ENGR 101 – Chapter 16

Iteration

Recall: Control Flow

- **Branching** and **iteration** are techniques for managing **control flow** in our programs.
  - The line of code that is currently executing is said to have "control".
- In particular, flowcharts are an effective tool for mapping out the **control flow** of our program design.
- **Control flow structures** like `if`, `for`, and `while` allow us to structure our code to follow the desired control flow.

while Loops

```
0 1 2 3 4 done!
```

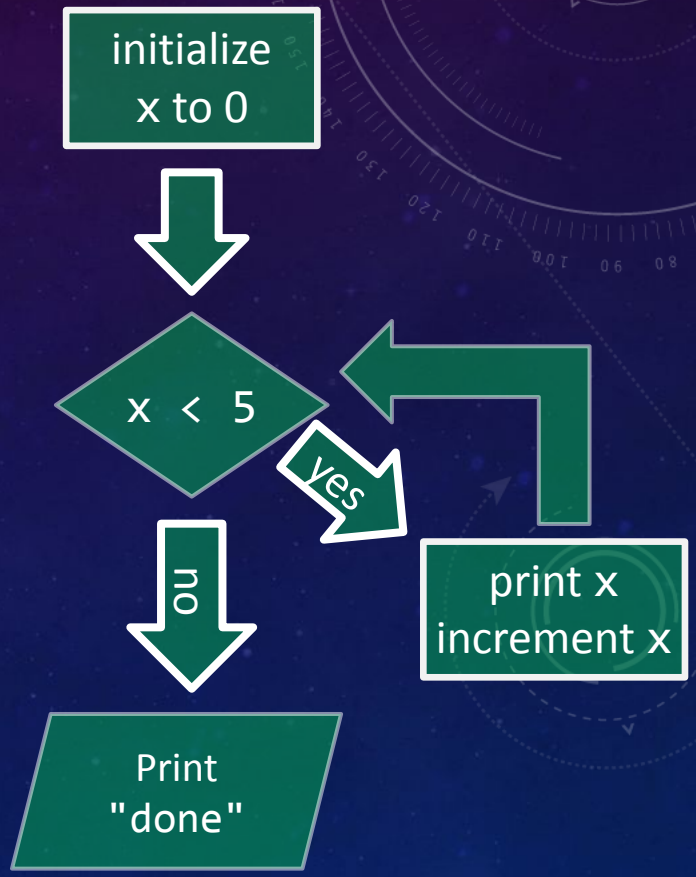
- while loops execute a block of code as long as some condition is true

```
int main() {
  int x = 0; // Start x at 0

  // keep going as long as x < 5
  while (x < 5) {
    // Print out the current value x
    cout << x << " ";

    // Increment x
    x = x + 1;
  }

  cout << "done!" << endl;
}
```



