We are 183

L05: Week 4 - Monday

Engineering Career Fair: January 26th-27th 1 – 6pm

North Campus
Hundreds of Companies
Go talk to them
Discover what they are looking for

Last Time… on EECS 183

User-defined Functions

Testing

Scope

RMEs

i>Clicker #1

```
void foo(string s, int n) {
    [...]
}
```

Which of the following is the correct way to **call** this function?

a) `int x = foo('3.14', 7);`
b) `foo("3.14", 7.0);`
c) `int x = foo(3.14, 7);`
d) None of the above

i>Clicker #1

```
void foo(string s, int n) {
    [...]
}
```

Which of the following is the correct way to **call** this function?

a) `int x = foo('3.14', 7);`

b) `foo("3.14", 7.0);`

c) `int x = foo(3.14, 7);`

d) None of the above

i>Clicker #2: What does this print?

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

void increment(int);

int main() {
    int x = 4;
    cout << x;
    increment(x);
    cout << x;
}

void increment(int n) {
    n = n + 1;
    cout << n;
    return;
}
```

A) 455
B) 555
C) 444
D) 454
E) Error

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

void increment(int);

int main() {
    int x = 4;
    cout << x;
    increment(x);
    cout << x;
}

void increment(int n) {
    n = n + 1;
    cout << n;
    return;
}
```

i>Clicker #2: What does this print?

A) 455
B) 555
C) 444
[✓ D) 454]
E) Error

i>Clicker #3: What does this print?

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

void increment(int);

int main() {
    int x = 4;
    cout << x;
    increment(x);
    cout << x;
}

void increment(int x) {
    x = x + 1;
    cout << x;
    return;
}
```

A) 455
B) 555
C) 444
D) 454
E) Error

i>Clicker #3: What does this print?

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

void increment(int);
```

```
int main() {
    int x = 4;
    cout << x;
    increment(x);
    cout << x;
}
```

Scope of x in 'main'

```
void increment(int x) {
    x = x + 1;
    cout << x;
    return;
}
```

Scope of x in 'increment'

A) 455
B) 555
C) 444
[✓ D) 454]
E) Error

i>Clicker #4
assume all compiles

```
int main() {
    int n = foo(0) + 1;
    cout << n << endl;
}

int bar(int n) {
    cout << n << " ";
    return n + 1;
}

int foo(int n) {
    n = bar(n) + 1;
    cout << n << " ";
    return n;
}
```

What would this program print?

A) 0 2 3
B) 1 2 3
C) 0 1 2
D) 0 1 1

i>Clicker #4

```
int main() {
    int n = foo(0) + 1;
    cout << n << endl;
}

int bar(int n) {
    cout << n << " ";
    return n + 1;
}

int foo(int n) {
    n = bar(n) + 1;
    cout << n << " ";
    return n;
}
```

What would this program print?

A) 0 2 3
B) 1 2 3
C) 0 1 2
D) 0 1 1

Today

Requires, Modifies, Effects (RMEs)
Global and Local Variables
Branches
Boolean Operators

Write the code "ToDo" – in teams

```
/**
 * compute converts a temperature
 * in Fahrenheit to Celsius
 * Requires: x is a temperature in Fahrenheit
 * Modifies: nothing
 * Effects: returns the same temperature in Celsius
 */
double compute(double x);
```

names tell us the purpose of a variable or function

helps in understanding what needs to be done

code is meant to be *read* as well as run

Write the code "ToDo" – in teams

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * Requires: tempF is a temperature in Fahrenheit
 * Modifies: nothing
 * Effects: returns the same temperature in Celsius
 */
double fahrenheitToCelsius(double tempF);
```

Note: to calculate celsius
take the fahrenheit temp
subtract 32
multiply the result by 5/9th

Write the code "ToDo" – in teams

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * Requires: tempF is a temperature in Fahrenheit
 * Modifies: nothing
 * Effects: returns the same temperature in Celsius
 */
double fahrenheitToCelsius(double tempF);
```

RME: describes **what** function does, **not** how

Write the code "ToDo" -- Solution

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * Requires: tempF is a temperature in Fahrenheit
 * Modifies: nothing
 * Effects: returns the same temperature in Celsius
 */
double fahrenheitToCelsius(double tempF);
```

```
double fahrenheitToCelsius(double tempF) {
    return (tempF - 32) * 5.0 / 9.0;
}
```

Requires, Modifies, Effects

- RMEs are a common convention for functions:
- **Requires** – What inputs do the arguments take? Can they be any value, or are there additional constraints (for example, must be positive)?
- **Modifies** – Are the inputs going to be changed by the function? How are they going to be changed?
- **Effects** – What does the function do? What value is returned? Does it print to cout?

Magic Numbers

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * Requires: tempF is a temperature in
 *           Fahrenheit
 * Modifies: nothing
 * Effects: calculates the same temperature in
 *          Celsius
 */
double fahrenheitToCelsius(double tempF) {
    return (tempF - 32) * 5.0 / 9.0;
}
```

What do these numbers mean?

