Survey of Scientific Computing
(SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Session 09
Functional Equations,
Equilibrium Simulation

4th Annual SCI-ED Day
hosted at BNL on November 4, 2014, provides a professional development opportunity for elementary and secondary science teachers.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimpleEvents
{
    public partial class Form1 : Form
    {
        Person person = new Person();

        public Form1()
        {
            InitializeComponent();
            person.FirstName = "Christian";
            person.LastName = "Rano";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            person.HairColor = textBox1.Text;
        }
    }
}
```

Session Goals

- More practice with the **for()** and **while()** loop constructs
- Understand the C++ **class** idiom
- Review “is-a” and “has-a” class relationships
- Calculate center of mass of a **Jenga** pile
- Model the construction of a *cantilever*
- Develop **functional equations** to confirm simulation results

A Shortcut

Carl Friedrich Gauss

(1777 – 1855)

Sum the integers from 1 to 100

1
2
3
4
5
6
7
8
9
10

1	9
2	8
3	7
4	6
5	
10	

4 matched rows that each sum to 10

1 row that is = 10 / 2 = 5

1 row that is = n = 10

$$n\left(\frac{n}{2}-1\right)+\frac{n}{2}+n \qquad = \frac{n*(n+1)}{2}$$

= 55

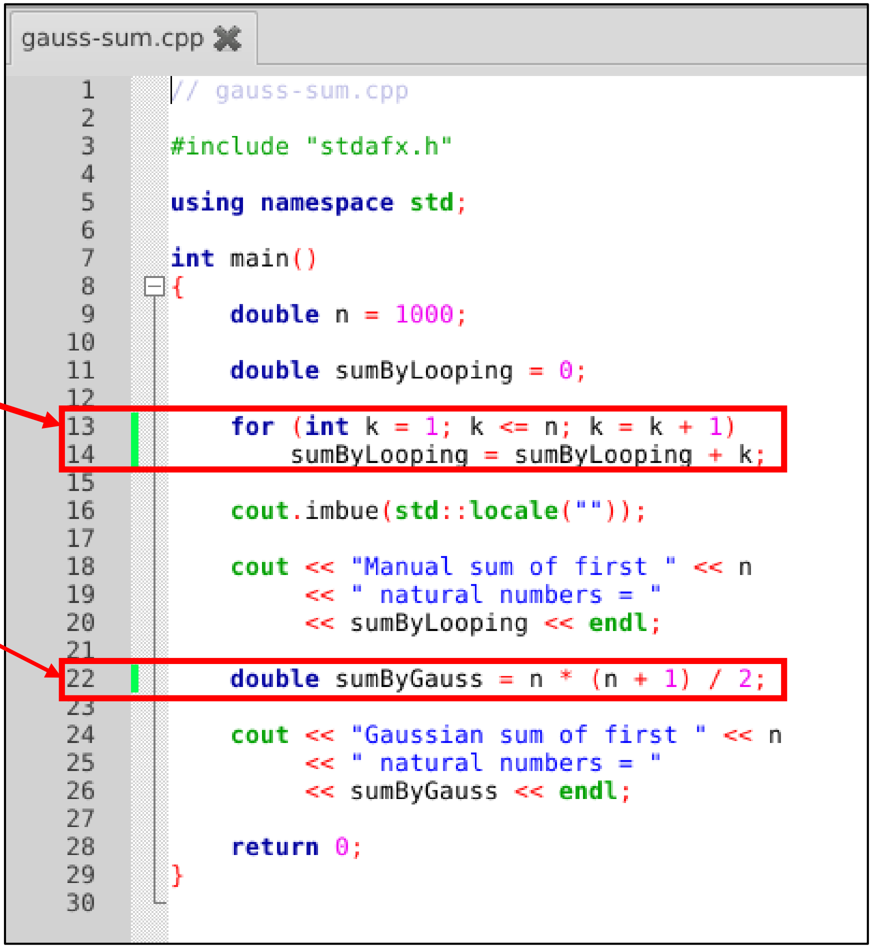
Gaussian Summation

- Create a program to sum the first **1,000** natural numbers
- Write a **for()** loop so it runs while $\mathbf{1} \le \boldsymbol{k} \le \boldsymbol{n}$, and increments k **by one** after each iteration
- Implement the **functional equation** for Gaussian summation:

$$\sum_{k=1}^{n} k = \frac{n * (n+1)}{2}$$

Open Lab 1 – Gaussian Summation

- Create a program to sum the first **1,000** natural numbers
- Write a **for()** loop so it runs while $1 \leq k \leq n$, and increments k by one after each iteration
- Implement the functional equation for Gaussian summation

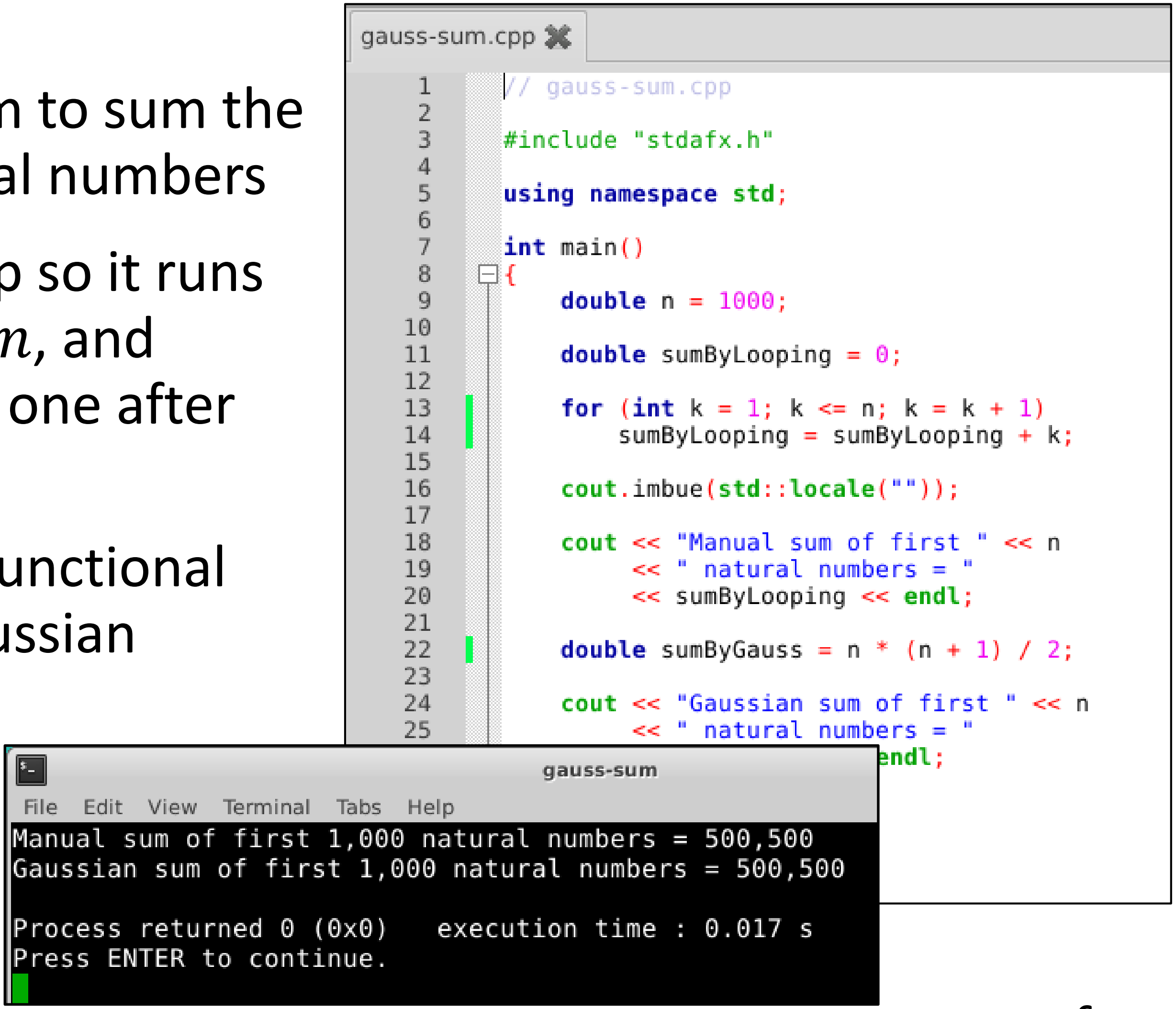


gauss-sum.cpp

```
// gauss-sum.cpp

#include "stdafx.h"

using namespace std;

int main()
{
    double n = 1000;

    double sumByLooping = 0;

    for (int k = 1; k <= n; k = k + 1)
        sumByLooping = sumByLooping + k;

    cout.imbue(std::locale(""));

    cout << "Manual sum of first " << n
         << " natural numbers = "
         << sumByLooping << endl;

    double sumByGauss = n * (n + 1) / 2;

    cout << "Gaussian sum of first " << n
         << " natural numbers = "
         << sumByGauss << endl;

    return 0;
}
```

Run Lab 1 – Gaussian Summation

- Create a program to sum the first **1,000** natural numbers
- Write a **for()** loop so it runs while $1 \le k \le n$, and increments k by one after each iteration
- Implement the functional equation for Gaussian summation



gauss-sum.cpp

```cpp
// gauss-sum.cpp

#include "stdafx.h"

using namespace std;

int main()
{
    double n = 1000;

    double sumByLooping = 0;

    for (int k = 1; k <= n; k = k + 1)
        sumByLooping = sumByLooping + k;

    cout.imbue(std::locale(""));

    cout << "Manual sum of first " << n
         << " natural numbers = "
         << sumByLooping << endl;

    double sumByGauss = n * (n + 1) / 2;

    cout << "Gaussian sum of first " << n
         << " natural numbers = "
                                   endl;
```

gauss-sum

```
Manual sum of first 1,000 natural numbers = 500,500
Gaussian sum of first 1,000 natural numbers = 500,500

