

Lecture 4: The Vector Enlistening

Bart Iver van Blokland

PSA: Lectures

- One lecture may be moved some time during the next few weeks

PSA: std::println()

- Issues on Windows have been fixed!
 - For new projects: should work right off the bat
 - For existing projects: open folder in VS Code, then use Create new project > GUI configuration
- Are you on MacOS and using Sequoia or Tahoe ($\geq 15.7.2$)?
 - If you get compilation errors because of `std::println()`, open Terminal and use the following command:
`sudo rm -rf /Library/Developer/CommandLineTools`
You will be asked your password. Press enter when done. Then:
`xcode-select -install`
Follow the on-screen instructions in the window that pops up
When it's done, running `clang --version` in the terminal should show version 17 or higher:

```
Valhalla: ~ % clang --version
Apple clang version 17.0.0 (clang-1700.6.3.2)
```


PSA: Insperaøving

- Must be approved to be able to take the exam!
- No need to prepare, the point is to get used to the process of writing and delivering code on the exam computers
 - This is intended to be a tutorial in the real exam setting, not a test of your knowledge!
- Held physically at Sluppen
- Dates are not entirely set, but currently looks like:
 - 14th of April 9:00 – 13:00
 - 17th of April 9:00 – 13:00

(later than initially planned!)

Last lecture

- C++ syntax
- Debugging
- Strings
- Functions
- Basics of graphics

Today

- **Obnoxious Phone Volcano Coat Quick**
- `std::vector` and `std::array`
- `Const` & `constexpr`
- The terminal (if there's time)