Counting from Zero

```
0 1 2 3 4 done!
```

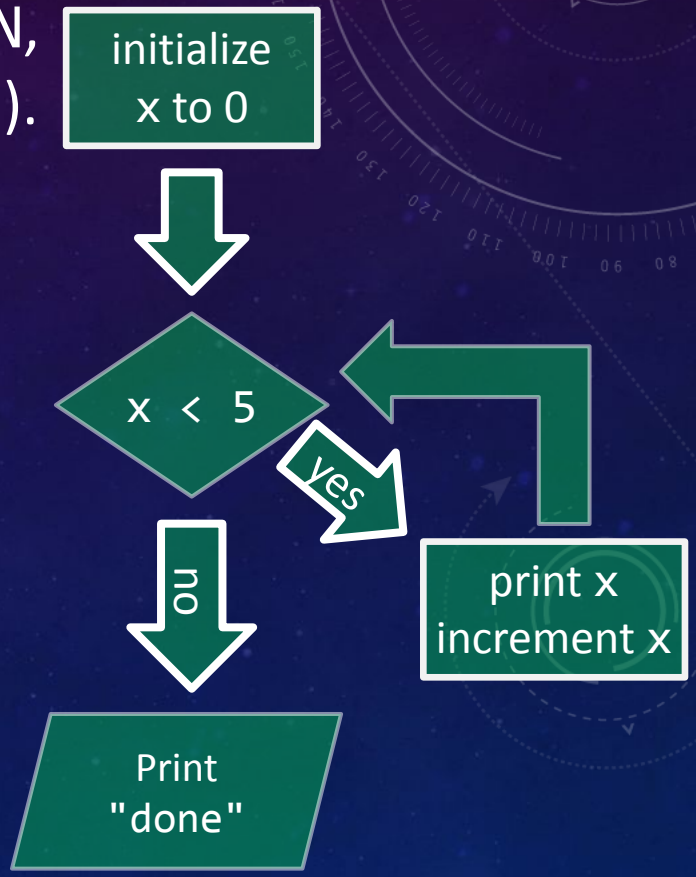
- If we start x at 0, and count while x < N, we iterate N times (with x as 0,1,…,N-1).

```
int main() {
  int x = 0; // Start x at 0

  // keep going as long as x < 5
  while (x < 5) {
    // Print out the current value x
    cout << x << " ";

    // Increment x
    x = x + 1;
  }

  cout << "done!" << endl;
}
```



Hint on Creating While Loops

- Some of the trickier parts of writing a correct `while` loop are:
  - Where does it start? (i.e. "what gets initialized before the loop starts? and to what value?")

```
int x = 0; // Start x at 0
```

  - When does it go? (i.e. "what goes in the parentheses?")

```
while (x < 5) { // keep going as long as x < 5
```

  - How do you get there? (i.e. "what's the increment or decrement?")

```
x = x + 1; // Increment x by 1 each time
```

- Answer these questions FIRST before you worry about the rest of what goes in the loop

3 min

Exercise: `while` Loops

```
9 7 5 3 1 done!
```

- Write code that uses a while loop to print out the first 5 odd numbers, in reverse order.

```
int main() {



  while (       ) {



  }

  cout << "done!" << endl;
}
```

Solution: while Loops

```
9 7 5 3 1 done!
```

- Write code that uses a while loop to print out the first 5 odd numbers, in reverse order.

```
int main() {
  int x = 9; // Start x at 9

  // keep going as long as x >= 1
  while (x >= 1) {
    // Print out the current value x
    cout << x << " ";

    // Decrement x by 2
    x = x - 2;
  }

  cout << "done!" << endl;
}
```

Increment and Decrement Operations

- C++ provides special operators for these common tasks:
  - **Increment** – Increase a variable by some amount

```
x = x + n;
```

```
x = x + 1;
```

```
x += n;
```

```
++x;
```

  - **Decrement** – Decrease a variable by some amount

```
x = x - n;
```

```
x = x - 1;
```

```
x -= n;
```

```
--x;
```

You can also write x++ or x--, but those technically do something a bit different. Prefer to use the ++x and --x versions.

