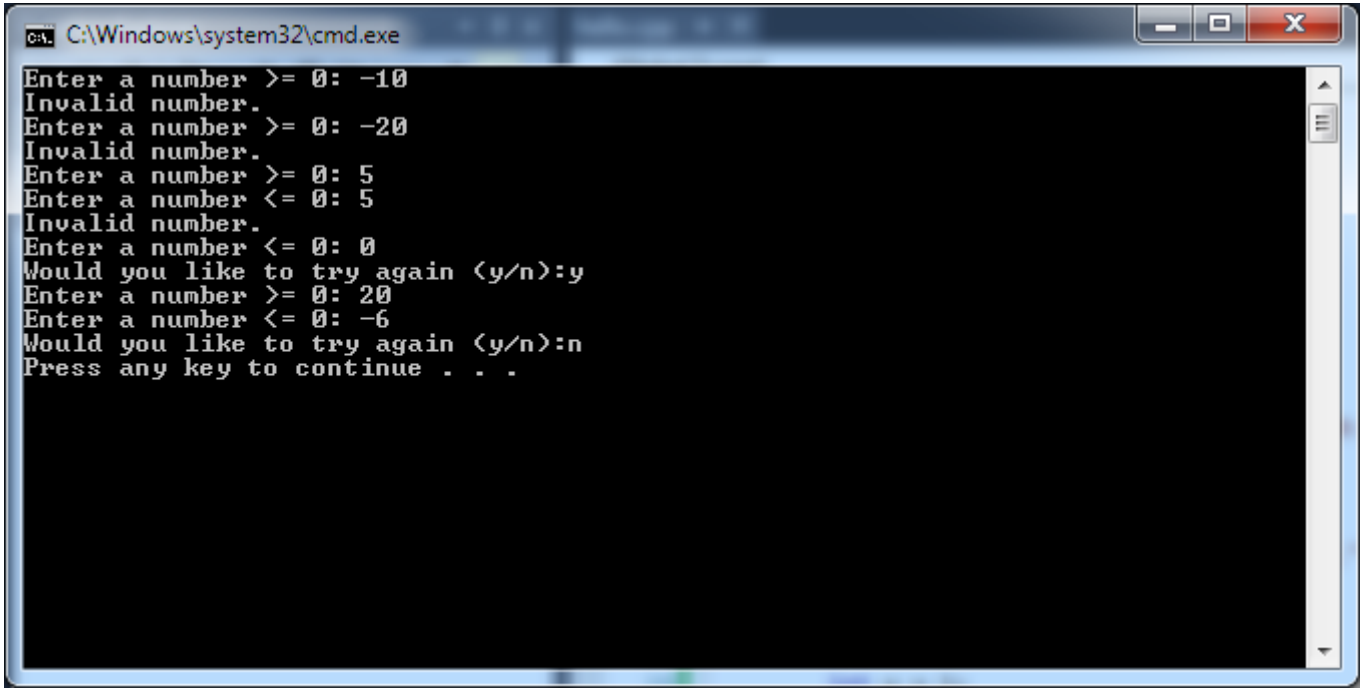
More Loops; Arrays

ITP 165 – Fall 2015
Week 4, Lecture 1

Nesting Loops – A complex example

```
bool tryAgain = true;
while (tryAgain) {
   int x = -1;
   while (x < 0) {
      std::cout << "Enter a number >= 0: ";
      std::cin >> x;
      if (x < 0) {
         std::cout << "Invalid number." << std::endl;
      }
   }

   int y = 1;
   while (y > 0) {
      std::cout << "Enter a number <= 0: ";
      std::cin >> y;
      if (y > 0) {
         std::cout << "Invalid number." << std::endl;
      }
   }

   std::string select;
   std::cout << "Would you like to try again (y/n):";
   std::cin >> select;
   if (select == "n" || select == "no") {
      tryAgain = false;
   }
}
```