Obnoxious Phone Volcano Coat Quick

The Random Number Generator

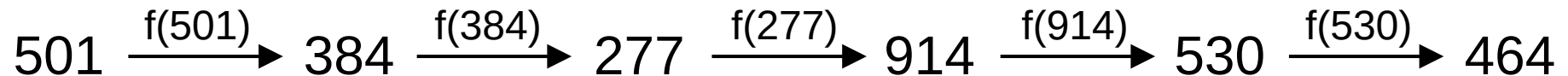
**Me trying to farm
a common item**



Demonstration: RNG in C++

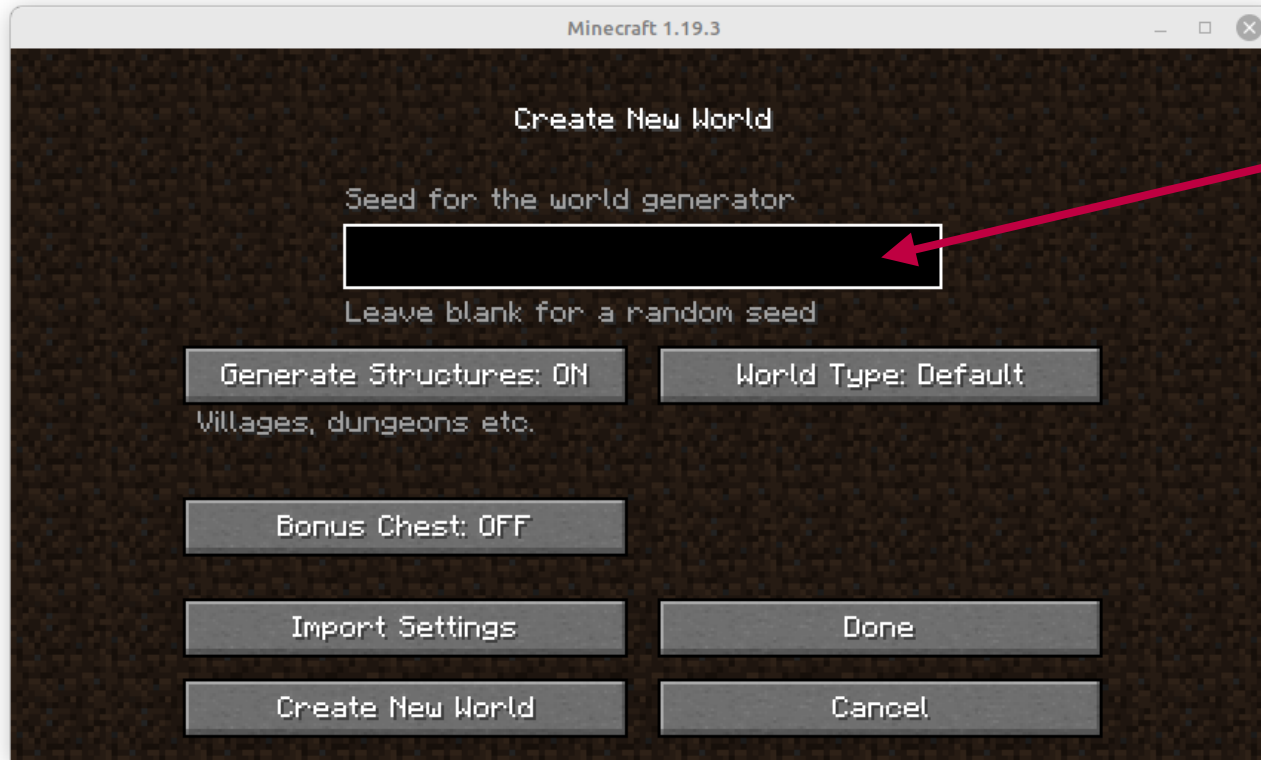
Random

- 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



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

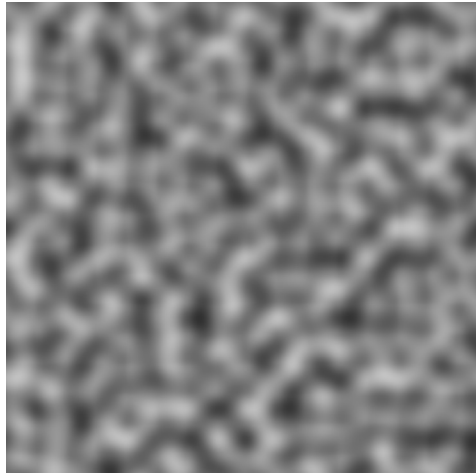


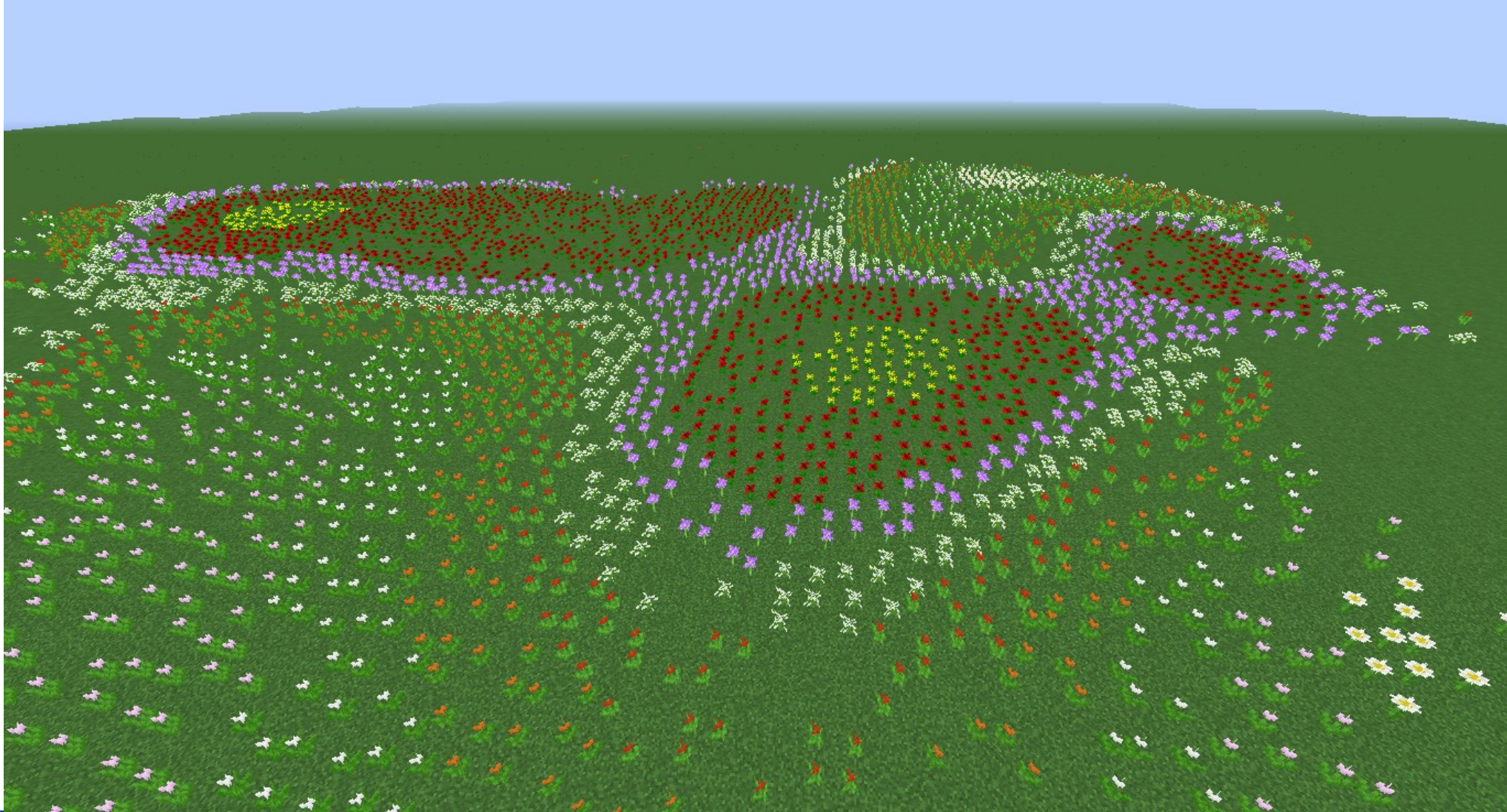


Some games even let you set the random seed value explicitly

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

Many noise functions exist. Perlin noise is a popular one due to its somewhat slow variability. Great for terrain generation!





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();
```

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

// Initialise the random engine

```
default_random_engine engine(randomSeed);
```

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

// 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 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
```

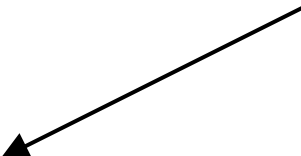
```
default_random_engine engine(randomSeed);
```

```
// 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;  
}
```


Today

- Obnoxious Phone Volcano Coat Quick
- **std::vector and std::array** ←
- Const & constexpr
- The terminal (if there's time)

Only thing that should be left for assignment 2!

From next week we'll be a bit further ahead in the lectures, and there will be 1 lecture per assignment topic :)

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
