# Lecture 4: Top 10 reasons why lists are awesome
## (#7 will confuse your debugger!)

Bart Iver van Blokland

(Rune Sætre)

# Today

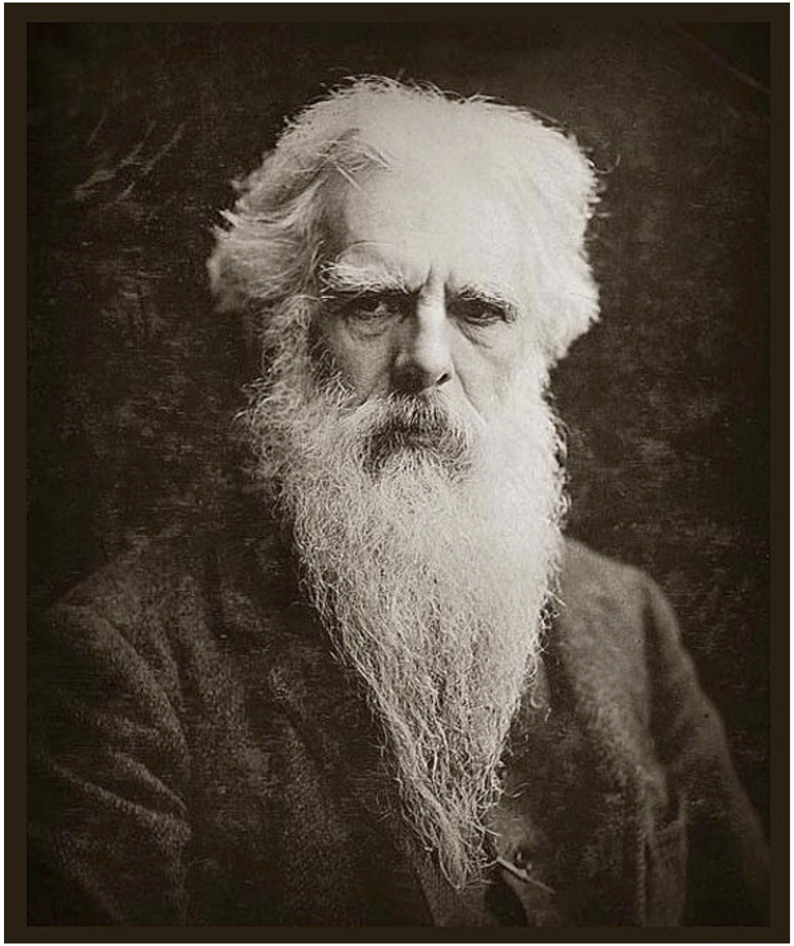
Everything you need to finish assignment 2!
(as well as animations)

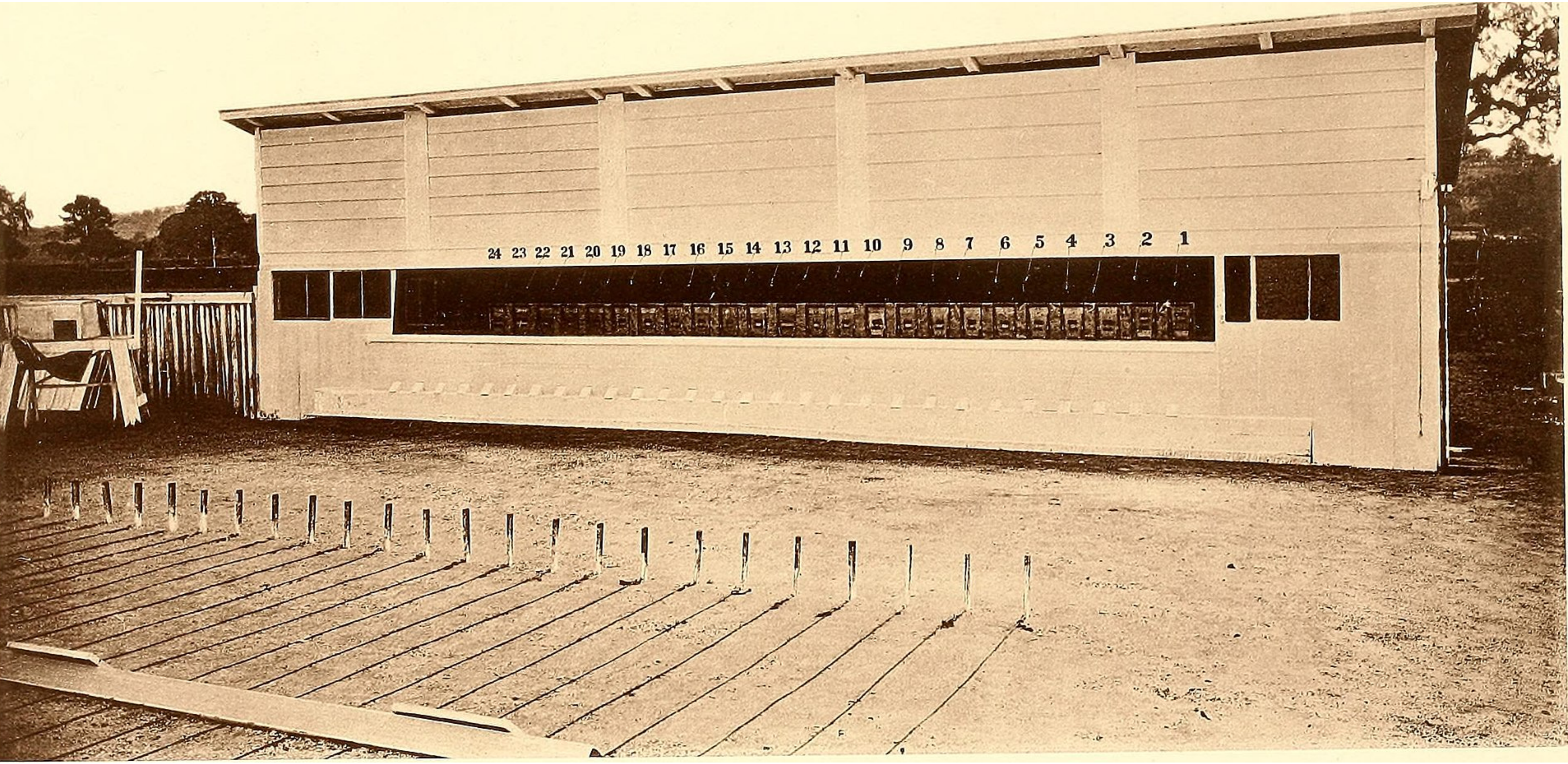
- **Animation**
- Vectors
- Arrays
- Random number generation

