Loops

COP 3223C – Introduction to Programming with C

Fall 2025

Yancy Vance Paredes, PhD

Recap

- Previously, we talked about how C programs can execute statements based on certain **conditions**

- By now, we can make interesting programs!

Scenario

- Write a program that asks the user to enter a **whole number**, which we assume as the **passcode**. The user only has **3 attempts** to enter the passcode correctly.

- If the user enters the **correct passcode** (`1234`), display "Welcome".

- Otherwise, display "Try Again!". However, after the 3rd incorrect attempt, display "Intruder! Calling 911!".

Motivation

- What if we want to execute a statement **more than once**?
- Well, can’t we just simply copy and paste codes?
- No!

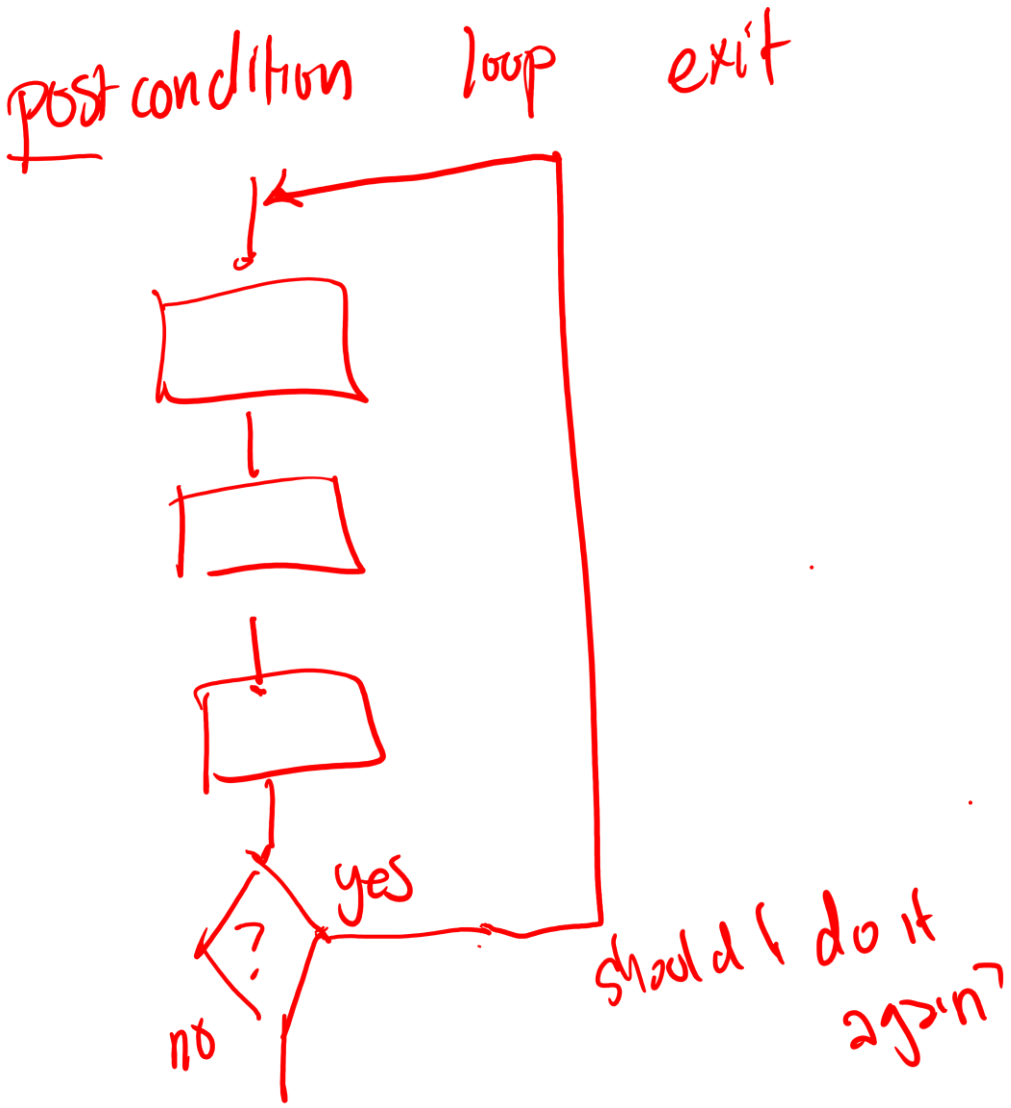
Repetition Structure /1

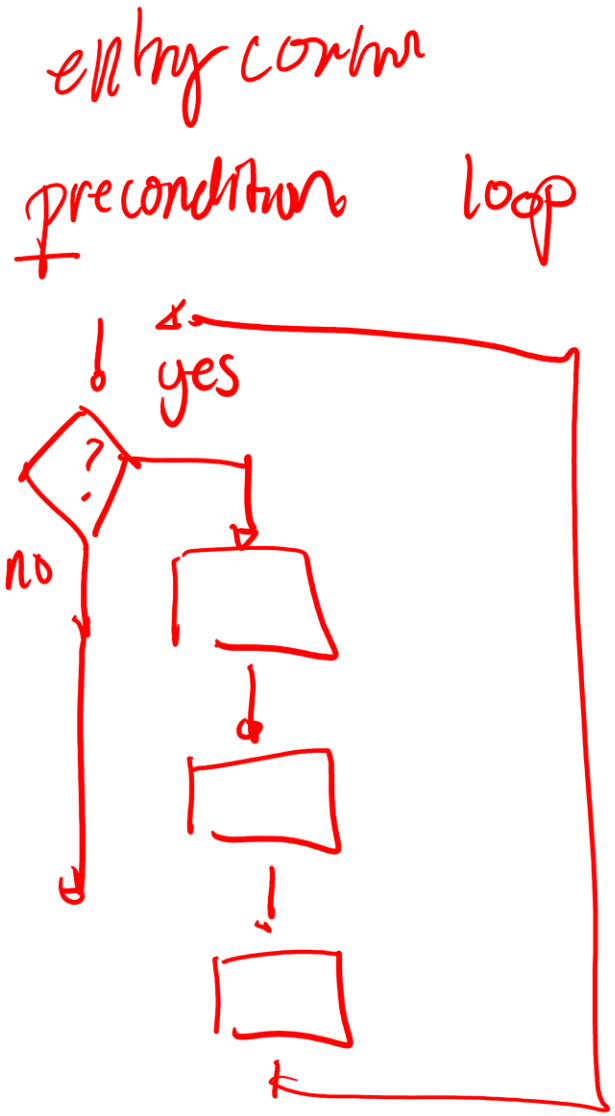
- A group of statements is **repeatedly executed**
- This group of statements is referred to as the **body of the loop**
- How many times? Well, it depends!
- Formally, the “times” is known as **iteration**

Repetition Structure /2

- It depends on a question
