Advanced Programming

Review of C++

Advanced Programming

Review of C++

Some recommended books:

- *Accelerated C++*, A. Koenig and B. E. Moo (slightly outdated but a good approach)

- Books by Bjarne Stroustrup, e.g. *Principles and Practice Using C++* and *The C++ Programming Language 4th Edition* (includes C++11 concepts)

- *Moving Planets Around - An Introduction to N-Body Simulations Applied to Exoplanetary Systems*, J. Roa et al. (how to write an N-body code)

Websites:

- http://en.cppreference.com

- ChatGPT or https://stackoverflow.com/ (general questions)

Advanced Programming

Review of C++

Aim:

- To refresh your knowledge of, or to learn, C++
- Primarily cover basic concepts, with some more advanced
- Led by examples and by what you are likely to need to do as a researcher/developer:
  - Analysing data
  - Using other people’s code

Advanced Programming

Review of C++

- C++ is a language developed in 1979 by Bjarne Stroustrup
- The first C++ ‘standard’ was developed in 1998
- It is an *object-oriented* language (we will unpack this later)
- Changes/additions to the standard are made as new C++ versions appear (C++98, C++03, C++11, C++14, C++17, C++20)
- The current most up-to-date version is C++20, but in practice, actively developed codes are probably using up to C++14/17
- New versions usually provide new advanced features

Advanced Programming

Review of C++

Advantages:

- C++ can be optimised well by modern compilers - it is **fast**
- Object-oriented approach means code is (in theory) easy to read
- The basic features are relatively intuitive

Disadvantages:

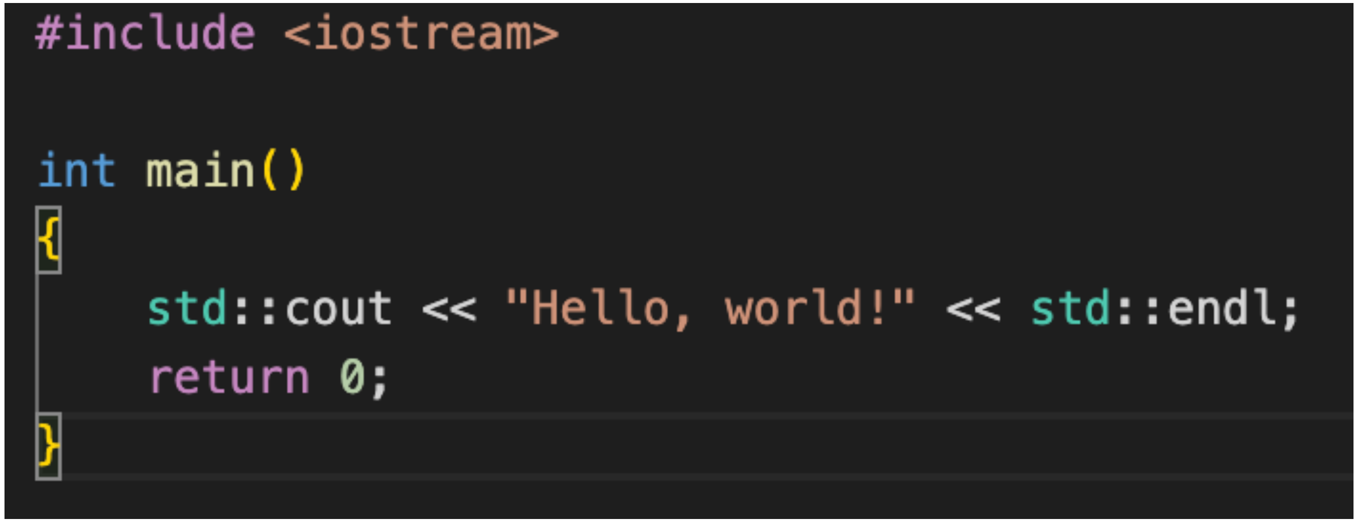
- In large codes, object-orientation can mean you must go down ‘rabbit holes’ to work out how the code works
- Can be complex if you want to implement advanced features

Advanced Programming

Review of C++

- We have written some code to output the text, ‘`Hello, world!`’, to the terminal (using ChatGPT)

