

# INTRODUCTION TO C++

Dr. Alex Hill

[a.d.hill@liverpool.ac.uk](mailto:a.d.hill@liverpool.ac.uk)

October 2022



# RECAP

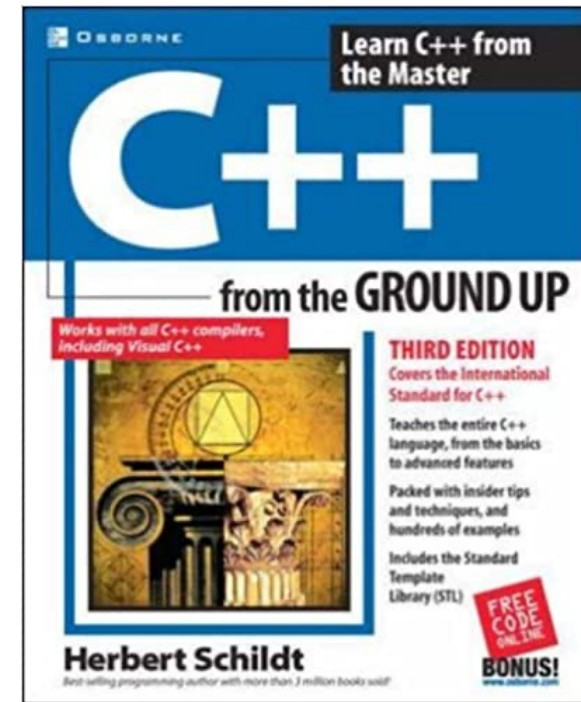


```
test.cpp
Get Started test.cpp x
Users > alexhill > Documents > UOL > Teaching > C++_Workshops > test.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6      int first_number, second_number, sum;
7
8      cout << "Enter two integers: ";
9      cin >> first_number >> second_number;
10
11     // sum of two numbers is stored in variable sumOfTwoNumbers
12     sum = first_number + second_number;
13     // prints sum
14     cout << first_number << " + " << second_number << " = " << sum;
15
16     cout << p;
17     return 0;
18 }
```



# RESOURCES

- [alex-hill94.github.io/#WS2](https://alex-hill94.github.io/#WS2)
- C++ from the ground up (roughly chapter two to five)
- [https://www.w3schools.com/cpp/cpp\\_variables.asp](https://www.w3schools.com/cpp/cpp_variables.asp)
- <https://www.programiz.com/cpp-programming/online-compiler/?ref=1a2efafc>





# AIM OF WORKSHOP TWO

- Variables and data types
- Functions
- For-loops
- Arrays and vectors





# FOLLOW ALONG ON YOUR LAPTOP

- Copy the text on the Powerpoint into your IDE
- Compile and run to assert that you get the same results

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

int main() {
    string a = "Interactive lessons are superior!";
    cout << a << endl;
    return 0;
}
```

```
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ g++ -o run test.cpp
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ ./run
Interactive lessons are superior!
```



# VARIABLES

- Different data types
- Type conversion
- Precision and limits
- Simple examples and operations list





# VARIABLES

- A variable is a location in memory space which may be named
- Variables are assigned values, which may be changed at any time
- In C++, you must tell the compiler what data type to expect for a variable





# VARIABLES

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

int main() {
    int a = 10;
    int b;
    b = 11;
    cout << a << " " << b;
    return 0;
}
```

Variables may be assigned values straight away, or later in the code



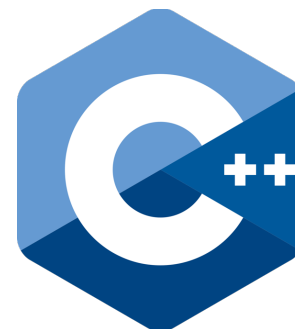


# DATA TYPES



Name	Description
<i>int</i>	Stores integers without decimals (e.g. 0, 1, 2...)
<i>double</i>	Stores floating point numbers without decimals (e.g. 1.21)
<i>char</i>	Stores single characters, which are loaded using single quotations ('a', 'b')
<i>string</i>	Stores text, loaded using double quotations ("Hello")
<i>bool</i>	Stores Boolean values: true, false
<i>float</i>	Stores floating point numbers without decimals (e.g. 1.21F)
<i>others</i>	There are more data types, and you can create your own





