UNIVERSITY OF KENTUCKY College of Engineering

CS 215: Introduction to Program Design, Abstraction and Problem Solving

Chapter 4 Loops

What is the Purpose of a Loop?

A loop is a statement that is used to:
execute one or more statements repeatedly until a goal is reached.
Sometimes these one-or-more statements will not be executed at all, if that is the way to reach the goal.

Three Looping statements

C++ has these three looping statements:

- **while**
- **for**
- **do-while**

While loop in detail

```
while (condition) {
    body
}
```

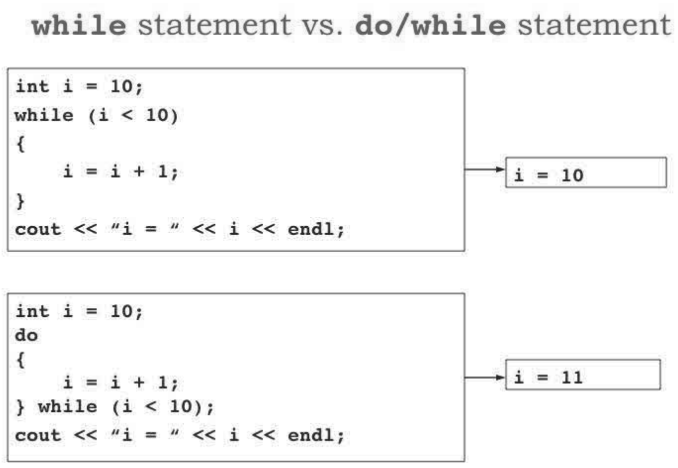
- Checks the condition at the beginning of the loop.
  - If the condition is false, stops the loop.
  - Otherwise, executes the body then repeats.
  - Condition, body, condition, body,· · · , condition, stop.
  - If it starts out false, the body won't run at all! Condition, stop.

do-while loop

- The body of a while loop executes zero or more times.
- Maybe we want it to always run <u>at least once</u>.
- We can do this with a **do-while** loop:

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

  - Like a while loop, but executes the body first.
  - Body, condition, body, condition, $\cdots$, body, condition,stop.
  - Semicolon is mandatory!



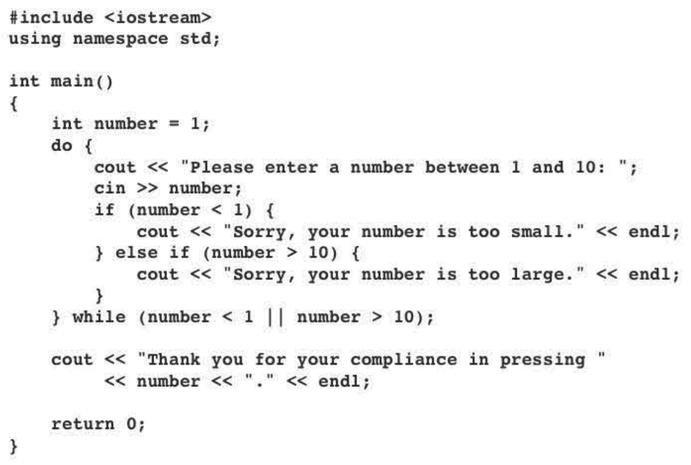
while statement vs. **do/while** statement

```
int i = 10;
while (i < 10)
{
    i = i + 1;
}
cout << "i = " << i << endl;
```

```
i = 10
```

```
int i = 10;
do
{
    i = i + 1;
} while (i < 10);
cout << "i = " << i << endl;
```

```
i = 11
```