# 1878

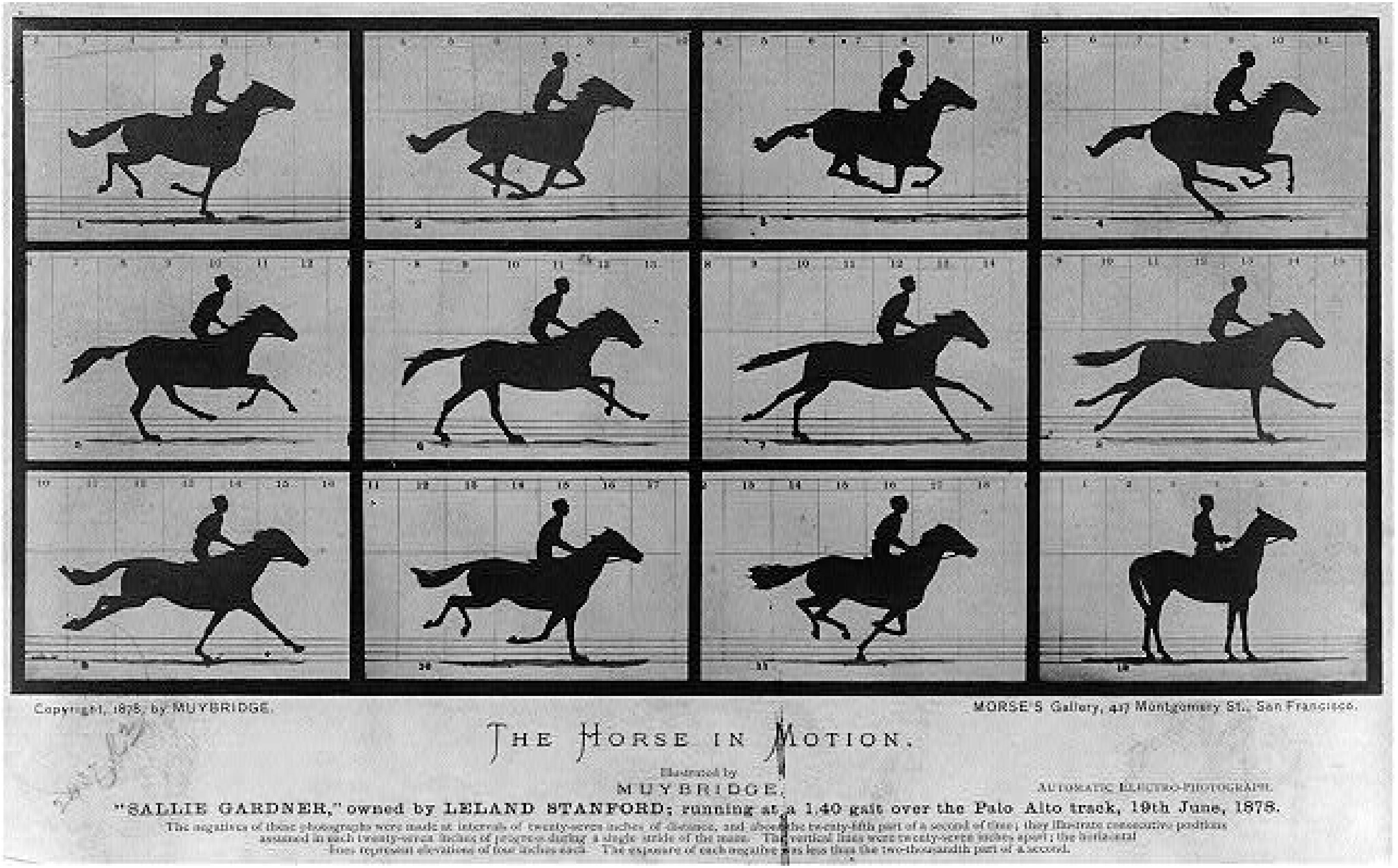
Leland Stanford

Eadweart Muybridge



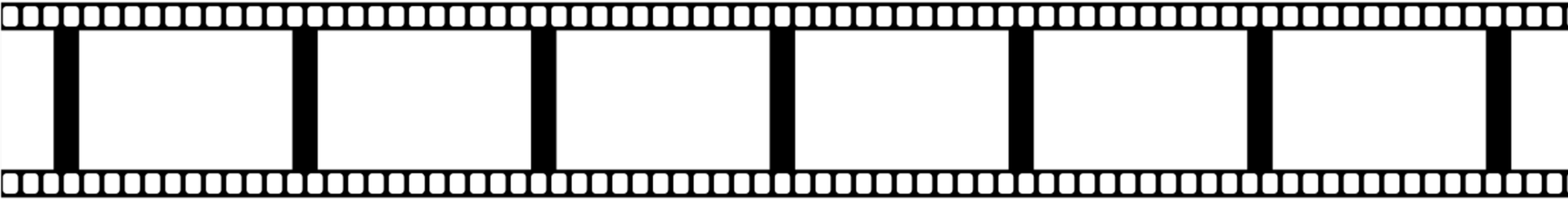



May? Yes of course you may! <3



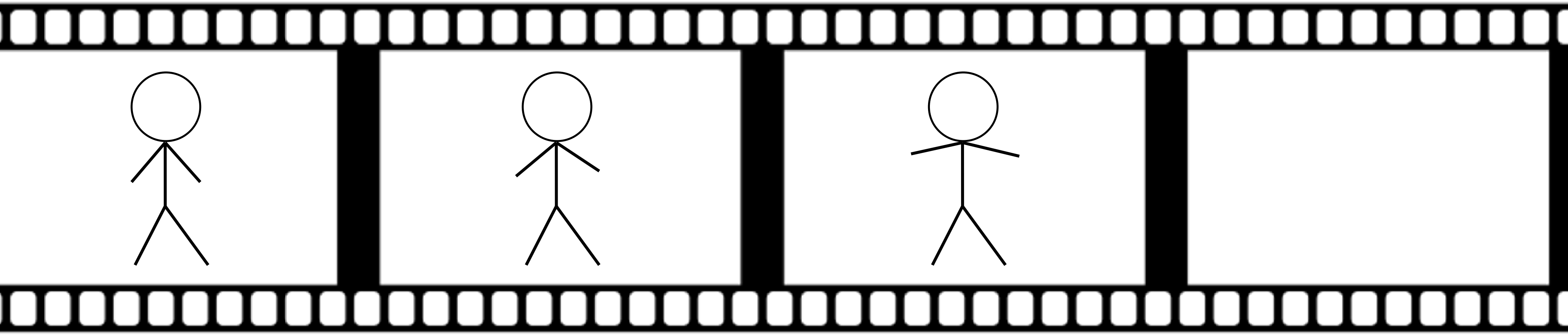

May? Yes of course you may! <3

# Let's make

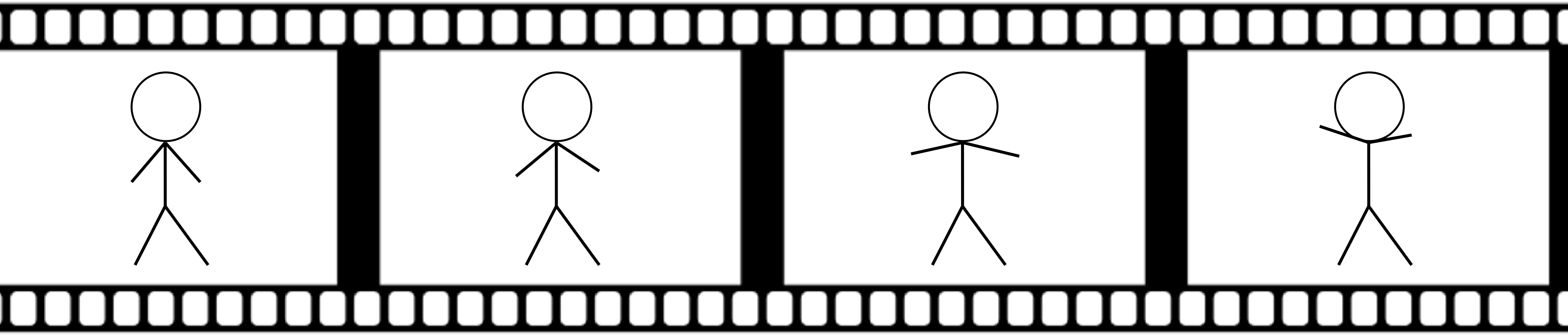


# an animation

# 1. Create a new empty frame



# 1. Create a new empty frame



# 2. Draw a picture in that frame

# Animation

Keep running until the window is closed

```
AnimationWindow window;
int xCoordinate = 0;
while(!window.should_close()) {
    xCoordinate++;
    if(xCoordinate > 500) {
        xCoordinate = 0;
    }
    window.draw_circle({xCoordinate, 100}, 50);
    window.next_frame();
}
```

To animate something, we just have to move it a little bit for each picture we draw

We have drawn a picture, and can move on to drawing the next one

# Today

Everything you need to finish assignment 2!
(as well as animations)

- Animation
- **Vectors**
- **Arrays**