# DOUBLES AND FLOATS

Floats	Doubles
Size: 4 bytes	Size: 8 bytes
7 decimal places	15 decimal places
17.0F	17.0
Used occasionally to speed up processes	Used most of the time



# COMPILING DATA TYPES



- The compiler will try to convert the value inputted to the chosen data type
- If there's an apparent discrepancy, warnings can be arise

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

int main() {
    int a = 1.5;
    cout << a << endl;
    return 0;
}
```

```
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ g++ -o run test.cpp
test.cpp:5:10: warning: implicit conversion from 'double' to 'int' changes
value from 1.5 to 1 [-Wliteral-conversion]
        int a = 1.5;
              ~   ^~~
1 warning generated.
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ ./run
1
```

# COMPILING DATA TYPES



- Sometimes there will be no warnings, or unintended consequences
- Be careful!

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

int main() {
    char a = 1;
    cout << a << endl;
    a = '%'
    cout << a << endl;
    return 0;
}
```

```
(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ g++ -o run test.cpp
(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ ./run

%
```





# TYPE CONVERSION

- Variables can have their data type changed implicitly or explicitly
- An example of implicit conversion: the double value is automatically converted to int

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

int main() {

    // assigning a double value to num_double
    double num_double = 9.1;
    // declaring an int variable
    int num_int;
    // implicit conversion
    // assigning double value to a int variable
    num_int = num_double;

    cout << "num_double = " << num_double << endl;
    cout << "num_int = " << num_int << endl;

    return 0;
}
```

```
$ ./run
num_double = 9.1
num_int = 9
```





# TYPE CONVERSION

- Variables can have their data type changed implicitly or explicitly
- An example of explicit conversion: the double value is automatically converted to int

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

int main() {

    double num_double = 9.1;
    int num_int;

    // explicit conversion

    num_int = int(num_double);

    cout << "num_double = " << num_double << endl;
    cout << "num_int = " << num_int << endl;

    return 0;
}
```

```
$ ./run
num_int = 9
num_double = 9.1
```





# TYPE CONVERSION

- Note that what we're doing here is converting a value, not the data type identifier in memory

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

int main() {

    double num_double = 9.1;
    double num_int;
    int num_int1;

    // explicit conversion

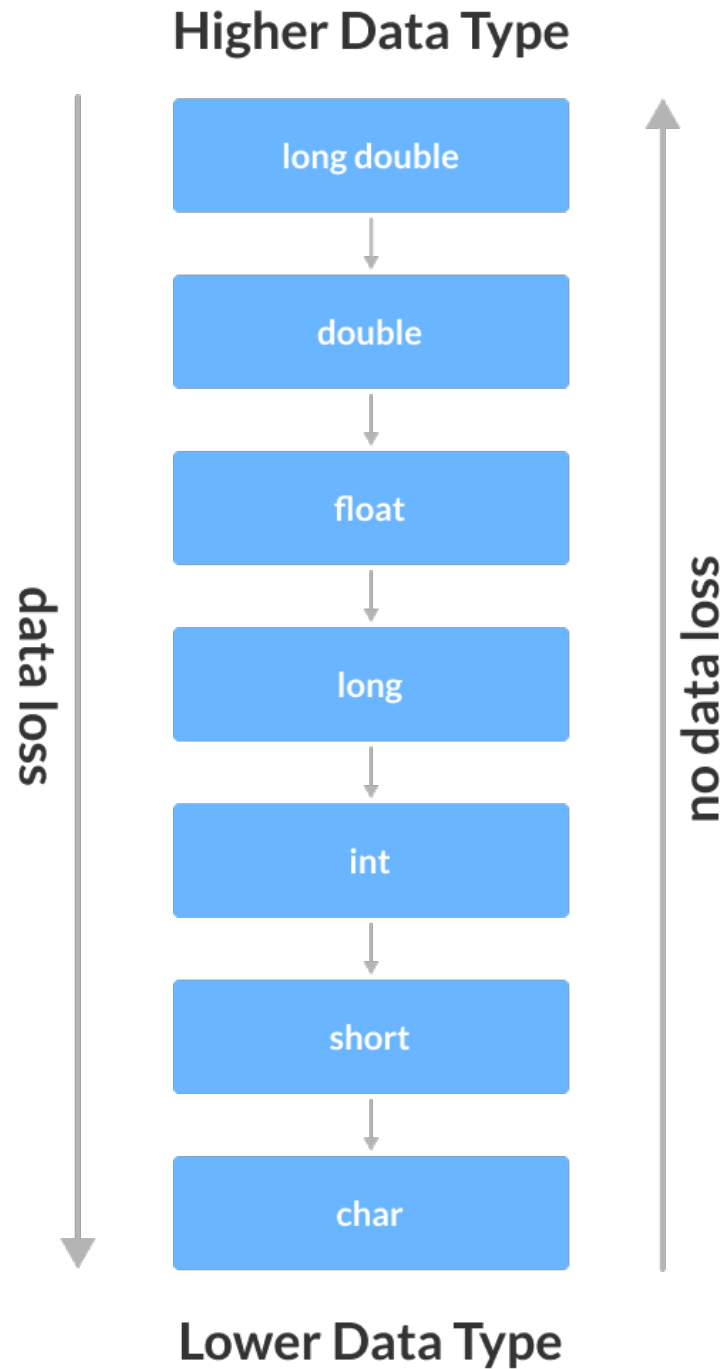
    num_int = int(num_double);
    num_int1 = int(num_double);