- **Important:** It will only repeat if the answer to the question is **yes**!
- In short, if the **condition is met**, *repeat* (cf. `if` statement)
- Otherwise, **stop**!

Repetition Structure /3





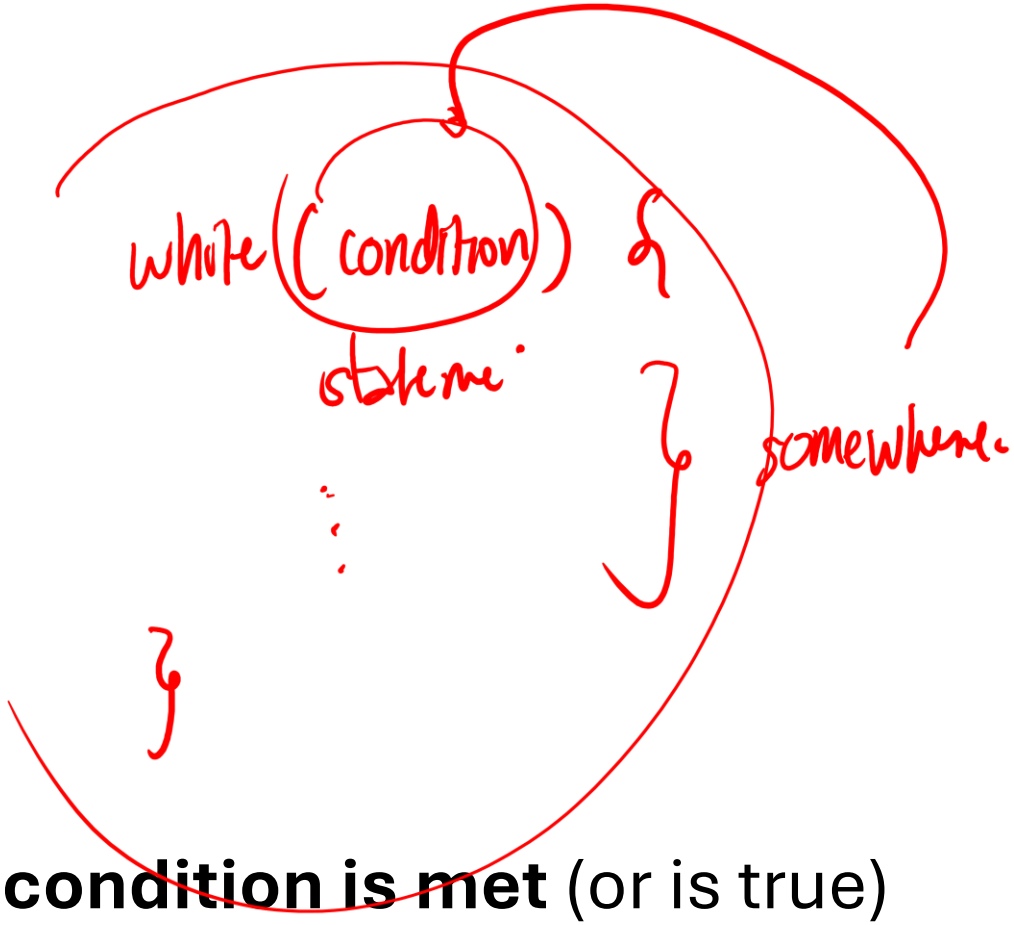
Postcondition vs. Precondition Loops

	Postcondition (Exit-controlled)	**Precondition (Entrance-controlled)**
When is the question asked?	after	before
Minimum number of times the statements are repeated	1	0

The `while` Statement

The syntax is:

```
while ( condition )
    statement;
```



Example of a **precondition loop**

Repeat the body of the loop while the **condition is met** (or is true)

Only stops when the condition is not true

Notes

- When dealing with loops, you must ensure that the condition will **eventually be false***
- This should be accomplished somewhere in the **body of the loop**
- Failure to do so could result to an **infinite loop**

Practice

Write a program that prompts the user to enter a nonnegative integer, **`N`**. The program should then print all the numbers from **`N`** down to 1 (inclusive), with each number displayed on its own line.