Magic Numbers

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * RME...
 */
double fahrenheitToCelsius(double tempF) {
    double C_DEGREE_PER_F_DEGREE = 5.0 / 9.0;
    double FREEZING_POINT_F = 32;
    return (tempF - FREEZING_POINT_F) *
        C_DEGREE_PER_F_DEGREE;
}
```

Better, but should be constants

Magic Numbers

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * RME...
 */
double fahrenheitToCelsius(double tempF) {
    const double C_DEGREE_PER_F_DEGREE =
        5.0 / 9.0;
    const double FREEZING_POINT_F = 32;
    return (tempF - FREEZING_POINT_F) *
        C_DEGREE_PER_F_DEGREE;
}
```

Better, but need to be defined in every function that uses them

Global Constants

```
const double C_DEGREE_PER_F_DEGREE = 5.0 / 9.0;
const double FREEZING_POINT_F = 32;
```

```
int main() {
}
```

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * RME...
 */
double fahrenheitToCelsius(double tempF) {
    return (tempF - FREEZING_POINT_F) *
        C_DEGREE_PER_F_DEGREE;
}
```

Global Constants

```
const double C_DEGREE_PER_F_DEGREE = 5.0 / 9.0;
const double FREEZING_POINT_F = 32;
```

```
int main()
}
```

Only use const global constants

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * RME...
 */
double fahrenheitToCelsius(double tempF) {
    return (tempF - FREEZING_POINT_F) *
        C_DEGREE_PER_F_DEGREE;
}
```

Global Constants

```
double C_DEGREE_PER_F_DEGREE = 5.0 / 9.0;
double FREEZING_POINT_F = 32;
```

```
int main()
}
```

NEVER use global variables

```
/**
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * RME...
 */
double fahrenheitToCelsius(double tempF) {
    return (tempF - FREEZING_POINT_F) *
        C_DEGREE_PER_F_DEGREE;
}
```

Scope of Global Variables

```
const double C_DEGREE_PER_F_DEGREE = 5.0 / 9.0;
const double FREEZING_POINT_F = 32;

int main() {
}

/*
 * fahrenheitToCelsius converts a temperature
 * in Fahrenheit to Celsius
 * RME...
 */
double fahrenheitToCelsius(double tempF) {
    return (tempF - FREEZING_POINT_F) *
        C_DEGREE_PER_F_DEGREE;
}
```

Scope is from declaration to end of file

Scope of Globals

```
/**
 * Pluralizes a word if needed.
 *
 * RME..
 */
```

```
string pluralize(string singular, string plural,
                 int number);

int main() {
   ...
}


...
```

Also true for function declarations

i>Clicker #5: What does this print?

```
#include <iostream>
using namespace std;
const int x = 42;
void increment(int);

int main() {
    int x = 4;
    cout << x;
    increment(x);
    cout << x;
}

void increment(int n) {
    n = n + 1;
    cout << n;
    return;
}
```

A) 455
B) 555
C) 444
D) 454
E) Error

i>Clicker #5: What does this print?

```
#include <iostream>
using namespace std;
const int x = 42;
void increment(int);

int main() {
    int x = 4;
    cout << x;
    increment(x);
    cout << x;
}

void increment(int n) {
    n = n + 1;
    cout << n;
    return;
}
```

