Boolean Logic; Conditionals

ITP 165 – Fall 2015
Week 2, Lecture 1

Where we left off last time…

```
#include <iostream>
int main()
{
    std::cout << "Enter a number:" << std::endl;
    int value = 0;
    std::cin >> value;
    value *= 5;
    std::cout << value << std::endl;
    return 0;
}
```

Boolean Logic

- Values are either ***true*** or ***false***

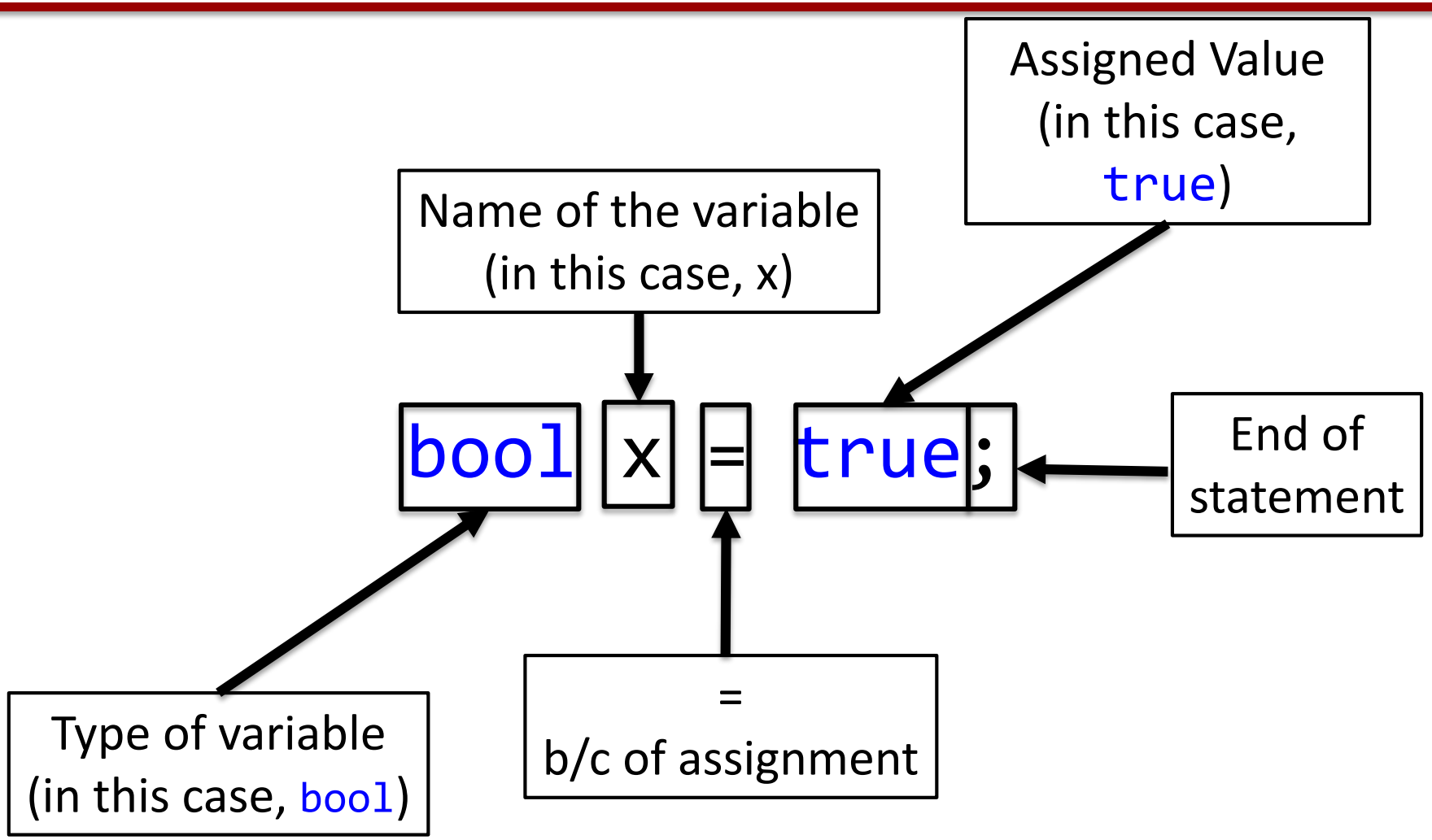
- Named after George Boole

Bool Type

- The `bool` type allows you to declare a value that can only be `true` or `false`

```
// Basic bool declarations
bool test = true;
bool anotherTest = false;
```

