Looping

Victor Eijkhout, Susan Lindsey

Fall 2019

'For' statement

Sometimes you need to repeat a statement a number of times. That's where the loop comes in. A loop has a counter, called a loop variable, which (usually) ranges from a lower bound to an upper bound.

Here is the syntax in the simplest case:

```
for (int var=low; var<upper; var++) {
  // statements involving var
  cout << "The square of " << var << " is " << var*var << endl;
}
```

Exercise 1

Read an integer value with `cin`, and print 'Hello world' that many times.

Loop syntax

- Loop variable is usually an integer.
- The stopping test be any test; can even be empty.
- The stopping test is performed at the start of each iteration.
- The increment can be a decrement or something like `var*=10`
- Any and all of initialization, test, increment can be empty:

  ```
  for(;;) loop_body;
  ```

- (The loop variable can be defined outside the loop:

  ```
  int var;
  for (var=low; var<upper; var++) {
  ```

  but it's cleaner to make it local.)
- Loop body is a single statement or a block.

Project Exercise 2

Read an integer and determine whether it is prime by testing for the smaller numbers whether they are a divisor of that number.

Print a final message

```
Your number is prime
```

or

```
Your number is not prime: it is divisible by ....
```

where you report just one found factor.

Review quiz 1

For each of the following loop headers, how many times is the body executed? (You can assume that the body does not change the loop variable.)

```
for (int i=0; i<7; i++)

for (int i=0; i<=7; i++)

for (int i=0; i<0; i++)
```

What is the last iteration executed?

```
for (int i=1; i<=2; i=i+2)

for (int i=1; i<=5; i*=2)

for (int i=0; i<0; i--)

for (int i=5; i>=0; i--)

for (int i=5; i>0; i--)
```

Popular increments

- `i++` for a loop that counts forward;
- `i--` for a loop that counts backward;
- `i+=2` to cover only odd or even numbers, depending on where you started;
- `i*=10` to cover powers of ten.

Nested loops

Traversing a matrix:

```
for (int row=0; row<m; row++)
  for (int col=0; col<n; col++)
    ...
```

This is called 'loop nest', with
row: outer loop
col: inner loop.

Exercise 3

Write an i, j loop nest that prints out all pairs with

$$1 \leq i, j \leq 10, \quad j \leq i.$$

Output one line for each `i` value.

Now write an i, j loop that prints all pairs with

$$1 \leq i, j \leq 10, \quad |i - j| < 2,$$

again printing one line per `i` value. Food for thought: this exercise is definitely easiest with a conditional in the inner loop, but can you do it without?

Optional exercise 4

Find all triples of integers u, v, w under 100 such that $u^2 + v^2 = w^2$. Make sure you omit duplicates of solutions you have already found.

Indefinite looping

Sometimes you want to iterate some statements not a predetermined number of times, but until a certain condition is met. There are two ways to do this.

First of all, you can use a 'for' loop and leave the upperbound unspecified:

```
for (int var=low; ; var=var+1) { ... }
```

Break out of a loop

This loop would run forever, so you need a different way to end it. For this, use the *break* statement:

```
for (int var=low; ; var=var+1) {
  statement;
  if (some_test) break;
  statement;
}
```

Where did the break happen?

Suppose you want to know what the loop variable was when the break happened. You need the loop variable to be global:

```
int var;
... code that sets var ...
for ( ; var<upper; var++) {
  ... statements ...
  if (some condition) break
  ... more statements ...
}
... code that uses the breaking value of var ...
```

In other cases: define the loop variable in the header!