    cout << "num_double = " << num_double << endl;
    cout << typeid(num_double).name() << endl;
    cout << "num_int = " << num_int << endl;
    cout << typeid(num_int).name() << endl;
    cout << "num_int1 = " << num_int1 << endl;
    cout << typeid(num_int1).name() << endl;

    return 0;
}
```

```
$ ./run
num_double = 9.1
d
num_int = 9
d
num_int1 = 9
i
```





Credit:  
[www.programiz.com/cpp-programming/type-conversion](http://www.programiz.com/cpp-programming/type-conversion)







# PRECISION: DOUBLE

- There is an inbuilt precision for `cout`

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

int main() {
    double a = 1.123456789;
    cout << a << endl;
    return 0;
}
```

```
$ ./run
1.12346
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Worksh
ops/WS2
$
```





# PRECISION: DOUBLE

- You can set the precision of `cout` that you need using the `setprecision` function

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

int main() {
    double a = 1.123456789;
    cout << setprecision(10);
    cout << a << endl;
    return 0;
}
```

```
$ ./run
1.123456789
```





# PRECISION: DOUBLE

- Note that if you set the precision beyond the capacity of the data type, you get (deterministic) junk after a certain point

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

int main() {
    double a =
    1.1234567891234567891234;
    cout << setprecision(20);
    cout << a << endl;
    return 0;
}
```

```
$ ./run
1.1234567891234568116
```



# LIMITS



- The data types have a max and min value depending on the number of bits they use in memory
- For *int*, this is 2147483647

```
#include <iostream>
using namespace std;
int main() {
int a = 2147483647;
int b = 2147483648;
cout << a << endl;
cout << b << endl;
}
```

```
$ g++ -o run test.cpp
test.cpp:5:13: warning: implicit conversion from
'long' to 'int' changes value from 2147483648 to
-2147483648 [-Wconstant-conversion]
    int b = 2147483648;
           ~~~~~^~~~~~
1 warning generated.
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops
/WS2
$ ./run
2147483647
-2147483648
```



# LIMITS



- If you need extra decimal places, you can use data types like `long int`, which uses more bits
- <https://learn.microsoft.com/en-us/cpp/c-language/cpp-integer-limits?view=msvc-l70>

```
#include <iostream>
using namespace std;
int main() {
int a = 2147483647;
long int b = 2147483648;
cout << a << endl;
cout << b << endl;
}
```

```
$ g++ -o run test.cpp
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ ./run
2147483647
2147483648
```





