جامعة بيرزيت

BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER SCIENCE DEPARTMENT

COMP1310
Introduction to Computer and Computing Ethics

REPETITION AND LOOP STATEMENTS

Loops

- We use loops to indicate a part of the code that needs to be repeated a certain number of times.

- Types of loops in C:

1. while loops
2. for loops
3. do-while loops

Controlling Loops

- Loops can be controlled in several ways:

1. Counter controlled loops: loops that count up or down until a specified value of the counter is reached.
2. Event controlled loops: loops that stop when a specific event happens, such as the input reaching 0, or another specific value.
3. Result controlled loops: loops that stop when a test determines that the loop reached the desired result, such as a numerical approximation.

The *while* Loop

```
initialization
while(condition) {
      statement(s)
      update statement
}
```

The *while* Loop

```
initialization
while(condition) {
      statement(s)
      update statement
}
```

we need initial values to test the condition

The *while* Loop

```
initialization
while(condition) {
      statement(s)
      update statement
}
```

we need initial values to test the condition

as long as the condition is true, the compiler will go into the loop

The *while* Loop

```
initialization
while(condition) {
      statement(s)
      update statement
}
```

initialization: we need initial values to test the condition

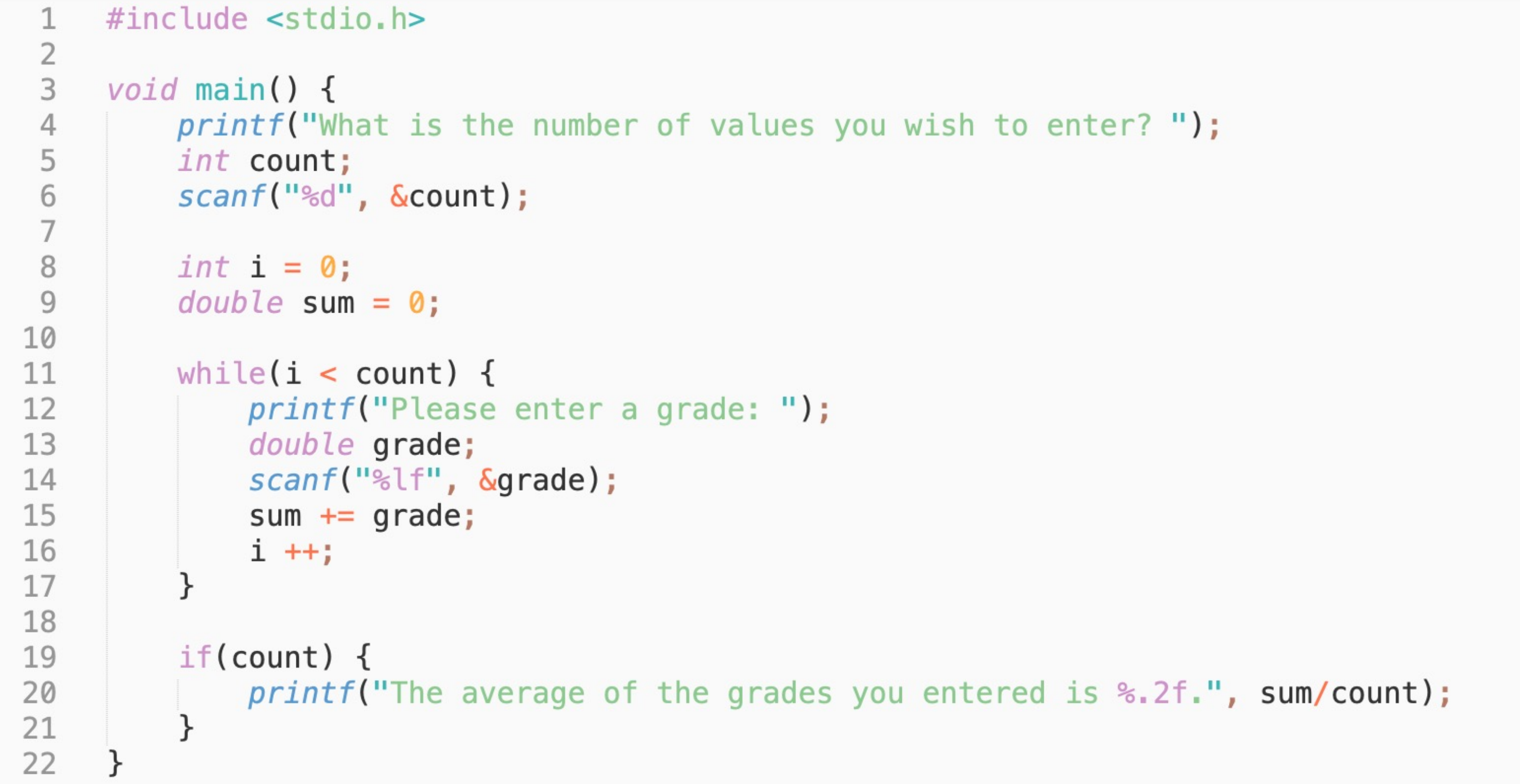
while(condition) {: as long as the condition is true, the compiler will go into the loop

update statement: an update statement is necessary to allow the condition to change at some point
if there is no update statement, the loop will go on indefinitely

The *while* loop – Counter Controlled

- Let’s write a program that calculates the average of students’ grades.
- The program will ask the user for the number of students they have in class.
- We can use this number as a counter to know how many loops we need to make.
- When we reach the number that the user entered, we can stop.

The *while* loop – Counter Controlled

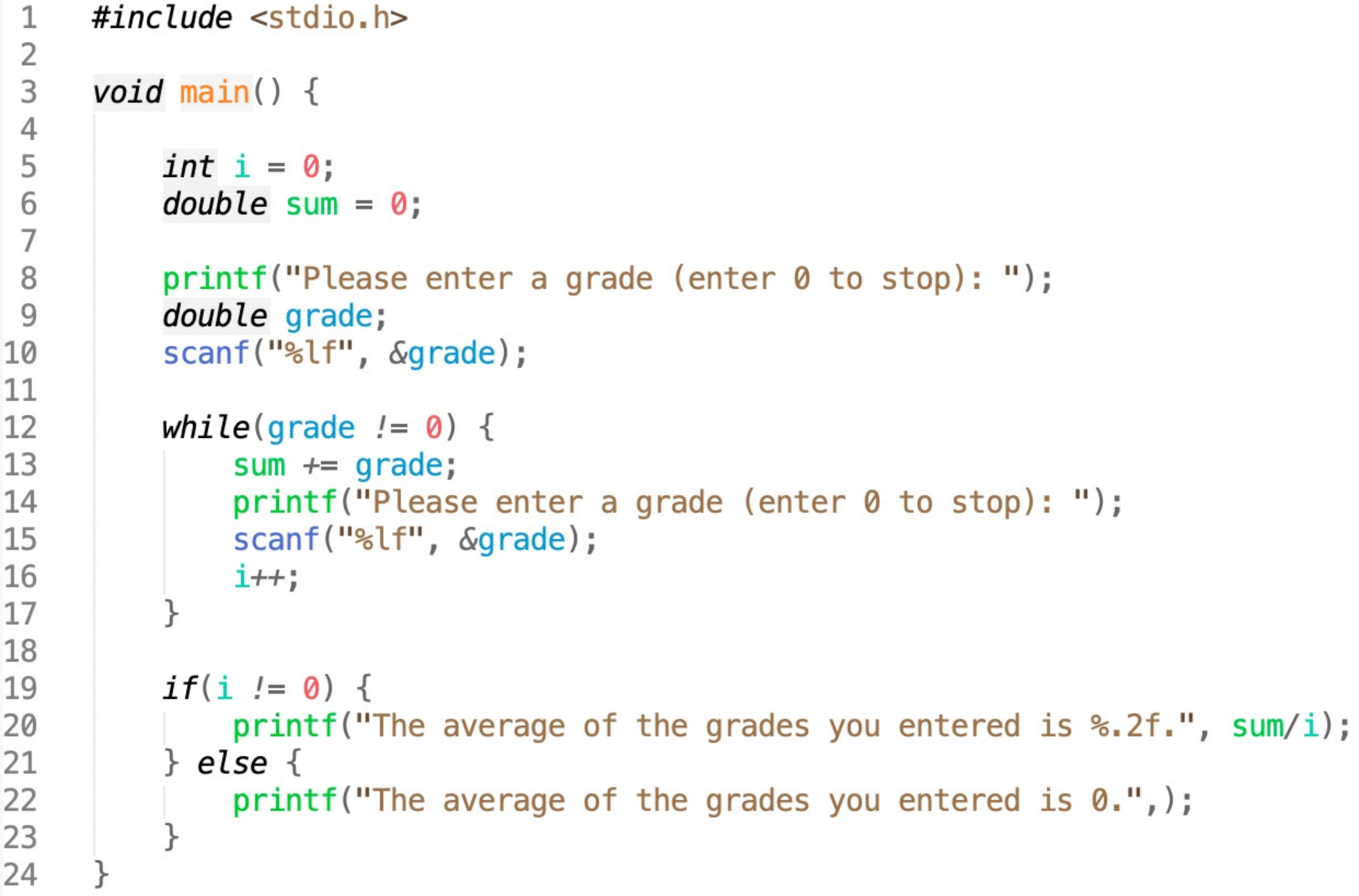


```c
#include <stdio.h>

void main() {
    printf("What is the number of values you wish to enter? ");
    int count;
    scanf("%d", &count);

    int i = 0;
    double sum = 0;

    while(i < count) {
        printf("Please enter a grade: ");
        double grade;
        scanf("%lf", &grade);
        sum += grade;
        i ++;
    }

    if(count) {
        printf("The average of the grades you entered is %.2f.", sum/count);
    }
}
```

The *while* Loop – Event Controlled

- Let’s write a program that calculates the average of students’ grades.
- This time, we will assume that the user does not initially know how many students they have in class.
- We will ask the user to enter the value 0 when they wish to exit the loop.
- This method assumes that there will be no 0 grades in the entered values.
- If we know that 0 is a possible grade, we need to pick a different condition to stop, such as -1.

The *while* Loop – Event Controlled



```c
#include <stdio.h>

void main() {

    int i = 0;
    double sum = 0;

    printf("Please enter a grade (enter 0 to stop): ");
    double grade;
    scanf("%lf", &grade);

    while(grade != 0) {
        sum += grade;
        printf("Please enter a grade (enter 0 to stop): ");
        scanf("%lf", &grade);
        i++;
    }

    if(i != 0) {
        printf("The average of the grades you entered is %.2f.", sum/i);
    } else {
        printf("The average of the grades you entered is 0.",);
    }
}
```

The *do-while* Loop

- The do-while loops differs form the regular while loop in that we *do* the statements initially, then we check the *while* condition to decide if we want to continue or exit the loop.

```
do {
        statements
} while(condition);
```

The *do-while* Loop

- The do-while loops differs form the regular while loop in that we *do* the statements initially, then we check the *while* condition to decide if we want to continue or exit the loop.

```
do {
        statements
} while(condition);
```

execute these statements first

The *do-while* Loop

- The do-while loops differs form the regular while loop in that we *do* the statements initially, then we check the *while* condition to decide if we want to continue or exit the loop.

```
do {
        statements
} while(condition);
```

execute these statements first

check if the condition is true or false:

if the condition is true, repeat the statements

if the condition is false, exit the loop

The *do-while* Loop

- The do-while loops differs form the regular while loop in that we *do* the statements initially, then we check the *while* condition to decide if we want to continue or exit the loop.

```
do {
        statements
} while(condition);
```

execute these statements first

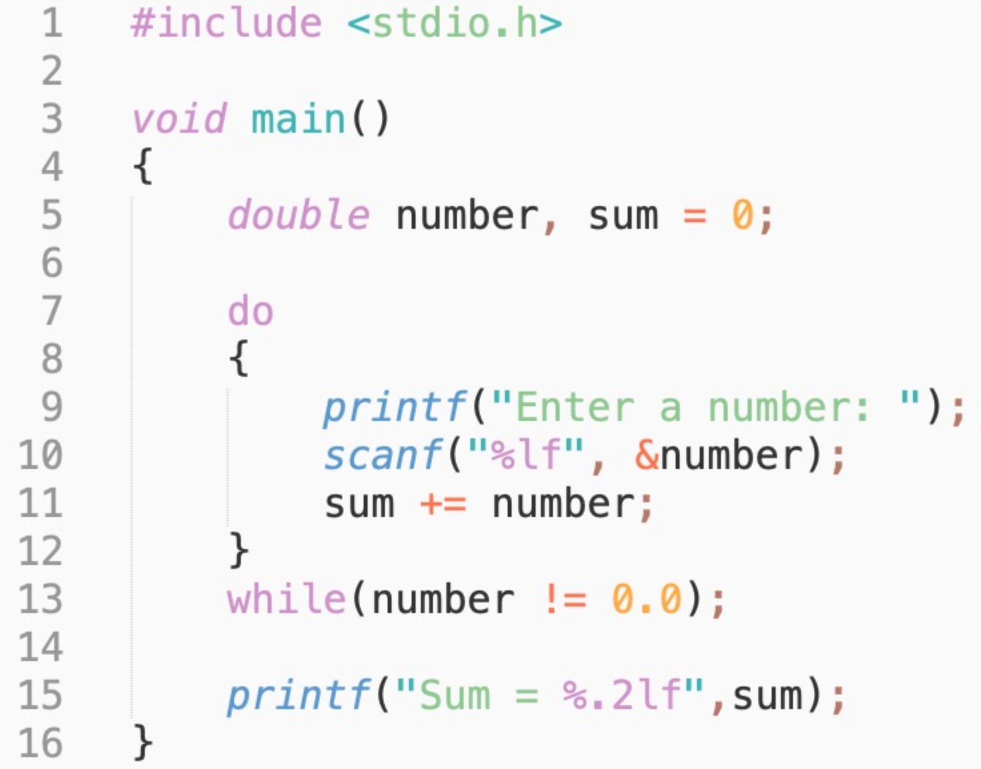
check if the condition is true or false:
if the condition is true, repeat the statements
if the condition is false, exit the loop

- *do-while* statements are less frequently used than other types of loops.

do-while Example

- Write a program that reads numbers from the user and prints out the sum of these numbers.
- Use the do-while, to make sure at least one number is read from the user.

do-while Example



```c
#include <stdio.h>

void main()
{
    double number, sum = 0;

    do
    {
        printf("Enter a number: ");
        scanf("%lf", &number);
        sum += number;
    }
    while(number != 0.0);

    printf("Sum = %.2lf",sum);
}
```

The *for* Loop

```
for (initial expression; condition; update statement) {
      statements;
}
```

The *for* Loop

```
for (initial expression; condition; update statement) {
        statements;
}
```

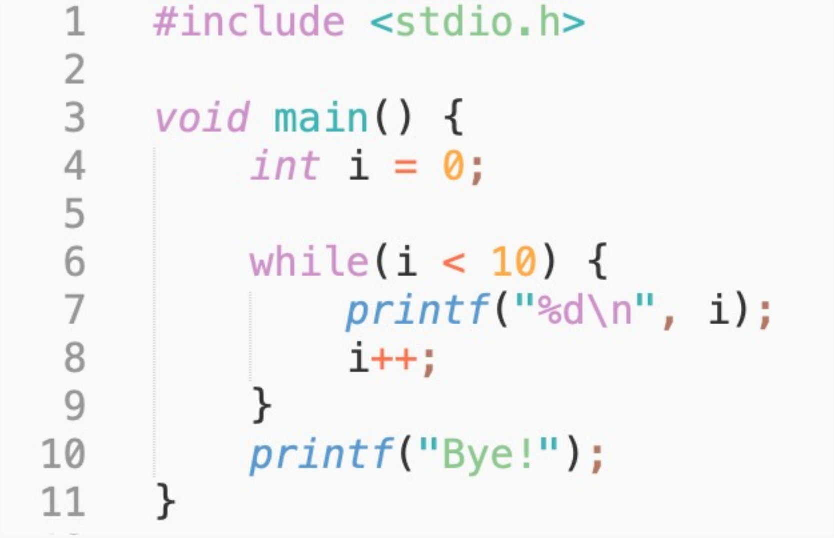
- If we omit the initial expression, all initialization must be done before the loop
- If we omit the condition, we need a break inside the loop
- If we omit the update statement, we need to update the values before the end of the loop

Loops – Example

Write a loop that prints the numbers from 0 to 9, each on a new line

Loops – Example

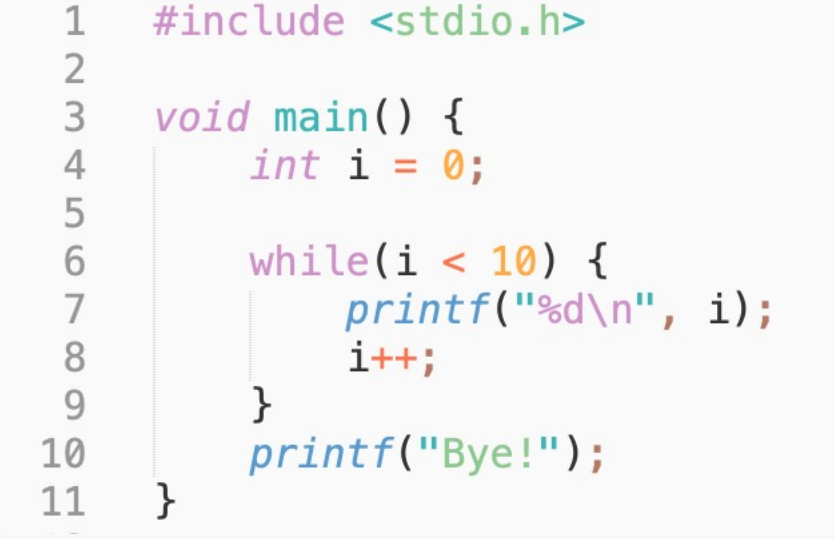
Write a loop that prints the numbers from 0 to 9, each on a new line



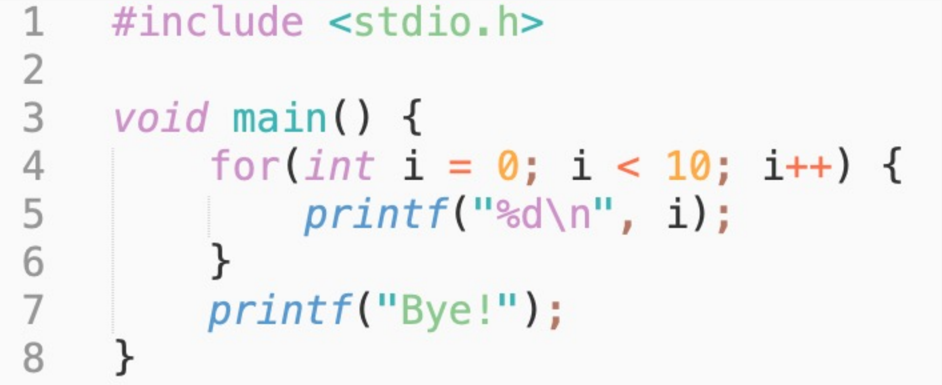
```c
#include <stdio.h>

void main() {
    int i = 0;

    while(i < 10) {
        printf("%d\n", i);
        i++;
    }
    printf("Bye!");
}
```