- What does this program demonstrate?



```cpp
#include <iostream>

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

Advanced Programming

Review of C++

- C++ is an *object-oriented programming language* - it is organised around ‘objects’ (in contrast to eg. functional programming)

- Each object will have a defined *type* - these can be already built in (`int`, `double` etc.) or user-defined (`class`)

- The type defines permitted operations and a ‘semantic meaning’ to the object

- You will often see ‘an object is an *instance* of a *class’* - but also true for built-in types

- Objects enable code *abstraction* - unnecessary implementation code can be hidden

- Other codes that mainly use object orientation are Python, Java

Advanced Programming

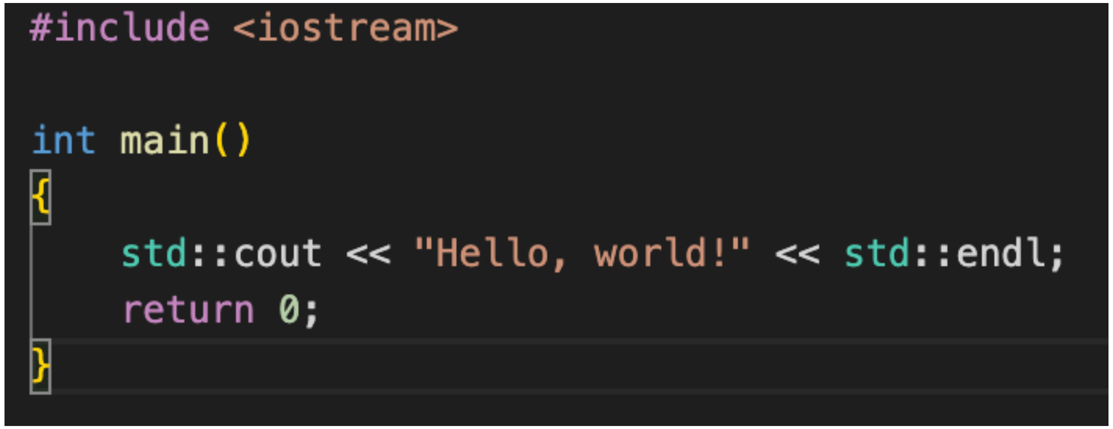
Review of C++

- In hello_world.cpp example, cout is an *object* of the *class* ostream - it is an object of *type* ostream

- Type ostream is defined by the external library, <iostream>

  - https://cplusplus.com/reference/iostream/

- The 0 returned by the main() function is also an object of type int



```
#include <iostream>

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

Advanced Programming

Review of C++

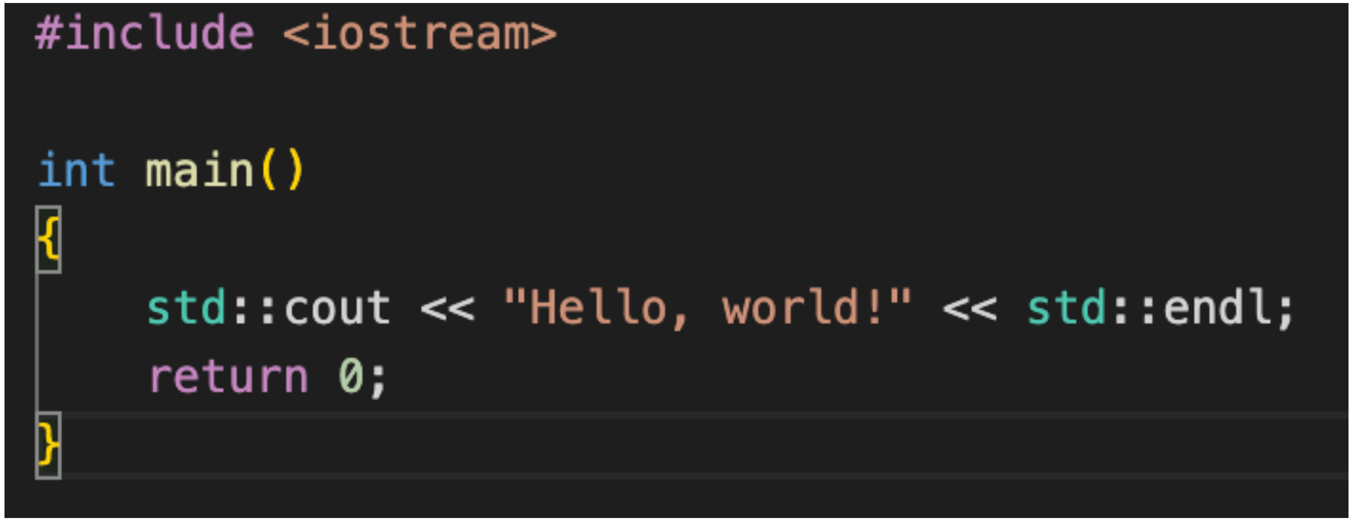
- The most used built-in C++ data types are:
  - `int` - integer (from -2147483648 to 2147483647, 2-4 bytes memory)
  - `float` - floating point (decimals and exponentials, 4 bytes memory)
  - `double` - floating point with double precision (8 bytes)
  - `bool` - boolean (`true` or `false`, used for conditional statements, 1 byte)
  - `void` - valueless - only used for functions that do not return any data
  - `char` - character (1 byte)
  - [`wchar_t` - wide character (character that takes up more than one byte)]