 - A `std::string` is *almost* exactly a `std::vector<char>`
-
- 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

```
std::vector<int> vec {57, 19,  
                    28, 89, 65, 68, 49};
```

```
vec.push_back(-43);
```

```
vec.at(0) -= 15;
```

```
vec.at(5)++;
```

```
vec.at(2) = vec.at(7);
```

```
vec.push_back(vec.size());
```

```
vec.resize(5);
```

```
vec.resize(8);
```

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

57	19	28	89	65	68	49
----	----	----	----	----	----	----

57	19	28	89	65	68	49	-43
----	----	----	----	----	----	----	------------

42	19	28	89	65	68	49	-43
-----------	----	----	----	----	----	----	-----

42	19	28	89	65	69	49	-43
----	----	----	----	----	-----------	----	-----

42	19	-43	89	65	69	49	-43
----	----	------------	----	----	----	----	-----

42	19	-43	89	65	69	49	-43	8
----	----	-----	----	----	----	----	-----	----------

42	19	-43	89	65
----	----	-----	----	----

42	19	-43	89	65	0	0	0
----	----	-----	----	----	---	---	---

Using arrays

If you do not specify all values, they will be uninitialised.



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

-65	86	10	63	??
-----	----	----	----	----

```
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
```


Demonstration: `std::vector`

Vectors and Arrays: cout

- Vectors cannot be used with cout

```
vector<int> numbers = {1, 2, 3, 4, 5};  
std::cout << numbers << std::endl; // Error!
```

- But they can be used with println()

```
// Prints: Contents: [1, 2, 3, 4, 5]  
std::println("Contents: {}", numbers);
```