Loops – Example

Write a loop that prints the numbers from 0 to 9, each on a new line



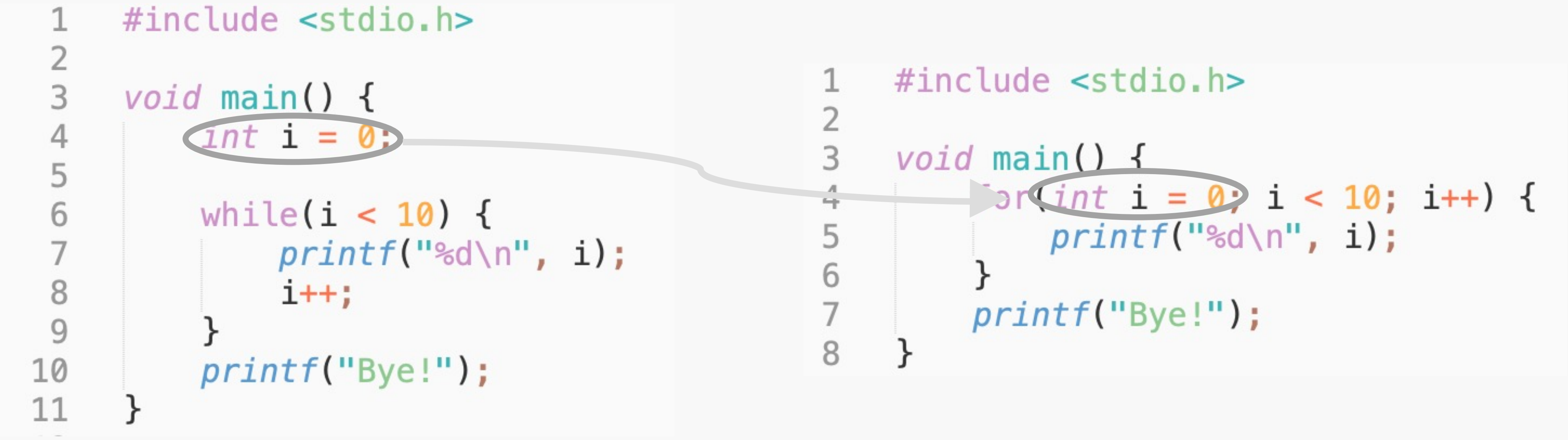
```c
#include <stdio.h>

void main() {
    int i = 0;

    while(i < 10) {
        printf("%d\n", i);
        i++;
    }
    printf("Bye!");
}
```



```c
#include <stdio.h>

void main() {
    for(int i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
    printf("Bye!");
}
```

Loops – Example

Write a loop that prints the numbers from 0 to 9, each on a new line

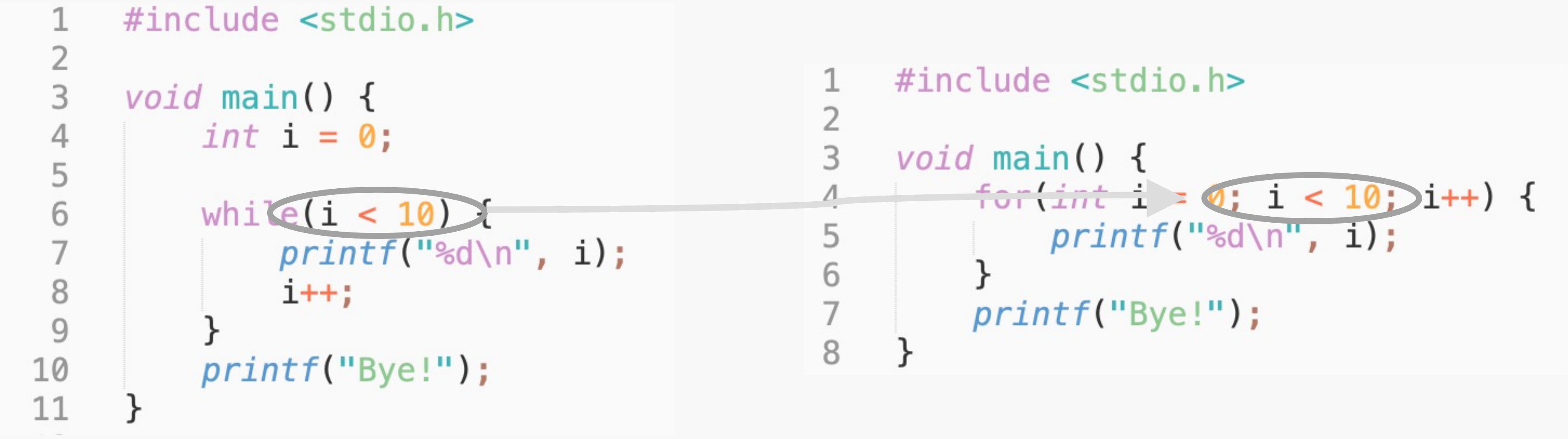


```c
#include <stdio.h>

void main() {
    int i = 0;

    while(i < 10) {
        printf("%d\n", i);
        i++;
    }
    printf("Bye!");
}
```

```c
#include <stdio.h>

void main() {
    for(int i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
    printf("Bye!");
}
```

Loops – Example

Write a loop that prints the numbers from 0 to 9, each on a new line

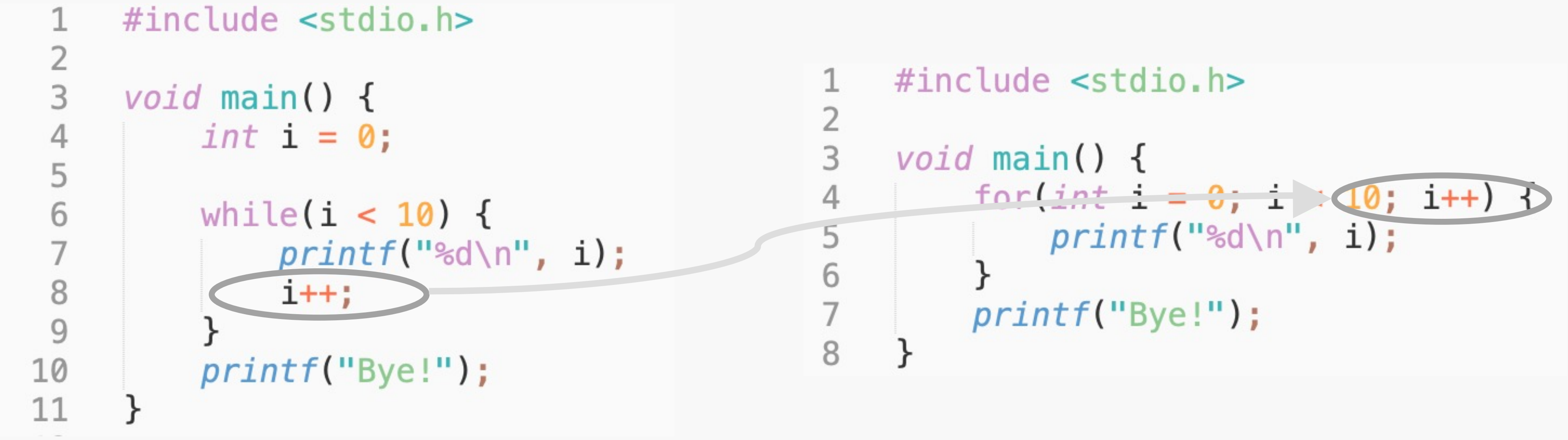


```c
#include <stdio.h>

void main() {
    int i = 0;

    while(i < 10) {
        printf("%d\n", i);
        i++;
    }
    printf("Bye!");
}
```

```c
#include <stdio.h>

void main() {
    for(int i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
    printf("Bye!");
}
```

Loops – Example

Write a loop that prints the numbers from 0 to 9, each on a new line



```c
#include <stdio.h>

void main() {
    int i = 0;

    while(i < 10) {
        printf("%d\n", i);
        i++;
    }
    printf("Bye!");
}
```

```c
#include <stdio.h>

void main() {
    for(int i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
    printf("Bye!");
}
```

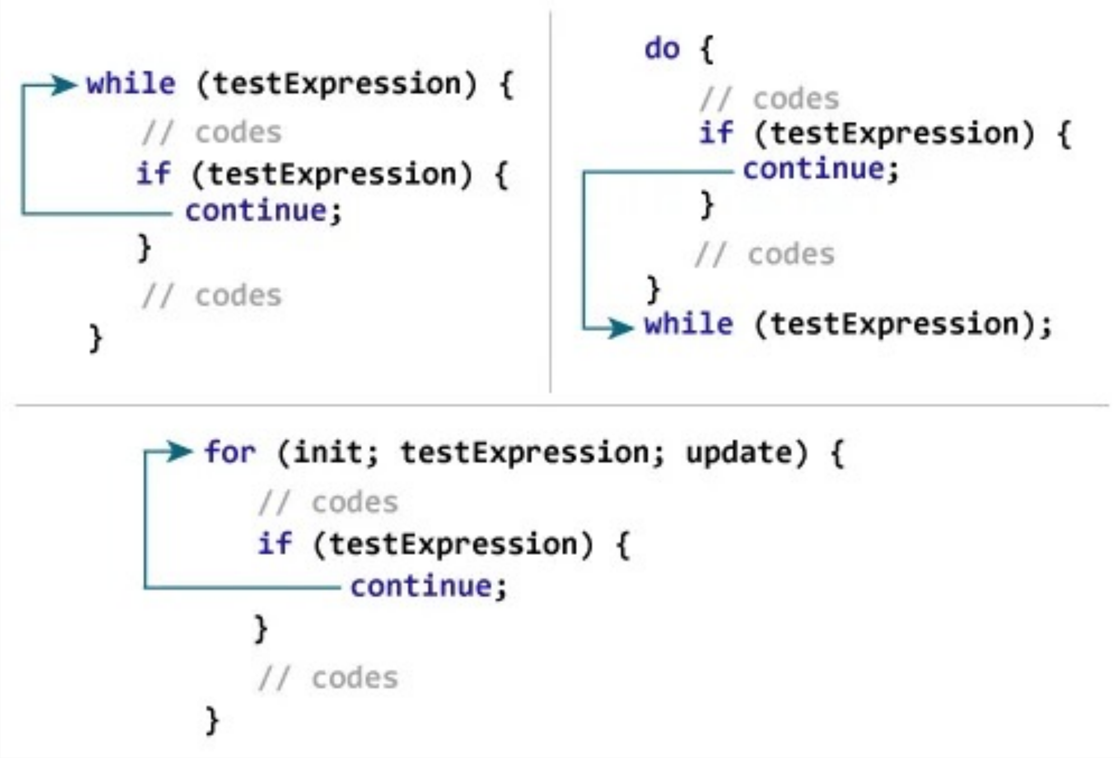
break and *continue*

- *break* and *continue* are reserved words that we need to change the flow of loop statements.
- *break* and *continue* statements are usually used with an if statement inside the loop.

- A *break* statements takes the control out of the loop. This means that if we reach a *break* statement, the next statement the compiler executes will be the one after the loop's end.
- A *continue* statement takes the control to the beginning of the loop. This means that if we reach a *continue* statement, the next statement the compiler executes will be the first statement inside the loop.

break and *continue*

Behavior of the *break* statement

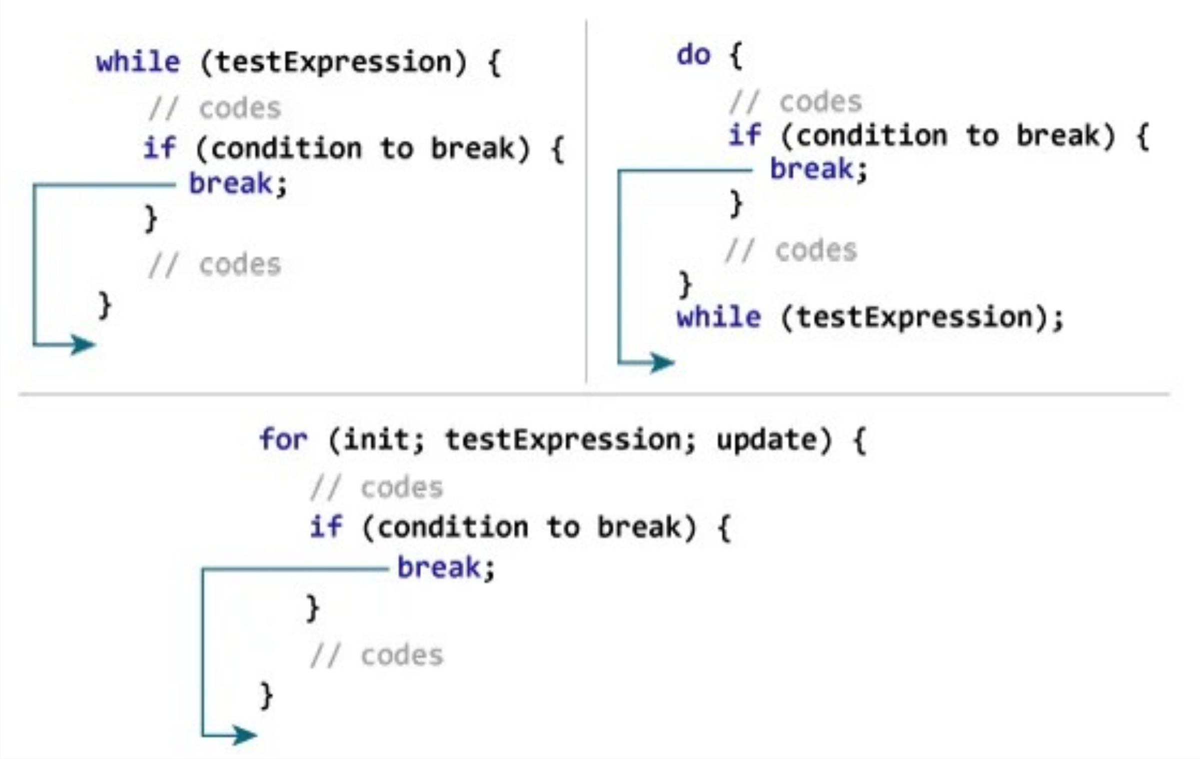


```
while (testExpression) {
   // codes
   if (testExpression) {
      continue;
   }
   // codes
}
```

```
do {
   // codes
   if (testExpression) {
      continue;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (testExpression) {
      continue;
   }
   // codes
}
```

Behavior of the *continue* statement



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

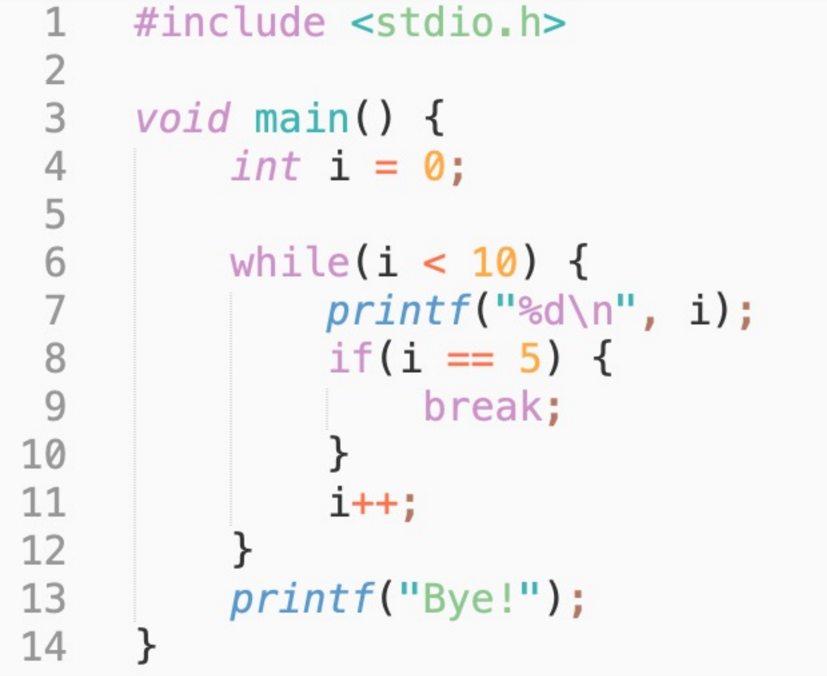
```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

Images from https://www.programiz.com/c-programming/c-break-continue-statement

break – Example

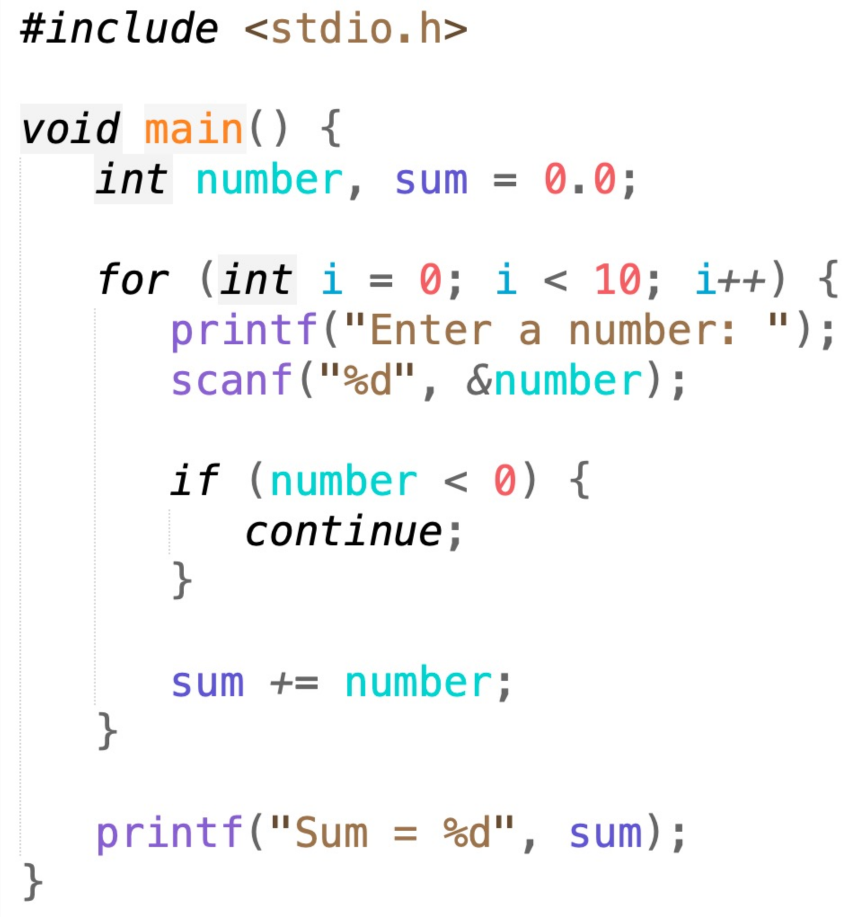
Write a loop that prints the numbers from 0 to 5