Variable Declaration w/ Assignment Syntax



Comparisons

- You can perform comparisons with values of any type in order to get a `true` or `false` Boolean value

```
int x = 5;
// == means "is equal to"
// Since x is 5, isFive is true
bool isFive = x == 5;

// != means "is not equal to"
// Since x is 5, this is false
bool isNotFive = x != 5;

// > means "greater than"
// Since x is not > 10, this is false
bool isGreater = x > 10;
```

Comparison Operators

- Each of these operators perform comparisons between variables

Operator	Comparison
==	Is equal to
!=	Is not equal to
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to

Boolean Expressions

- We can also express slightly more complicated Boolean expressions using Boolean operators

```
bool a = true;
bool b = false;

// ! is "not" or the logical negation
// So since a is true, !a would be false
bool c = !a;

// We can also make more complex expressions...
bool d = !(20 < 10);
```

And/Or

- Given a and b from the last slide…

```
// || means "or" as in if either are true,
// the Boolean expression is true
bool e = a || b;

// && means "and" as in if both are true,
// the Boolean expression is true
bool f = a && b;
```

- && and || have lower precedence than the comparison operators

Truth Table

a	b	!a	a || b	a && b
false	false	true	false	false
false	true	true	true	false
true	false	false	true	false
true	true	false	true	true

Short Circuiting

- If a is `true`, then a `||` b is always `true`, so there is no need to evaluate b

- Similarly, if a is `false`, then a `&&` b is always `false`, so there is no need to evaluate b

- This is called ***short circuiting*** – we won't really use it for now, but we will at some point in the future

More Compound Boolean Expressions

- A few more complex examples…

```
int x = 20;
// This is true since x is
// less than 100 and greater than 0
bool test = (x < 100) && (x > 0);

int x = -20;
bool allowNeg = true;
// Is this true or false?
bool test2 = (x < 100) && (x > 0 || allowNeg);
```

More Compound Boolean Expressions

- A few more examples…

```
bool isTrue = true;
int x = 5;
//Result is FALSE: isTrue short circuits the OR
//but the NOT negates the true.
bool result = !(isTrue || (x < 100));

bool isTrue = false;
int x = 5;
//Result is TRUE: isTrue short circuits the AND
//but the NOT negates the false.
bool result = !(isTrue && (x < 100));
```

More Compound Boolean Expressions

- The NOT operator can be overlooked because it comes BEFORE the expression but is evaluated AFTER the expression in the parentheses is evaluated.

- Sometimes tricky…

- Using NOT in the last two expressions is an example of De Morgan's Law

De Morgan's Law

- A law that allows you to, in some cases, simplify Boolean expressions

- Not (A and B) is the same as (Not A) or (Not B):

```
!(a && b) == (!a || !b)
```

- Not (A or B) is the same as (Not A) and (Not B):

```
!(a || b) == (!a && !b)
```

Input Booleans

- How do we input Booleans from the keyboard? (cin)
- Will this work?
  - TRUE
- NO! Any text will give us errors
  - While it's not technically incorrect, it won't behave the way we want
- Let's use output to see how C++ handles booleans

