Pattern 1.11

At this point, you understand for-loops. We will look at many common uses of for-loops and nested for-loops.

A single loop

We will look at different examples of a single for-loop here:

1. Loop with k taking values from 0 to $n - 1$:

   ```
   for ( int k{0}; k < n; ++k ) {
       // loop body
   }
   ```

2. Less seldom, loop with k taking values from 1 to n:

   ```
   for ( int k{1}; k <= n; ++k ) {
       // loop body
   }
   ```

3. Loop with k taking on values from $n - 1$ to 0 in reverse order:

   ```
   for ( int k{n - 1}; k >= 0; --k ) {
       // loop body
   }
   ```

4. Less common, loop with k taking on values from n to 1 in reverse order:

   ```
   for ( int k{n}; k > 0; --k ) {
       // loop body
   }
   ```

5. Loop with k taking on values 0, 2, 4, 6, ... up to the largest even number less than n

   ```
   for ( int k{0}; k < n; k += 2 ) {
       // loop body
   }
   ```

6. Loop with k taking on values 1, 3, 5, 7, ... up to the largest odd number less than n

   ```
   for ( int k{1}; k < n; k += 2 ) {
       // loop body
   }
   ```

7. Loop with k taking on values n, $n - 2$, $n - 4$, $n - 6$, ... down to either 0 or 1, whichever has parity[1] with n

   ```
   for ( int k{n}; k >= 0; k -= 2 ) {
       // loop body
   ```

[1] Two numbers have the same *parity* if they are both even or they are both odd.

```
}
```

8. Loop with k taking powers of 2 (1, 2, 4, 8, 16, 32, ...) up to the highest power of two less than n:

```
for ( int k{1}; k < n; k *= 2 ) {
    // loop body
}
```

9. Loop with k starting with n, each time dividing the previous value by two and discarding any remainder, down to 1:

```
for ( int k{n}; k > 0; k /= 2 ) {
    // loop body
}
```

A pair of nested for-loops

A pair of nested for-loop has two loop variables, and inside the body of the inner for-loop, both loop variables are defined:

1. Loops with i taking values from 0 to $m - 1$, but for each value of i, j ttakes on values from 0 to $n - 1$ are very common:

```
for ( int i{0}; i < m; ++i ) {
    // Only 'i' is declared here

    for ( int j{0}; j < n; ++j ) {
        // nested loop body, with both 'i' and 'j' declared
    }

    // Again, only 'i' is declared here
}
```

To visualize what is happening, note that each entry of this matrix contains a pair (i, j), so the top-left corner is when $i = 0$ and $j = 0$, and immediate to the right is the pair (0, 1) when $i = 0$ and $j = 1$.

$$\begin{pmatrix} (0,0) & (0,1) & (0,2) & \cdots & (0,n-1) \\ (1,0) & (1,1) & (1,2) & \cdots & (1,n-1) \\ (2,0) & (2,1) & (2,2) & \cdots & (2,n-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (m-1,0) & (m-1,1) & (m-1,2) & \cdots & (m-1,n-1) \end{pmatrix}$$

The nested for-loop visits every pair starting with the top-left corner and going row-by-row from left to right.

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{0}; j < n; ++j ) {
        std::cout << "(" << i << "," << j << ") ";
    }
```

```
    std::cout << std::endl;
}
```

This may be used if you need to consider all possible pairs (i, j), and very often the upper bounds of the loop (m and n) will be the same, such as when you have a square matrix.

2. Loops with i taking values from 0 to $m - 1$, but for each value of i, j takes on values from i to $n - 1$ is also not uncommon, as you may observe from linear algebra.

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{i}; j < n; ++j ) {
        // nested loop body
    }
}
```

Like before, we are visiting pairs in the matrix, but we don't visit any pair in the strict lower triangular component (entries below the diagonal):

$$\begin{pmatrix} (0,0) & (0,1) & (0,2) & \cdots & (0,n-1) \\ & (1,1) & (1,2) & \cdots & (1,n-1) \\ & & (2,2) & \cdots & (2,n-1) \\ & & & \ddots & \vdots \\ & & & & \end{pmatrix}$$

The nested for-loop visits every pair starting with the top-left corner and going row-by-row from left to right, but only starting at the diagonal.

