

This is 183

Lo2: Week 2 - Monday

# Announcements

- P1 has been released
- Discussion sections start this week
- iClicker starts on Wednesday
- Office hours start this week
  - Monday 3-6pm, Tues-Fri 3-8pm
  - Duderstadt 3<sup>rd</sup> floor, west side
  - Sign in sheet – name, issue, location
    - All workstations are numbered, give the closest one
    - You can request a CAEN account, visit CAEN desk, 1<sup>st</sup> floor

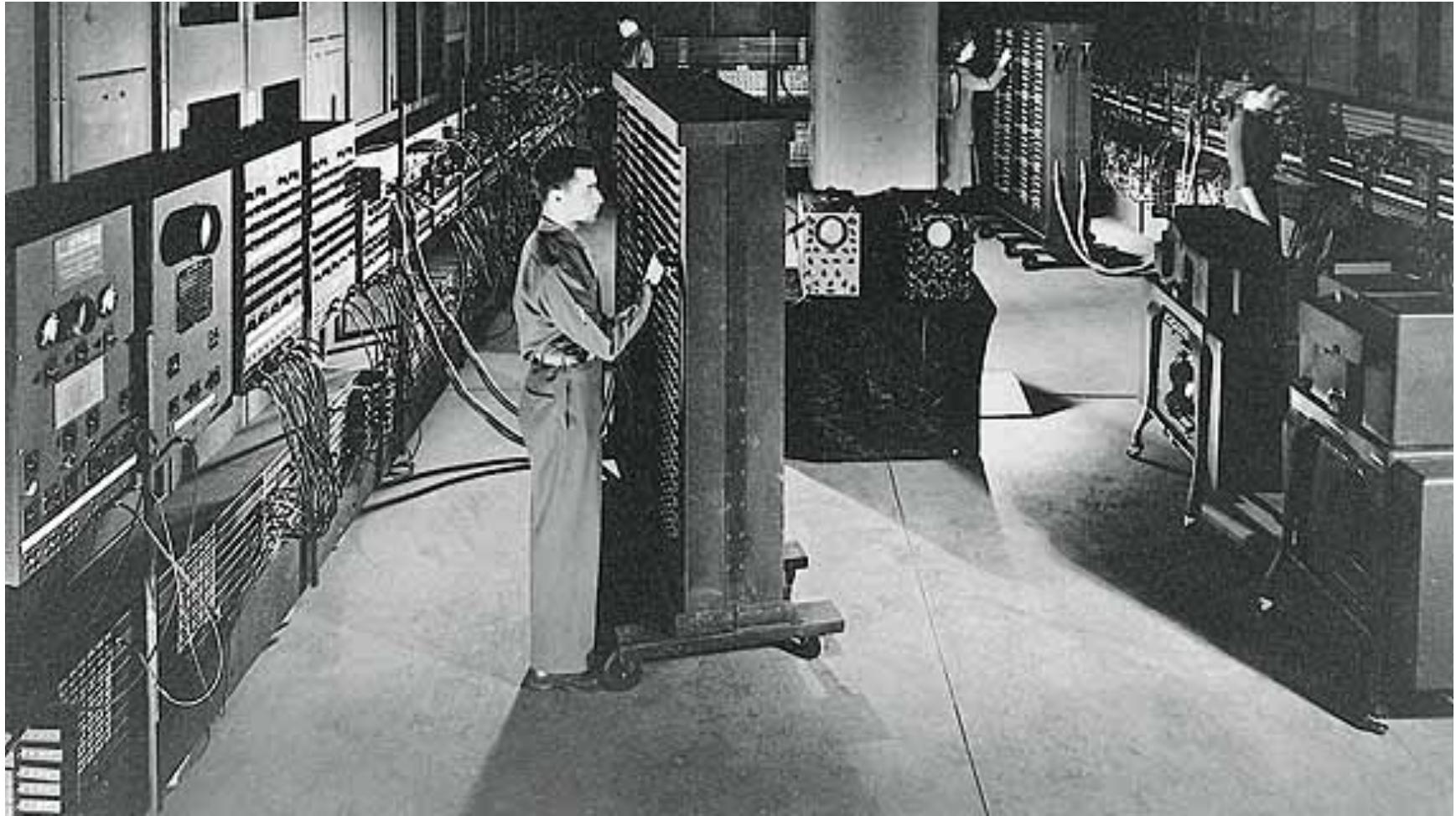
# Announcements

- Office hours start this week
  - Monday 3-6pm, Tues-Fri 3-8pm
  - Duderstadt 3<sup>rd</sup> floor, west side
  - Sign in sheet – name, issue, location
  - All workstations are numbered, give the closest one
  - You can request a CAEN account, visit CAEN desk,  
1<sup>st</sup> floor

# Peer Mentorship Opportunity

- **Who:** gEECS
- **What:** peer mentorship and tutoring groups
- **Where:** TBD
- **When:** throughout the semester
- **Note:** This is open to all students to sign up
- Offered to EECS 183, EECS 280, ENGR 101
- **Apply by Jan. 16** (application link in email announcement)

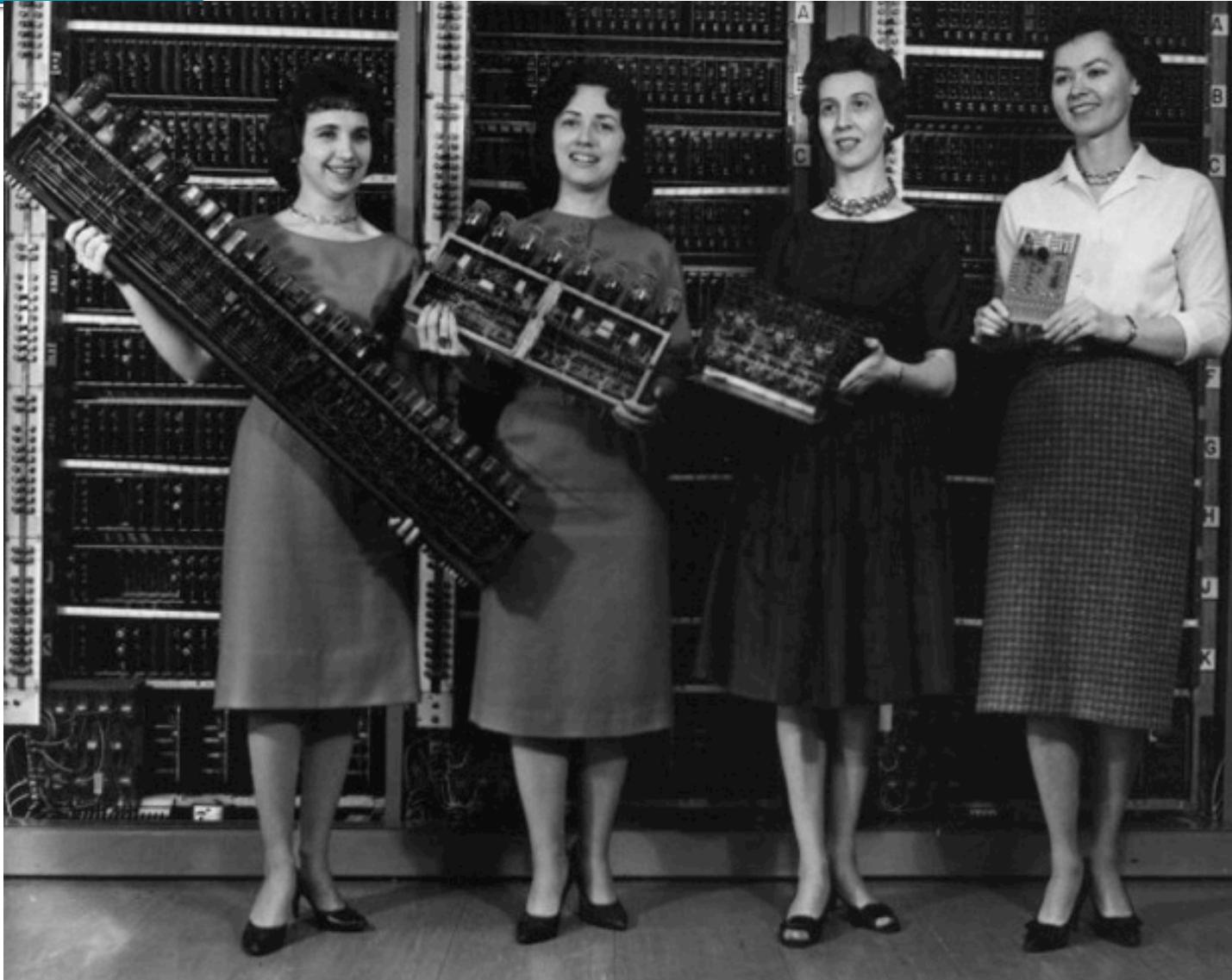
# Eniac – 1<sup>st</sup> general purpose digital computer - 1946