Advanced Programming

Review of C++

In example program:

- `int` (integer) is a built-in type
- `‘Hello, world!’` is made up of several `char`



```cpp
#include <iostream>

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

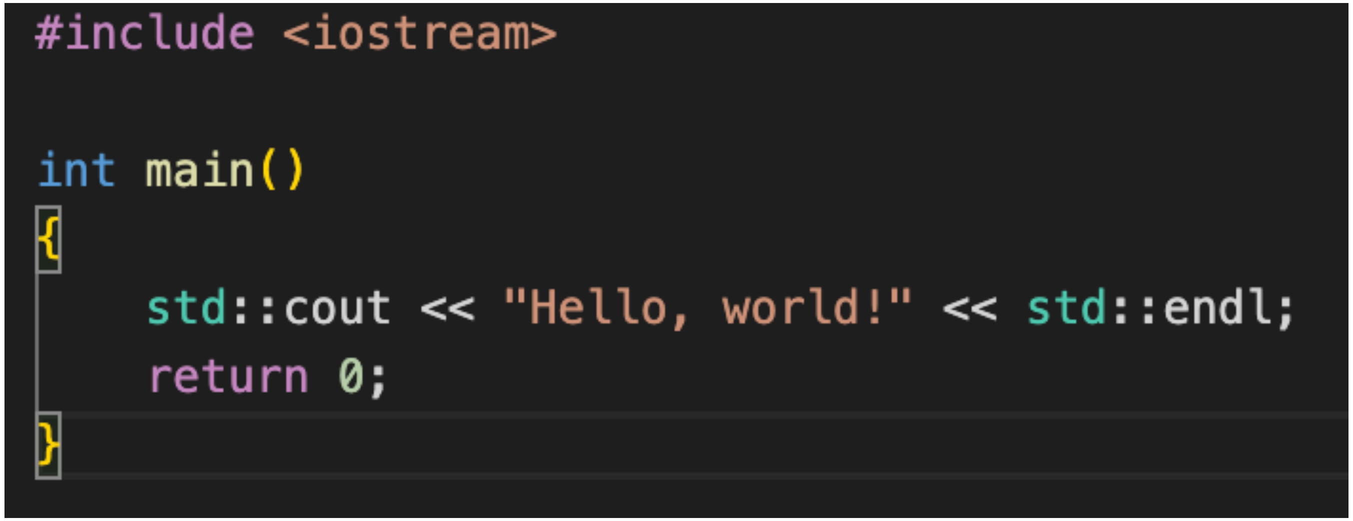
Advanced Programming

Review of C++

- Every C++ program must have a `main()` function that returns `0` when successful

- This is what is called (by the C runtime library initialisation code) when you run the code

- We note the `#include <iostream>` - this is an external library