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

int main()
{
    int number = 1;
    do {
        cout << "Please enter a number between 1 and 10: ";
        cin >> number;
        if (number < 1) {
            cout << "Sorry, your number is too small." << endl;
        } else if (number > 10) {
            cout << "Sorry, your number is too large." << endl;
        }
    } while (number < 1 || number > 10);

    cout << "Thank you for your compliance in pressing "
         << number << "." << endl;

    return 0;
}
```

DIY

Challenge Activity 4.23.4 from ZyBook

DIY

Challenge Activity 4.23.4 from ZyBook

Block scope

- The scope of a variable runs to the end of the enclosing block.
- What happens if you declare a variable inside a loop?
  - It won't be accessible outside of the loop.
  - It is a different variable every time the loop runs!
  - The scope of the variable is *one execution* of the body.
  - If you need to pass information from one iteration to the next, or if you need some information after the loop finishes, you must declare your variable *before* the loop.
- It is good C++ style to place the variable so the scope is as small as possible, but no smaller.

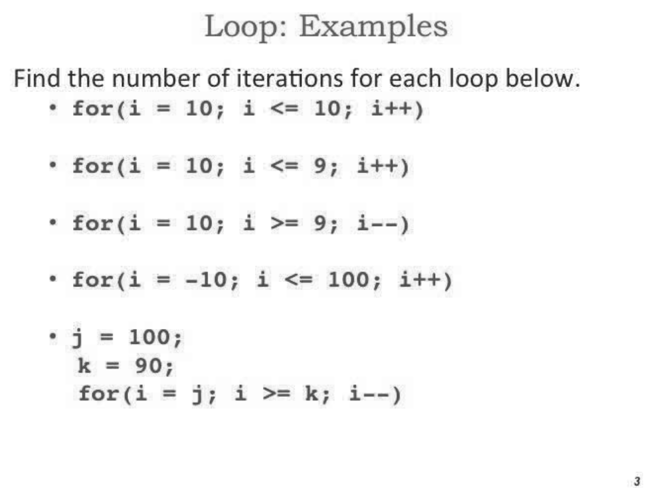
For loops

```
for (int i = 1; i <= 10; i++)
    cout << i << endl;
```

```
int i = 1;
while (i <= 10) {
    cout << i << endl;
    i++;
   }
```

- `for (`*initializer*`;` *condition*`;` *update*`) {`
  body `}`
  - Initializer is an expression (usually assignment) or declaration.
    `for (int i = 1; i <= 10; i++)`
    - ★ If it is a declaration, the scope is the whole loop.
    - ★ Not just one iteration.
    - ★ Still not accessible outside the loop.

-
  ```
  initializer;
  while (condition) {
      body
      update;
  }
  ```

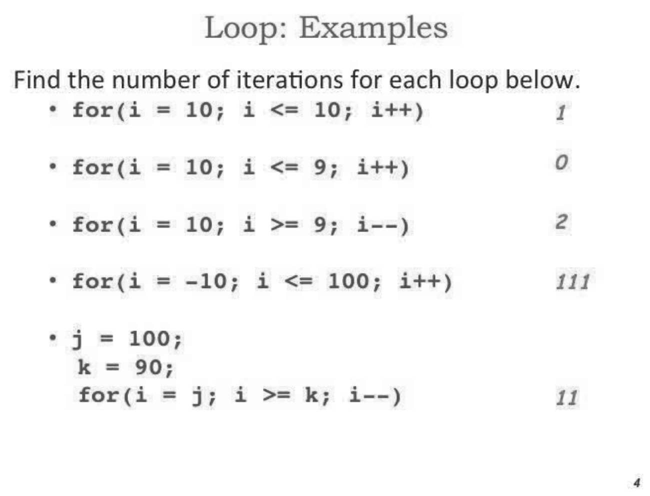


Loop: Examples

Find the number of iterations for each loop below.

- `for(i = 10; i <= 10; i++)`
- `for(i = 10; i <= 9; i++)`
- `for(i = 10; i >= 9; i--)`
- `for(i = -10; i <= 100; i++)`

-
  ```
  j = 100;
  k = 90;
  for(i = j; i >= k; i--)
  ```



Loop: Examples

Find the number of iterations for each loop below.

