CSCI 200: Foundational Programming Concepts & Design Lecture 05

Repetition using Loops

Open Canvas 9/01 Quiz To Follow Along

Access Code: jetpack

Previously in CSCI 200

- Conditional branches
  - `if` / `else if` / `else`
  - `switch`
- Relational & Logic Operators

Clarifying a small note

- && vs & || vs | == vs =
- && and || are logical operators, eval T / F
- & and | are bitwise operators
  - Take two integers and convert to binary
  - Perform AND or OR operation bit by bit

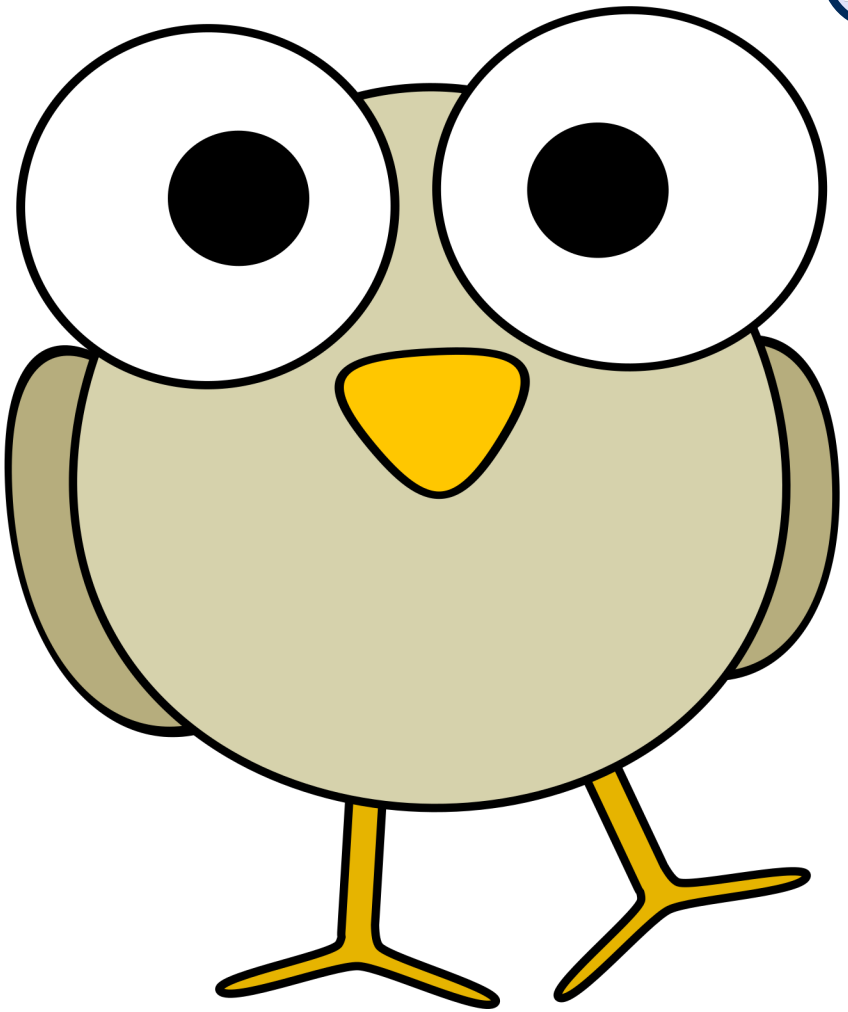
4 0100	7 0111
2 0010	13 1101
4 | 2 == 6 == 0110	7 & 13 == 5 == 0101

Additional Clarification

- Despite `and` and `or` being C++ keywords...

- Use `&&` and `||` for logical operators
  - Be explicit about which and/or you want
    - Logical, not bitwise
  - Almost every example you see will use proper logical operators – and these carry to other languages as well

Questions?



Learning Outcomes For Today

- Identify C++ repetition structures and explain the following terms: looping parameter, stopping condition, and looping parameter modification.
- Explain the appropriate use and differences between a while loop, for loop, and a do-while loop.
- Convert a program written with a for loop to a program using a while loop and vice versa.

