October 2023

Introduction to C++: Workshop Two

Dr. Alexander Hill

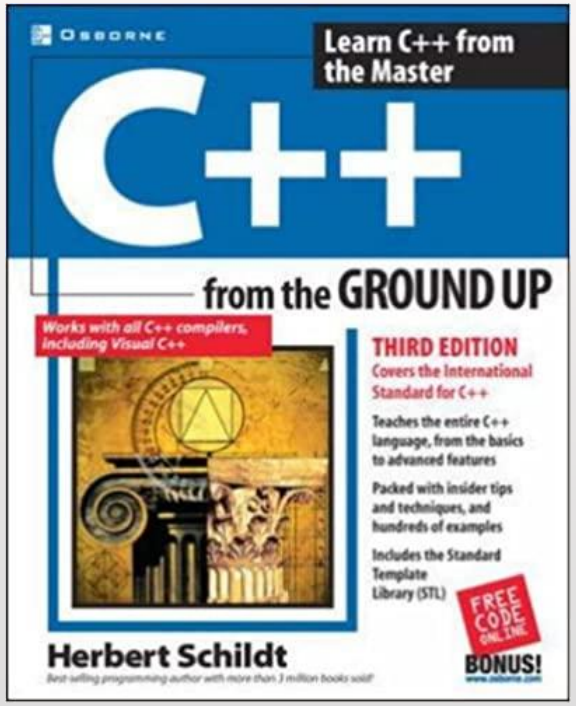
a.d.hill@liverpool.ac.uk

Recap

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

int main() {
int first_number, second_number, sum;

cout << "Enter two integers: ";
cin >> first_number >> second_number;

// sum of two numbers in stored in variable sumOfTwoNumbers
sum = first_number + second_number;

// prints sum
cout << first_number << " + " << second_number << " = " << sum << "\n";

return 0;
}
```

Resources

- alex-hill94.github.io/#WS2
- C++ from the ground up, Herbert Schildt (Roughly Chapters 2-5)
- Online compiler: https://www.programiz.com/cpp-programming/online-compiler/
- https://www.w3schools.com/cpp/cpp_variables.asp


OSBORNE
Learn C++ from the Master
C++
from the GROUND UP
Works with all C++ compilers, including Visual C++
THIRD EDITION
Covers the International Standard for C++
Teaches the entire C++ language, from the basics to advanced features
Packed with insider tips and techniques, and hundreds of examples
Includes the Standard Template Library (STL)
FREE CODE ONLINE
BONUS!
Herbert Schildt
Best-selling programming author with more than 3 million books sold!

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 (find slides on my site)
- 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 exercises and operations list

Variables

- A variable is a container for data

- A named location in memory space

- 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

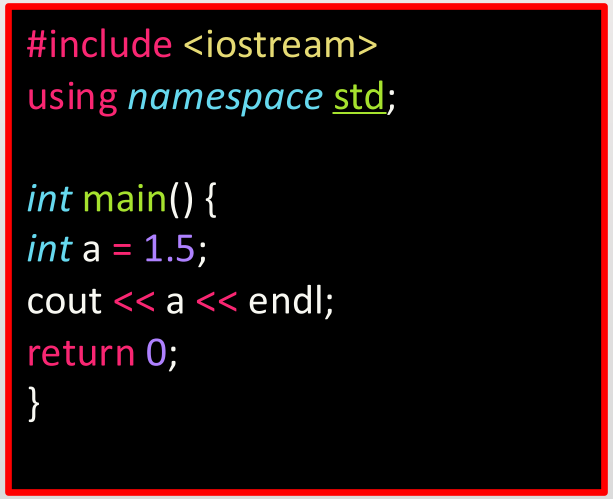
Name	Description
int	Stores integers without decimals (e.g. 0, 1, 2…)
double	Stores floating point numbers without decimals (e.g. 1.21)
char	Stores single characters, which are loaded using single quotations ('a’,’b’)
string	Stores text, loaded using double quotations (“Hello”)
bool	Stores Boolean values: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

Doubles and Floats

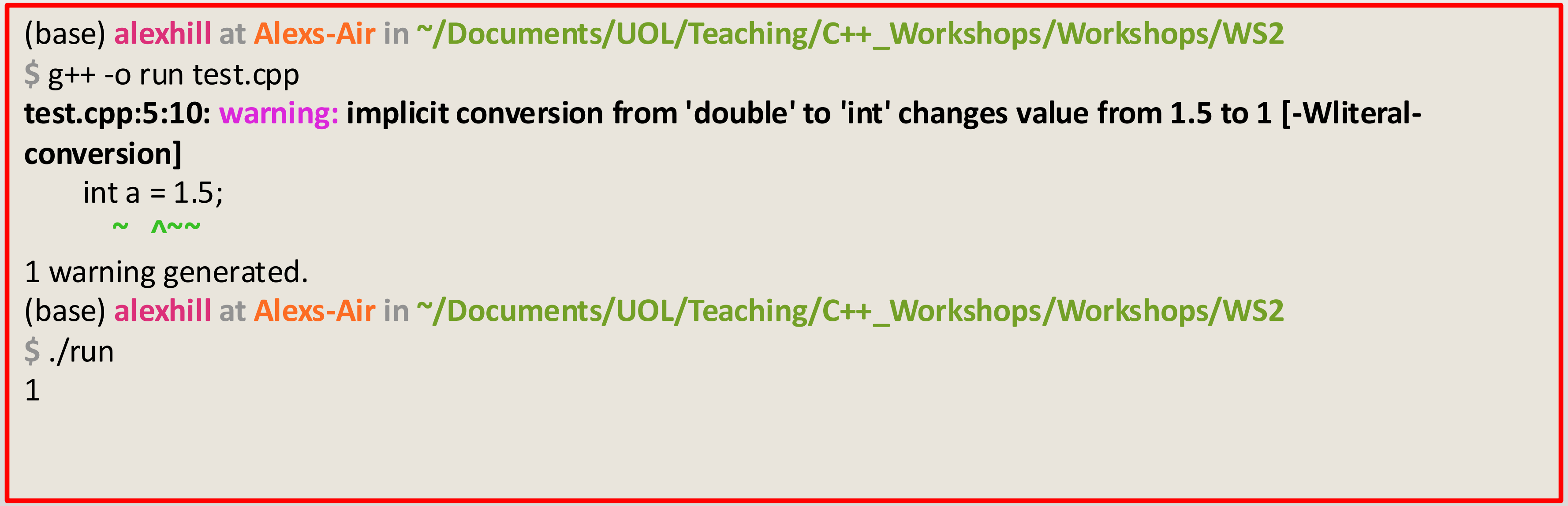
Floats	Doubles
Size: 4 bytes	Size: 8 bytes
7 decimal places	15 decimal places
17.0F	17.0
Used occasionally to speed up processes	Used most of the time

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 arise



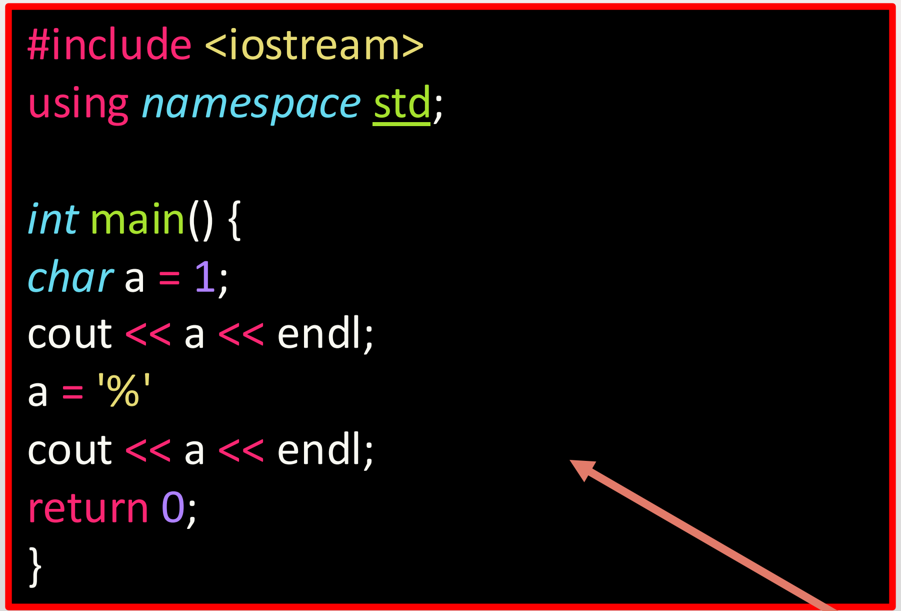
```
#include <iostream>
using namespace std;

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
```

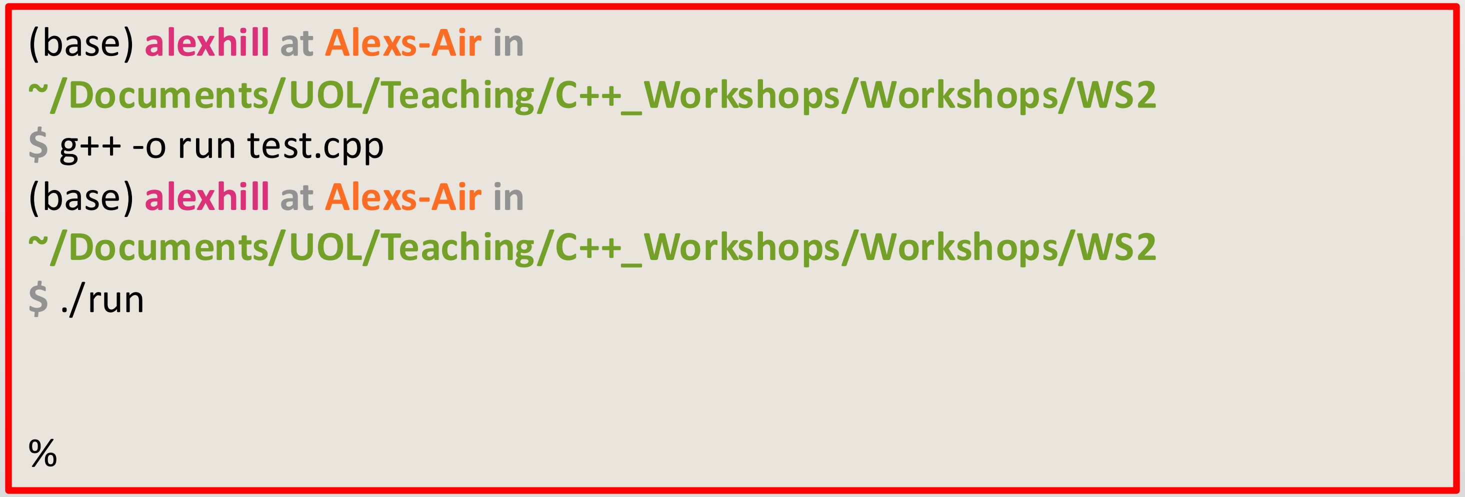
Compiling Data Types

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



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

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

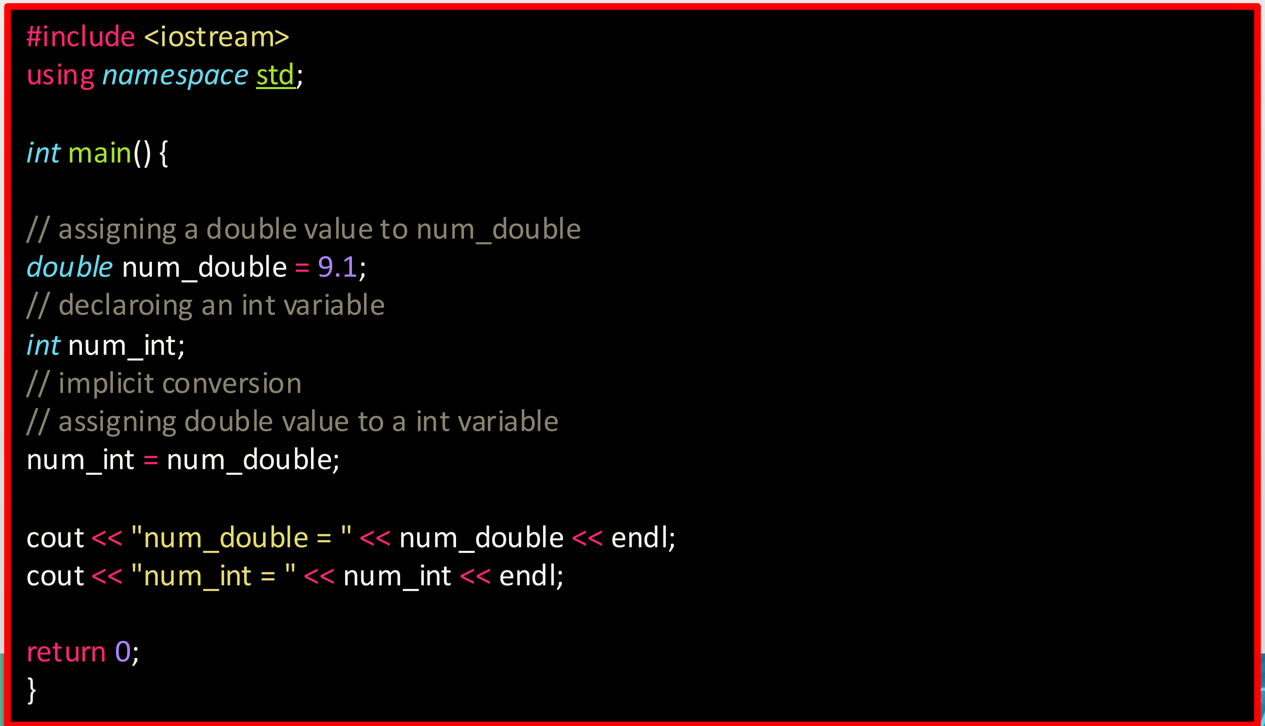
‘End line’ function



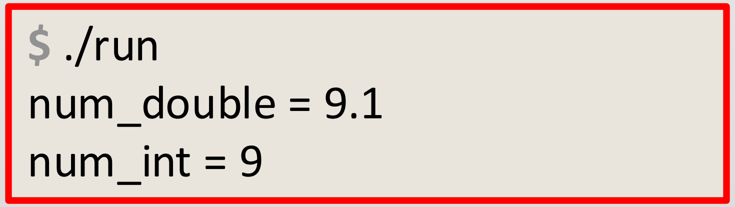
```
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ g++ -o run test.cpp
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/WS2
$ ./run