```cpp
#include <iostream>

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

Advanced Programming

Review of C++

- In the example, `std` refers to the ‘standard *namespace*’
- Namespaces are collections of related names
- `std` is used by standard libraries to contain all the names that it defines, eg. `std::cout`
- If we decided to use a different namespace, we could have another function eg. `otherlib::cout`

Advanced Programming

Review of C++

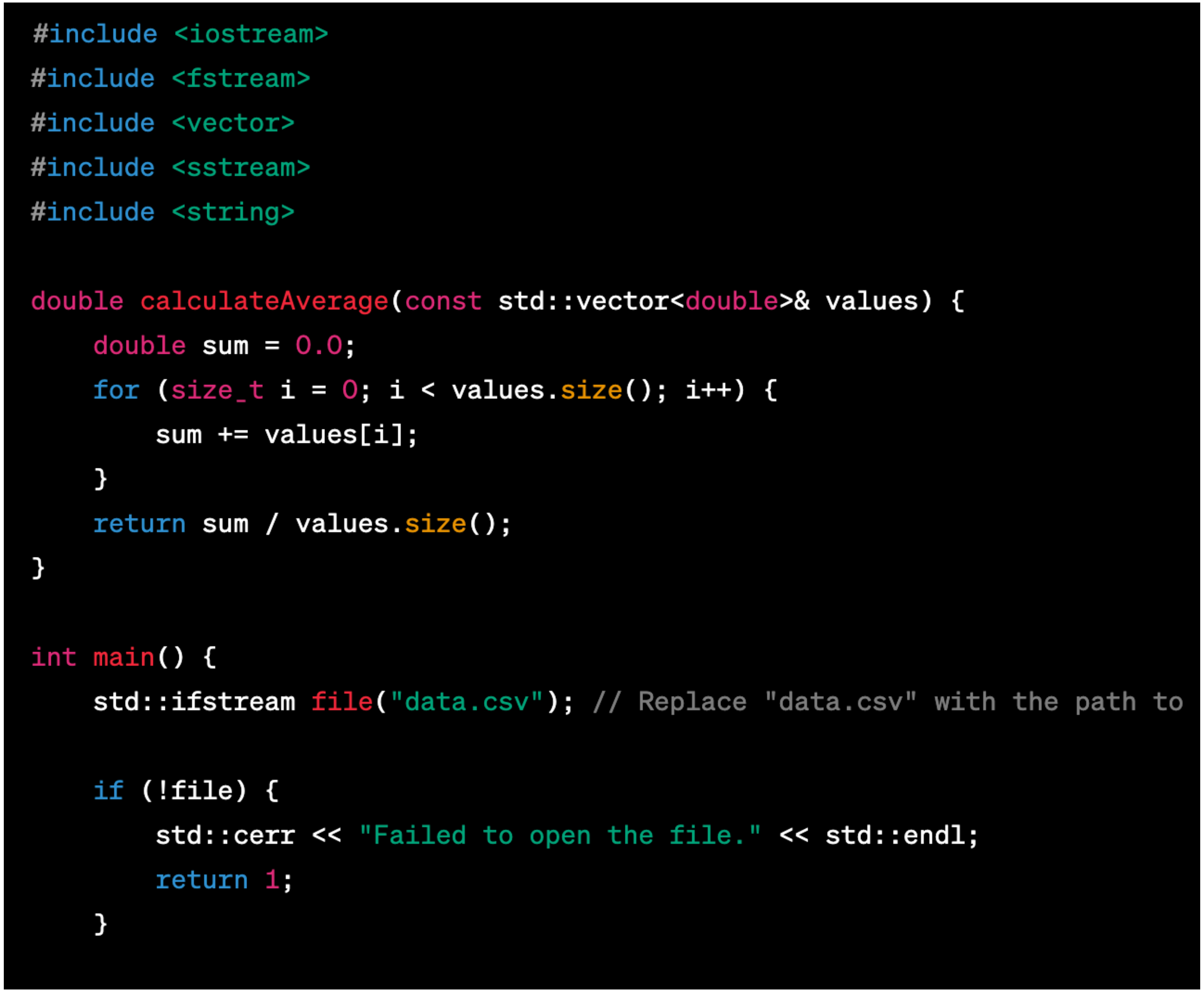
- Now that we know how to output text to the terminal, we might like to write a program for a scientific purpose

