More Conditionals; Nested Conditionals; Switch Statements

ITP 165 – Fall 2015
Week 2, Lecture 2

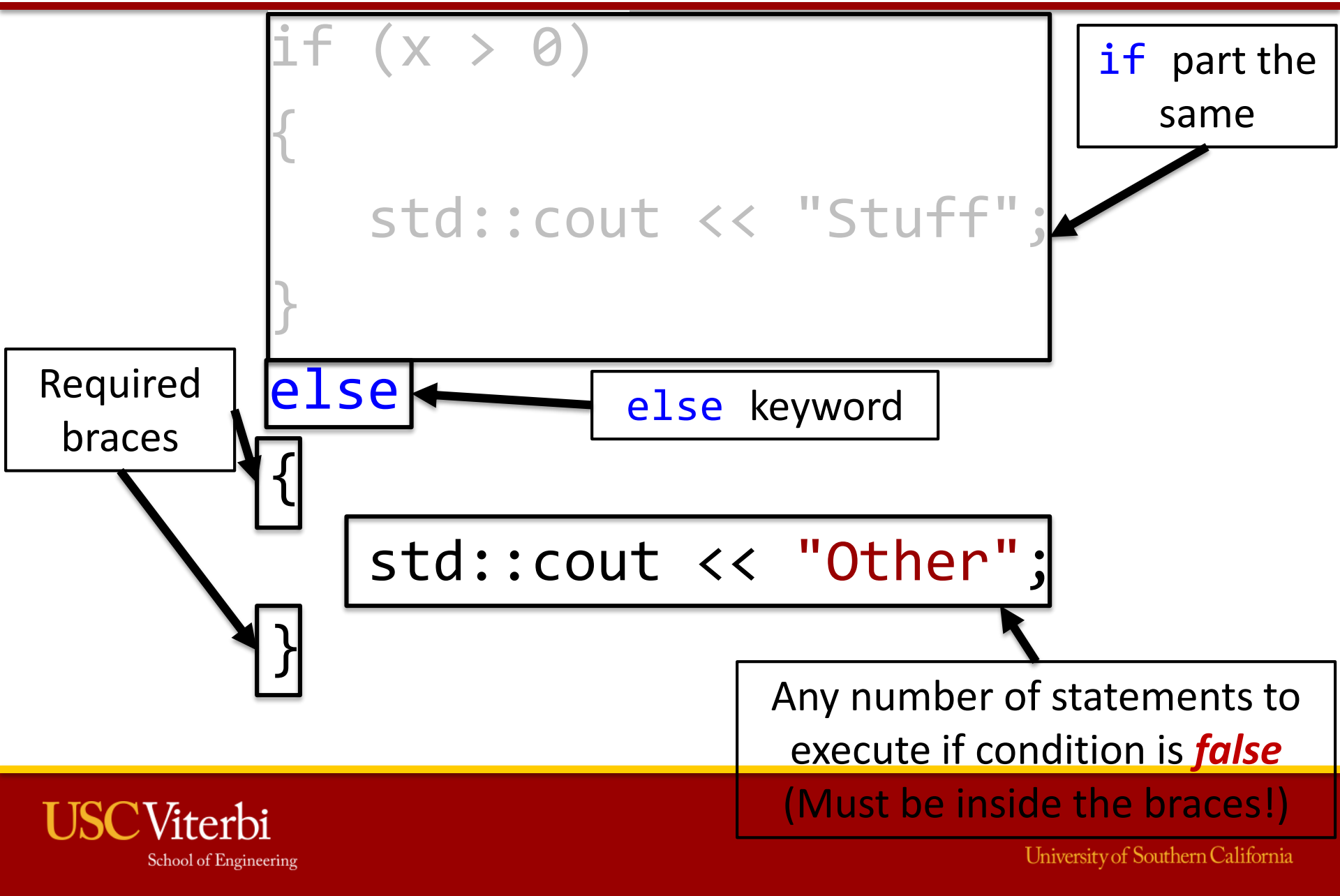
Sample Program – if with else

```
#include <iostream>
int main()
{
   int x = 0;
   std::cout << "Enter a number:";
   std::cin >> x;
   if (x >= 0)
   {
      std::cout << "You entered a positive number.";
   }
   else
   {
      std::cout << "You entered a negative number.";
   }

   return 0;
}
```