<http://www.youtube.com/watch?v=HJUo8t220Rk>

# Parts of the Eniac

<http://www.maximumpc.com/12-things-you-didnt-know-about-eniac/>



# 1947: A "bug"

9/9

0800 Action started  
1000 stopped - action ✓  
1300 (032) MP-MC 2.130476415  
022 PRO 2 2.130476415  
control 2.130476415

Relay 6-2 in 022 failed special speed test  
in relay 10,000 test.

1100 Relay changed  
Started Cosine Tape (Sinc check)  
1525 Started Multi Adder Test.

1545



Relay #70 Panel F  
(moth) in relay.

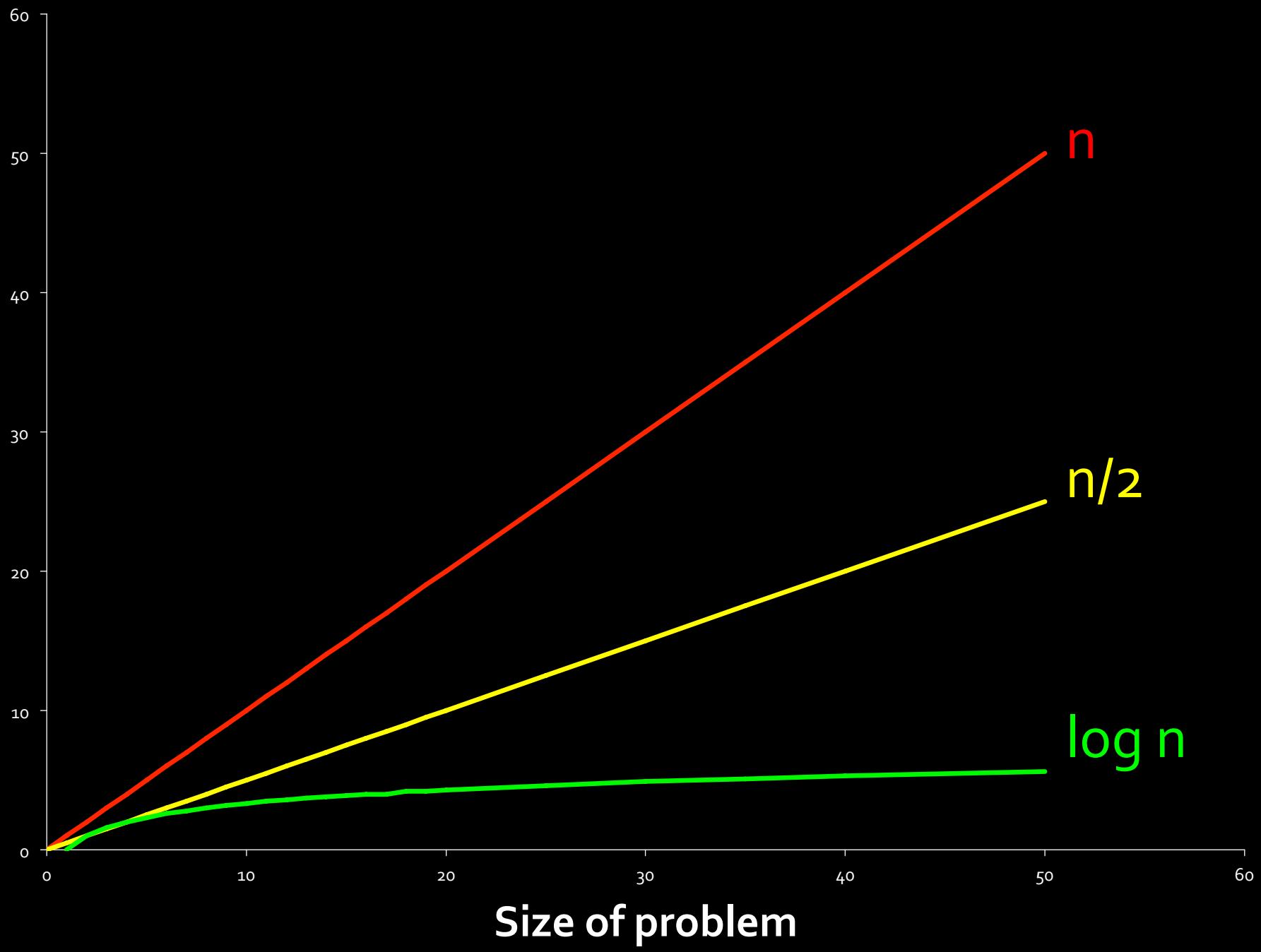
1600 Action started.  
1700 closed form.

# Last Time

# Algorithms & Pseudocode

- An algorithm is a specific approach to solve a problem.
  - Counting people in the room
  - Finding a certain page in a book
- Different algorithms vary in efficiency and speed
  - By halving the problem each time, we vastly improved the search through the book

Time to solve



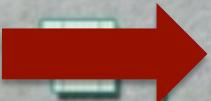
**Pseudo code**  
(RESEMBLING)  
(PROGRAMMING LANGUAGE)

## Pseudocode

let **N** = 0

for each person in room

Set **N** = **N** + 1



let **N** = 0



for each person in room



Set **N** = **N** + 1

# Loop

Do something repeatedly



let **N** = 0



For each person in room



Set **N** = **N** + 1



let **N** = 0



for each person in room

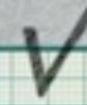


Set **N** = **N** + 1

# Testing is Critical

You must always test your algorithms.