Process returned 0 (0x0)   execution time : 0.017 s
Press ENTER to continue.
```

Another Shortcut

Sum of first **n** *natural* numbers:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$

Sum of squares of first **n** *natural* numbers:

n	n^2	Sum
1	1	1
2	4	5
3	9	14
4	16	30
5	25	55
6	36	91
7	49	140
8	64	204
9	81	285
10	100	385

$$P_n = \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = \boxed{\frac{2n^3 + 3n^2 + n}{6}}.$$

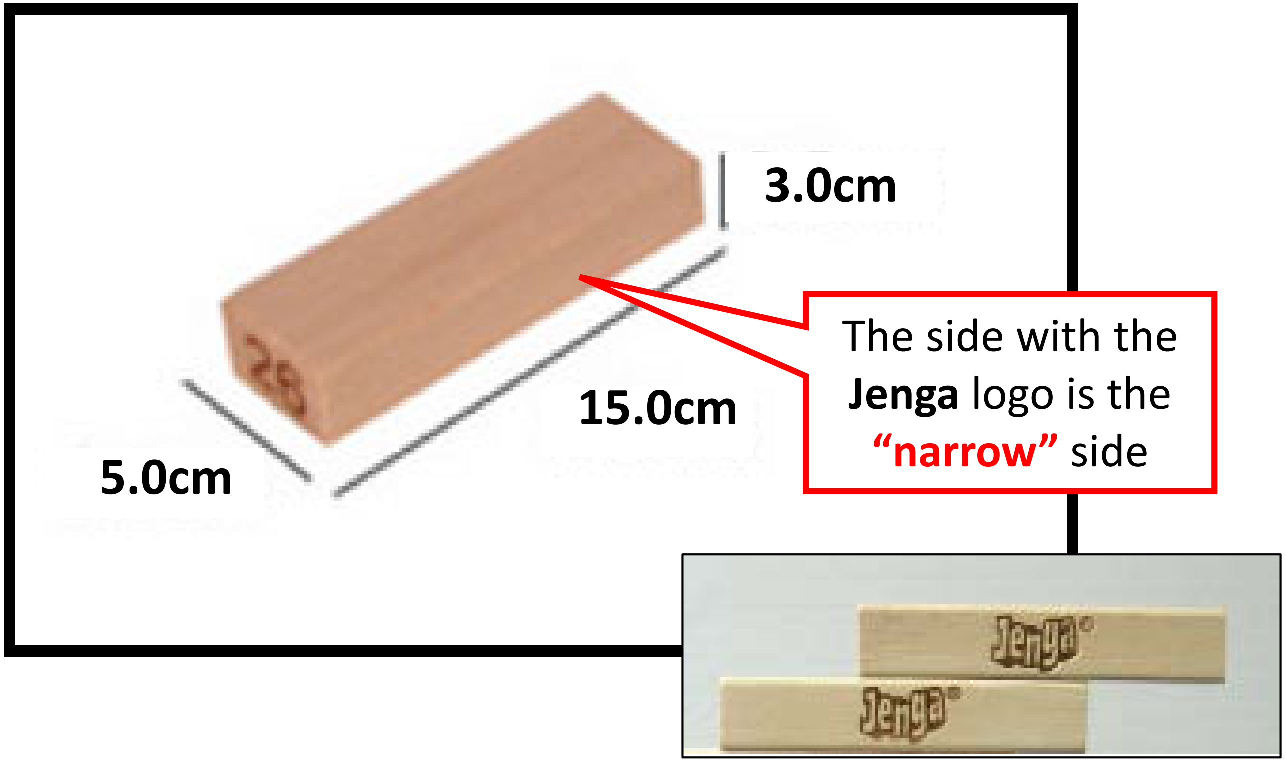
These are functional equations - we can now calculate the sums *immediately* without having to loop over every element!

Cantilever Building Design

Jenga Block Dimensions

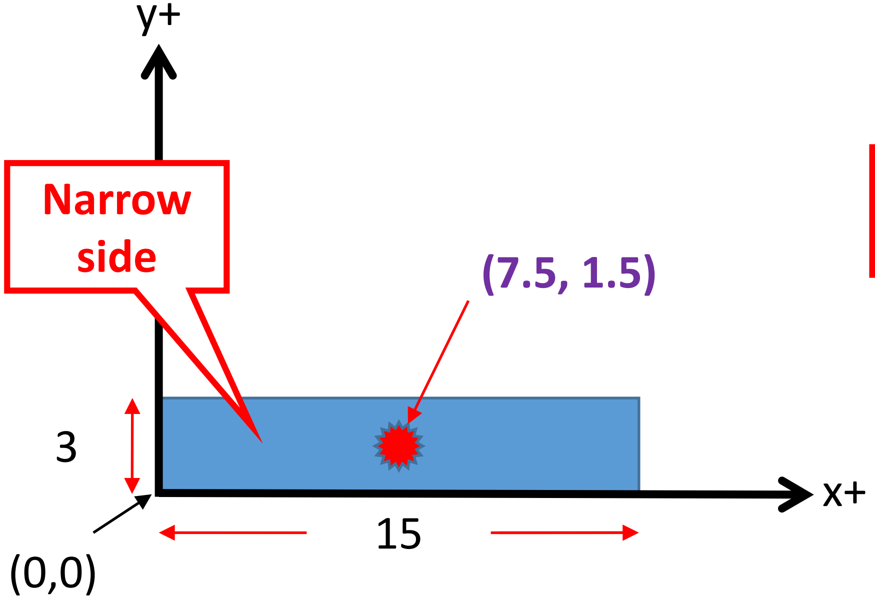
Leslie Scott created Jenga in 1983

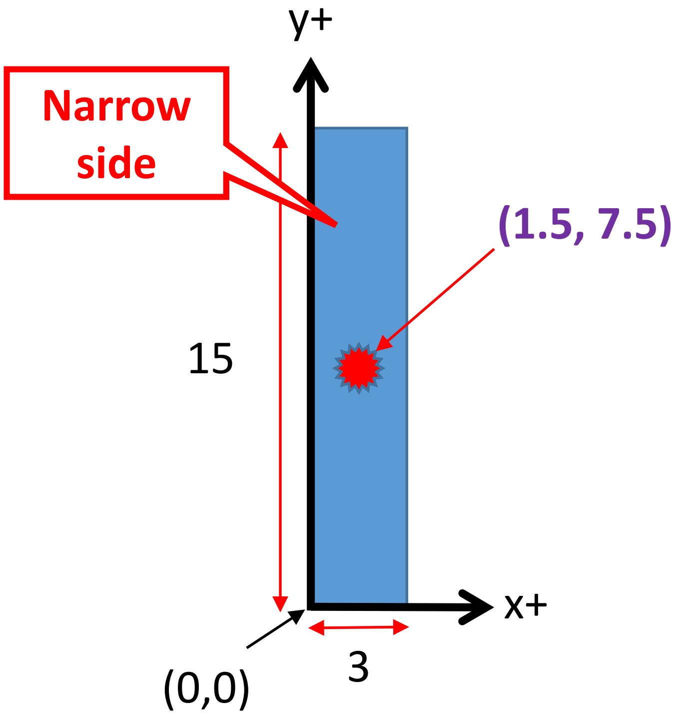
Jenga Block Dimensions



The side with the **Jenga** logo is the **“narrow”** side

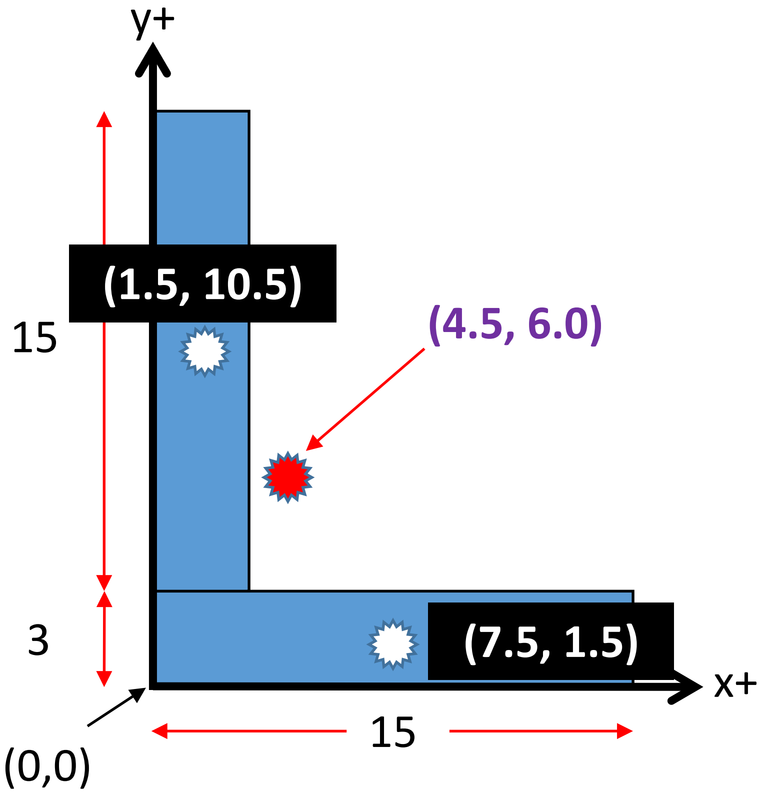
Center of Mass (COM)





Center of Mass = Average of All Corner Coordinates (in **each** **X** & **Y** dimensions)

1^{st} Ensemble *(first two blocks)*

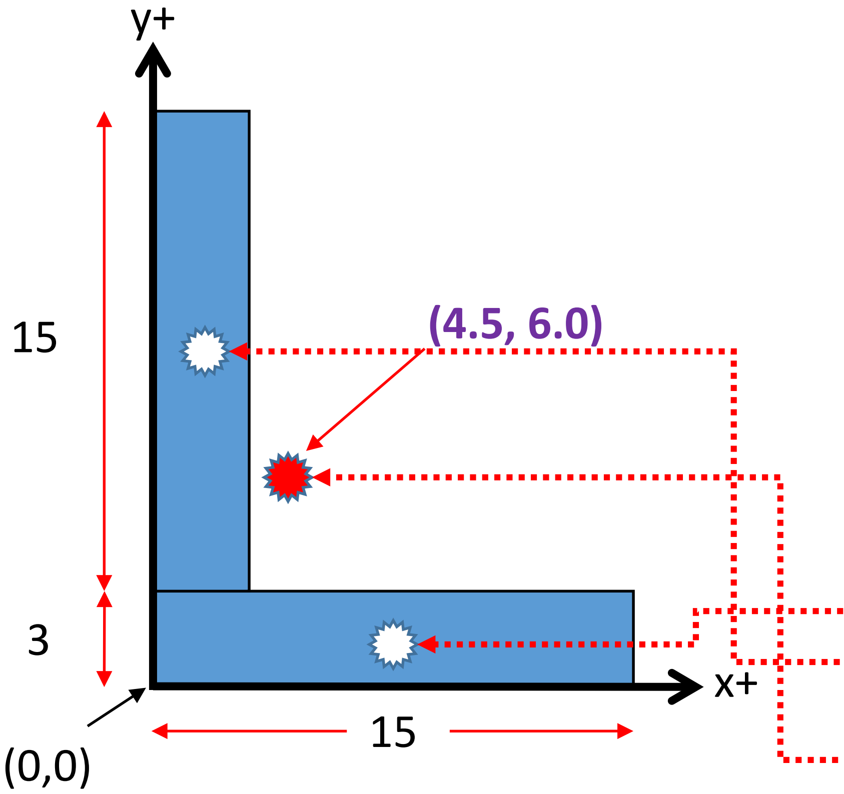


Center of Mass = Average of the centers of all blocks (in **each** **X** & **Y** dimension)

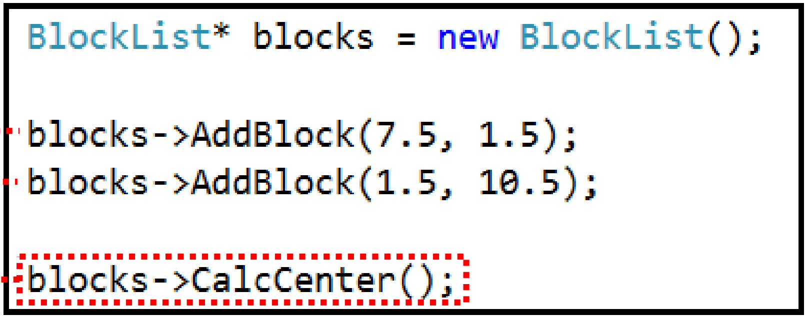
$$C_x = \frac{1.5 + 7.5}{2} = 4.5$$

$$C_y = \frac{10.5 + 1.5}{2} = 6.0$$

1st Ensemble *(first two blocks)*

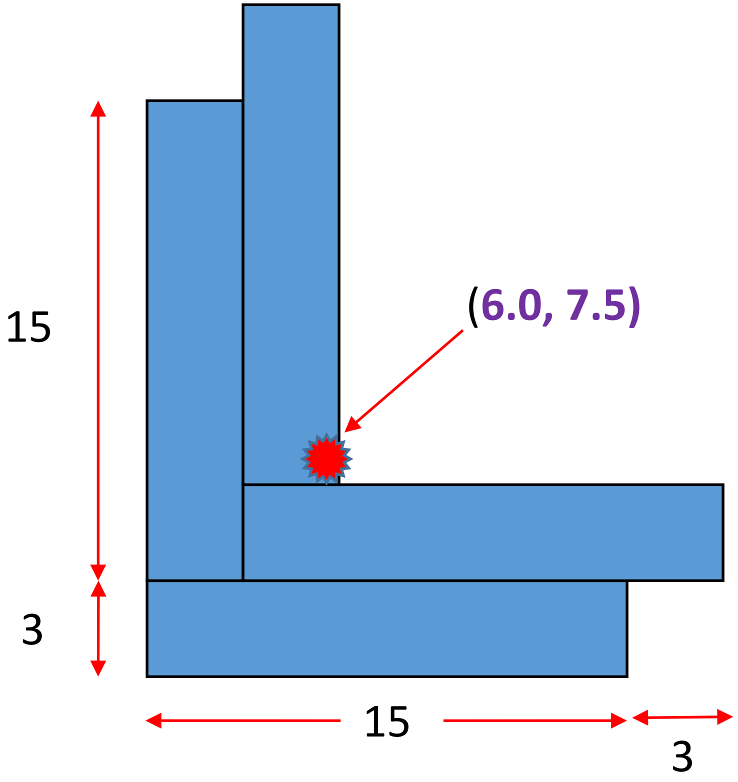


Center of Mass = Average of the centers of all blocks (in **each X** & **Y** dimension)



```
BlockList* blocks = new BlockList();

blocks->AddBlock(7.5, 1.5);
blocks->AddBlock(1.5, 10.5);

blocks->CalcCenter();
```

2nd Ensemble *(next two blocks)*

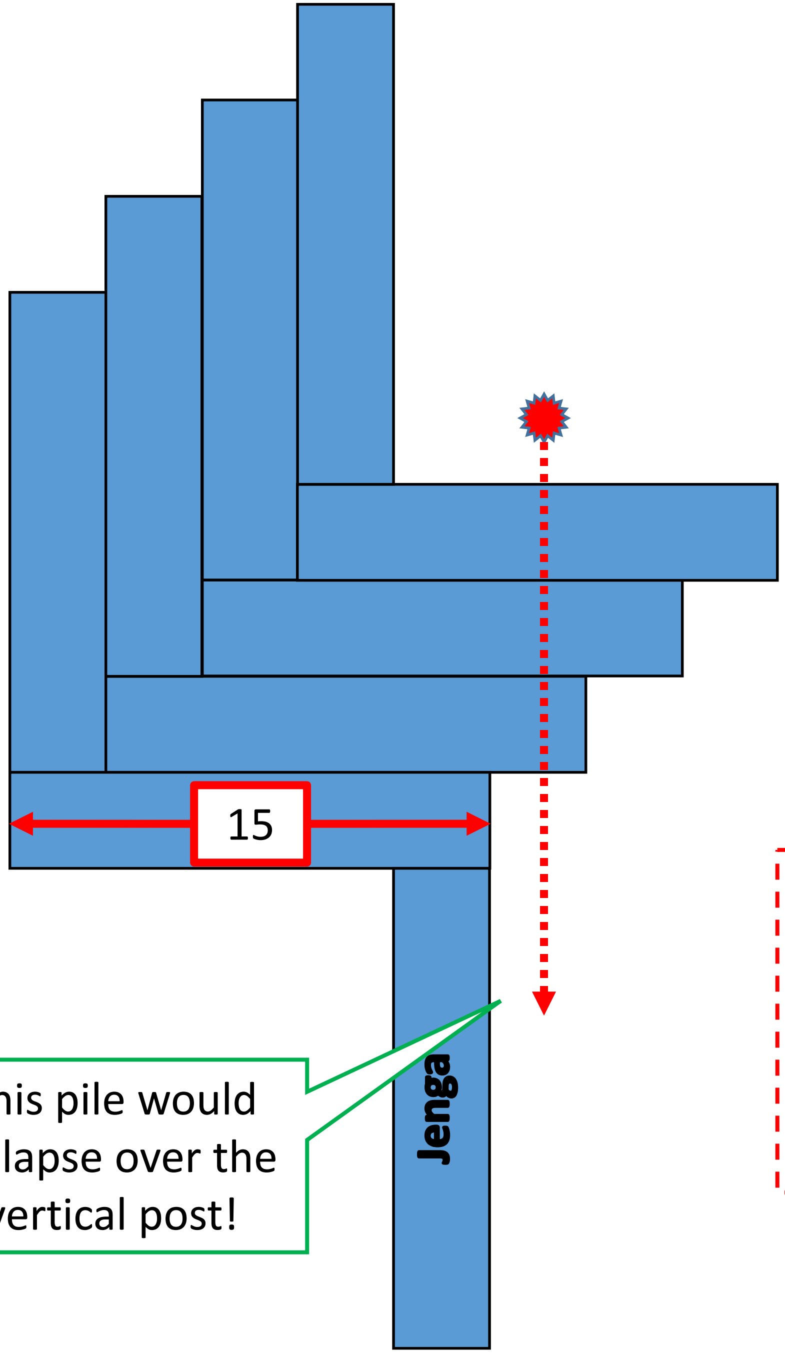