A) 455
B) 555
C) 444
D) 454
E) Error

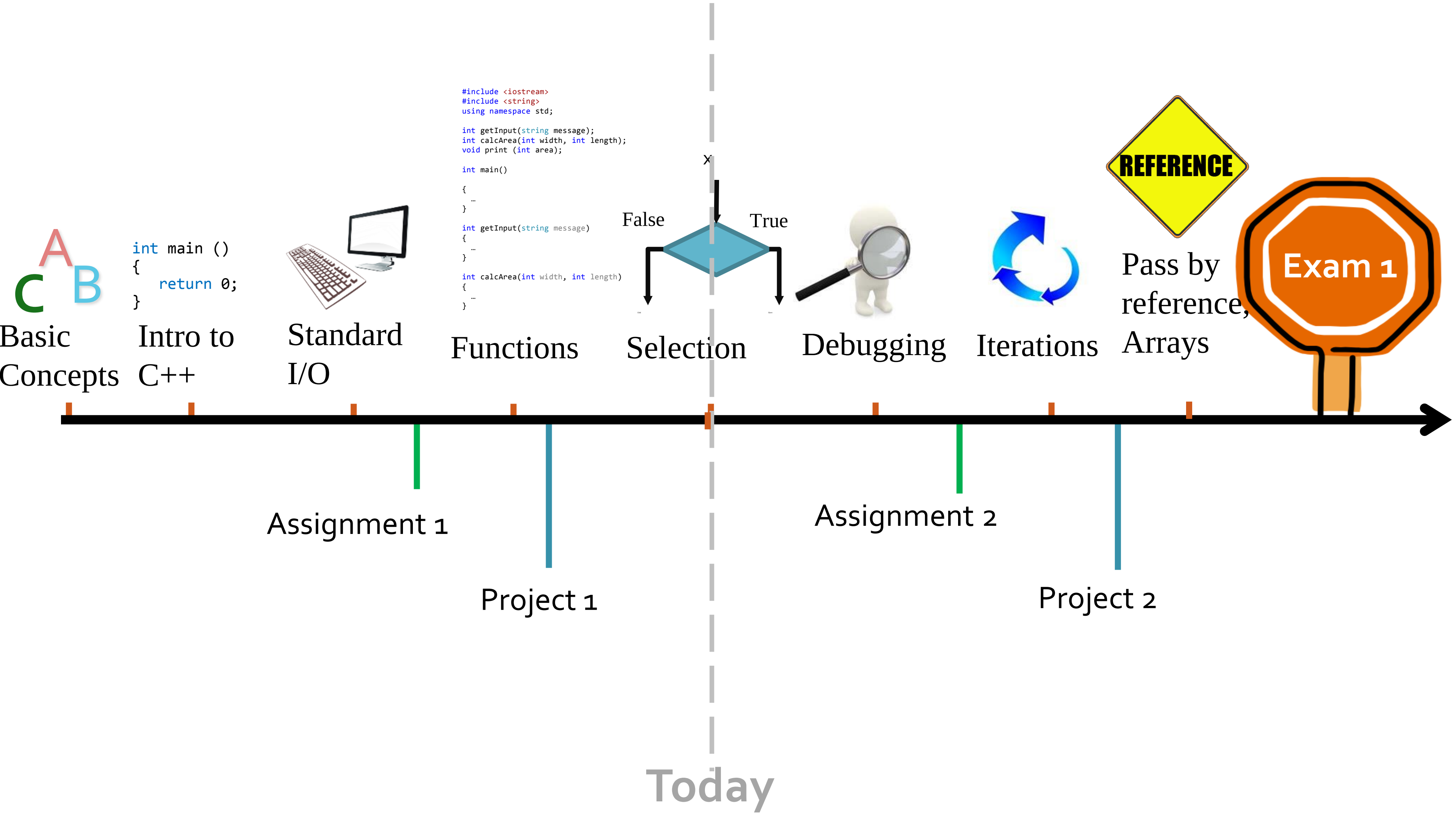
will use 'x' in the closest scope

Nested Scope: Horrid code – do NOT write this

```
int main()
{
    int n = 1;
    {
        int n = 2;
        {
            int n = 3;
            cout << n << endl;
        }
    }
}
```

This would print "3".

The Course So Far



Basic Concepts

Intro to C++

```
int main ()
{
    return 0;
}
```

Standard I/O

Functions

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

int getInput(string message);
int calcArea(int width, int length);
void print (int area);

int main()

