C++ - Pre-lecture 3

Strings and vectors

Caterina Doglioni, PHYS30762 - OOP in C++ 2023

Who am I

- **Professor in particle physics** at the University of Manchester

- Previously:
  - ***senior lecturer*** at Lund University (Sweden)
    - still working there (remotely) 1 day a week
  - ***post-doctoral researcher*** at the University of Geneva
  - ***PhD student*** at the University of Oxford
  - ***undergraduate student*** at the University of Rome "Sapienza", degree in particle physics

    - This is where I got a taste for C++ as an undergraduate …and really liked it →now I use every day for research

    - I hope I can pass some of this enthusiasm to you as well

Where I can be found in order of contactability

- **In the labs on Thursdays**
- **On Teams**
- Unless you want to discuss personal issues, please avoid direct messages and use common channels
    - No stupid questions: if you have that question, it's likely someone else will have the same question
        - By asking in public, the answer helps everyone
- **In my office in Schuster 6.52** (not recommended)
    - If I'm in my office it means I'm in a meeting with postdocs / MPhys / PhD students / people at CERN and can't see you right away
        - Also the reason why I may not respond promptly to Teams queries
    - If you do need to see me, please schedule a meeting first

In this (pre-)lecture

- We will learn to use two very useful features of C++
  - **Strings**
    - Store and manipulate groups of **characters**
  - **Vectors**
    - Store and manipulate groups of **numbers**
      - note: **any object** can be stored in a vector
      - Advanced topic: C++ & related libraries also offer mathematical vectors
- These are a part of the <u>standard library</u>
  - The standard libraries need to be included, see pre-lectures 1 and 2

Part 1: strings

1.1 strings as array

Why strings?

They are essential for any programming language

- A **string** is an **array of characters**
  - We will first see this implementation (that some codes still use)
  - …but it is less practical than an implementation that has a number of features already available for you: **`std::string`**
- Examples of usage for strings:
  - Names of files to read and write data
  - Parsing text (e.g. counting the number of words, plagiarism checking)
  - Storing human-readable information in databases (e.g. names and addresses, degree courses)

Strings as array of characters `char[]`

How to define them

- Relatively simple: use an **array** of type `char`

```
const size_t no_char{100}; // Size of array, keep constant as it won't change
char string1[no_char]; //fixed length array to store string
// Or we can do it this way
char *string2;
string2 = new char[no_char];
// Get the string filled with something
sprintf(string2,"This is a string that has fewer than 100 characters.");
delete string2;
```

You can copy&paste&experiment with this code

- These character arrays are known as *null-terminated strings*
  - This is because the last character (1-byte) is used for the *null-character*, which indicates the end of the string
  - You don't have to explicitly add this character (but you must have enough space in the array for it)
  - If you do add this character before the end of the string, then your string will be seen as terminating prematurely…

Strings as array of characters char[]

How to manipulate them: filling with text

- There is a library that helps with operations on char[] strings

```
#include <cstring>
```

- Once the arrays are there, you can use <cstring> functions to add text and manipulate them, e.g.
    - Let’s add content to a string using the strcpy function

```
strcpy(string1,"This is string1");
strcpy(string2,"This is string2”);
```

