Learning Objectives: While Loops

- $\bullet \ \ Explain \ \hbox{while loop syntax} \\$
- Identify the causes of an infinite while loop
- Describe the break statement as it relates to a while loop

While Loops

While Loop Syntax

while loops, like for loops, use curly braces {} and indents for all commands that should be repeated. However, for loops generally contain 3 elements (an initialized variable, a boolean expression involving that variable, and a change in the value of that variable) while a while loop usually contains just a boolean expression. The for and while loops below produce the same results.

```
for (int i = 0; i < 5; i++) {
   cout << "Loop#: " << i <<
       endl;
}

int i = 0;
while (i < 5) {
   cout << "Loop# " << i <<
       i++;
}</pre>
```

Note that the variable declaration and initialization happen *before* the start of the while loop and any changes that occur to the variable happen *within* the body of the curly braces {}. On the other hand, everything happens in one step within parentheses () when using a for loop.

Here is another example of a while loop that prints Hello based on the value of the variable count.

```
int count = 5; // some random number set by user
while (count > 0) {
  cout << "Hello" << endl;
  count--;
}</pre>
```

Code Visualizer

challenge

What happens if you:

- Change the while statement to while (count > -1 * count)?
- Replace count -- in the code above with count = count 2?
- Change the while statement to while (count < 10)?

▼ How does while (count > -1 * count) work?

To understand how the loop works, it's best to substitute values in for the variable count. In the first iteration, we have while (5 > -1 * 5), this statement is true so the print command is executed. Since count gets decremented by 1 with each iteration, the while loop condition changes slightly every time like as follow:

```
while (4 > -1 * 4)
while (3 > -1 * 3)
while (2 > -1 * 2)
while (1 > -1 * 1)
```

Once the condition gets to while (0 > -1 * 0), it no longer holds true and the while loop ends. The result is 5 Hellos being printed to the screen.

Infinite Loops

Infinite loops are loops that do not have a test condition that causes them to stop. The following is a common mistake that results in an infinite loop:

```
int count = 5; // some random number set by user
while (count > 0) {
  cout << "Hello" << endl;
}</pre>
```

Since the variable count never gets decremented. It remains at 5, and 5 will forever be greater than 0, so the loop will never stop.

```
warning
```

Copy the code above and TRY IT to see what happens. C++ will eventually stop the loop due to an output limit, but it may take some time before this happens.

Why Use a While Loop?

If a while loop does the same thing as a for loop, then what is the purpose of having both? while loops are actually more useful when you are waiting for a certain event to occur. Imagine you are making a video game. The game should continue until the player loses all of their lives. You don't know how long this will take, so a while loop is more appropriate. On the other hand, if you have more specific loop parameters, a for loop will be better.

```
int player_lives = 3;
while (player_lives > 0) {
   // video game code
   // goes here
}
```

Turtle Coding: While Loop

Instead of a for loop, recreate the images below using a while loop.

▼ Turtle Graphics Review

- tina.forward(n) Where n represents the number of pixels.
- tina.backward(n) Where n represents the number of pixels.
- tina.right(d) Where d represents the number of degrees.
- tina.left(d) Where d represents the number of degrees.
- tina.pencolor({"COLOR"}) Where COLOR represents the track or line color you want tina to leave behind.
- tina.width(W) Where W represents how wide (in pixels) tina's track is.
- tina.shape("SHAPE") Where SHAPE represents the shape tina takes.
- tina.speed(SPEED) Where SPEED represents how fast tina moves

Challenge 1

.guides/img/TurtleChallenge1

▼ Hint

The pattern is still the same:

- 1. Go forward (creating a long line).
- 2. Make a right turn.
- 3. Go forward (creating a small line).
- 4. Make a right turn.
- 5. Go forward (creating another small line).
- 6. Make a right turn.
- 7. Go forward (creating a final small line).
- 8. Repeat steps #1 through #7 three more times for a total of **four** iterations.

However, a while loop usually contains only a boolean expression(s) in its header. Thus, you must initialize a **counting variable** *before* the start of the while loop. Also, that counting variable should be **incremented** *inside* the

body of the loop. The pattern you are trying to iterate is still the same:



Challenge 2

.guides/img/TurtleChallenge2

▼ Hint

Since a circle has 360 degrees, you will need a loop that repeats 360 times. Be careful about how far the turtle moves forward and turns. The circle can get very big, very quickly.

Challenge 3

>

.guides/img/TurtleChallenge3

▼ Hint

The pattern here is to move forward and make a right turn.



The trick lies within the fact that the distance the turtle moves has to get larger as the loop advances. Think of some operators that you can use to make the loop iterator variable get bigger during each iteration.

Here are some sample solutions using while loops:

```
tina.pencolor({"blue"});
tina.width(2);
tina.shape("arrow");
tina.speed(TS_SLOWEST);

int i = 0;
while (i < 4) {
   tina.forward(75);
   tina.right(90);
   tina.forward(25);
   tina.right(90);
   tina.forward(25);
   tina.forward(25);
   tina.forward(25);
   tina.forward(25);
   tina.forward(25);
   tina.forward(25);
   ina.forward(25);
   ina.forward(25);
```

```
tina.pencolor({"red"});
tina.width(2);
tina.shape("square");
tina.speed(TS_FASTEST);

int i = 0;
while (i < 360) {
   tina.forward(1);
   tina.right(1);
   i++;
}</pre>
```

```
tina.pencolor({"green"});
tina.width(2);
tina.shape("triangle");
tina.speed(TS_NORMAL);

int i = 10;
while (i <= 200) {
   tina.forward(i);
   tina.right(90);
   i+=10;
}</pre>
```

info

NOTE

In **most** cases, for loops and while loops can be used interchangeably. It is up to you to decide which one is better suited for your task.

Break Statement

Infinite Loops Are Bad, Right?

Well, that depends. If an infinite loop is caused because the counting variable isn't incremented, then it's a bug. However, some programmers purposely create a condition that will always evaluate to true. Therefore, the loop will always run. In such cases, a break statement is used to stop the loop at a particular point in the program.

Take a look at the following program (also shown in the text editor on the left).

```
#include <iostream>
using namespace std;

int main() {

    srand(time(NULL)); // start randomizer every time program runs
    while (true) {
        cout << "This is an infinite loop" << endl;
        int randNum = rand() % 100 + 1; // generate random number
            between 1 and 100

    if (randNum > 75) {
        cout << "The loop ends" << endl;
        break; // stop the loop
    } // end if condition
    } // end while loop

    cout << "The program ends" << endl;
    return 0;
}</pre>
```

Then click on the TRY IT button below *a few times* to run the code and see the resulting outputs. You can also click on the ++Code Visualizer++ link below to see how the code runs behind-the-scenes.

Code Visualizer

Even though while (true) will always evaluate as a true statement, the loop never becomes infinite because of the break statement.

challenge

What happens if you:

- Remove break; from the program?
- Add break; to the line before cout << "The loop ends" << endl;?

Code Visualizer

Comparing While Loops

The while loops introduced on the previous pages look different from the while loop covered on this page; however, they both have the same components and behave similarly.

```
int count = 5;
while (count > 0) {
  cout << "Hello" << endl;
  count--;
}

srand(time(NULL));
while (true) {
  cout << "This is an infinite loop" << endl;
  int randNum = rand() % 100 + 1;
  if (randNum > 75) {
    cout << "The loop ends" << endl;
    break;
  }
}

cout << "The program ends" << endl;</pre>

Variable to be tested

Test to end
the loop

Change the variable
```

.guides/img/WhileLoopComparison