Task

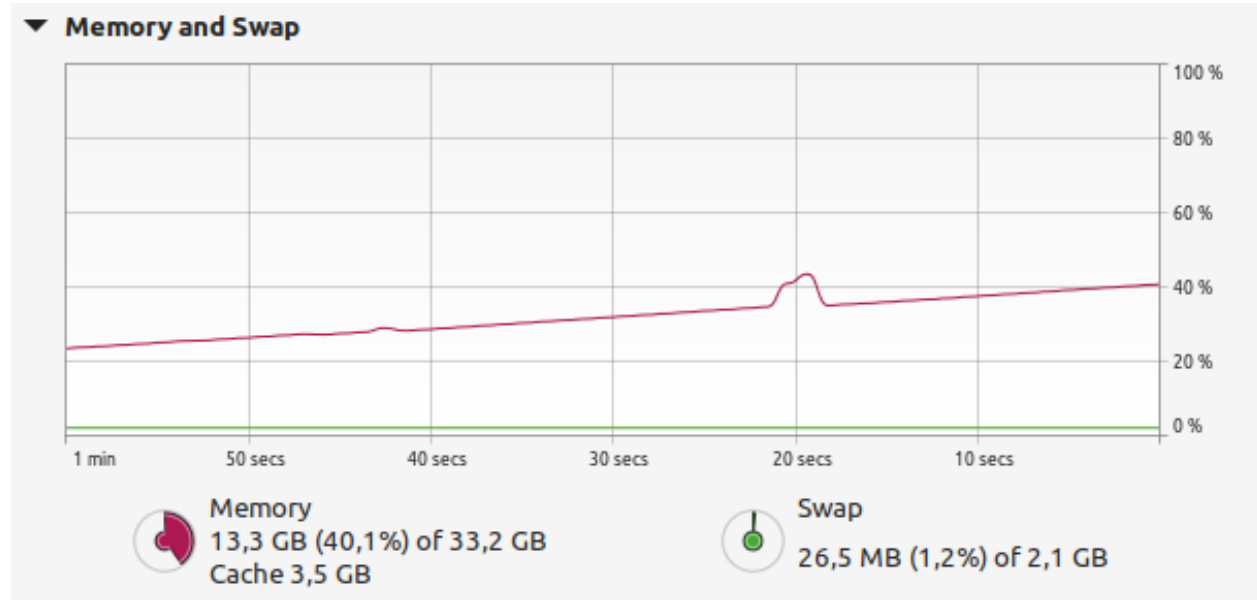
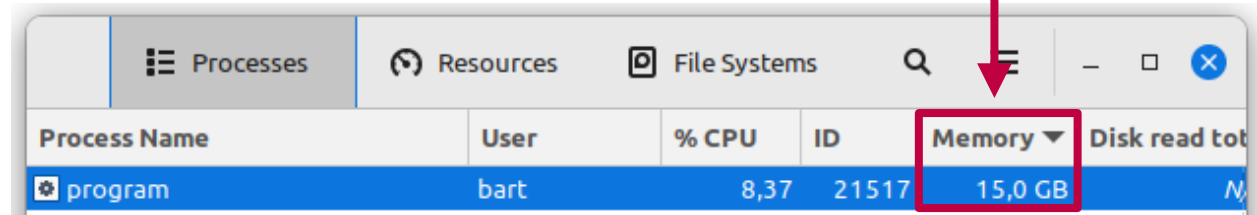
- Setup
 - In VS Code, open a new folder using File > Open folder...
 - Use Create New Project > Lectures > Lecture 04 > Task – vector
 - Once Meson finishes, compile the project
- Objective:
 - In the main() function, write some code that reverses the vector of integers.
 - Challenge / exam preparation: write a function that sorts the list of numbers
- Come see me if you run into any technical problems :)

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


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

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

```
std::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:

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

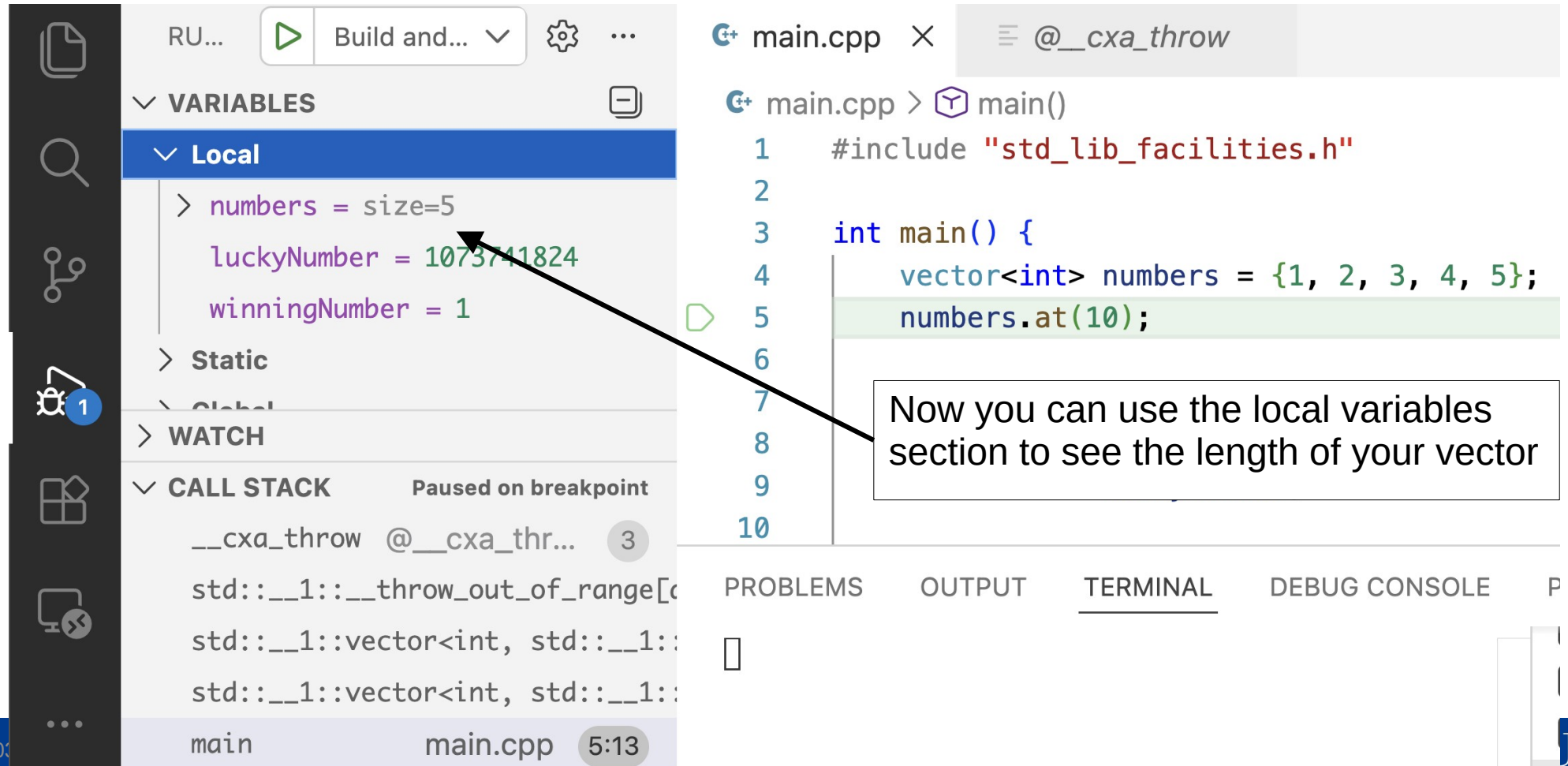

You *really* shouldn't use the [] operator

- 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};  
std::cout << vec[5] << std::endl;  
std::cout << vec.at(5) << std::endl; // Does not compile!
```