Sample Run

```
Enter Number: 4
4
3
2
1
```

num = 4

num > 0	num >1
num = 4 → 1	1
num = 3 → 1	1
num = 2 → 1	1
num = 1 → 1	0

Practice

Write a program that prompts the user to enter a nonnegative integer, `N`. The program should then print all the numbers from 1 through `N` (inclusive), with each number shown on its own line.

Sample Run

```
Enter Number: 4
1
2
3
4
```

Discussion

By starting with simple count-up and count-down programs, you can build many other types of programs

The key point to notice is the **behavior**: regardless of whether the loop counts up or down, the body still executes exactly **`N`** times

Practice

Write a program that prompts the user to enter a nonnegative integer, **N**. The program should then print **N** lines, starting from 1. Each line should display the line number followed by **Hello World**.

Sample Run

```
Enter Number: 4
1: Hello World
2: Hello World
3: Hello World
4: Hello World
```

Caution

- Common issue is to be off by an iteration (e.g., **off-by-one**)

- Caused by either incorrect **initialization** OR **condition**

Discussion /1

- We have seen various examples of a loop
- Notice how many times the **body of the loop** is repeated?
- It is somewhat *known*
- Those were examples of **counter-controlled loop**

Discussion /2

- We updated the variable `i` at each iteration to ensure we don’t end with an **infinite loop**

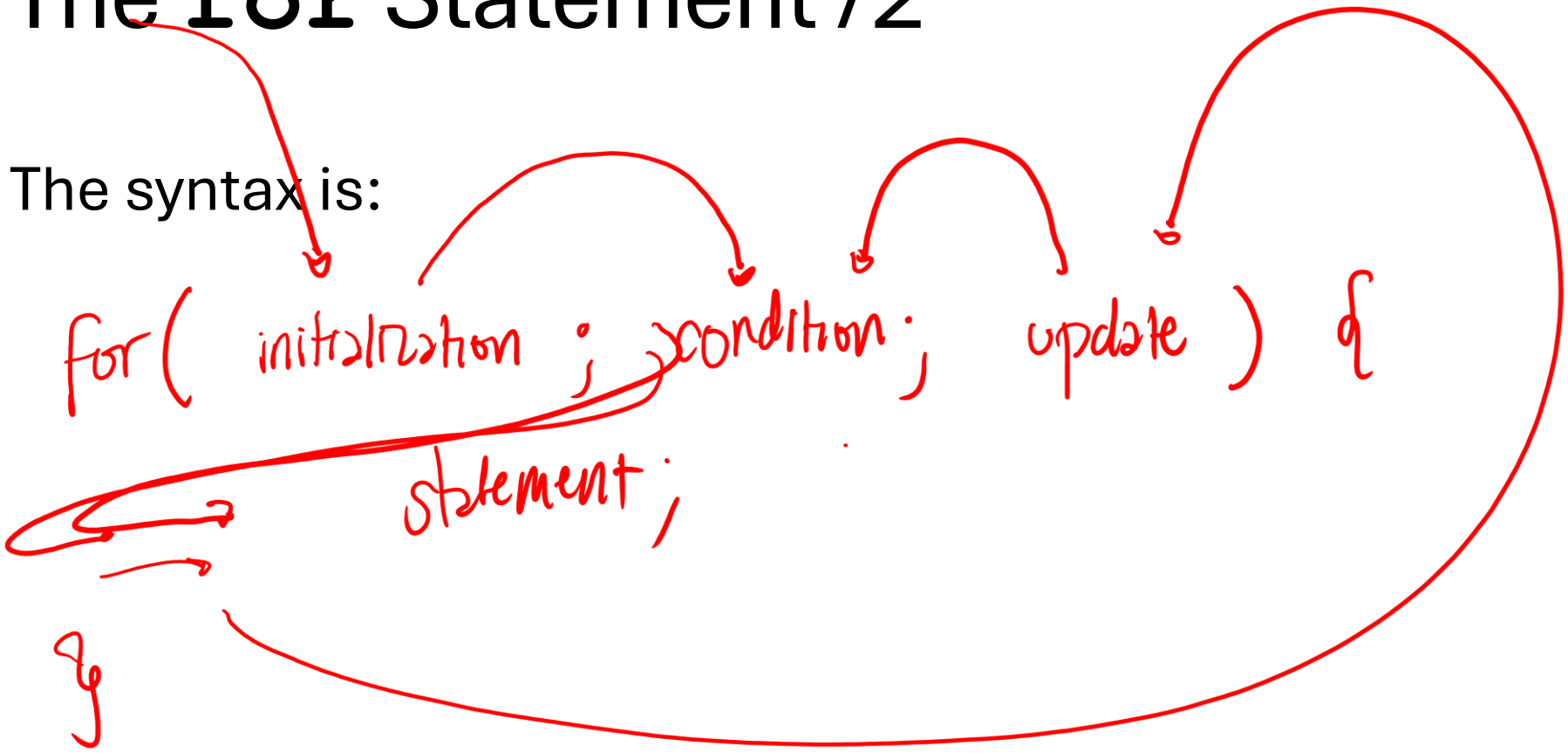
- Ensure you design your logic correctly!