%
```

Type Conversion

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



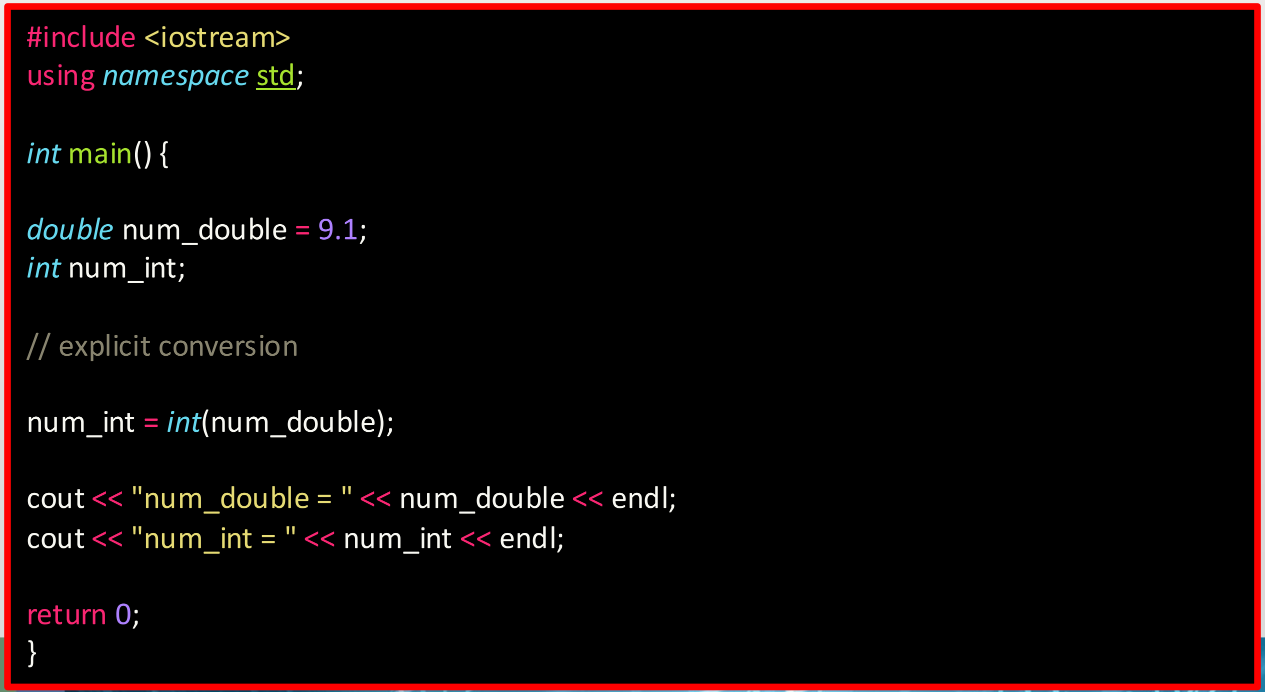
```
#include <iostream>
using namespace std;

int main() {

// assigning a double value to num_double
double num_double = 9.1;
// declaroing 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;
}
```



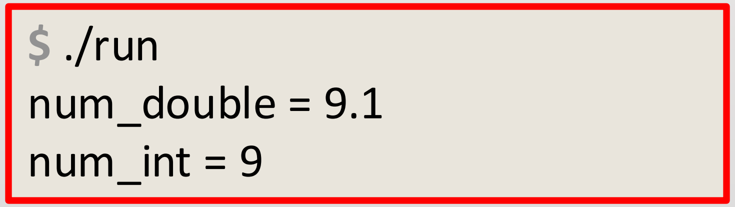
```
$ ./run
num_double = 9.1
num_int = 9
```

Type Conversion

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



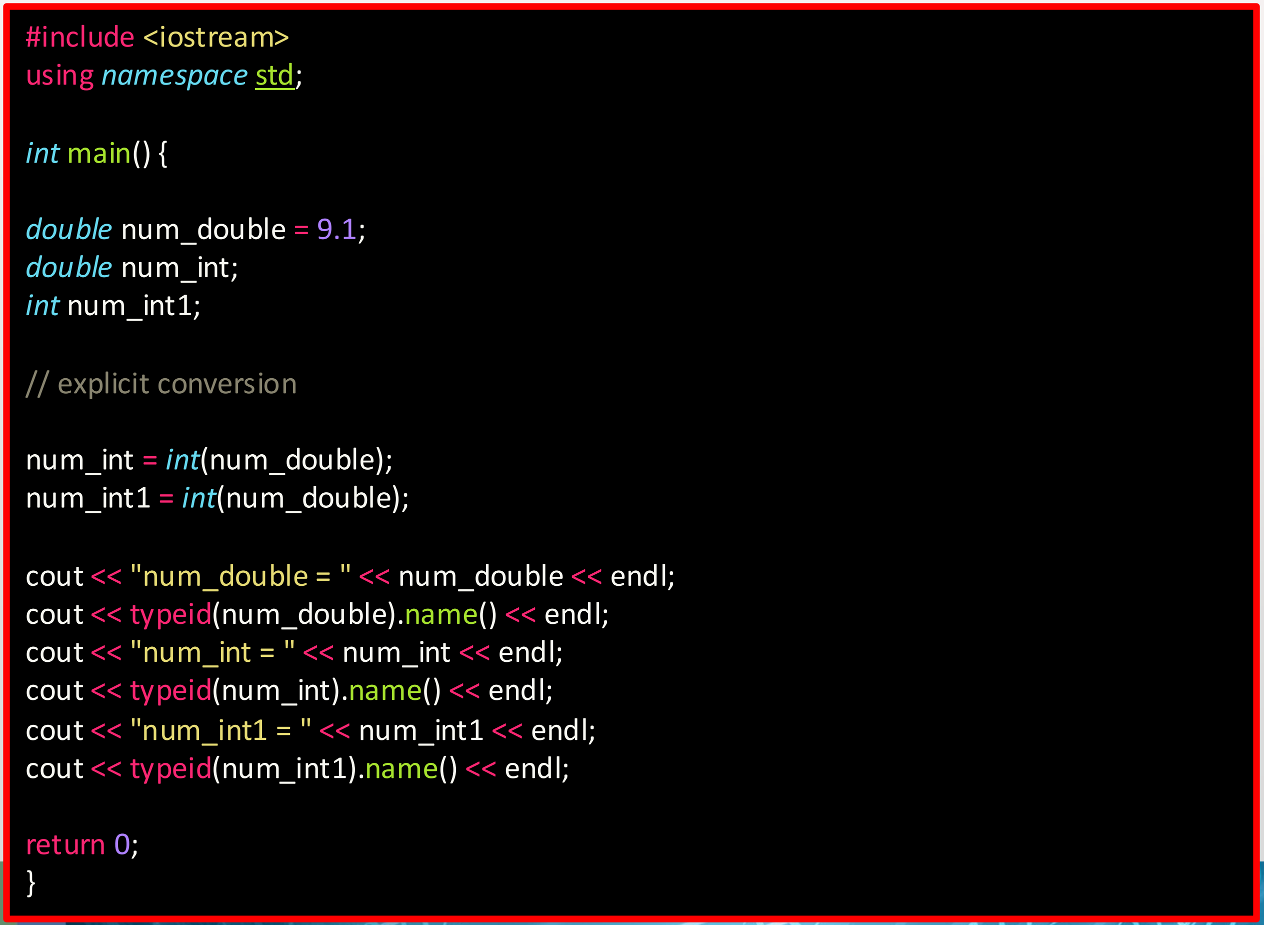
```
#include <iostream>
using namespace std;