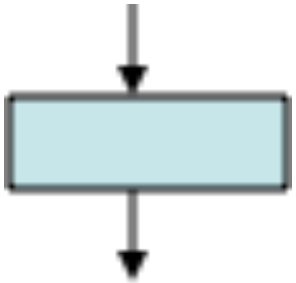
On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

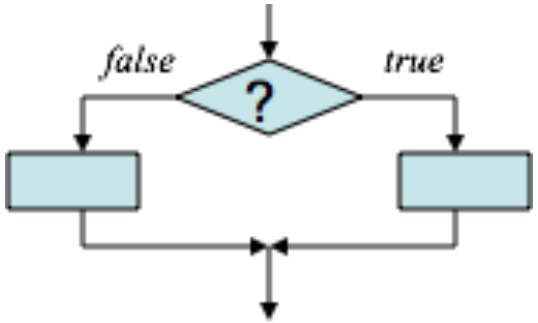
On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

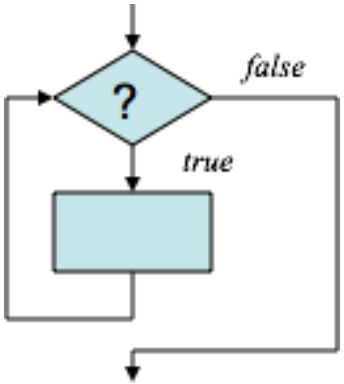
Statement Types



Sequential



Conditional



Repetition

Repetition

- Telling the computer to do a repetitive task
  - Over and over and over

The wrong way

```
01  int main() {
02
03
04
05
06
07
08
09
10
11
12
13
14    return 0;
15  }
```