```
blocks->MoveBlocks(3, 3);
blocks->AddBlock(7.5, 1.5);
blocks->AddBlock(1.5, 10.5);
blocks->CalcCenter();
```

Before adding next ensemble, we first **move** each *existing* block by $+\boldsymbol{\Delta x}, +\boldsymbol{\Delta y}$

The effect is to move all *existing* blocks **upwards and to the right**, *before* the next iteration.

$$\Delta x = +3, \Delta y = +3$$

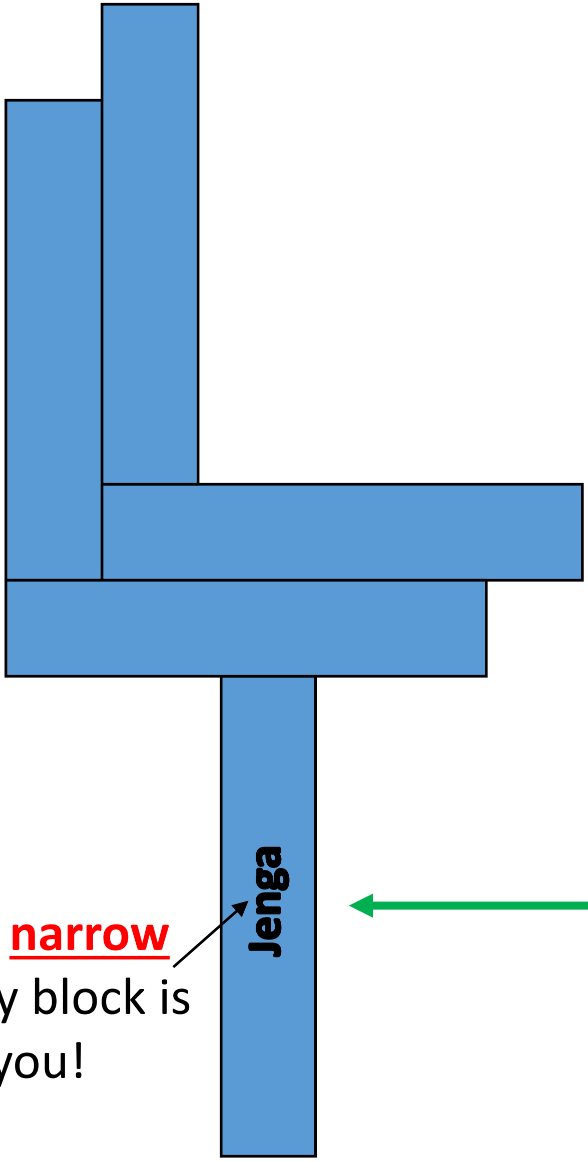


This pile would collapse over the vertical post!

How many **2-block** ensembles can be balanced on a single vertical post?

The center of mass (of the entire pile) in the **x-dimension** must remain **< 15.0** or else the bottom *horizontal* block **cannot** be supported by a vertical post!

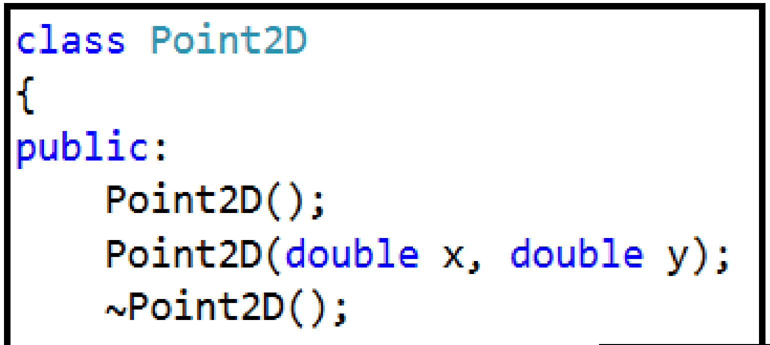
How many **2-block** ensembles can be balanced on a vertical post?



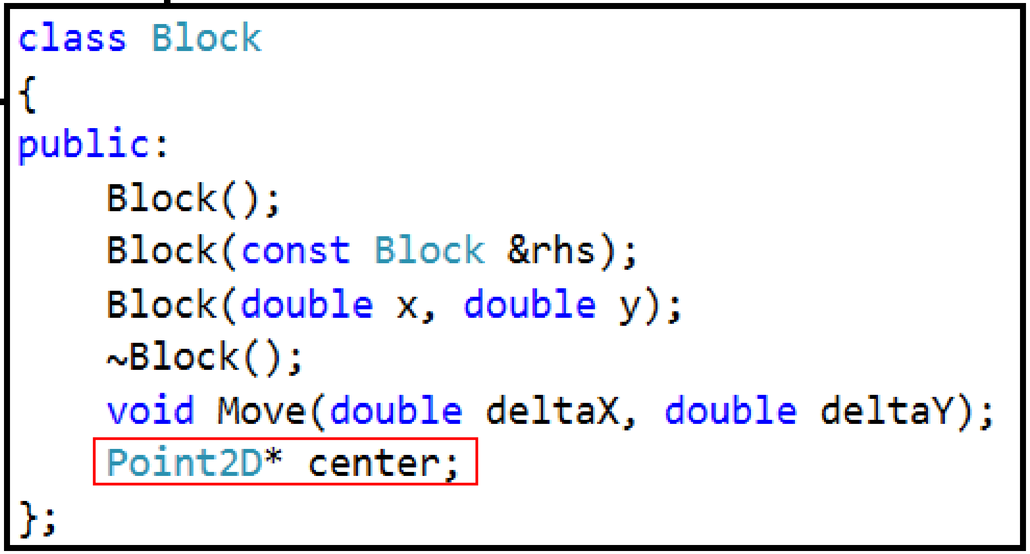
Ensure the **narrow** side of every block is facing you!

Try this now with your blocks

C++ Class Hierarchy



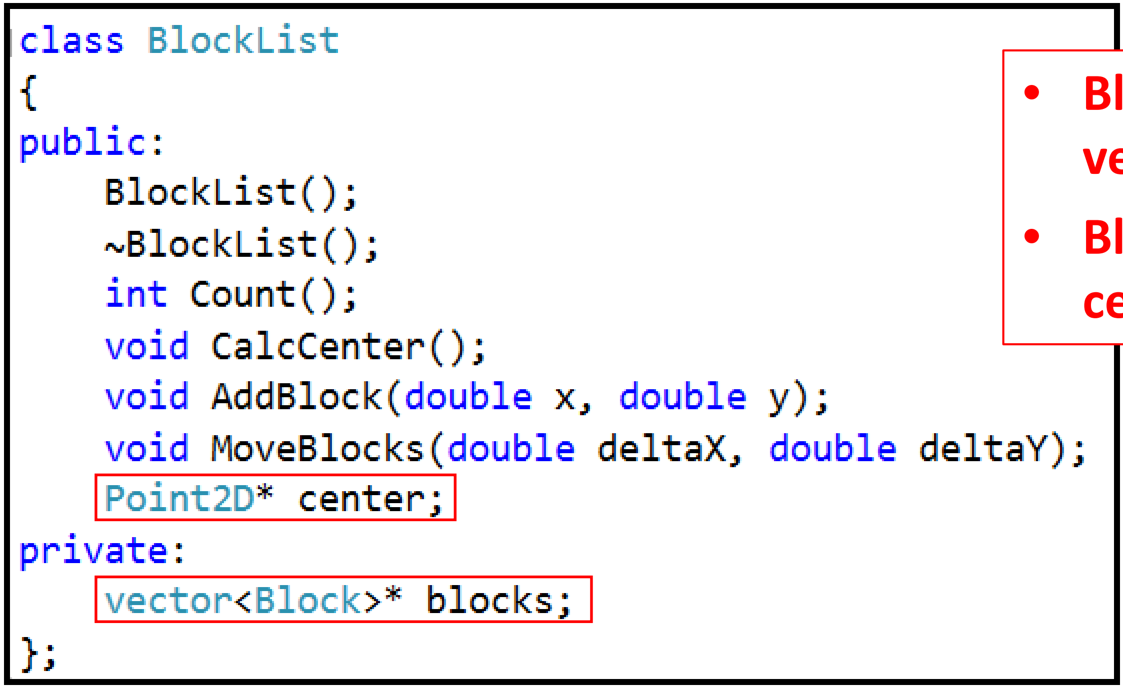
```
class Point2D
{
public:
    Point2D();
    Point2D(double x, double y);
    ~Point2D();
    double x, y;
};
```



```
class Block
{
public:
    Block();
    Block(const Block &rhs);
    Block(double x, double y);
    ~Block();
    void Move(double deltaX, double deltaY);
    Point2D* center;
};
```

- **A block “has-a” center *(of mass)***
- **A center “is-a” Point2D**

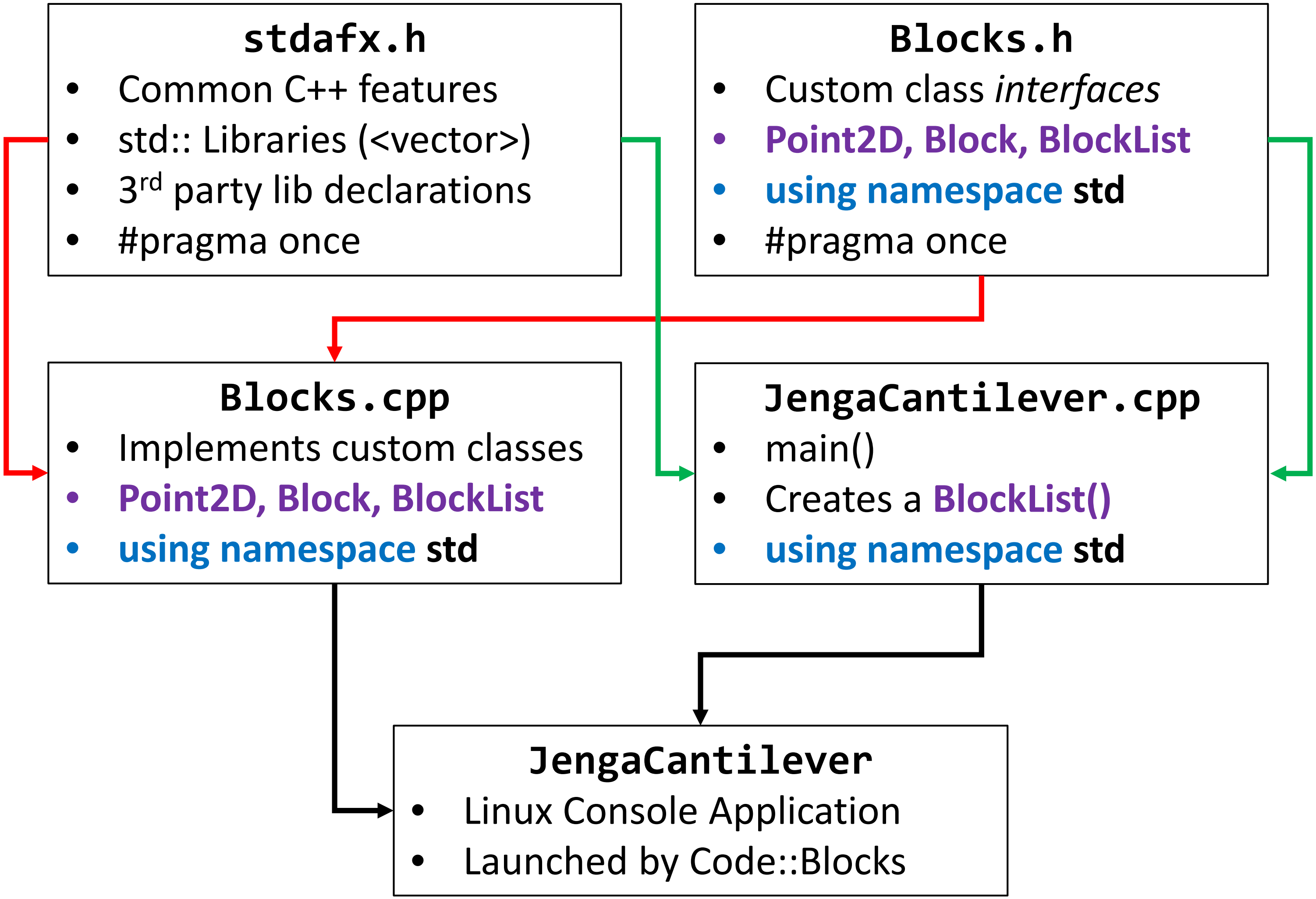
C++ Class Hierarchy