# LIMITS

- You can check how many bits a data type uses with the `sizeof()` function

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

int main() {
    cout << "int:" << sizeof(int) << endl;
    cout << "float:" << sizeof(float) << endl;
    cout << "double:" << sizeof(double) << endl;
    cout << "long int:" << sizeof(long int) <<
    endl;
}
```

```
$ ./run
int:4
float:4
double:8
long int:8
```



# SIMPLE BOOLEAN EXAMPLE



- The Relational operators in C++ are the same as they are in Python (==, !=)
- To check the value of a variable, you must first create a Boolean variable
- 1 = true, 0 = false
- You can force cout to return 'true' and 'false' using `cout << boolalpha`

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

int main() {
    int a = 10;
    bool b;
    bool c;
    b = a == 10;
    c = a == 11;
    cout << b << endl;
    cout << c << endl;
}
```

```
$ ./run
1
0
```



# OPERATORS



- The arithmetic operators in C++ are very close to those in Python (+, -, /, \*, %)
- Others (like logical operators) are a bit different
- See <https://www.programiz.com/cpp-programming/operators> for a more complete list





# FUNCTIONS

- Explore how functions are created





# FUNCTIONS

- Functions are the building blocks of C++ programmes
  - A good point to aim for is each function doing only one thing
- C++ does not allow nested functions, however one function can call another
- You can call your function anything except `main()`, which is reserved for the programme execution



# FUNCTIONS

`void` is a null data type, used here as the function returns nothing

'Prototype' here indicates that the function needs to be declared prior to its definition. The compiler needs to know what data type will be returned and what inputs it will take before it's first called



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

void myfunc(); // 'Prototype'

int main() {
    cout << "What is love?" << endl;
    myfunc(); // Call myfunc
    cout << "No more" << endl;
    return 0;
}

void myfunc(){
    cout << "Baby don't hurt me" << endl;
};
```



# CHALLENGE ONE:



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

void myfunc(); // 'Prototype'

int main() {
    cout << "Inside main" << endl;
    myfunc(); // Call myfunc
    cout << "Back inside" << endl;
    return 0;
}

void myfunc(){
    cout << "Inside myfunc" << endl;
};
```

I would like you to tell a knock-knock joke using multiple functions.

The main() function should prompt the user to write “Who’s there?” and “XXXXX who?” into the terminal, while the other functions should tell the other parts of the joke

Best/worst joke wins! Send your scripts to my email address.



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

void knock(); // 'Prototype'

void setup();

void punchline();

int main() {

    string who, there;
    string blank_who;

    knock();
    cin >> who >> there;
    setup(); // Call myfunc
    cin >> blank_who;
    punchline(); // Call myfunc

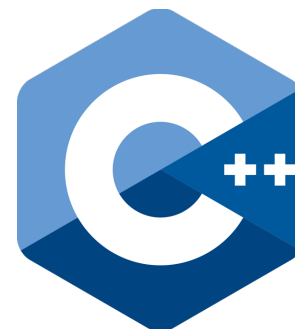
    return 0;
}

void knock(){
    cout << "Knock knock" << endl;
};

void setup(){
    cout << "Beets" << endl;
};

void punchline(){
    cout << "Beets me!" << endl;
};
```





# FUNCTIONS

`int` as my  
function `mul`  
will return an  
integer

`int` as my  
function `mul`  
has integer  
arguments

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

int mul(int val_one, int val_two);

int main() {
    int a;
    a = mul(1, 4);
    cout << a << endl;
    return 0;
}

int mul(int val_one, int val_two){
    return val_one * val_two;
}
```



# CHALLENGE TWO:



```
#include <iostream>
#include <cmath> // Need this for pow()
using namespace std;

int mul(int val_one, int val_two);

int main() {
    int a;
    a = mul(1, 4);
    cout << a << endl;
    return 0;
}

int mul(int val_one, int val_two){
    return val_one * val_two;
}
```

I would like you to compute the below equation using two functions called 'add' and 'divide'

$$(12.12 + 7.01) / (6.352 + 23.4)$$

No arithmetic operators in main()!