The `for` Statement /1

- Often used for **counter-controlled** loops
- Example of a **precondition loop**
- Can easily convert from `while` to `for` loops, and vice versa
- Different from Python's for loop!

The `for` Statement /2

The syntax is:



For now, declare all the variables at the top portion of your **`main()`**

Discussion

What is the flow of control in a **`for`** statement?

Revisiting: Practice

Write a program that prompts the user to enter a nonnegative integer, **`N`**. The program should then print **`N`** lines, starting from 1. Each line should display the line number followed by **Hello World**.

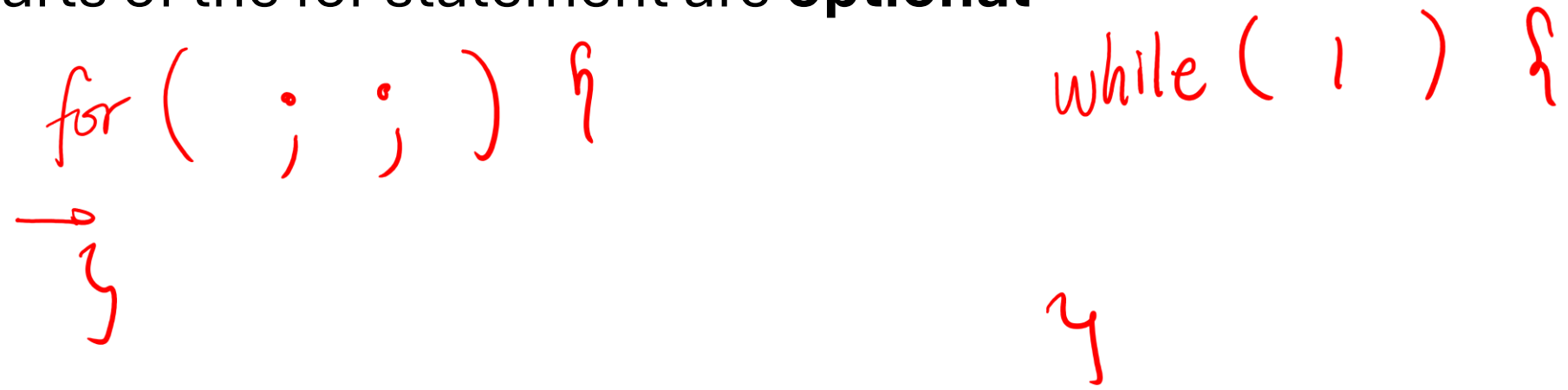
Solve this problem using a **`for`** loop.

Sample Run

```
Enter Number: 4
1: Hello World
2: Hello World
3: Hello World
4: Hello World
```

Notes

The 3 parts of the for statement are **optional**



However, the three ; (**semicolon**) are still required

We often use the shortcuts for **increment** (++) and **decrement** (−−)

Discussion

The body of a loop can also include other control structures, such as selection statements (e.g., if/else).

Practice

Write a program that prompts the user to enter a nonnegative integer, **N**. The program should then print all the even numbers from 1 through **N** (inclusive), with each number shown on its own line.