```
class BlockList
{
public:
    BlockList();
    ~BlockList();
    int Count();
    void CalcCenter();
    void AddBlock(double x, double y);
    void MoveBlocks(double deltaX, double deltaY);
    Point2D* center;
private:
    vector<Block>* blocks;
};
```

- **BlockList “has-a” vector of Blocks**
- **BlockList “has-a” center *(of mass)***

Header and Source File Dependency



`stdafx.h`
- Common C++ features
- std:: Libraries (<vector>)
- 3rd party lib declarations
- #pragma once

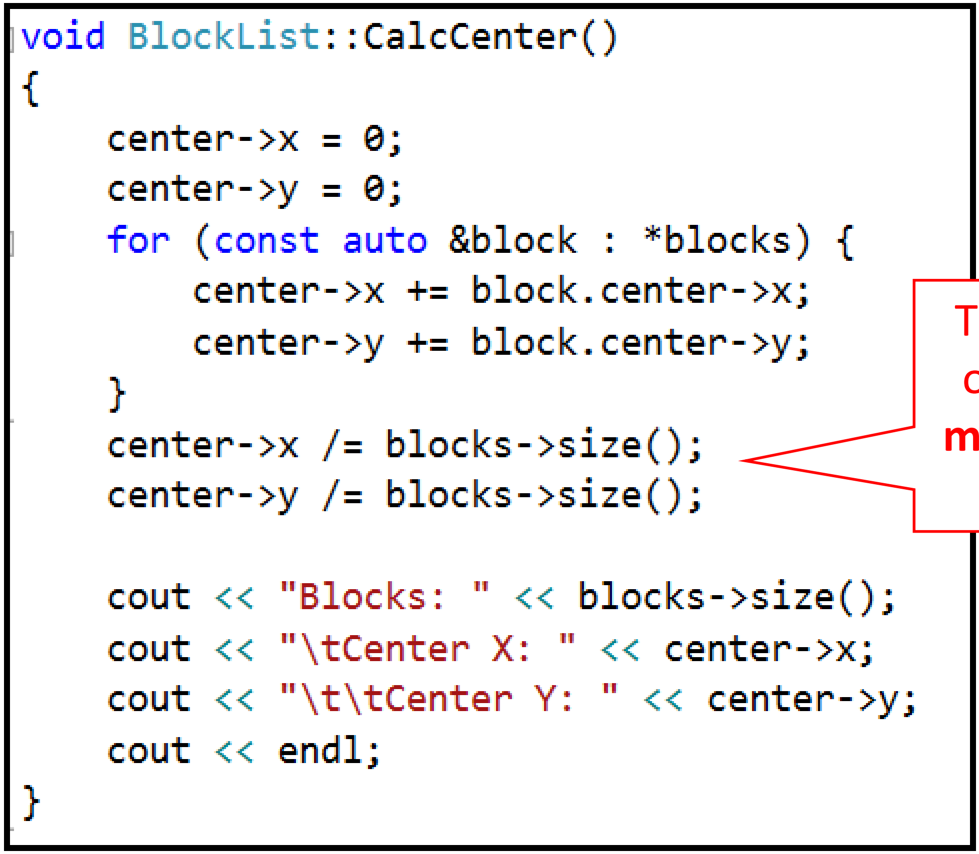
`Blocks.h`
- Custom class *interfaces*
- **Point2D, Block, BlockList**
- **using namespace std**
- #pragma once

`Blocks.cpp`
- Implements custom classes
- **Point2D, Block, BlockList**
- **using namespace std**

`JengaCantilever.cpp`
- main()
- Creates a **BlockList()**
- **using namespace std**

`JengaCantilever`
- Linux Console Application
- Launched by Code::Blocks

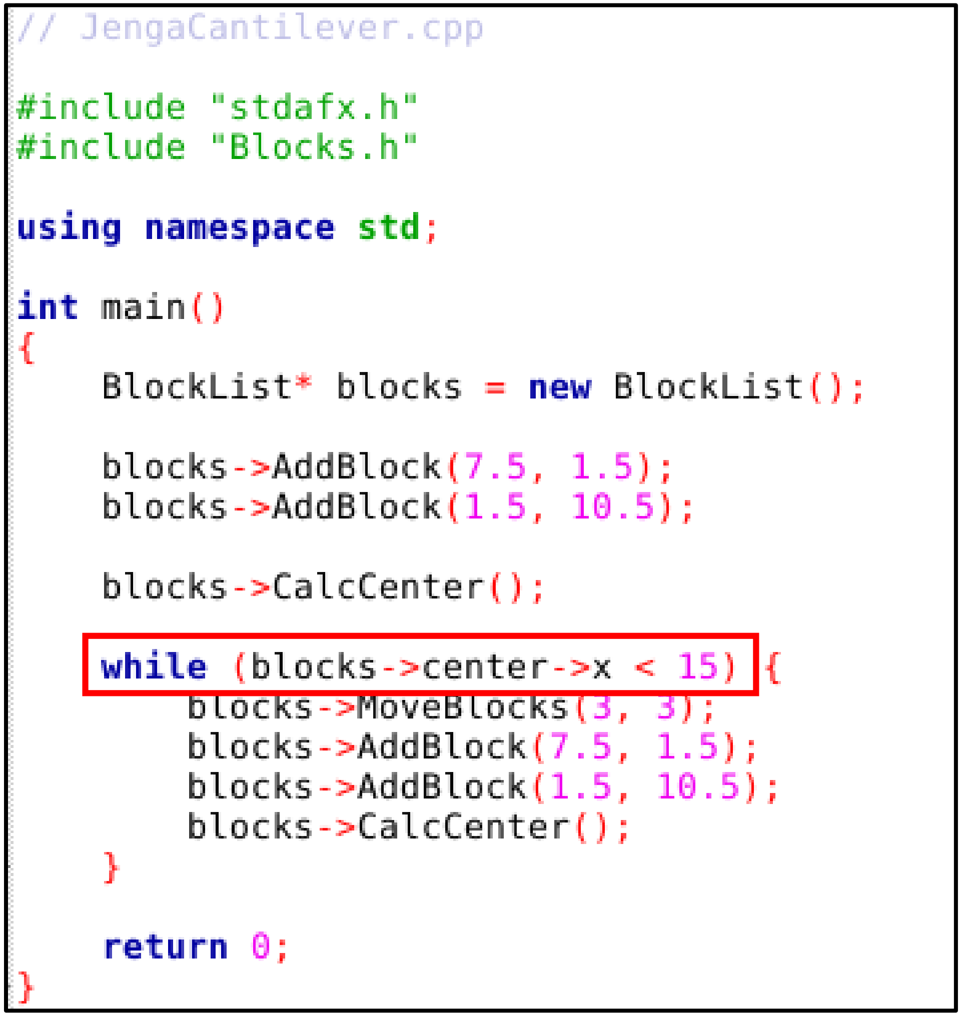
The Center of Mass is a Mean



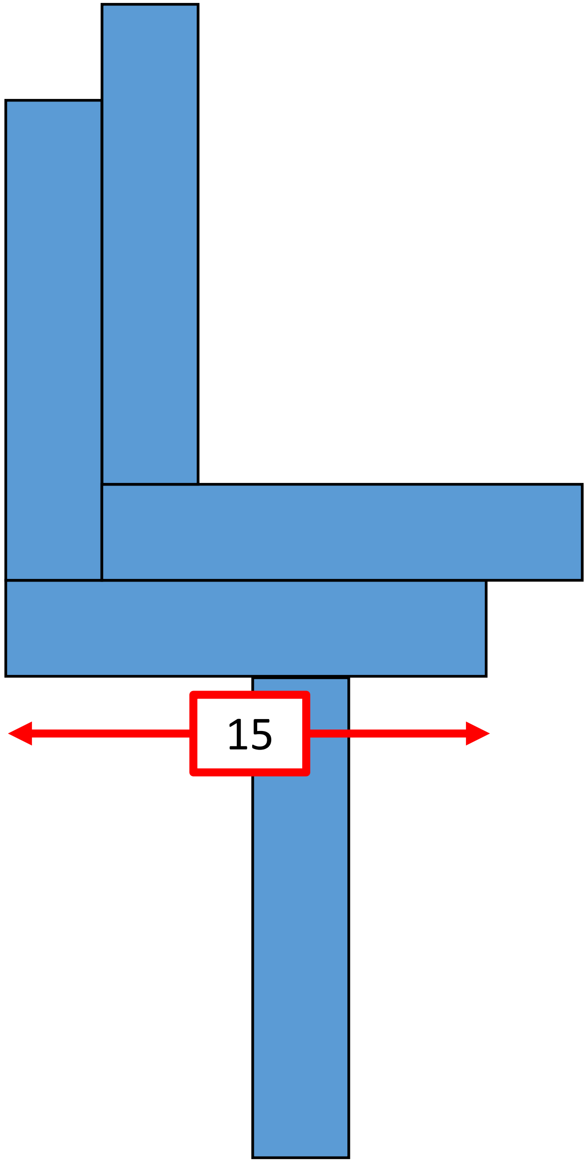
```
void BlockList::CalcCenter()
{
    center->x = 0;
    center->y = 0;
    for (const auto &block : *blocks) {
        center->x += block.center->x;
        center->y += block.center->y;
    }
    center->x /= blocks->size();
    center->y /= blocks->size();

    cout << "Blocks: " << blocks->size();
    cout << "\tCenter X: " << center->x;
    cout << "\t\tCenter Y: " << center->y;
    cout << endl;
}
```

The center of mass of a pile is just the **mean** of the centers of every block

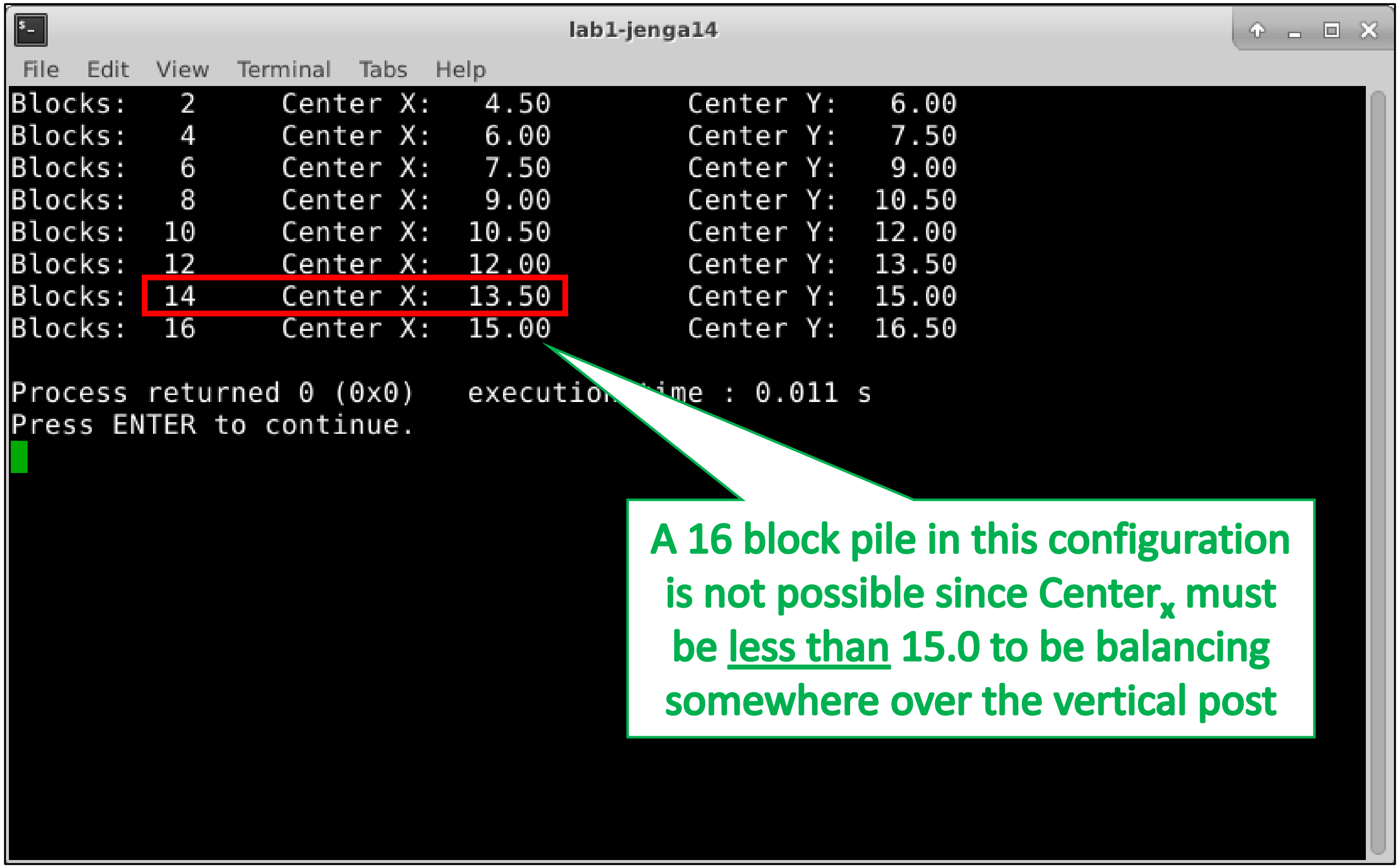
Open Lab 2 – **Jenga-14**