TEST COMPLETE

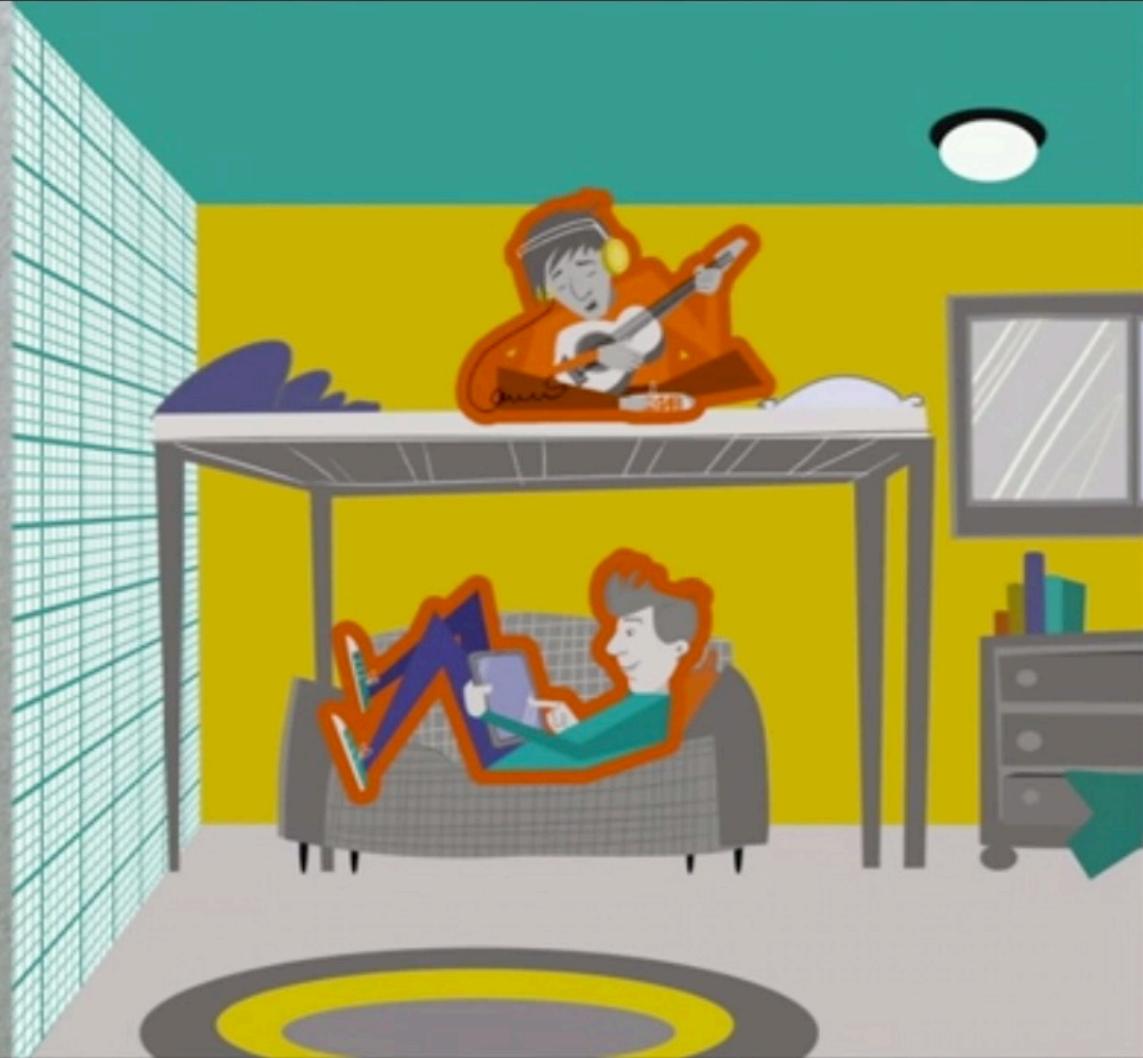


let **N** = 0

For each person in room

Set **N** = **N** + 1

2



Initial algorithm for 1 at-a-time worked ok.

TEST COMPLETE

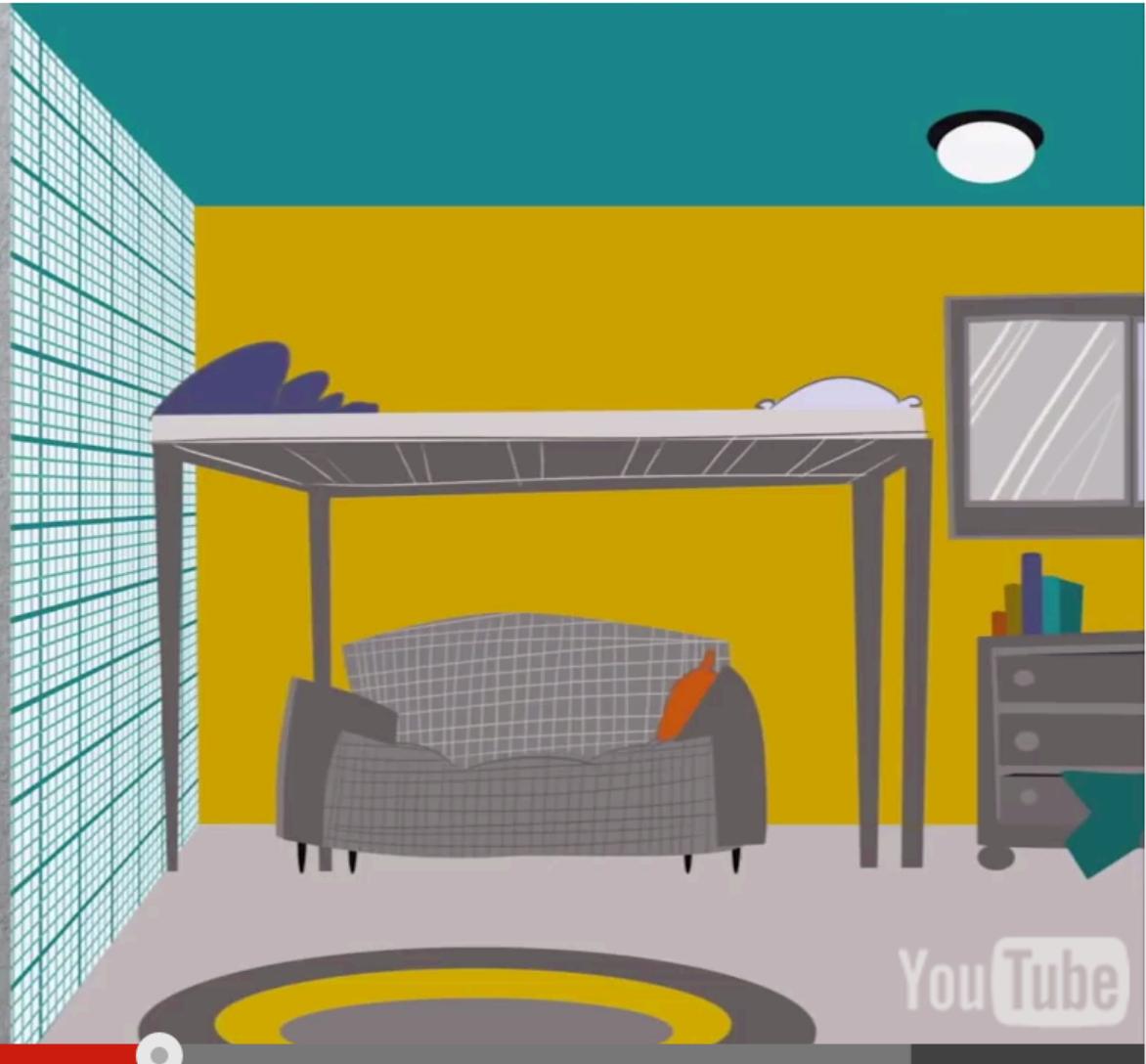


let **N** = 0

for each person in room

Set **N** = **N** + 1

0



YouTube



2:20 / 4:57



It even worked with 0 people in the room.