- You can check what is in each string by printing to screen, using cout

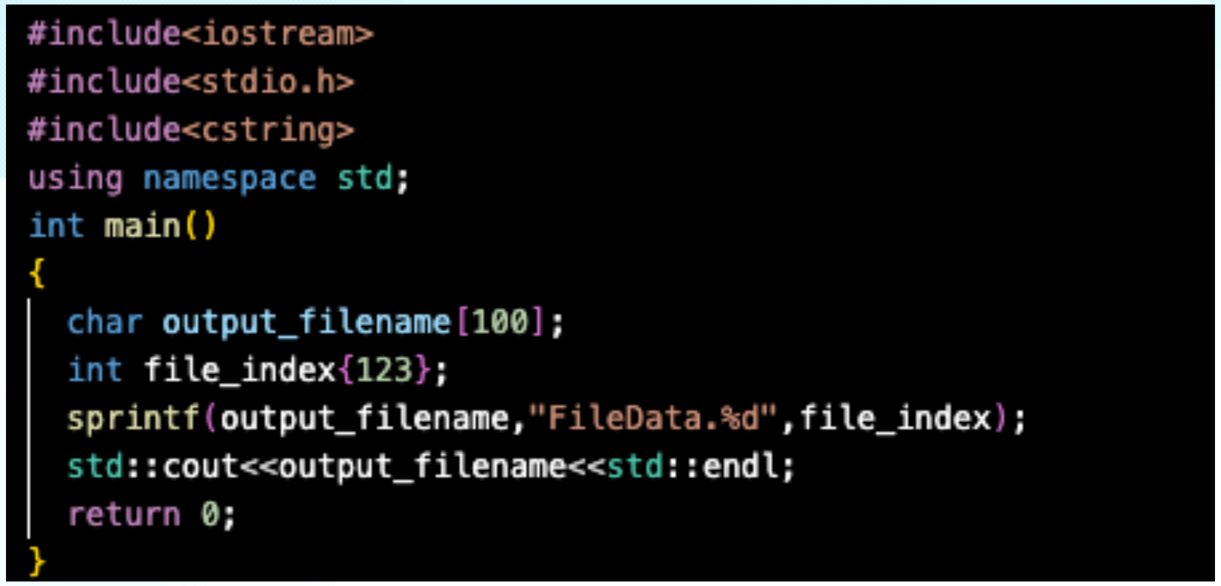
```
std::cout<<string1<<std::endl;
```

MANCHESTER 1824 The University of Manchester

Strings as array of characters `char[]`

How to manipulate them: filling with text & more

- How do we incorporate non-string data and format it correctly?
  - Example: a filename with an integer identifier
- C allows us to use the `sprintf` function



```cpp
#include<iostream>
#include<stdio.h>
#include<cstring>
using namespace std;
int main()
{
  char output_filename[100];
  int file_index{123};
  sprintf(output_filename,"FileData.%d",file_index);
  std::cout<<output_filename<<std::endl;
  return 0;
}
```

You can't copy and paste this code, but you can find it on the course's GitHub repository

Find this code at Prelecture3/cstringsprintf.cpp

- Downside: we need to be telling this function what format to expect, see e.g. here
- Upside: you can control significant digits (important for scientific code outputs)

Strings as array of characters `char[]`

How to manipulate them: comparisons

- Let's compare strings with the `strcmp` function
  - You can make the two strings the same

```
strcpy(string1,"This is string1");
strcpy(string2,"This is string1”);
```

  - Then check if they are the same

```
std::cout<<"comparing string1 and string2: “;
if(strcmp(string1,string2)){
   std::cout<<"strings are not equal”<<std::endl;
   }
else {
   std::cout<<"strings are equal"<<std::endl;
   std::cout<<string1<<std::endl;
   }
```

MANCHESTER 1824 The University of Manchester

Strings as array of characters `char[]`

How to manipulate them: joining, copying, determining length

- Join two strings together with `strcat` (2nd added to 1st)

```
strcat(string1,string2);
```

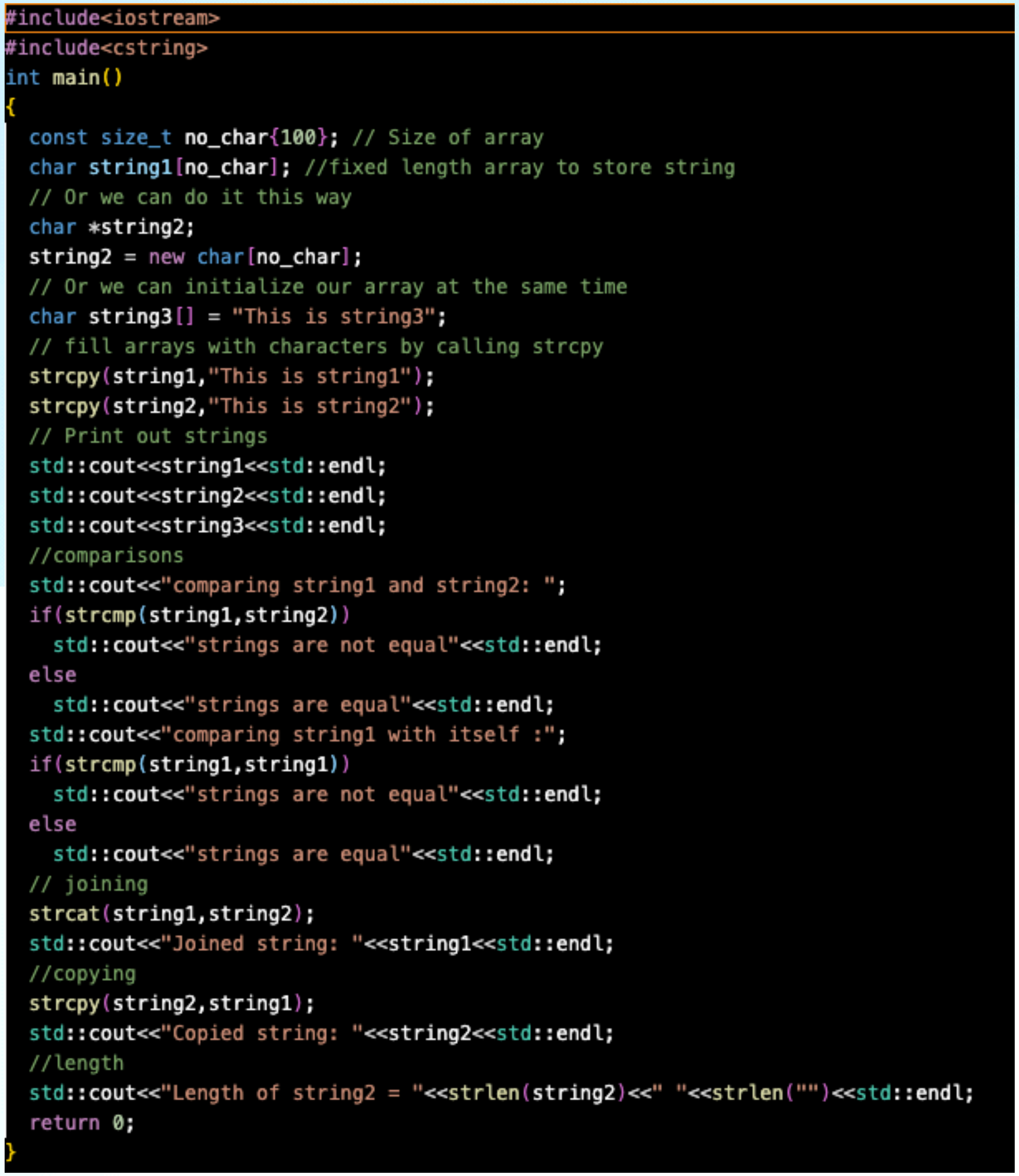
- Copy strings with `strcpy` (2nd added to 1st)

```
strcpy(string1,string2);
```