The complex example in action

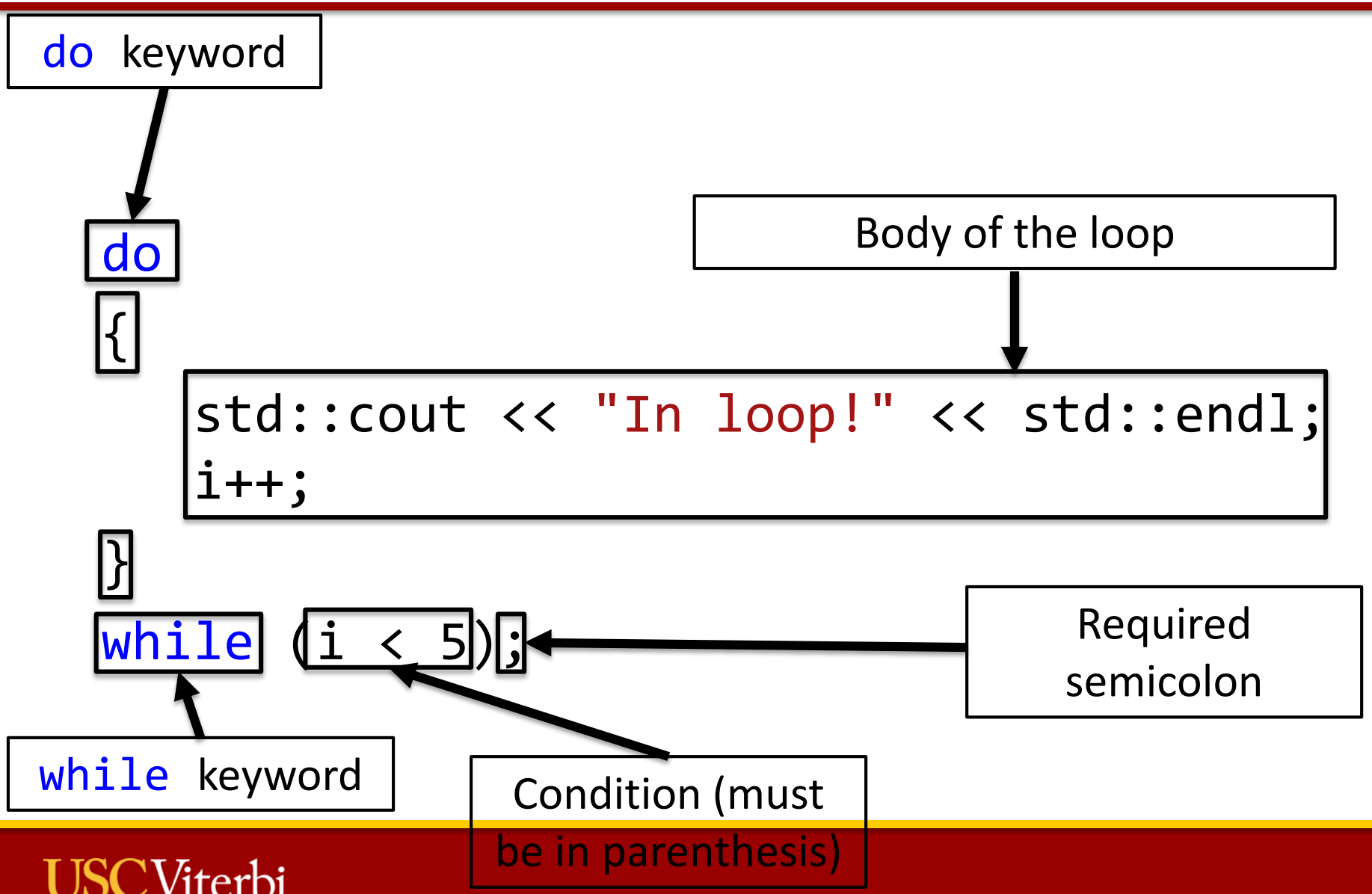


Do-While Loop

```
int number = -1;
do
{
   std::cout << "Enter a number >= 0: ";
   std::cin >> number;
   if (number < 0)
   {
      std::cout << "Invalid number." << std::endl;
   }
}
while (number < 0);
```

1. Execute the body of the loop
2. Check the condition – exit if false, otherwise start the next iteration

Do-While Loop Syntax



Do-While vs. While

- The only difference is that in a `do-while` loop, the body will ***always*** execute once

- In a `while` loop, the body will only execute initially if the condition is true initially

- That's it!

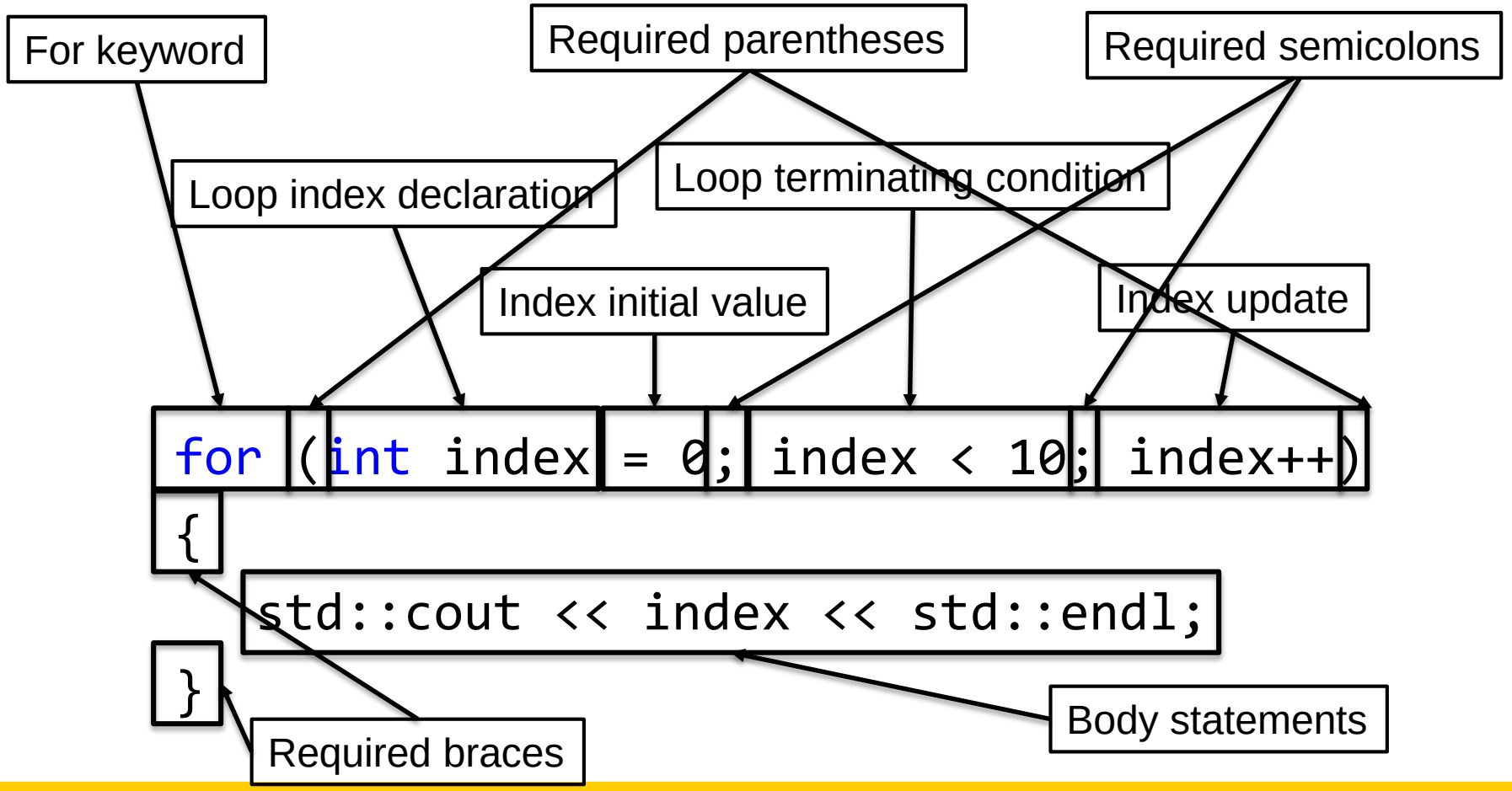
For Loop

- All loops can do the same things
- Some loops are better than others in certain cases
- While loops good when you don't know exactly when to end
  - Defined by some limit or user input
- Do While loops good when you know you need it at least once, but don't know when to end
  - Same as While loop
- For loops good when you know how many times to run before you end