- Eg. We would like to analyse some data in a text file, `data.csv`

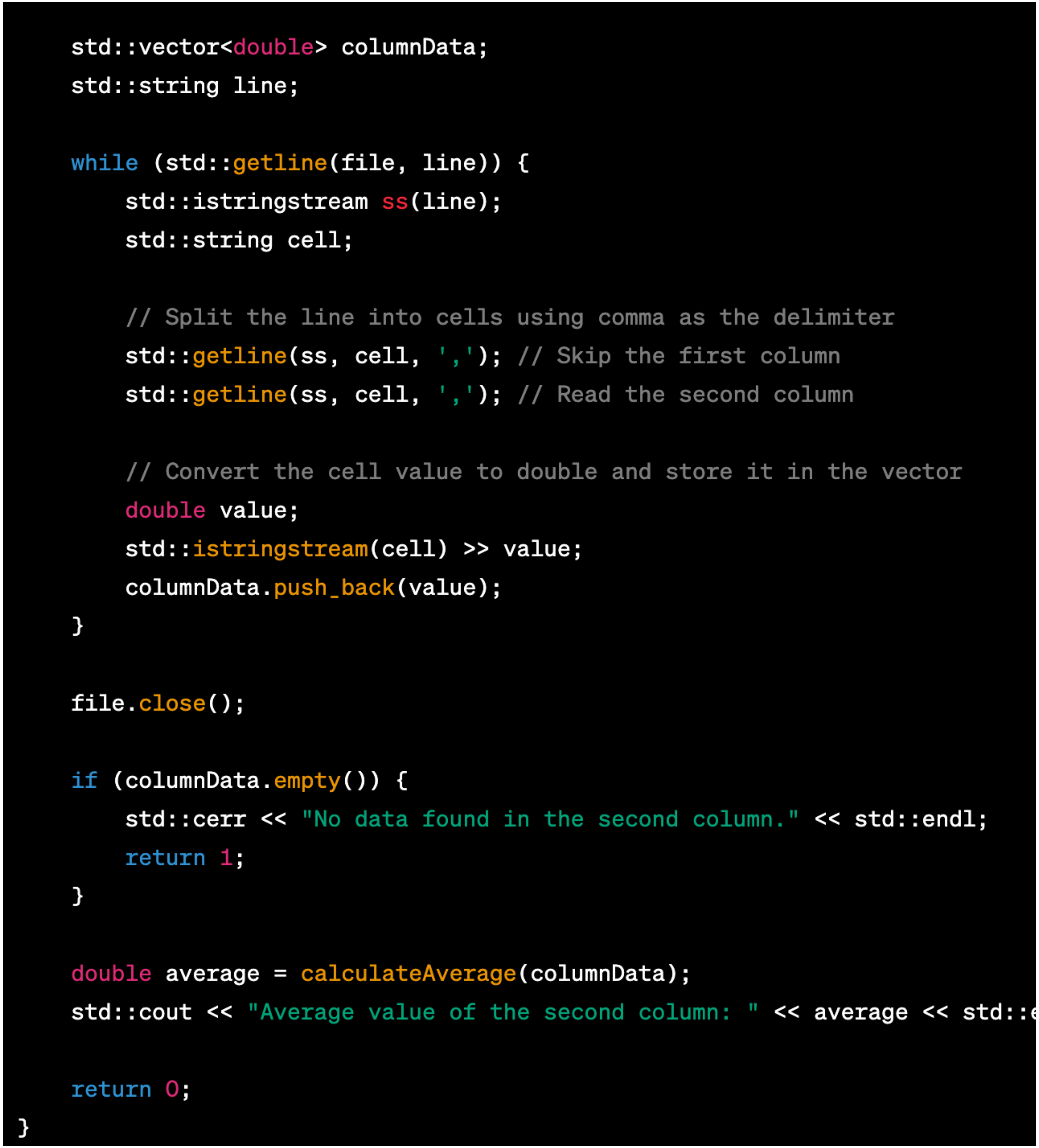
Practical Task: Use ChatGPT to create a C++ program that calculates the average value of the second column of a csv file (comma separated values)