```
// JengaCantilever.cpp

#include "stdafx.h"
#include "Blocks.h"

using namespace std;

int main()
{
    BlockList* blocks = new BlockList();

    blocks->AddBlock(7.5, 1.5);
    blocks->AddBlock(1.5, 10.5);

    blocks->CalcCenter();

    while (blocks->center->x < 15) {
        blocks->MoveBlocks(3, 3);
        blocks->AddBlock(7.5, 1.5);
        blocks->AddBlock(1.5, 10.5);
        blocks->CalcCenter();
    }

    return 0;
}
```



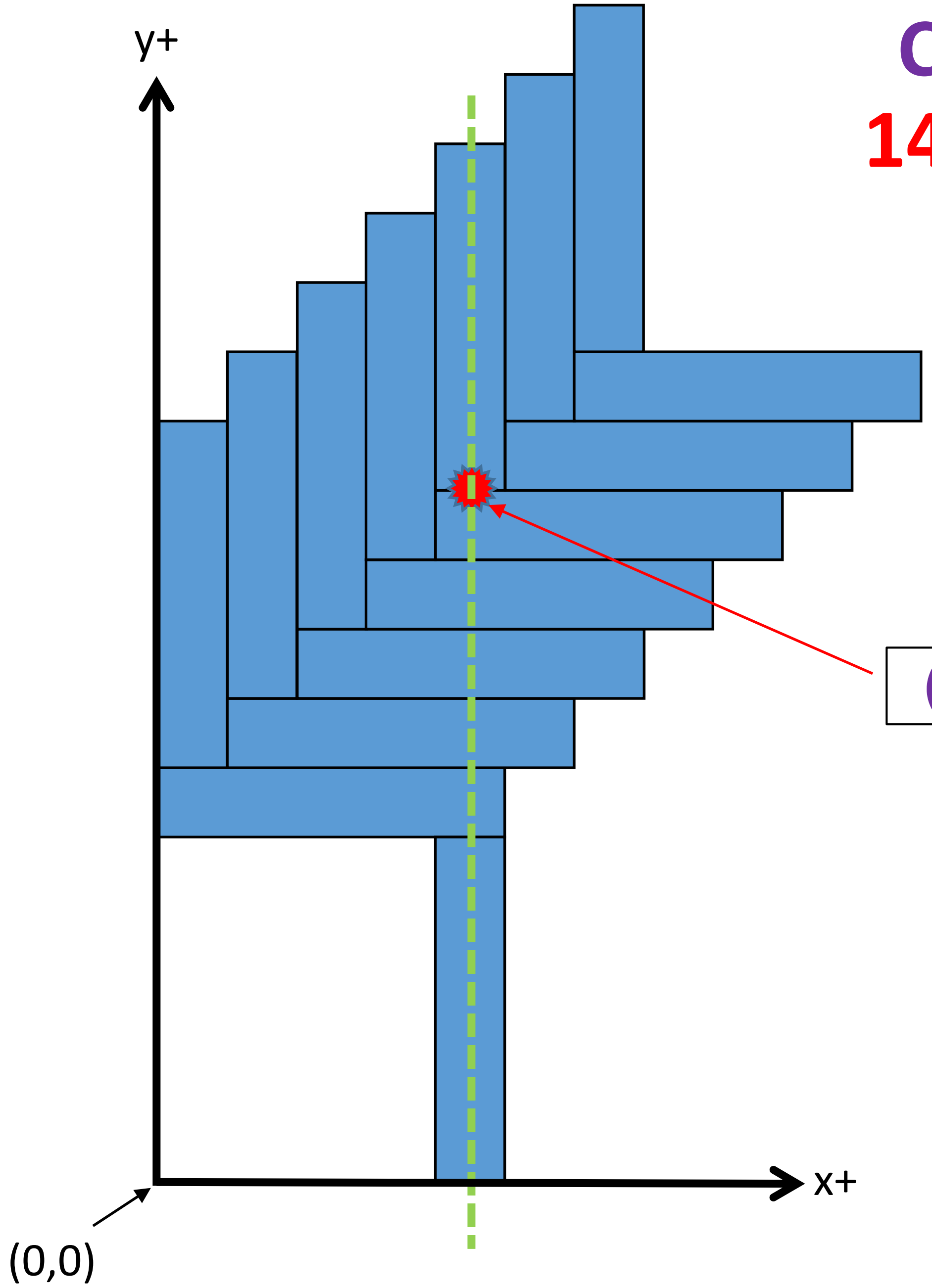
Run Lab 2
14 Block Jenga Cantilever



```
Blocks:   2      Center X:   4.50         Center Y:   6.00
Blocks:   4      Center X:   6.00         Center Y:   7.50
Blocks:   6      Center X:   7.50         Center Y:   9.00
Blocks:   8      Center X:   9.00         Center Y:  10.50
Blocks:  10      Center X:  10.50         Center Y:  12.00
Blocks:  12      Center X:  12.00         Center Y:  13.50
Blocks:  14      Center X:  13.50         Center Y:  15.00
Blocks:  16      Center X:  15.00         Center Y:  16.50