Test in the loop header

If the test comes at the start or end of an iteration, you can move it to the loop header:

```
bool some_test{false};
for (int var=low; !some_test ; var++) {
  ... some code ...
  some_test = ... some condition ...
}
```

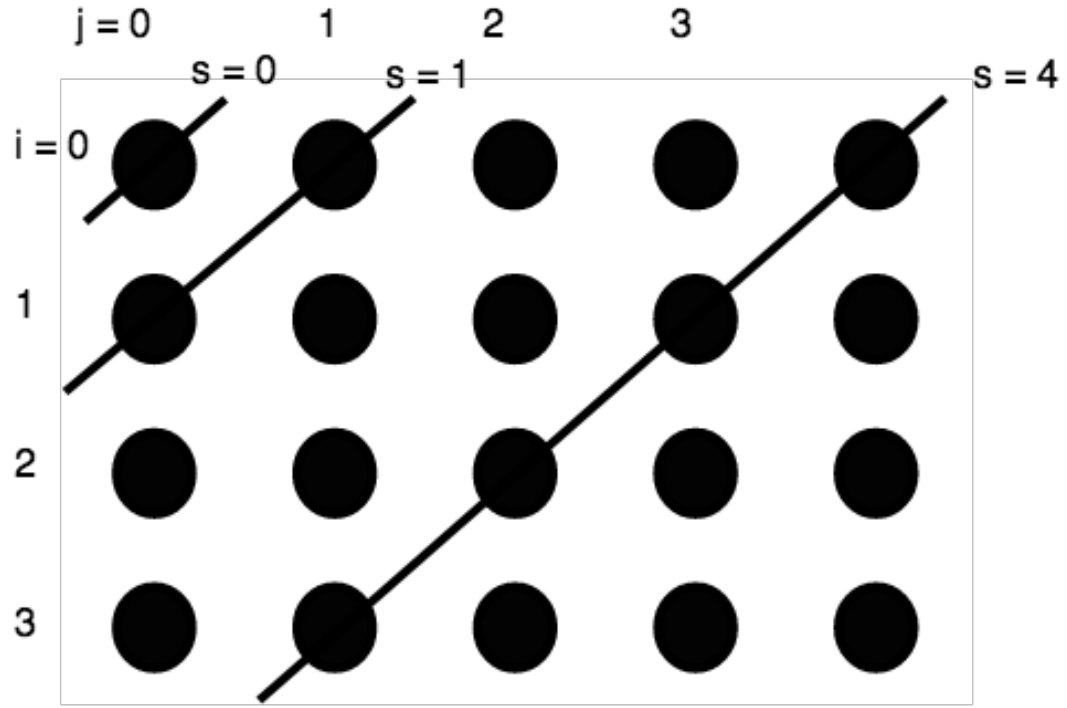
Exercise 5

Write a double loop over $0 \leq i, j < 10$ that prints the first pair where the product of indices satisfies $i \cdot j > N$, where N is a number your read in. A good test case is $N = 40$.