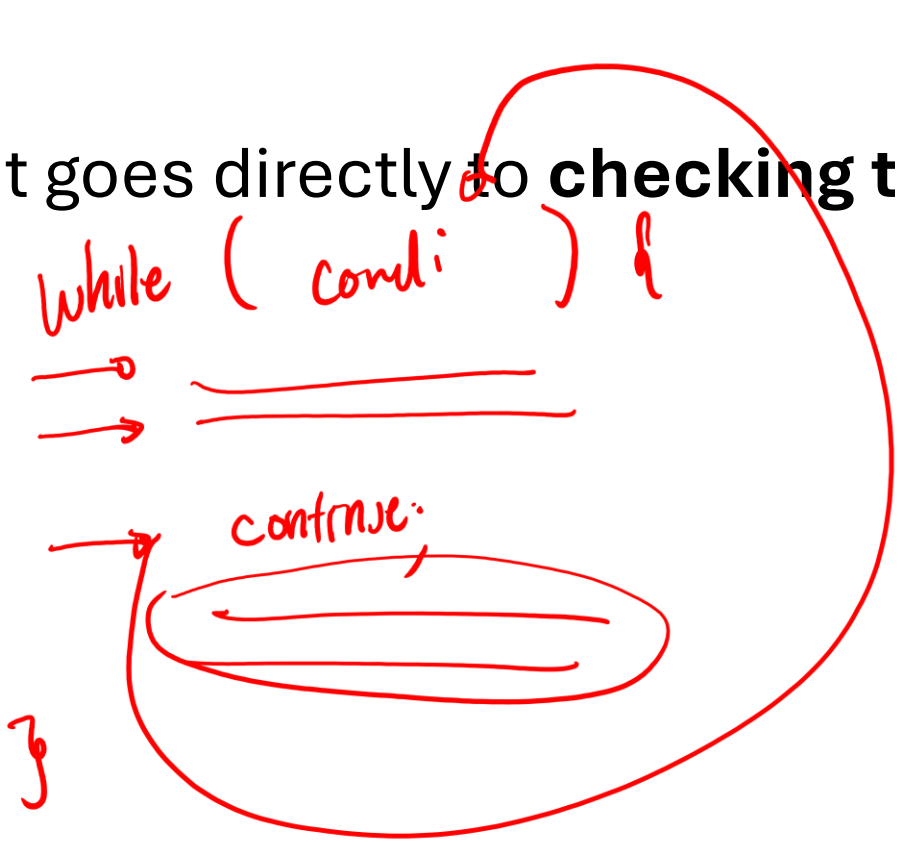
Sample Run

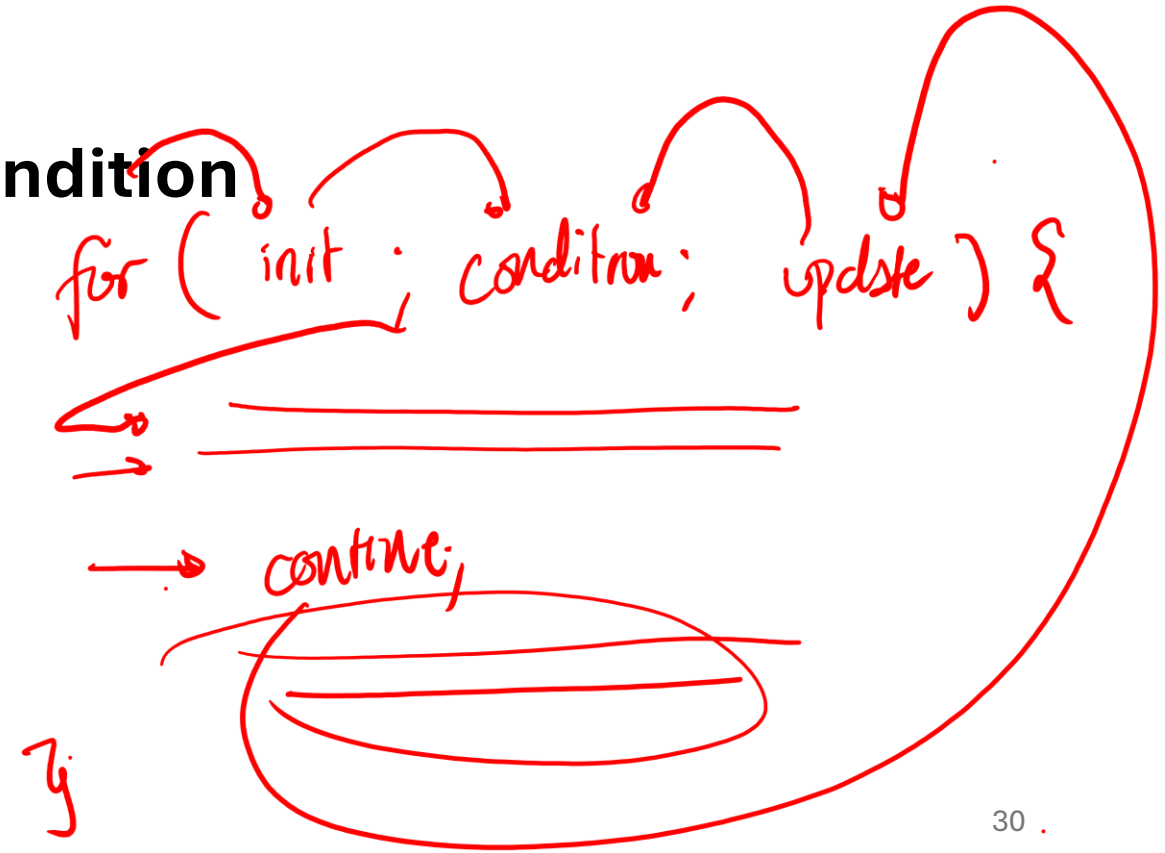
```
Enter Number: 4
2
4
```

The `continue` Statement

- Special keyword in C that **skips any remaining statements** for the current iteration (i.e., *jump*)

- It goes directly to **checking the condition**





Revisiting: Practice

Write a program that prompts the user to enter a nonnegative integer, **`N`**. The program should then print all the even numbers from 1 through **`N`** (inclusive), with each number shown on its own line.

Solve this using the **`continue`** statement.

Notes

- Any statements **within the loop** that are after the **`continue`** statement **will be skipped**

- If used with the **`while`** loop, it jumps to the **condition**

- If used with the **`for`** loop, it jumps to the **update**, then condition

Practice

Write a program that allows the user to enter an arbitrary number of nonnegative integers. The program should keep running until the user enters `-1`. For each number entered (other than `-1`), print its square on a new line.