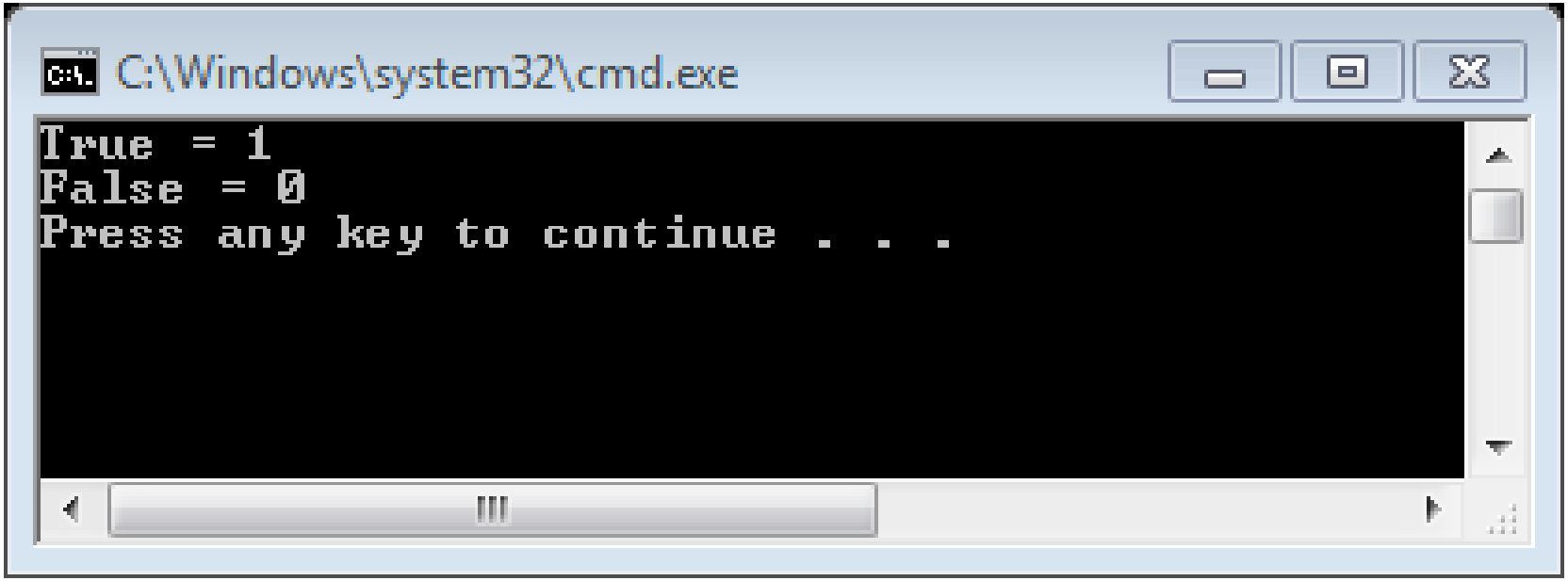
Output Booleans

- Consider:

```
int main()
{
  bool trueBool = true, falseBool = false;

  std::cout << "True = " << trueBool << std::endl;
  std::cout << "False = " << falseBool << std::endl;
  return 0;
}
```

Output Booleans



Booleans

- Booleans are represented as 0's, and 1's (like binary)

- In reality:
  - 0 = False
  - Any number except 0 = True

- For now, use 1 and 0 for true and false when asking for input

Conditionals

- Conditionals are powerful statements which allow specific code to run ***only if*** the condition is true

- The “condition” is a Boolean expression

- Without conditionals, it would not be possible for programs to do any sort of basic decision-making

Sample Program with Conditional

```
#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.";
    }

    return 0;
}
```