while Loops

```
0 1 2 3 4 done!
```

- while loops execute a block of code as long as some condition is true

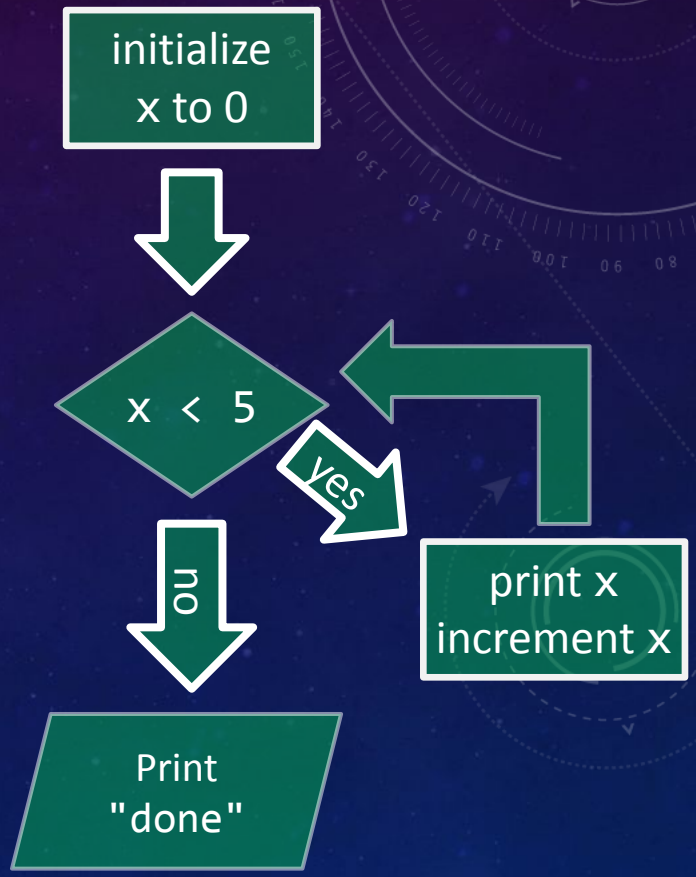
```
int main() {
  int x = 0; // Start x at 0

  // keep going as long as x < 5
  while (x < 5) {
    // Print out the current value x
    cout << x << " ";

    // Increment x
    x = x + 1;
  }

  cout << "done!" << endl;
}
```

++x;



for Loops

- Many loops we write fall into a particular pattern:

```
int main() {
  int x = 0;

  while (x < 5) {

    cout << x << " ";

    ++x;
  }
  cout << "done!" << endl;
}
```

1. Initialize
2. Condition
3. Body
4. Increment

Step 1 is performed **only once**. Steps 2-4 are **repeated** until the condition is `false`.

for Loops

- C++ provides the `for` loop specifically for this pattern.

```
int main() {

  for (int x = 0; x < 5; ++x) {

    cout << x << " ";

  }

  cout << "done!" << endl;
}
```

1. Initialize
2. Condition
3. Body
4. Increment

Step 1 is performed **only once**. Steps 2-4 are **repeated** until the condition is `false`.

for Loops: Syntax and Control Flow

- Note the semicolons used to separate parts of the `for`.

```
int main() {

    for (int x = 0; x < 5; ++x) {

        cout << x << " ";

    }

    cout << "done!" << endl;
}
```

Use semicolons to separate these.

1 2 4

3

Step 1 is performed **only once**. Steps 2-4 are **repeated** until the condition is `false`.

3 min

Exercise: for Loops

Example for N = 6:
1 2 4 8 16 32 done!

- Translate this while loop into a for loop.

```
int main() {
  int N = 6;

  int val = 1;
  int x = 0;
  while (x < N) {
    cout << val << " ";

    val *= 2; // Update val by doubling it
    ++x;
  }
  cout << "done!" << endl;
}
```

The *= operator works analogously to +=.

Solution: for Loops

Example for N = 6:
1 2 4 8 16 32 done!

- Translate this while loop into a for loop.

```
int main() {
  int N = 6;

  int val = 1;
  int x = 0;
  while (x < N) {
    cout << val << " ";

    val *= 2;
    ++x;
  }
  cout << "done!" << endl;
}
```

```
int main() {
  int N = 6;

  int val = 1;
  for (int x = 0; x < N; ++x) {
    cout << val << " ";

    val *= 2;
  }
  cout << "done!" << endl;
}
```

Break Time

We'll start again in 5 minutes.

Recall: `if` Statement Syntax

condition
Any expression that can be converted to a `bool`. Written inside ( ).

```
if (condition) {

   statement;
   statement;
   ...

}
```

braces
Always use these around the body.

body
A sequence of statements that will be executed if and only if the condition is true.

Recall: Scope

- A variable can only be used…
  - …after its declaration
  - …within its **scope.**

- If you try to use a variable before its declaration or outside its scope, you'll get a compiler error!