TEST COMPLETE

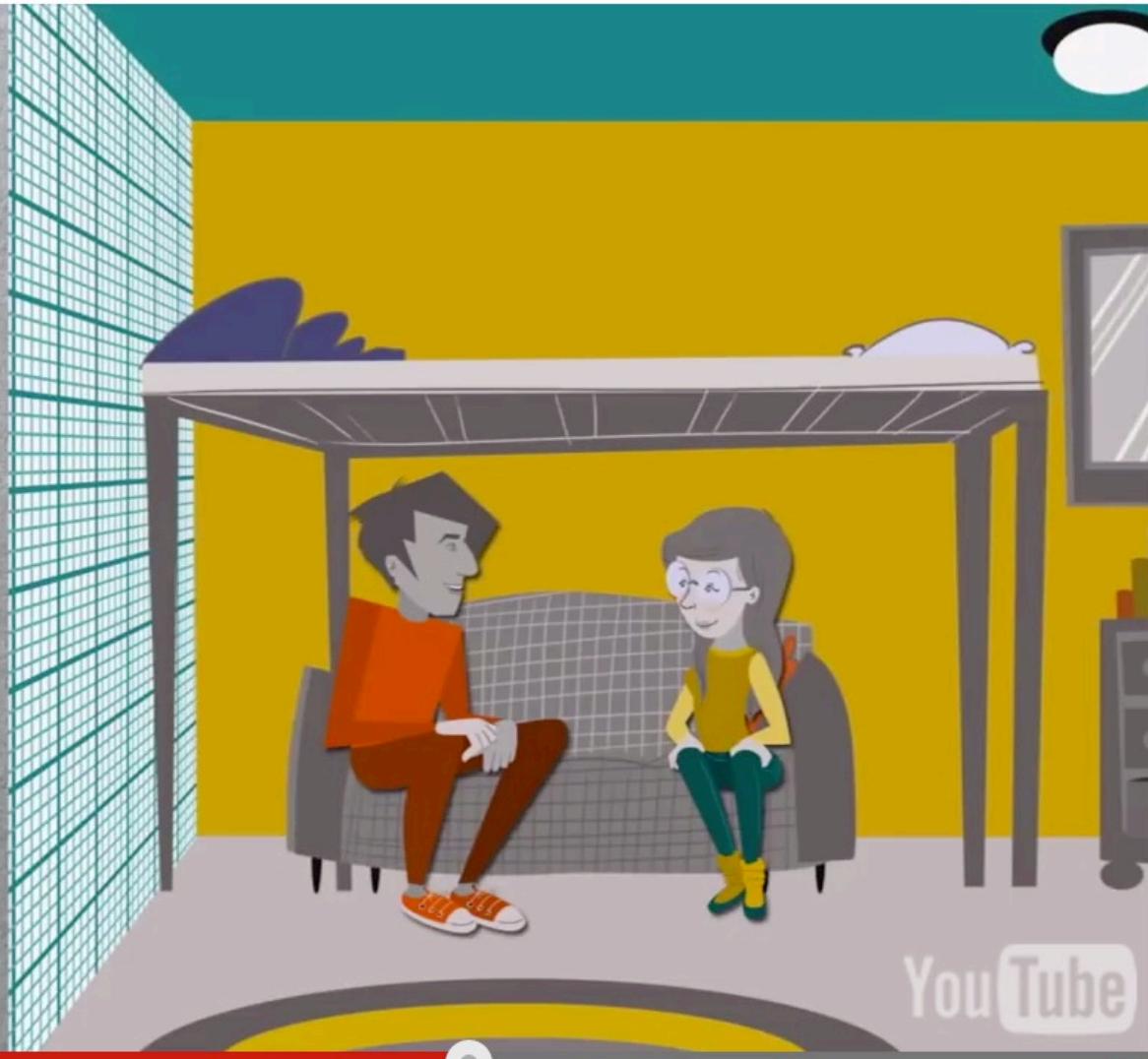


let **N** = 0

For each pair of people in room

Set **N** = **N** + 2

2



Initial algorithm for 2 at-a-time seemed ok.

**BUGGY!**



let **N** = 0



For each pair of people in room



Set **N** = **N** + 2



3:49 / 4:57



**But, it broke when we tried counting an odd number.**

# Condition

Do different things based on a condition

TEST COMPLETE



let **N** = 0

For each pair of people in room

Set **N** = **N** + 2

If 1 person remains then

Set **N** = **N** + 1

3



If one person remains...

# Code

# Source Code

Pseudocode: An algorithm expressed in  
your own words

Source code: An Algorithm expressed in a  
programming language.

<http://learn.code.org/hoc/8>

C O  
D E

Puzzle 4 20

Blocks

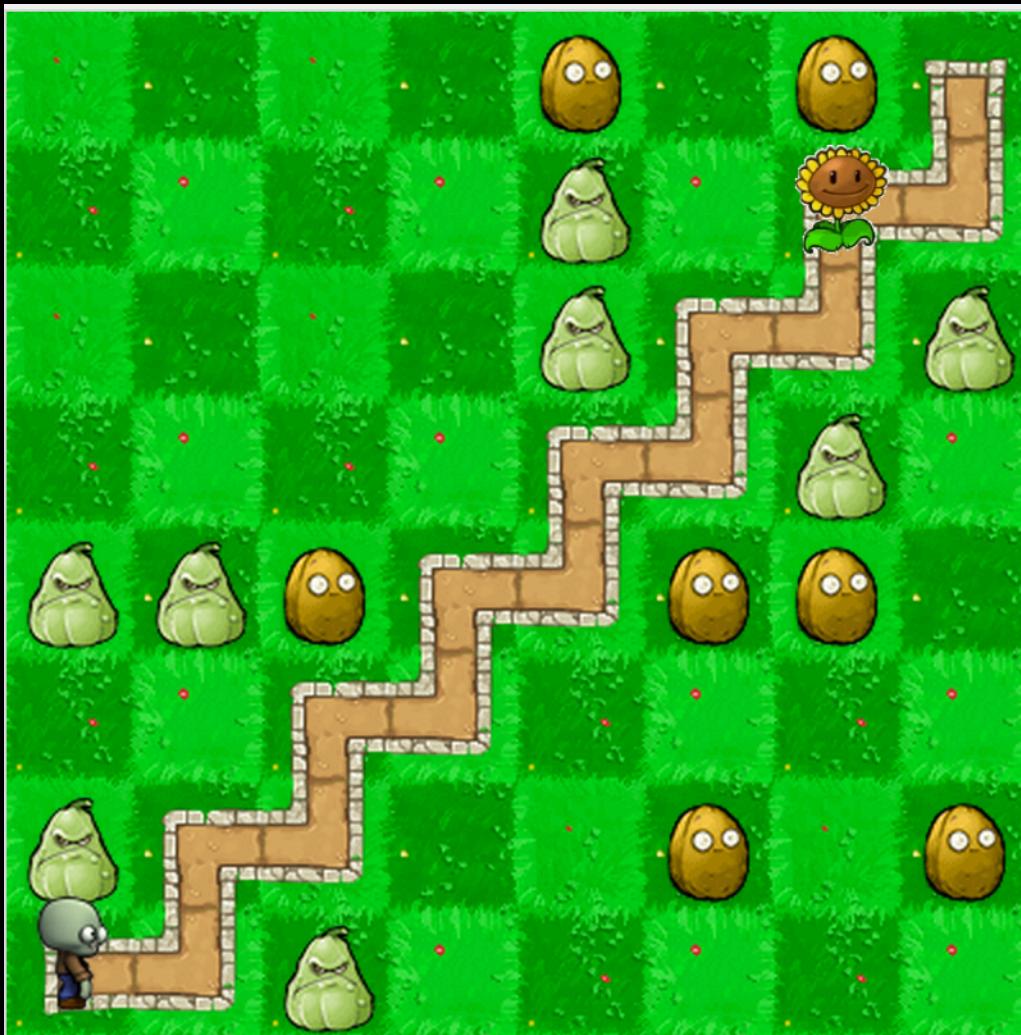
- move forward
- turn left ⌂ ▼
- turn right ⌂ ▼

Run

Guide me to the green evilness! (Watch out for TNT)

Need help? See these videos and hints

<http://studio.code.org/hoc/12>



▶ Run

Blocks

move forward

turn left ⌂ ▼

turn right ⌃ ▼

repeat until   
do

# Source Code in C++

Sunday tutorial was about  
Meet the Staff  
Xcode / Visual Studio

**If you missed it:**  
The video should be online

# Getting Started With an IDE

- Course Website: [eecs183.org](http://eecs183.org)
- Look in: Assignments > Project 1>  
Creating a Project
  - Getting Started with Visual Studio (Windows)
  - Getting Started with Xcode (Mac)

# Code Editor

The screenshot shows the Xcode IDE interface. The title bar reads "Hello World W15 | Build Hello World W15: Succeeded | 1/7/15 at 10:26 AM". The toolbar has standard Mac OS X icons. The navigation bar shows the file path: "Hello World W15 > Hello World W15 > main.cpp > No Selection". The left sidebar displays the project structure:

- Hello World W15 (target)
- Hello World W15 (group)
- main.cpp (selected)
- Products

The main editor area contains the following C++ code:

```
// main.cpp
// Hello World W15
//
// Created by Jeremy Gibson on 1/7/15.

#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!\n";
    return 0;
}
```

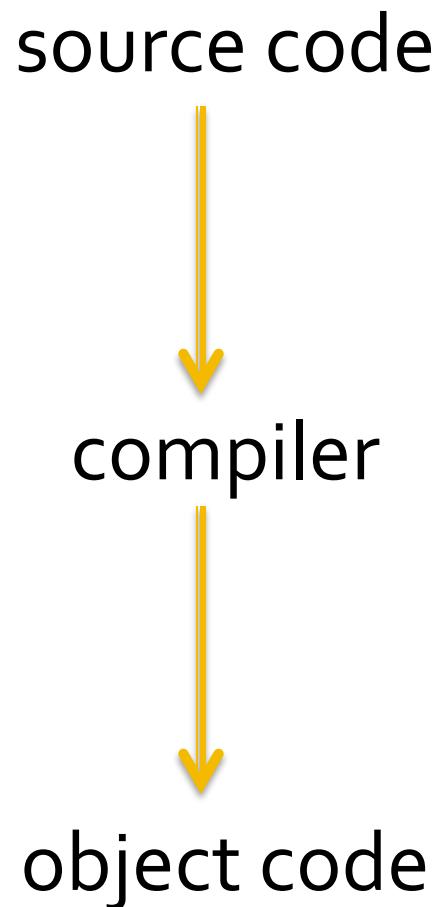
# Compiler

source code



compiler

# Compiler



10000011 00000001 00010001 00000000 00111101 11111100 01110100 00111101  
00000000 01000000 00000000 00000000 00000000 00000000 00000000 00000000  
10010000 00000000 00000000 00000000 01010000 00000000 00000111 00110000  
00001011 00000001 00001011 00000011 00001010 00000000 00000000 00000000  
00000000 00100000 00000000 00000000 00000000 00000000 00000000 00000000  
00000000 00100000 00000000 00000000 00000000 00000000 00000000 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
01110000 00010000 00000000 00100000 00000001 00000000 00000000 00000000  
00000000 00000000 00000000 00100000 00000001 00000000 00000000 00000000  
00000000 00000000 00000000 01000000 00000001 00000000 00000000 00000000  
00000000 00100000 00000000 01000000 00000001 00000000 00000000 00000000  
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111  
10010000 10000000 00000000 01000000 00000001 00000000 00000000 00000000  
00101110 01100100 01111001 01101110 01100001 01101101 01101001 01100011  
10100000 00000001 00000000 00000000 00000000 00000000 00000000 00000000  
10110000 00000100 00000000 00000000 00000000 00000000 00000000 00000000

# Source Code in C++

Now, let's look at that C++ source code

# a "Hello World" program in C++

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

int main(void) {
    cout << "Hello World!";
    return 0;
}
```

# Zyante readings - Questions

# Q1:

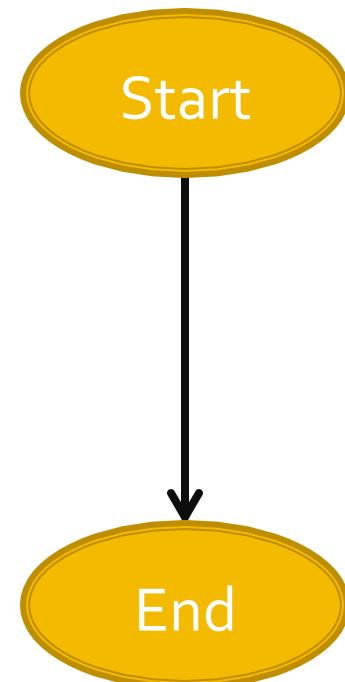
```
1 #include <iostream>
2 using namespace std;
3
4 int main(void) {
5     cout << "Hello World!";
6     return 0;
7 }
```

Where does the program begin?

- A) line 1
- B) line 2
- C) line 4
- D) line 5
- E) line 6

# First Program

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



.. and go back to caller  
(usually the OS)

# First Program

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

where it all starts  
"main" function



# First Program

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

values passed to  
the program  
(nothing in this  
case)

# First Program

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

Go back to where the function  
was called from  
(OS in case of 'main')



# First Program

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

## Return Value

(When 0 is returned to the OS from main,  
it means “all is good”)

# First Program

## Return Type

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

# First Program

## Return Type

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

these MUST match in types

## Return Value

# First Program

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

Denotes the end of a  
simple statement

# First Program

```
int main (void)
```

```
{
```

```
    return 0;
```

```
}
```

Braces denote the beginning and the end of code blocks. In this case, they define where the function starts and ends.

# "Hello World" in C++

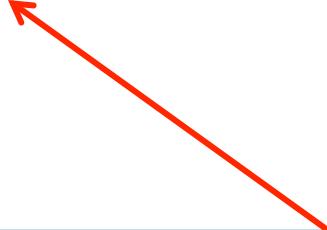
```
#include <iostream>
using namespace std;

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

# "Hello World" in C++

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```



A *Statement*. Prints "Hello World!" to the standard output stream

# "Hello World" in C++

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

object to send the output  
Basically, says to print to standard  
output stream

# "Hello World" in C++

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

insertion operator for printing  
used to put things into output stream

# "Hello World" in C++

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

The "Hello World!" string literal

# Let's add output

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Denotes end of line



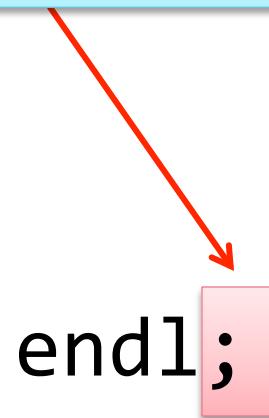
# Let's add output

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

C++

this is a STATEMENT

```
int main (void)  
{  
    cout << "Hello World!" << endl;  
    return 0;  
}
```



# Let's add output

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

I/O operations require including the I/O standard file from the C++ standard library. Enables cout

# Let's add output

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

A directive that tells the preprocessor to include code of iostream library before compilation

# Let's add output

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

All files in the C++ standard library declare all of its entities within the std namespace.

## Q2:

Which statement prints: Welcome!

- A) cout << Welcome! ;
- B) cout >> "Welcome! " ;
- C) cout << "Welcome! " ;

# Q3:

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

int main() {
    int wage = 20;
    cout << "Salary is ";
    cout << wage * 40 * 50;
    cout << endl;