If you can do this quickly, write a code that computes:

$$y = mx^2 + c$$

For x specified in the terminal, and m and c defined in the script (`#include <cmath>` // Need this for pow())





```
#include <iostream>
#include <cmath> // Need this for pow()
using namespace std;

double add(double val_one, double val_two);
double divide(double val_one, double val_two);

int main() {
    double a = 12.12;
    double b = 7.01;
    double c = 6.352;
    double d = 23.4;
    double ans;

    ans = divide(add(a, b) , add(c, d) );
    cout << ans << endl;
    return 0;
}

double add(double val_one, double val_two){
    return val_one + val_two;
}

double divide(double val_one, double val_two){
    return val_one/val_two;
}
```





# FOR LOOPS

- Explore the syntax of conditional and ranged for loops





# CONDITIONAL FOR LOOPS

- Introducing for loops in C++

condition

Update (optional)

initialization

```
#include <iostream>
using namespace std;
int main() {
for(int i = 1; i <= 5; ++i) {
cout << i << " ";
}
return 0;
}
```

Block of code within loop

A diagram illustrating the components of a C++ for loop. The code is shown in a dark box with syntax highlighting. Four orange arrows point from text labels to specific parts of the code: 'initialization' points to 'int i = 1', 'condition' points to 'i <= 5', 'Update (optional)' points to '++i', and 'Block of code within loop' points to the block between the curly braces of the for loop.

# CHALLENGE THREE:



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

int main() {
    int num, sum;
    sum = 0;

    cout << "Enter a positive integer: ";
    cin >> num;

    for (int i = 1; i <= num; ++i) {
        sum += i;
    }

    cout << "Sum = " << sum << endl;

    return 0;
}
```

- This code computes the sum of numbers up to num
- Can you adapt this to compute the mean of numbers up to num?





It is necessary to initialize num and sum as float objects to ensure this calculation is float/float

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

int main() {
    double num, sum;
    double mean;
    sum = 0;

    cout << "Enter a positive integer: ";
    cin >> num;

    for (int i = 1; i <= num; ++i) {
        sum += i;
        cout << i << " " << sum << endl;
    }

    cout << "Sum = " << sum << endl;
    cout << "Num = " << num << endl;
    mean = sum/num;
    cout << "Mean = " << mean << endl;

    return 0;
}
```

```
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ ./run
Enter a positive integer: 14
1 1
2 3
3 6
4 10
5 15
6 21
7 28
8 36
9 45
10 55
11 66
12 78
13 91
14 105
Sum = 105
Num = 14
Mean = 7.5
```





# RANGED FOR LOOP: NEW FOR C++11

Array object

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

int main() {
    int num_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    for (int n : num_array) {
        cout << n << " ";
    }

    return 0;
}
```



# RANGED FOR LOOP

Disable warnings with this argument




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

int main() {
    int num_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    for (int n : num_array) {
        cout << n << " ";
    }

    return 0;
}
```



```
$ g++ -std=c++11 -o run test.cpp
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshop
s/Workshops/WS2
$ ./run
1 2 3 4 5 6 7 8 9 10 (base) alexhill
at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshop
s/Workshops/WS2
```