- Determine string length with `strlen`

```
std::cout<<"Length of string2 = "<<strlen(string2)<<" “<<strlen("")<<std::endl;
```

Summary code: strings as arrays



```cpp
#include<iostream>
#include<cstring>
int main()
{
  const size_t no_char{100}; // Size of array
  char string1[no_char]; //fixed length array to store string
  // Or we can do it this way
  char *string2;
  string2 = new char[no_char];
  // Or we can initialize our array at the same time
  char string3[] = "This is string3";
  // fill arrays with characters by calling strcpy
  strcpy(string1,"This is string1");
  strcpy(string2,"This is string2");
  // Print out strings
  std::cout<<string1<<std::endl;
  std::cout<<string2<<std::endl;
  std::cout<<string3<<std::endl;
  //comparisons
  std::cout<<"comparing string1 and string2: ";
  if(strcmp(string1,string2))
    std::cout<<"strings are not equal"<<std::endl;
  else
    std::cout<<"strings are equal"<<std::endl;
  std::cout<<"comparing string1 with itself :";
  if(strcmp(string1,string1))
    std::cout<<"strings are not equal"<<std::endl;
  else
    std::cout<<"strings are equal"<<std::endl;
  // joining
  strcat(string1,string2);
  std::cout<<"Joined string: "<<string1<<std::endl;
  //copying
  strcpy(string2,string1);
  std::cout<<"Copied string: "<<string2<<std::endl;
  //length
  std::cout<<"Length of string2 = "<<strlen(string2)<<" "<<strlen("")<<std::endl;
  return 0;
}
```

Find this code at: Prelecture3/cstring_summary.cpp

Part 1: strings
1.2 strings from the standard library

MANCHESTER 1824 The University of Manchester

Strings using the standard library

How to define them, instead of arrays of chars

- C++ provides a string datatype it's actually a class…more next week

```
#include <string>
```

- Create a string (object)

```
std::string my_first_string{"Hello, world!"};
```

- Use the length function to find out its length

```
std::cout << "Length of string = "<<my_first_string.length()<<std::endl;
```

- …and note that a standard library string can be used as array of characters as well!

```
std::cout<<"2nd character in string is "<<my_first_string[1]<<std::endl;
```

MANCHESTER 1824 The University of Manchester

Strings using the standard library

How to define them from terminal input / comparisons

- Strings can also be defined from the input stream cin
  - No need to specify length of string in advance!

```
std::string input_string;
std::cout<<"Enter a phrase: ";
std::cin>>input_string;
```

  - This only extracts one word as the input terminates at first whitespace character (leaving rest of text and newline in buffer)
  - If you want an entire sentence, use getline() still included in string

```
getline(std::cin,input_string);
```

- String comparison works as for other data types, using ==

```
string my_first_string{"Hello, world!"};
string my_second_string{"Hello, world!"};
if(my_first_string == my_second_string)
std::cout<<"Strings match!"<<std::endl;
```

MANCHESTER 1824 The University of Manchester

Strings using the standard library

How to join and append strings, and extract substrings

- Joining two strings is easy - just add together with **+**

```
string my_first_string{“Eight bytes walk into a bar.  Bartender:‘Can I get you anything? ’ ”};
string my_second_string{“Bytes: yes, make us a double.”};
string joined_string{my_first_string + my_second_string};
std::cout<<"Joined string: “<< joined_string <<std::endl;
```

- Can also append strings to each other

```
my_first_string += my_second_string;
std::cout<<"Appended string: "<<my_first_string <<std::endl;
```