Sample Program – if with else

```
#include <iostream>
int main()
{
   int x = 0;
   std::cout << "Enter a number:";
   std::cin >> x;
   if (x >= 0)
   {
      std::cout << "You entered a positive number.";
   }
   else
   {
      std::cout << "You entered a negative number.";
   }

   return 0;
}
```

If with else Syntax



If with else - Scoping

```
int main()
{  // Scope for "main" begins
   int x = 0;
   if (x > 0)
   {  // Scope for if begins

   }  // Scope for if ends
   else
   {  // Scope for else begins

   }  // Scope for else ends

   return 0;
}  // Scope for "main" ends
```

Sample Program – if, else if, else

```
#include <iostream>
int main()
{
   int x = 0;
   std::cout << "Enter a number:";
   std::cin >> x;
   if (x == 0)
   {
      std::cout << "0 is neither positive nor negative.";
   }
   else if (x > 0)
   {
      std::cout << "You entered a positive number.";
   }
   else
   {
      std::cout << "You entered a negative number.";
   }
   return 0;
}
```

Else if, more closely

```
if (x == 0)
{
   std::cout << "0 is neither positive nor negative.";
}
else if (x > 0)
{
   std::cout << "You entered a positive number.";
}
else
{
   std::cout << "You entered a negative number.";
}
```

Else if, Cont'd

- `else if` must always occur ***immediately after*** an `if` or an `else if`

- Eg. this code is nonsensical:

```
else if (x == 0)
{
   std::cout << "0 is neither positive nor negative.";
}
if (x > 0)
{
   std::cout << "You entered a positive number.";
}
```

Else if, Cont'd

- An `else if` does not necessarily need to be followed by an `else`

- Eg. this is fine:

```
if (x > 0)
{
   std::cout << "You entered a positive number.";
}
else if (x < 0)
{
   std::cout << "You entered a negative number.";
}
```

Else if, Cont'd

- You can have any number of `else if`s in succession
- So this is fine:

```
if (x > 0)
{
   std::cout << "You entered a positive number.";
}
else if (x < 0)
{
   std::cout << "You entered a negative number.";
}
else if (x == 0)
{
   std::cout << "0 is neither positive nor negative.";
}
```

Else if, cont'd

- If you have an `else`, it must be after all of the `else if`s

```
if (x == 5)
{
   std::cout << "x equals 5.";
}
else if (x == 10)
{
   std::cout << "x equals 10.";
}
else if (x == 20)
{
   std::cout << "x equals 20.";
}
else
{
   std::cout << "x is neither 5, 10, nor 20.";
}
```

Else if, cont'd

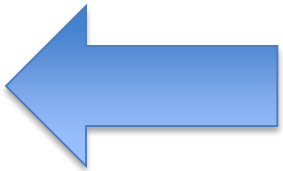
- Else-if statements imply ***mutual exclusivity***

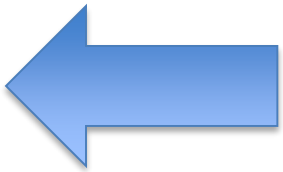
- Only **ONE** of the statements will execute

For example…

Else if, cont'd

```
if (x > 10)
{
    //...
}
else if (x > 5)
{
    //...
}
else if (x > 0)
{
    //...
}
else
{
    //...
}
```

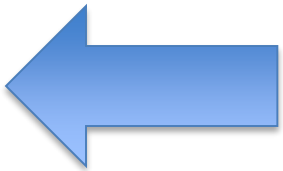




If x = 7, it satisfies these conditions

Else if, cont'd

```
if (x > 10)
{
    //...
}
else if (x > 5)
{
    //...
}
else if (x > 0)
{
    //...
}
else
{
    //...
}
```



Because of the `else if`, only the first condition is executed

Nesting if statements

- Since an if statement is a type of statement, we can nest them.
- For example:

```
int x = 50;
if (x < 0)
{
   if (x < -100)
   {
      std::cout << "X is really negative...";
   }
   else
   {
      std::cout << "X is a little negative...";
   }
}
```