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;
}
```



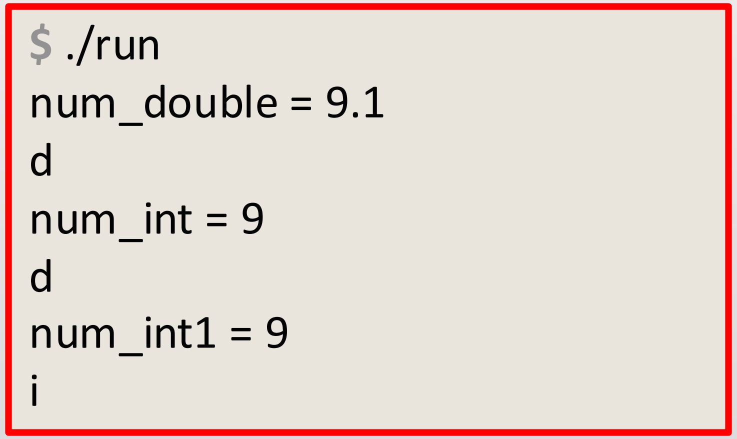
```
$ ./run
num_double = 9.1
num_int = 9
```

Type Conversion

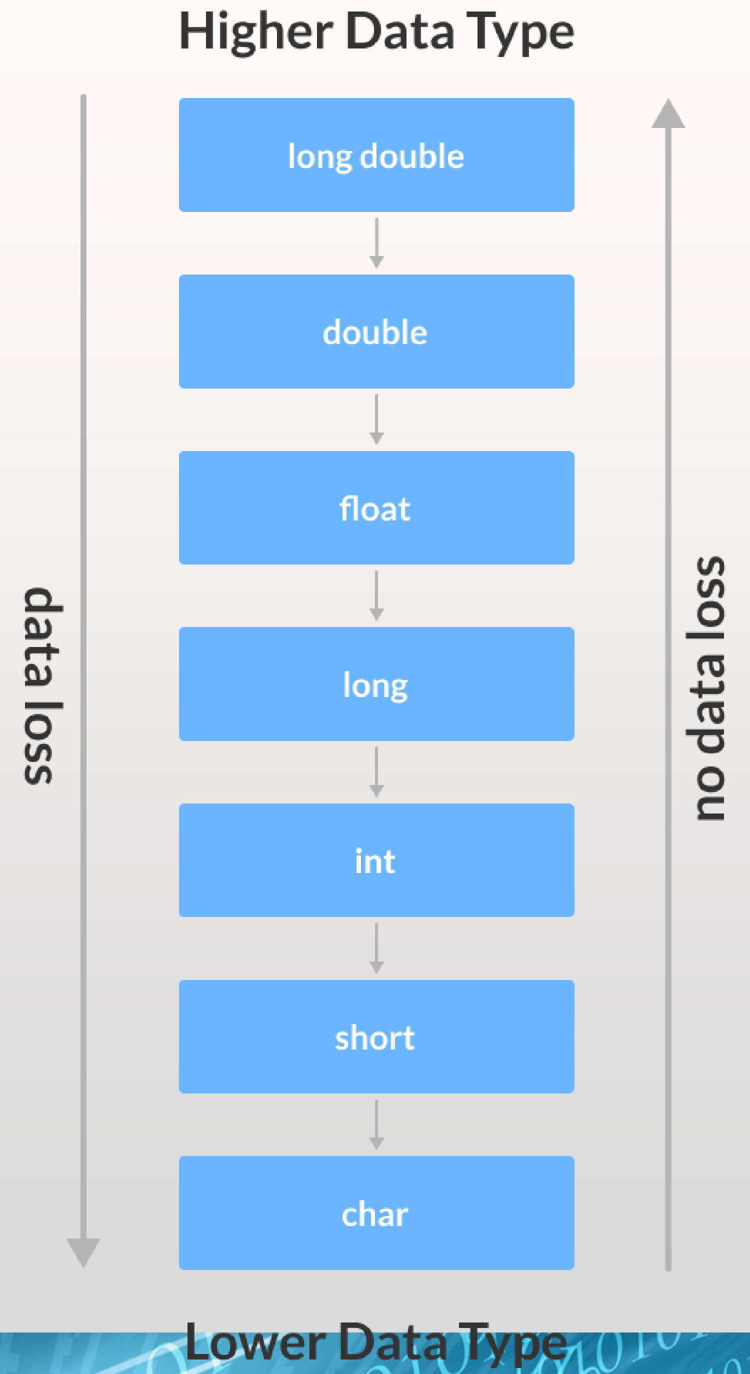
- Note that what we're doing here is converting a value, not the data type in memory



```
#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
```



Higher Data Type

long double

double

float

long

int

short

char

data loss

no data loss

Lower Data Type

Credit:
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
```

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

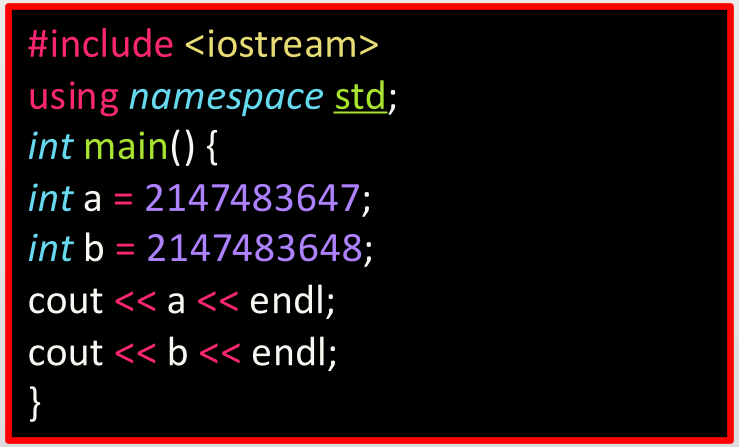
- Note that if you set the precision beyond the capacity of the data type, you get (deterministic) junk after a certain point

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

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

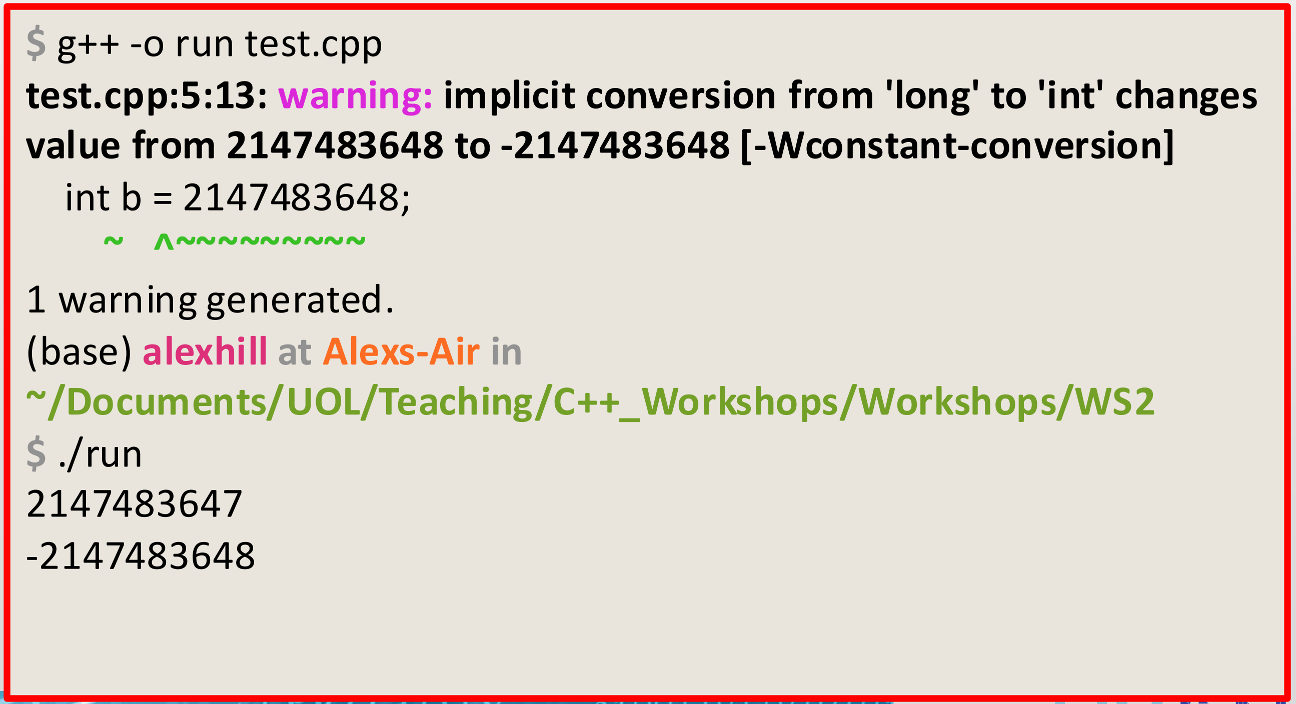
```
$ ./run
1.1234567891234568116
```

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



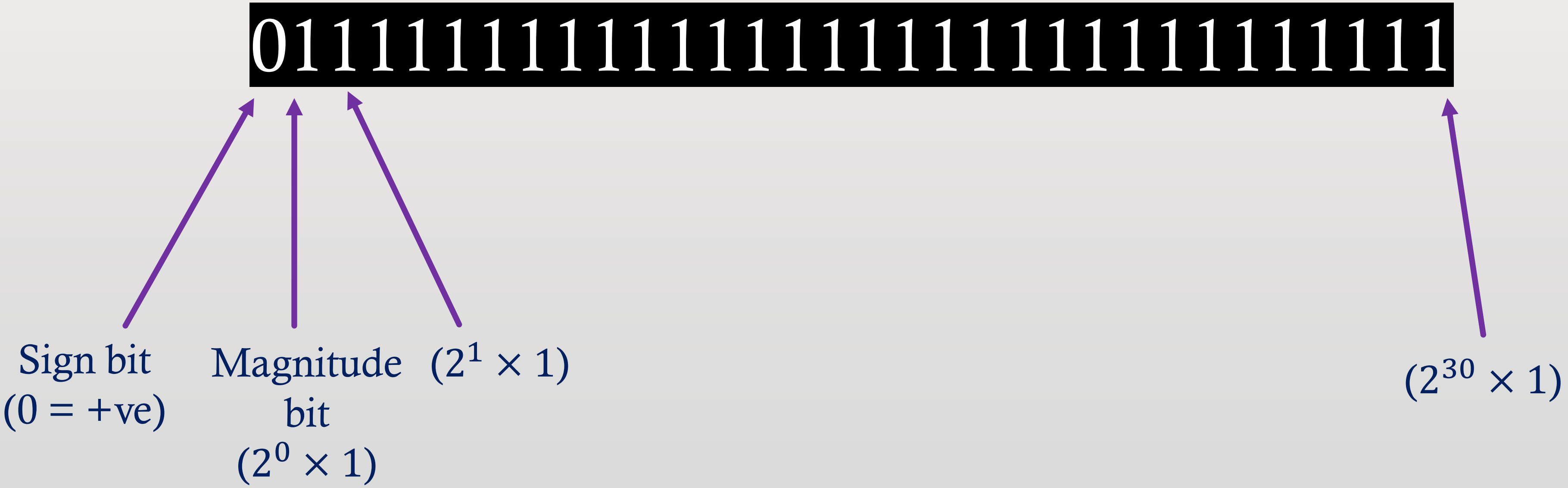
```
#include <iostream>
using namespace std;
int main() {
int a = 2147483647;
int b = 2147483648;
cout << a << endl;
cout << b << endl;
}
```



```
$ 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

For *int* (32 bits) this is 2147483647



01111111111111111111111111111111

Sign bit (0 = +ve)

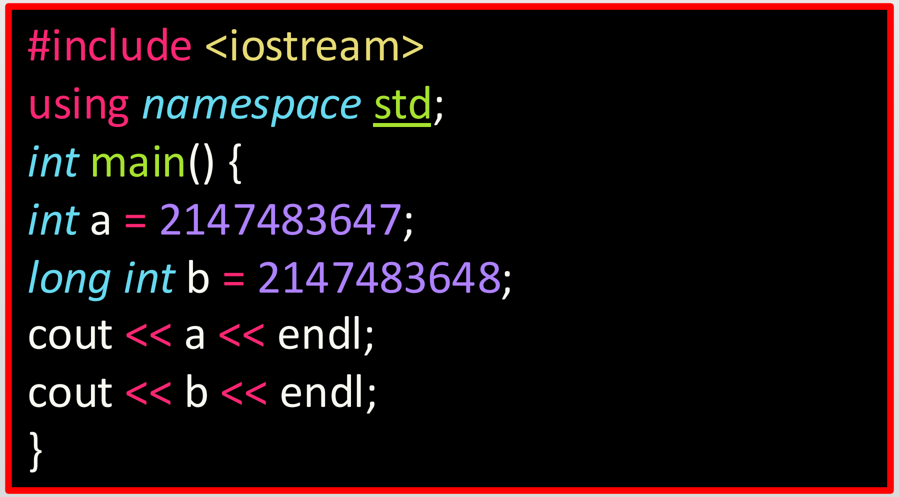
Magnitude bit $(2^0 \times 1)$

$(2^1 \times 1)$

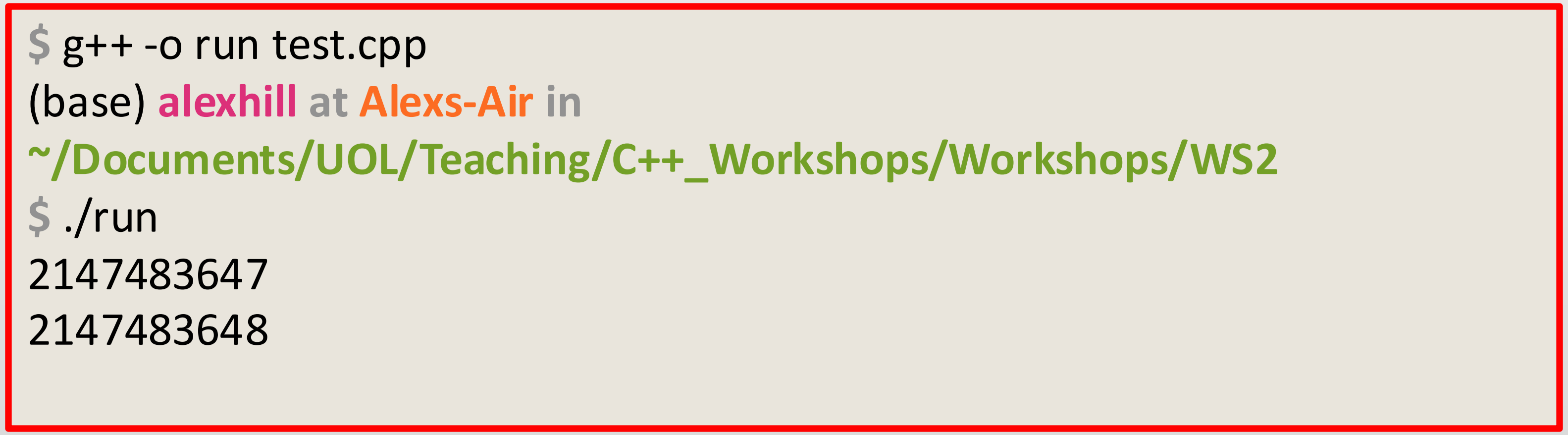
$(2^{30} \times 1)$

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-170