For Loop

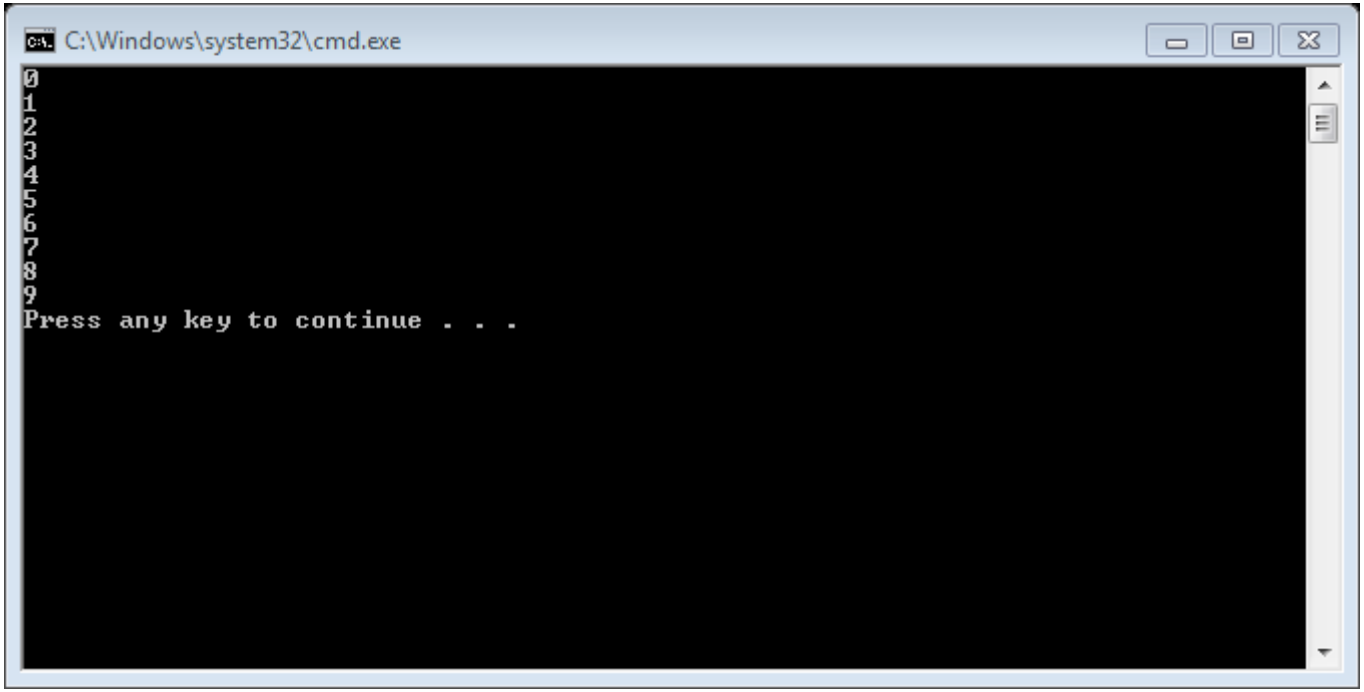
```
#include <iostream>

int main()
{
   for (int index = 0; index < 10; index++)
   {
      std::cout << index << std::endl;
   }

   return 0;
}
```

For Loops


For keyword
Required parentheses
Required semicolons
Loop index declaration
Loop terminating condition
Index initial value
Index update
for (int index = 0; index < 10; index++)
{
std::cout << index << std::endl;
}
Required braces
Body statements

For Loops

- The result of the program

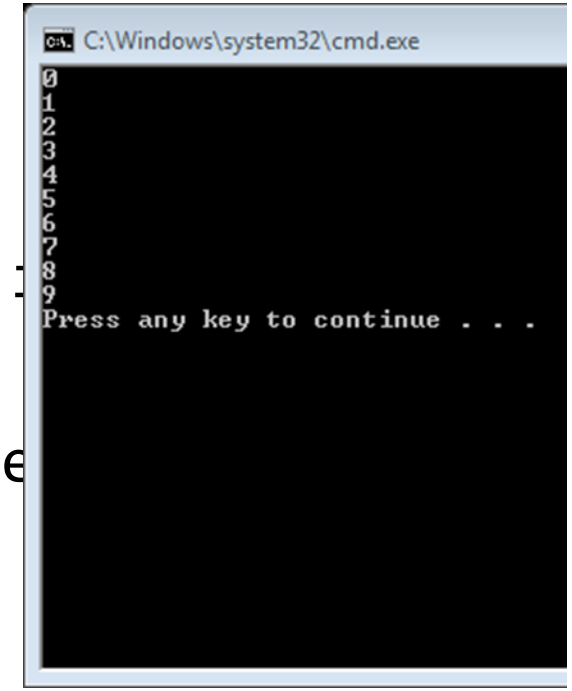


For Loops

- The index of the loop does not need to be used in the body statements

- It is a separate variable that is created *within the scope of the loop* specifically for looping purposes

```
int counter = 0;
for (int index = 0; index < 10; i
{
   std::cout << counter << std::e
   counter++;
}
```