    return 0;
}
```

What is wage?

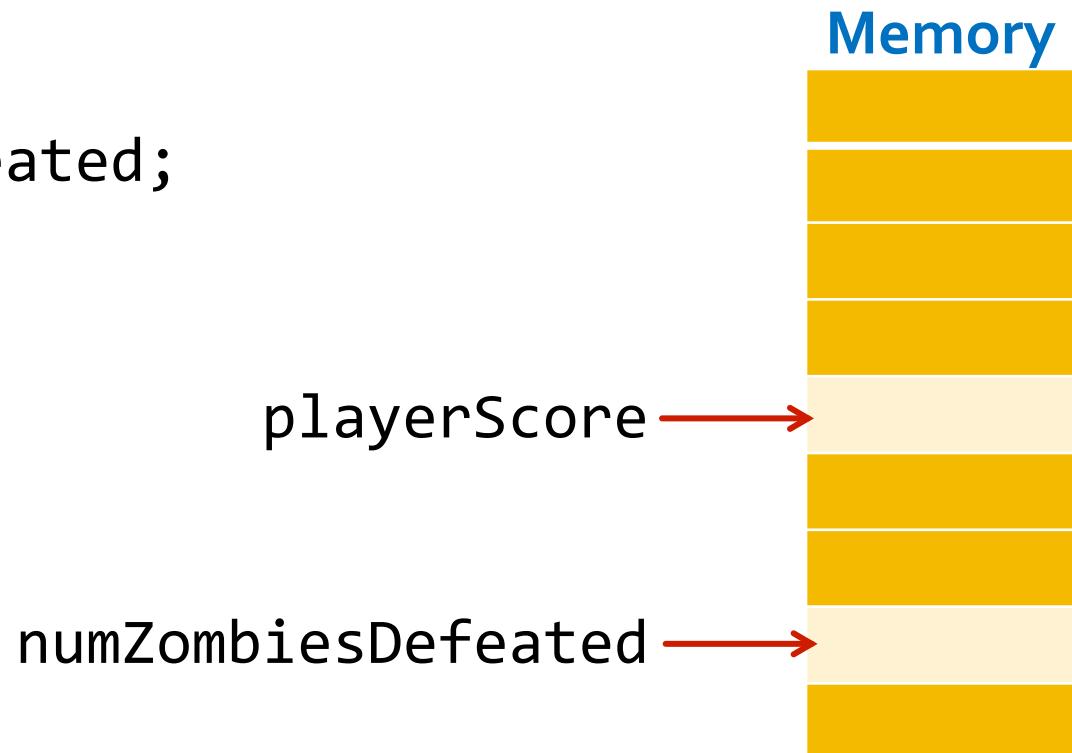
- A) is a variable
- B) represents a particular memory location
- C) holds the value of 20
- D) all of the above

# Variables

- A **variable** is a name for something
- Sets up a location in memory
- Stores a value into that location

# Variable Declaration

- Give it a name that describes its purpose
- Specify the type of info it will hold
- Examples:
  - `int playerScore;`
  - `int numZombiesDefeated;`



# Variable / Identifier Rules

- 1) Start with a letter or underscore ('\_')
- 2) After the first character, any number of letters, underscores, or digits
- 3) Can't be a *reserved* word  
(also known as *key words*)

- Max size 31 chars (most systems)
- C++ is case-sensitive (vName != Vname)
- System identifiers usually start with '\_'

# Reserved Words

- Words reserved by C++ language
- Have special meanings
- Can't be used as identifiers  
(e.g. can't be used as function names)

# Reserved Words

There are 5 reserved words in  
this code:

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

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

# Reserved Words

There are 5 reserved words in  
this code:

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

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

using, namespace, int, void, return

# Reserved Words

alignas	class	enum	namespace	return	try
alignof	compl	explicit	new	short	typedef
and	const	export	noexcept	signed	typeid
and_eq	constexpr	extern	not	sizeof	typename
asm	const_cast	false	not_eq	static	union
auto(1)	continue	float	nullptr	static_assert	unsigned
bitand	decltype	for	operator	static_cast	using
bitor	default	friend	or	struct	virtual
bool	delete	goto	or_eq	switch	void
break	do	if	private	template	volatile
case	double	inline	protected	this	wchar_t
catch	dynamic_cast	int	public	thread_local	while
char	else	long	register	throw	xor
char16_t		mutable	reinterpret_cast	true	xor_eq
char32_t					

# Reserved Words

alignas	<b>class</b>	enum	<b>namespace</b>	<b>return</b>	try
alignof	compl	explicit	new	short	typedef
and	<b>const</b>	export	noexcept	signed	typeid
and_eq	constexpr	extern	not	sizeof	typename
asm	const_cast	<b>false</b>	not_eq	static	union
auto(1)	continue	<b>float</b>	nullptr	static_assert	<b>unsigned</b>
bitand	decltype	<b>for</b>	<b>operator</b>	<b>static_cast</b>	<b>using</b>
bitor	<b>default</b>	friend	or	<b>struct</b>	virtual
<b>bool</b>	delete	goto	or_eq	<b>switch</b>	<b>void</b>
<b>break</b>	<b>do</b>	<b>if</b>	<b>private</b>	template	volatile
<b>case</b>	<b>double</b>	inline	protected	this	wchar_t
catch	dynamic_cast	<b>int</b>	<b>public</b>	thread_local	<b>while</b>
<b>char</b>	<b>else</b>	long	register	throw	xor
char16_t		mutable	reinterpret_cast	<b>true</b>	xor_eq
char32_t					

All Variables have a *Type*

# data types

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

int main() {
    int wage = 20;
    cout << "Salary is ";
    cout << wage * 40 * 50;
    cout << endl;

    return 0;
}
```

# Data Types

- bool
- int
- double
- char
- string

# Data Types - boolean

- booleans only have two values
  - true
  - false

# Data Types - int

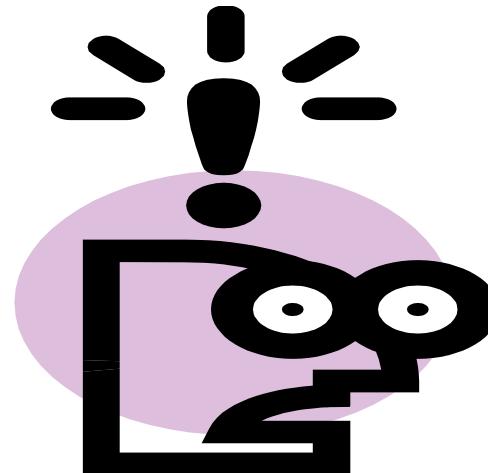
- Examples: 5, -1, 323, 1000
- NOT
  - 042      1,000

# Data Types - int

- Modern ints are 32-bit
- $2^{32} = 4,294,967,296$
- Range of a C++ int:
  - Signed: -2,147,483,468 to 2,147,483,467
  - Unsigned: 0 to 4,294,967,295

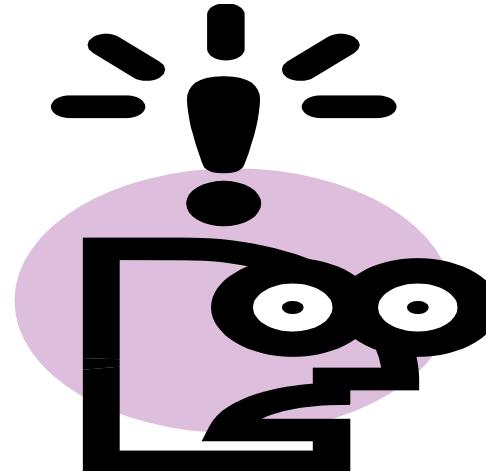
#@/&%

I can't remember those!!!



#@/&%

I can't remember those!!!



**How about:**

Remember that it's about – to + 2 billion

# Data Types - double

- Examples :

**1.0**

**3.141592**

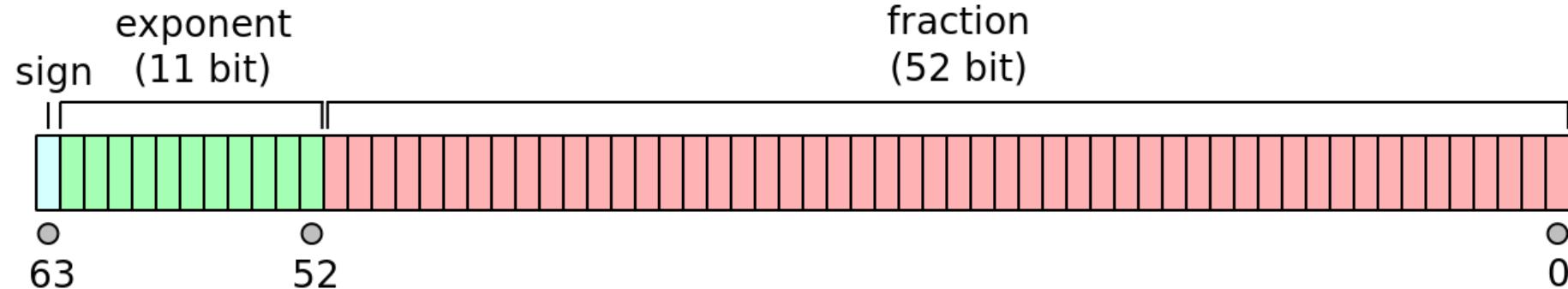
**-42.15**

**1.32e2**

**3.0E4**

**5E-3**

- Numbers with decimal values



# Data Types - double

## double

- Size: 8 bytes (64 bits)
- Range:  $\pm 2.22507e-308$  to  
 $\pm 1.79769e+308$
- "default" data type for Real numbers
  - Also called "floating point" numbers