Nested if statements, Cont'd

- In many cases, nested ifs can be equivalent to if-else if-else chains
- The previous slide's code could be rewritten as:

```
int x = 50;
if (x < -100)
{
   std::cout << "X is really negative...";
}
else if (x < 0)
{
   std::cout << "X is a little negative...";
}
```

Nested if statements, Cont'd

```
std::cout << "Enter 1 for drinks or 2 for food: ";
int select = 0;
std::cin >> select;
if (select == 1)
{
   std::cout << "Enter 1 for soft drink or 2 for alcohol: ";
   int drinkSelect = 0;
   std::cin >> drinkSelect;
   if (drinkSelect == 1)
   {
      // Soft Drink
   }
   else
   {
      // Alcohol
   }
}
else
{
   // Similar code for food...
}
```

Spaghetti Code

- This term is used sometimes if you have lots of confusing conditionals and it's really unclear which way the code flows

- So when deciding to nest or not nest if statements, you should think a moment about if the opposite approach would be clearer

Many else if statements

- Suppose you had a menu that allows someone to select options 1 through 4. One way to write this code is:

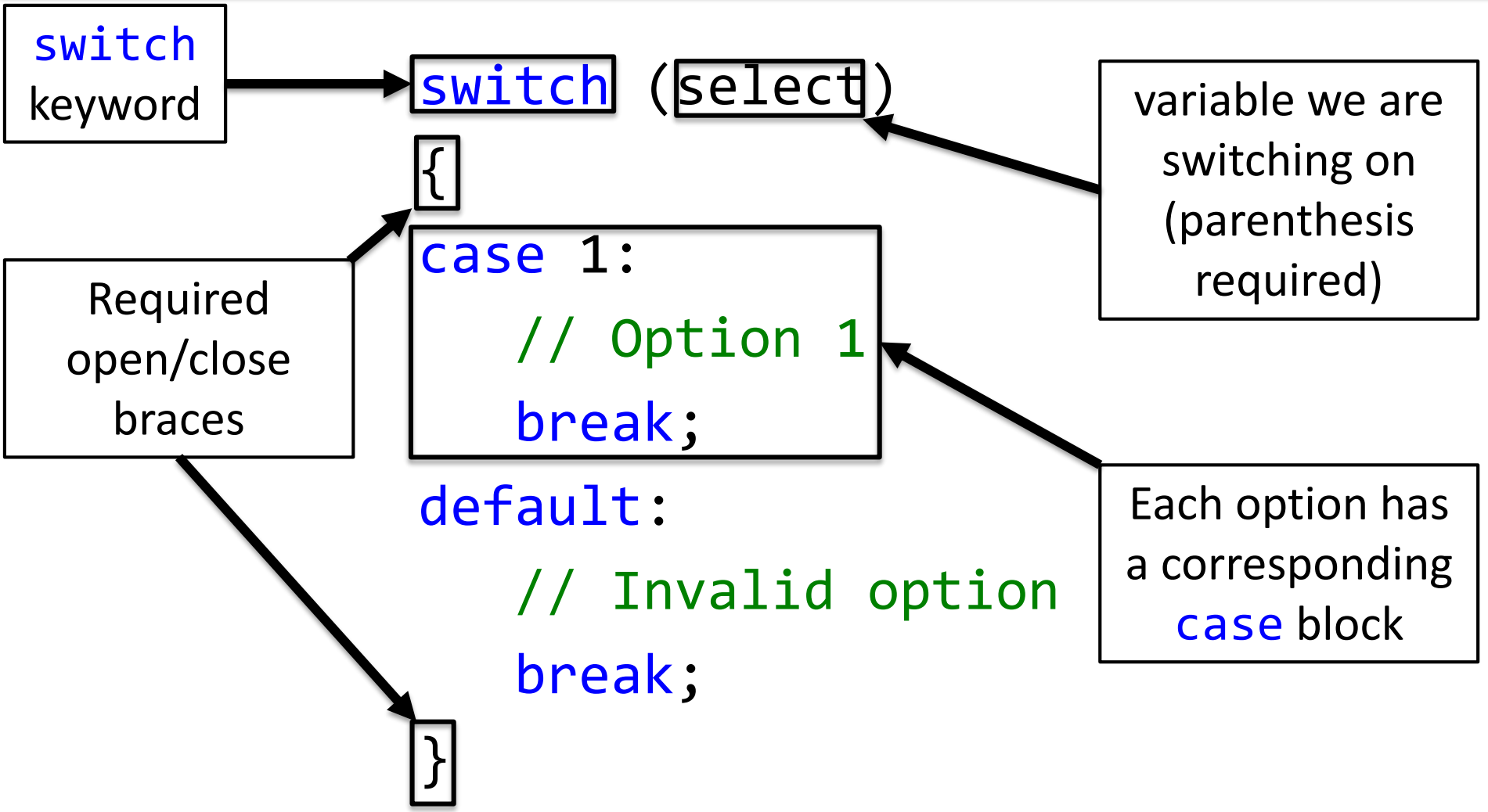
```
if (select == 1)
{
   // Option 1
}
else if (select == 2)
{
   // Option 2
}
else if (select == 3)
{
   // Option 3
}
else if (select == 4)
{
   // Option 4
}
else
{
   // Invalid option
}
```