{
  …
}

int getInput(string message)
{
  …
}

int calcArea(int width, int length)
{
  …
}
```

Selection

Debugging

Iterations

Pass by reference, Arrays

Exam 1

Assignment 1

Project 1

Assignment 2

Project 2

Today

The Fun Stuff

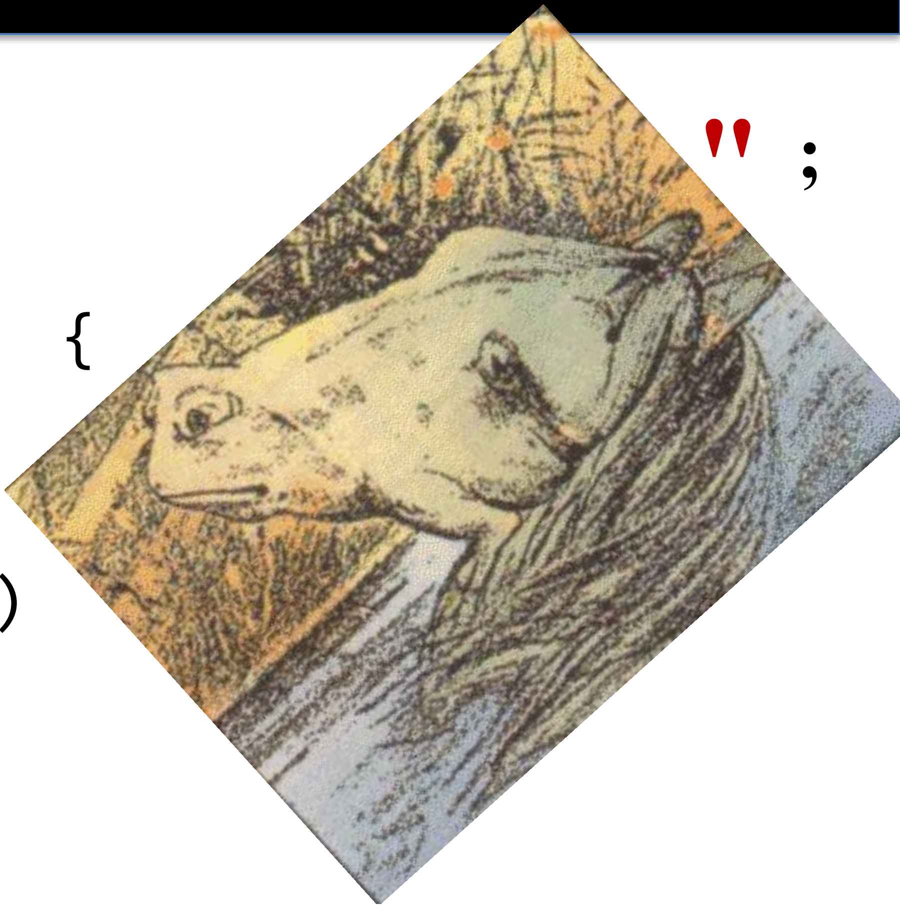
- So far, your programs can only do one thing:
  - It can print out different numbers (e.g. How many cupcakes to make), but it can't print out other stuff (e.g. If you want muffins instead)

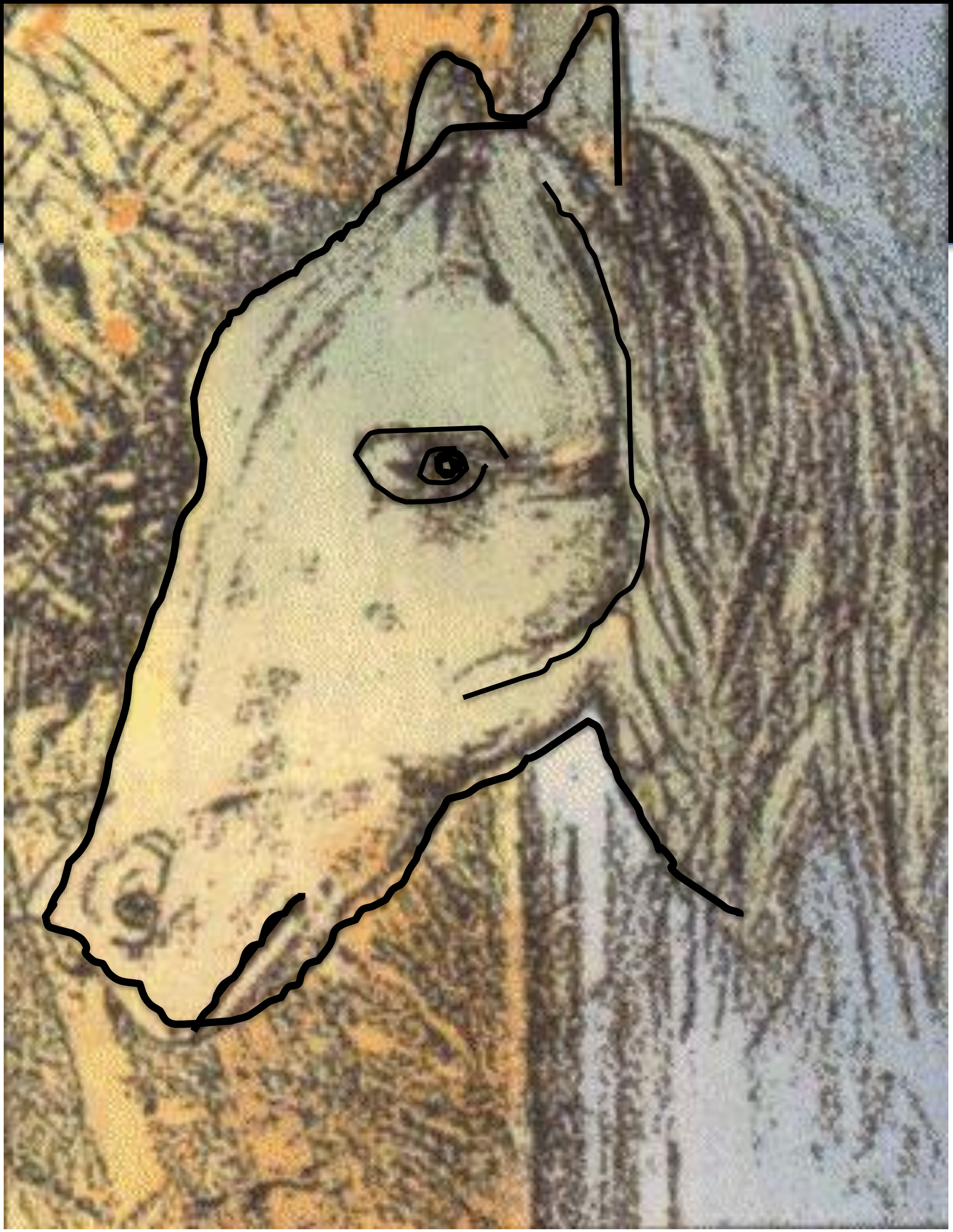
- That's what today's lecture is about.

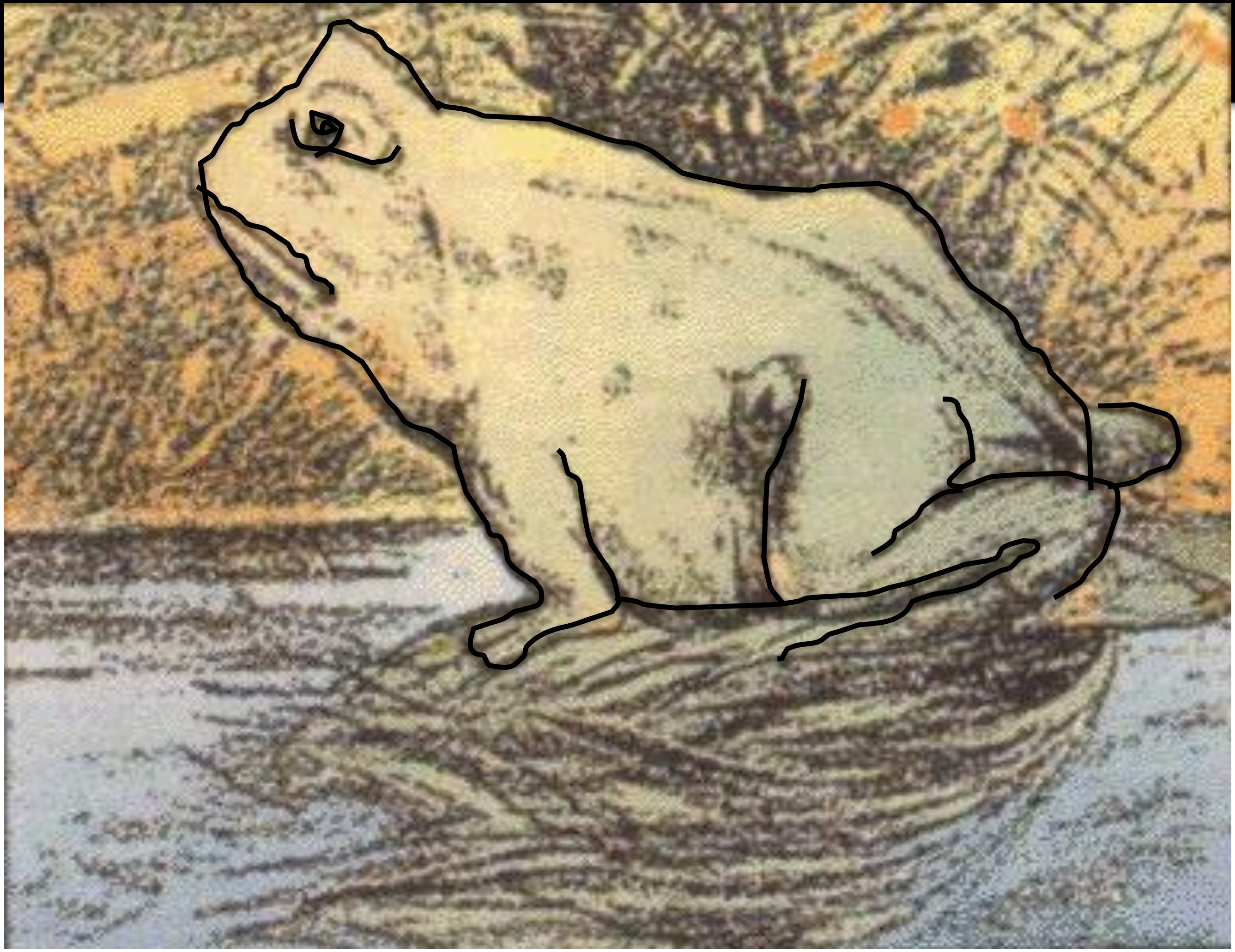
```
void fun () {
    char iClicker;