For Loops

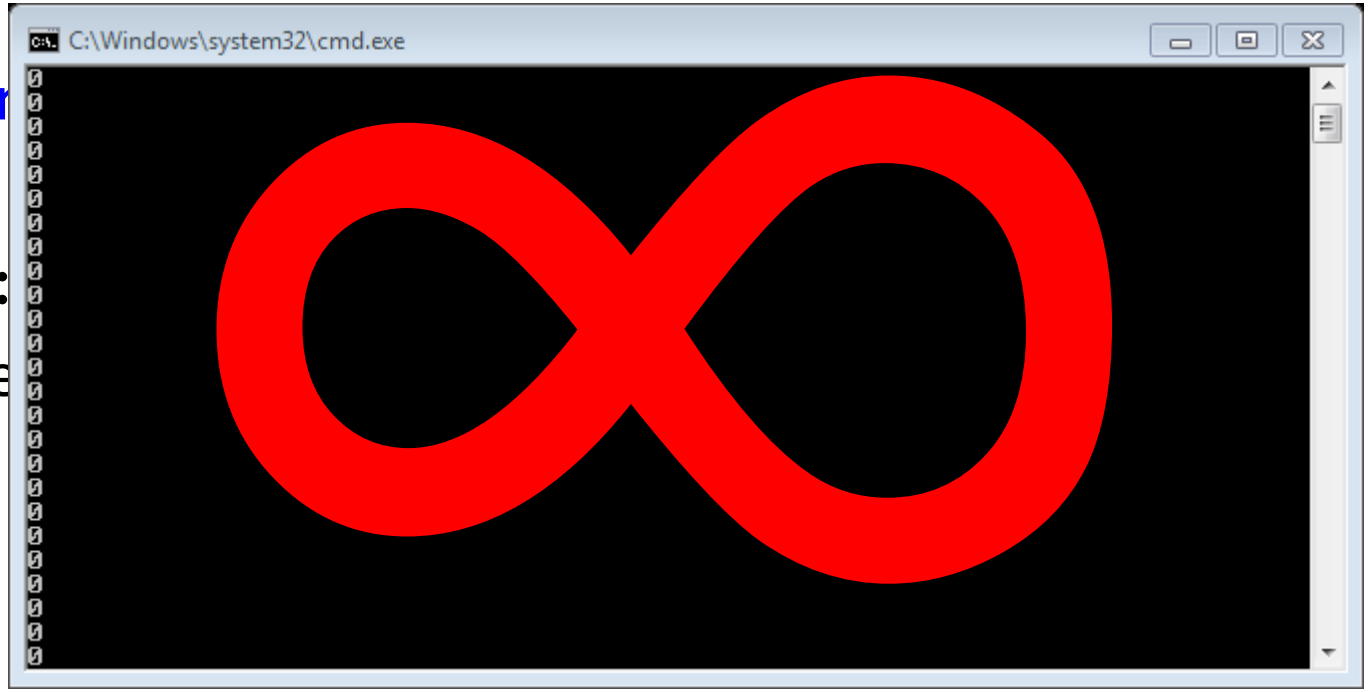
- Loop indexes don't have to be integers, but work best when they are
- Loop indexes don't have to increase by 1, but work best when they do
- Loop indexes can also decrease by 1 (--)
- For Loops run ***as long as the conditional is true***, which for integer values is usually a set number of iterations.