Process returned 0 (0x0)   execution time : 0.011 s
Press ENTER to continue.
```

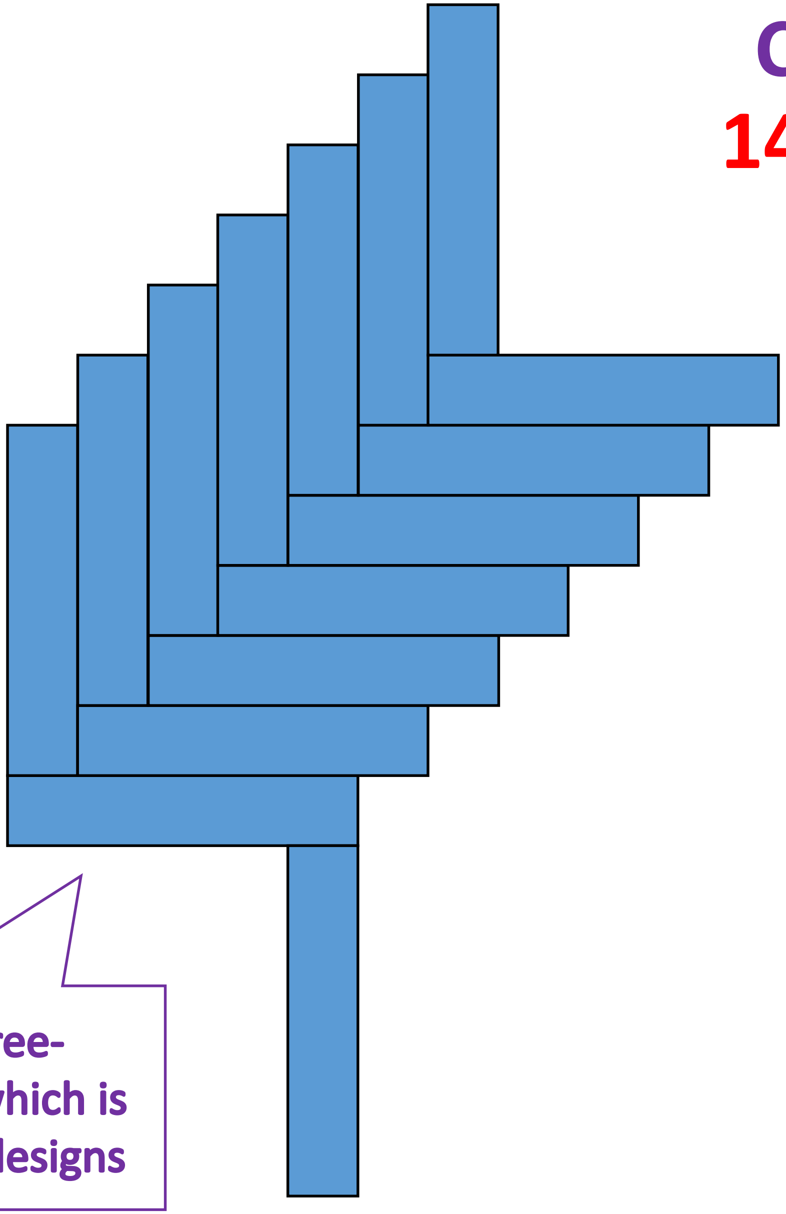
A 16 block pile in this configuration is not possible since Center$_x$ must be less than 15.0 to be balancing somewhere over the vertical post

Check Lab 1
14 Block Jenga Cantilever

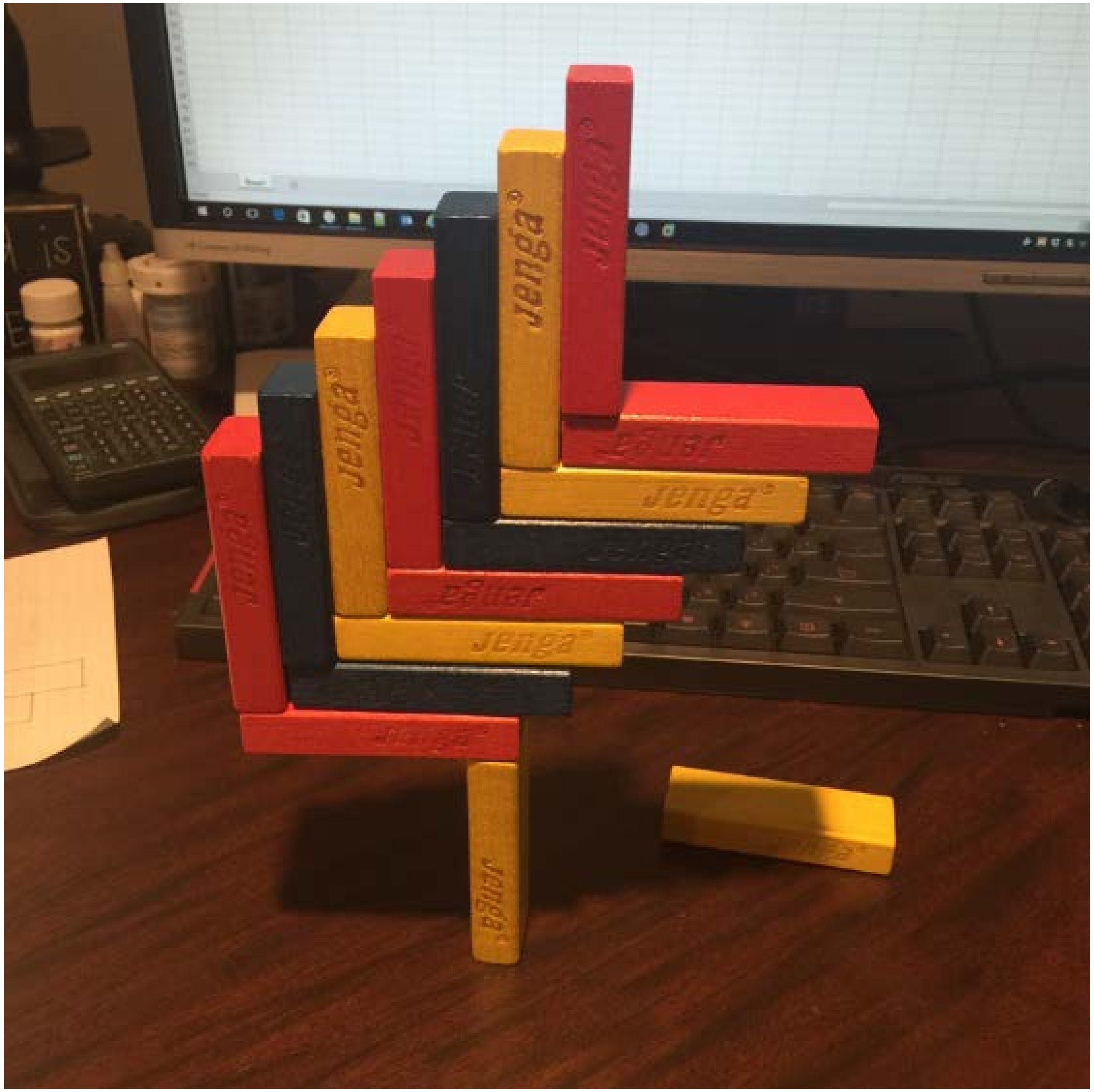


(13.5, 15.0)

Check Lab 1
14 Block Jenga Cantilever

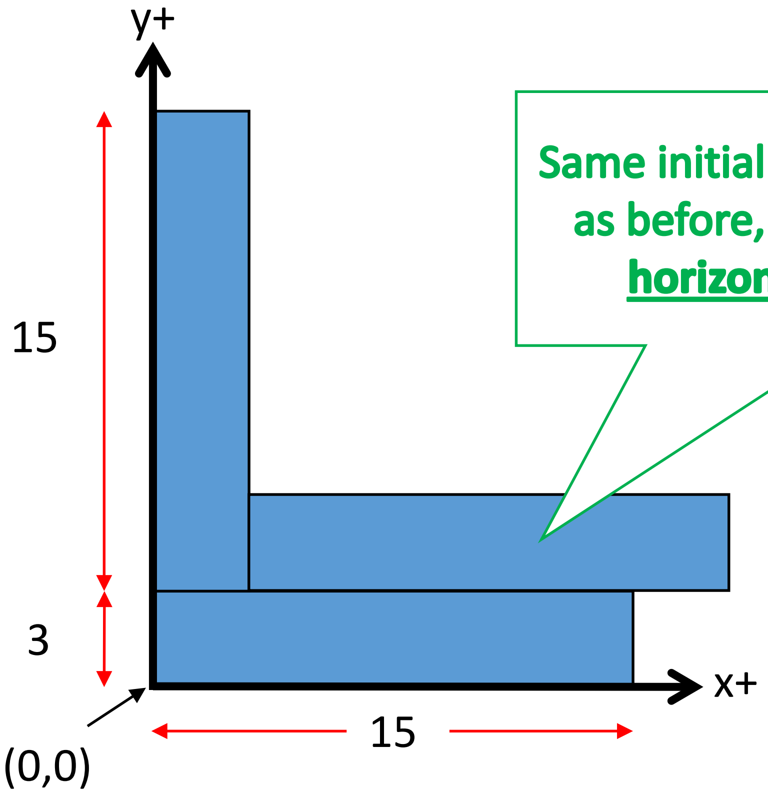


Notice the large free-standing overhang, which is typical of cantilever designs



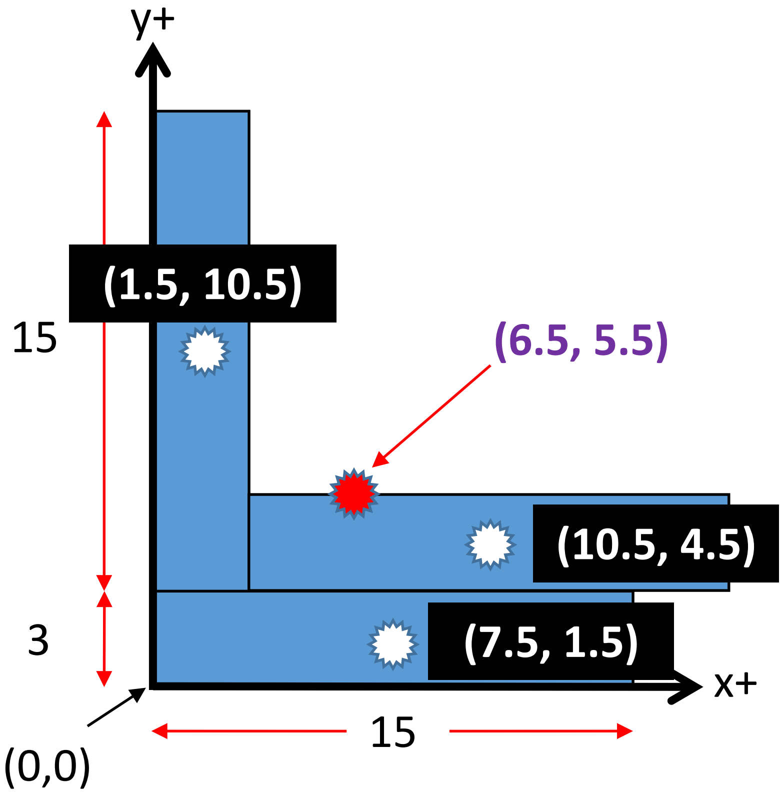
Can we *simulate* the construction of a 15 block Jenga cantilever?

15 Block Pile: 1st Ensemble *(3 blocks)*



Same initial "L" (two-block ensemble) as before, but now with one *extra* **<u>horizontal</u>** block starting out

15 Block Pile: 1st Ensemble *(3 blocks)*



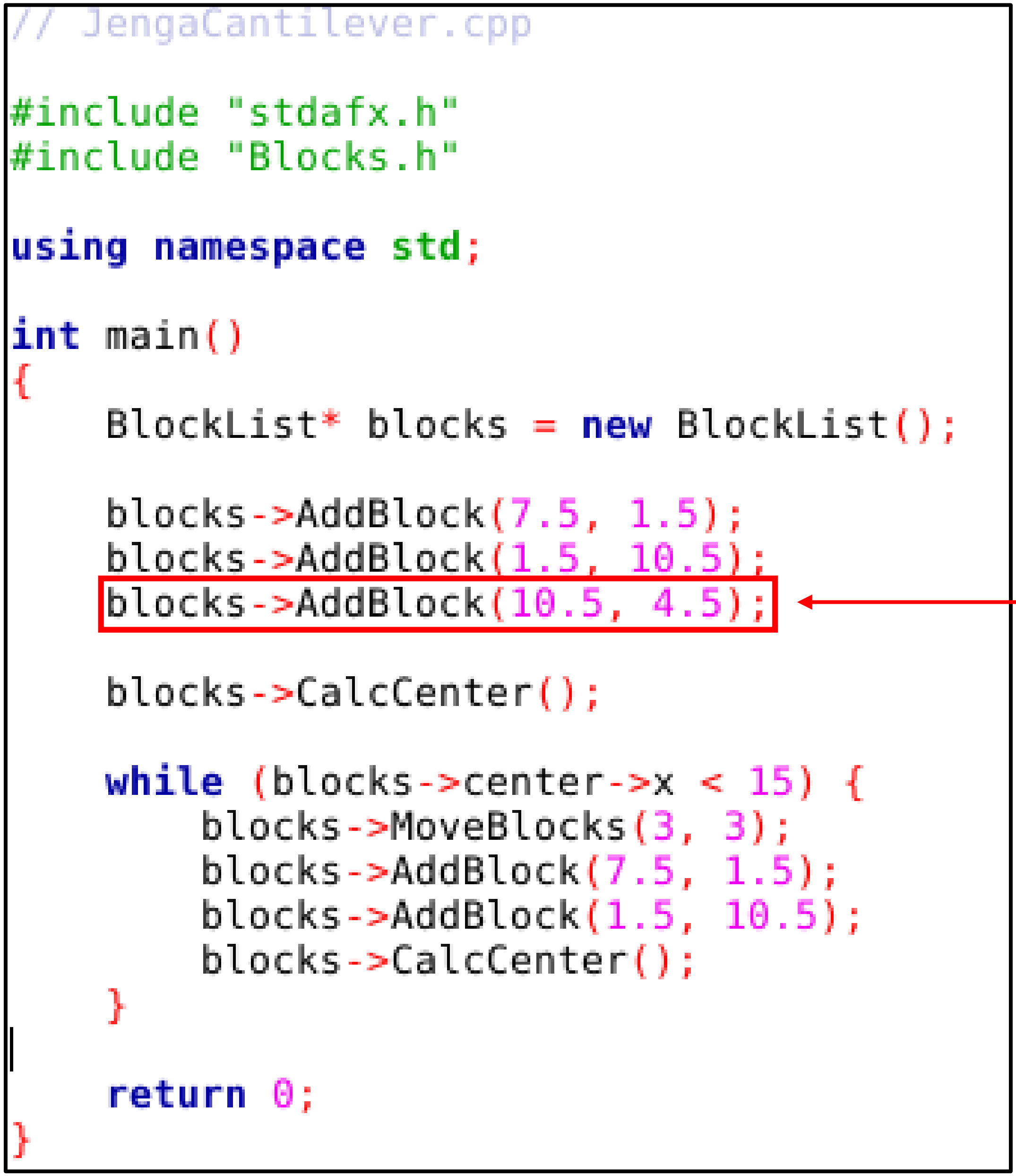
Center of Mass = Average of the centers of all blocks (in **each** **X** & **Y** dimension)

$$C_x = \frac{1.5 + 7.5 + 10.5}{3} = 6.5$$

$$C_y = \frac{10.5 + 1.5 + 4.5}{3} = 5.5$$

Edit

Lab 3 - Jenga-15

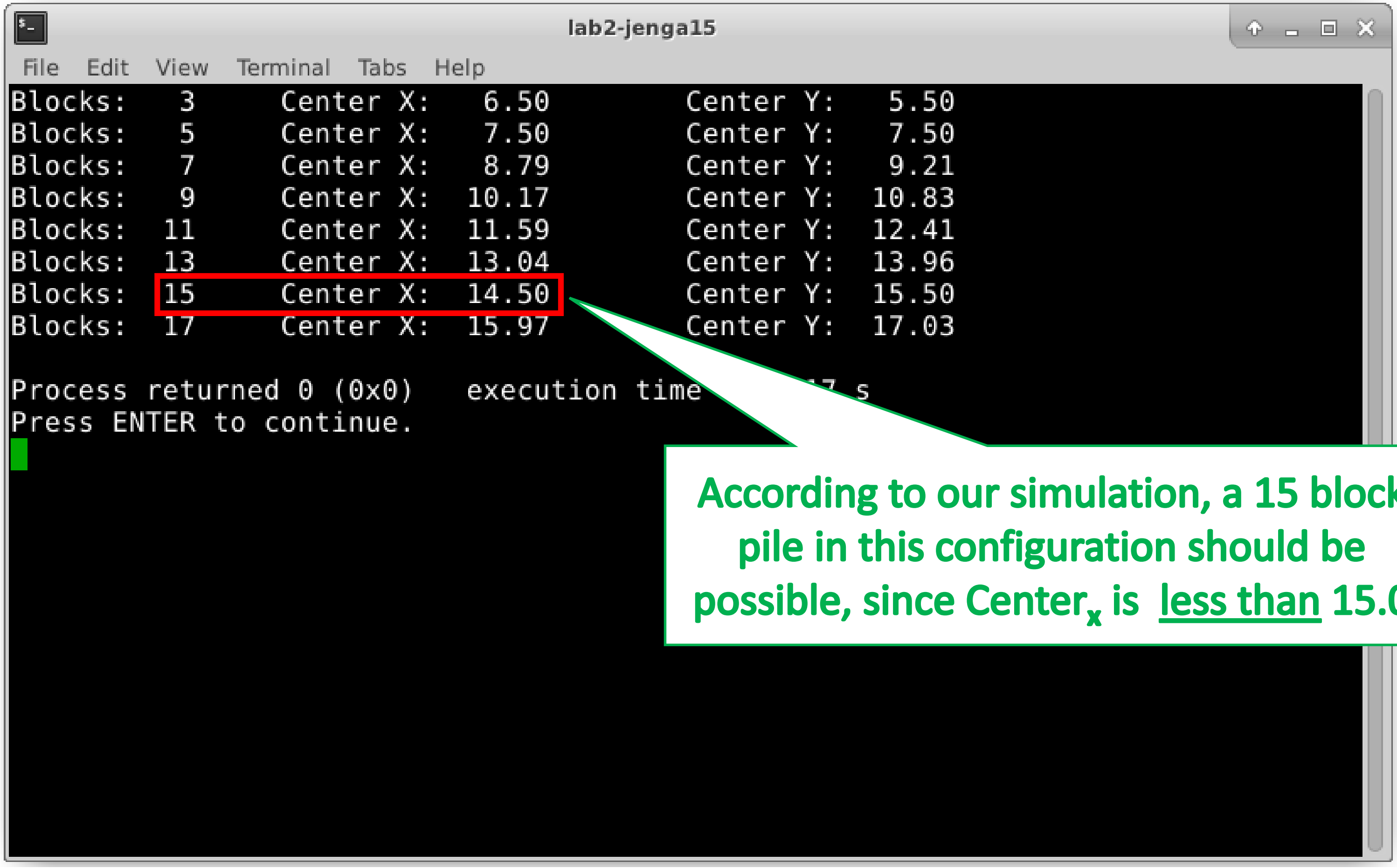