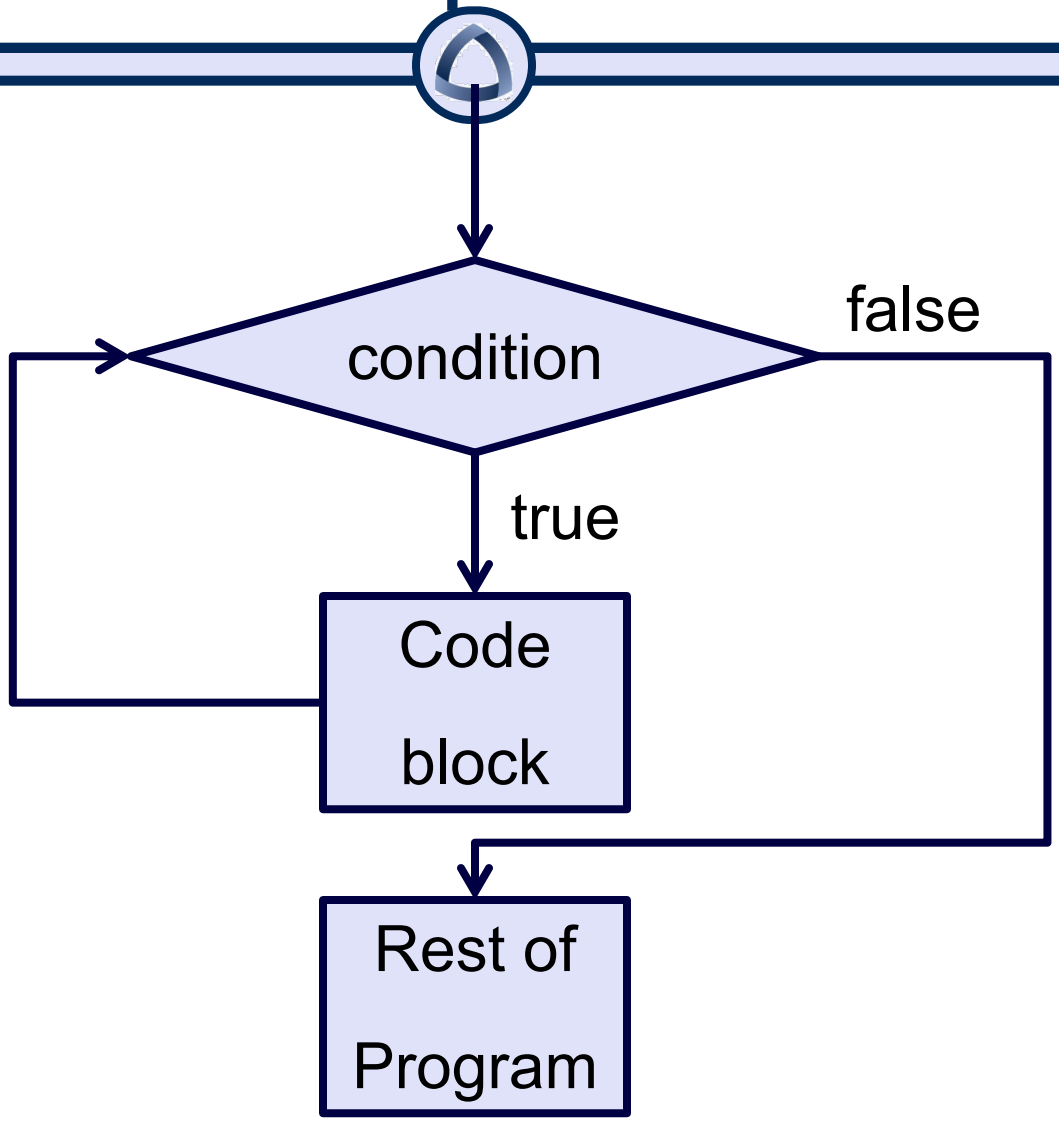
Two Kinds of Loops

- `while` loop
  - When you're not sure how many times to loop
- `for` loop
  - When you know how many times to loop
- (These are rules of thumb, other scenarios exist and both can be used equivalently)

On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

while Loop Structure



while Loop

- Computer, as long as this condition is true, run these commands

```
while( condition ) {
   statement(s);
}
```

while Loop Example

```
char userChoice = 'a';

while( userChoice != 'q' ) {

   cout << "Enter a letter: ";

   cin >> userChoice;

}
```

Parts of a while Loop

```
char userChoice = 'a';

while( userChoice != 'q' ) {

  cout << "Enter a letter: ";

  cin >> userChoice;

}
```

Looping Parameter

Stopping Condition

Loop Body

Parameter Modification

Three Questions To Ask With Loops

1. What is the initial value of the looping parameter?
2. What condition must be met for the looping sequence to execute? (What condition causes the loop to exit?)
3. How is the looping parameter modified in the looping sequence? (What happens if it doesn't change?)

while Loop

```
int x = 0;
while( x < 100 ) {
   cout << "What am I doing?" << endl;
}
```

while Loop

```
int x = 0;
while( x < 100 ) {
   cout << "I'll run 100 times!" << endl;
   x++;
}
```

Increment / Decrement

- Another shorthand
  - All three are equivalent
  - Increment

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

  - Decrement

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

Increment / Decrement

- Two versions

Postfix	Prefix	
`x++;`	`++x;`	`x = x + 1;`
`x--;`	`--x;`	`x = x - 1;`

Precedence Table

Precedence	Operator	Associativity
1	Parenthesis: ( )	Innermost First
2	Postfix Unary Operators: a++ a--	Left to Right
3	Prefix Unary Operators: ++a --a +a -a !a (type)a	Right to Left
4	Binary Operators: a*b a/b a%b	Left to Right
5	Binary Operators: a+b a-b	Left to Right
6	Relational Operators: a<b a>b a<=b a>=b	Left to Right
7	Relational Operators: a==b a!=b	Left to Right
8	Logical Operators: a&&b	Left to Right
9	Logical Operators: a||b	Left to Right
10	Assignment Operators: a=b a+=b a-=b a*=b a/=b a%=b	Right to Left

Order Matters!

```
int x, y;
```

Postfix

```
x = 5;
y = x++;
```

x	y
6	5

Prefix

```
x = 5;
y = ++x;
```

x	y
6	6

Practice

```
int x, y;
x = 10;
y = ++x - 3;
cout << y << endl;
```

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

x == 11

```
y = x - 3;
```

y == 8

```
int a, b;
a = 10;
b = a++ - 3;
cout << b << endl;
```

```
b = a - 3;
```

b == 7

```
a = a + 1;
```

a == 11

Practice

- What is the output?