```
• for(i = 10; i <= 10; i++)          1

• for(i = 10; i <= 9; i++)           0

• for(i = 10; i >= 9; i--)           2

• for(i = -10; i <= 100; i++)        111

• j = 100;
  k = 90;
  for(i = j; i >= k; i--)            11
```

4

break and continue

- In a loop statement, a condition test is used to determine whether another iteration should occur.
  - The `while` and `for` loop statements carry out the test before beginning an iteration.
  - The `do-while` loop statement performs the test after each iteration.
- However, in some applications, the test should occur at a point in the middle of the iteration, and the program should be able to alter loop control. To handle these situations, C++ provides two special loop handling methods:
  - `break` statement
  - `continue` statement

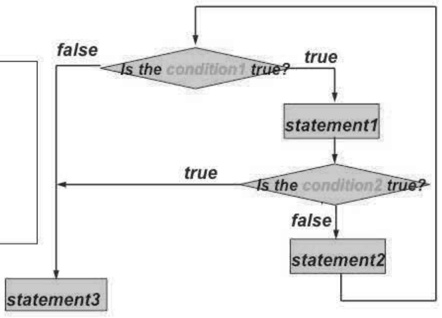
`break` statement

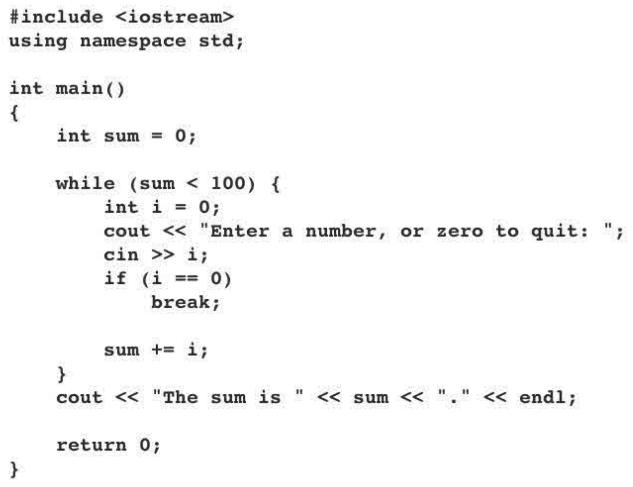
- At any place in a loop body, a `break` statement immediately transfers the control to the first statement following the loop.
  - `break;` — exit the enclosing loop immediately.
  - Doesn't finish executing the body.
  - In a **nested** loop (one inside another), only exits the inner loop.
- The `break` statement is NOT allowed outside a loop!

break Statement

- The `break` statement is used to break out of the enclosing loop, independent of the loop condition

```
while (condition1)
{
   statement1
   if (condition2)   break;
   statement2
}
statement3
```





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

int main()
{
    int sum = 0;

    while (sum < 100) {
        int i = 0;
        cout << "Enter a number, or zero to quit: ";
        cin >> i;
        if (i == 0)
            break;

        sum += i;
    }
    cout << "The sum is " << sum << "." << endl;

    return 0;
}
```

`continue` statement

- The `continue` statement is used to skip all remaining statements in the current iteration and begins the next iteration.
- It makes possible to skip the rest of this iteration without exiting the loop.
  - `continue;` — end the current loop iteration.
  - In a `for` loop, skips to the update.
  - In a `while` or `do-while` loop, skips to the condition.
- The `continue` statement is NOT allowed outside a loop!