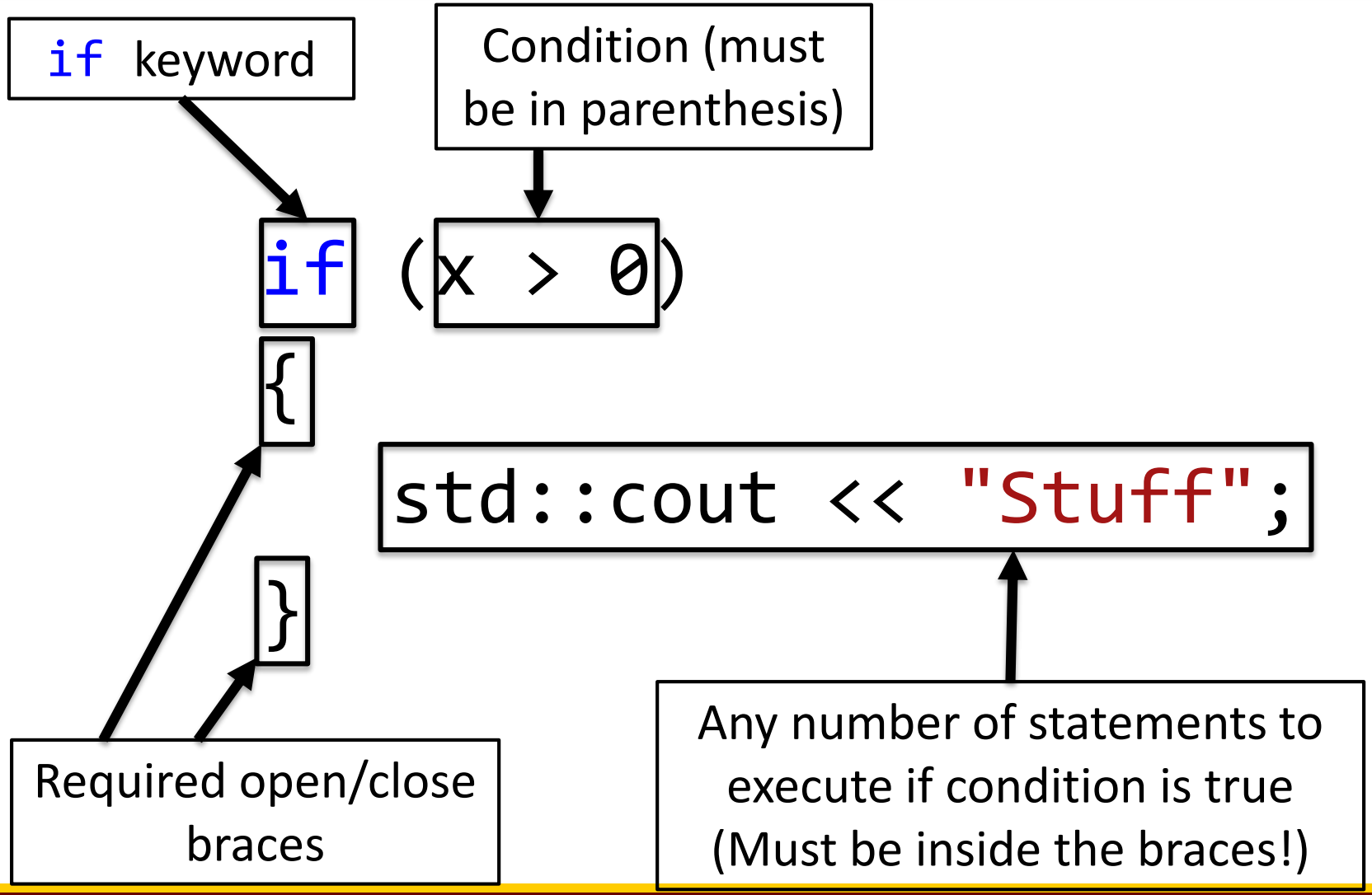
Sample Program with Conditional

```
#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.";
    }

    return 0;
}
```

Basic If Statement Syntax



If Statement w/ Multiple sub-statements

- All of the statements inside the braces will execute only if the condition is true:

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

Matching Braces

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

    if (x > 0)
    {
        // ...
    }

    return 0;
}
```

Matching Braces, Cont'd

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

    if (x > 0)
    {
        // ...
    }

    return 0;
}
```

Matching Braces, Cont'd

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

    if (x > 0)
    {
        // ...
    }

    return 0;
}
```

Scope

- A ***scope*** starts at an opening brace, and ends at the matching closing brace

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

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

Scopes and Indenting

- When you start a new scope, you should ALWAYS add indention for the statements within that scope
- This makes the code easier to digest

```
// This is poorly indented, so it's difficult to read
int main()
{
int x = 0;

if (x > 0)
{
// ...
}

return 0;
}
```

Scopes and Indenting, Cont'd

```
// Code is properly indented, easier to read!
int main()
{
    int x = 0;

    if (x > 0)
    {
        // ...
    }

    return 0;
}
```

Scopes and Variables

- A variable exists only until the end of the scope it was declared in
- For example:

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

    if (x > 0)
    {
        int y = 0;
    }

    return 0;
}
```

y is declared in the if scope.

The `if` scope ends here. So y no longer exists after this brace.

Scopes and Variables, Cont'd

- So this would be an error…

```
#include <iostream>
int main()
{
    int x = 0;

    if (x > 0)
    {
        int y = 0;
    }
    std::cout << y; // Error: y is undefined

    return 0;
}
```

Scopes and Variables, Cont'd

- This would be okay, however...

```
#include <iostream>
int main()
{
    int x = 0;

    if (x > 0)
    {
        std::cout << x; // Works
    }

    return 0;
}
```

x is declared in the main scope.

The main scope ends here. So x is valid until this closing brace.

Scopes and Variables, Summary

- Always declare a variable in the topmost scope that it is necessary
- If a temporary variable only needs to exist inside an if statement, it can be declared inside the if statement
- If a variable must exist after the if statement, it must be declared in the scope before the if statement

Lab Practical #2