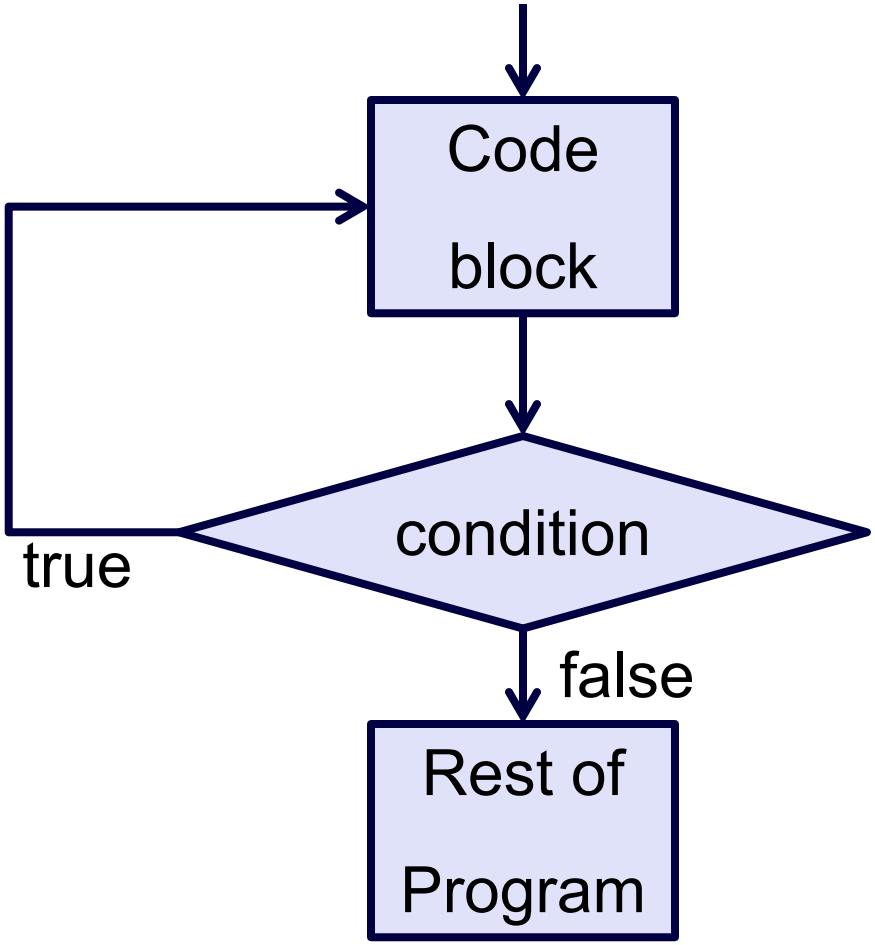
outside while n is -4

```
int n(-4);
while( n > 0 ) {
   cout << n << " ";
   n--;
}
cout << "outside while";
cout << " n is " << n << endl;
```

On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

do-while Loop Structure



do-while Loop

- Computer, run this command and as long as this condition is true, keep running those commands

```
do {
   statement(s);
} while( condition );
```

NOTE THE SEMICOLON!

Parts of a do-while Loop

```
int n = 0;
```

Looping Parameter

```
do {
    cout << n << endl;

    n = n + 1;

} while (n >= 0 && n <= 10);
```

Loop Body

Parameter Modification

Stopping Condition

Practice

- What is the output?