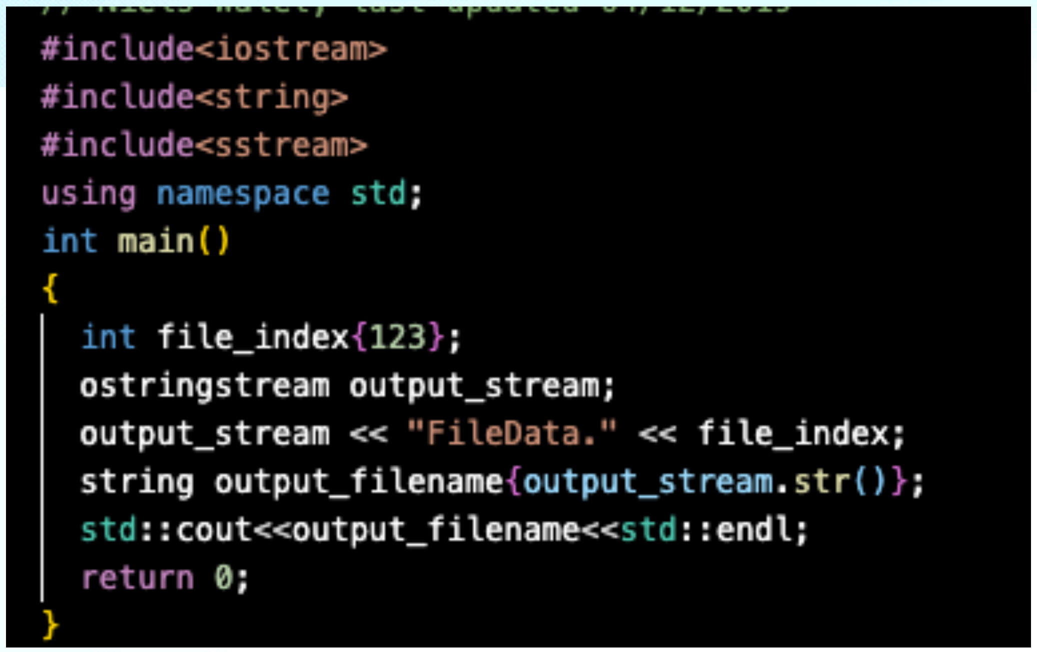
- Extracting substrings (e.g. words) is also easy using `substr`
  - Arguments are: position of the first character (minimum is zero) and (optional) length of substring

```
const size_t first{0};
const size_t last{5};
string a_word = joined_string.substr(first,last-first+1);
```

Strings and streams

How to add non-character input (e.g. numbers) to a string?

- Crucial requirement when dealing with strings: incorporating non-string data and formatting it correctly
  - Example: a filename with an integer identifier
- C++ allows us to use *string streams (*ostringstream*)*



```cpp
#include<iostream>
#include<string>
#include<sstream>
using namespace std;
int main()
{
  int file_index{123};
  ostringstream output_stream;
  output_stream << "FileData." << file_index;
  string output_filename{output_stream.str()};
  std::cout<<output_filename<<std::endl;
  return 0;
}
```

Find this code at: Prelecture3/cppstringstream.cpp
Compare it with Prelecture3/cstringsprintf.cpp

- At this point we can add input to the string via the << operator

Strings and streams

How to add non-character input (e.g. numbers) to a string?

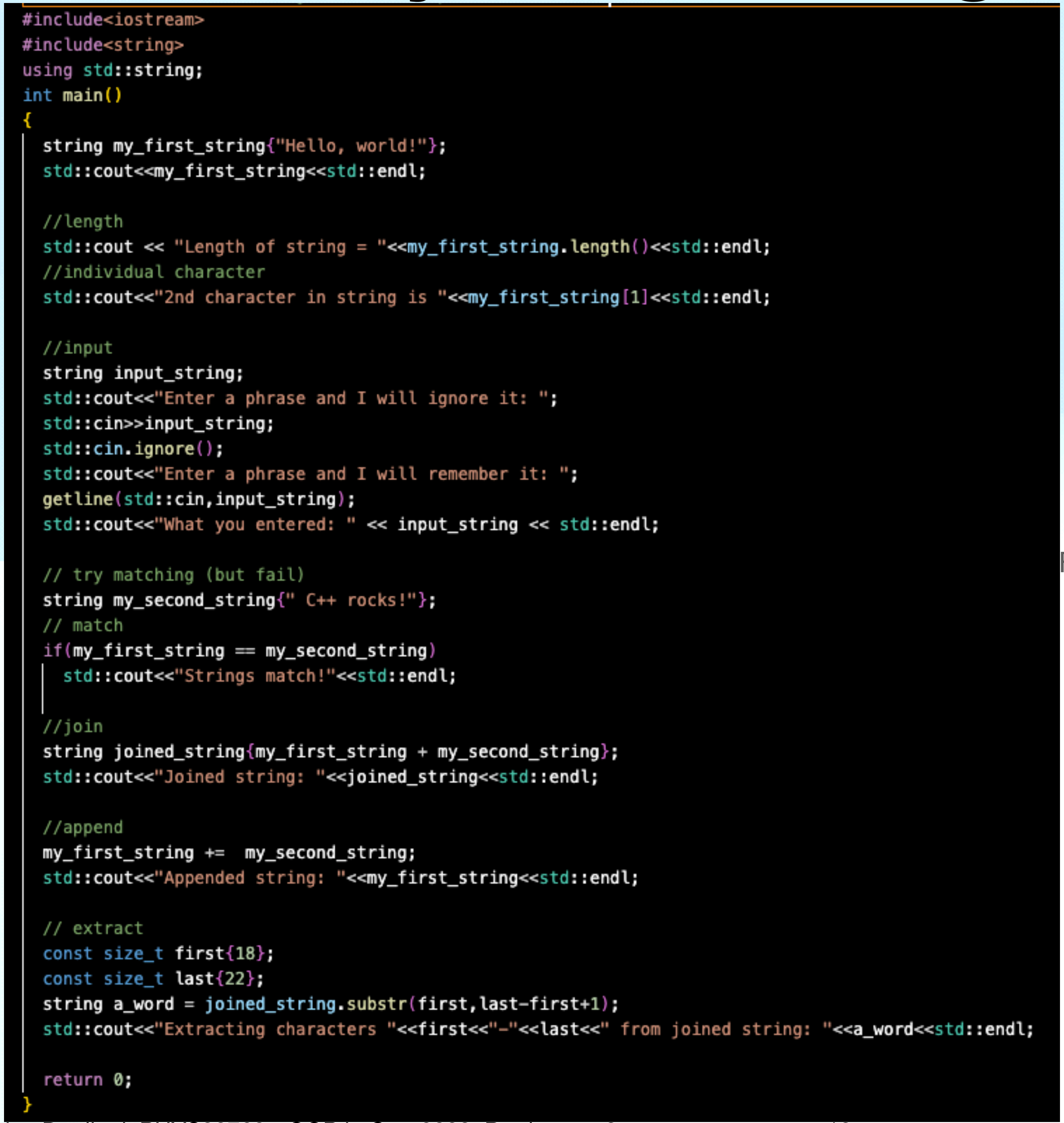
- To extract data from a stringstream, we append .str() to the stringstream variable
  (preview of later lecture: this mean that we call the “class function” str())

```
string output_filename{output_stream.str()};
std::cout<<“string output of stringstream: "<<output_filename
<<std::endl;
```