Switch statement

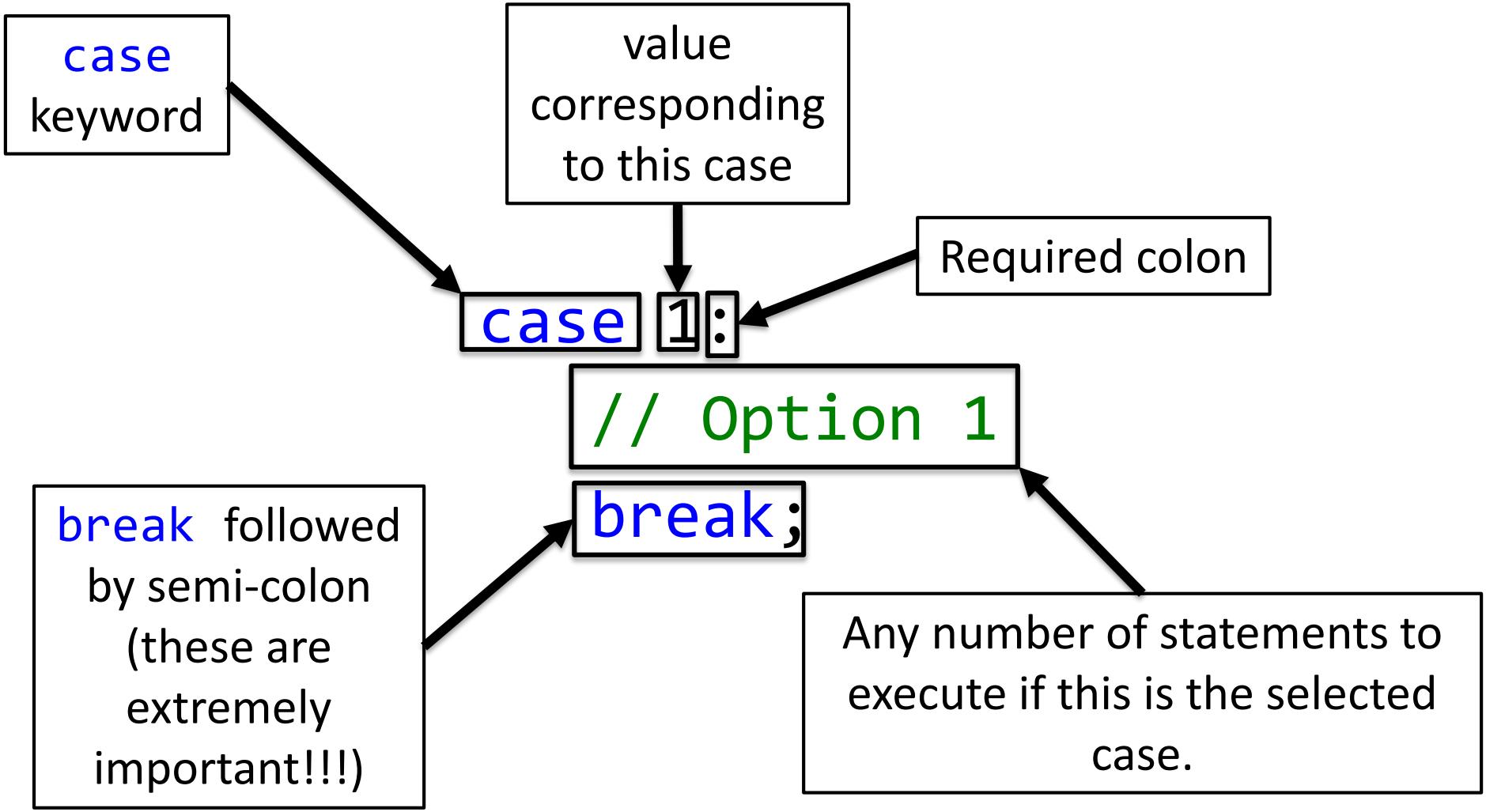
- The previous slide's code can be rewritten as a `switch` statement:

```
switch (select)
{
case 1:
   // Option 1
   break;
case 2:
   // Option 2
   break;
case 3:
   // Option 3
   break;
case 4:
   // Option 4
   break;
default:
   // Invalid option
   break;
}
```