```
#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 bytes (eight bits per byte) 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
```

Boolean

- 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`

```cpp
#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 in programmes

Functions

- Functions are the building blocks of C++ programmes
  - A good practice is one function doing only one job

- 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

*https://stackoverflow.com/questions/4324763/can-we-have-functions-inside-functions-in-c

Functions

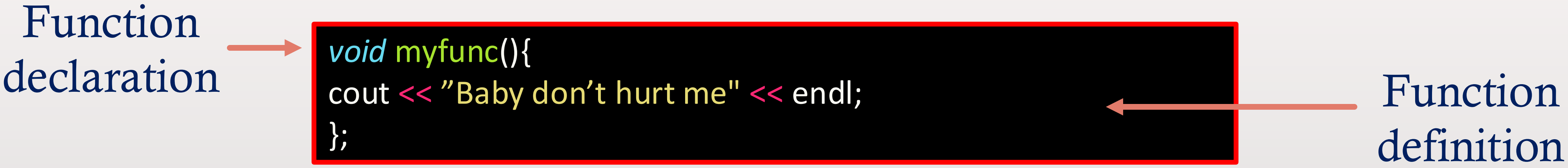
void is a null data type, used here as the function returns nothing

myfunc() calls the function we’ve created

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

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

void myfunc2(){
cout << ”Don’t hurt me" << endl;
};

int main() {
cout << ”What is love?" << endl;
myfunc(); // Call myfunc
myfunc2(); // Call myfunc2
cout << ”No more" << endl;
return 0;
}
```

Code is executed through main

Functions



Function declaration

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

Function definition

Functions: Declaration after main

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

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;
};
```

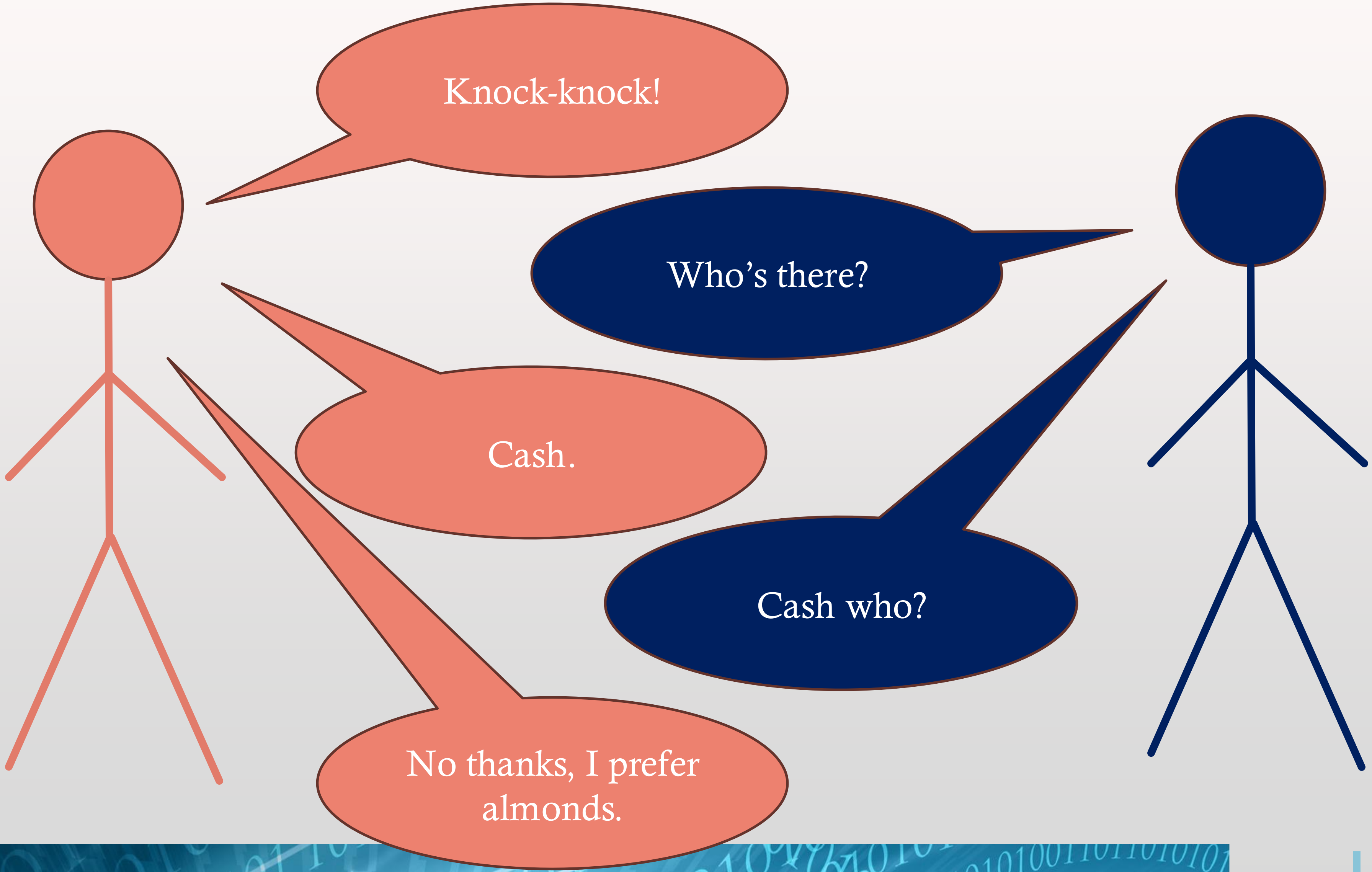
```
$ g++ -o output test.cpp
test.cpp:6:1: error: use of undeclared
identifier 'myfunc'
myfunc(); // Call myfunc
^
1 error generated.
```

Functions: Splitting the Declaration and the Definition

```
#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;
};
```

You can declare a function before its definition. The compiler needs to know what data type will be returned and what inputs it will take before it’s first called

Knock-knock jokes



Knock-knock!

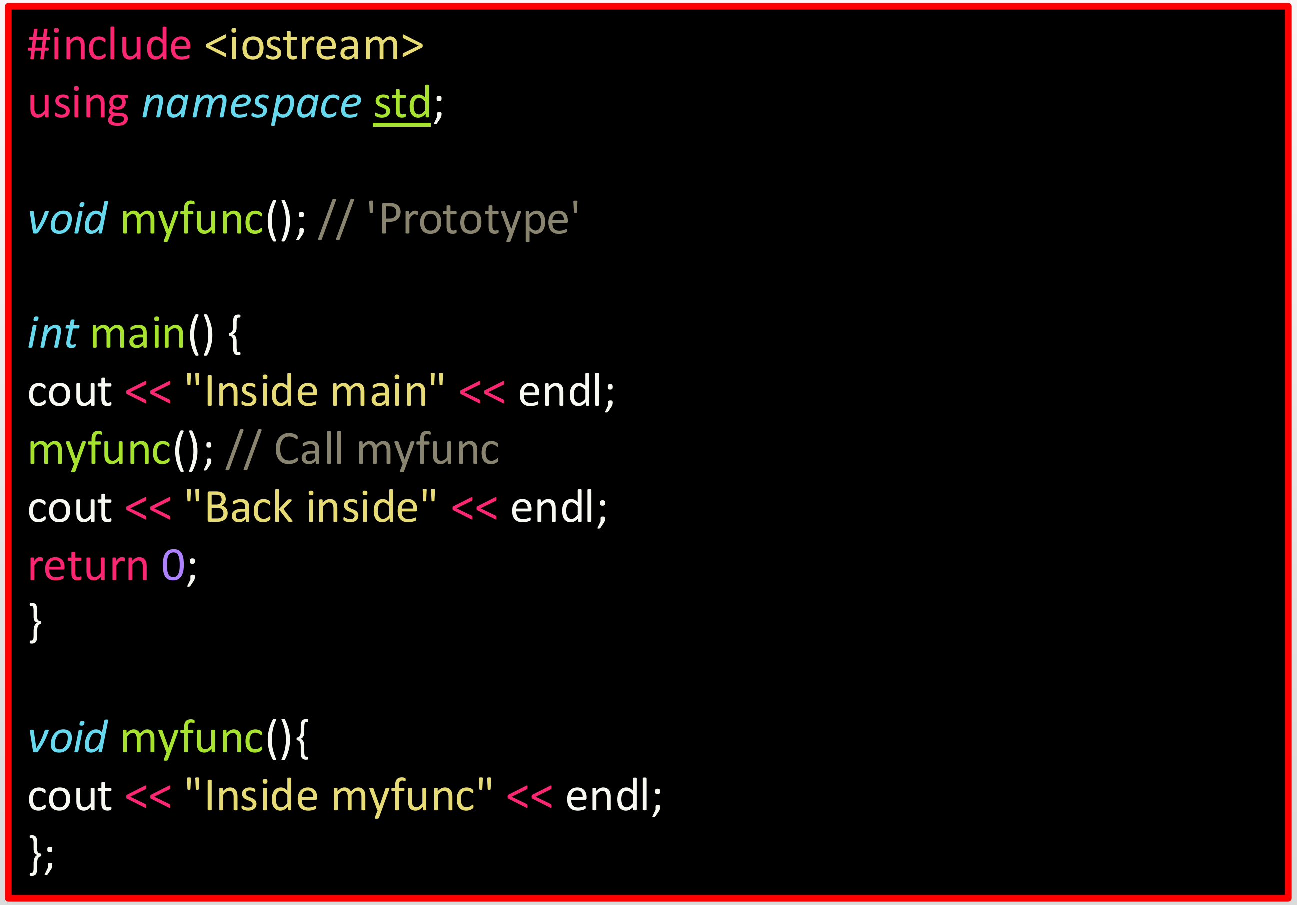
Who’s there?

Cash.

Cash who?

No thanks, I prefer almonds.