- We can also use .str() to set the content of the stream, e.g. to clear the content of the buffer

```
output_stream.str("");
```

Summary code: strings and streams



```cpp
#include<iostream>
#include<string>
using std::string;
int main()
{
  string my_first_string{"Hello, world!"};
  std::cout<<my_first_string<<std::endl;

  //length
  std::cout << "Length of string = "<<my_first_string.length()<<std::endl;
  //individual character
  std::cout<<"2nd character in string is "<<my_first_string[1]<<std::endl;

  //input
  string input_string;
  std::cout<<"Enter a phrase and I will ignore it: ";
  std::cin>>input_string;
  std::cin.ignore();
  std::cout<<"Enter a phrase and I will remember it: ";
  getline(std::cin,input_string);
  std::cout<<"What you entered: " << input_string << std::endl;

  // try matching (but fail)
  string my_second_string{" C++ rocks!"};
  // match
  if(my_first_string == my_second_string)
    std::cout<<"Strings match!"<<std::endl;

  //join
  string joined_string{my_first_string + my_second_string};
  std::cout<<"Joined string: "<<joined_string<<std::endl;

  //append
  my_first_string +=  my_second_string;
  std::cout<<"Appended string: "<<my_first_string<<std::endl;

  // extract
  const size_t first{18};
  const size_t last{22};
  string a_word = joined_string.substr(first,last-first+1);
  std::cout<<"Extracting characters "<<first<<"-"<<last<<" from joined string: "<<a_word<<std::endl;

  return 0;
}
```