```c
#include <stdio.h>

void main() {
    int i = 0;

    while(i < 10) {
        printf("%d\n", i);
        if(i == 5) {
            break;
        }
        i++;
    }
    printf("Bye!");
}
```

continue – Example

Write a loop that reads 10 numbers from the user and sums the positive numbers only



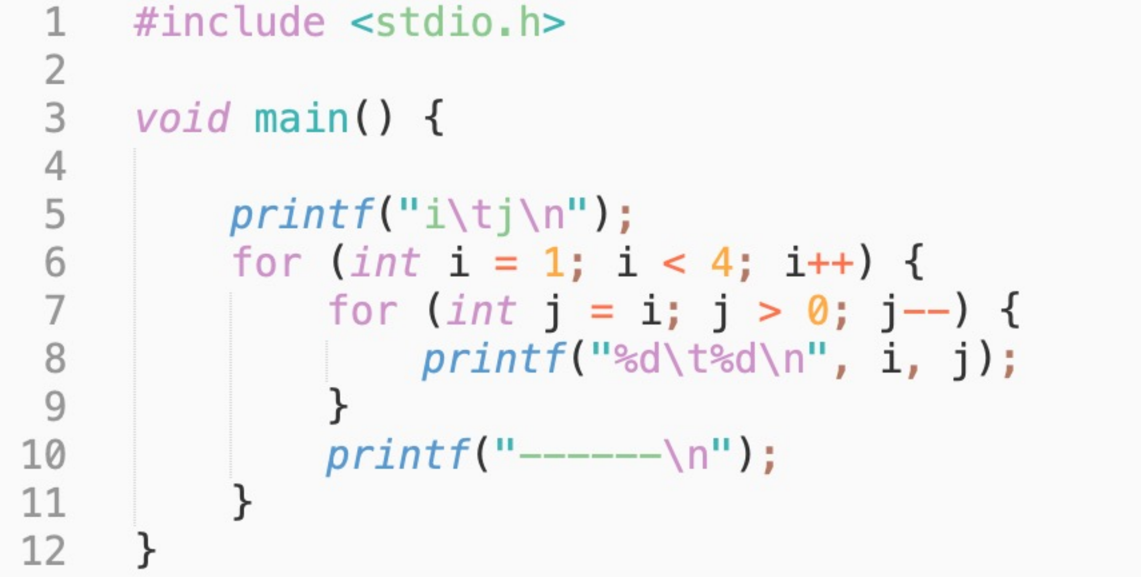
```
#include <stdio.h>

void main() {
    int number, sum = 0.0;

    for (int i = 0; i < 10; i++) {
        printf("Enter a number: ");
        scanf("%d", &number);

        if (number < 0) {
            continue;
        }

        sum += number;
    }

    printf("Sum = %d", sum);
}
```

Nested Loops

- When we have a nested loop, the inner loop needs to finish completely before moving on to the next round in the outer loop.
- Nested loops can be of the same type (e.g. two for loops) or from different types (a for loop and a while loop).
- There can be more than two nested loops within each others.
- HOWEVER, be careful that nested loops mean longer processing time.

Nested Loops

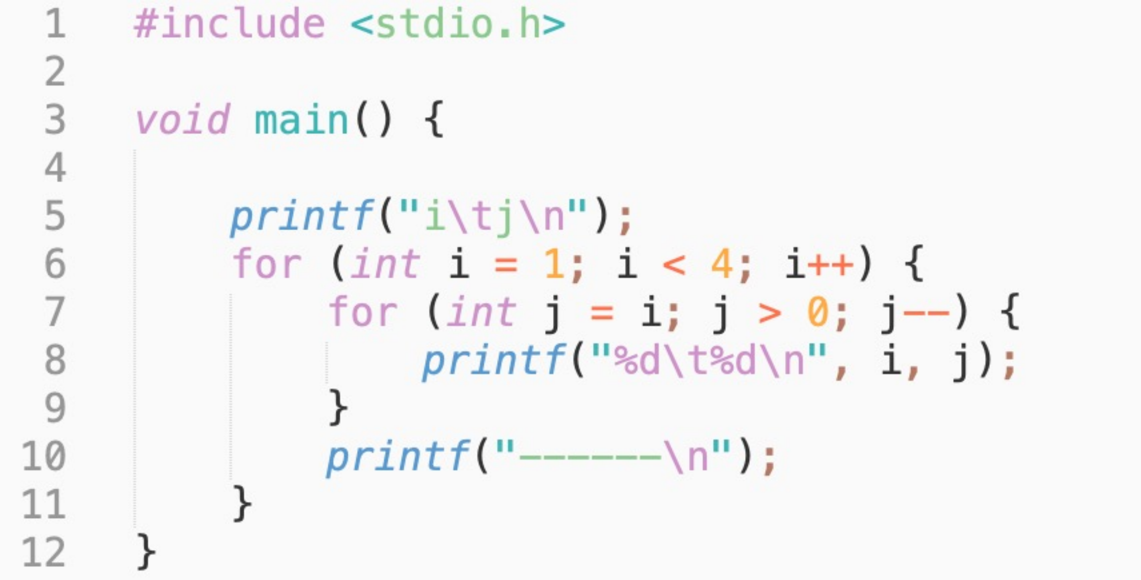
- What is the output of the following code:



```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

- What is the output of the following code:

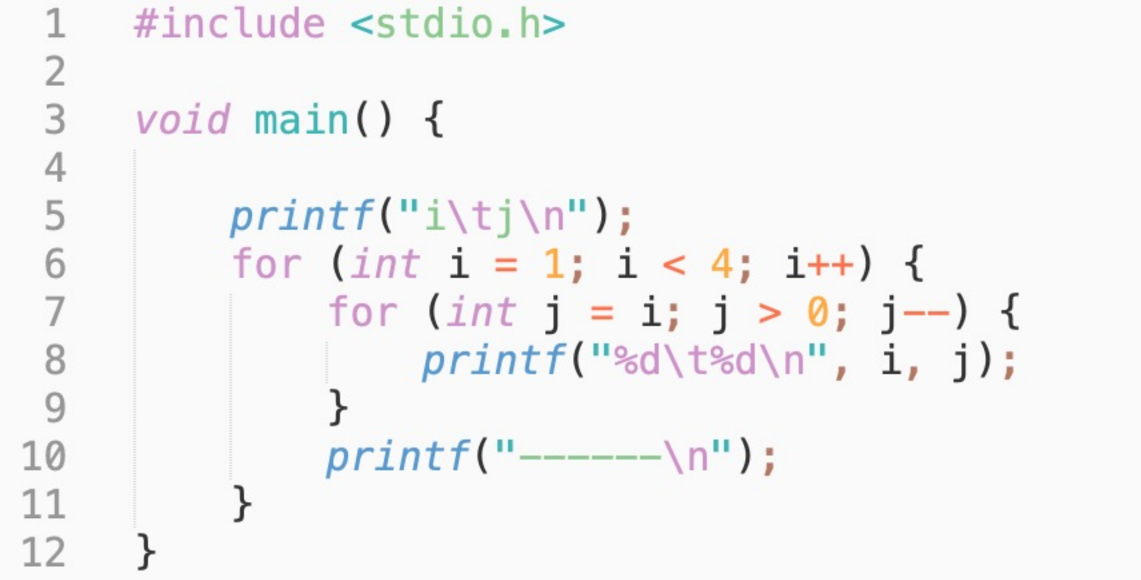


```
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action

- What is the output of the following code:

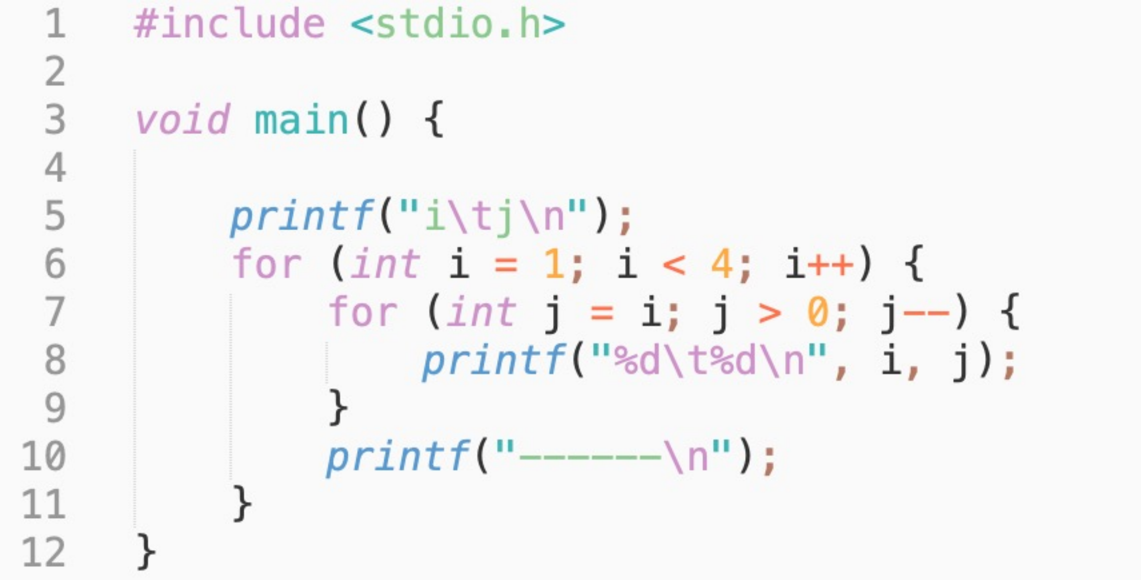


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1		

- What is the output of the following code:

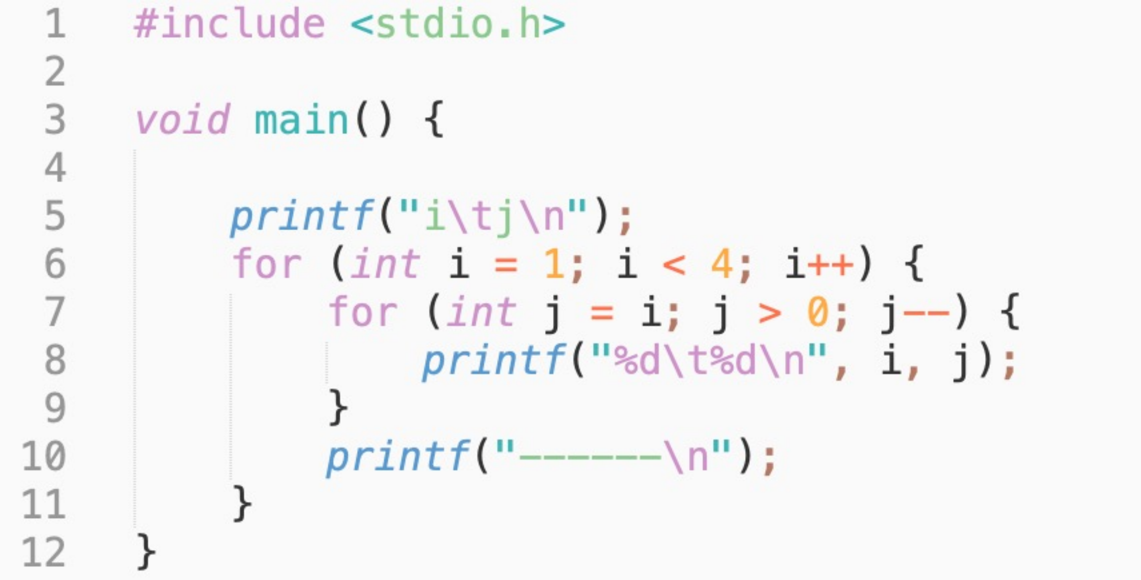


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	

- What is the output of the following code:

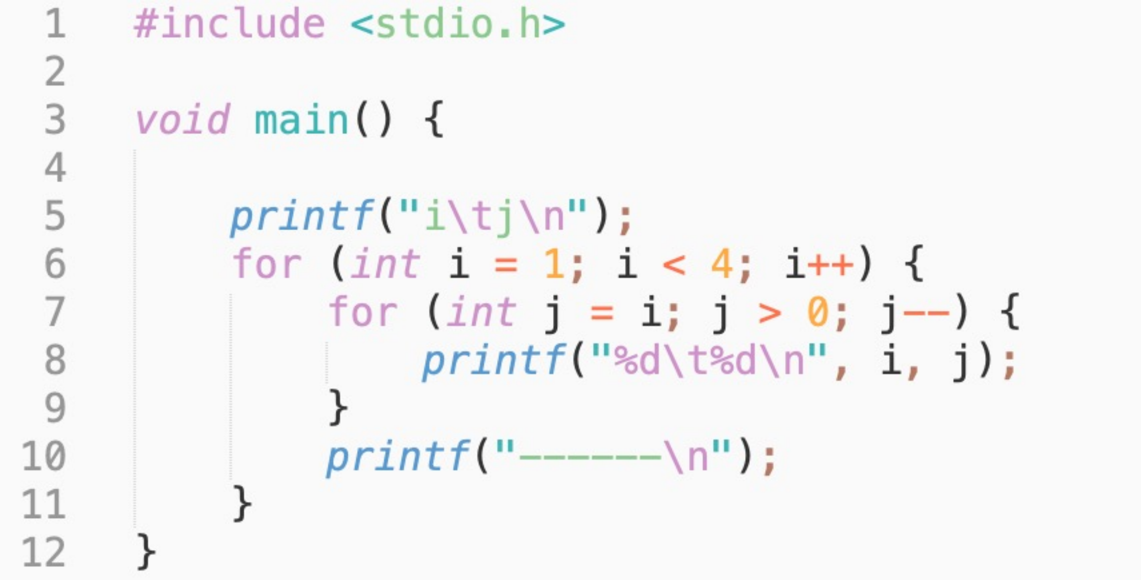


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	print (j)

- What is the following code's output:

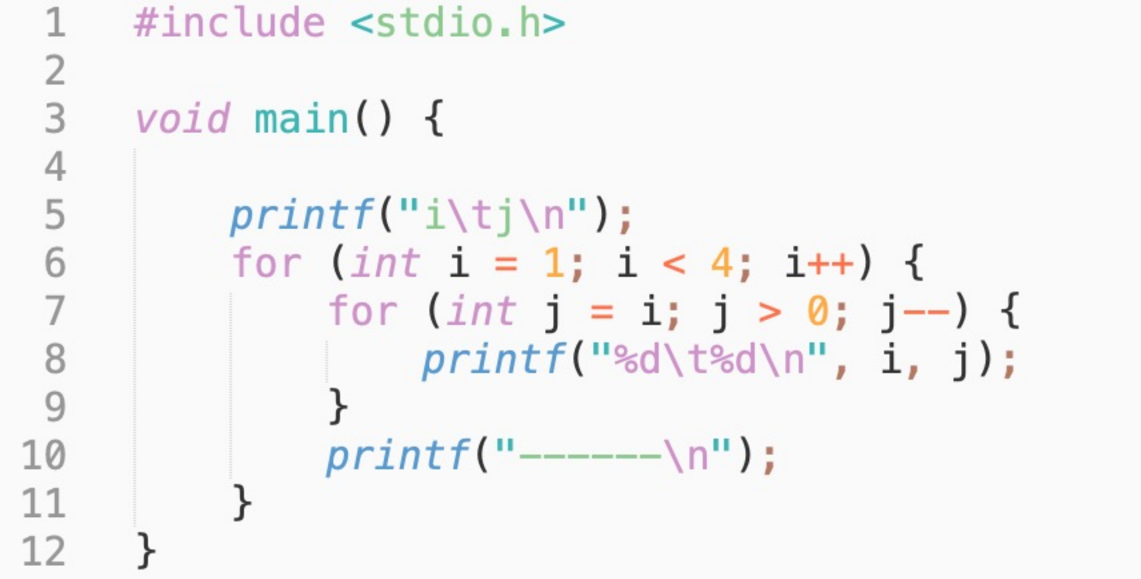


```
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	print (j)
	0	

- What is the output of the following code:

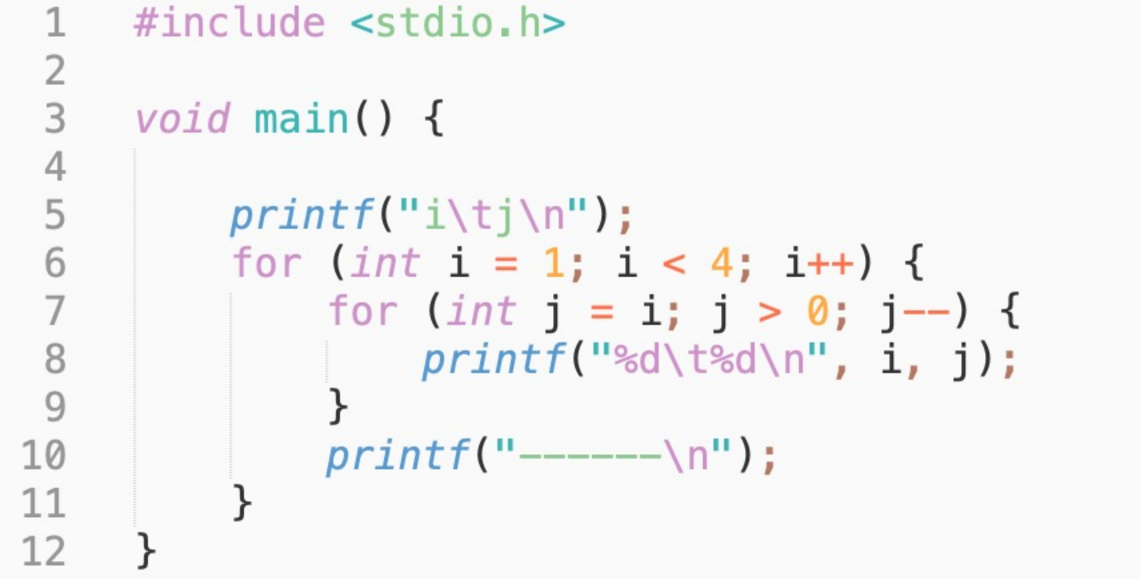


```
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	print (j)
	0	exit (j)

- What is the output of the following code:

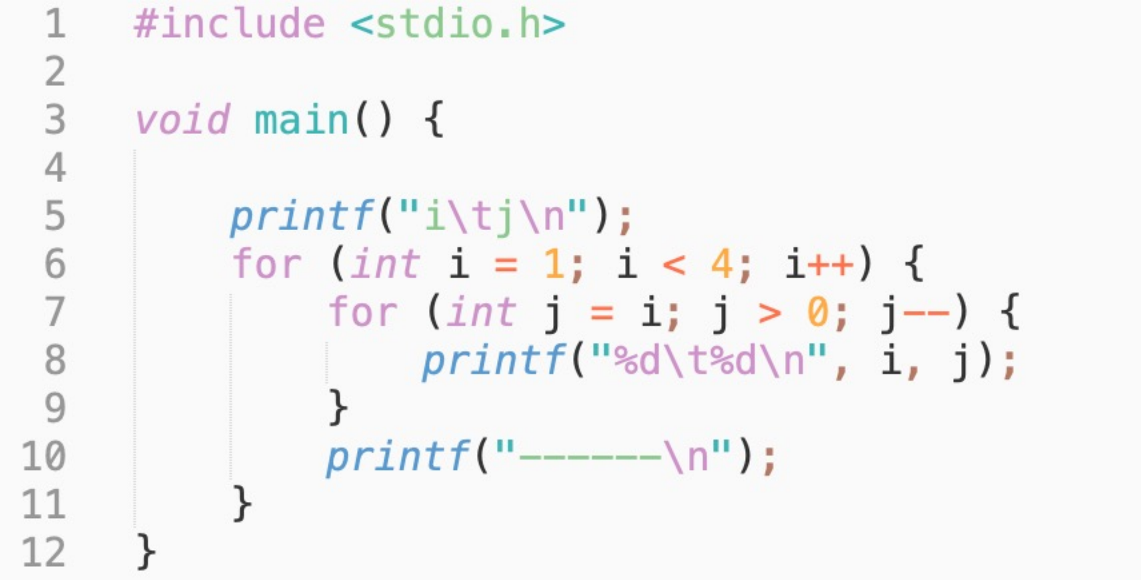


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)

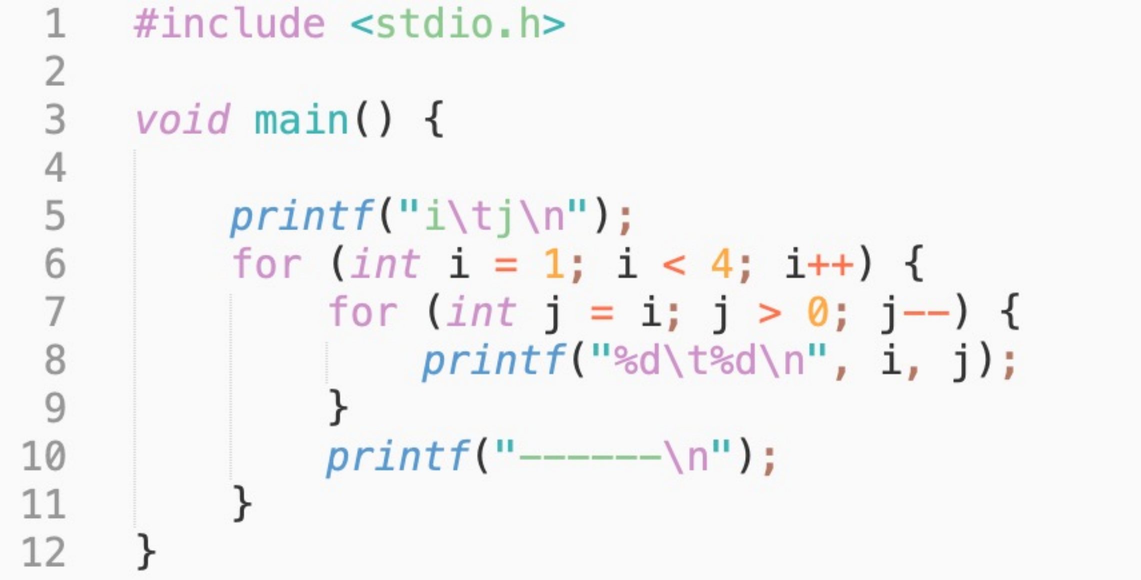
- What is the output of the following code:



```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

- What is the output of the following code:



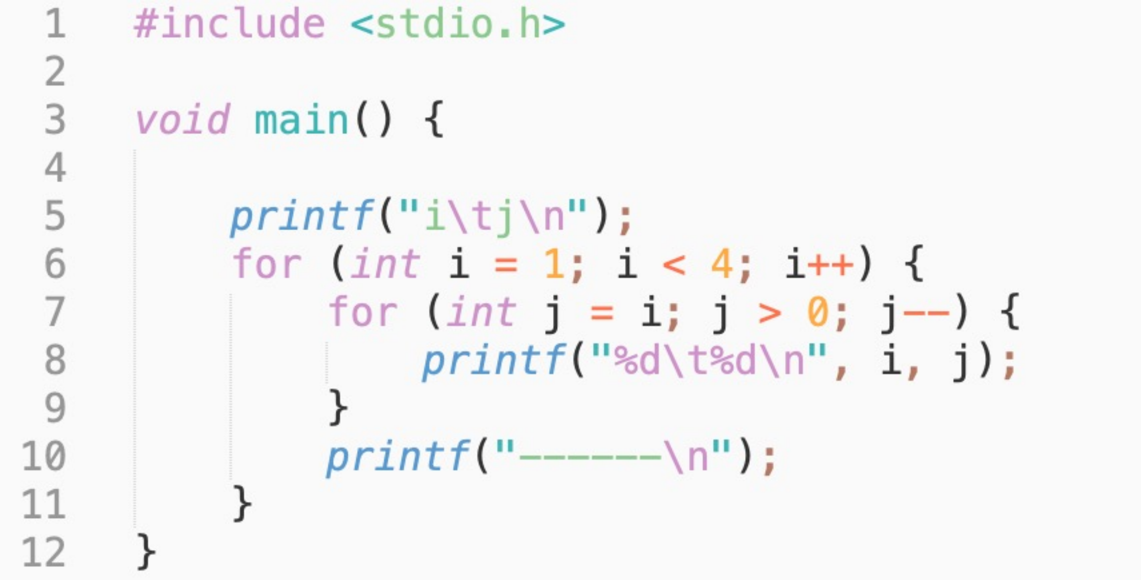
```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2		

Nested Loops

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	

- What is the output of the following code:

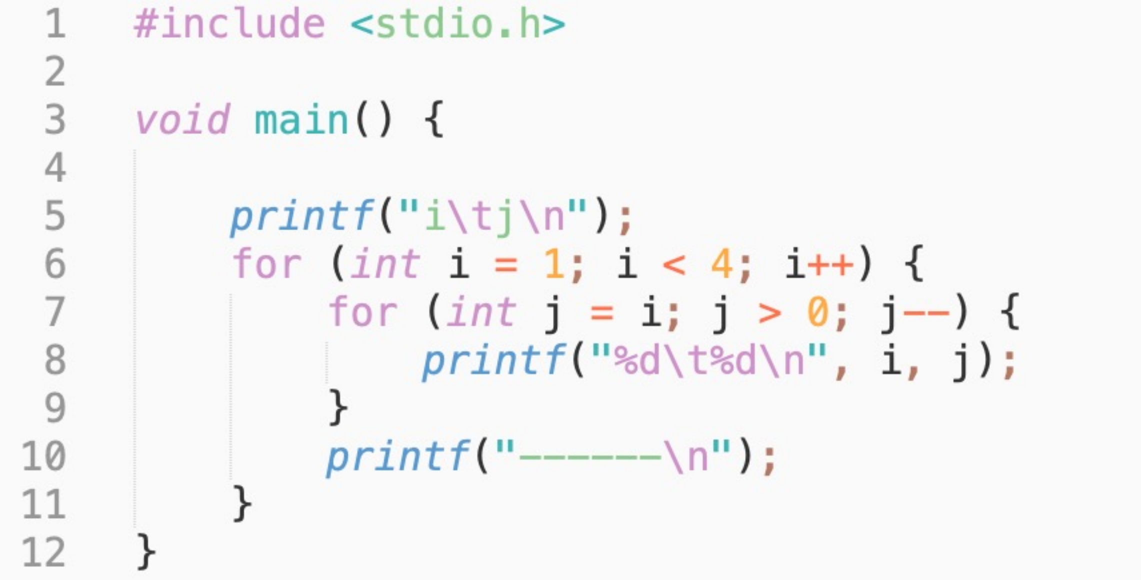


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)

- What is the output of the following code:

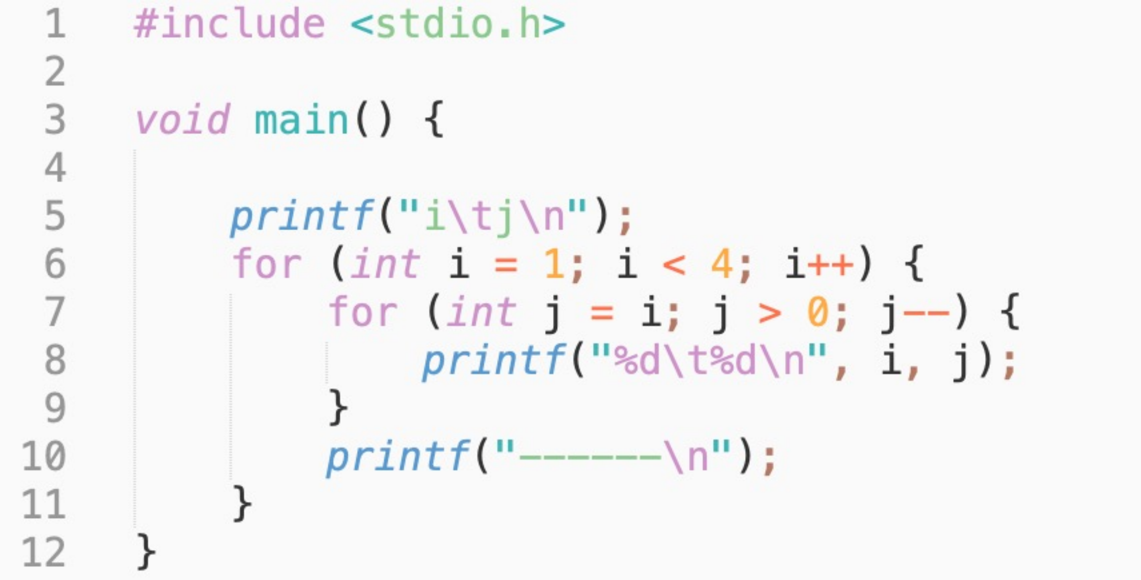


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	

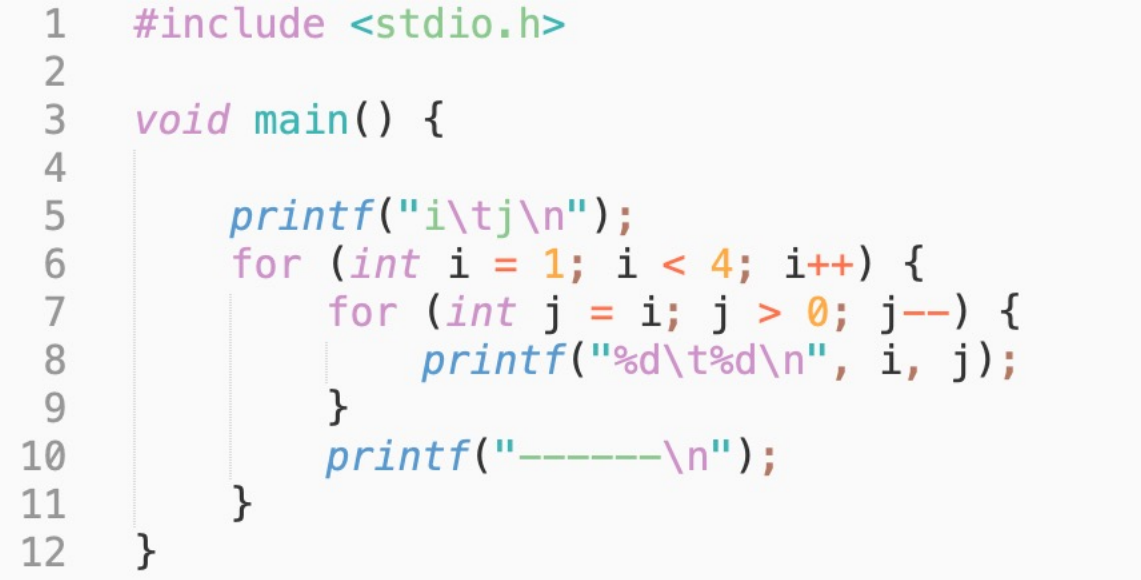
- What is the output of the following code:



```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

- What is the output of the following code:

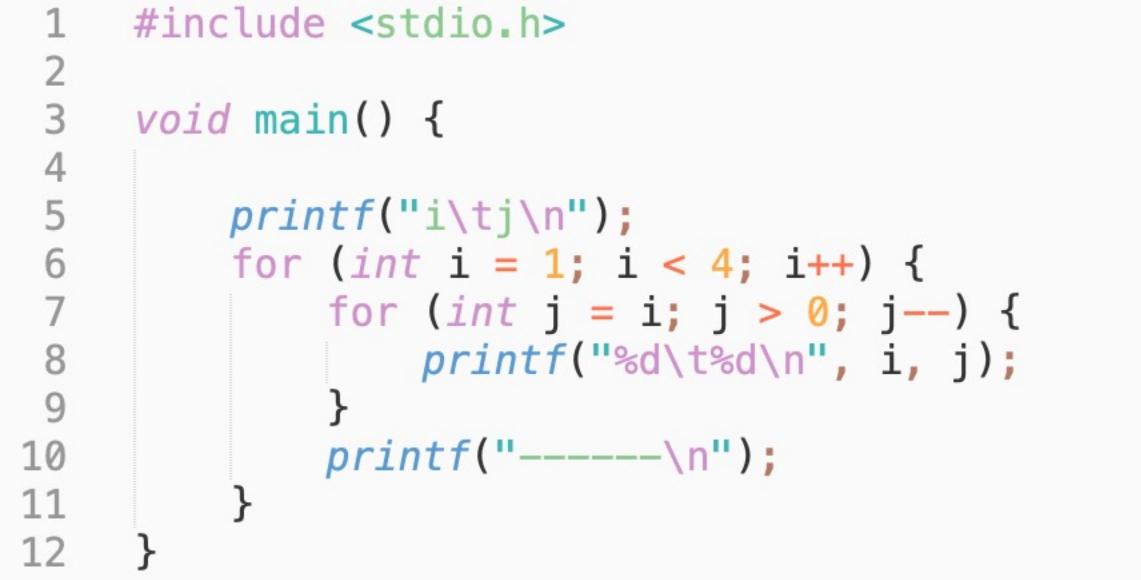


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)

Nested Loops

- What is the output of the following code:



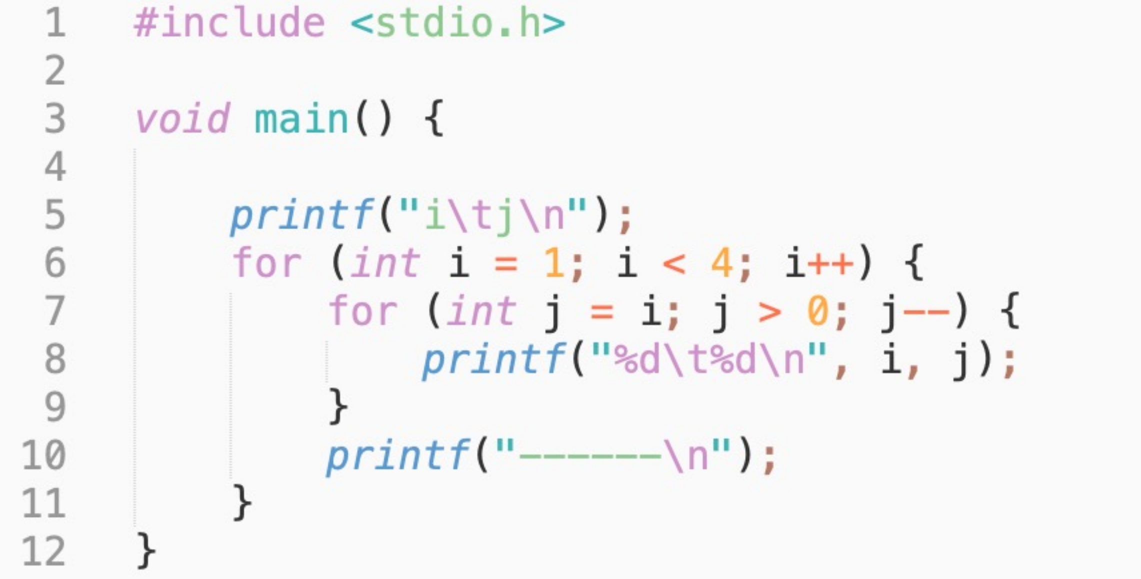
```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	

Nested Loops

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)

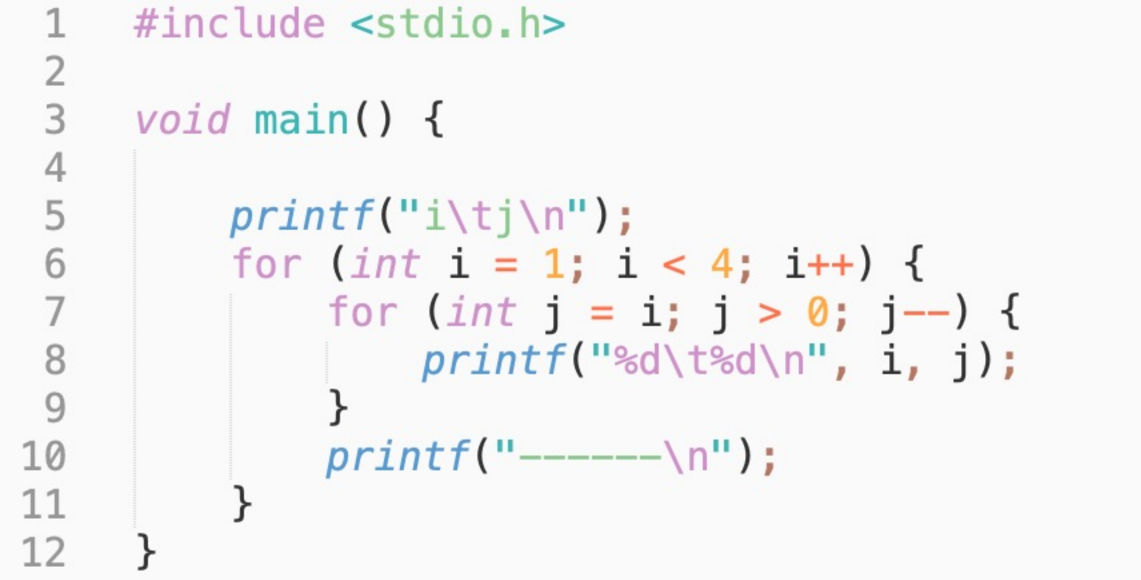
- What is the output of the following code:



```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

Nested Loops

- What is the output of the following code:

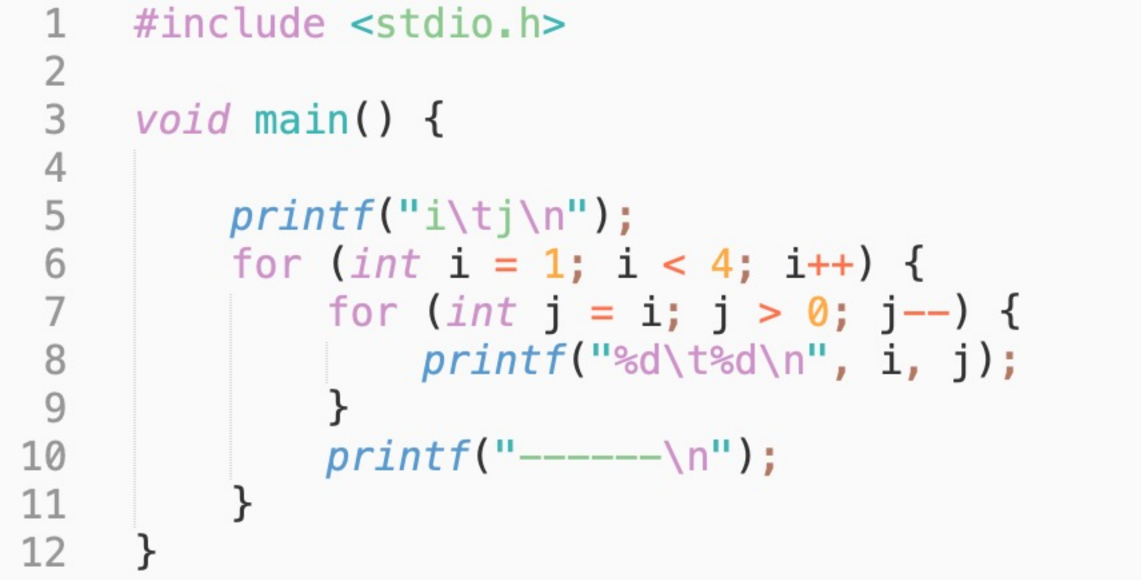


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)

Nested Loops

- What is the output of the following code:

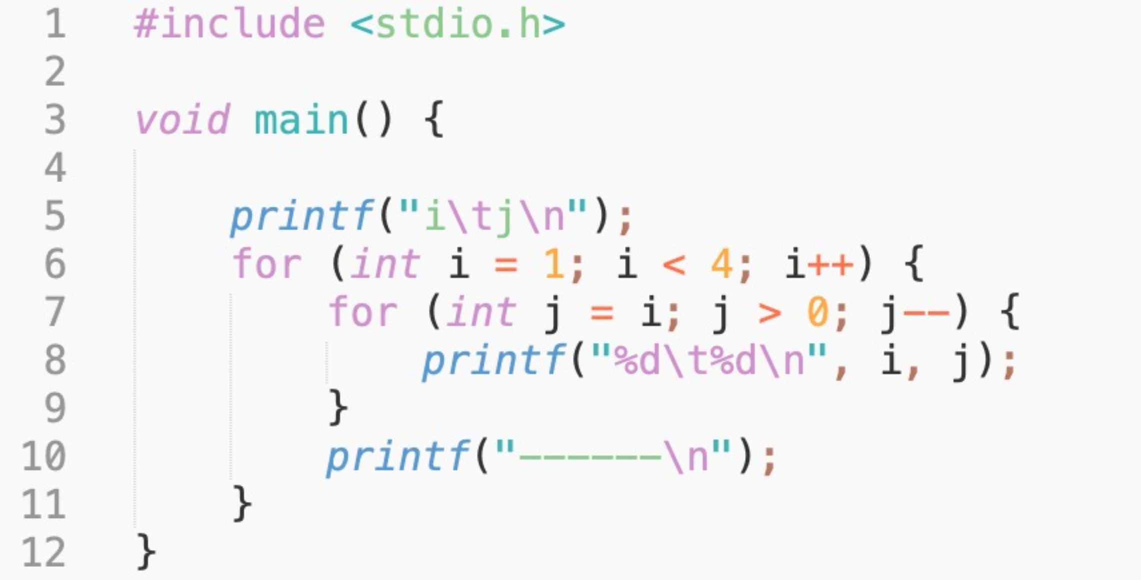


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3		

Nested Loops

- What is the output of the following code:

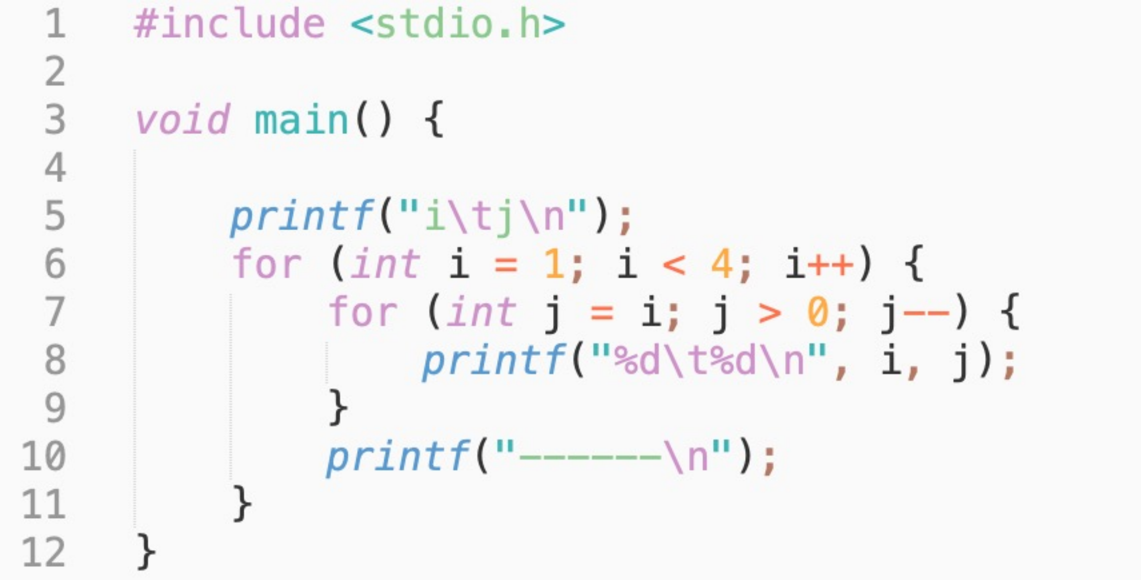


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	

Nested Loops

- What is the output of the following code:

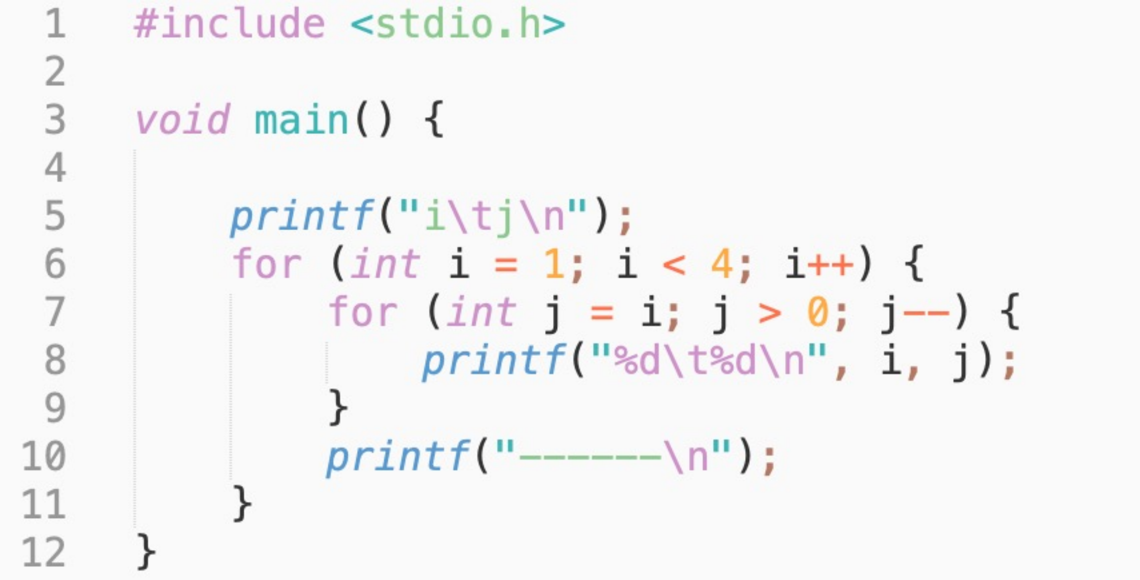


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)

Nested Loops

- What is the output of the following code:

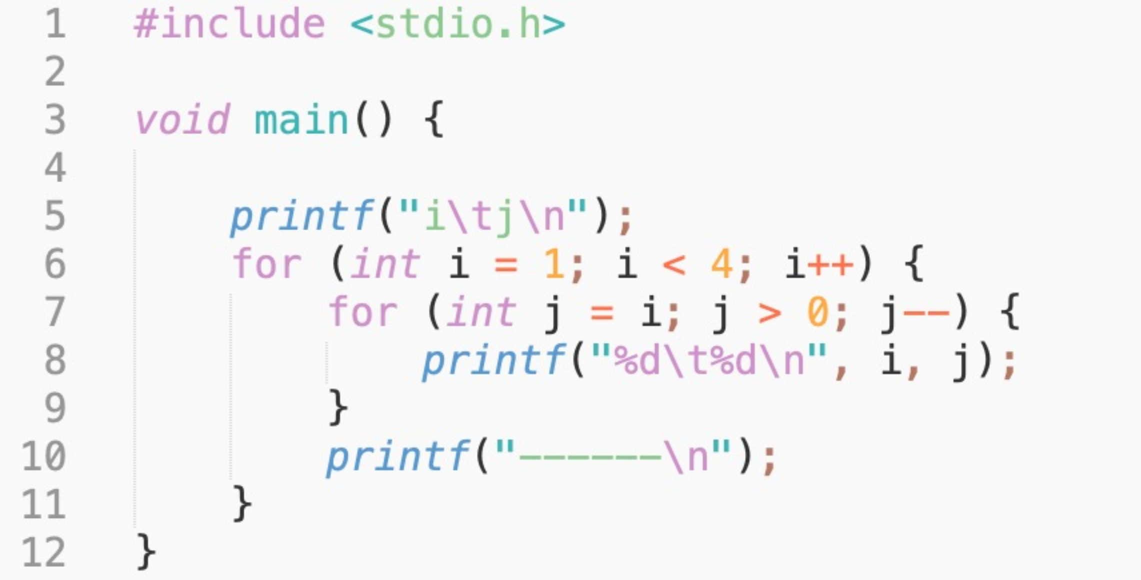


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	

Nested Loops

- What is the output of the following code:

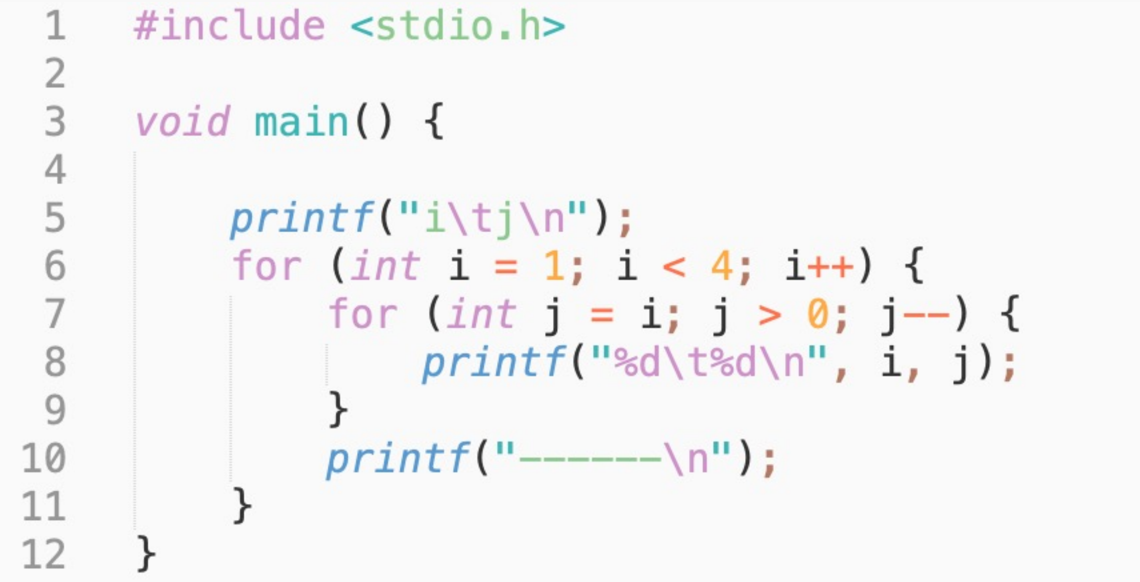


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)

Nested Loops

- What is the output of the following code:

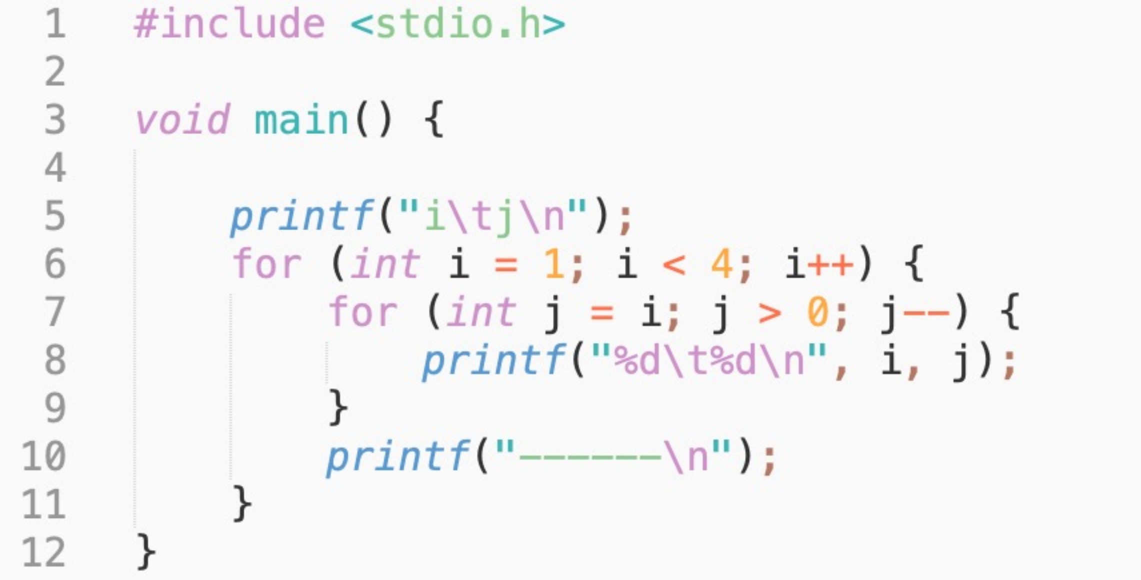


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	

Nested Loops

- What is the output of the following code:

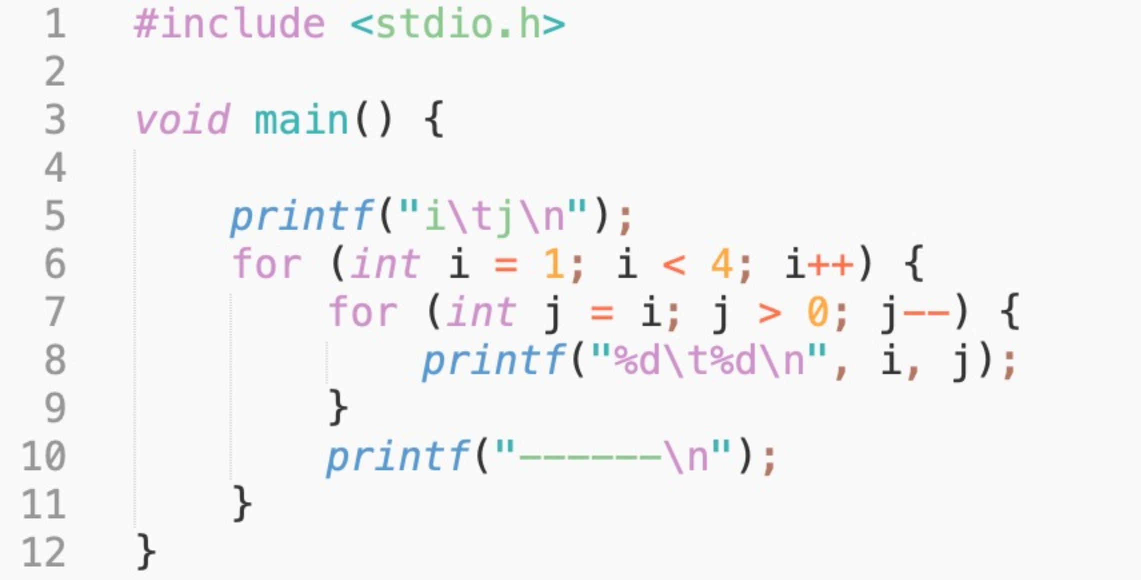


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	print (j)

Nested Loops

- What is the output of the following code:

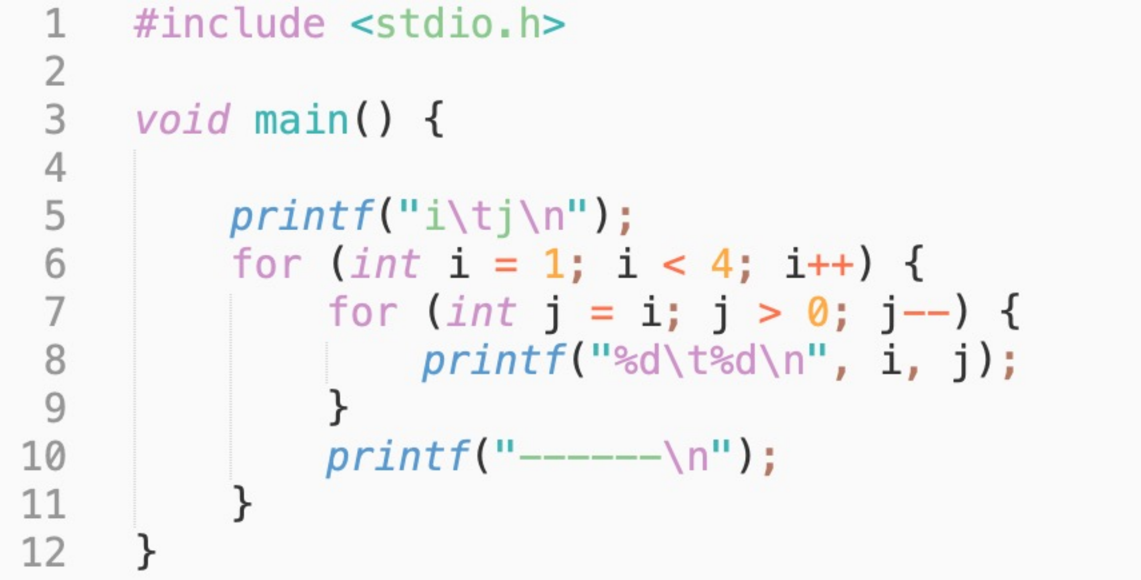


```
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	print (j)
	0	

Nested Loops

- What is the output of the following code:

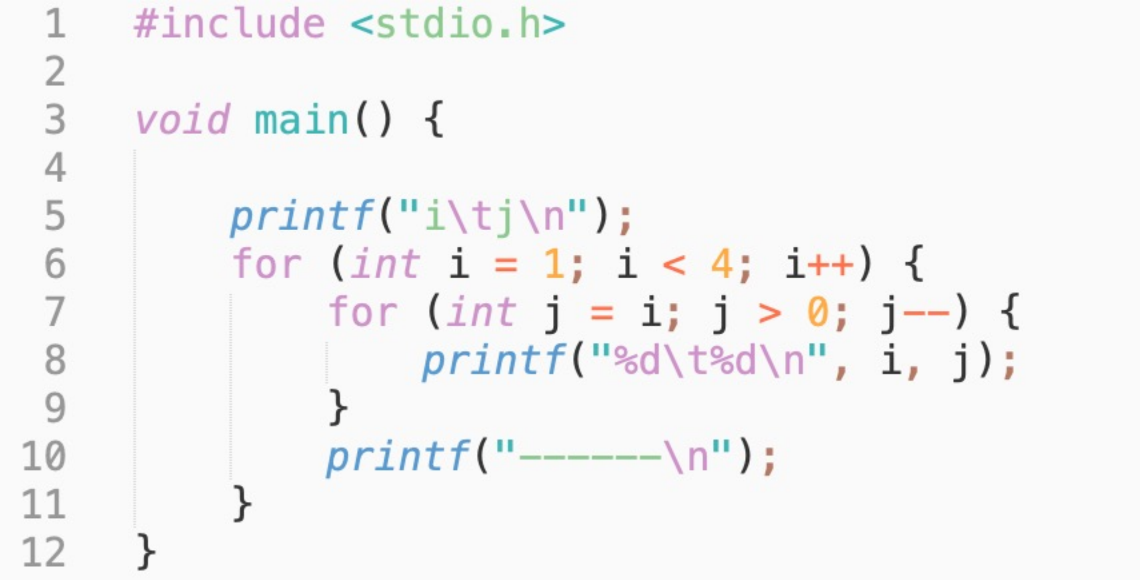


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	print (j)
	0	exit (j)

Nested Loops

- What is the output of the following code:

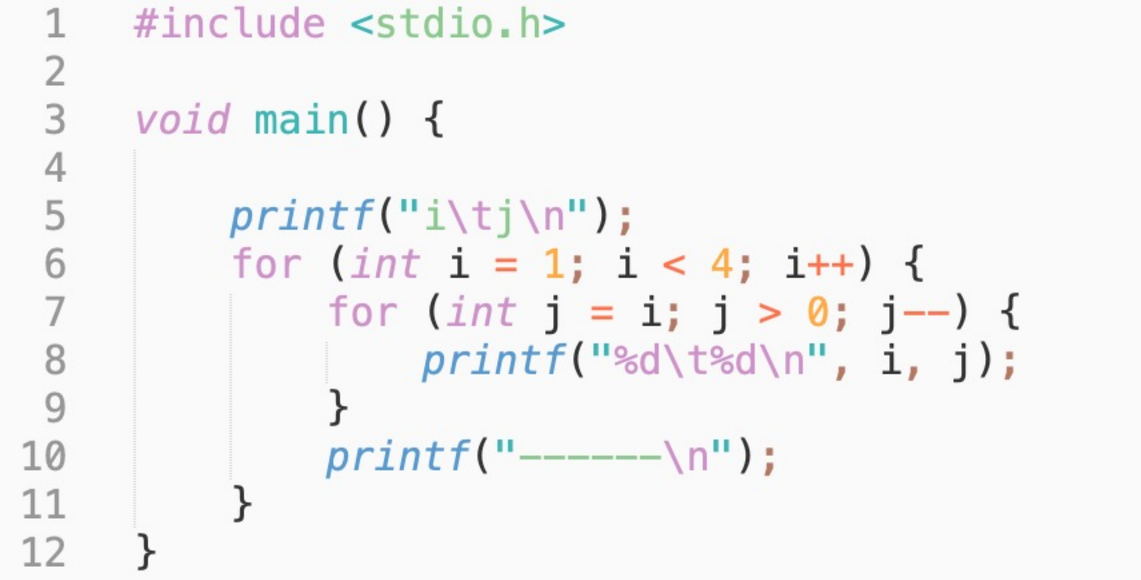


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)

Nested Loops

- What is the output of the following code:

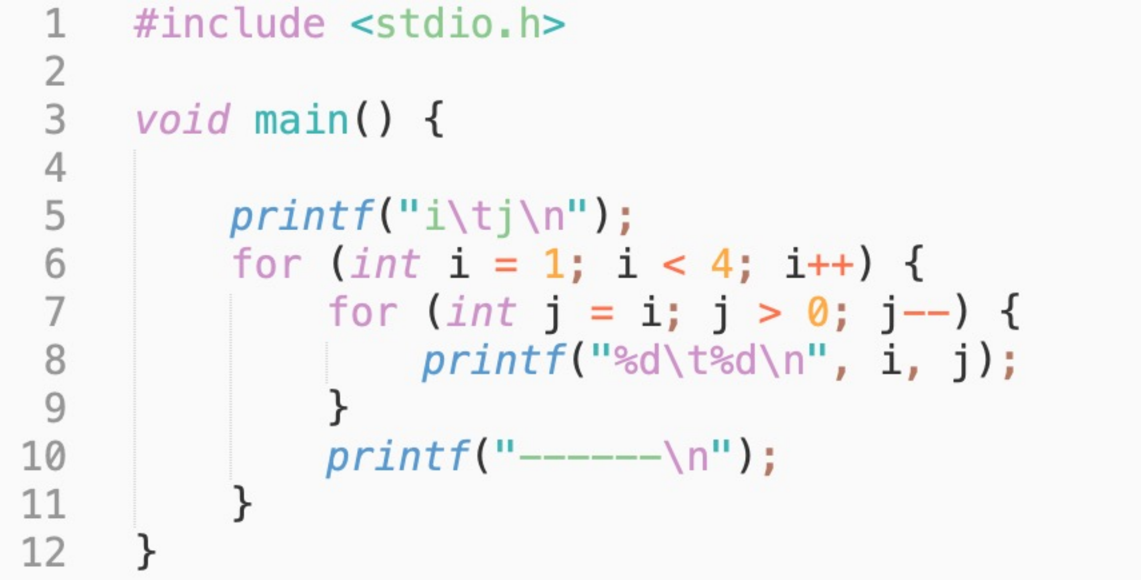


```
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
4		

Nested Loops

- What is the output of the following code:

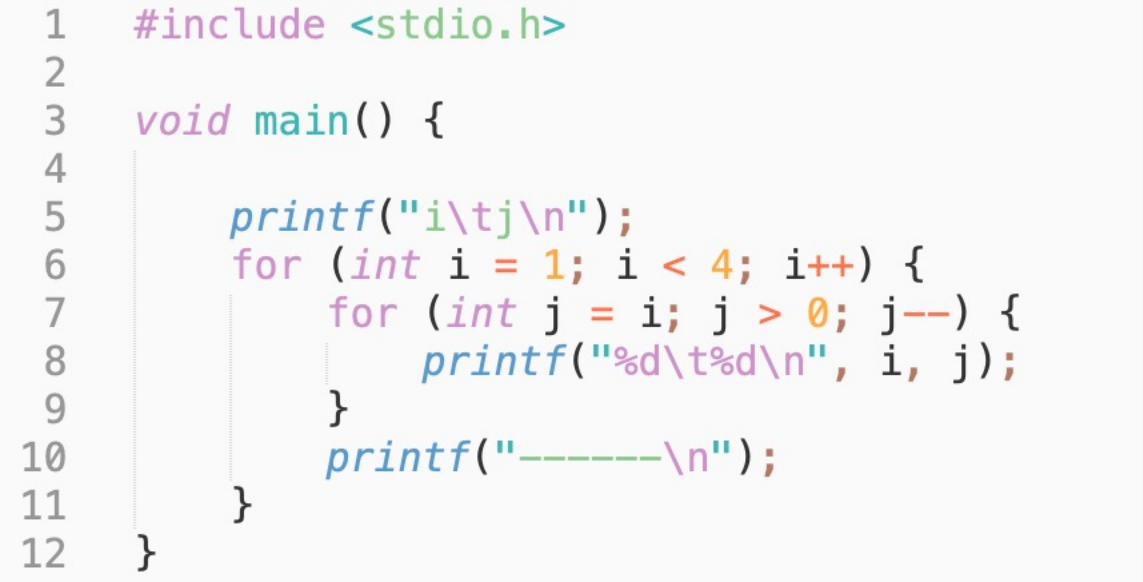


```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

i	j	action
1	1	print (j)
	0	exit (j)
		print (i)
2	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
3	3	print (j)
	2	print (j)
	1	print (j)
	0	exit (j)
		print (i)
4		exit (i)

Nested Loops

- What is the output of the following code:



```c
#include <stdio.h>

void main() {

    printf("i\tj\n");
    for (int i = 1; i < 4; i++) {
        for (int j = i; j > 0; j--) {
            printf("%d\t%d\n", i, j);
        }
        printf("------\n");
    }
}
```

```
i       j
1       1
------------
2       2
2       1
------------
3       3
3       2
3       1
------------
```

Loops – Examples: Perfect Number

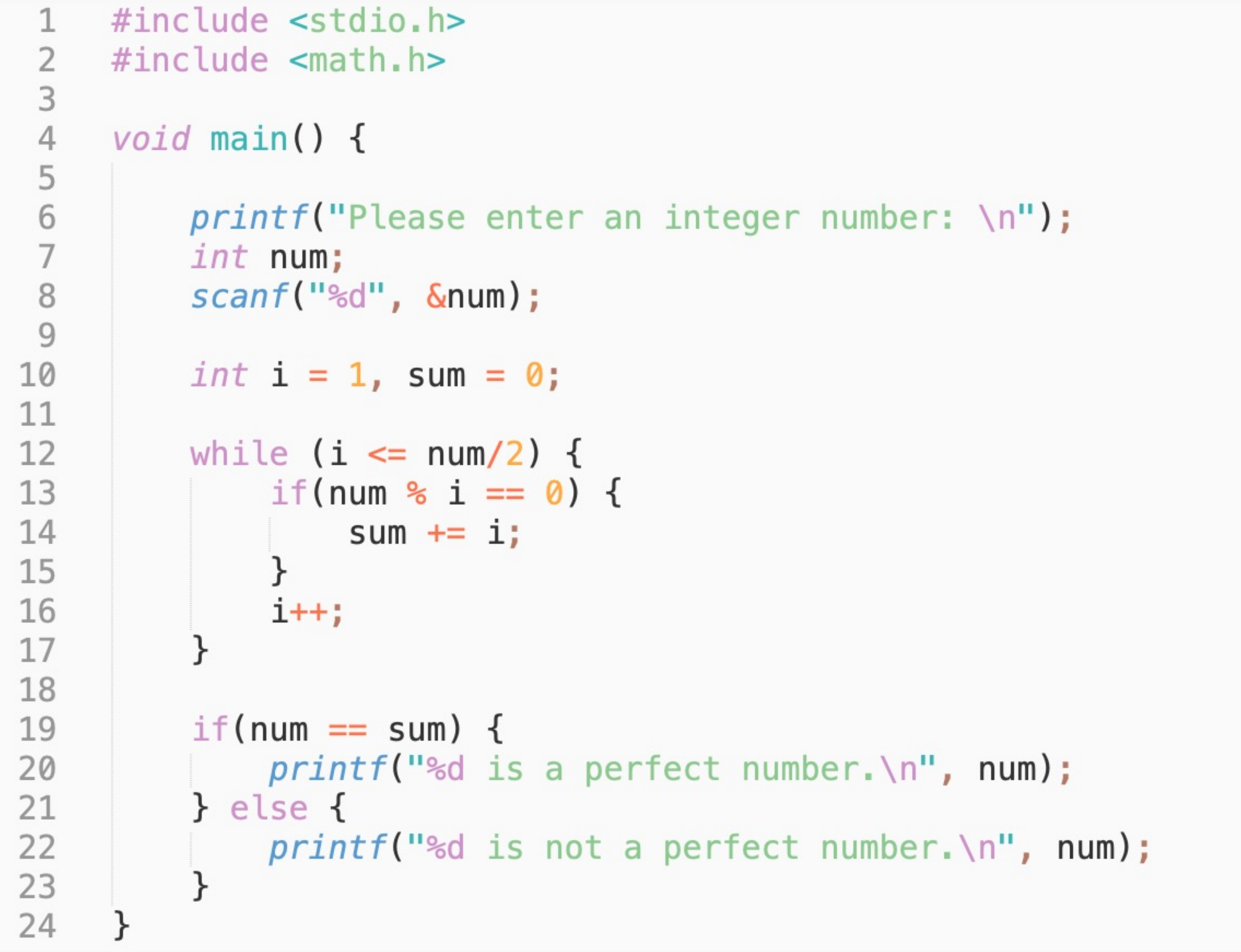
- Let’s write the code that reads a number from the user and finds if the number is a perfect number.
- A perfect number is the number that the sum of all its divisors, excluding itself, equals the number itself.
- For example:

6 is a perfect number because 1 + 2 + 3 = 6.

28 is a perfect number because 1 + 2 + 4 + 7 + 14 = 28.

30 is not a perfect number because 1 + 2 + 3 + 5 + 6 + 10 + 15 = 42.

Loops – Examples: Perfect Number



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

void main() {

    printf("Please enter an integer number: \n");
    int num;
    scanf("%d", &num);

    int i = 1, sum = 0;

    while (i <= num/2) {
        if(num % i == 0) {
            sum += i;
        }
        i++;
    }

    if(num == sum) {
        printf("%d is a perfect number.\n", num);
    } else {
        printf("%d is not a perfect number.\n", num);
    }
}
```

Loops – Examples: Narcissistic Numbers

- Let’s write the code that reads a number from the user and finds if the number is a narcissistic number.
- A narcissistic number is the number that the sum of each of its digit raised to the n^{th}, where n is the number of digits in the number, equals the number itself.
- For example:

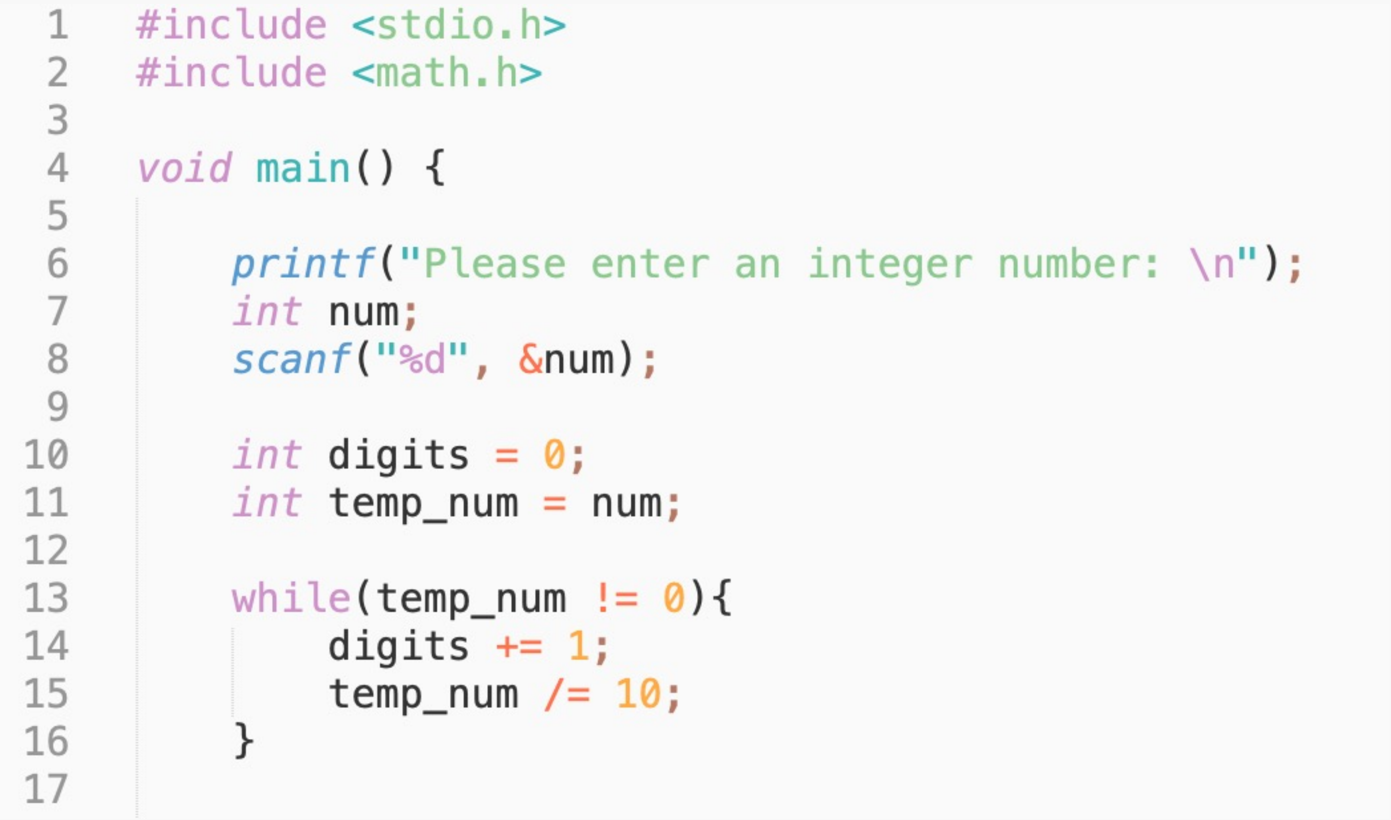
4 is a narcissistic number because $4^1 = 4$.

