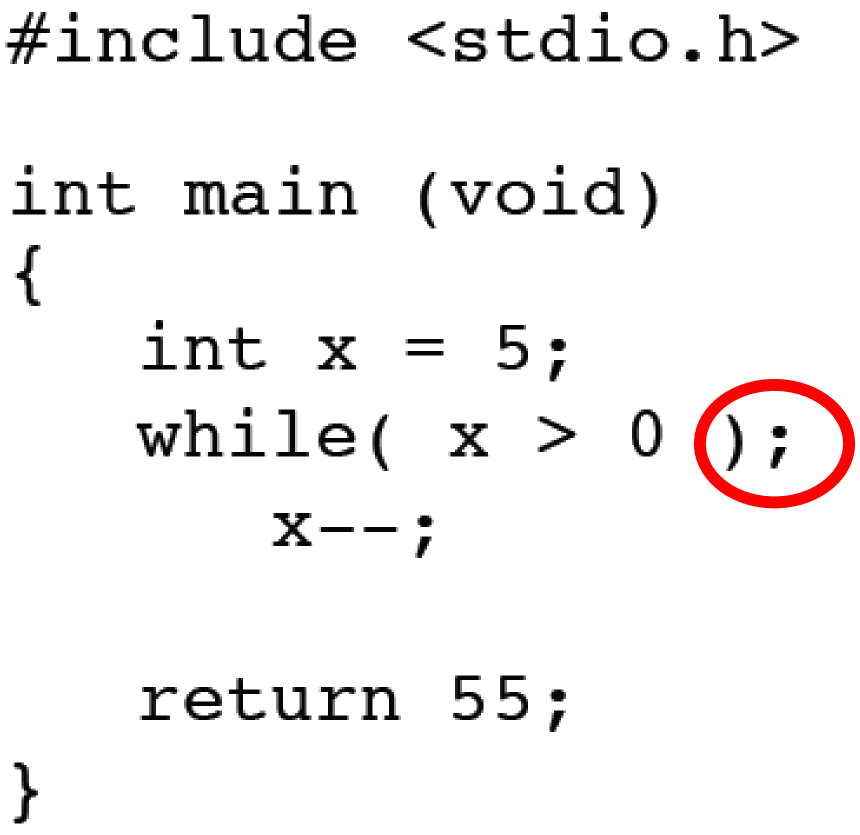
Should be semicolons, not commas

✗

```c
#include <stdio.h>

int main(void)
{
     int n;
     for (n = 1; n != 50; n *= 2) {
          printf(“%d\n”, n);
     }

     return 0;
}
```

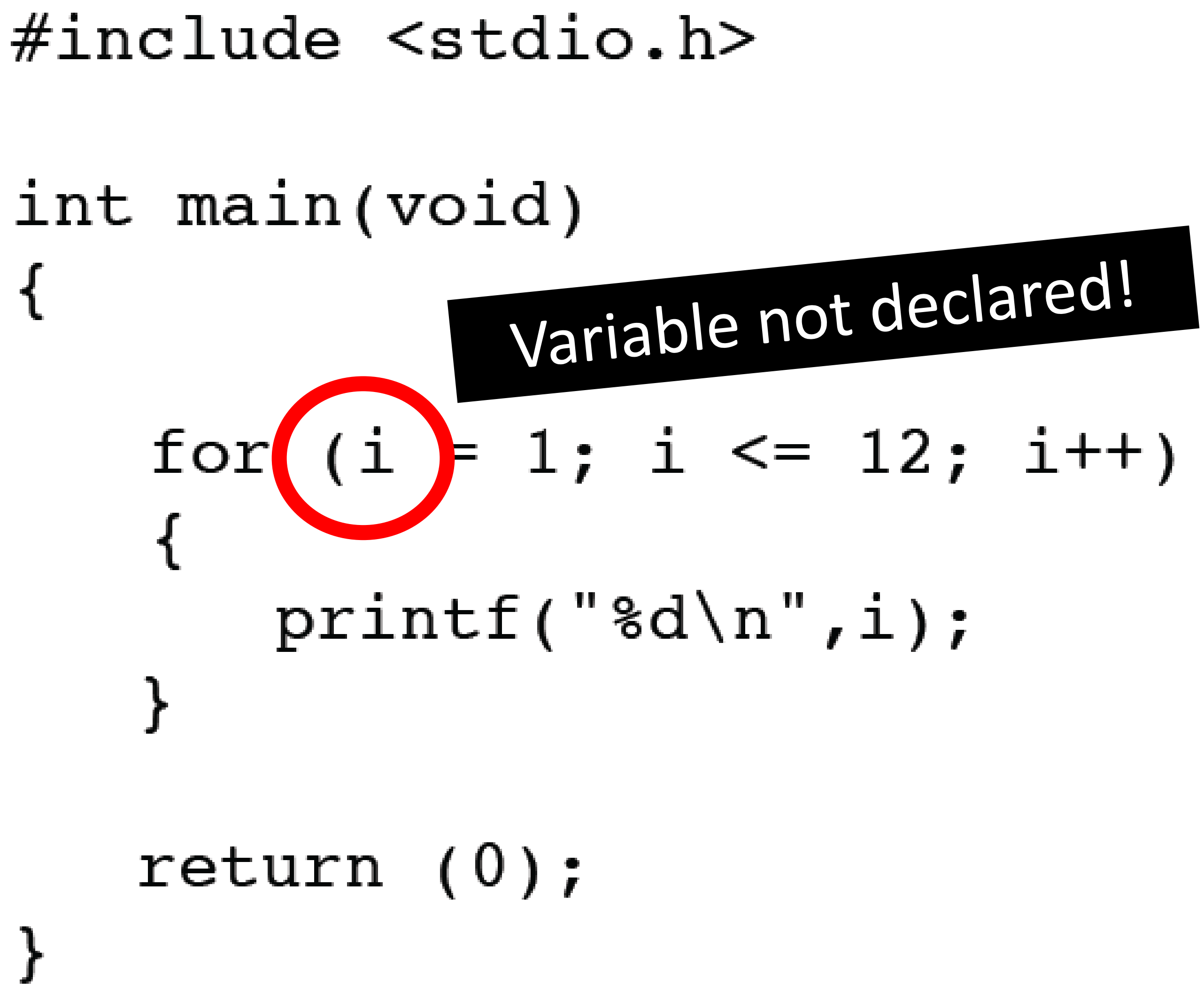
∞

✗



```
#include <stdio.h>

int main (void)
{
    int x = 5;
    while( x > 0 );
        x--;

    return 55;
}
```

✗



```
#include <stdio.h>

int main(void)
{

    for (i = 1; i <= 12; i++)
    {
        printf("%d\n",i);
    }

    return (0);
}
```

Variable not declared!

```c
#include <stdio.h>

int main(void)
{
    int n;

    do
    {
        printf("Enter a number ");
        printf("between 1 and 9: ");
        scanf ("%d", &n);
    }while (n < 1 || n > 9)

 return (0);
}
```

Questions?