- Random number generation

Note! The book is slightly outdated on this topic and only discusses older C-style arrays.

# Vectors and Arrays

- We often want to not just store one value, but a list of multiple values that are related.

  For example:
  - Measurements from sensors
  - At what times you unlocked your phone throughout the day
  - Your player inventory in a game

- Vector: list that can expand dynamically
  - Similar to a List in Python

- Array: list that has a constant number of elements
  - Similar to a Tuple in Python

- The primary difference with Python is that in C++ lists, all elements must have the same datatype.

# Using vectors

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| `vector<int> vec {57, 19, 28, 89, 65, 68, 49};` | 57 | 19 | 28 | 89 | 65 | 68 | 49 | | |
| `vec.push_back(-43);` | 57 | 19 | 28 | 89 | 65 | 68 | 49 | **-43** | |
| `vec.at(0) -= 15;` | **42** | 19 | 28 | 89 | 65 | 68 | 49 | -43 | |
| `vec.at(5)++;` | 42 | 19 | 28 | 89 | 65 | **69** | 49 | -43 | |
| `vec.at(2) = vec.at(7);` | 42 | 19 | **-43** | 89 | 65 | 69 | 49 | -43 | |
| `vec.push_back(vec.size());` | 42 | 19 | -43 | 89 | 65 | 69 | 49 | -43 | **8** |
| `vec.resize(5);` | 42 | 19 | -43 | 89 | 65 | | | | |
| `vec.resize(8);` | 42 | 19 | -43 | 89 | 65 | 0 | 0 | 0 | |

```
vec.at(10) = 5;  // ERROR: index out of range
```

# Using arrays

If you do not specify all values, they will be the default value.

```
array<int, 5> arr {-65, 86, 10, 63};
```

| | | | | |
|---|---|---|---|---|
| -65 | 86 | 10 | 63 | 0 |

```
arr.at(4) = 51;
```

| | | | | |
|---|---|---|---|---|
| -65 | 86 | 10 | 63 | **51** |

```
arr.at(2)++;
```

| | | | | |
|---|---|---|---|---|
| -65 | 86 | **11** | 63 | 51 |

```
arr.at(0) = arr.at(1) + 10;
```

| | | | | |
|---|---|---|---|---|
| **96** | 86 | 11 | 63 | 51 |

```
arr.at(3) = arr.size();
```

| | | | | |
|---|---|---|---|---|
| 96 | 86 | 11 | **5** | 51 |

```
arr.fill(20); // vector does not have this!
```