# ARRAYS AND VECTORS

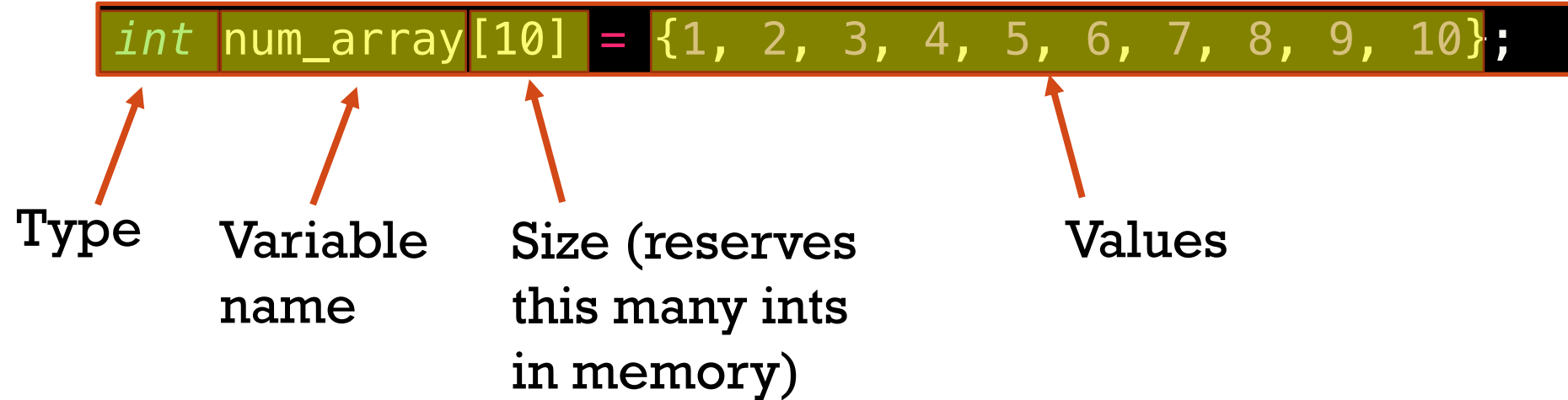
- Explore the difference between arrays and vectors
- Combine all we've learned today to create some more complex programmes





# ARRAYS

- A one-dimensional array is a list of related variables







# ARRAY INDEXING

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

int main() {
    int my_array[7];
    int j;

    for(j = 0; j < 7; j++)
    {
        my_array[j] = j;
        cout << my_array[j] << endl;
    }

    return 0;
}
```

- Arrays consist of contiguous memory locations, the lowest address is the first element etc.
- Elements are indexed similarly to Python (e.g. `my_array[0]`)
- Warning, there's no



# ARRAY INDEXING



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

int main() {
    int crash[10], i;

    for(i = 0; i < 100; i++)
    {
        crash[i] = i;
        cout << crash[i] << endl;
    }

    return 0;
}
```

- Warning, there are no boundary checks
- Here the loop iterates 100 times, even though crash is only 10 elements long!
- This will cause important information to be overwritten





# VECTORS

- Vectors are like arrays, but can grow dynamically

```
#include <vector>

...

vector<int> my_vector; // initialise vector
```

Call vector object

Data type

Name of vector





# VECTOR INITIALISATION

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

int main() {

    // initialiser list
    vector<int> vector1 = {1, 2, 3, 4, 5};

    // uniform initialisation
    vector<int> vector2{6, 7, 8, 9, 10};

    // method 3
    vector<int> vector3(5, 12);

    for (int i: vector3)
        std::cout << i << ' ';

    return 0;
}
```

vector1 and vector1 are initialised with set values

vector3 creates an array of length five, consisting of repeating twelves

You can't print out a full vector, you need to loop over all the elements





# VECTOR MANIPULATION

```
vector<int> vector1 = {1, 2, 3, 4, 5};  
// add the integers 6 and 7 to the vector  
vector.push_back(6);  
vector.push_back(7);  
// remove the last element  
num.pop_back(7);
```

```
// access vector elements  
vector.at(0);  
// or  
vector[0];  
// However, the at() function  
// is preferred over [] because  
// at() throws an exception  
// whenever the vector is out of  
// bound, while [] gives a garbage value.
```

```
// change elements at indexes 1  
and 4  
num.at(1) = 9;  
num.at(4) = 7;
```

For other vector functions, see:  
<https://www.programiz.com/cpp-programming/vectors>



# CHALLENGE FOUR



- Create an evenly-spaced array between 0 and  $\pi$  (you'll need to import `<cmath>`)
- Create a function called `sin_2x` which returns `sin(2x)`
- Loop over your array and pass the elements to `sin_2x`
- Save the results to a new array of the same length

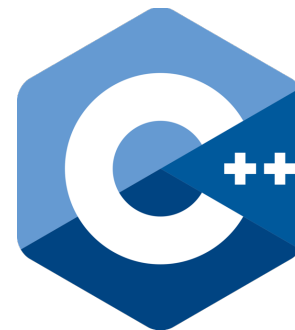




# NEXT WEEK

- Passing vectors into functions
- Plotting data (really this time)
- Introduction to Monte Carlo methods





**THANKS!**