Challenge One:



```cpp
#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;
};
```

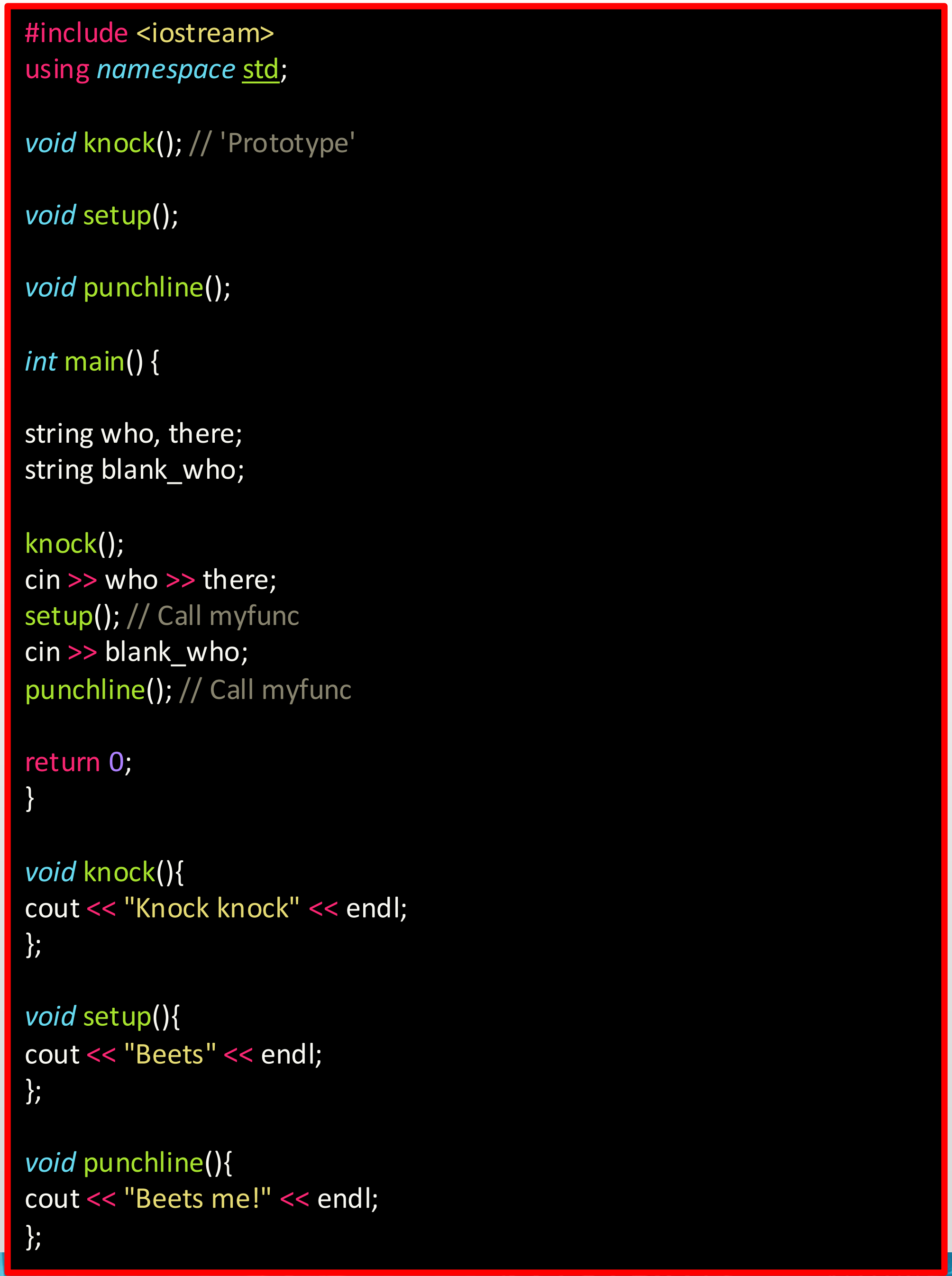
I would like you to tell a knock-knock joke using multiple functions.

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

You’ll need to use ‘cin’ for this (note that entering two words separated by a space will be taken as two inputs)

Best/worst joke wins! Send your scripts to my email address or post on the slack channel

a.d.hill@liverpool.ac.uk



```
#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;
};
```

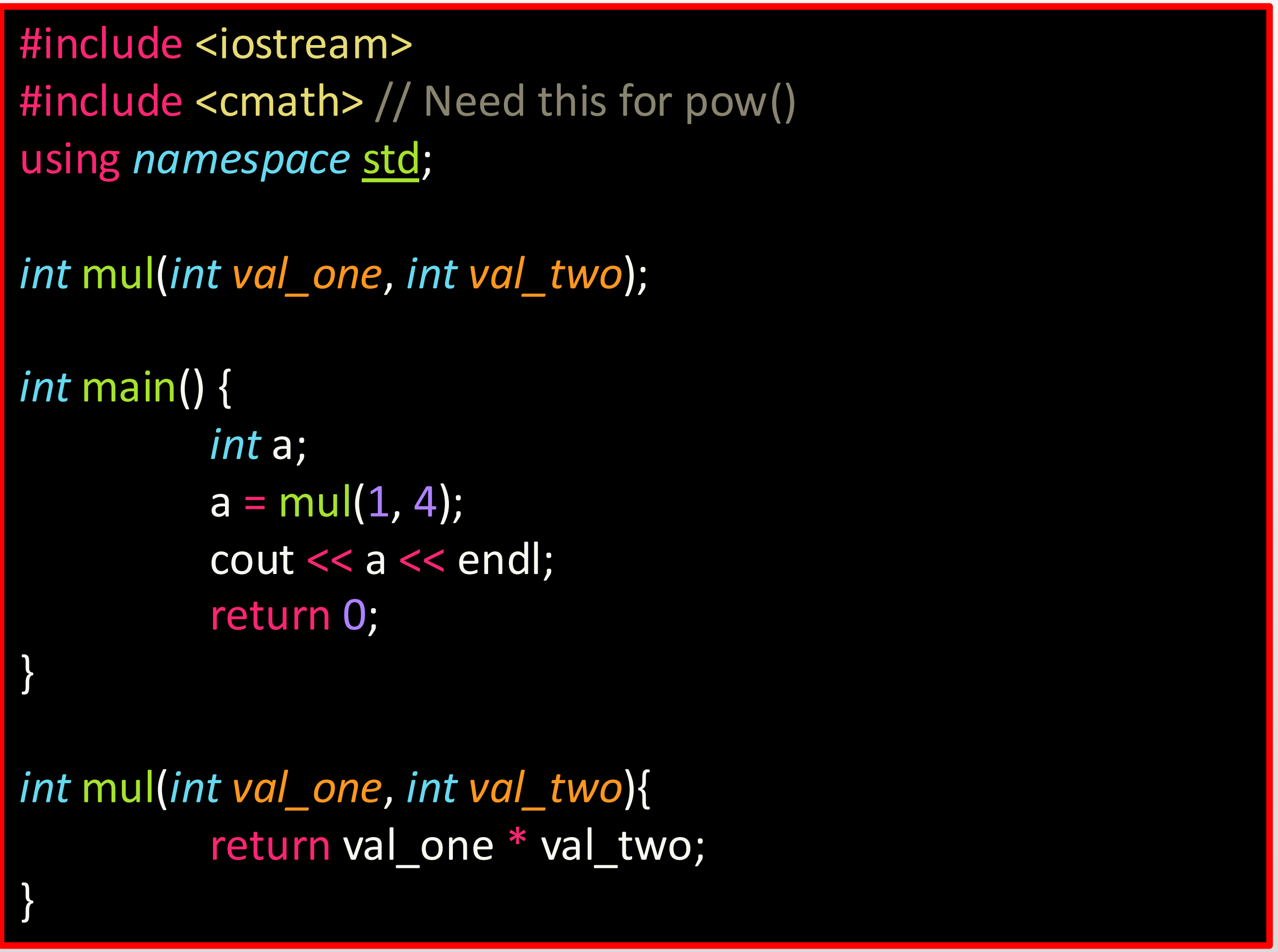
Functions: Arguments

int: my function mul will return an integer

```
#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;
}
```

int: my function mul has integer arguments

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())

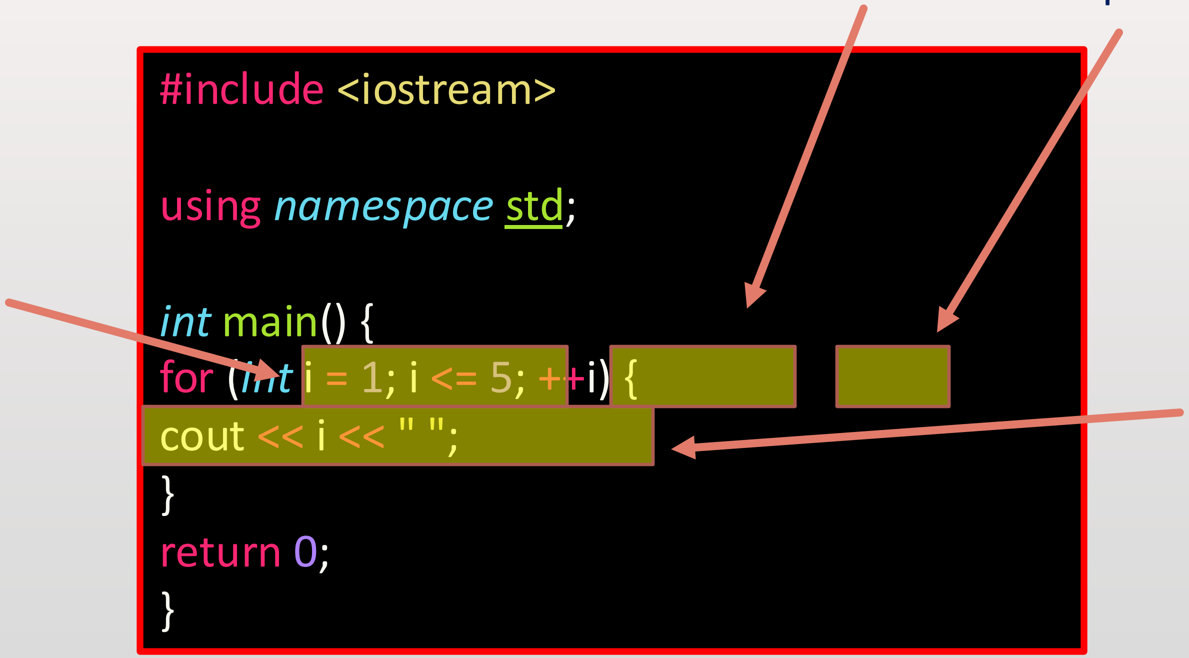
```
#include <iostream>
#include <cmath> // Need this for pow()
using namespace std;

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++



```
#include <iostream>

using namespace std;