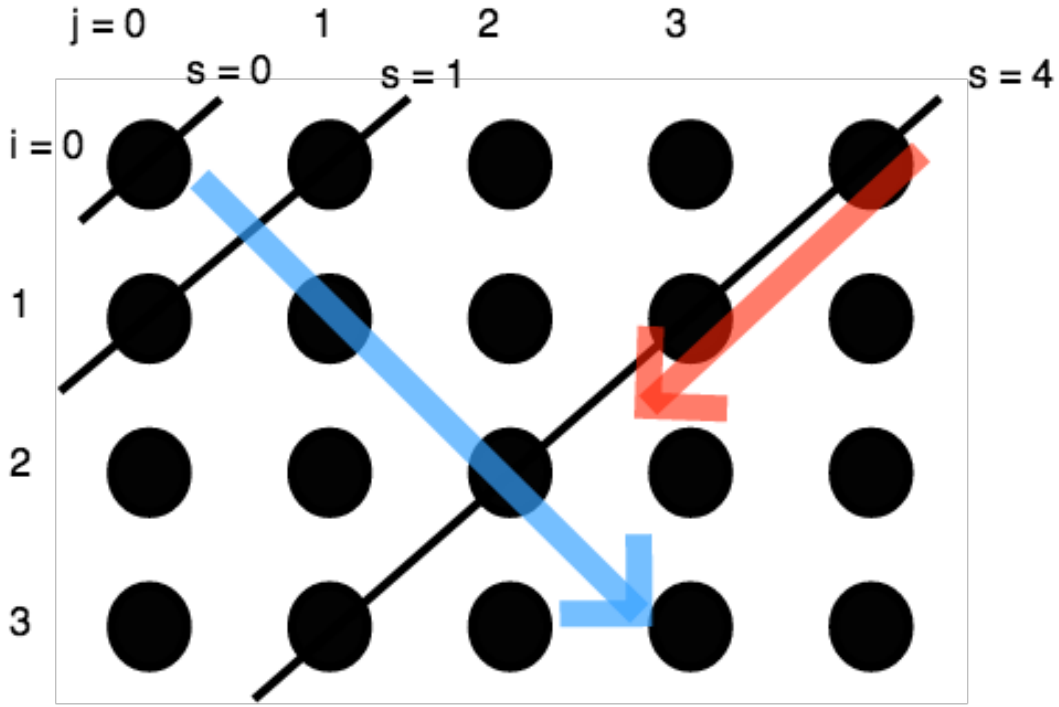
Secondly, find pair with $i \cdot j > N$, but with the smallest value for $i + j$. Can you traverse the i, j indices such that they first enumerate all pairs $i + j = 1$, then $i + j = 2$, then $i + j = 3$ et cetera? Hint: write a loop over the sum value $1, 2, 3, \ldots$, then find i, j.

You program should print out both pairs, each on a separate line, with the numbers separated with a comma, for instance `8,5`.

Suggestive picture 1



Suggestive picture 2



Turn it in!

- If you have compiled your program, do:

  `sdstestij yourprogram.cc`

  where '`yourprogram.cc`' stands for the name of your source file.
- Is it reporting that your program is correct? If so, do:

  `sdstestij -s yourprogram.cc`

  where the `-s` flag stands for 'submit'.

Skip iteration

```
for (int var=low; var<N; var++) {
  statement;
  if (some_test) {
    statement;
    statement;
  }
}
```

Alternative:

```
for (int var=low; var<N; var++) {
  statement;
  if (!some_test) continue;
  statement;
  statement;
}
```

The only difference is in layout.

While loop

The other possibility for 'looping until' is a *while* loop, which repeats until a condition is met.

Syntax:

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

or

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

The while loop does not have a counter or an update statement; if you need those, you have to create them yourself.

While syntax 1

```
cout << "Enter a positive number: " ;
cin >> invar;
while (invar>0) {
  cout << "Enter a positive number: " ;
  cin >> invar;
}
cout << "Sorry, " << invar << " is negative" << endl;
```

Problem: code duplication.

While syntax 2

```
do {
  cout << "Enter a positive number: " ;
  cin >> invar;
} while (invar>0);
cout << "Sorry, " << invar << " is negative" << endl;
```

The post-test syntax leads to more elegant code.

Exercise 6

The integer sequence

$$u_{n+1} = \begin{cases} u_n/2 & \text{if } u_n \text{ is even} \\ 3u_n + 1 & \text{if } u_n \text{ is odd} \end{cases}$$

leads to the Collatz conjecture: no matter the starting guess u_1, the sequence $n \mapsto u_n$ will always terminate at 1.

$$5 \to 16 \to 8 \to 4 \to 2 \to 1$$

$$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \cdots$$

(What happens if you keep iterating after reaching 1?)

Try all starting values $u_1 = 1, \ldots, 1000$ to find the values that lead to the longest sequence: every time you find a sequence that is longer than the previous maximum, print out the starting number.

Optional exercise 7

A horse is tied to a pole with a 1 meter elastic band. A spider that was sitting on the pole starts walking to the horse over the band, at 1cm/sec. This startles the horse, which runs away at 1m/sec. Assuming that the elastic band is infinitely stretchable, will the spider ever reach the horse?