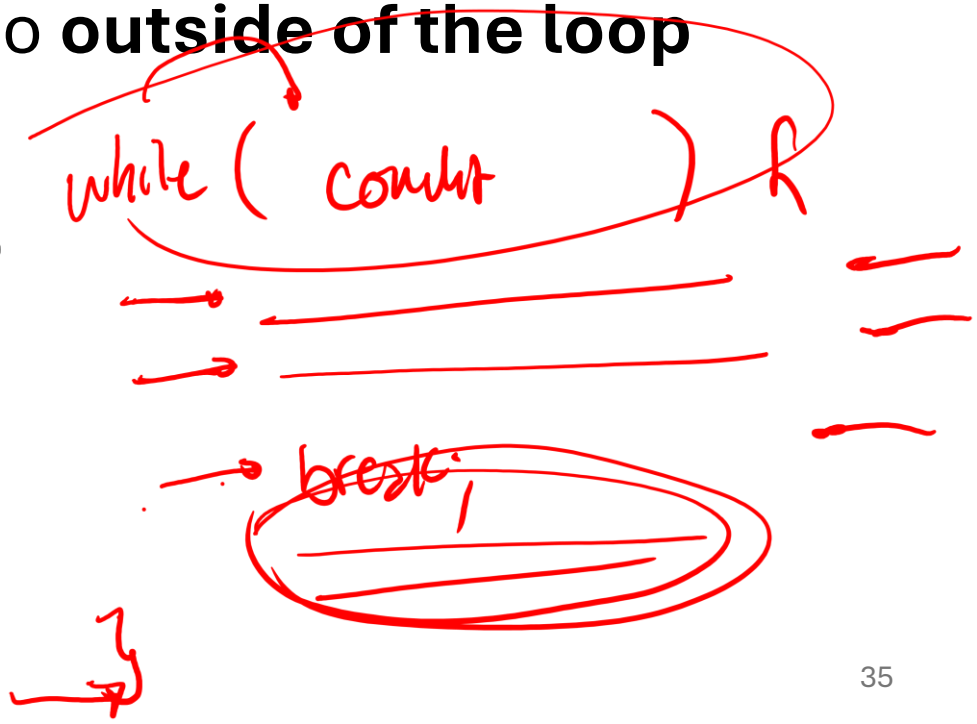
Sample Run

```
Enter Number: 2
4
Enter Number: 10
100
Enter Number: 3
9
Enter Number: -1
```

sentinel

The **`break`** Statement

- Special keyword in C that **skips any remaining statements** for the current iteration (i.e., *jump*)
- Unlike continue, control goes directly to **outside of the loop**
- In short, we **immediately exit** the loop



Discussion

- We saw another kind of a loop
- The number of times the body of the loop is executed is *unknown*
- A **special value** eventually stops the loop
- This is an example of a **sentinel-controlled loop**

Discussion

A common programming pattern is to accumulate values, often based on user input

For example, the sum pattern involves reading multiple numbers and keeping a running total until all inputs have been processed

Some other patterns include count, min/max, and average

Your Turn!

Write a program that repeatedly asks the user to enter whole numbers. The input continues until the user enters `-1`, which signals the end of input. Afterward, calculate and display the sum of all the numbers entered (excluding the `-1`).

Sample Run

```
Enter #1: 10
Enter #2: 20
Enter #3: 30
Enter #4: -1
Sum: 60
```

Your Turn!

Write a program that first asks the user to enter a number **`N`**. Then, prompt the user to input **`N`** whole numbers. Finally, calculate and display the sum of those **`N`** numbers.

Sample Run

```
Enter N: 3
Enter #1: 10
Enter #2: 20
Enter #3: 30
Sum: 60
```

Discussion

- Some problems are easier to solve when you use **`continue`** or **`break`** (i.e., natural translation)

- Use them carefully and do not overuse

- Always keep in mind how these statements change the flow of control inside a loop

Challenge

Write a program that first asks the user to enter a number **N**. Then, prompt the user to input **N** distinct positive whole numbers. Finally, print the greatest number found.

```
highest = 0;
for(                    ) {
    // num is provided by user
    if ( num > highest) {
        highest = num;

    }
}
```

Sample Run

```
Enter N: 3
Enter #1: 10
Enter #2: 30
Enter #3: 20
Max: 30
```

Notes

- The previous problem is an example of a **find max problem**
- We leveraged the assumption that numbers are positive
- What happens if this assumption is removed?

Challenge

Write a program that repeatedly asks the user to enter whole numbers. The input continues until the user enters `-1`, which signals the end of input. Afterward, print the greatest number found (excluding the `-1`). You may assume that the user will enter distinct positive whole numbers.