| | | | | |
|---|---|---|---|---|
| **20** | **20** | **20** | **20** | **20** |

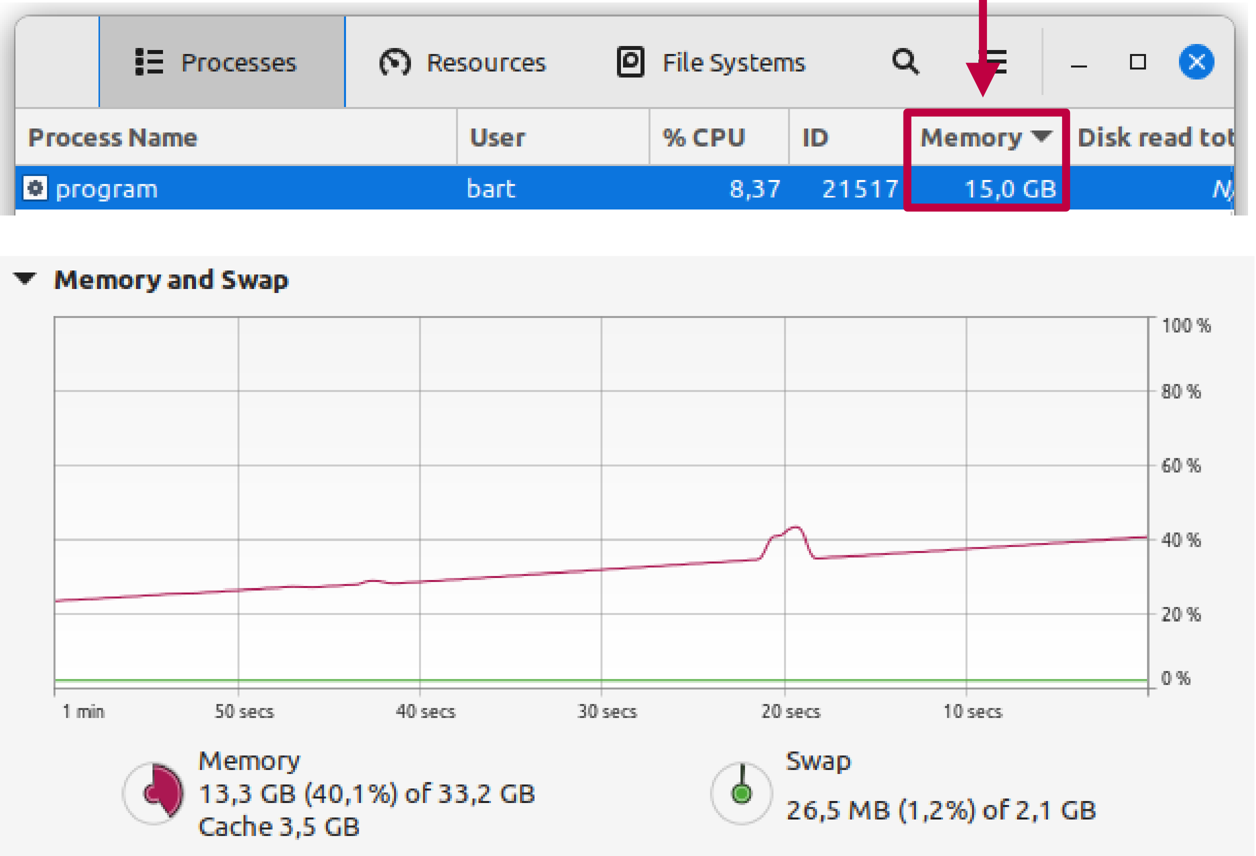
```
arr.at(-1) = 5;  // ERROR: index out of range
```

Vectors can store as much data as your computer has space for

This test program creates a massive vector.

You can even see the memory the program is using increase in Activity Monitor (Mac) or Task Manager (Windows)

```
int main() {
    vector<int> vec;
    while(true) {
        vec.push_back(5);
    }
}
```



# Which should you use?

- Vector:
  If you need a list, the vector is the most flexible of the two, and if you don't have a good reason to use an array, the vector should be your default choice.

- Array:
  If you need to store a fixed (usually small) list whose length will never change, the array has a speed advantage over the vector, and communicates its size to someone reading your code.

You can see the function expects a 3D (maths) vector

```
double computeSpeed(array<double, 3> distanceCovered, double timeSpent) { /* .. */ }
double computeSpeed(vector<double> distanceCovered, double timeSpent) { /* .. */ }
```

# You *really* shouldn't use the [ ] operator

- The [ ] operator that Python uses for lists is also in C++:

```
vector<int> vec {1, 2, 3, 4, 5};
vec[3] = 28;
cout << vec[1] << endl;
```

- Except there is one small difference: it does not check if the index you put in is inside the bounds of the list:

```
vector<int> vec {1, 2, 3, 4, 5};
cout << vec[5] << endl; // no error!
cout << vec.at(5) << endl; // error: out of range
```

# You *really* shouldn't use the [ ] operator

- C++ has C-style arrays that only have the [ ] operator and no bounds checking. They're not recommended!