We'll talk more about these kinds of arrays in a later lecture

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



The image shows a C++ IDE with a debugger window on the left and a code editor on the right. The debugger window is paused on a breakpoint, and the 'LOCALS' pane shows the following variables:

- `numbers` = size=5
- `luckyNumber` = 1073741824
- `winningNumber` = 1

An arrow points from the `numbers` variable in the LOCALS pane to the `numbers.at(10);` line in the code editor. A text box explains that the LOCALS section can be used to see the length of the vector.

The code editor shows the following code:

```
main.cpp > main()
1  #include "std_lib_facilities.h"
2
3  int main() {
4      vector<int> numbers = {1, 2, 3, 4, 5};
5      numbers.at(10);
6
7
8
9
10
```

Now you can use the local variables section to see the length of your vector

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

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

- Here is what an out of bounds error looks like:

The screenshot shows a debugger interface with a call stack on the left and a memory dump on the right. The call stack is paused on a breakpoint in the function `__cxa_throw` at line 3. The memory dump shows a sequence of memory addresses and their corresponding values. A callout box points to the entry for `main` in the call stack, indicating that clicking through the mess to a call stack entry that is from one of your functions!

Call Stack:

- `__cxa_throw` `@__cxa_thr...` 3
- `std::__1::__throw_out_of_range[c`
- `std::__1::vector<int, std::__1::`
- `std::__1::vector<int, std::__1::`
- `main` `main.cpp` 5:13
- `start` `@start` 1522

Memory Dump:

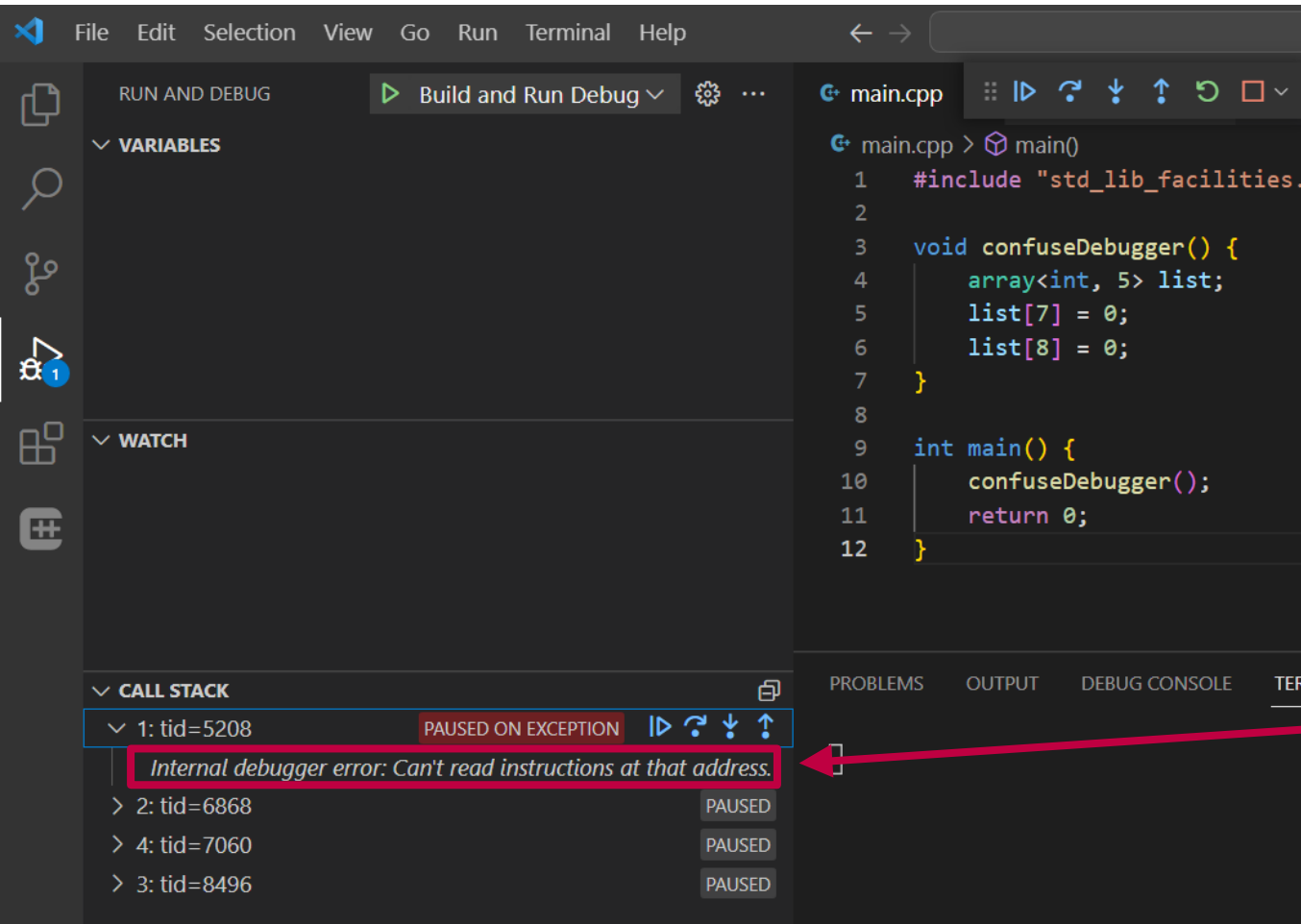
- 1 ; Symbol: `__cxa_throw`
- 2 ; Source: unknown
- 3 `187B4FB7C:` 7F 23 03 D5
- 4 `187B4FB80:` F6 57 BD A9
- 5 `187B4FB84:` F4 4F 01 A9
- 6 `187B4FB88:` FD 7B 02 A9
- 7 `187B4FB8C:` FD 83 00 91
- 8 `187B4FB90:` F3 03 02 AA
- 9 Click through the mess to a call stack entry that is from one of your functions!
- 10
- 11 `187B4FB9C:` 61 F2 FF 97

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;  
    std::array<int, 5> list;  
    list[5] = 5; // Use -1 on MacOS  
    std::cout << value << std::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..

CVE

<

Today

- Obnoxious Phone Volcano Coat Quick
- `std::vector` and `std::array`
- **Const & constexpr**
- The terminal (if there's time)

Const

- Used for constants
- The **const** keyword is added in front of the data type
- Const variables can only be assigned a value **once**, and the variable becomes read-only after it is initialised

```
const int count = 10;  
count++; // error  
std::cout << count << endl; // ok!
```

- A **const** value can be still be copied to another variable

```
int copyOfCount = count;  
copyOfCount++; // ok!
```


Const

- Const also works on more complex data types like `std::string` and `std::vector`

```
const std::string constantMessage = "Hello there";
```

expression must be a modifiable lvalue C/C++(137)

```
const std::__1::string constantMessage
```

[View Problem \(⌘F8\)](#) [Quick Fix... \(⌘.\)](#)

```
constantMessage.at(4) = 'l';
```


Const

- Const will be making appearances over the course of the course (heh)
 - Usually used to communicate that something is not intended to be modified
- Const also allows the compiler to apply specific optimisations

Constexpr

- A more advanced version of const
- Indicates that a value should be computed by the compiler while it is compiling your program

```
constexpr int precomputed = 15 + 16;
```

```
constexpr double estimatePi() {  
    double pi = 0;  
    int sign = 1;  
    for(int i = 1; i < 50000; i++) {  
        pi += 4.0 / (((2 * i) - 1) * sign);  
        sign *= -1;  
    }  
    return pi;  
}  
constexpr double pi = estimatePi();
```