# Data Types - char

- Examples: 'A' '\$' '5' 'e'
- A single character (i.e., letter)

# Data Types - string

- Examples: "Hello World!"  
"10 of Spades"
- A collection of multiple chars
  - Note the double quotes " to define it

# Operators

+      -      \*      /      %

```
int main() {
    int numZombies = 20;

    cout << "Number of Zombies you want to kill? ";
    int numKilled;
    cin >> numKilled;

    cout << "Number of zombies left: ";
    cout << numZombies - numKilled;

    return 0;
}
```

```
int main() {  
    Execution int numZombies = 20;  
  
    cout << "Number of Zombies you want to kill";  
    int numKilled;  
    cin >> numKilled;  
  
    cout << "Number of zombies left: ";  
    cout << numZombies - numKilled;  
  
    return 0;  
}
```

The diagram illustrates the state of variables during the execution of the program. It features two vertical columns of colored boxes. The first column, representing variable `numZombies`, has four yellow boxes stacked vertically. The second column, representing variable `numKilled`, has three yellow boxes stacked vertically. A pink arrow labeled "Execution" points from the assignment statement to the `numZombies` variable. A pink box labeled `numZombies` contains the value `20`, which is also displayed in the fourth yellow box of the `numZombies` column.

```
int main() {  
    int numZombies = 20;  
  
    Execution → cout << "Number of Zombies you want to kill"  
    int numKilled;  
    cin >> numKilled;  
  
    cout << "Number of zombies left: ";  
    cout << numZombies - numKilled;  
  
    return 0;  
}
```

The diagram illustrates the state of variables during the execution of the program. It features two vertical columns of colored boxes. The first column, labeled 'numZombies', has a yellow box at the top and a pink box below it containing the value '20'. The second column, labeled 'numKilled', has a yellow box at the top and a pink box below it. To the left of the code, a red arrow points from the text 'Execution →' towards the first column, indicating the flow of control.

Console

Number of Zombies you want to kill?<Enter>

```
int main() {  
    int numZombies = 20;  
  
    cout << "Number of Zombies you want to kill";  
    int numKilled;  
    cin >> numKilled;  
  
    cout << "Number of zombies left: ";  
    cout << numZombies - numKilled;  
  
    return 0;  
}
```

Execution →

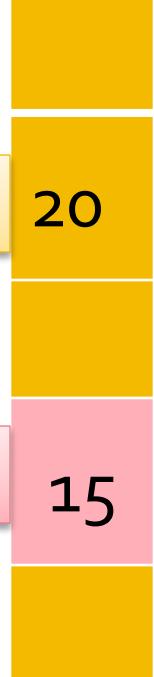
numZombies

20

numKilled

Console  
Number of Zombies you want to kill?<Enter>

```
int main() {  
    int numZombies = 20;  
  
    cout << "Number of Zombies you want to kill";  
    int numKilled;  
    cin >> numKilled;  
  
    cout << "Number of zombies left: ";  
    cout << numZombies - numKilled;  
  
    return 0;  
}
```

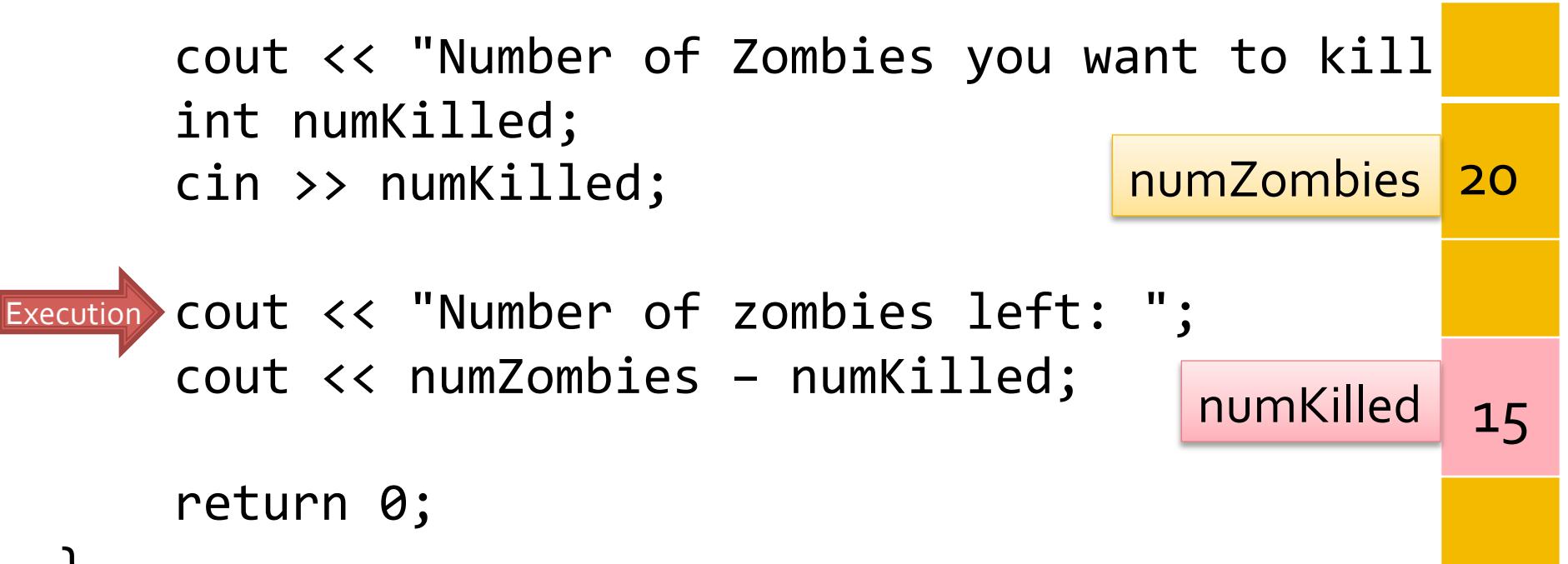


The diagram illustrates the state of variables during the execution of the program. It shows three boxes: a yellow box for `numZombies` containing the value 20, a pink box for `numKilled` containing the value 15, and a yellow box for the result of the subtraction containing the value 5.

Console

Number of Zombies you want to kill?<E15nter>

```
int main() {  
    int numZombies = 20;  
  
    cout << "Number of Zombies you want to kill";  
    int numKilled;  
    cin >> numKilled;  
  
    cout << "Number of zombies left: ";  
    cout << numZombies - numKilled;  
  
    return 0;  
}
```