```
// JengaCantilever.cpp

#include "stdafx.h"
#include "Blocks.h"

using namespace std;

int main()
{
    BlockList* blocks = new BlockList();

    blocks->AddBlock(7.5, 1.5);
    blocks->AddBlock(1.5, 10.5);
    blocks->AddBlock(10.5, 4.5);

    blocks->CalcCenter();

    while (blocks->center->x < 15) {
        blocks->MoveBlocks(3, 3);
        blocks->AddBlock(7.5, 1.5);
        blocks->AddBlock(1.5, 10.5);
        blocks->CalcCenter();
    }

    return 0;
}
```

Add these two *Center of Mass* coordinates as we will now begin with a **3-block** ensemble

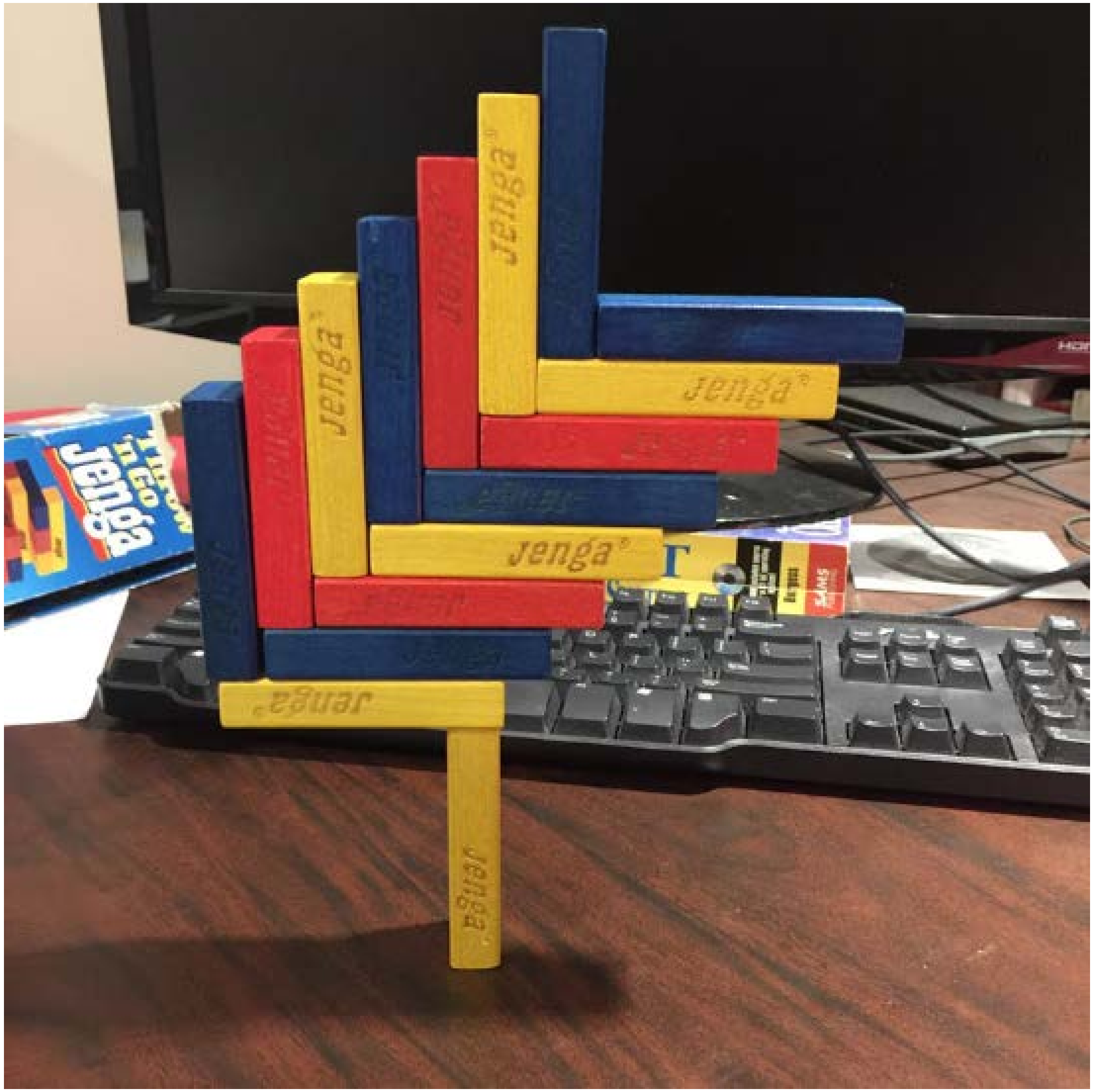
Run

Lab 3 - Jenga-15



```
Blocks:   3     Center X:   6.50        Center Y:   5.50
Blocks:   5     Center X:   7.50        Center Y:   7.50
Blocks:   7     Center X:   8.79        Center Y:   9.21
Blocks:   9     Center X:  10.17        Center Y:  10.83
Blocks:  11     Center X:  11.59        Center Y:  12.41
Blocks:  13     Center X:  13.04        Center Y:  13.96
Blocks:  15     Center X:  14.50        Center Y:  15.50
Blocks:  17     Center X:  15.97        Center Y:  17.03

