Nested loops, break, continue

- added loop control: break, continue
- loops within loops (nested loops)
- indentation: functions, loops, if/else, etc
- common loop errors/issues

Added loop control: break

- break allows us to exit a loop early, before the next condition check
- usually placed in an if statement, so if some-special-condition is true then break out of loop
- as a style/habit it is somewhat avoided
  - creates multiple potential exit points from the loop instead of just one
  - tends to be more difficult to read/maintain code where break statements embedded inside the loop body
  - preferred approach is to have the loop exit/repeat logic obvious in the loop test itself

break example

```
// get 10 values from user, compute sum
// quit early if user enters a 0
int N = 10;
int sum = 0;

while (N > 0) {
    int x;
    cout << “Enter an int”;
    cin >> x;
    if (x == 0) {
        break;
    }
    sum = sum + x;
    N--;
}

cout << sum << endl;
```

Added loop control: continue

- allows code to skip the lower part of the loop body, skipping ahead to the next time the condition check occurs
- similar style as break
- similar concerns with respect to obscuring loop functionality

Continue example

```
// get 10 values from user, compute sum
// don't count negative entries
int N = 10;
int sum = 0;

while (N > 0) {
    int x;
    cout << “Enter an int”;
    cin >> x;
    if (x < 0) {
        continue;
    }
    sum = sum + x;
    N--;
}

cout << sum << endl;
```

Nested loops

- can declare one loop inside another (any types)
- the inner loop must be completely within the body of the outer loop
- the inner loop runs to completion (likely multiple times) each time you make one pass through the body of the outer loop

Nesting example

- print 5 lines of output, on each line print values 1..10
- outer loop controls which of the 5 lines we're printing, inner loop controls printing the 10 values for the current line

```
for (int line = 1; line <= 5; line++) {
    cout << “Line “ << line << “: “;
    for (int n=1; n<=10; n++) {
        cout << “ “ << n;
    }
    cout << endl;
}
```

```
Line 1: 1 2 3 4 5 6 7 8 9 10
Line 2: 1 2 3 4 5 6 7 8 9 10
Line 3: 1 2 3 4 5 6 7 8 9 10
Line 4: 1 2 3 4 5 6 7 8 9 10
Line 5: 1 2 3 4 5 6 7 8 9 10
```

Indentation habits

- global items lined up at left hand margin (functions, main, global variables/constants, #includes, etc)
- code within the body of a function/main is indented (eg 4 spaces)
- code within a control structure is indented an additional 4 spaces
- indentation increases with each layer of control structure, e.g. inside an if statement that is inside a loop we'll indent 12 spaces (4 for the function, 4 for the loop, 4 for the if)
- thus visually obvious where control structures begin/end

Common loop errors

- infinite loops
  - (forgetting to update the variables used in your condition test, or the condition check isn't quite appropriate to guarantee termination)
- off-by-one errors
  - running body of the loop one extra time or one time too few, usually just requires a tweak to the condition test (e.g. mixup between < and <=)
- scope errors
  - declaring a variable inside the loop when it should be outside, or having two variable scopes overlap
  - e.g. an x in the function outside the loop and another x inside the loop, mixing up which one is updating/visible where