Sample Run

```
Enter #1: 10
Enter #2: 30
Enter #3: 20
Enter #4: -1
Max: 30
```

Code Tracing /1

```
int i, N = 4;

for(i = 0; i < N; i++)
    printf("%d ", N-i);
```

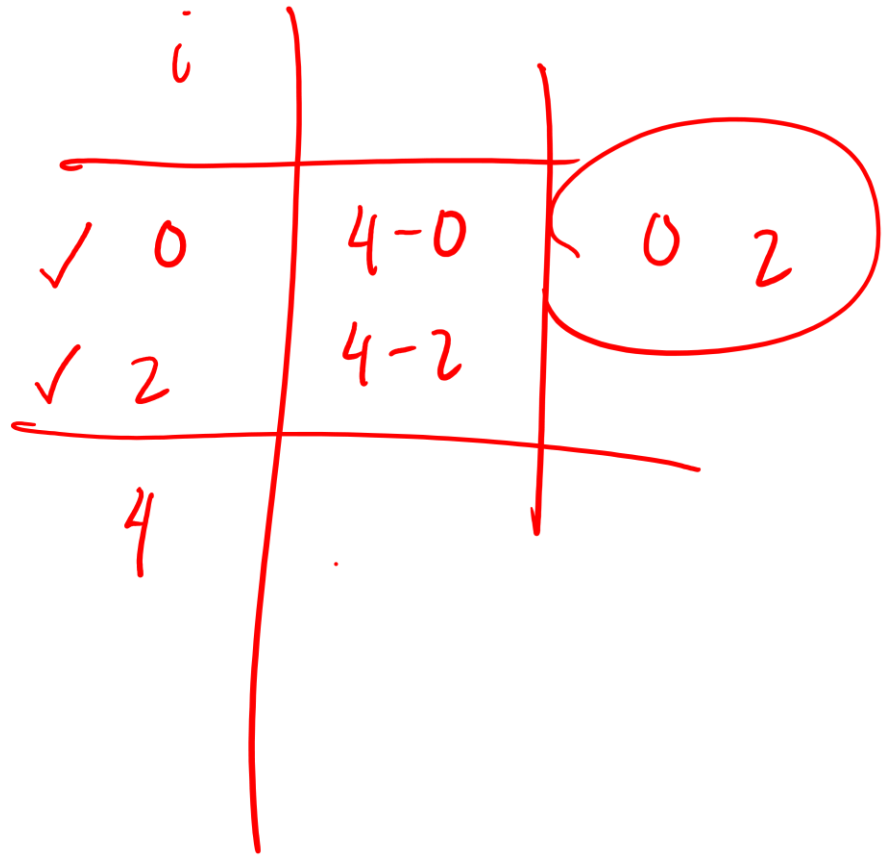
0 1 2 3

4-0 4-1 4-2 4-3

4 3 2 1

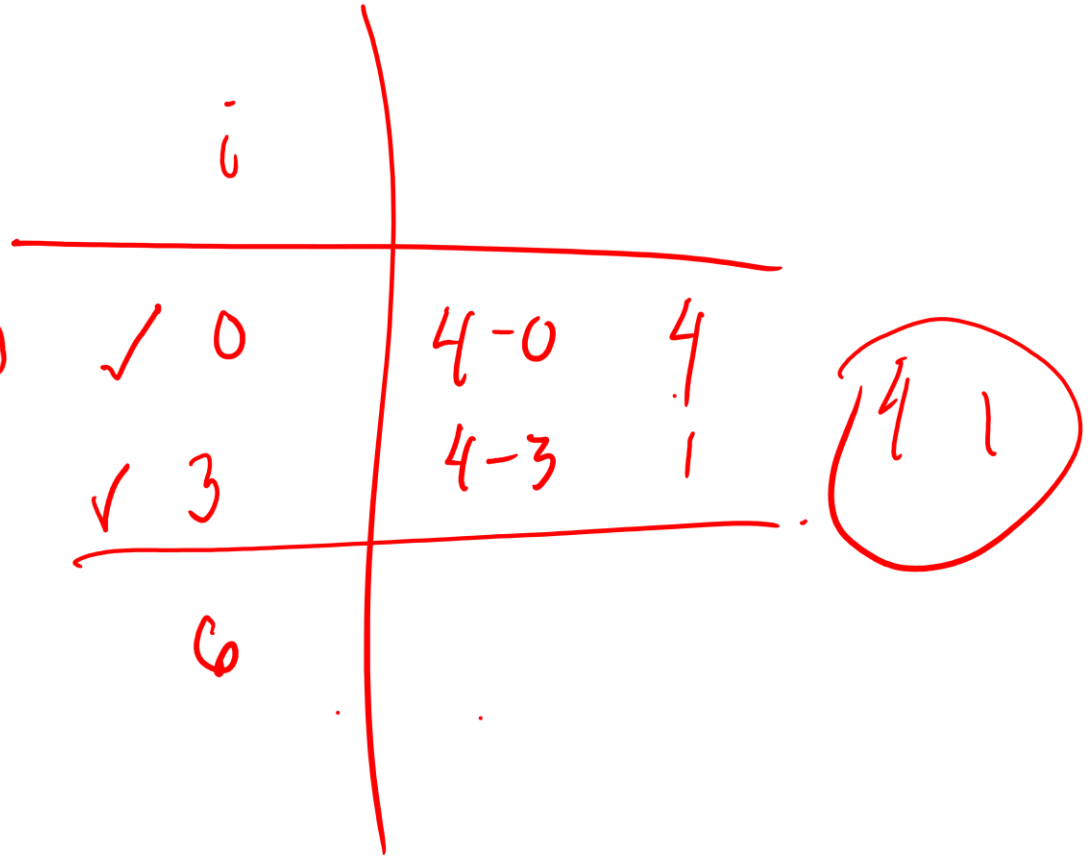
Code Tracing /2

```
int i, N = 4;

for(i = 0; i < N; i = i + 2)
    printf("%d ", N-i);
```



Code Tracing /3

```
int i, N = 4;

for(i = 0; i < N; i += 3)
    printf("%d ", N-i);
```



Code Tracing /4

```
int i, N = 4;

for(i = 0; i < N; )
    printf("%d ", N-i);
```

Practice

Write a program that prompts the user to enter a nonnegative integer, **N**. The program should then print **N** lines. Each line should display a single * (asterisk).