Advanced Programming

Review of C++



```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string>

double calculateAverage(const std::vector<double>& values) {
    double sum = 0.0;
    for (size_t i = 0; i < values.size(); i++) {
        sum += values[i];
    }
    return sum / values.size();
}

int main() {
    std::ifstream file("data.csv"); // Replace "data.csv" with the path to

    if (!file) {
        std::cerr << "Failed to open the file." << std::endl;
        return 1;
    }
```

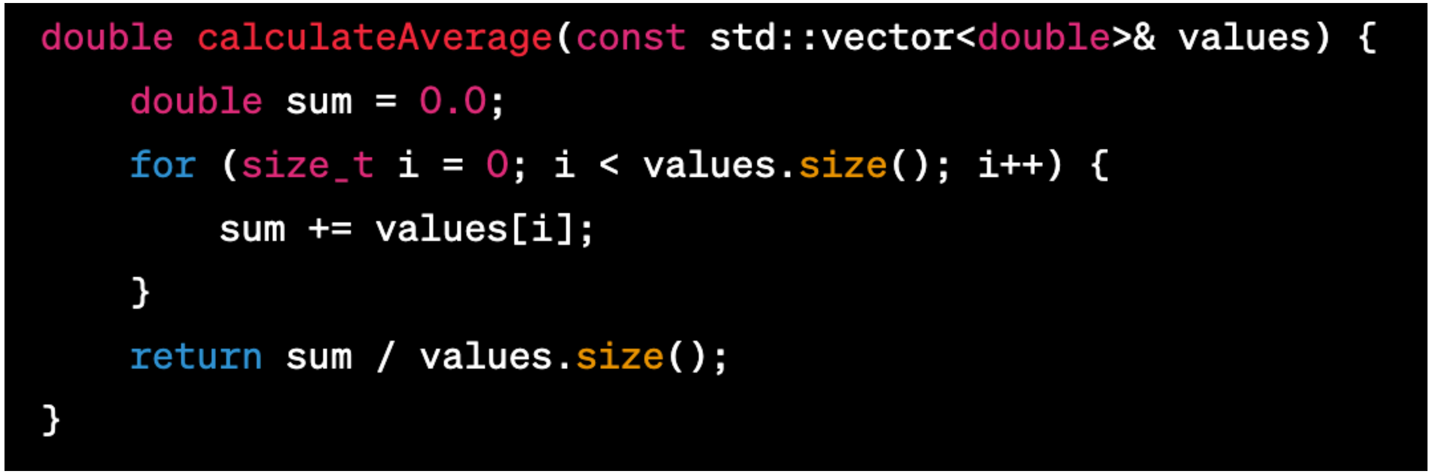


```
    std::vector<double> columnData;
    std::string line;

    while (std::getline(file, line)) {
        std::istringstream ss(line);
        std::string cell;

        // Split the line into cells using comma as the delimiter
        std::getline(ss, cell, ','); // Skip the first column
        std::getline(ss, cell, ','); // Read the second column

        // Convert the cell value to double and store it in the vector
        double value;
        std::istringstream(cell) >> value;
        columnData.push_back(value);
    }

    file.close();

    if (columnData.empty()) {
        std::cerr << "No data found in the second column." << std::endl;
        return 1;
    }

    double average = calculateAverage(columnData);
    std::cout << "Average value of the second column: " << average << std::e

    return 0;
}
```

Advanced Programming

Review of C++

- We have defined a new function, `calculateAverage()`
  - What type of output will this function give?
  - This function is called in `main()` and requires a `vector` input



```
double calculateAverage(const std::vector<double>& values) {
    double sum = 0.0;
    for (size_t i = 0; i < values.size(); i++) {
        sum += values[i];
    }
    return sum / values.size();
}
```

Advanced Programming

Review of C++

- This introduces the concept of a derived data type - a type created by combining built-in data types
  - `function` - a code segment for a specific purpose, saves us from having to duplicate code

    eg. `sometypea calculateAverage(sometypeb parameters) {`

    `//Content of file`

    `}` (types can be the same, but don't have to be)

    called by

    `calculateAverage(myparams)`

Advanced Programming

Review of C++

- `array`
  - Set of elements of the same type kept in memory in a continuous way
  - Allows us to store data with a single variable name, in sequence
  - Need to know the number of elements before you define