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{i}; j < n; ++j ) {
        std::cout << "(" << i << "," << j << ") ";
    }

    std::cout << std::endl;
}
```

When $m = n$, this will consider all pairs (i, j) when order doesn't matter.

3. Loops with i taking values from 0 to $m - 1$, but for each value of i, j takes on values from $i + 1$ to $n - 1$ is also not uncommon if there is no need to consider the cases (i, i):

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{i + 1}; j < n; ++j ) {
        // loop body
    }
}
```

Like before, we are visiting pairs in the matrix, but we only visit those entries in the strictly upper-triangular component (each index above the diagonal):

$$\begin{pmatrix} (0,1) & (0,2) & (0,3) & \cdots & (0,n-1) \\ & (1,2) & (1,3) & \cdots & (1,n-1) \\ & & (2,3) & \cdots & (2,n-1) \\ & & & \ddots & \vdots \\ & & & & \end{pmatrix}$$

The nested for-loop visits every pair starting with the pair (0, 1) and going row-by-row from left to right starting at an entry to the right of the diagonal.

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{i + 1}; j < n; ++j ) {
        std::cout << "(" << i << "," << j << ") ";
    }

    std::cout << std::endl;
}
```

When $m = n$, this will consider all pairs (i, j) when order doesn't matter and when we don't care about the case (i, i). For example, finding the cost to fly between n cities, assuming that the cost to fly from Toronto to Montreal is the same as flying from Montreal to Toronto.

4. Loops with i taking values from 0 to $m - 1$, but for each value of i, j takes on values from 0 to i is a reflection of a previous case:

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{0}; j <= i; ++j ) {
        // loop body
    }
}
```

Like before, we are visiting pairs in the matrix, but we only visit those entries in the lower-triangular component (all entries on or below the diagonal):

$$\begin{pmatrix} (0,0) & & & \\ (1,0) & (1,1) & & \\ (2,0) & (2,1) & (2,2) & \\ \vdots & \vdots & \vdots & \ddots \\ (m-1,0) & (m-1,1) & (m-1,2) & \cdots \end{pmatrix}$$

The nested for-loop visits every pair starting with the top-left corner and going row-by-row from left to right, but stopping at the diagonal:

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{0}; j <= i; ++j ) {
        std::cout << "(" << i << "," << j << ") ";
    }

    std::cout << std::endl;
}
```

5. Loops with i taking values from 0 to $m-1$, but for each value of i, j takes on values from 0 to $i-1$ is also a reflection of a previous case:

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{0}; j < i; ++j ) {
        // loop body
    }
}
```

Like before, we are visiting pairs in the matrix, but we only visit those entries in the strict lower-triangular component (all entries below the diagonal):

$$\begin{pmatrix} & & & \\ (1,0) & & & \\ (2,0) & (2,1) & & \\ (3,0) & (3,1) & (3,2) & \\ \vdots & \vdots & \vdots & \ddots \\ (m-1,0) & (m-1,1) & (m-1,2) & \cdots \end{pmatrix}$$

The nested for-loop visits every pair starting with the top-left corner and going row-by-row from left to right, but stopping at the diagonal:

```
for ( int i{0}; i < m; ++i ) {
    for ( int j{0}; j <= i; ++j ) {
        std::cout << "(" << i << "," << j << ") ";
    }

    std::cout << std::endl;
}
```

Summary

Most for-loops and nested for-loops are of the form above; for example, in linear algebra. More generally, a single for-loop is used when searching a range of values, while two nested for-loops are used when, for example, considering all possible pairs of the form (i, j).