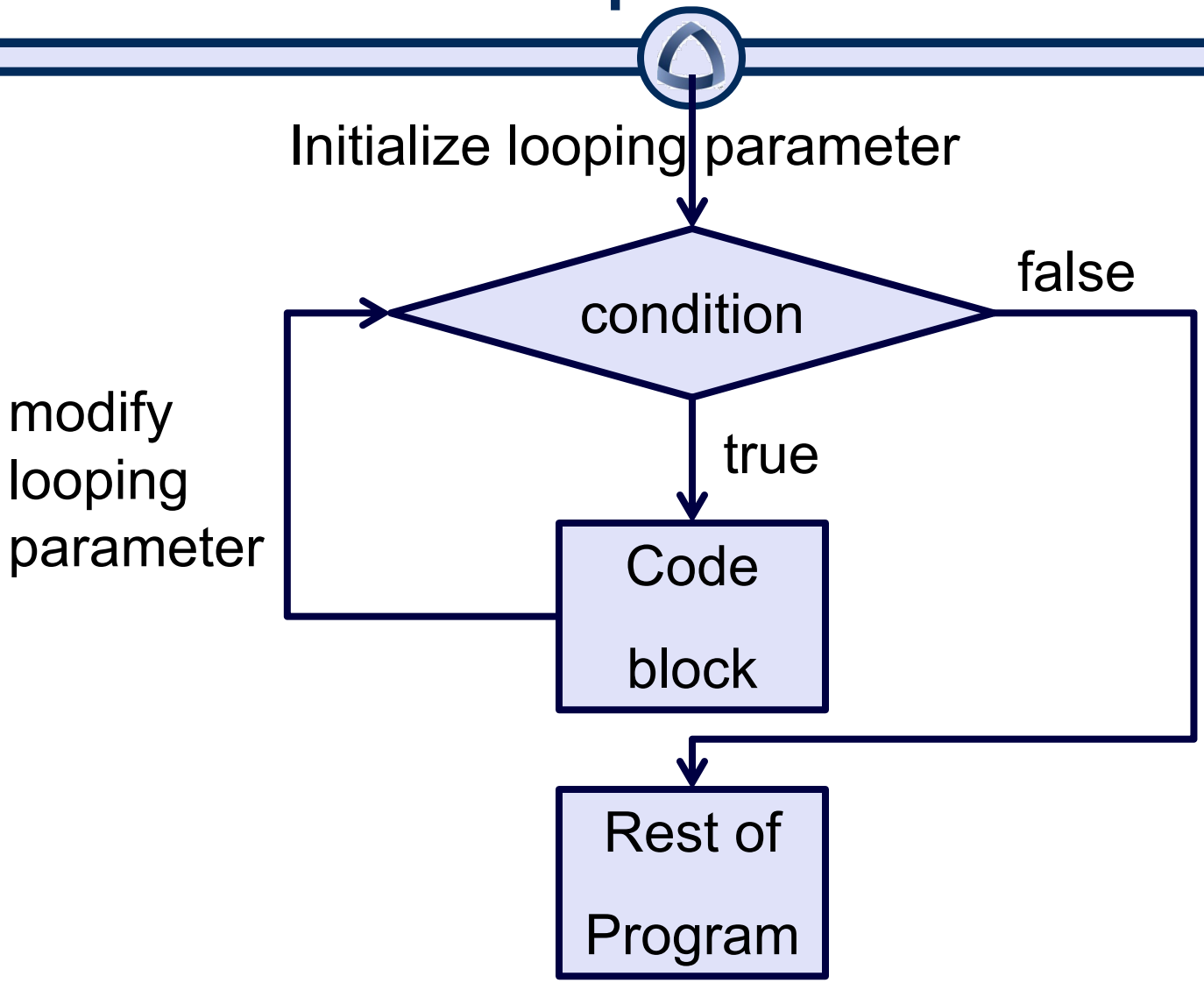
-4 outside do-while n is -5

```
int n(-4);
do {
   cout << n << " ";
   n--;
} while( n > 0 );
cout << "outside do-while";
cout << " n is " << n << endl;
```

On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

for Loop Structure



for loop

- Computer, as long as this condition is true keep doing this stuff

```
for( initialValue; condition; modification ) {
    statement(s);
}
```

for Loop Example

```
int cookies;

for( cookies = 0; cookies <= 8; ++cookies ) {

    cout << "I ate " << cookies
         << " cookies." << endl;

}
```

Parts of a for Loop

```
int cookies;
for( cookies = 0; cookies <= 8; ++cookies ) {
    Loop Body
    cout << "I ate " << cookies
         << " cookies." << endl;

}
```

Looping Parameter

Stopping Condition

Parameter Modification

Practice

```
int sum(0), i;
for( i = 1; i <= 10; ++i ) {
     sum += i;
}
cout << sum << "  ";


sum = 0;
for( i = 1; i <= 10; i += 2 ) {
     sum += i;
}
cout << sum;
```

What gets printed?

55 25

How many times does each loop run?

10 5

Practice

```
int count(0), i, j;
for( i = 3; i <= 5; ++i ) {
   for( j = 10; j >= 5; --j ) {
      count++;
   }
}
cout << count;
```

What gets printed?

18

Practice

```
int count(0), i, j;
for( i = 0; i <= 5; ++i ) {
   for( j = 0; j < i; ++j ) {
      count++;
   }
}
cout << count;
```

What gets printed?