Switch statement syntax



Switch statement syntax – case block



Default case

- The default case is what's executed if the variable was not equal to any of the other cases
- Generally, it's a good practice to have a default case

```
switch (select)
{
case 1:
    // Option 1
    break;
default:
    // Invalid option
    break;
}
```

In this example, the `default` case is selected in the event that `select != 1`

Default case, Cont'd

```
switch (select)
{
case 1:
   // Option 1
   break;
case 2:
   // Option 2
   break;
case 3:
   // Option 3
   break;
case 4:
   // Option 4
   break;
default:
   // Invalid option
   break;
}
```

In this example, the `default` case is selected in the event that

```
(select != 1 &&
 select != 2 &&
 select != 3 &&
 select != 4 )
```

Switch Caveat #1

- `switch` ***only*** works on whole numbers

- This means that if you must select based on a double or string, you can't use a switch
- For example, this is not valid:

```
double test;
switch (test) // Error: Not a whole number
{
case 1.0:
    break;
default:
    break;
}
```

Switch Caveat #2

- Switch only works in the instance where the condition is that the value equals a specific number

- If you need complex behavior like “greater than 0 but less than 100” you need to use if/else if statements

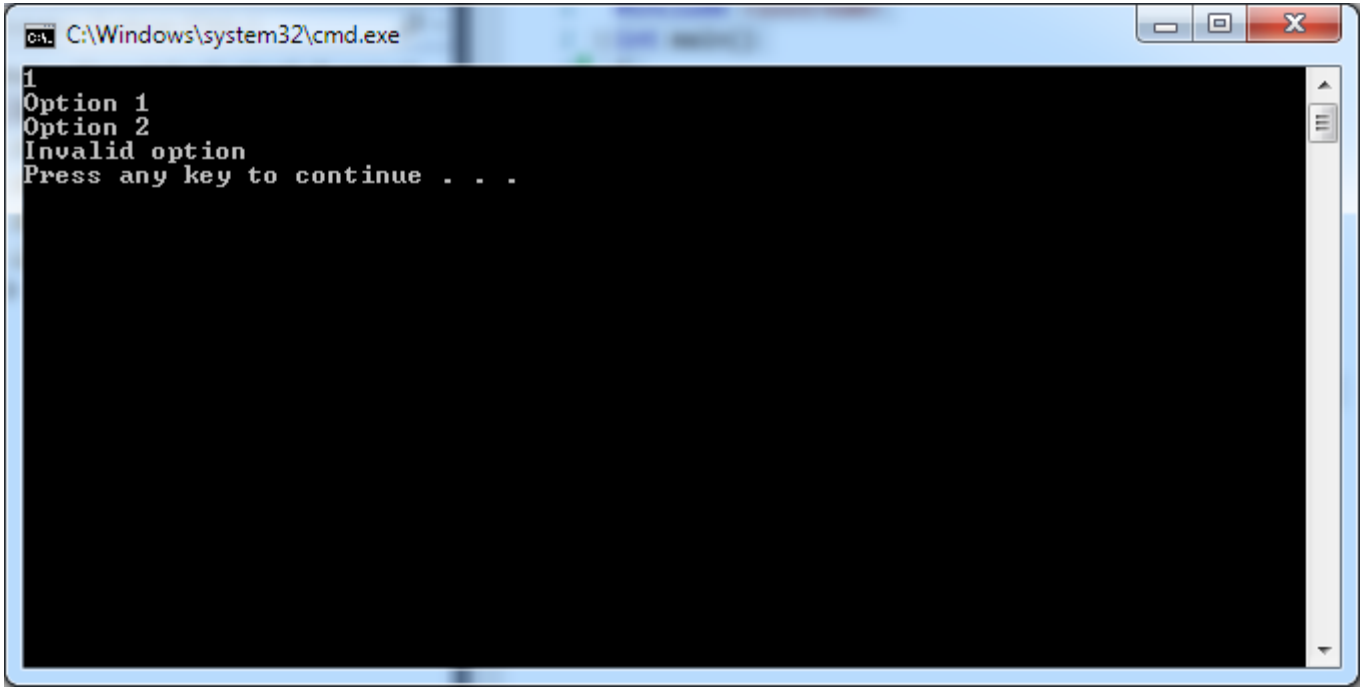
Switch Caveat #3

- Don’t forget the `break` after every case
- Otherwise, you will have weird, behavior…
- For example, this is wrong:

```
#include <iostream>
int main()
{
   int select = 0;
   std::cin >> select;
   switch (select)
   {
   case 1:
      std::cout << "Option 1" << std::endl;
   case 2:
      std::cout << "Option 2" << std::endl;
   default:
      std::cout << "Invalid option" << std::endl;
   }

   return 0;
}
```

Switch Caveat #3, Cont'd

- If we run the code on the previous slide…



- ?!?!?!?!

Switch Caveat #2 – case fall-through

```
switch (select)
{
case 1:
    std::cout << "Option 1" << std::endl;
case 2:
    std::cout << "Option 2" << std::endl;
default:
    std::cout << "Invalid option" << std::endl;
}
```

Case fall-through can be sort of useful…

- There are some instances where taking advantage of the fall-through might be useful

- But this is a more advanced use of `switch` statements…for now just always put a `break` after each `case`!

Switch Programming Hint

- Keep case-related calculations and statements inside each case
- Keep generic-case related calculations and statements outside of the switch
- Example:
  - A program that has options for add, subtract, multiply, and divide using a switch.
  - The user is to enter two numbers and then choose an operation.
  - Regardless of their choice of operation, they still require two numbers
    - Ask the user for the two numbers BEFORE the switch to save time on code writing.
- Note:
  - Won't always work the first time around. You may have to play a little with it first to see what can be written outside.