Constexpr

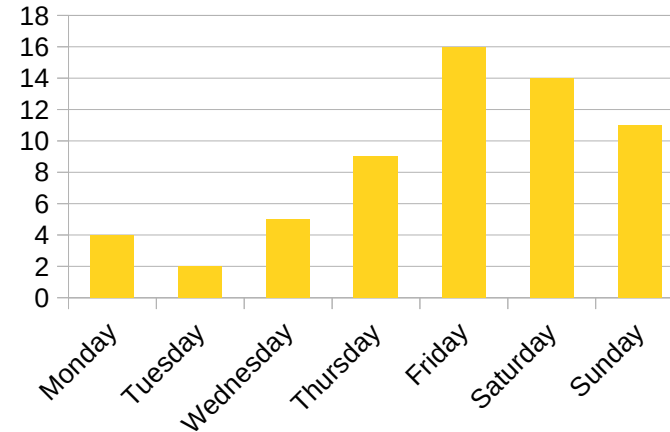
- Compilers are *encouraged*, but not *required*, to perform computations marked with **constexpr** at compile time
- Conversely, per the C++ standard, a compiler is not necessarily allowed to run functions at compile time *unless* they are marked as **constexpr**
- Not all functions are supported:
 - Only literal data types as parameters (mostly numbers)
 - Only literal data types as return type
 - Only short functions
 - Compiler complains if it takes too long

Task: which data type?

Apples
Bread
Crisps

1) Shopping list

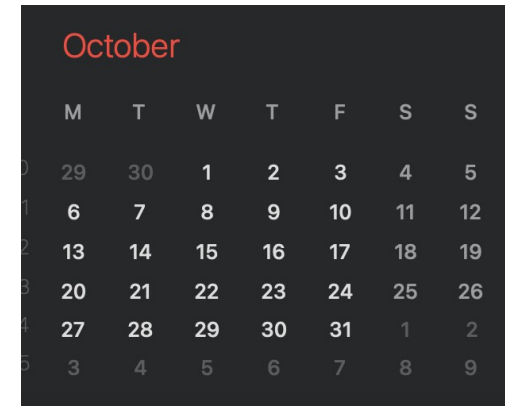
2) Histogram



3) Tier list

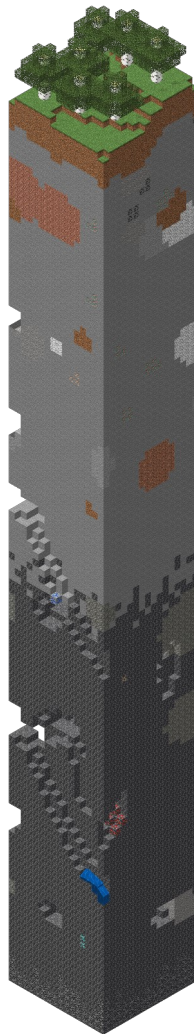
4) A chunk in Minecraft

5) A calendar with a list of appointments for each day



S		
A		
B		
C		
D		
E		
F		

Pick a data type that can store the data of each of the above



Today

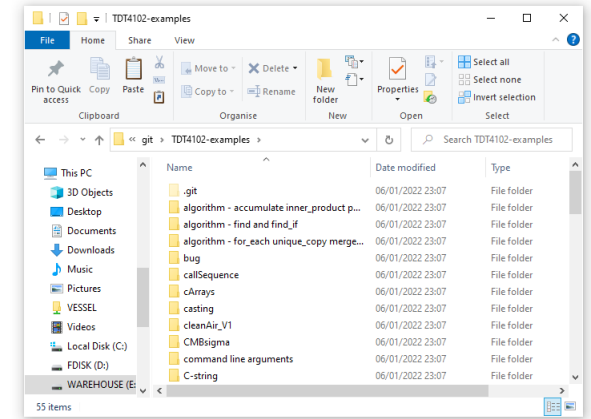
- Obnoxious Phone Volcano Coat Quick
- `std::vector` and `std::array`
- `Const` & `constexpr`
- **The terminal (if there's time)**

The terminal

- A text-based interface to your computer, much like the graphical interface you use every day
- Terminals run one command at a time, then wait for you to put in a new one
- In contrast to the graphical interface, you can run programs with parameters that affect their behaviour

The terminal

- Terminals have a «current directory», like File Explorer on Windows or Finder in MacOS
- Any files you use in commands will be relative to the current directory



Relative file paths

A relative file path specifies the location of a file or directory relative to the current directory

Syntax:

- Directories are separated using a /
Example: `builddir/meson-logs/meson-log.txt`
- A period (.) means “this directory”
Example: `./builddir/program.exe`
- A double period (..) means “the directory above”
Example: `../../main.cpp`