    Picture P    "  =                        " ;

    if (P isA "Horse Head")  {
        cout << "Press A";
    } else if (P isA "Frog")
        cout << "Press B";
    }
    cin >> iClicker;
}
```







Chat Bots

```
Are you happy [yes/no]?
```

- yes:
  - say "Yay!"
- Otherwise:
  - say "Oh no! Robot hugs!!!"

Branches

- If-statements to the rescue!

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
   if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

The **if** keyword means it’s the start of a branch

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

This is the *condition*, which determines what happens. This particular condition is true if `number` is equal to 1.

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

This is the *true branch*; code here will execute only if the condition is true.

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

Otherwise
key word that separates the two branches

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

This is the *false branch*; code here will execute only if the condition is **not** true.

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

In English, this says "**if** the number is 1, then return the singular form of the word; **otherwise**, return the plural."

Branches

```
if (2 > 1) {
    cout << "2 is bigger than 1";
} else {
    cout << "2 is not bigger than 1";
}
```

What gets printed out?

Branches

```
if (2 > 1) {
    cout << "2 is bigger than 1";
} else {
    cout << "2 is not bigger than 1";
}
```

What gets printed out?

"2 is bigger than 1"

Branches

```
if (-1 <= 2) {
    cout << "The sky is green.";
} else {
    cout << "Water is dry.";
}
```

What gets printed out?

Branches

```
if (-1 <= 2) {
    cout << "The sky is green.";
} else {
    cout << "Water is dry.";
}
```

What gets printed out?

`"The sky is green."`

i>Clicker #6

```
int a = 0;
int b = 1;
if (a < b) {
    a = 1;
} else {
    a = 2;
}
cout << a << endl;
```

What gets printed?

A) 0

B) 1

C) 2

D) None of the above

i>Clicker #6

```
int a = 0;
int b = 1;
if (a < b) {
    a = 1;
} else {
    a = 2;
}
cout << a << endl;
```

What gets printed?
A) 0
B) 1
C) 2
D) None of the above

i>Clicker #7

```
int a = 0;
int b = 1;
if (a < b) {
    a = 1;
    if (a < b) {
        a = 3;
    } else {
        a = 4;
    }
} else {
    a = 2;
}
cout << a << endl;
```

What gets printed out?

A) 1

B) 2

C) 3

D) 4

i>Clicker #7

```
int a = 0;
int b = 1;
if (a < b) {
    a = 1;
    if (a < b) {
        a = 3;
    } else {
        a = 4;
    }
} else {
    a = 2;
}
cout << a;
```

True

False

Sets a = 4

What gets printed out?

A) 1

B) 2

C) 3

D) 4

Branches

```
string pluralize(string singular,
                 string plural,
                 int number) {
    if (number == 1) {
        return singular;
    } else {
        return plural;
    }
}
```

This is ***equal to*** operator.

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

     // What goes here?

}
```

If the user says "yes", print

```
  Yay!
```

Otherwise, print

```
  Oh no! Robot hugs!!!
```

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (     ) {

    } else {

    }
}
```

If the user says "yes", print

```
  Yay!
```

Otherwise, print

```
  Oh no! Robot hugs!!!
```

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {

    } else {

    }
}
```

If the user says "yes", print

```
  Yay!
```

Otherwise, print

```
  Oh no! Robot hugs!!!
```

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {
        ???
    } else {

    }
}
```

If the user says "yes", print

    Yay!

Otherwise, print

    Oh no! Robot hugs!!!

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {
        cout << "Yay!" << endl;
    } else {

    }
}
```

If the user says "yes", print

```
  Yay!
```

Otherwise, print

```
  Oh no! Robot hugs!!!
```

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {
        cout << "Yay!" << endl;
    } else {
        ???
    }
}
```

If the user says "yes", print

`Yay!`

Otherwise, print

`Oh no! Robot hugs!!!`

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {
        cout << "Yay!" << endl;
    } else {
        cout << "Oh no! Robot hugs!!!" << endl;
    }
}
```

Intermission

Fun Fact: Branches are so known because they resemble tree branches – the program could go two or more ways at a branch.

Robot Hugs

You think I'm kidding about robot hugs:

https://www.youtube.com/watch?v=_YtpNwC7BNc

Boolean Operators

We saw the **==** operator earlier, that says two numbers/strings are equal; here are all the operators

Expression	Meaning
(a == b)	a is **equal to** b
(a != b)	a is **not equal to** b
(a > b)	a is **greater than** b
(a >= b)	a is **greater than or equal to** b
(a < b)	a is **less than** b
(a <= b)	a is **less than or equal to** b

Boolean Operators

We can also combine multiple Boolean expressions:

Expression	Meaning
• `(a` **`&&`** `b)`	**`both`** `a and b are` **`true`**
• `(a || b)`	**`at least one`** `of a and b is` **`true`**
• `(!a)`	**`not`** `a`

Boolean Operators

Shown Within Operator Precedence

Order	Operator	Meaning	Associativity
1	()	Group or cast	Left to right
2	**!x** +x -x	Not, negate	Right to left
3	* / %	Multiply, divide, modulo	Left to right
4	+ -	Add, subtract	Left to right
5	**< <= >= >**	Greater/less than (or equal to)	Left to right
6	**== !=**	(Not) equal	Left to right
7	**&&**	Logical and	Left to right
8	**||**	Logical or	Left to right
9	=	assign	Right to left

i>Clicker #8

`bool` expressions:

- `true`
- `false`
- `(a && b)`
- `(a || b)`
- `(!a)`
- `(a == b)`
- `(a != b)`
- `(a > b)`
- `(a >= b)`
- `(a < b)`
- `(a <= b)`

How do you write
"1 is less than 2
and 2 is less than 3
and 3 is less than 4"?