For Loops

- Hard to run in to an infinite loop with a for loop

- But beware:

```
for (ir
{
    std:
    inde
}
```



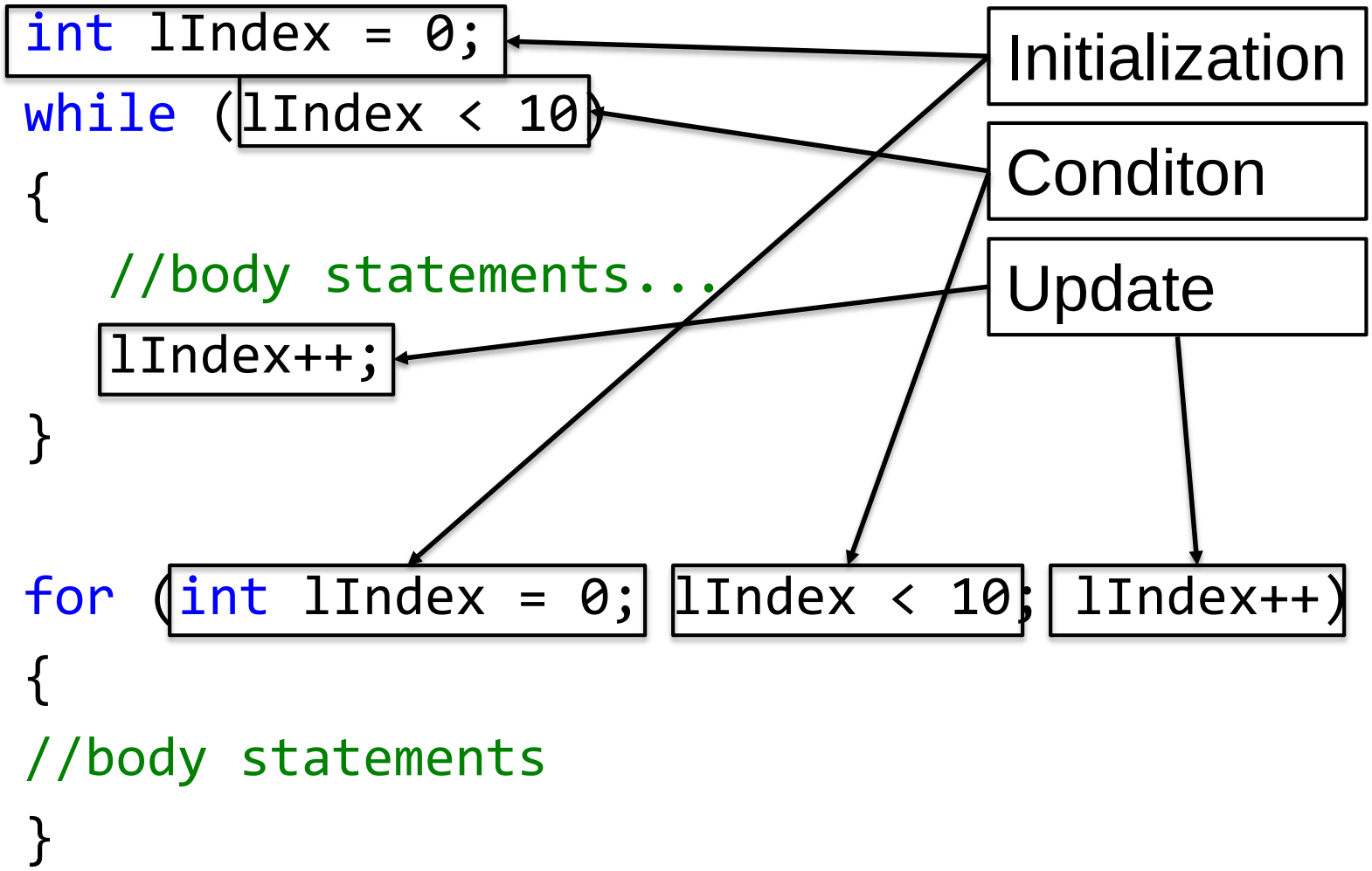
Loops in General

- Loops have 3 defining features:
  - Intialization: Starting of the loop index
  - Condition: Must be true for loop to continue
  - Update: Changes in loop index

- While loops:
  - Initialization: happens **BEFORE THE LOOP**
  - Condition: happens when **DEFINING THE LOOP**
  - Update: happens **IN THE LOOP**

- For loops:
  - Initialization, Condition, Update: happens when **DEFINING THE LOOP**

Loops in General



Problem: Class Roster

- Suppose we have 5 students in a class

- We want to store the names of students in variables. Based on what we've covered so far, we'd have to make 5 different string variables:

```
std::string name1 = "James";
std::string name2 = "Mary";
std::string name3 = "John";
std::string name4 = "Patricia";
std::string name5 = "Robert";
```

Problem: Outputting Class Roster

- Then to output the class roster, we would have to say:

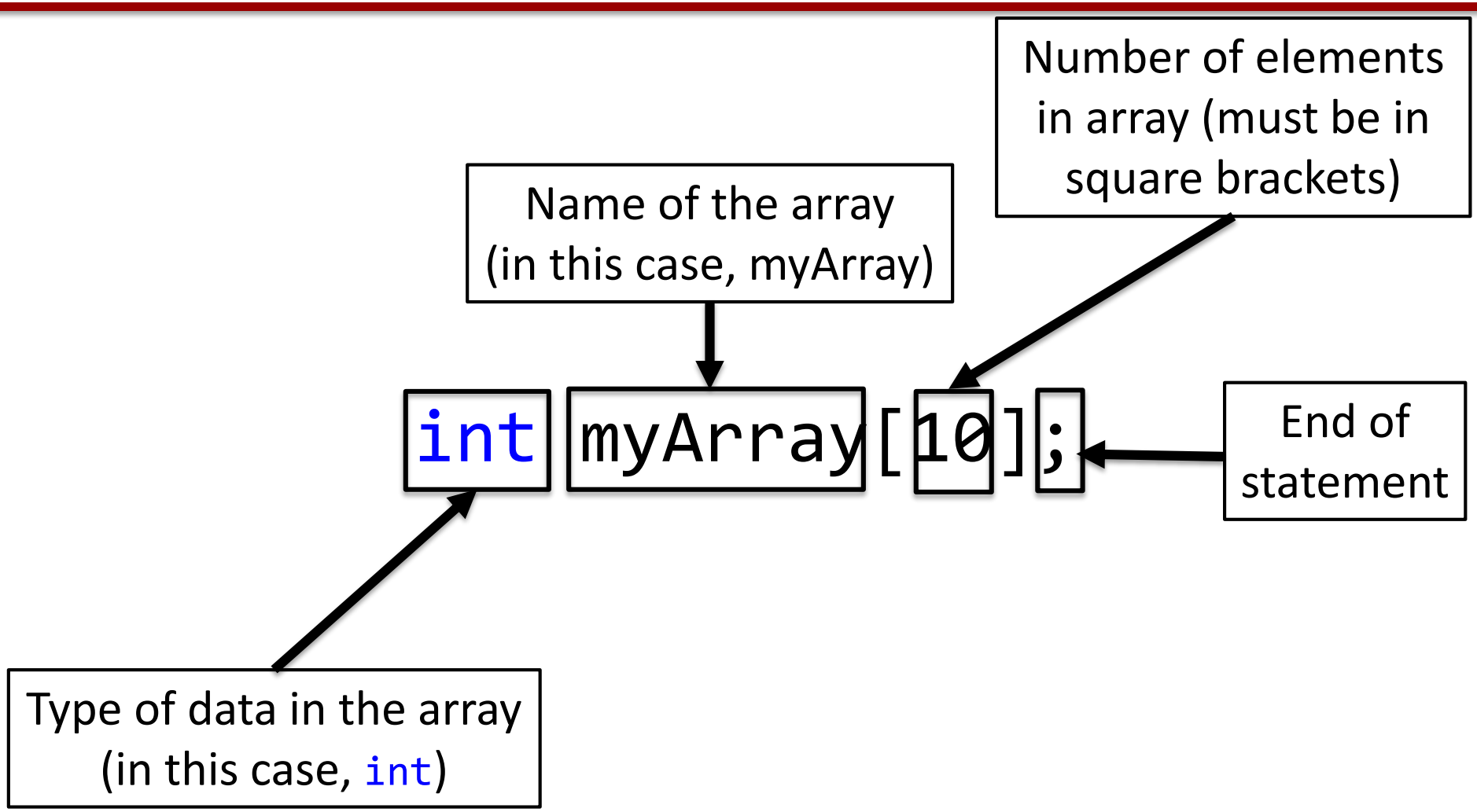
```
std::cout << name1 << std::endl;
std::cout << name2 << std::endl;
std::cout << name3 << std::endl;
std::cout << name4 << std::endl;
std::cout << name5 << std::endl;
```

Arrays

- An ***array*** is a collection of one or more elements that share the same type

- So rather than having five separate name variables, we can have one name array that contains all five names!