Find this code at: Prelecture3/cppstring_summary.cpp

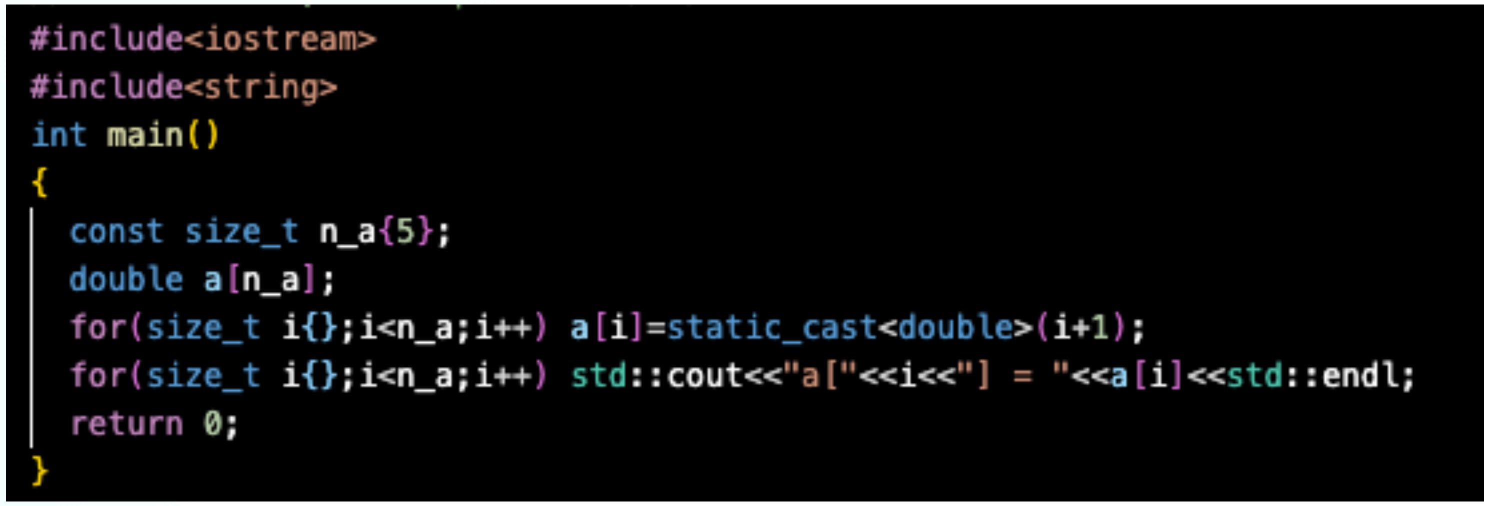
Part 2: arrays and vectors

1.2 c-style arrays

Arrays as lists of numbers

How to define them and use them

- Another crucial feature of a programming language is the use of *arrays*
  - These are generally lists of numbers
- Obvious use: store a set of indexed (consecutive) data
- Usage is straightforward, e.g.



```
#include<iostream>
#include<string>
int main()
{
  const size_t n_a{5};
  double a[n_a];
  for(size_t i{};i<n_a;i++) a[i]=static_cast<double>(i+1);
  for(size_t i{};i<n_a;i++) std::cout<<"a["<<i<<"] = "<<a[i]<<std::endl;
  return 0;
}
```

Find this code at: Prelecture3/carray.cpp

index	**content**
0	45
1	47
2	35
3	21

Part 2: arrays and vectors
1.2 standard library vectors

Arrays as standard library vectors

How to define them and fill them

- C++ provides an extension to do this (and much more): *vectors*
  - These are lists of numbers, but they can be lists of anything
  - Note that those are not necessarily vectors in physics (there are more useful libraries for these)
- Include the vector header

```
#include <vector>
```

- Define the vector, what the vector will store and its size

```
vector<int> my_vector(4);
my_vector[0] = 45;
my_vector[1] = 47;
my_vector[2] = 35;
my_vector[3] = 21;
```

- Other examples of vector initialisation in the code example

index	**content**
0	45
1	47
2	35
3	21

Find this code at: Prelecture3/vector.cpp

MANCHESTER 1824 The University of Manchester

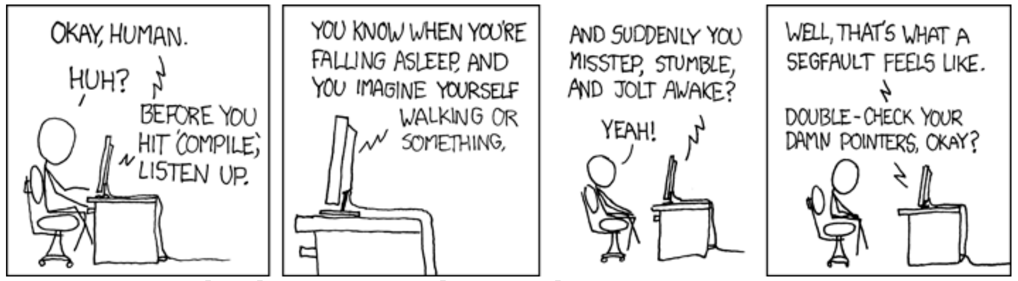
Arrays as standard library vectors

Safety notes

- No one but you can know the length of the vector…but no one prevents you from trying to write outside its bounds!

```
std::vector<int> my_vector(4);
my_vector[0] = 45;
my_vector[1] = 47;
my_vector[2] = 35;
my_vector[3] = 21;
my_vector[4] = 18; //this will compile but it's bad!
```

- Going *out of bounds* on either vectors or arrays is not good
  - It can produce a segmentation fault (=code crashes on runtime)



OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE,' LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING,
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

  - Or it can pick up something random in memory, and the code will run but the output won't make sense

- Important to always check vector/array bounds before reading something from it or adding something to it using []

Arrays as standard library vectors

Useful functions

- A number of useful standard functions are part of the vector class:
  - `empty()`: tests whether the vector is empty, returns a boolean
  - `size()`: returns an integer == length of vector
  - `push_back(..)`: adds an element at the end of a vector, extending it
  - `pop_back(..)`: removes the last element at the end of the vector, shrinking it
  - `clear()`: empties the content of the vector
- Try to add them in your example program and see what they do

Arrays as standard library vectors

Iterators

- Looping over a vector:vector<int> my_vector(4);

```
vector<int> my_vector[0] = 45;
my_vector[0] = 45;
my_vector[1] = 47;
my_vector[2] = 35;
my_vector[3] = 21;
for ((size_t i{}; i<my_vector.size(); ++i)
    std::cout<<"my_vector["<<i<<"] = "<<my_vector[i]<<std::endl;
```