Process returned 0 (0x0)   execution time
Press ENTER to continue.
```

According to our simulation, a 15 block pile in this configuration should be possible, since Center$_x$ is less than 15.0



Can you build a **15** Block Jenga Cantilever ??

Try this now with your blocks

Functional Equation

$$Pile\ Center\ of\ Mass_X = \frac{\sum_{n=1}^{blocks} \sum_{k=1}^{4} (Coordinate_x\ of\ Corner_k\ of\ Block_n)}{4 * Number\ of\ Blocks}$$

$$C_x = \frac{1}{4}(B_{1x} + B_{2x} + B_{3x} + B_{4x})$$

$$= \frac{1}{4}\big((B_{1x} + \Delta x) + (B_{2x} + \Delta x) + (B_{3x} + \Delta x) + (B_{4x} + \Delta x)\big)$$

$$= \frac{1}{4}\big(4\Delta x + (B_{1x} + B_{2x} + B_{3x} + B_{4x})\big)$$

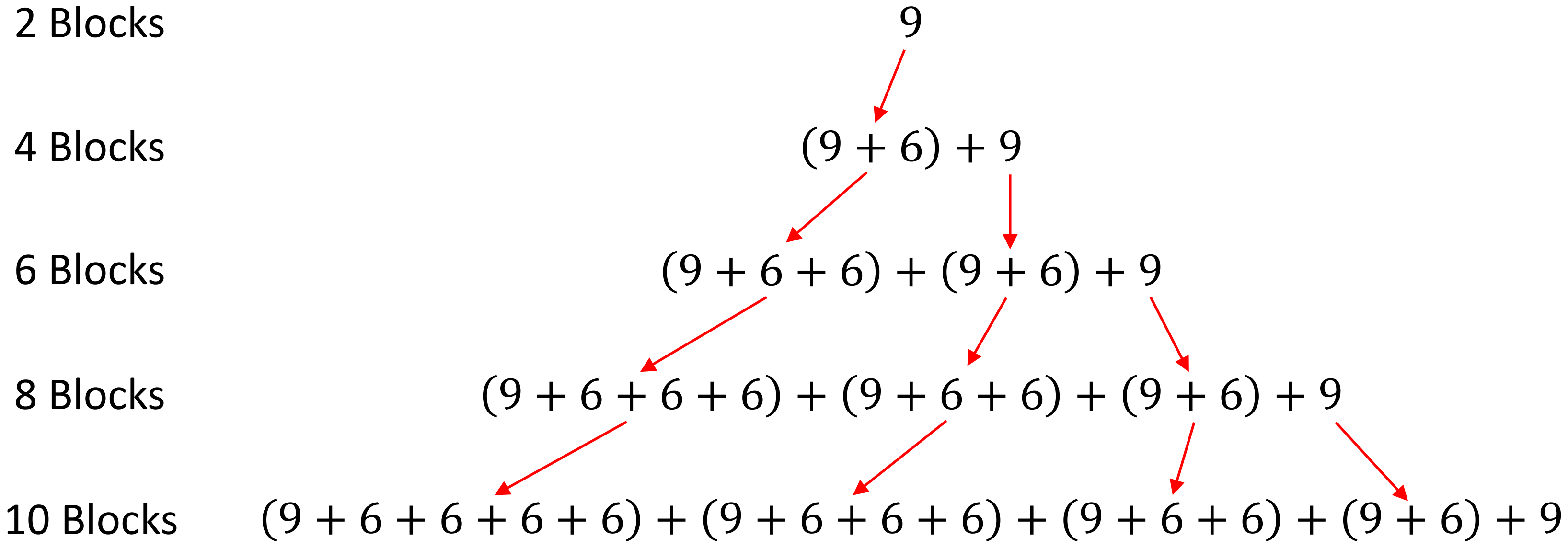
Moving all blocks by Δx moves C_x by Δx

$$C'_x = \Delta x + \frac{1}{4}(B_{1x} + B_{2x} + B_{3x} + B_{4x})$$

Functional Equation – 14 Block Cantilever

$$\sum X_{centers}\ \text{(of a 2 block ensemble)} = 7.5 + 1.5 = \mathbf{9}$$

$$\sum \Delta X_{centers}\ \ \text{(after moving an ensemble)} = 3 + 3 = \mathbf{6}$$



2 Blocks	9
4 Blocks	$(9+6)+9$
6 Blocks	$(9+6+6)+(9+6)+9$
8 Blocks	$(9+6+6+6)+(9+6+6)+(9+6)+9$
10 Blocks	$(9+6+6+6+6)+(9+6+6+6)+(9+6+6)+(9+6)+9$

Functional Equation – 14 Block Cantilever

# of 9's	**# of 6's**	$\sum Center_x \, of$ **Ensembles**
1	**0**	9
2	**1**	$(9+6)+9$
3	**3**	$(9+6+6)+(9+6)+9$
4	**6**	$(9+6+6+6)+(9+6+6)+(9+6)+9$
5	**10**	$(9+6+6+6+6)+(9+6+6+6)+(9+6+6)+(9+6)+9$

Functional Equation – 14 Block Cantilever

# of 9's	**# of 6's**	**n**
1	**0**	1st Ensemble
2	**1**	2nd Ensemble
3	**3**	3rd Ensemble
4	**6**	4th Ensemble
5	**10**	5th Ensemble

$$9n + 6\left(\frac{n^2 - n}{2}\right)$$

$$9n + 3(n^2 - n)$$

$$3(3n + n^2 - n) = 3n(n + 2)$$

$$\boxed{Center\ of\ Mass_x = \frac{3n(n+2)}{2n}}$$

There are two blocks per ensemble

Functional Equations

Jenga 14 Block Cantilever

$$Center\ of\ Mass_X = \frac{3n(n+2)}{2n}$$

$$n \equiv Number\ of\ Ensembles$$

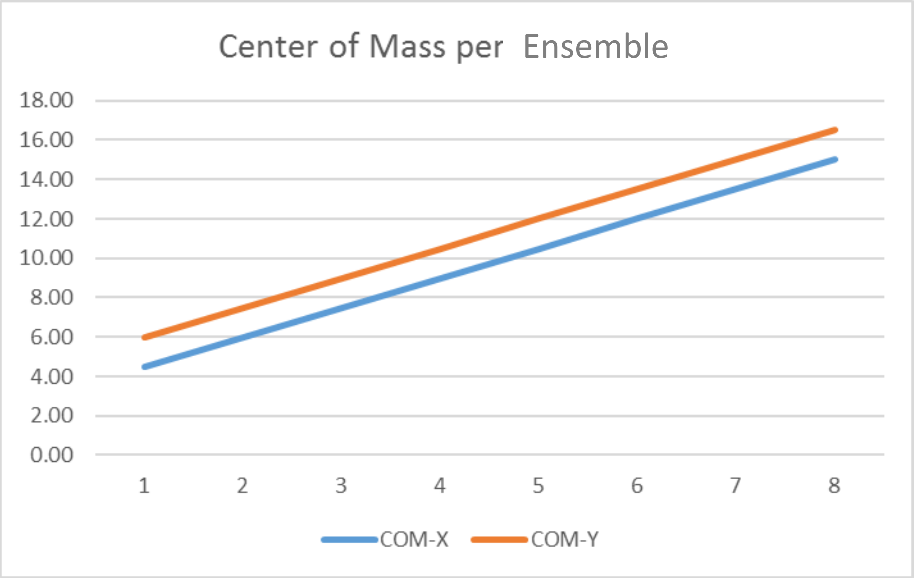
Jenga 15 Block Cantilever

$$Center\ of\ Mass_X = \frac{19.5 + 3(n-1)(n+4)}{2n+1}$$

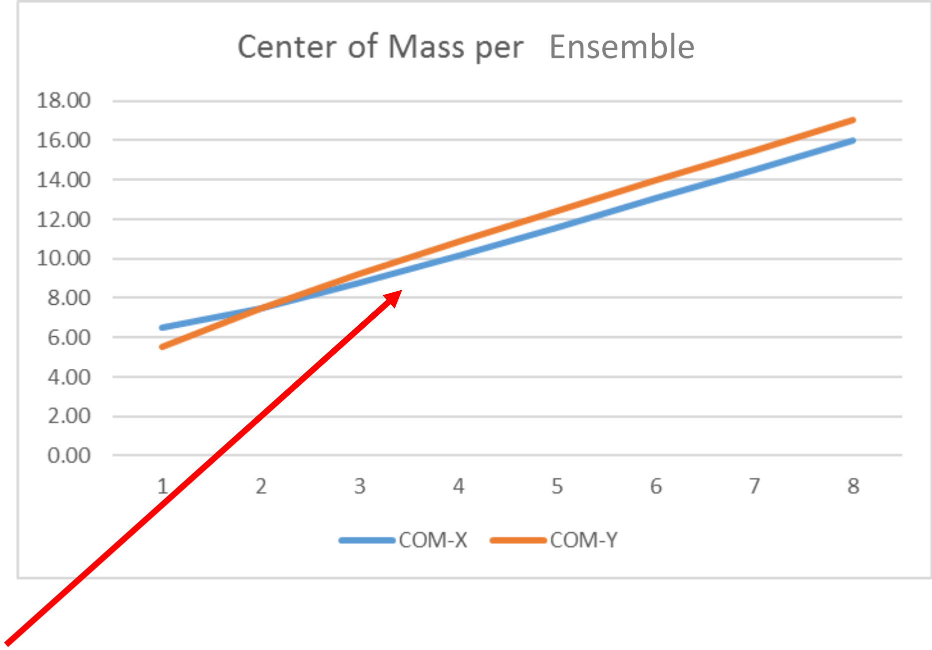
$$n \equiv Number\ of\ Ensembles$$

Functional Equations

Jenga 14 Block Cantilever

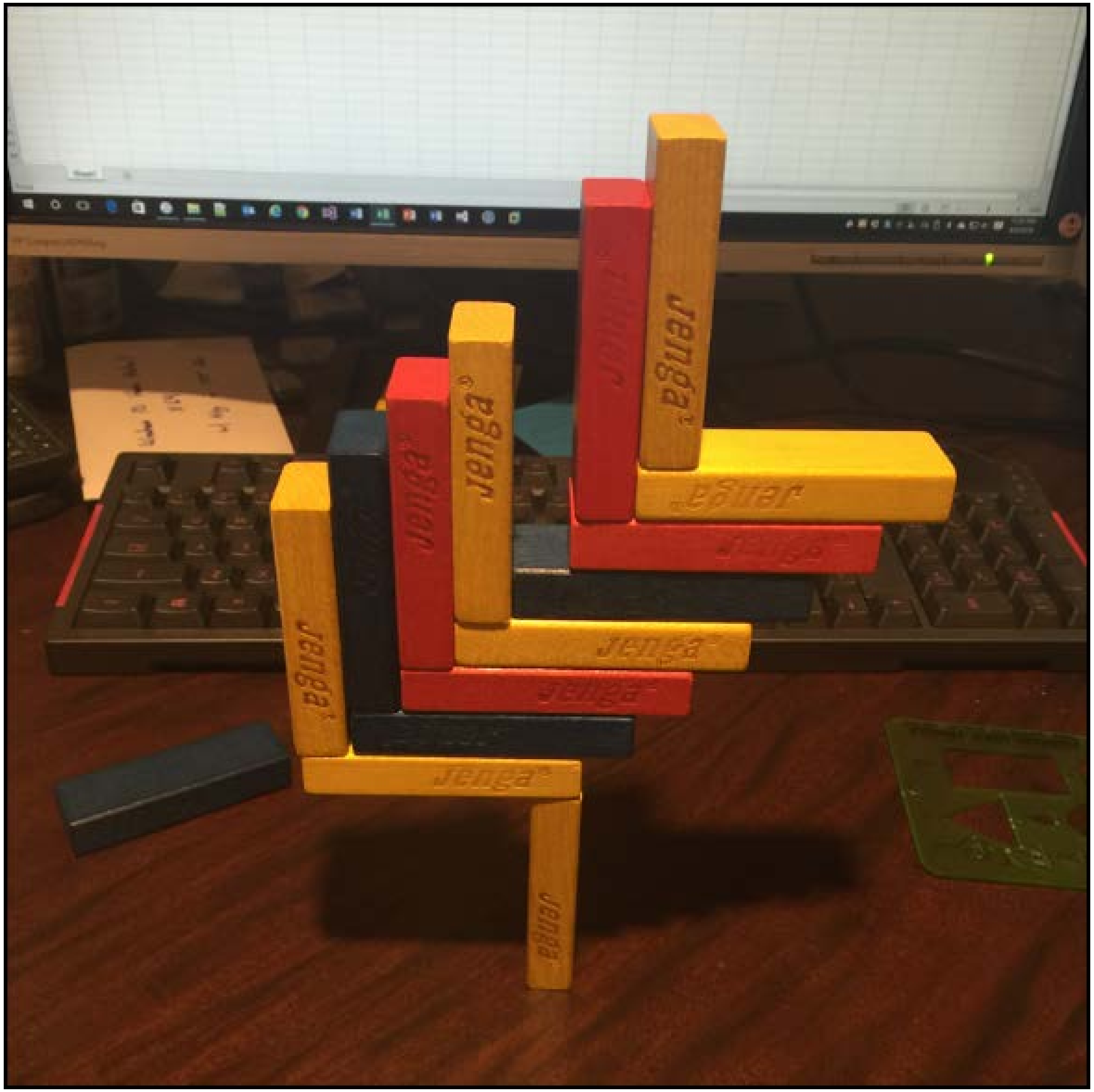


Jenga 15 Block Cantilever

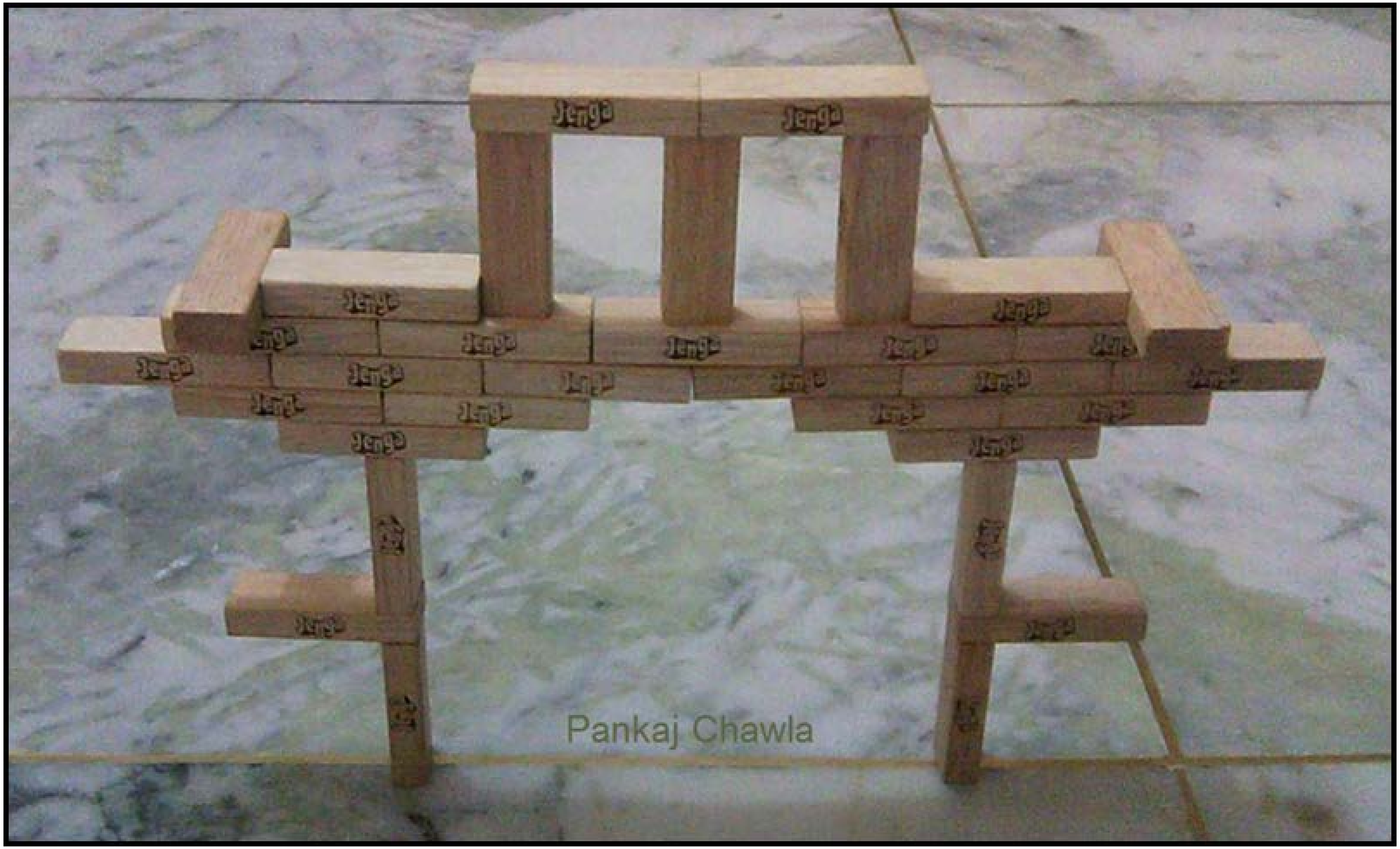


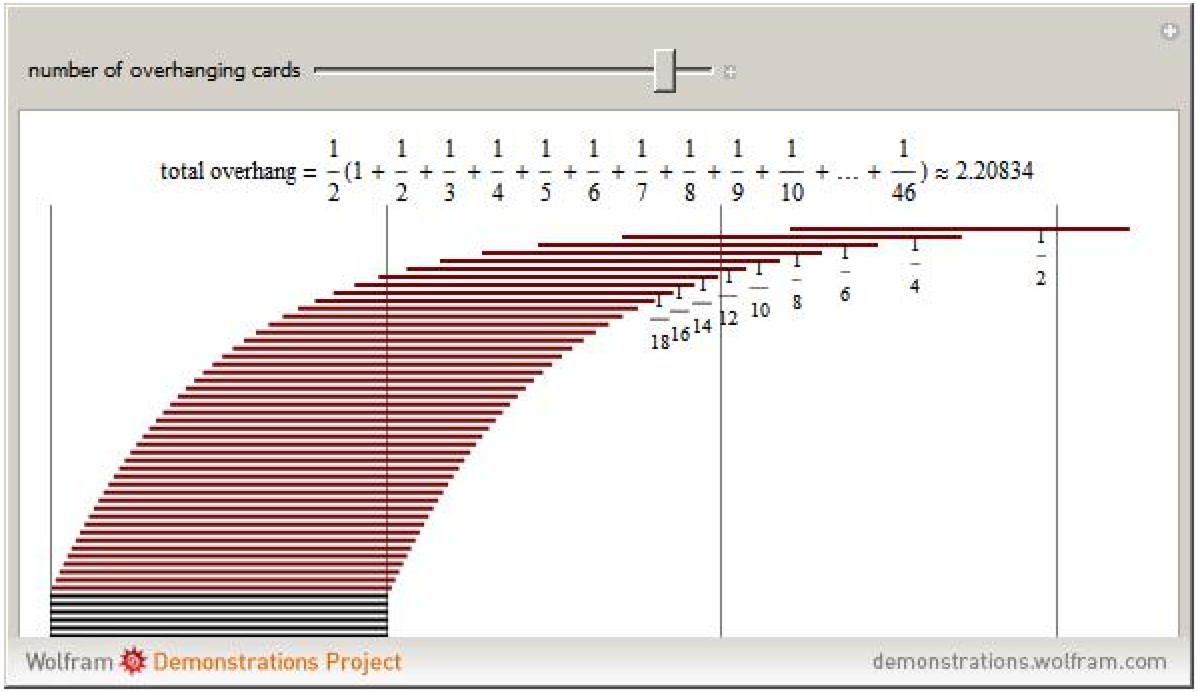
The center-of-mass in the X & Y dimensions are closer in a 15 block cantilever so it can rotate (tip over) more easily than a 14 block cantilever

Cantilever Building Design



13 Block Jenga Cantilever (with missing middle)


Pankaj Chawla



number of overhanging cards

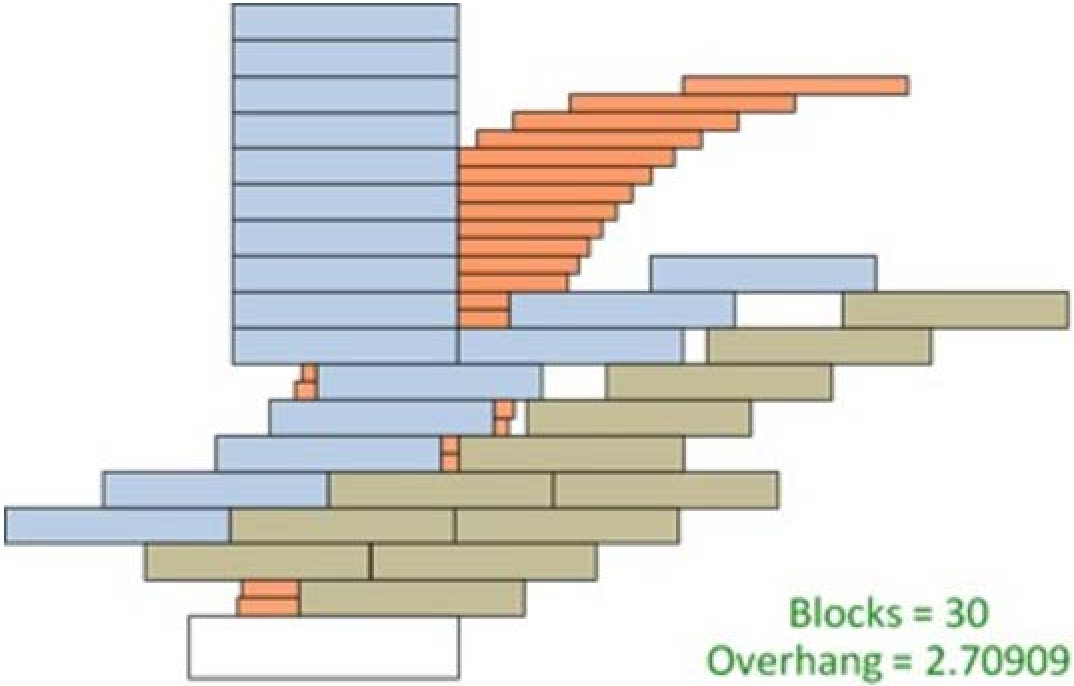
total overhang = $\frac{1}{2}(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \ldots + \frac{1}{46}) \approx 2.20834$

Maximum Overhang, Optimum Reward

March 3, 2011 | Posted by Microsoft Research Blog

By Janie Chang, Writer, Microsoft Research

Yuval Peres, princip s advocates both
healthy skepticism prised when a
paper he co-autho merica (MAA) for
taking a completel



A classic stack of 30 blocks (in orange) compared to the same number of blocks configured as a jagged wall (in gray) held stable by counterweights (in blue).

Now you know…

- Looping with **for()** and **while()** statements
- Using a **class** to group data elements and functions
- Identity = “is-a” while ownership (contains) = “has-a”
- A **vector<>** template is a flexible container of similar types
- SciComp often involves ***simulating*** the real world before spending $$$
- Developing a **functional equation** can often produce the answer faster than looping to calculate it
- A complex construction of objects balances on its **virtual center of mass**
- Cantilever designs can reduce load on external walls to increase building **survivability**