153 is a narcissistic number because $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

8208 is a narcissistic number because $8^4 + 2^4 + 0^4 + 8^8 = 4096 + 16 + 0 + 4096 = 8208$.

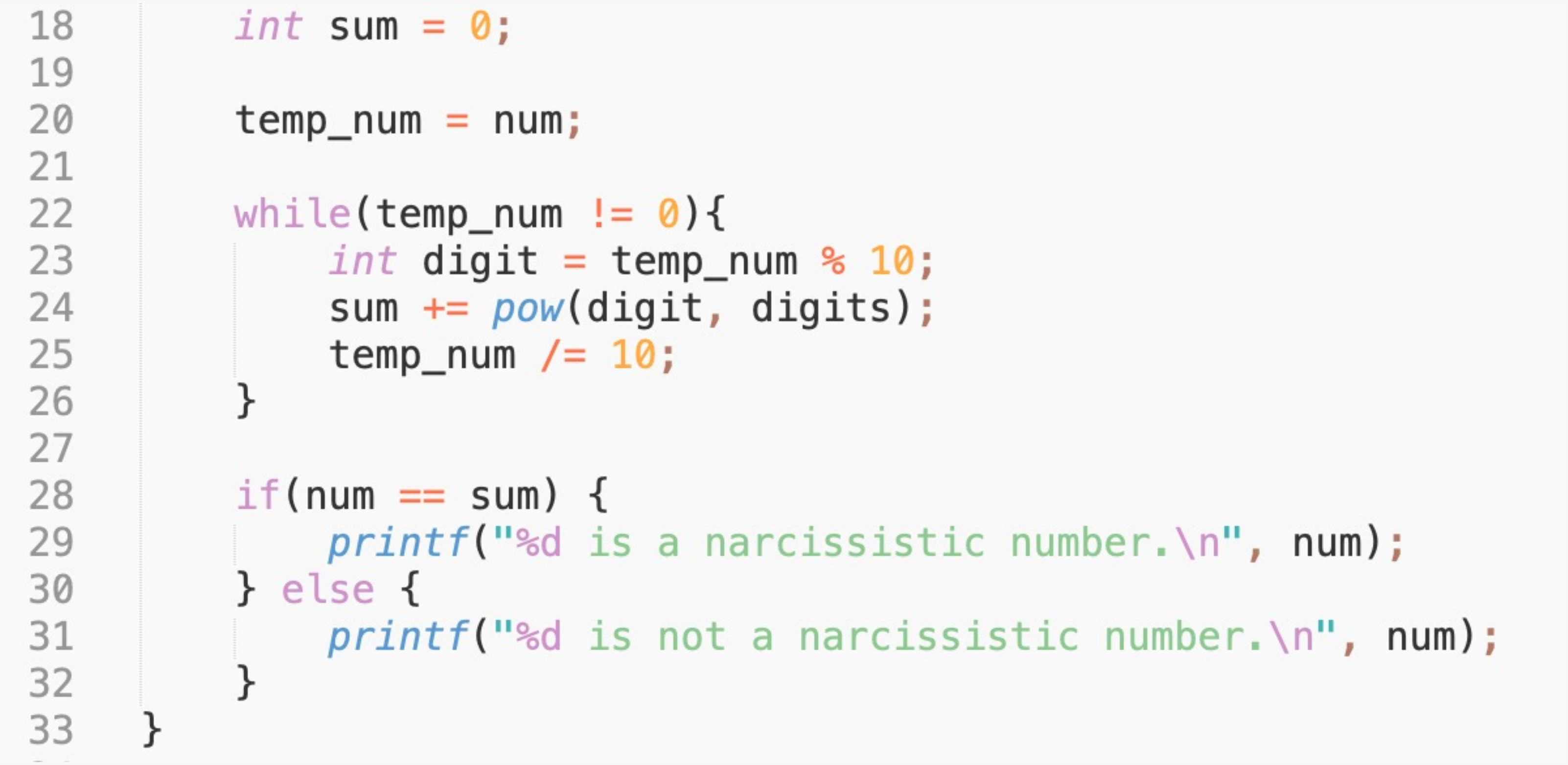
25 is not a narcissistic number because $2^2 + 5^2 = 4 + 25 = 29$.

Loops – Examples: Narcissistic Numbers



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

void main() {

    printf("Please enter an integer number: \n");
    int num;
    scanf("%d", &num);

    int digits = 0;
    int temp_num = num;

    while(temp_num != 0){
        digits += 1;
        temp_num /= 10;
    }

```

Loops – Examples: Narcissistic Numbers

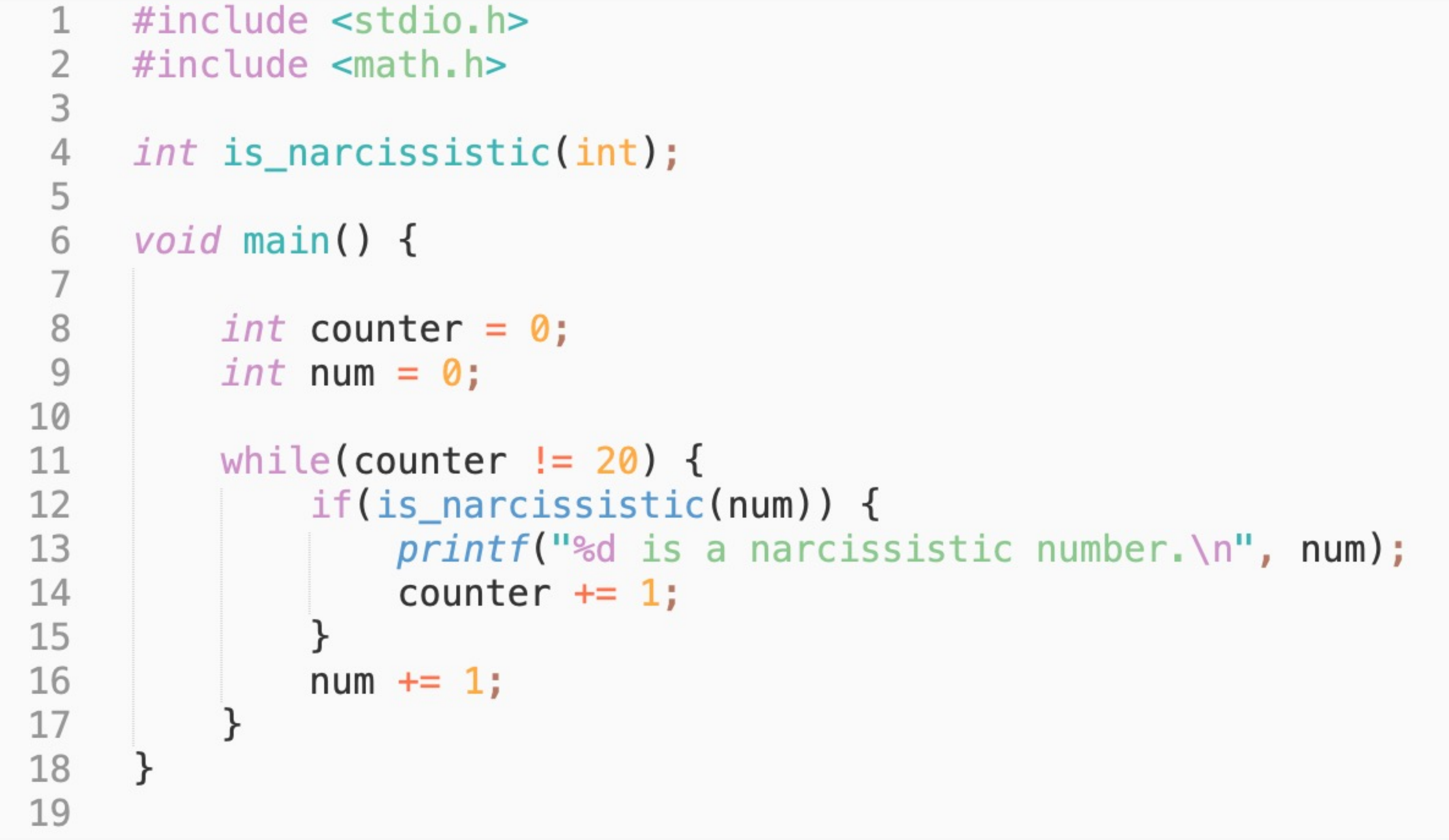


```c
    int sum = 0;

    temp_num = num;

    while(temp_num != 0){
        int digit = temp_num % 10;
        sum += pow(digit, digits);
        temp_num /= 10;
    }

    if(num == sum) {
        printf("%d is a narcissistic number.\n", num);
    } else {
        printf("%d is not a narcissistic number.\n", num);
    }
}
```

Loops – Examples: The First 20 Narcissistic Numbers

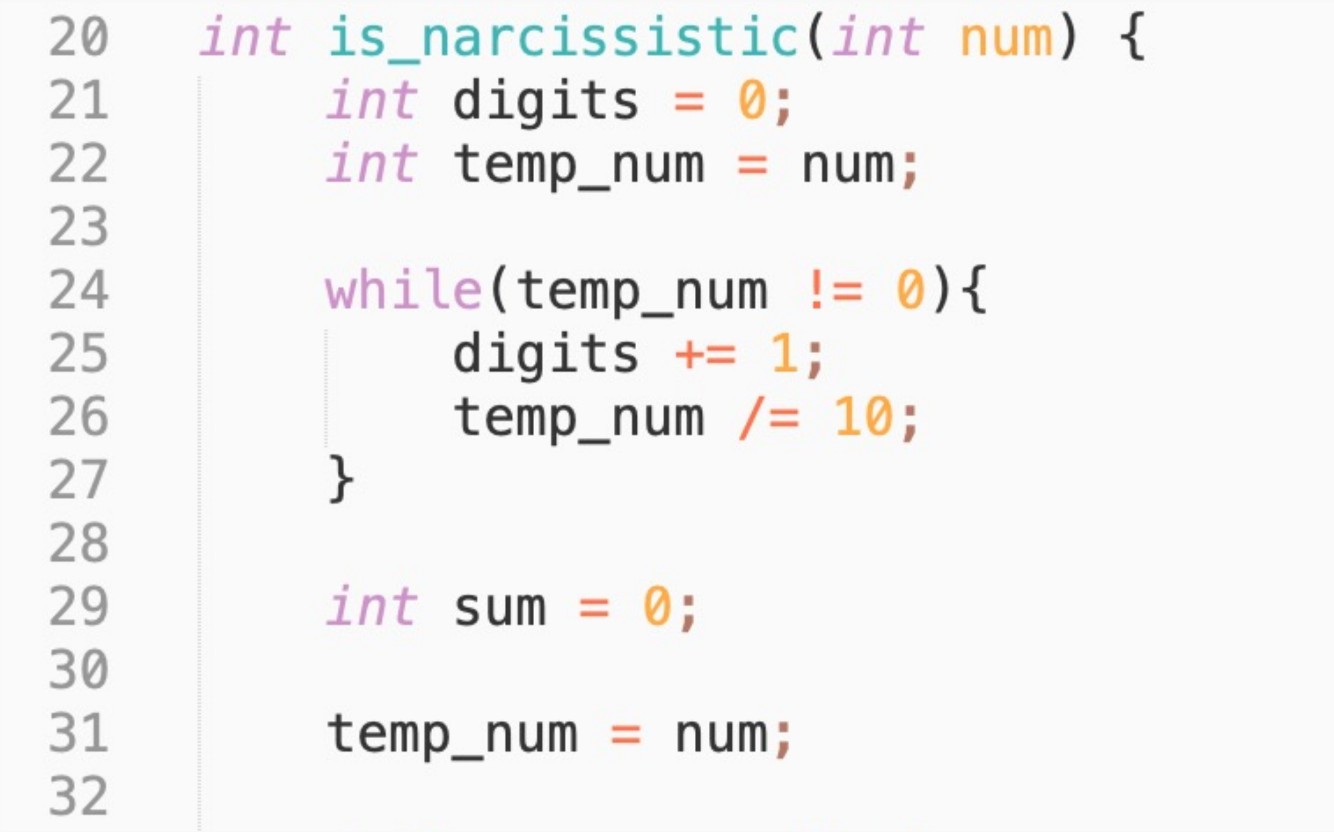
- Let’s write the code prints out the first 20 narcissistic numbers.
- We can create a function that takes a number and returns 1 if it is a narcissistic number, and 0 if it is not.
- We can create a result-controlled loop that exits if we reached 20 narcissistic numbers.

Loops – Examples: The First 20 Narcissistic Numbers



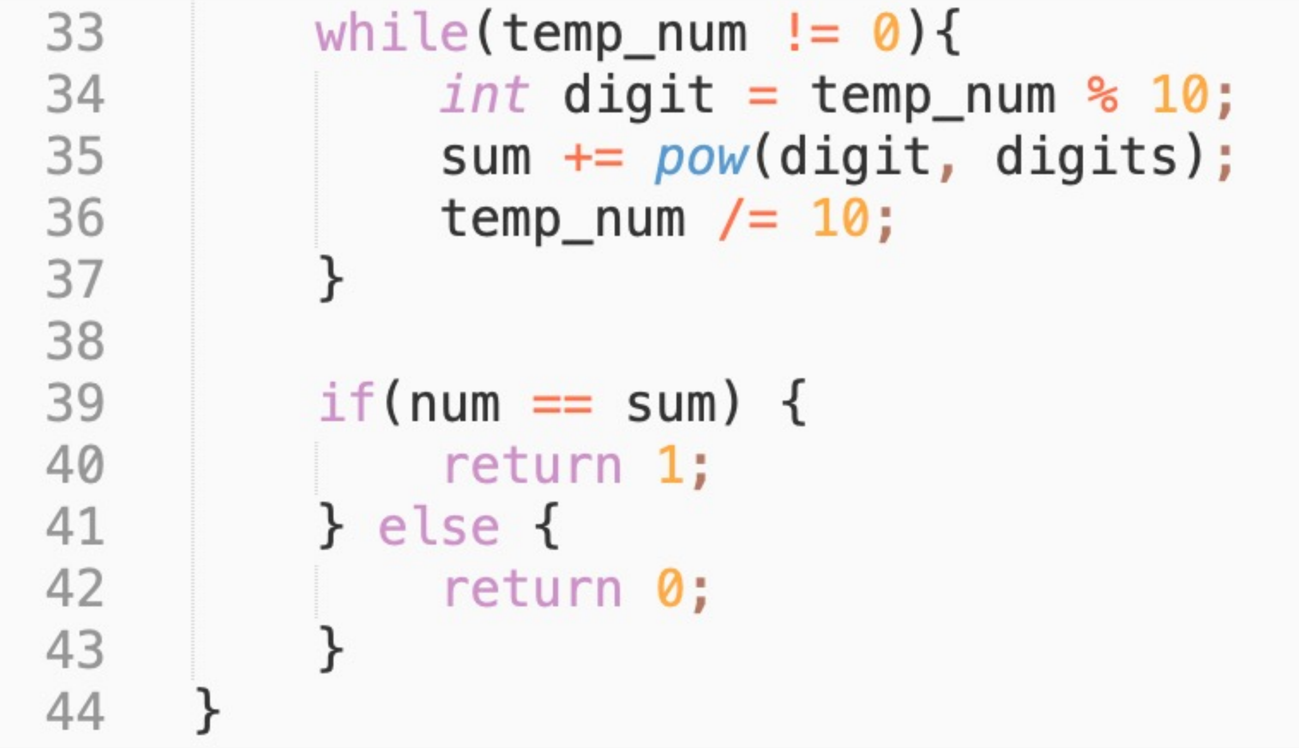
```c
#include <stdio.h>
#include <math.h>

int is_narcissistic(int);

void main() {

    int counter = 0;
    int num = 0;

    while(counter != 20) {
        if(is_narcissistic(num)) {
            printf("%d is a narcissistic number.\n", num);
            counter += 1;
        }
        num += 1;
    }
}

```

Loops – Examples: The First 20 Narcissistic Numbers