int main() {
for (int i = 1; i <= 5; ++i) {
cout << i << " ";
}
return 0;
}
```

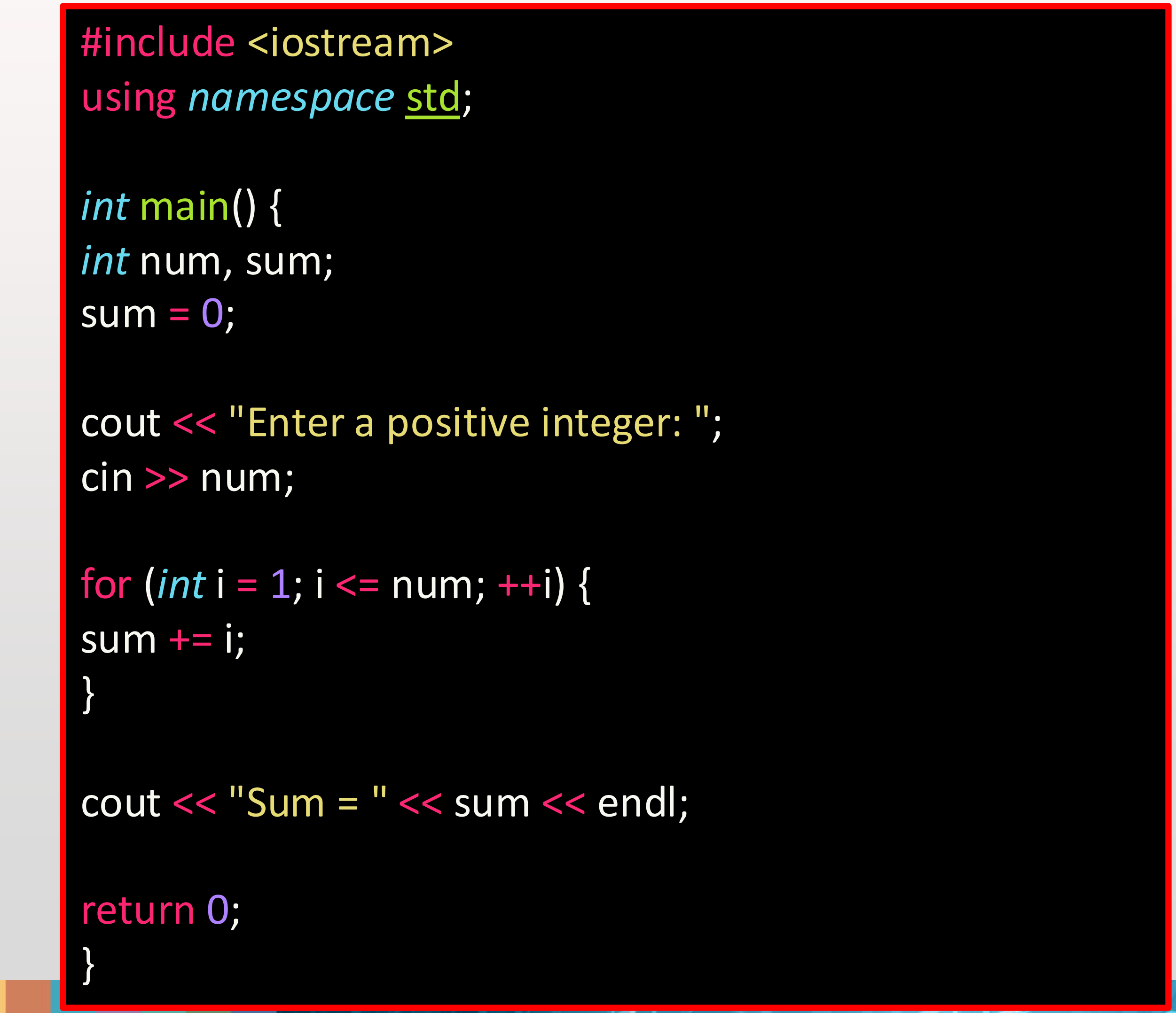
initialization

condition

Update (optional)

Block of code within loop

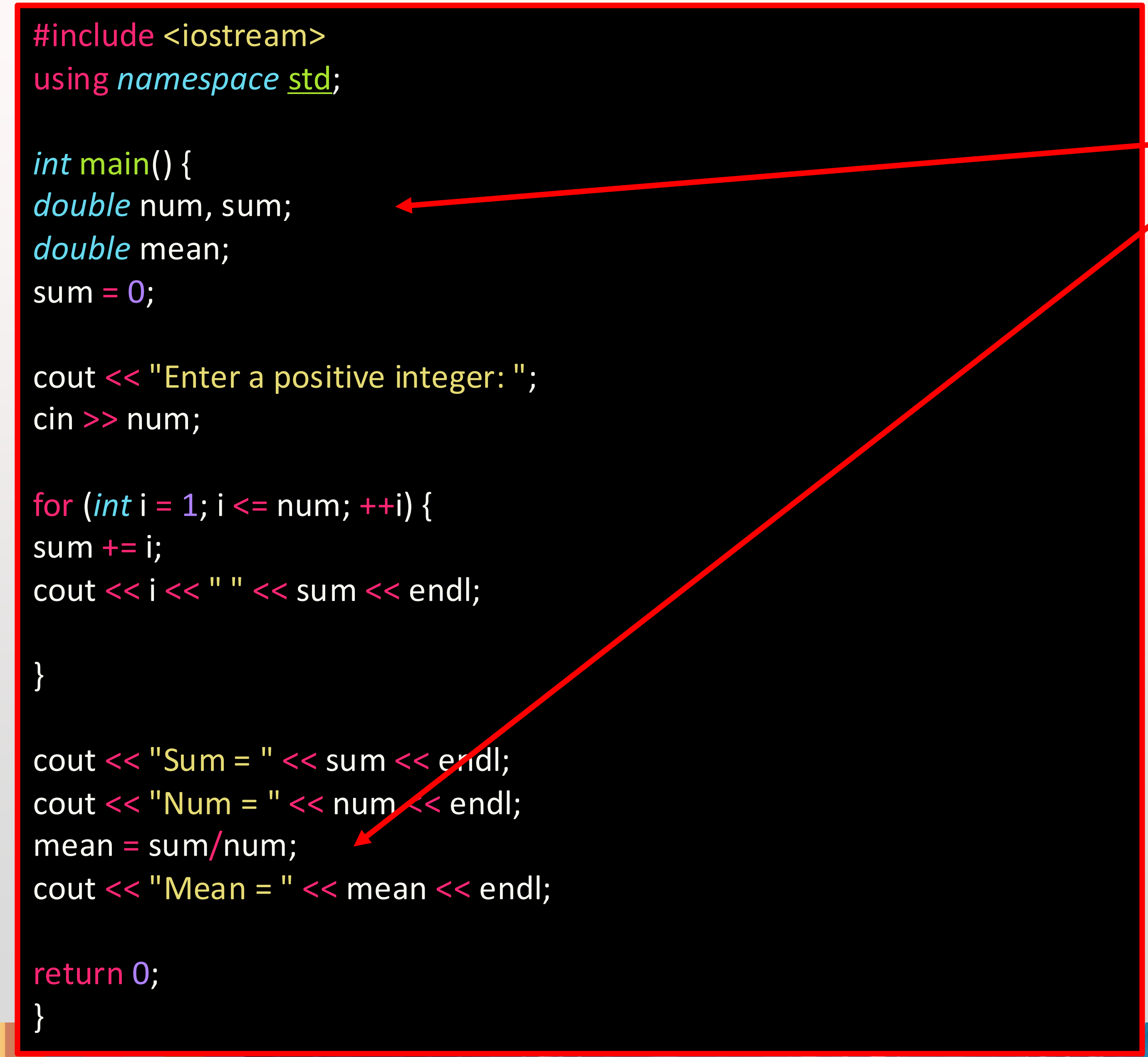
Challenge Three:



```
#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;
}
```

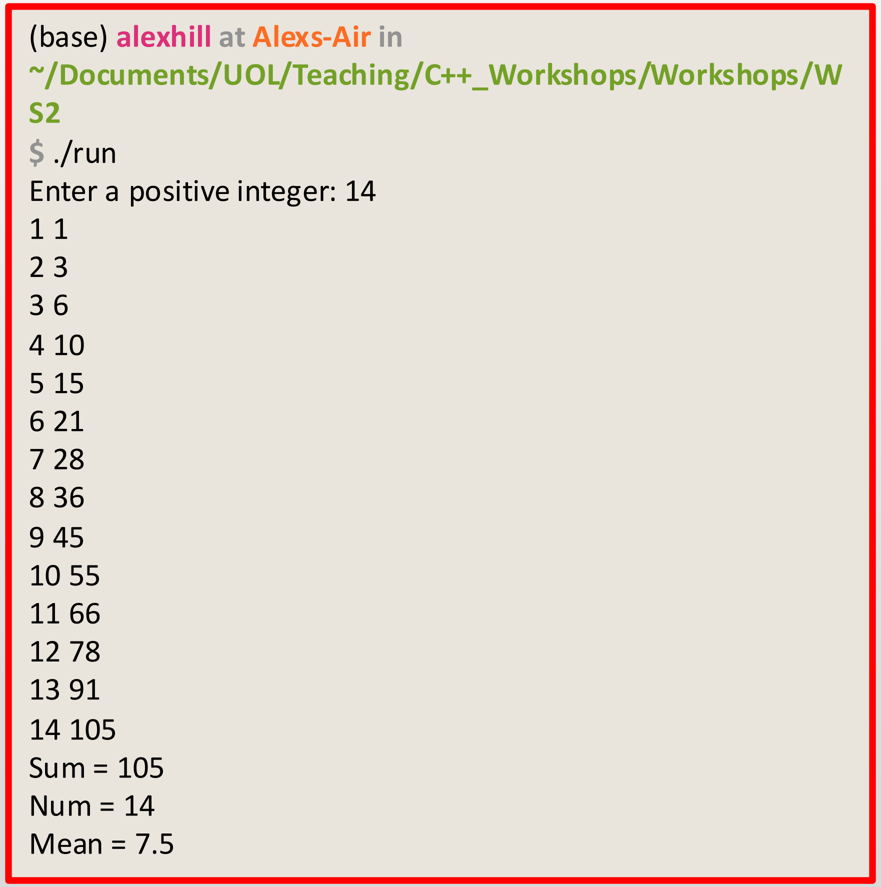
- This code computes the sum of numbers up to num

- Can you adapt this to compute the mean of numbers up to num?



```cpp
#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;
}
```

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



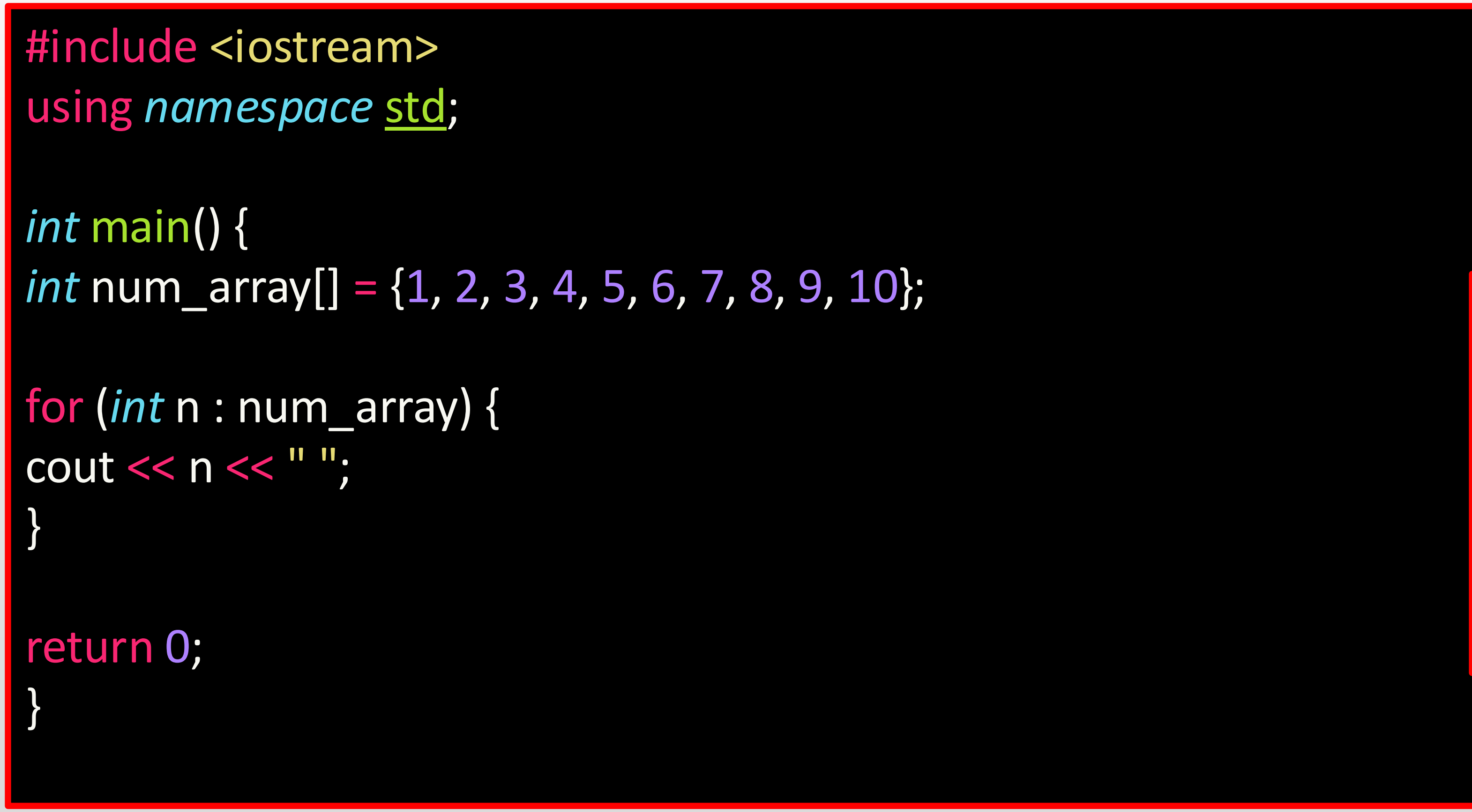
```
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Workshops/W
S2
$ ./run
Enter a positive integer: 14
1 1
2 3
3 6
4 10
5 15
6 21
7 28
8 36
9 45
10 55
11 66
12 78
13 91
14 105
Sum = 105
Num = 14
Mean = 7.5
```

Ranged For Loop (New For C++11)

Array object

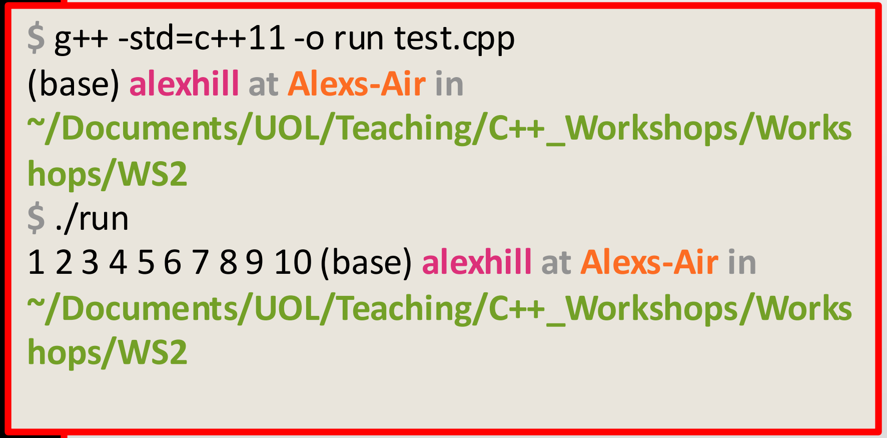
```
#include <iostream>
using namespace std;