Declaring an Array



Array Indices

- To access a specific element in an array, we use a whole number ***index***

- So if I have an array of 10 elements, there would be 10 indices

- The indices…
  - Start at 0 (eg. the first element in the array is at ***index 0***)
  - End at number of elements minus 1

Array Example

```
std::string names[5];
```

- The name of the array is `names`
- The type of the data in the array is `std::string`
- There are 5 elements in the array
- The index of the first element is index 0
- The index of the last element is index 4

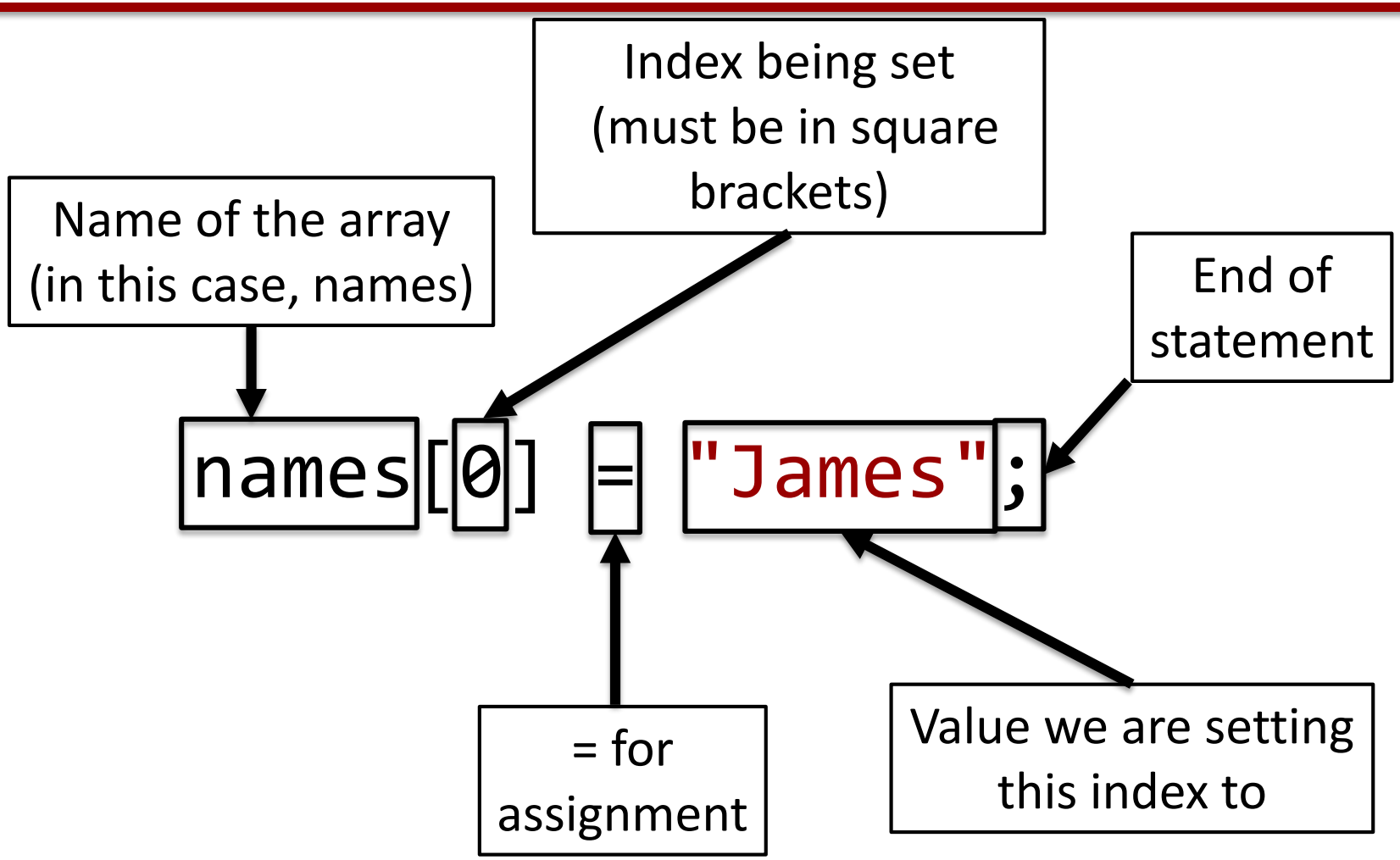
Setting a Value at a Specific Index

- After I declare the array, I can then set a value at a specific index.

- For example, this declares the array, and then sets the value at index 0 to “James”:

```
std::string names[5];
names[0] = "James";
```

Setting a Value at an Index



Accessing a Value at an Index

- To access a value at an index, we use the same `[index]` syntax, also called a ***subscript***:

```
std::string names[5];
// Set index 0 to "James"
names[0] = "James";
// cout the value at index 0
std::cout << names[0] << std::endl;
```

Revisiting the Class Roster…

- So if we wanted to create an array of student names, we could use the following code:

```
std::string names[5];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";
```

- This is preferable to having 5 different variables – especially because if we need to change it to 10 students, we just need to change the size of the array!