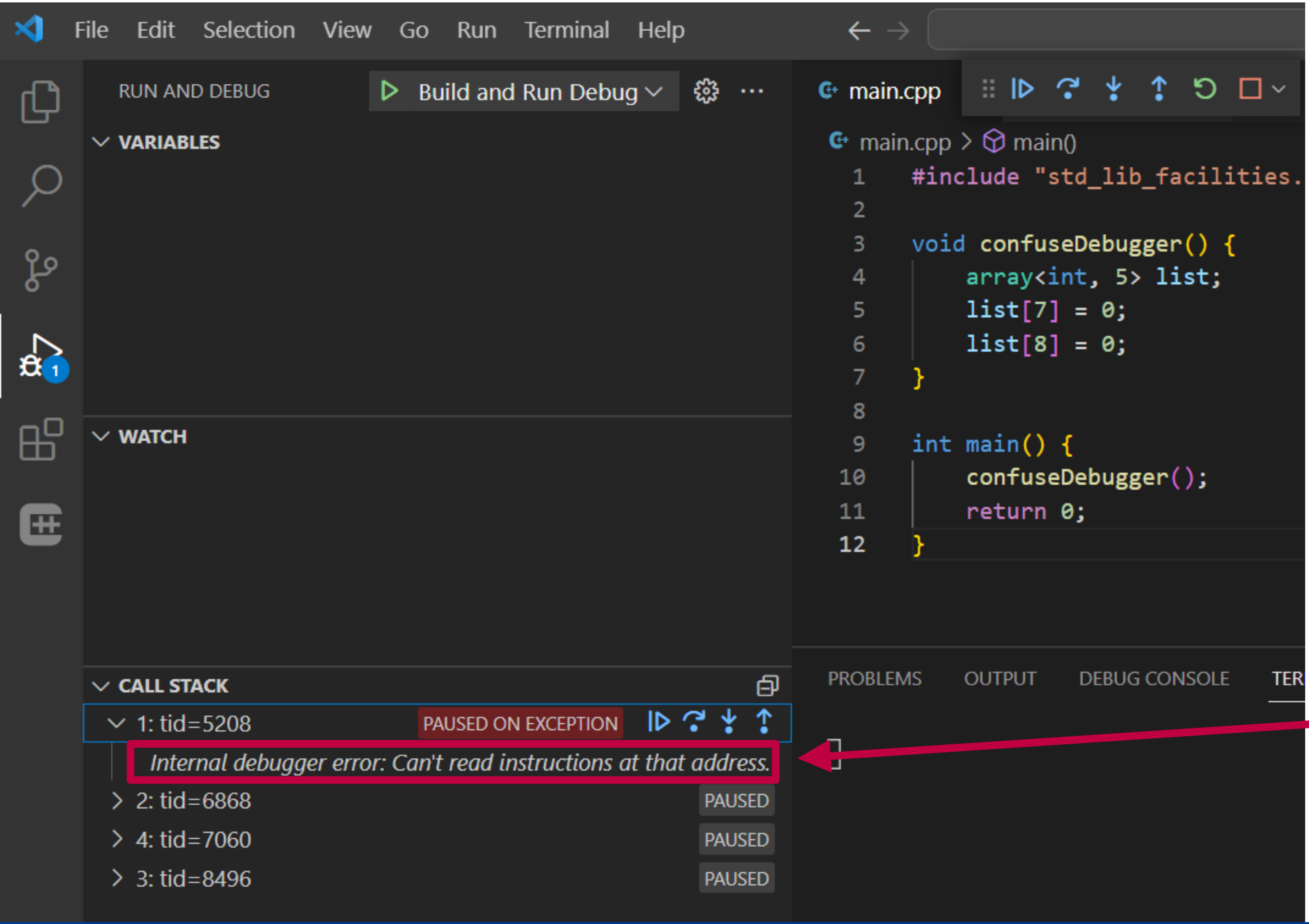
```
int vec[5] {1, 2, 3, 4, 5};
cout << vec[5] << endl;
cout << vec.at(5) << endl; // Does not compile!
```

# It may not sound like a big deal, but it really is!

- At best your program is not affected, but it is likely to cause bugs, and the program will not tell you.
- It can also cause some *very* funky behaviour:

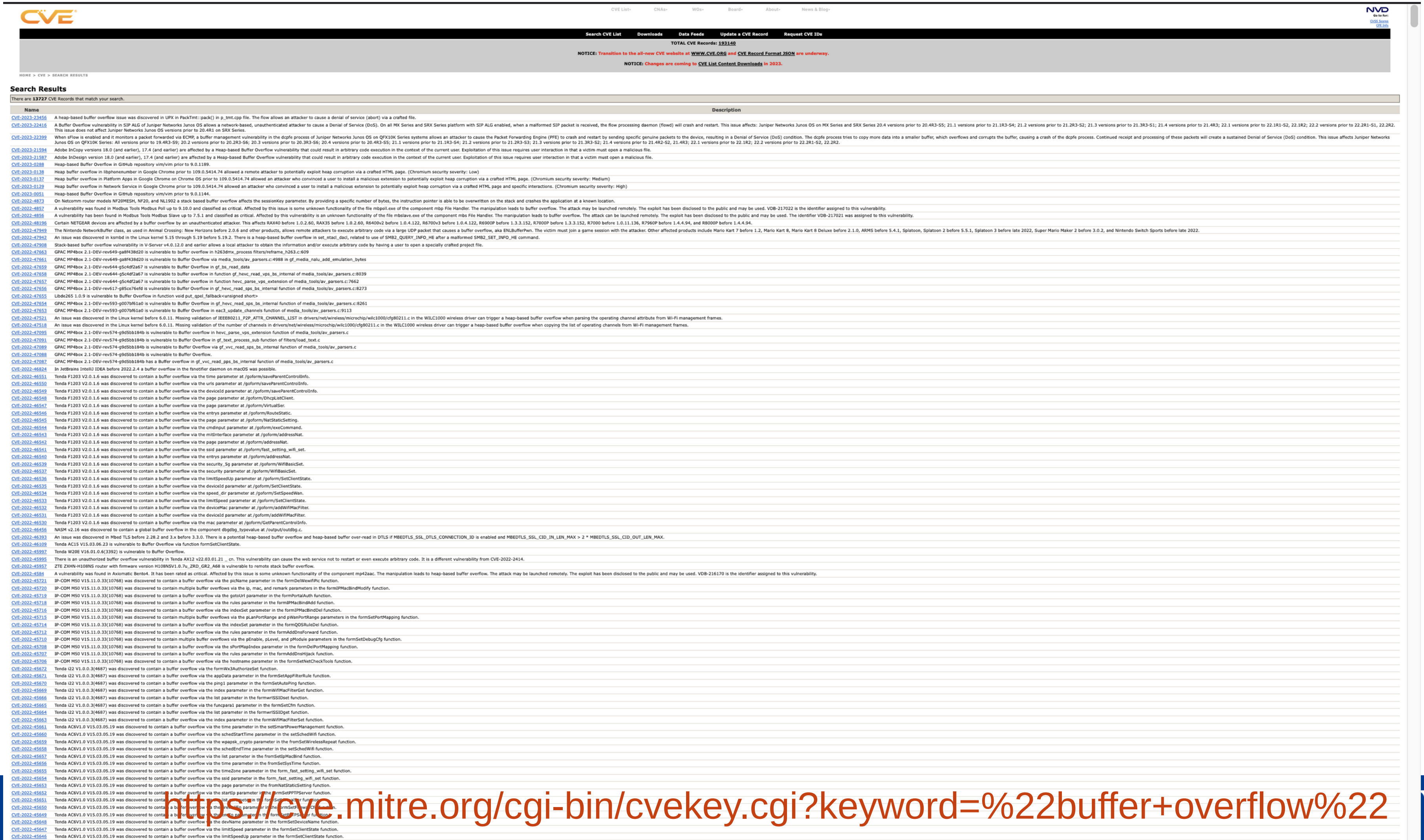
```
int main() {
    int value = 10;
    array<int, 5> list;
    list[5] = 5;
    cout << value << endl; // often prints 5
    return 0;
}
```

# You can even crash your program!




To the point that the debugger is confused!

# Or exploit it in nefarious ways..



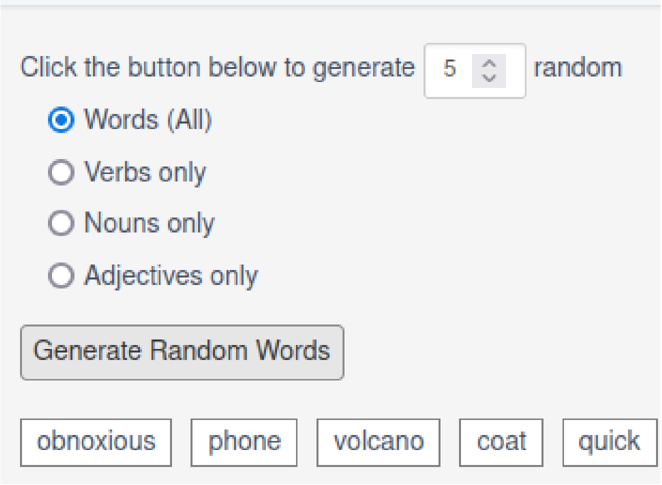

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=%22buffer+overflow%22

# Obnoxious Phone Volcano Coat Quick

# Obnoxious Phone Volcano Coat Quick

Everything you need to finish assignment 2!