The terminal: current directory

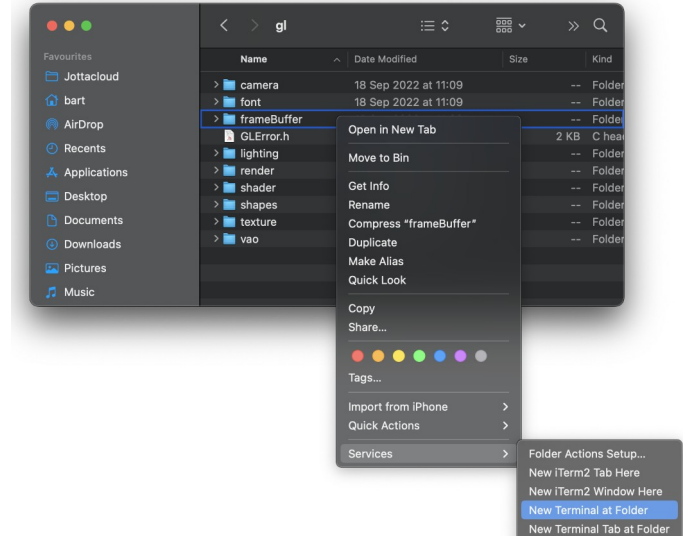
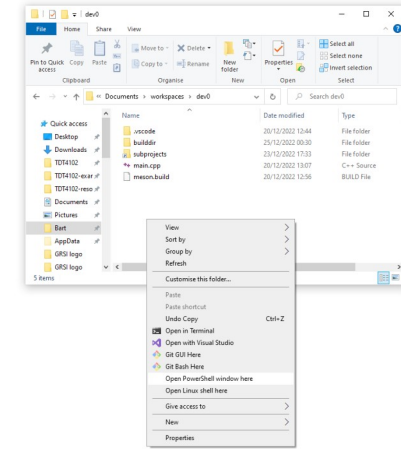
- List the contents of the current directory:
`ls`
- Change the current directory:
`cd [directory]`
- Show the path of the current directory:
`echo "${PWD}"`

The terminal: files and directories

- Create a new directory:
`mkdir [directory name]`
- Delete a directory (and its contents):
Windows: `rmdir [directory name]`
MacOS / Linux: `rm -rf [directory name]`
- Create a new file:
Windows: `New-Item -ItemType file [filename]`
MacOS / Linux: `touch [filename]`
- Delete a file:
`rm [filename]`

Open a terminal at a folder

- Windows:
 - Hold shift
 - Right click in an empty part of a folder
 - Select “open PowerShell window here” or “open Terminal window here”
- Mac:
 - Right click on a folder
 - Select services
 - Select “New Terminal at Folder”



Compiling a C++ program

The first thing we need is a C++ source file.

We'll use the basic one for now:

```
#include "std_lib_facilities.h"

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

Next we need a compiler!

There are many compilers..

- LLVM / Clang
 - Used in the course
 - LLVM is a set of tools to build compilers
- GCC (GNU Compiler Collection)
 - Developed by the Free Software Foundation
- MSVC (Microsoft Visual C++)
 - Microsoft's C++ compiler
 - Only available on Windows
- icc (Intel C++ compiler)

And a LOT more!

But how do they differ?

- Implement C++ (and its standard library) in a slightly different way
- Optimises your code in different ways
- Sometimes have specific language extensions or features
- Some tools only work with a specific compiler
E.g. clang-tidy

Compiling a program

Windows:

```
clang++ main.cpp -o program.exe
```

MacOS / Linux:

`clang++ main.cpp -o program`

The diagram illustrates the components of the compilation command `clang++ main.cpp -o program` for MacOS and Linux. It features three arrows pointing to specific parts of the command with descriptive text:

- An arrow points from the text "The compiler" to `clang++`.
- An arrow points from the text "Name of the C++ file you want to compile" to `main.cpp`.
- An arrow points from the text "Name of the executable file" to `program`.
- An additional arrow points from the text "Indicates that the filename of the executable follows" to the `-o` flag.

We need to make some changes..

```
#include "std_lib_facilities.h"

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

```
main.cpp:1:10: fatal error:
'std_lib_facilities.h' file not found
#include "std_lib_facilities.h"
      ^~~~~~
1 error generated.
```


Now the program compiles!

```
#include <iostream>
```

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

```
bart@TURBONINJA:~/workspace$ clang++ main.cpp -o program  
bart@TURBONINJA:~/workspace$
```


Running a program

Windows:

```
./program.exe
```

MacOS / Linux:

```
./program
```

```
bart@TURBONINJA:~/workspace$ ./program  
Hello there!  
bart@TURBONINJA:~/workspace$
```


We are vectorious!

And done for today :)

Next time: how to create projects with multiple
.cpp source files!