Sample Run

```
Enter Number: 4
*
*
*
*
```

```
for (i = 1 ; i <= N; i++)
    printf("*\n");
```

Discussion

- Which loop would you prefer using for this problem?
- The **`while`** or the **`for`**?
- Why?

Practice

Write a program that prompts the user to enter a nonnegative integer, **N**. The program should then print **N** lines. Each line should display 4 * (asterisks).

Sample Run

```
Enter Number: 4
****
****
****
****
```

Practice

Write a program that prompts the user to enter a nonnegative integer, **`N`**. The program should then print **`N`** lines. Each line should display **`N`** **`*`** (asterisks).

Sample Run

```
Enter Number: 5
*****
*****
*****
*****
*****
```

Nested Loops

- It is possible to have a **loop within a loop**
- Often refer to as **outer** and **inner** loops
- Not limited to a loop within a loop!
- Ensure you know how the program executes these

Code Tracing

```
int i, j, N = 4;

for(i = 0; i < N; i++)
    for(j = i; j < N; j++)
        printf("%d ", i*j);
```

i	j	i * j
0	0	0
0	1	0
0	2	0
0	3	0
	4	
1	1	1
1	2	2

0 0 0 0 1 2

Discussion /1

When **`break`** is encountered inside a loop, it immediately terminates the loop, skipping the rest of the body and exiting the loop entirely

When **`continue`** is encountered inside a loop, it skips the rest of the current iteration and jumps to the next condition check (or the update step in a **`for`** loop)

Discussion /2

In **nested loops**, the **`break`** and **`continue`** statements only affect the loop in which they are written

If written inside an **inner loop**, they apply only to that inner loop

If written at the **outer loop** level, they apply to the outer loop

Your Turn!

Write a program that prompts the user to enter two nonnegative integers, **`N`** and **`M`**. Afterward, display a matrix of **`*`** (asterisk) with dimension of **`N`** by **`M`** (row by column).

Sample Run

```
Enter Numbers: 5 10
**********
**********
**********
**********
**********
```

Challenge

Write a program that asks the user to enter a positive integer, **N**. This number represents the height of a left-aligned right triangle. Your program should then print the triangle using * (asterisks), where the row number determines how many asterisks appear on that row.

Sample Run

```
Enter Number: 5
*
**
***
****
*****
```

Notes

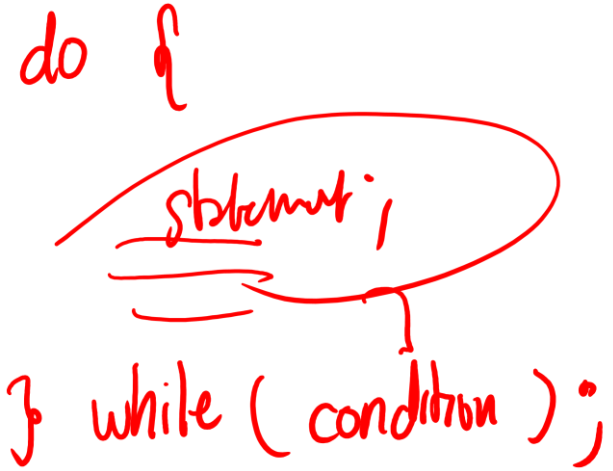
- It looks like it is just a variation of the **`N`** x **`N`** matrix
- However, instead of printing **`N`** asterisks per row, we print **`R`** asterisks where **`R`** is the row number
- So, we simply update the inner loop's condition

The `do...while` Loop

Example of a **postcondition loop**

Often associated with scenarios involving **data validation**

The syntax is:



Practice

Write a program that asks the user to enter a **positive whole number**, `N`. If the user enters an invalid number (i.e., not positive), ask the user to enter again. Keep doing so until the user enters a valid one.

Sample Run

```
Enter Number: -5
Enter Number: 0
Enter Number: 10
```

Your Turn!

- Write a program that asks the user to enter a **whole number**, which we assume as the **passcode**.

- If the user enters the **correct passcode** (`1234`), display "Welcome".

- Otherwise, display "Try Again!" and ask the user to enter again.

Final Notes /1

- We have seen 3 loops supported by C
- Depending on the problem, one loop is **more convenient** to use
- For now, we declared all the variables at the beginning of `main()`

Final Notes /2

The **`while`** and **`for`** loops are for pre-condition loops

The **`do-while`** is for post-condition loops

Challenge

Write a program that asks the user to enter a whole number. Afterward, print the sum of its digits.

Sample Run

```
Enter Number: 214
Sum is 7
```

Sample Run (Simplified)

```
Enter Number: 123
Sum is 6
```

Questions?