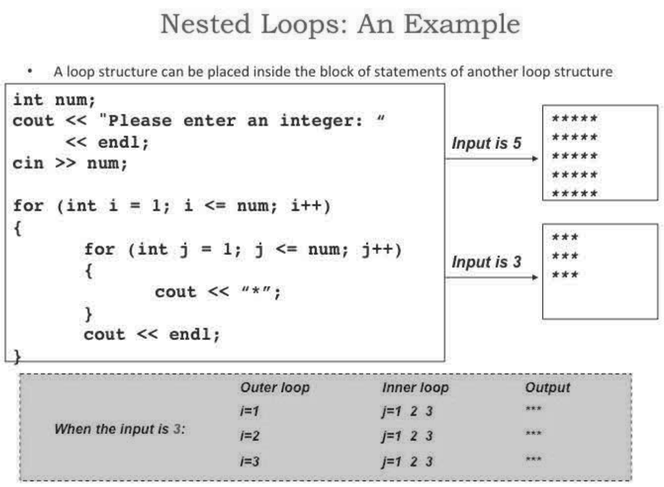
Coding with your instructor

Please download the source file from the following link:

https://www.cs.uky.edu/~yipike/CS215/LoopExamples.cpp

Passcode for today's Lecture quiz:

217CS215



Nested Loops: An Example

- A loop structure can be placed inside the block of statements of another loop structure

```
int num;
cout << "Please enter an integer: “
     << endl;
cin >> num;

for (int i = 1; i <= num; i++)
{
       for (int j = 1; j <= num; j++)
       {
              cout << “*”;
       }
       cout << endl;
}
```

Input is 5

```
*****
*****
*****
*****
*****
```

Input is 3

```
***
***
***
```

	Outer loop	Inner loop	Output
When the input is 3:	i=1	j=1 2 3	***
	i=2	j=1 2 3	***
	i=3	j=1 2 3	***

DIY

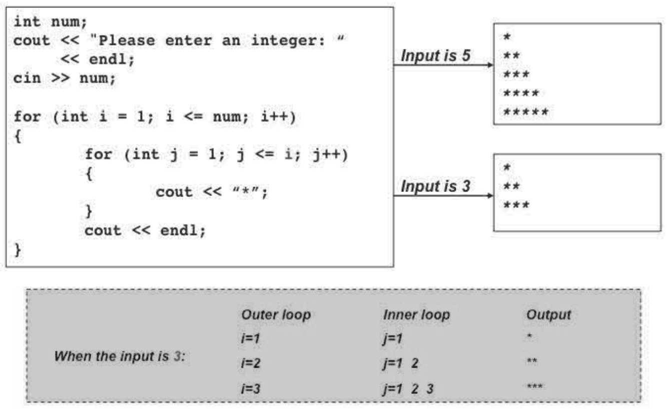
Challenge Activity 4.29.1 from ZyBook

DIY

Challenge Activity 4.29.1 from ZyBook

Nested Loops: An Example

- Sometimes the iteration count of the inner loop depends on the outer loop.



```
int num;
cout << "Please enter an integer: "
     << endl;
cin >> num;

for (int i = 1; i <= num; i++)
{
        for (int j = 1; j <= i; j++)
        {
                cout << "*";
        }
        cout << endl;
}
```

```
*
**
***
****
*****
```

```
*
**
***
```

When the input is 3:	Outer loop	Inner loop	Output
	i=1	j=1	*
	i=2	j=1 2	**
	i=3	j=1 2 3	***

Nested Loop: Some Examples

- Find the number of iterations for each nested loop below.

  -
    ```
    for(i = 1; i <= 10; i++)
        for(j = 1; j <= 10; j++)
    ```

  -
    ```
    for(i = 1; i <= 10; i++)
        for(j = 1; j <= i; j++)
    ```

Example

- What does the following program segment do?

```
for (int i=1; i<=5; i++)
{
      for (int j=1; j<=3; j++)
      {
            int k=5;
            while (k <= 200)
            {
                  cout << "*";
                  k = k * 3;
            }

            cout << endl;
      }

      cout << endl;
}
```

Examples: Nested Loops

- Use nested loops to print the following patterns

```
1            123456            1       123456
12           12345            21        12345
123          1234            321         1234
1234         123            4321          123
12345        12            54321           12
123456       1            654321            1
```