See also: Prelecture3/vector2.cpp

- Iterators are another way to loop over a vector

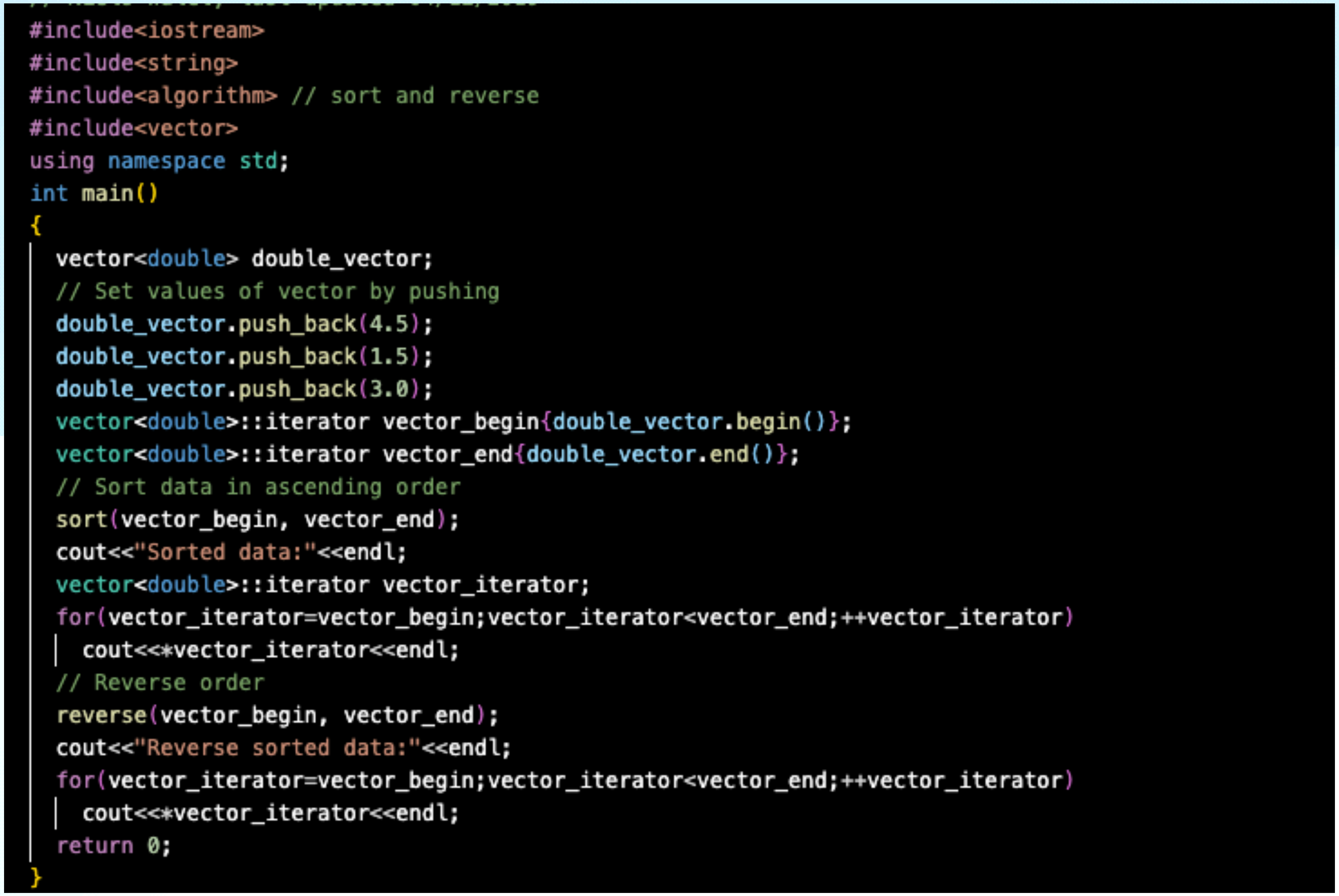
```
std::vector<double >::iterator vector_begin{vector_double.begin()};
std::vector<double >::iterator vector_end{vector_double.end()};
std::vector<double >::iterator vector_iterator;

for(vector_iterator=vector_begin; vector_iterator <vector_end; ++vector_iterator)
    std::cout<<*vector_iterator <<std::endl; //the * is important, we'll see why later
```

See also: Prelecture3/vector3.cpp

- That seems a lot of trouble for something you can do with just an index…but iterators apply to many more objects (e.g. strings) and are used in functions that can e.g. be used to sort the vector

Arrays as standard library vectors

Example of sorting a vector using iterators



```
#include<iostream>
#include<string>
#include<algorithm> // sort and reverse
#include<vector>
using namespace std;
int main()
{
  vector<double> double_vector;
  // Set values of vector by pushing
  double_vector.push_back(4.5);
  double_vector.push_back(1.5);
  double_vector.push_back(3.0);
  vector<double>::iterator vector_begin{double_vector.begin()};
  vector<double>::iterator vector_end{double_vector.end()};
  // Sort data in ascending order
  sort(vector_begin, vector_end);
  cout<<"Sorted data:"<<endl;
  vector<double>::iterator vector_iterator;
  for(vector_iterator=vector_begin;vector_iterator<vector_end;++vector_iterator)
    cout<<*vector_iterator<<endl;
  // Reverse order
  reverse(vector_begin, vector_end);
  cout<<"Reverse sorted data:"<<endl;
  for(vector_iterator=vector_begin;vector_iterator<vector_end;++vector_iterator)
    cout<<*vector_iterator<<endl;
  return 0;
}
```