```c
int is_narcissistic(int num) {
    int digits = 0;
    int temp_num = num;

    while(temp_num != 0){
        digits += 1;
        temp_num /= 10;
    }

    int sum = 0;

    temp_num = num;

```

Loops – Examples: The First 20 Narcissistic Numbers



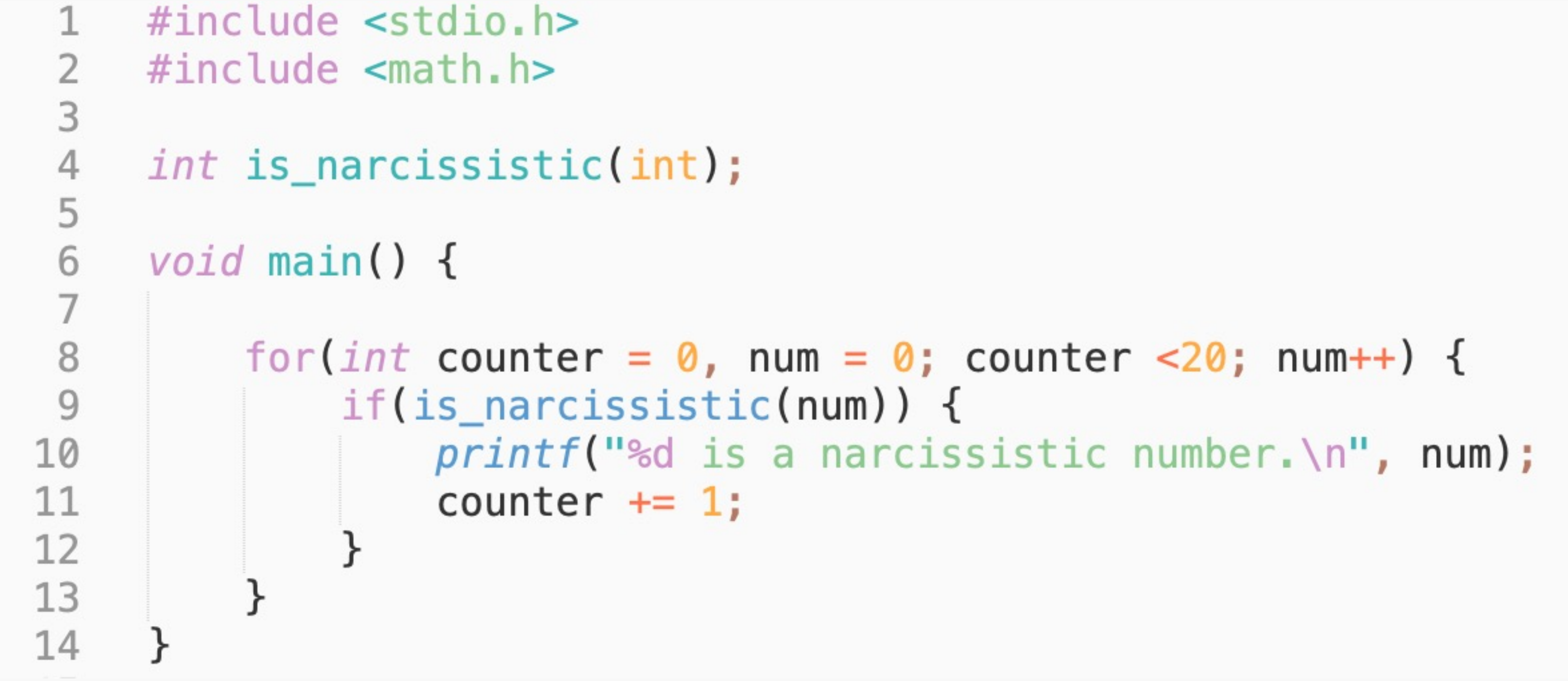
```
    while(temp_num != 0){
        int digit = temp_num % 10;
        sum += pow(digit, digits);
        temp_num /= 10;
    }

    if(num == sum) {
        return 1;
    } else {
        return 0;
    }
}
```

Loops – Examples: The First 20 Narcissistic Numbers

- Can we rewrite the main part of the previous example using a *for* loop instead of a *while* loop?

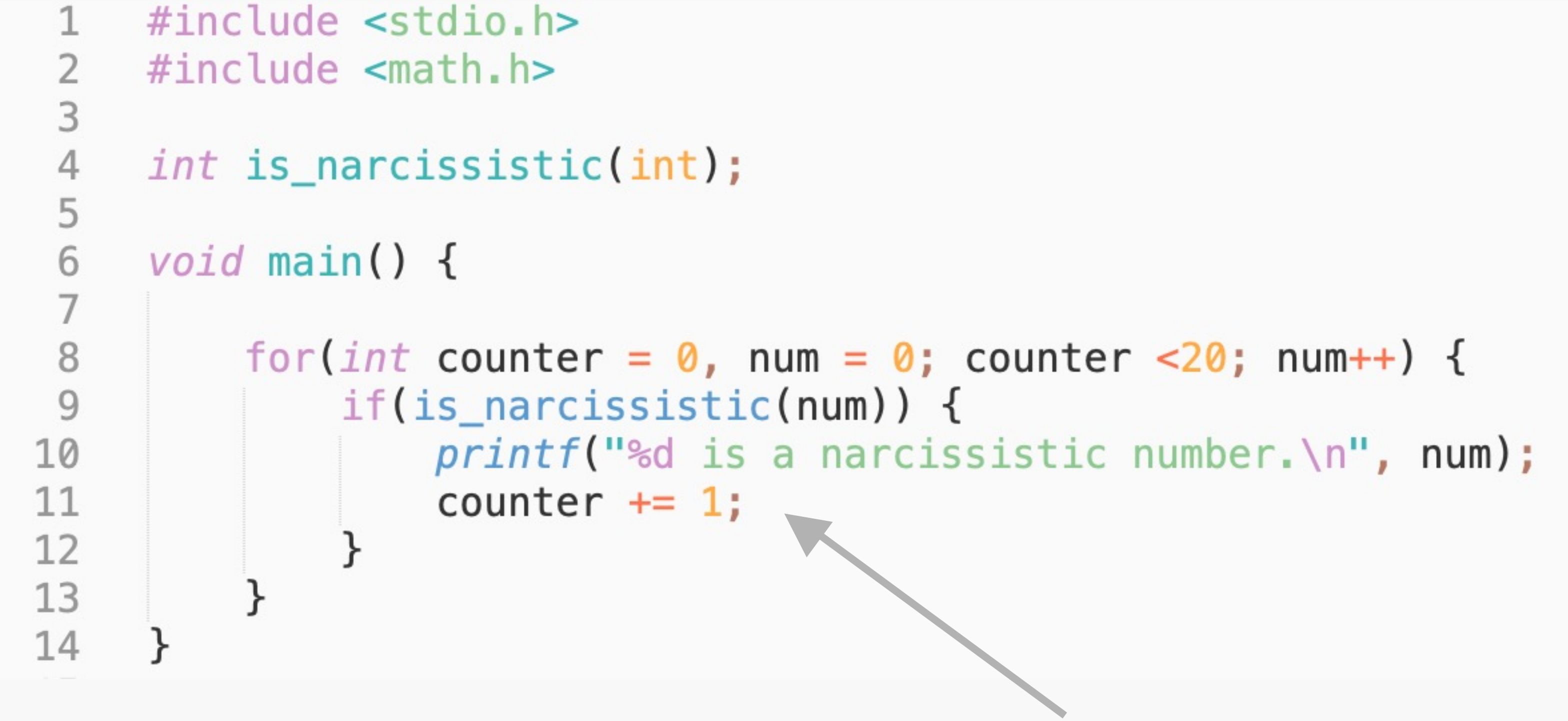
- To do that, we need to determine the following:
  - *What are the initializing statements?*
  - *What is the loop condition?*
  - *What are the update statements?*

Loops – Examples: The First 20 Narcissistic Numbers



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

int is_narcissistic(int);

void main() {

    for(int counter = 0, num = 0; counter <20; num++) {
        if(is_narcissistic(num)) {
            printf("%d is a narcissistic number.\n", num);
            counter += 1;
        }
    }
}
```

Loops – Examples: The First 20 Narcissistic Numbers



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

int is_narcissistic(int);

void main() {

    for(int counter = 0, num = 0; counter <20; num++) {
        if(is_narcissistic(num)) {
            printf("%d is a narcissistic number.\n", num);
            counter += 1;
        }
    }
}
```

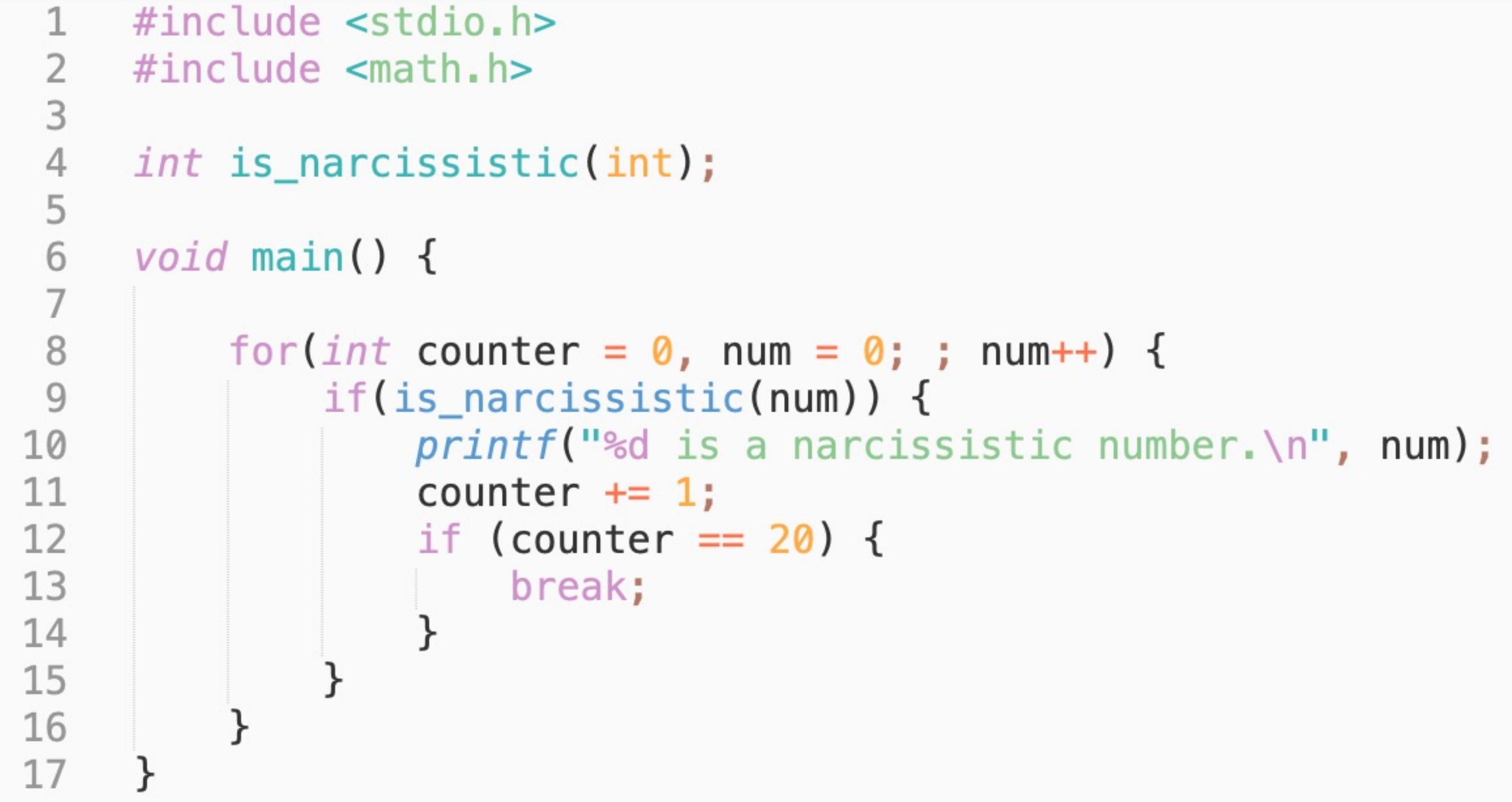
This is not an update statement because it doesn’t happen in every loop

Loops – Examples: The First 20 Narcissistic Numbers

- Can we rewrite *for* loop without a condition?

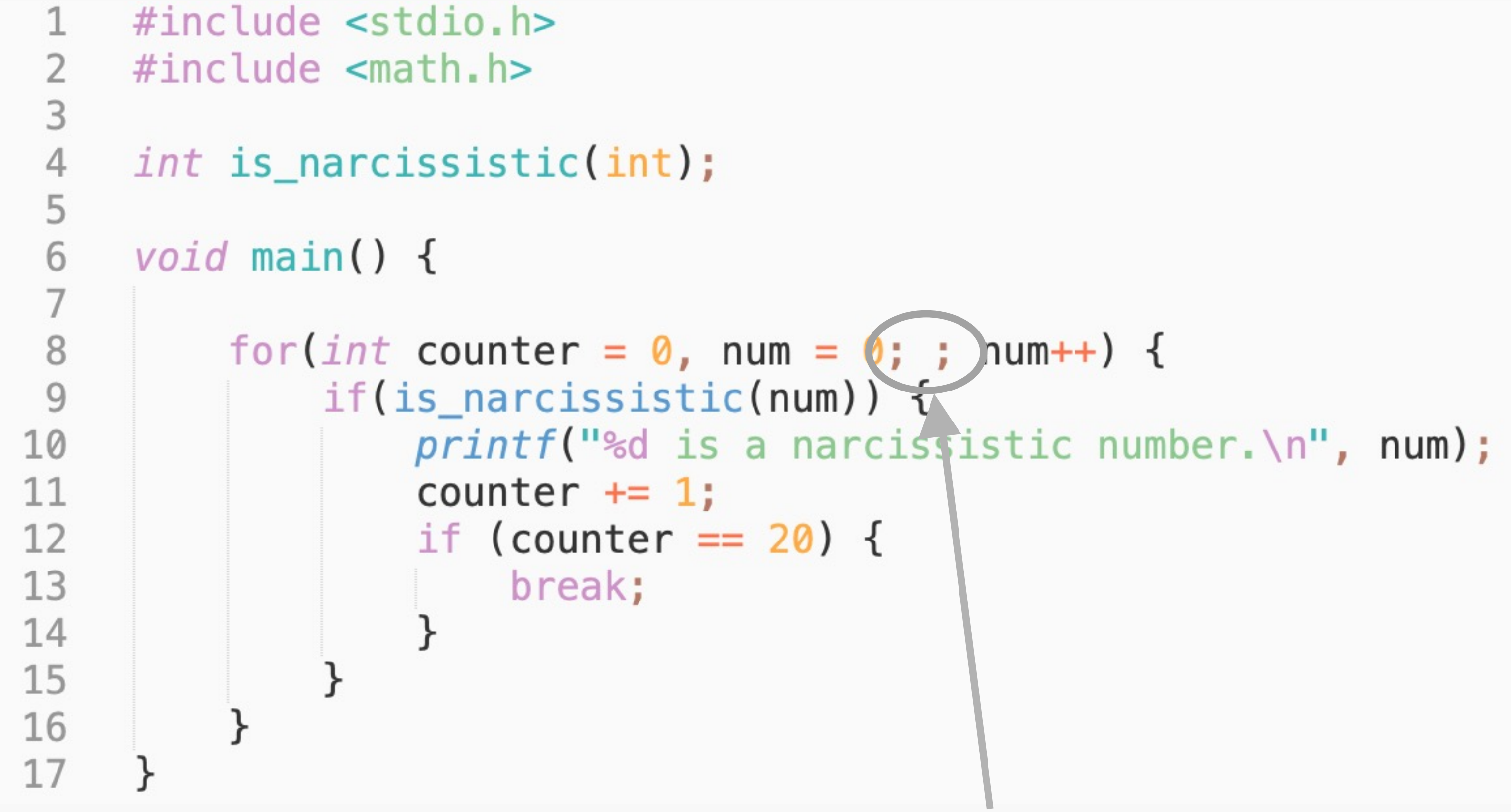
- We need to make sure that the compiler knows when to leave the loop.
  - *We use the break statement.*

Loops – Examples: The First 20 Narcissistic Numbers



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

int is_narcissistic(int);

void main() {

    for(int counter = 0, num = 0; ; num++) {
        if(is_narcissistic(num)) {
            printf("%d is a narcissistic number.\n", num);
            counter += 1;
            if (counter == 20) {
                break;
            }
        }
    }
}
```

Loops – Examples: The First 20 Narcissistic Numbers



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

int is_narcissistic(int);

void main() {

    for(int counter = 0, num = 0; ; num++) {
        if(is_narcissistic(num)) {
            printf("%d is a narcissistic number.\n", num);
            counter += 1;
            if (counter == 20) {
                break;
            }
        }
    }
}
```

Notice the semicolons that we need to keep

INCREMENTS

Pre- and post-increments and pre- and post-decrements

Pre- and Post-Increment

- We have been using expressions that look like

  `i++` and `++i`

  to update values in loops.

- What do these mean exactly? And how are they different?

before

- ++i is called a pre-increment.
- i++ is called a post-increment.

after

Examples

Pre-increment

```
a = ++x * b;
```

↔

```
x = x + 1;
a = x * b;
```

Post-increment

```
a = x++ * b;
```

↔

```
a = x * b;
x = x + 1;
```

Pre- and Post-Decrements

- A pre-decrement looks like this:

```
--x;
```

and a post-decrement looks like this:

```
x--;
```

- Everything we will learn about increments works just the same for decrements, with the exception that decrements decrease one from the variable.

Examples

Pre-decrement

```
a = --x * b;
```

```
x = x - 1;
a = x * b;
```

Post-decrement

```
a = x-- * b;
```

```
a = x * b;
x = x - 1;
```

Practical Example 1

```
int a=2, b=3, c;
c = ++a * b++;
printf(“%d\n”, a);
printf(“%d\n”, b);
printf(“%d\n”, c);
```

a	b	c

Practical Example 1

```
int a=2, b=3, c;
c = ++a * b++;
printf(“%d\n”, a);
printf(“%d\n”, b);
printf(“%d\n”, c);
```

a	b	c
2	3	

Practical Example 1

```
int a=2, b=3, c;
c = ++a * b++;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

a	b	c
2	3	
3	3	

Practical Example 1

```
int a=2, b=3, c;
c = ++a * b++;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

a	b	c
2	3	
3	3	
3	3	9

Practical Example 1

```
int a=2, b=3, c;
c = ++a * b++;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

a	b	c
2	3	
3	3	
3	3	9
3	4	9

Practical Example 1

```
int a=2, b=3, c;
c = ++a * b++;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

```
Output:
3
4
9
```

a	b	c
2	3	
3	3	
3	3	9
3	4	9

Practical Example 2

```
int a=4, b=20;
b /= ++a;
printf(“%d\n”, a);
printf(“%d\n”, b);
```

a	b

Practical Example 2

```
int a=4, b=20;
b /= ++a;
printf(“%d\n”, a);
printf(“%d\n”, b);
```

a	b
4	20

Practical Example 2

```
int a=4, b=20;
b /= ++a;
printf("%d\n", a);
printf("%d\n", b);
```

a	b
4	20
5	20

Practical Example 2

```
int a=4, b=20;
b /= ++a;
printf(“%d\n”, a);
printf(“%d\n”, b);
```

a	b
4	20
5	20
5	4

Practical Example 2

```
int a=4, b=20;
b /= ++a;
printf(“%d\n”, a);
printf(“%d\n”, b);
```

a	b
4	20
5	20
5	4

```
Output:
5
4
```

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf(“%d\n”, a);
printf(“%d\n”, b);
printf(“%d\n”, c);
```

a	b	c

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

a	b	c
4	20	3

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

a	b	c
4	20	3
5	20	3

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf(“%d\n”, a);
printf(“%d\n”, b);
printf(“%d\n”, c);
```

$\boxed{8}$

a	b	c
4	20	3
5	20	3

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf("%d\n", a);
printf("%d\n", b);
printf("%d\n", c);
```

$\boxed{8}$

a	b	c
4	20	3
5	20	3
5	2	3

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf(“%d\n”, a);
printf(“%d\n”, b);
printf(“%d\n”, c);
```

a	b	c
4	20	3
5	20	3
5	2	3
5	2	2

Practical Example 3

```
int a=4, b=20, c=3;
b /= ++a + c--;
printf(“%d\n”, a);
printf(“%d\n”, b);
printf(“%d\n”, c);
```

a	b	c
4	20	3
5	20	3
5	2	3
5	2	2

Output:

```
5
2
2
```

Final Note on Increments

- If the statements consists solely of a pre-increment or decrement, or a post-increment or decrement, their behaviour will be the same.

- This means that

```
i++;
```

is equivalent to

```
++i;
```

and

```
i--;
```

is equivalent to

```
--i;
```