A) `(1 < 2 && 2 < 3 && 3 < 4)`
B) `(1 > 2 && 2 > 3 && 3 > 4)`
C) `(1 < 2 || 2 < 3 || 3 < 4)`
D) `(1 > 2 || 2 > 3 || 3 > 4)`

i>Clicker #8

- `(a && b)`
- `(a || b)`
- `(!a)`
- `(a == b)`
- `(a != b)`
- `(a > b)`
- `(a >= b)`
- `(a < b)`
- `(a <= b)`

How do you write
"1 is less than 2
and 2 is less than 3
and 3 is less than 4"?

A) `(1 < 2 && 2 < 3 && 3 < 4)`
B) `(1 > 2 && 2 > 3 && 3 > 4)`
C) `(1 < 2 || 2 < 3 || 3 < 4)`
D) `(1 > 2 || 2 > 3 || 3 > 4)`

i>Clicker #8

- `(a && b)`
- `(a || b)`
- `(!a)`
- `(a == b)`
- `(a != b)`
- `(a > b)`
- `(a >= b)`
- `(a < b)`
- `(a <= b)`

How do you write
"1 is less than 2
and 2 is less than 3
and 3 is less than 4"?

```
A) (1 < 2 && 2 < 3 && 3 < 4)
B) (1 > 2 && 2 > 3 && 3 > 4)
C) (1 < 2 || 2 < 3 || 3 < 4)
D) (1 > 2 || 2 > 3 || 3 > 4)
```

Do not write:

```
(1 < 2 < 3 < 4)
```

Truth Value

```
cout << (1 == 1) << endl;  ⇐  prints 1
cout << (0 == 1) << endl;  ⇐  prints 0
```

Truth Value

```
cout << (1 == 1) << endl; ← prints 1
cout << (0 == 1) << endl; ← prints 0


if (3) {
    cout << "true" << endl; ← this prints
} else {
    cout << "false" << endl;
}
```

Truth Value

```
cout << (1 == 1) << endl;
cout << (0 == 1) << endl;
```

prints 1

prints 0

Do NOT do this.

```
if (3) {
    cout << "true" << endl;
} else {
    cout << "false" << endl;
}
```

this prints

Use:

`value == 0` or `value != 0`

Truth and Style

```
bool isEven;
if (isEven == true) {
    ...
}
if (isEven == false) {
    ...
}
```

Not OK: redundant

```
if (isEven) {
    ...
}
if (!isEven) {
    ...
}
```

Good

Truth Value

```
cout << (1 == 1) << endl;     prints 1
cout << (0 == 1) << endl;     prints 0
```

`(1 < 2 < 3)` is evaluated as

```
((1 < 2) < 3) -> (1 < 3) -> 1     correct
```

But

```
(3 > 2 > 1) evaluates as
((3 > 2) > 1) -> (1 > 1) -> 0     INCORRECT
```

i>Clicker #9

- `(a && b)`
- `(a || b)`
- `(!a)`
- `(a == b)`
- `(a != b)`
- `(a > b)`
- `(a >= b)`
- `(a < b)`
- `(a <= b)`

Which is the only one of the following that *does* *NOT* do "a or b, but not both"?

```
A) (a || b)
B) ((a || b) && !(a && b))
C) ((a && !b) || (!a && b))
D) ((a != b) && (a || b))
```

i>Clicker #9

- `(a && b)`
- `(a || b)`
- `(!a)`
- `(a == b)`
- `(a != b)`
- `(a > b)`
- `(a >= b)`
- `(a < b)`
- `(a <= b)`

Which is the only one of the following that *does* ***NOT*** do "a or b, but not both"?

```
A) (a || b)
B) ((a || b) && !(a && b))
C) ((a && !b) || (!a && b))
D) ((a != b) && (a || b))
```

20th Century Philosophy

Bertrand Russell comes out of the hospital where his wife has just given birth.

A journalist comes up to him and excitedly asks: "Is it a girl or a boy?"

Bertrand Russell replies: "Yes."