Recall: Local Scope / Block Scope

- Many variables have **local** scope, also known as **block scope**.
- A **block** is a chunk of code enclosed by curly braces { }.
  - Technically, "chunk of code" means a sequence of statements.

```
int main() {
  int x = 5;
  if( x % 2 == 0 ) { // if x is even
    int y = x / 2;
  }
  cout << x << endl;
  cout << y << endl;
}
```

These curly braces define a block. The variable y lives inside this block.

Error! y used out of scope.

Recall: Local Scope / Block Scope

- Block scope applies to any block of code, including the bodies of control flow structures like `if`, `for`, and `while`.

```
int main() {
  int a = 0;
  while(a < 10) {
    int b = a + 1;

    if( b % 2 == 0 ) { // if b is even
      int x = 2 * b;
      cout << x << endl;
    }
    a += x;
  }
  cout << b << endl;
}
```

General rule: A variable is allowed to "enter" a nested block, but it can't leave its own block.

Error! x used out of scope.

Error! b used out of scope.

Local Scope / Block Scope

- For scoping purposes, the top of a `for` loop is treated as if it were inside the loop body.

```
int main() {
  for(int x = 0; x < 10; ++x) {
    cout << x << endl;
  }
  cout << "Final value of x: " << x << endl;
}
```

Error! x used out of scope.

- To use a variable after the loop, move its declaration outside.

```
int main() {
  int x;
  for(x = 0; x < 10; ++x) {
    cout << x << endl;
  }
  cout << "Final value of x: " << x << endl;
}
```

Ok. The scope of x extends throughout the body of `main`.

Nested Loops

- Just like you can have nested `if` statements, you can have nested loops:

output

```
0 1 2
0 2 4
0 3 6
0 4 8
0 5 10
done!
```

```
int main() {
  int N = 3;
  for (int i = 1; i < 2 * N; ++i) {     // outer loop
    for (int j = 0; j < N; ++j) {       // inner loop
      int val = i * j;      // calculates a value
      cout << val << " ";   // prints a message
    }
  cout << endl; // go to a new line after the inner loop finishes
  }
cout << "done!" << endl; // prints after the outer loop finishes
}
```

- You can mix-and-match branching and iteration in lots of different ways. See the Extra Practice Exercises at the end of the lectures slides for examples!

6 min

Exercise: Nested Loops

- Write a program to print out a triangle of Xs.

```
int main() {
  int N = 5;

  // YOUR CODE HERE



}
```

Example for N = 5:

```
X
XX
XXX
XXXX
XXXXX
done!
```

Solution: Nested Loops

- Write a program to print out a triangle of Xs.

```
int main() {
  int N = 5;

  for (int r = 1; r <= N; ++r) {

    for (int x = 0; x < r; ++x) {
      cout << "X";
    }

    cout << endl;
  }
}
```

Example for N = 5:

```
X
XX
XXX
XXXX
XXXXX
done!
```

Alternate Solution: Nested Loops

- This solution uses `while` loops. There's nothing wrong with that in itself, but we've introduced a bug – can you find it?

```
int main() {
  int N = 5;
  int r = 1;
  int x = 0;

  while (r <= N) {
    while (x < r) {
      cout << "X";
      ++x;
    }

    cout << endl;
    ++r;
  }
}
```

~~We moved the declarations of the loop variables out here at the top of the function for "convenience"~~[1]

Oops! x never gets reset back to 0. It should be declared inside the outer loop.

The **scope** of x was **too wide** – it lived and retained its old value instead of getting reset with the outer loop. You can also get logical errors from scope that is too **narrow**.

(Incorrect)
Output for N = 5:
X
X
X
X
X
done!