Prelecture3/vector_4.cpp

Summary

- Today we have learned to use two very useful features of C++
  - **Strings**
    - Store and manipulate groups of **characters**
      - as array of characters, and as strings from the std library
  - **Vectors**
    - Store and manipulate groups of **numbers**
      - as array of numbers, and as vectors from the std library
      - introduction of the *iterator* to perform operations on elements of a vector

Bonus: a preview

And an aOOPpetizer: headers, implementations, interfaces

Strings and vectors are classes

A preview of the next lecture

- Some of you may know (or have guessed) already: `vector` and `string` are examples of *C++ objects*
- Other *objects* we have seen before: `cout`, `cin`, file streams
- Objects are made from elements
    - Data and their organisation: *data members*
    - The operations that can be performed on the objects (e.g. calculate size of a vector): *functions*
- These are collected together within a *class*
- The vector and string classes are pre-defined as part of the C++ Standard Library
- In the next lecture, we will learn how to write our own classes.

MANCHESTER 1824 The University of Manchester

Headers and implementation

We will see more of this in the last few lectures

- For now, enough to know:
  - C++ code is usually split into two different parts, often in different files
    - *Header* (file extension: .h, .hpp) = where things are defined
      - Think of it as the index of a book
      - Usually headers contain interfaces that help you get an overview of what something (e.g a function) does, without the clutter of the full implementation
    - *Implementation* (file extension: .cxx, .cpp) = where things are implemented
      - Think of it as the book content
      - The implementation of a function goes here

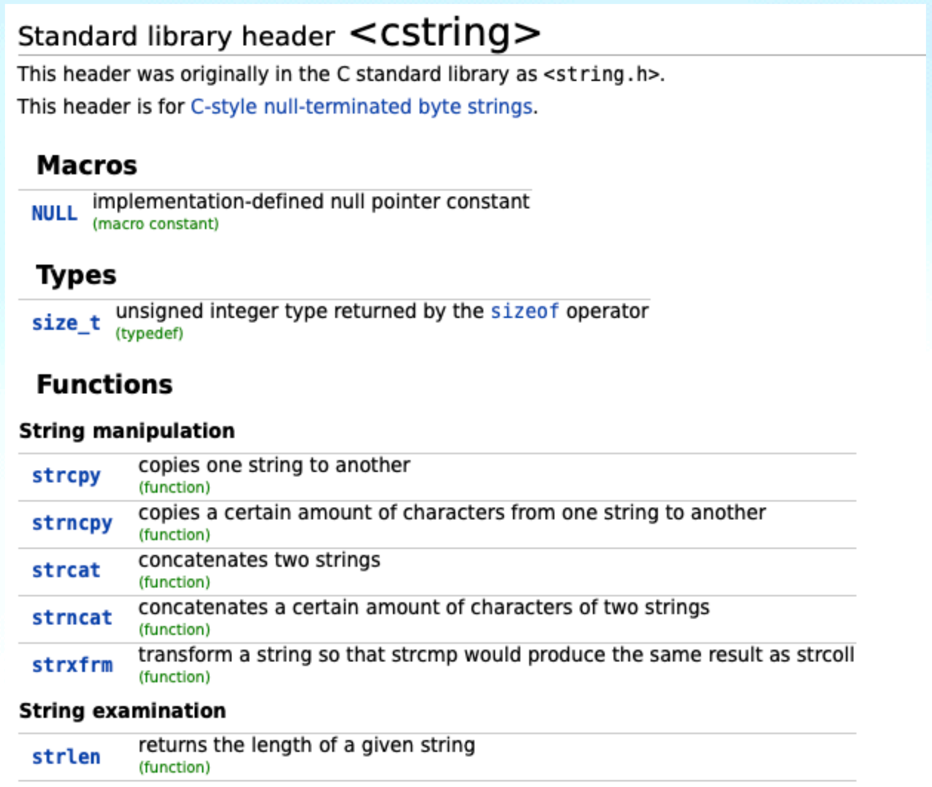
Interfaces are your friends

Where to find information about what a class can do?

- The standard library (as well as many other libraries) is well documented on sites such as https://en.cppreference.com/w/
  - Example from https://en.cppreference.com/w/cpp/header/cstring

Definition of new types in the header

Description of functions



Standard library header <cstring>

This header was originally in the C standard library as `<string.h>`.

This header is for C-style null-terminated byte strings.

Macros

`NULL`	implementation-defined null pointer constant (macro constant)

Types

`size_t`	unsigned integer type returned by the `sizeof` operator (typedef)

Functions

String manipulation

`strcpy`	copies one string to another (function)
`strncpy`	copies a certain amount of characters from one string to another (function)
`strcat`	concatenates two strings (function)
`strncat`	concatenates a certain amount of characters of two strings (function)
`strxfrm`	transform a string so that strcmp would produce the same result as strcoll (function)

String examination

`strlen`	returns the length of a given string (function)

- You can get similar insight by looking at the interface (.h file)
- Websites like cppreference also give you some more information and examples

- Other libraries: https://en.cppreference.com/w/cpp/links/libs
  - These are a lot!
  - You will only need the standard library for this course

MANCHESTER 1824 The University of Manchester