Robot Hugs

Write a program that:

1. Asks if the user is happy
2. If they are happy,
   asks if they know they’re happy
3. If they’re happy and they know it,
   tell them to clap their hands
4. If they’re happy and they don’t know it,
   tell them now you do!
5. If they’re not happy,
   offer them robot hugs

Happy Robots

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {
        cout << "Yay!" << endl;
    } else {
        cout << "Oh no! Robot hugs!!!" << endl;
    }
}
```

Robot Hugs

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "yes") {
        cout << "Do you know it?" << endl;
        cin >> response;
        if (response == "yes") {
            cout << "clap your hands!" << endl;
        } else {
            cout << "well now you do!" << endl;
        }
    } else {
        cout << "Oh no! Robot hugs!!!" << endl;
    }
}
```

Robot Hugs

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "no") {
        cout << "Oh no! Robot hugs!!!" << endl;
        return 0;
    }
    cout << "Do you know it?" << endl;
    cin >> response;
    if (response == "yes") {
        cout << "clap your hands!" << endl;
    } else {
        cout << "well now you do!" << endl;
    }
}
```

Note the `return` to exit `main()`

Robot Hugs

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "no") {
        cout << "Oh no! Robot hugs!!!" << endl;
        return 0;
    }
    cout << "Do you know it?" << endl;
    cin >> response;
    if (response == "yes") {
        cout << "clap your hands!" << endl;
    } else {
        cout << "well now you do!" << endl;
    }
}
```

Note no `else` clause needed!

Robot Hugs

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "no") {
        cout << "Oh no! Robot hugs!!!" << endl;
        return 0;
    }
    cout << "Do you know it?" << endl;
    cin >> response;
    if (response == "yes") {
        cout << "clap your hands!" << endl;
    } else {
        cout << "well now you do!" << endl;
    }
}
```

Robot Hugs

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "no") {
        cout << "Oh no! Robot hugs!!!" << endl;
        return 0;
    }
    cout << "Do you know it?" << endl;
    cin >> response;
    if (response == "yes") {
        cout << "clap your hands!" << endl;
    } else {
        cout << "well now you do!" << endl;
    }
}
```

Robot Hugs

```
int main() {
    string response;
    cout << "Are you happy (yes/no)? ";
    cin >> response;

    if (response == "no") {
        cout << "Oh no! Robot hugs!!!" << endl;
    } else {
        cout << "Do you know it?" << endl;
        cin >> response;
        if (response == "yes") {
            cout << "clap your hands!" << endl;
        } else {
            cout << "well now you do!" << endl;
        }
    }
}
```

Next time

More conditionals and functions

Debugging

Which of the following prints 5? (good to review for exam1)

```
int calc(int a, int b) {
    a = a % 10;
    b = b / 10;
    return a - b;
}
```

A) `cout << calc(3, -29);`

B) `cout << calc(calc(3, -29), 8);`

C) `cout << calc(55, calc(58, 31));`

D) All of the above will print 5

E) Only A and B will print 5

Question

- What is the output of the following code?

```
int grade = 75;
if(grade >= 50){
    cout << "Pass ";
} else {
    cout << "Fail ";
    cout << "Done";
}
```

A) Pass
B) Pass Done
C) Fail
D) Fail Done
E) Pass Fail Done

Question

What is the output of the following code:

```
if (10 > 5) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
if (10 >= 5) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
if (10 == 5) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
if (10 != 5) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
if (true) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
if (0) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
if (-1) {
  cout << "Yes";
} else {
  cout << "No";
}
```

A) Yes
B) No
C) Prints nothing

Question

What is the output of the following code:

```
int a = 6;
int b = 2;
int c = 3 + (b <= a);
cout << c;
```

A) 3
B) 4
C) 1
D) true
E) Error

Question

Write a C++ conditional expression that evaluates to true if and only if an integer variable x is odd or divisible by 3

Question

Write a C++ conditional expression that evaluates to true if and only if an integer variable x is odd or divisible by 3

$$\underbrace{x \,\%\, 2 == 1}_{\text{x is odd}} \;\underset{\text{OR}}{||}\; \underbrace{x \,\%\, 3 == 0}_{\text{x is divisible by 3}}$$

On Your Own:

Determine the value, true or false, for each statement

```
int count = 0, limit = 10;
```

```
a) (count == 0) && (limit < 20)
b) count == 0 && limit < 20
c) (limit > 20) || (count < 5)
d) !(count == 12)
e) (count == 1) && (x < y)
f) (count < 10) || (x < y)
g) !((count < 10) || (x < y)) && (count >= 0)
h) ((limit / count) > 7) || (limit < 20)
i) (limit < 20) || ((limit/count) > 7)
j) (5 && 7) + (!6)
```

Engineering Career Fair:
Sept 28 - Sept 29   10-4 pm

- North Campus
- 800 Companies
- Go talk to them
- Discover what they are looking for

Social Robots

- https://www.youtube.com/watch?v=_YtpNwC7BNc