Accessing an Out-of-Bounds Index

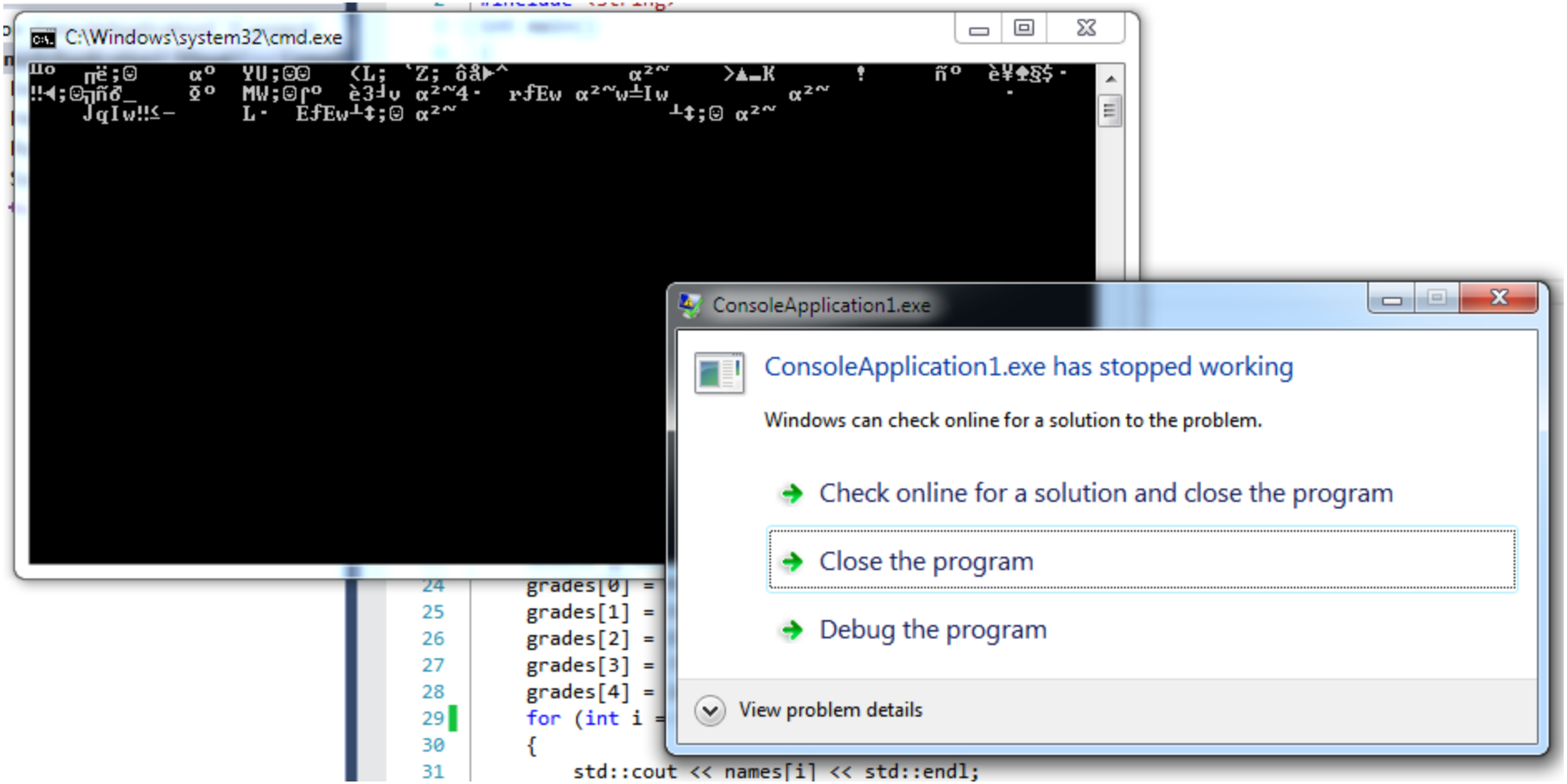
- C++ will not tell us if we went outside the valid bounds of the array
- It will defer to you and assume you know what you're doing, even though it's wrong
- For example, this code will compile and execute:

```
std::string names[5];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";
// This cout is out of bounds!
std::cout << names[5] << std::endl;
```

- But the result of what happens is undefined…

Accessing an Out-of-Bounds Index, Cont'd

- This is what happened with the bad code from the previous slide:



Arrays and Types

- Remember, we can use arrays and any type, not just `std::string`

- So another example…

```
double grades[5];
grades[0] = 95;
grades[1] = 93.2;
grades[2] = 87.6;
grades[3] = 75.9;
grades[4] = 100.0;
```

Looping through the array

- In order to output all the strings in this names array, we could use the following algorithm:

1. Start at index 0

2. Output the name at the current index

3. Add one to the current index

4. If the current index is greater than the max index, exit the loop. Otherwise, goto step 2.

Class Roster Loop

- So if this is our array:

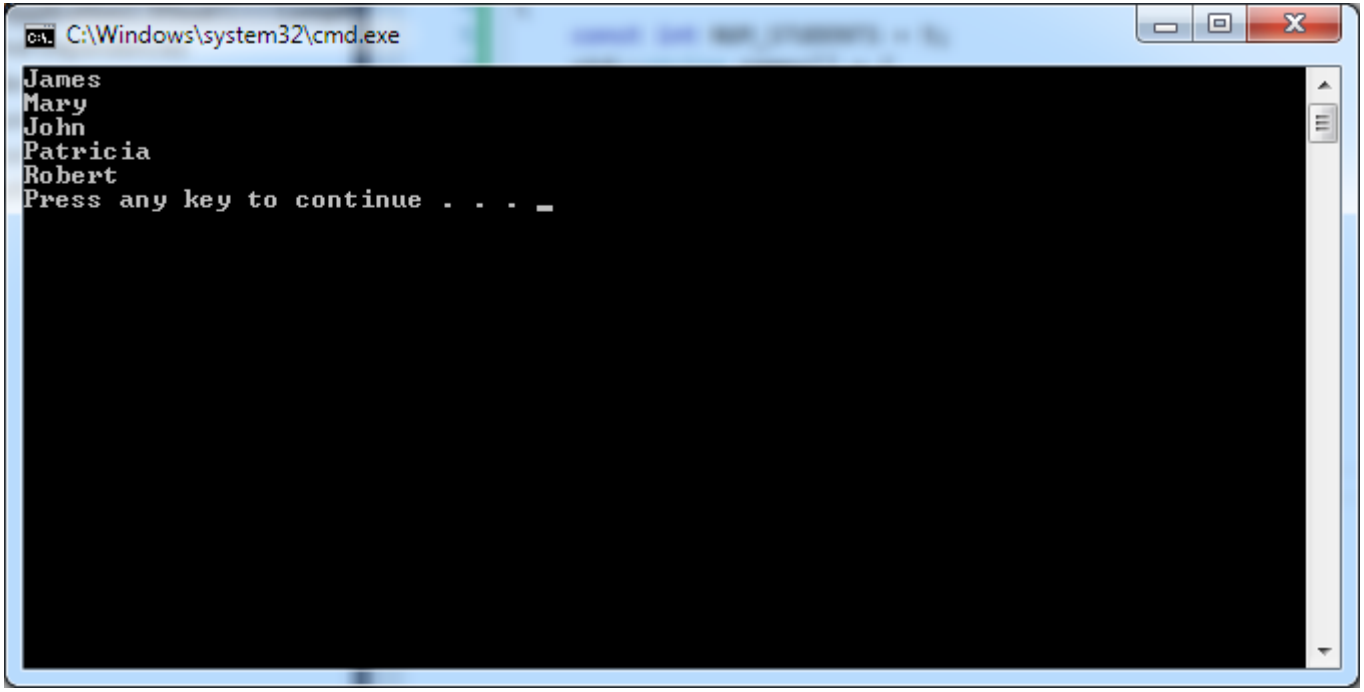
```
std::string names[5];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";
```

- If we want to output all of the names in the array using a loop, what should the condition of our loop be (assuming we initialize our index variable to 0)?

Class Roster Code w/ Loop

```
std::string names[5];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";

int i = 0;
while (i < 5)
{
    std::cout << names[i] << std::endl;
    i++;
}
```

Class Roster Code w/ Loop, In Action



Looping and “Off by one” error

- An “off by one” error happens when the condition for your loop is wrong
- Pretty much, your looping through an array should always be like this:

- If you’re going forward…
  - Start at `index = 0`
  - Condition should be `index < size_of_array`

- If you’re going backwards…
  - Start at `index = (size_of_array - 1)`
  - Condition should be `index >= 0`

What if we want to have 10 students?

```
std::string names[5];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";

int i = 0;
while (i < 5)
{
    std::cout << names[i] << std::endl;
    i++;
}
```

You have to remember to change a 5 to a 10 in two different places.

Could we use a variable?

Really Important Array Constraint

- The size of the array has to be known at the declaration line

- Furthermore, the size of the array cannot be an arbitrary expression – it must be a known constant value

- So this ***will not work***:

```
int num_students = 5;
// Error below: "expected constant expression"
std::string names[num_students];
```

The Const Type Specifier

- `const` is a type specifier that says "I guarantee I will never change this variable"

- So this code ***would*** work:

```
const int num_students = 5;
std::string names[num_students];
```

Naming Const Variables

- I recommend that you always name a `const` variable in ALL CAPS

- So the previous example should probably be written as:

```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS];
```

Going Back to the Full Example

```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";

int i = 0;
while (i < NUM_STUDENTS)
{
    std::cout << names[i] << std::endl;
    i++;
}
```

If you try to change a const after declaration…

- You get an error!

- So this won't work:

```
const int NUM_STUDENTS = 5;
NUM_STUDENTS = 10; // Error: Can't assign to const
```

Let's Look at the Loop Again

```
int i = 0;
while (i < NUM_STUDENTS)
{
    std::cout << names[i] << std::endl;
    i++;
}
```

- Problems:
  - I have to declare the variable i outside the scope of the loop
  - If I forget the i++, the loop will be infinite
- Really, all I want is a loop that loops from 0 to NUM_STUDENTS – 1…

The For Loop

- I could rewrite the preceding code as a for loop:

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	0

1. Execute the initialization code

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	0

2. Check the condition, if the condition is false, exit loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	0

3. Execute the body of the loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	1

4. Execute the increment or decrement code
5. Goto step 2

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	1

2. Check the condition, if the condition is false, exit loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	1

3. Execute the body of the loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
   std::cout << names[i] << std::endl;
}
```

Variable	Value
i	2

4. Execute the increment or decrement code
5. Goto step 2

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	2

2. Check the condition, if the condition is false, exit loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	2

3. Execute the body of the loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	3

4. Execute the increment or decrement code
5. Goto step 2

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	3

2. Check the condition, if the condition is false, exit loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	3

3. Execute the body of the loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
   std::cout << names[i] << std::endl;
}
```

Variable	Value
i	4

4. Execute the increment or decrement code
5. Goto step 2

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	4

2. Check the condition, if the condition is false, exit loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	4

3. Execute the body of the loop

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	5

4. Execute the increment or decrement code
5. Goto step 2

For loop – Step by Step

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

Variable	Value
i	5

2. Check the condition, ***if the condition is false***, exit loop

The For Loop, w/ Advantages

```
for (int i = 0; i < NUM_STUDENTS; i++)
{
    std::cout << names[i] << std::endl;
}
```

- Advantages over `while` loop:
  - The variable i will now only be valid during the scope of the loop
  - I'm a lot less likely to forget to put the i++, since it's on the first line rather than all the way at the bottom of the body of the loop

Lab Practical #5