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



```
#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;
}
```

Disable warnings with this argument



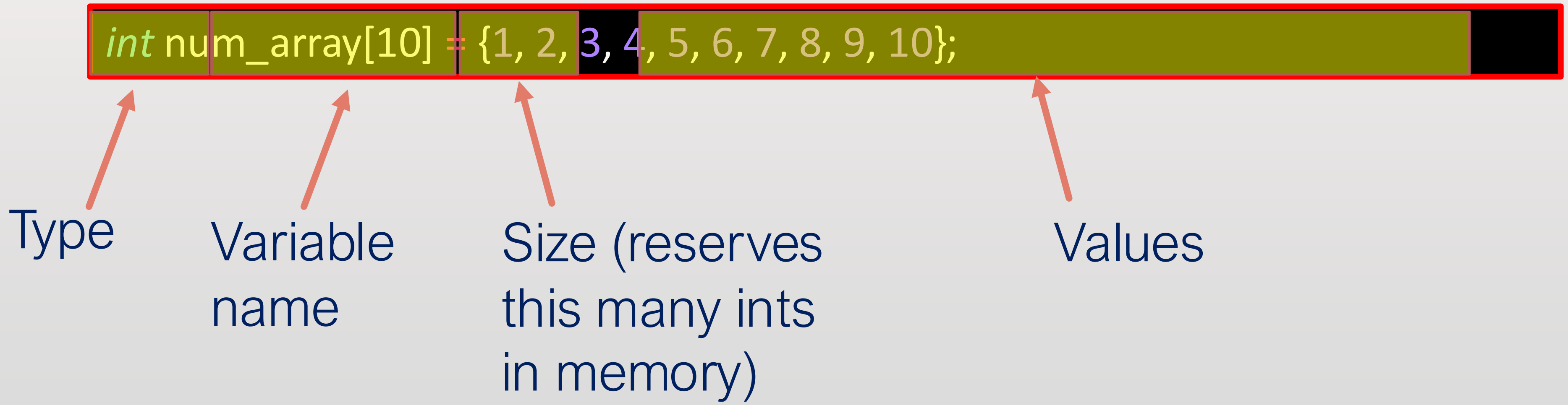
```
$ g++ -std=c++11 -o run test.cpp
(base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Works
hops/WS2
$ ./run
1 2 3 4 5 6 7 8 9 10 (base) alexhill at Alexs-Air in
~/Documents/UOL/Teaching/C++_Workshops/Works
hops/WS2
```

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



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

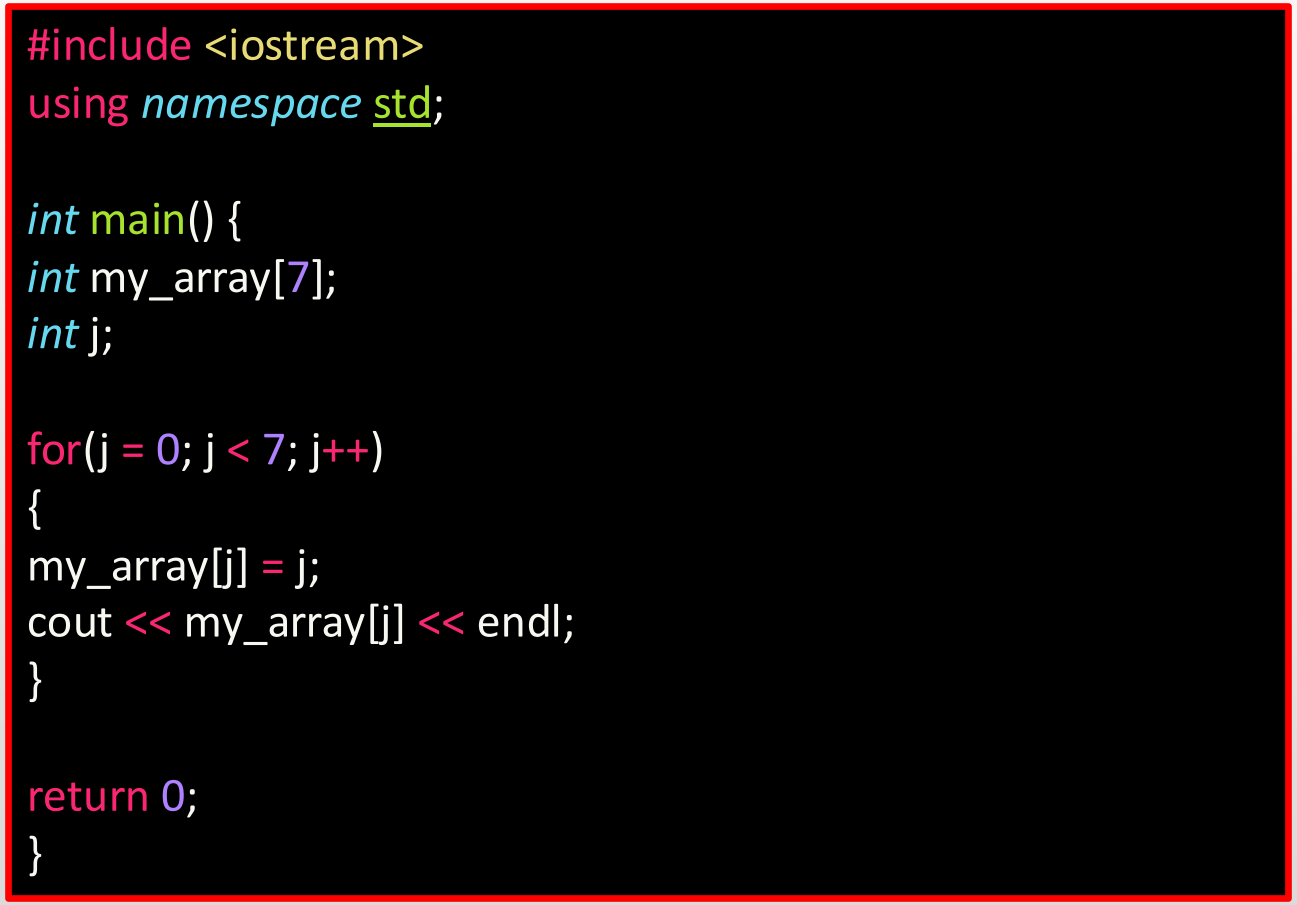
Type

Variable name

Size (reserves this many ints in memory)

Values

Array Indexing

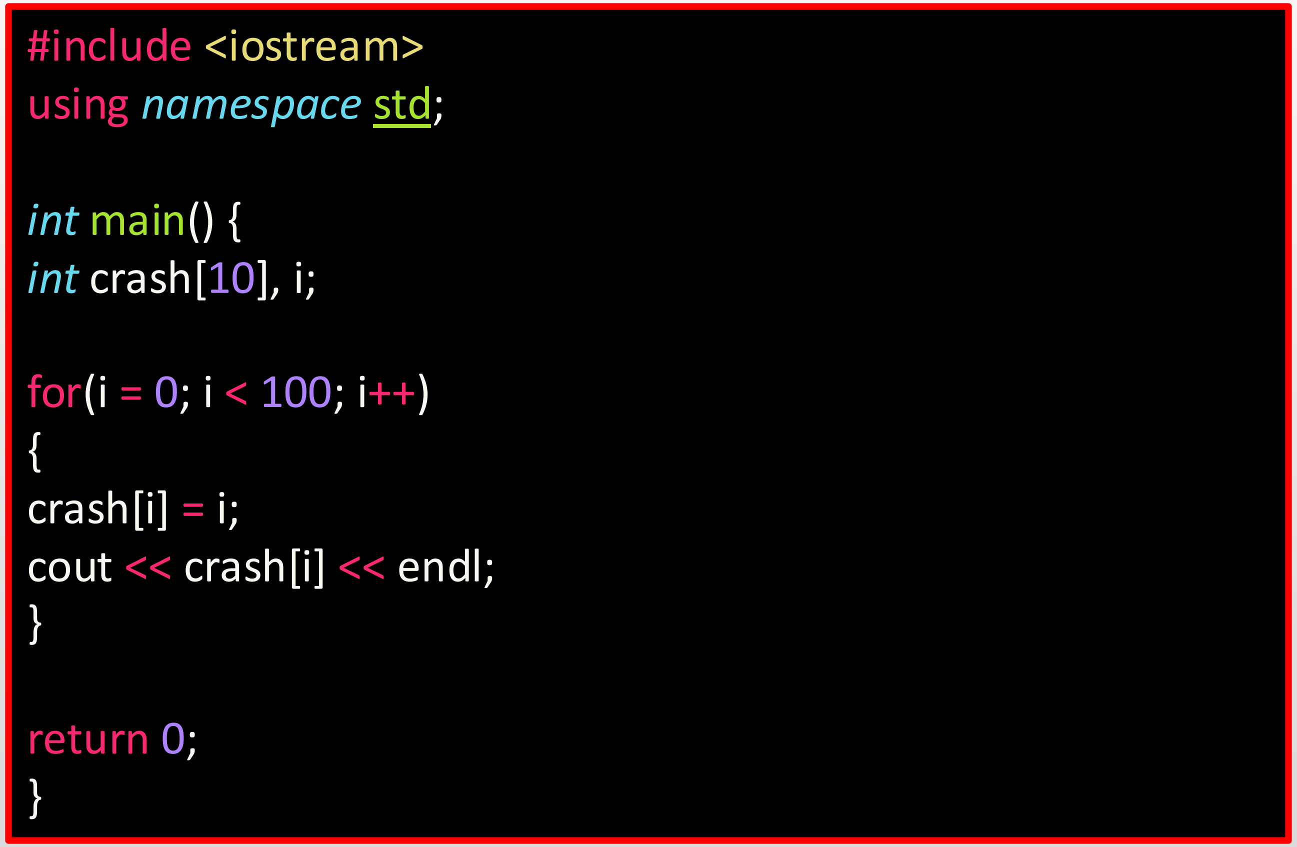


```
#include <iostream>
using namespace std;

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])

Array Indexing

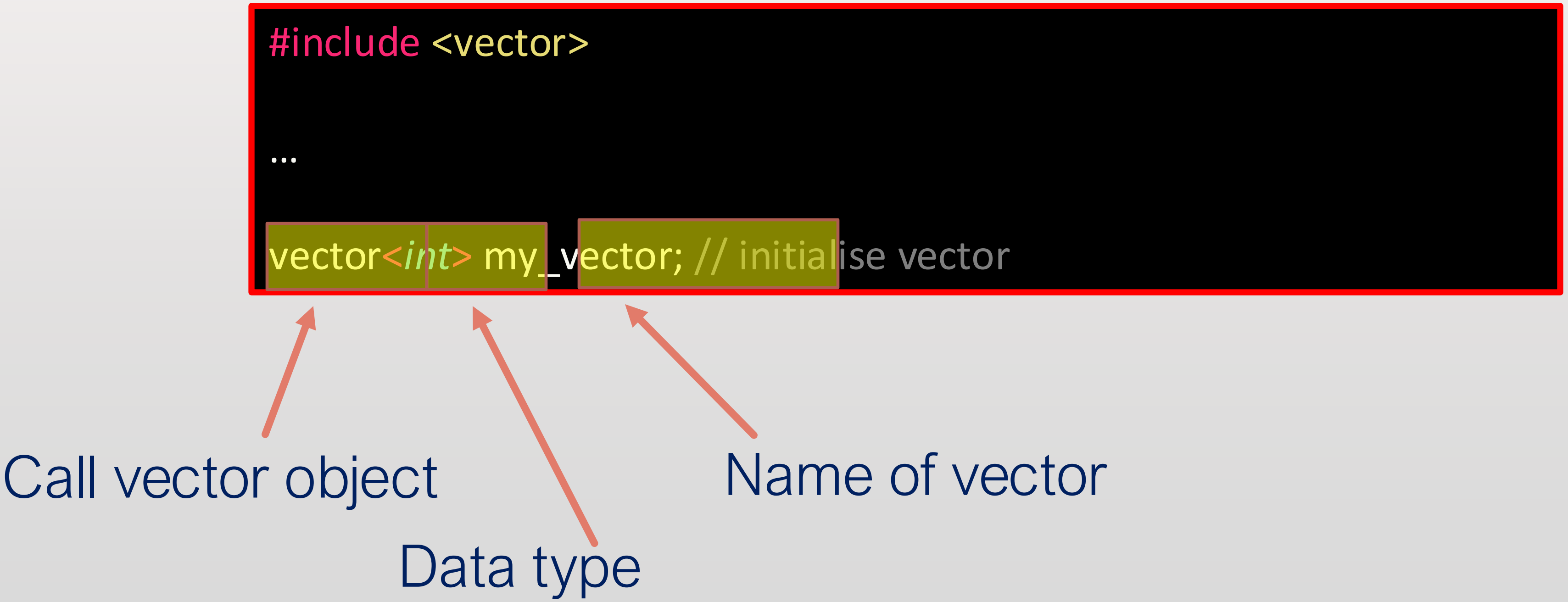