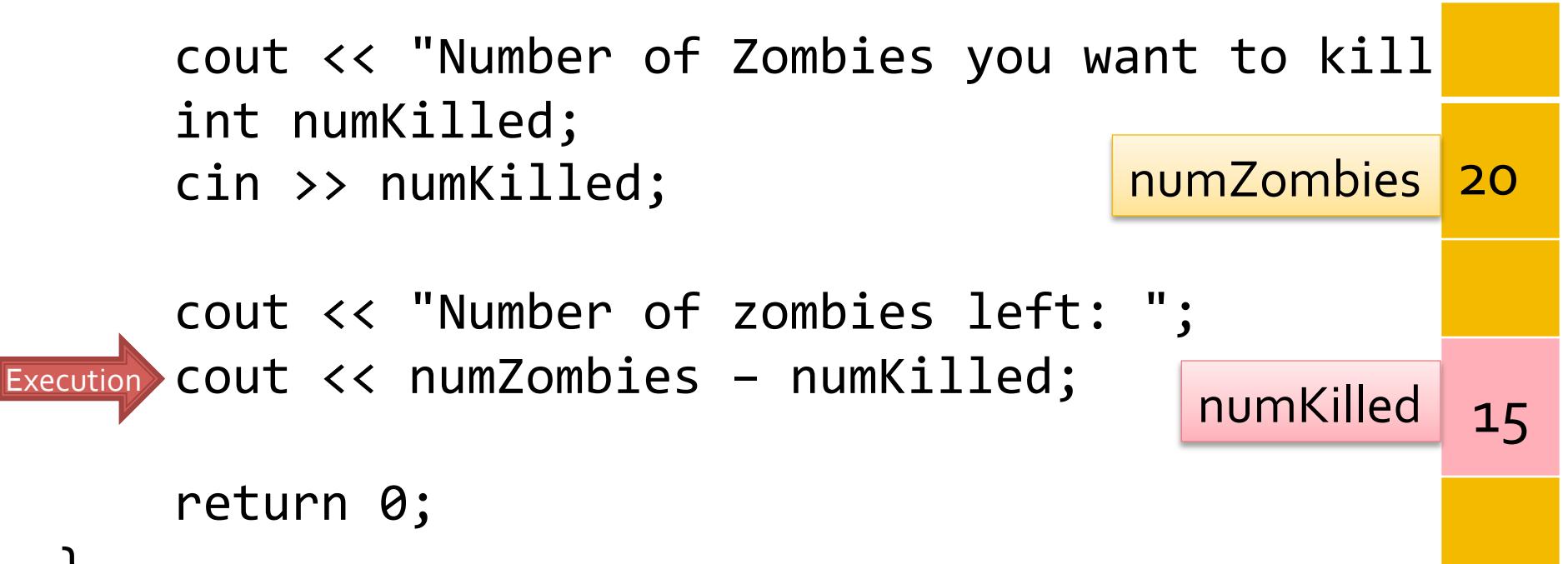
The diagram illustrates the state of variables during the execution of the program. On the right, there are two vertical columns of colored boxes representing memory. The top column is yellow and contains the variable `numZombies` with the value `20`. The bottom column is pink and contains the variable `numKilled` with the value `15`. A red arrow points from the text "Execution" to the `cout` statement `cout << "Number of zombies left: ";`, indicating the flow of control.

Console

Number of Zombies you want to kill?<E15nte

Number of zombies left:

```
int main() {  
    int numZombies = 20;  
  
    cout << "Number of Zombies you want to kill";  
    int numKilled;  
    cin >> numKilled;  
  
    cout << "Number of zombies left: ";  
    cout << numZombies - numKilled;  
  
    return 0;  
}
```



Execution →

Console

Number of Zombies you want to kill?<E15ntr

Number of zombies left: 5

# Operators

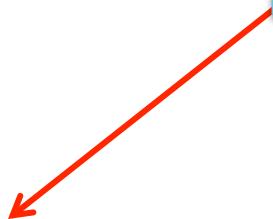
- In order to compute things, computers need operators
- Some of the most common are mathematical
  - + – Add
  - - – Subtract
  - \* – Multiply
  - / – Divide
  - % – Modulo

# The Addition Operator

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

int main(void)
{
    cout << ( 1 + 1 ) << endl;
    return 0;
}
```

We're just replacing this statement.



# The Addition Operator

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

int main(void)
```

```
{
```

```
    cout << ( 1 + 1 ) << endl;
```

```
    return 0;
```

```
}
```

Parentheses force order of operations

# The Subtraction Operator

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

int main(void)
{
    cout << ( 5 - 2 ) << endl;
    return 0;
}
```

# The Multiplication Operator

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

int main(void)
{
    cout << ( 3 * 4 ) << endl;
    return 0;
}
```

# The Division Operator

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

int main(void)
{
    cout << ( 10 / 3 ) << endl;
    return 0;
}
```

Why was the output 3?

# The Division Operator

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

int main(void)
{
    cout << ( 10 / 3 ) << endl;
    return 0;
}
```

10 and 3 are both integers

# int Division Always Truncates

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

int main(void)
{
    cout << ( 19 / 10 ) << endl;
    return 0;
}
```

The output is 1.

# The Modulo Operator

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

int main(void)
{
    cout << ( 19 % 10 ) << endl;
    return 0;
}
```

**Modulo (%) gives you the remainder.**

# The Modulo Operator

How many hours and minutes in  
142 minutes?

# The Modulo Operator

How many hours and minutes in  
142 minutes?

```
// 60 minutes per hour  
int hours = 142 / 60;
```

```
// how many minutes are left over  
int minutes = 142 % 60;
```

# More Complex Output

Combining strings with numbers

# More Complex cout

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

int main() {
    cout << "142 minutes is:" << endl
        << ( 142 / 60 ) << " Hours";
    cout << " and" << endl;
    cout << (142 % 60) << " minutes"
        << endl;
    return 0;
}
```

**Output:**

**142 minutes is:  
2 Hours and  
22 minutes**

# Gathering Input

using `cin` for input

# Using `cin` to Get User Info

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

int main() {
    cout << "Enter your name: ";
    string userString;
    cin >> userString;
    cout << "Hello, " << userString << ".";
    cout << endl;
    return 0;
}
```

# Using `cin` to Get User Info

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

int main() {
    cout << "Enter your name: ";
    string userString;
    cin >> userString;
    cout << "Hello, " << userString << ".";
    cout << endl;
    return 0;
}
```

The code demonstrates how to use `cout` to output a prompt to the user and `cin` to read input from the user. It declares a variable `userString` of type `string`, reads a user input into it, and then outputs a personalized greeting back to the user.

**Annotations:**

- Enter your name:** The text "Enter your name:" is highlighted in red.
- userString**: A callout box with a blue border and white text points to the declaration of the variable `userString`. The text inside the box reads: "Declares variable *userString*".

# Using `cin` to Get User Info

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

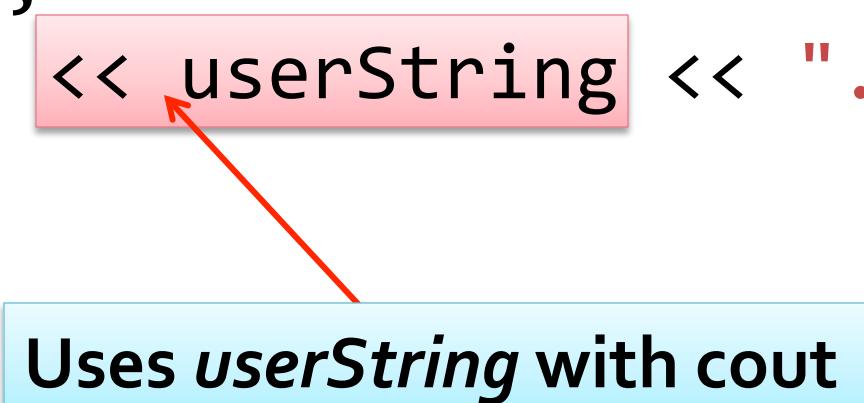
int main() {
    cout << "Enter your name: ";
    string userString;
    cin >> userString;
    cout << "Hello, " << userString << ".";
    cout << endl;
    return 0;
}
```

Uses `cin` to define *userString*

# Using `cin` to Get User Info

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

int main() {
    cout << "Enter your name: ";
    string userString;
    cin >> userString;
    cout << "Hello, " << userString << ".";
    cout << endl;
    return 0;
}
```



Uses *userString* with `cout`

# Next Time on EECS 183...

More details on: Variables and Data Types  
Casting from One Type to Another  
Imprecision, Compile and run-time errors  
Libraries, Testing and Debugging  
Pre-defined functions, e.g., cmath

# **Remember!**

**Bring your i>Clicker **next lecture****

**CodeLab is due **this Friday****

**Discussion Sections start **this week!****