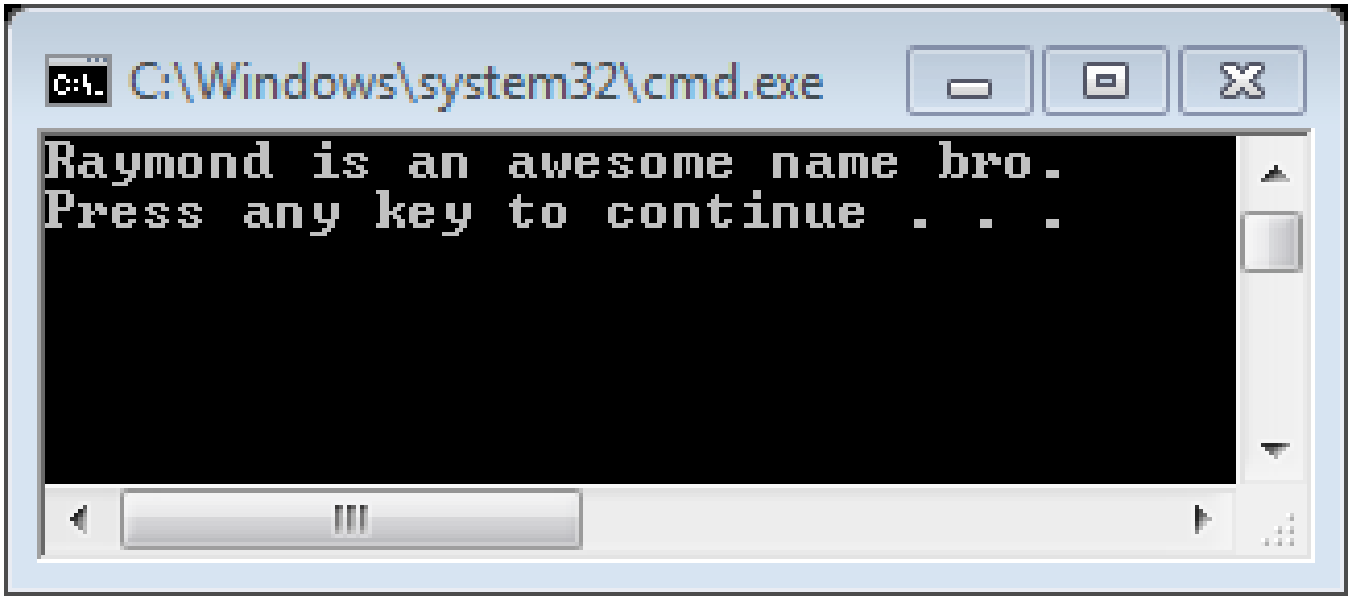
Notes on Conditionals

- You can compare text too

```
int main()
{
    std::string name = "Raymond";

    if (name == "Raymond")
    {
        std::cout << name << " is an awesome name bro.";
    }
    else
    {
        std::cout << name << " is a lame name bro.";
    }

    std::cout << std::endl;

    return 0;
}
```

Notes on Conditionals

- We get…



Notes on Conditionals

- Don’t forget, C++ is ***case-sensitive***

Notes on Conditionals

- If you want to satisfy two conditions at once, you can combine the conditions using a ***Boolean operator***

- For example:

$$0 \leq X \leq 10$$

- This is like saying “X is greater than or equal to 0” AND “X is less than or equal to 10”

```
((X >= 0) && (X <= 10))
```

Notes on Conditionals

- If we flip the range from the last slide, we get:

$$X \leq 0$$

$$X \geq 10$$

- Our condition becomes: “X less than or equal to 0” OR “X greater than or equal to 10”

```
((X <= 0) || (X >= 10))
```

Notes on Conditionals

- Briefly mentioned order of operations for conditionals

Operator	Description
`< or <=`	Less than (or equal to)
`> or >=`	Greater than (or equal to)
`== or !=`	Equal to or Not equal to
&	Bitwise AND
^	Bitwise XOR (exclusive OR)
|	Bitwise OR
&&	Logical AND
||	Logical OR

Notes on Conditionals

- Don’t always rely on the order of operations to sort out which evaluation to do first
- Always make your conditionals explicitly clear
- Use ***parentheses!!!***

Float Comparison

- Remember when we said decimals are ***approximations***?

- If we run this code…

```
int main()
{
    double num1 = 1.9, num2 = 0.9;

    if ((num1 - num2) == 1.0)
    {
        std::cout << "TRUE" << std::endl;
    }
    else
    {
        std::cout << "FALSE" << std::endl;
    }

    return 0;
}
```

Float Comparison

- We get…

Float Comparison

- How do we deal with our minds being blown?
- If decimals are approximations, there is some *wiggle-room* we can play with
- Referred to as a ***tolerance***
- Simply put:
  - Tolerance is a percentage of a value such that

$$\textbf{Value} \pm \textbf{Tolerance} \approx \textbf{Value}$$

Tolerance

- Think of tolerance like a range

- The actual value is between the *expected* value minus the tolerance and the *expected* value plus the tolerance

$$(eValue - tol) \leq aValue \leq (eValue + tol)$$

Tolerance

- What's a good tolerance?
- It depends…
- Generally, the smaller the better
  - But how small is too small?

Lab Practical #3