1 This is actually a terrible idea. Don't do it!

break

- There may be times when you need to end a loop early.
- Use the break function to exit the loop gracefully.

```
int main() {
  int N = 10;
  double limit = 5.0;

  cout << "Values: " << endl;
  for (int i = 1; i < N; ++i) {
      double val = i * 1.43;      // calculates a value
      if (val > limit){
        break;              // stop if val reaches the limit
      }
      cout << val << " " << endl;   // prints the value
    }
cout << "Limit reached!" << endl;
}
```

```
Values:
1.43
2.86
4.29
7.72
8.58
10.01
11.44
12.87
Limit reached!
```

continue

- There may be times when you need to “skip a loop”.
- Use the `continue` function to end the loop at that point and go back to the top of the loop for the next iteration.

```
int main() {
  int N = 10;

  cout << "Values: " << endl;
  for (int i = 3; i < N; ++i) {

      if (i == 5){
        continue;          // skip when i == 5
      }

    cout << i << endl; // prints i
    }
cout << "No more numbers!" << endl;
}
```

```
Values:
3
4
6
7
8
9
No more numbers!
```

This statement does not execute when `i == 5` because the rest of the loop is “skipped” when `continue` is called during that iteration.

EXTRA PRACTICE EXERCISES

Exercise: Iteration and Branching

4 min

- Write code to find the first N numbers that are NOT divisible by a and NOT divisible by b.

```
int main() {
  int N = 5;
  int a = 2;
  int b = 3;
  int x = 1; // HINT: Use x to search through numbers
  while(      ) { // HINT: Keep going until you find enough

    if(                        ) { // Check divisibility
      cout << x << " ";
      // HINT: In addition to printing x, update the count
      //       of how many you've found here.
    }

    ++x;
  }
  cout << "done!" << endl;
}
```

Recall: x is divisible by y if x % y == 0.

Solution: Iteration and Branching

- Write code to find the first N numbers that are NOT divisible by a and NOT divisible by b.

```
int main() {
  int N = 5; // Plan: decrement each time we find one, down to 0
  int a = 2;
  int b = 3;
  int x = 1; // Plan: repeatedly increment by 1 to search values
  while(N > 0) {

    if(x % a != 0 && x % b != 0) { // Check divisibility
      cout << x << " ";
      --N; // This happens only on "successful" iterations
    }

    ++x; // This happens every iteration
  }
  cout << "done!" << endl;
}
```

Example: Finding Prime Numbers

- The previous example of checking divisibility is a first step toward an algorithm for finding prime numbers…
  - We checked whether x was NOT divisible by 2 or by 3.
  - For primes, we need to check that x is NOT divisible by any number between 2 and `x-1`.

- An algorithm for finding the first N primes:
  - Loop through numbers x, starting at `0`, until we find N that are prime.
    - To determine if a number x is prime, loop through all numbers y from 2 through `x-1` and check that x is not divisible by any of them.

Example: Nested Loops

- An algorithm for finding the first N primes:
  - Loop through numbers x, starting at 0, until we find N that are prime.
    - To determine if a number x is prime, loop through all numbers y from 2 through x-1 and check that x is not divisible by any of them.

```
int main() {
  int N = 5;
  int x = 1;
  // Outer loop: iterate through candidate x values
  while(N > 0) {
    // Inner loop: check y values to make sure none divide x
    for (int y = 2; y < x; ++y) {
      // TODO: Check divisibility
    }
    // TODO: Was x divisible by any y? If not, print it and --N
    ++x;
  }
  cout << "done!" << endl;
}
```

Example: Nested Loops

- An algorithm for finding the first N primes:

```
int main() {
  int N = 5;
  int x = 1;
  // Outer loop: iterate through candidate x values
  while(N > 0) {
    bool anyDivisible = false;
    // Inner loop: check y values to make sure none divide x
    for (int y = 2; y < x; ++y) {
      if( x % y == 0 ) { // Check divisibility
        anyDivisible = true;
      }
    }
    if( !anyDivisible ) { // were any divisible?
      cout << x << " ";
      --N;
    }
    ++x;
  }
  cout << "done!" << endl;
}
```