15

On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

Breaking Loops Option #1

- `break;`
  - Leave the loop completely
  - Execution continues with first statement following the loop

- (We saw this in `switch`, it works the same way there)

Break example

```
while( true ) {
  double realPositiveNumber;
  cout << "Enter a number to square root: ";
  cin >> realPositiveNumber;

  if( realPositiveNumber < 0 ) {
    break;
  }

  cout << "sqrt is: " << sqrt( realPositiveNumber ) << endl;
}

cout << "Thanks for playing!" << endl;
```

Breaking Loops Option #2

- `continue;`
  - Terminate current iteration of loop
  - While and do-while loops:
    - Execution continues at the condition statement
  - For loop:
    - Execution continues at the modification of the looping parameter

Practice: Continue example

```
int i;
for( i = 0; i < 10; ++i ) {
    if( i % 2 ) {
        continue;
    }
    cout << i << endl;
}
```

What gets printed?

```
0
2
4
6
8
```

Practice: Break example

```
int i;
for( i = 0; i < 10; ++i ) {
    if( i % 2 ) {
        break;
    }
    cout << i << endl;
}
```

What gets printed?

0

On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

Local Scope

- { } denote a code block
- Variables only exist within that code block
  - Concept of “scope”

Scope Notes

```
int main() {
  int x = 4;
  cout << x << endl;    // prints 4
  int x = 5;            // compiler error! redefinition of x
  if( true ) {
    int x = 2;          // ok, "shadows" prior declaration
    int y = 3;
    cout << x << endl; // prints 2
  }
  cout << x << endl;    // prints 4
  cout << y << endl;    // compiler error! y undeclared
  return 0;
}
```

Scope

- Determines where variables can be referenced
  - Referenceable in ALL code blocks
    - Global Scope
  - Referenceable in A SINGLE code block
    - Local Scope

Global Scope

- Variables that are available anywhere in our program that follows the definition
  - Defined above main()

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

const double PI_CONSTANT = 3.14159;

int main() {
    double area = 5.0 * 5.0 * PI_CONSTANT;
    return 0;
}
```

Local Scope

- Variables that are available only within the code block in which they are defined

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

int main() { // begin code block 1
    int x = 0;
    cout << x << endl;
    if( x < 10 ) { // begin CB  2
        int y = x + 3;       // OK :)
        cout << y << endl;   // OK :)
    } // end CB  2
    return 0;
} // end code block 1
```

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

int main() { // begin code block 1
    int x = 0;
    cout << x << endl;
    if( x < 10 ) { // begin CB 2
        int y = x + 3;  // OK :)
    } // end CB 2
    cout << y << endl; // ERROR!   :(
    return 0;
} // end code block 1
```

Local Scope: Loop Example

- Variables that are available only within the code block which they are defined

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

int main() { // begin code block 1
    int x = 0;
    for( x = 0; x < 10; ++x ) {
                            // begin CB 2
        int y = x + 3;
        cout << y << endl;  // OK :)
    } // end CB 2
    return 0;
} // end code block 1
```

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

int main() { // begin code block 1
    int x = 0;
    for( x = 0; x < 10; ++x ) {
                            // begin CB 2
        int y = x + 3;
    } // end CB 2
    cout << y << endl; // ERROR! :(
    return 0;
} // end code block 1
```

Local Scope & `for` Loops

- Can define looping parameter in the `for` loop declaration

```
int main() { // begin code block 1
    int x = 0;
    // begin CB 2
    for( x = 0; x < 10; ++x ) {
        cout << x << endl;  // OK :)
    } // end CB 2
    cout << x << endl;  // OK :)
    return 0;
} // end code block 1
```

```
int main() { // begin code block 1
    // begin CB 2
    for( int x = 0; x < 10; ++x ) {
        cout << x << endl;  // OK :)
    } // end CB 2
    cout << x << endl;  // ERROR!  :(
    return 0;
} // end code block 1
```

On Tap For Today

- Repetition Structures
  - `while` loop / `do-while` loop
  - `for` loop
- `break` / `continue`
- Scope
- Practice

To Do For Next Time

- Continue with Set1: Due Thurs Sep 7
- Catch up with zyBooks

- No class Monday (Labor Day)

- Wednesday: The Debugger!
- Friday: Quiz 1 in class
  - More details coming next time