```
#include <iostream>
using namespace std;

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



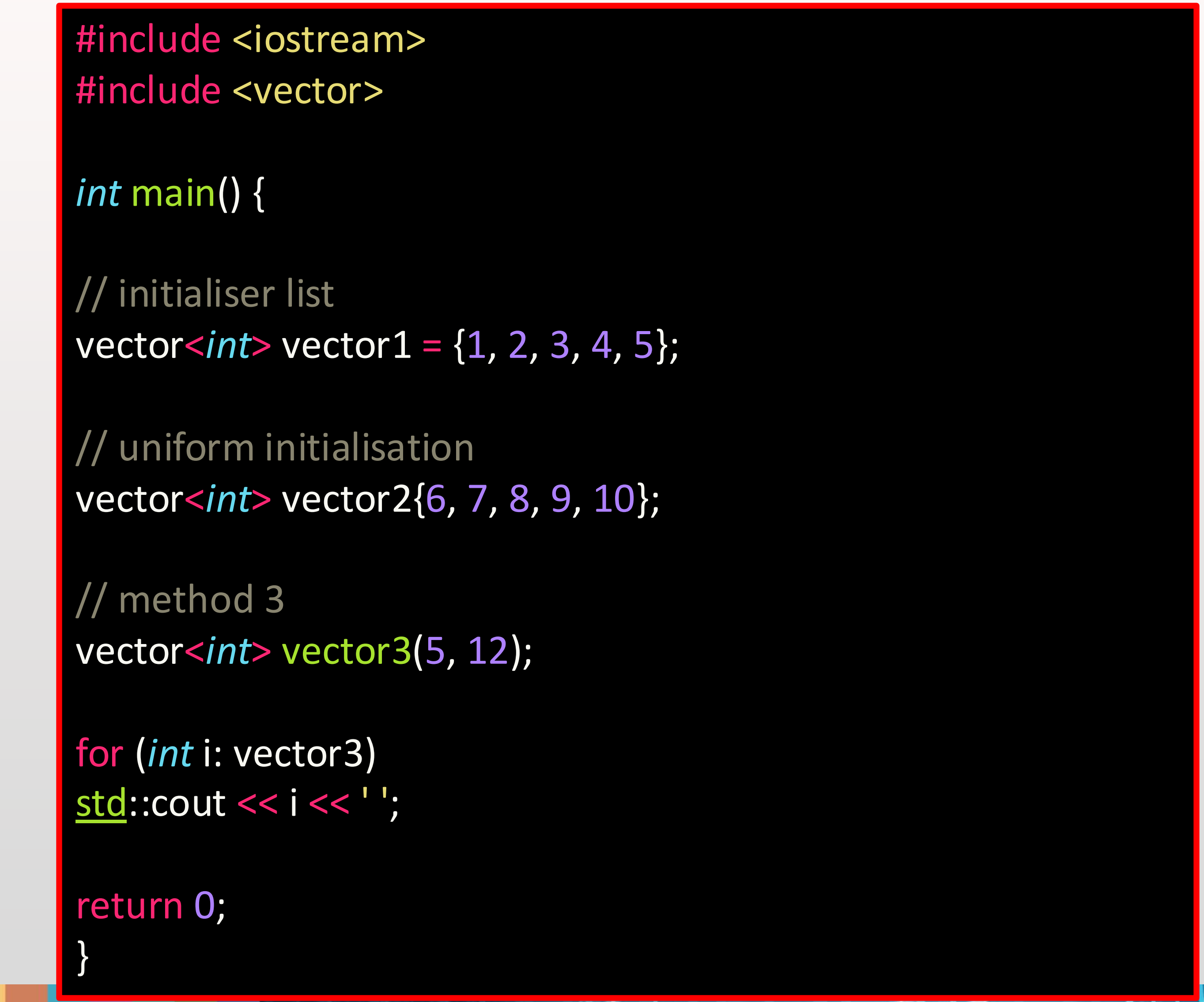
```
#include <vector>

…

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

Call vector object

Data type

Name of vector

Vector Initialisation



```
#include <iostream>
#include <vector>

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 vector2 are intialised 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> v1 = {1, 2, 3, 4, 5};
// add the integers 6 and 7 to the vector
v1.push_back(6);
v1.push_back(7);
// remove the last element
v1.pop_back();
```

```
// change elements at indexes 1 and 4
v1.at(1) = 9;
v1.at(4) = 7;
```

```
// access vector elements
v1.at(0);
// or
v1[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.
```

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

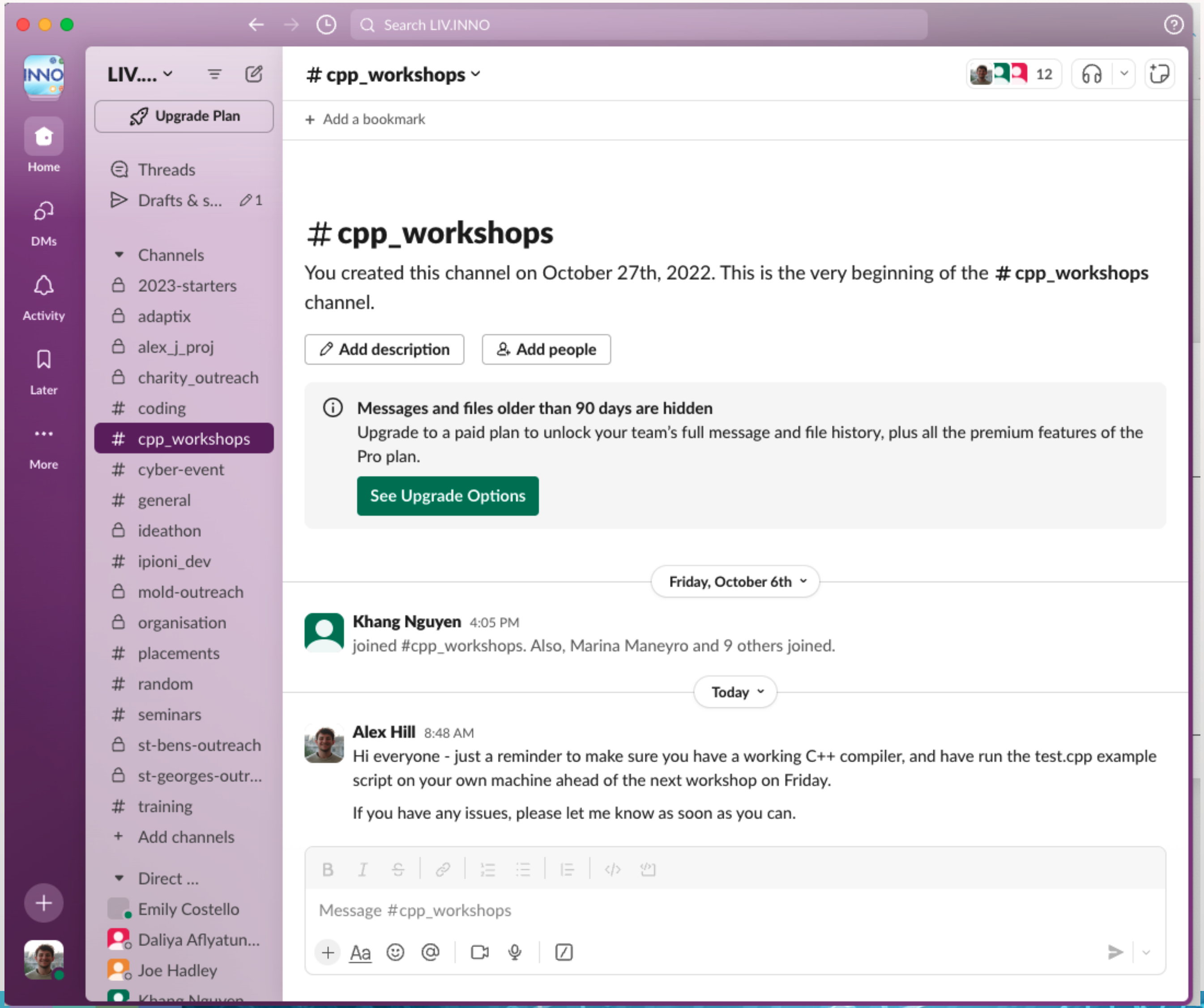
Challenge Four (Homework)

- Create an evenly-space array (or vector) between 0 and π (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
- Send me your scripts by Wednesday evening next week (18/11/23)

Next Week

- Passing vectors into functions (pointers)
- Plotting data
- Introduction to Monte Carlo methods

Thanks!



cpp_workshops

You created this channel on October 27th, 2022. This is the very beginning of the # cpp_workshops channel.

Messages and files older than 90 days are hidden
Upgrade to a paid plan to unlock your team's full message and file history, plus all the premium features of the Pro plan.

Khang Nguyen 4:05 PM
joined #cpp_workshops. Also, Marina Maneyro and 9 others joined.

Alex Hill 8:48 AM
Hi everyone - just a reminder to make sure you have a working C++ compiler, and have run the test.cpp example script on your own machine ahead of the next workshop on Friday.

If you have any issues, please let me know as soon as you can.

Any questions in the Slack Channel, or message me privately