- Animation
- Vectors
- Arrays
- **Random Number Generation (RNG)**



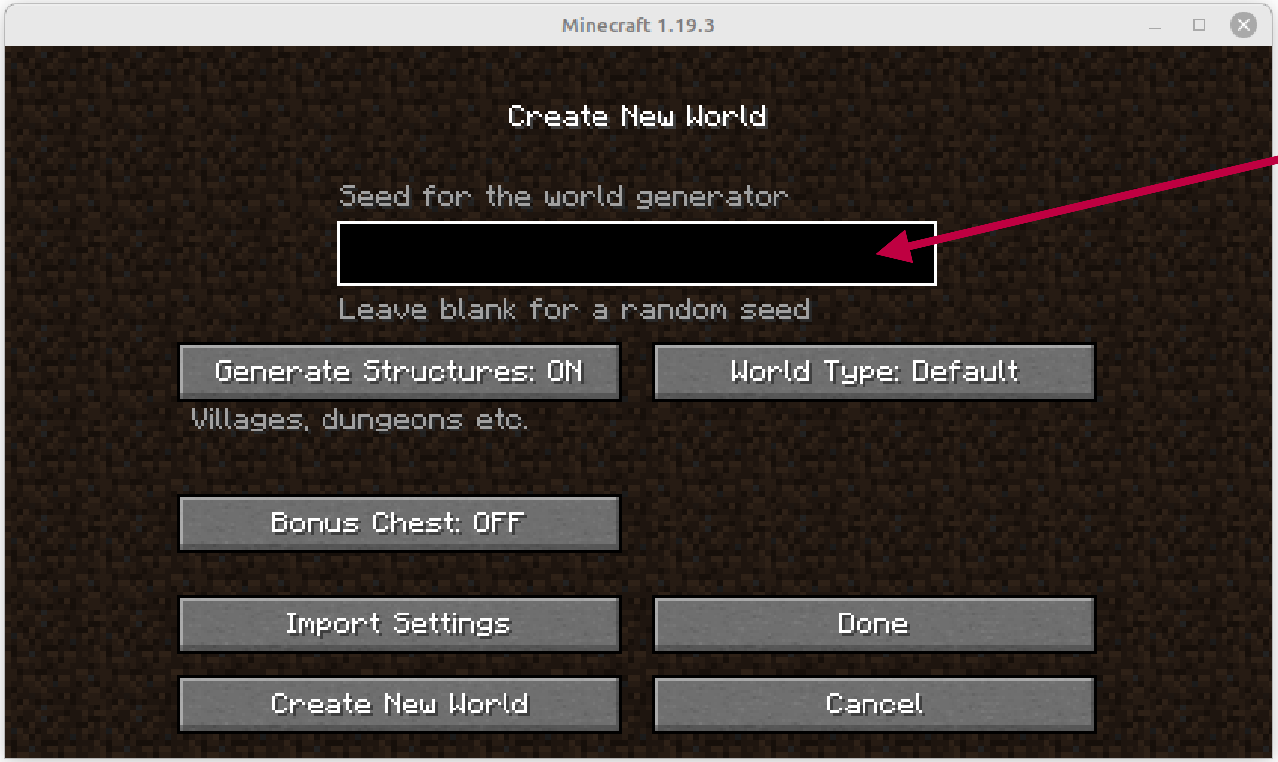

https://i.guim.co.uk/img/media/66b1153025abd2b615be1845b9b6c1b6d7a7f2ab/240_0_3600_2160/master/3600.jpg?width=1200&quality=85&auto=format&fit=max&s=4eb919da519aabdbfb36c8d00216f1db

# Random

- Computers are great at computing things, but not so much at choosing things randomly
- We therefore compute pseudorandom numbers
- A pseudorandom number generator function computes a new random value based on the one it generated previously.
- The initial value is called a «seed», and is often derived from another source of randomness

$$501 \xrightarrow{f(501)} 384 \xrightarrow{f(384)} 277 \xrightarrow{f(277)} 914 \xrightarrow{f(914)} 530 \xrightarrow{f(530)} 464$$

# One way to generate randomness: point a camera at lava lamps!




Some games even let you set the random seed value explicitly

This can also have an advantage: random numbers are now predictable!

# Random

- Using the C++ standard library to compute random numbers

```
// Generate a random seed using the operating system
random_device device;
unsigned long randomSeed = device();

// Initialise the random engine
default_random_engine engine(randomSeed);

// Now you can generate numbers!
uniform_int_distribution integerDistribution(0, 10000);
for(int i = 0; i < 100; i++) {
    cout << "Generated a random integer: " << integerDistribution(engine) << endl;
}
```

This tends to be costly, so we only use it to generate a random seed.

You can use any value you please as the random seed. We'll use the randomly generated one here.

This is the range in which random numbers will be generated (between 0 and 10,000).

This generates the random number

# Random

- Using the C++ standard library to compute random numbers

```
// Generate a random seed using the operating system
random_device device;
unsigned long randomSeed = device();

// Initialise the random engine
default_random_engine engine(randomSeed);

// Now you can generate numbers!
uniform_real_distribution floatDistribution(0.0, 10000.0);
for(int i = 0; i < 100; i++) {
    cout << "Generated a random float: " << floatDistribution(engine) << endl;
}
```

Want floating point numbers instead?
It's almost exactly the same process!
You just use a uniform_real_distribution

# Random

- Pseudorandom number generator functions can be surprisingly simple. Here’s the one used in many implementations of the C++ standard library:

```
unsigned int random(unsigned int previousNumber) {
    unsigned long multiplier = 16807;
    unsigned long divider = 2147483647;
    return (multiplier * previousNumber) % divider;
}
```

# We are vectorious!

And done for today :)

Next time: how do we add new files to our project?

Reminder: Assignment 1 is due this Friday!