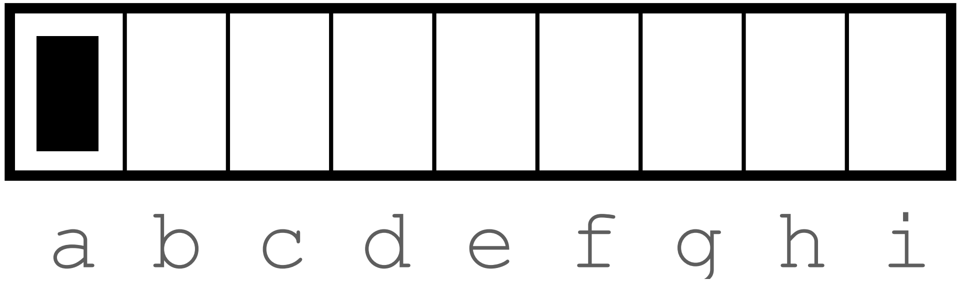
eg. `int time_in_seconds[5] = {0,1,2,3,4};`

Advanced Programming

Review of C++

- `pointer` *
  - a variable that holds the address in memory that another variable is stored
- `reference &` (address-of)

eg. `int  black_box;`
`int* pointer_to_box = &black_box;` (the value is `a`)



Advanced Programming

Review of C++

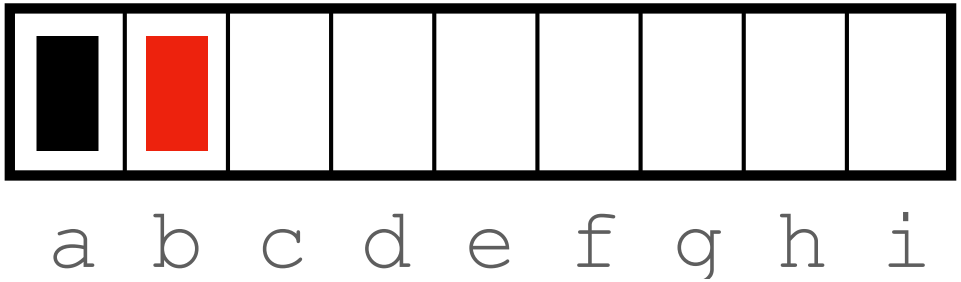
Note, `*` can also be used to *dereference* pointers

eg. `int red_box = *pointer_to_box;`

- Here, `red_box` is an *integer* set equal to ‘the variable pointed to by `pointer_to_box`’

- The new variable will be stored in a new memory address, `b`

- When passed as a function parameter, this is known as ‘passing by value’

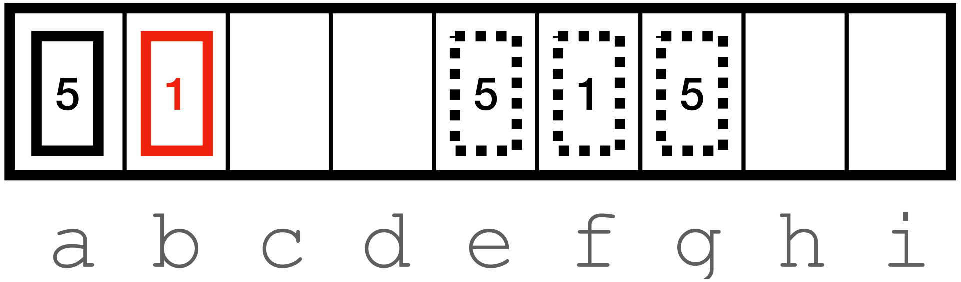


Advanced Programming

Review of C++

Example: what is returned here?

```
void swap(int x, int y) {
    int temp = x; //eg. stored in location e
    x = y;        //eg. stored in location f
    y = temp;     //eg. stored in location g
}
int main() {
    int black_box = 5;
    int red_box = 1;
    swap(black_box, red_box);
    return 0;
}
```



Advanced Programming

Review of C++

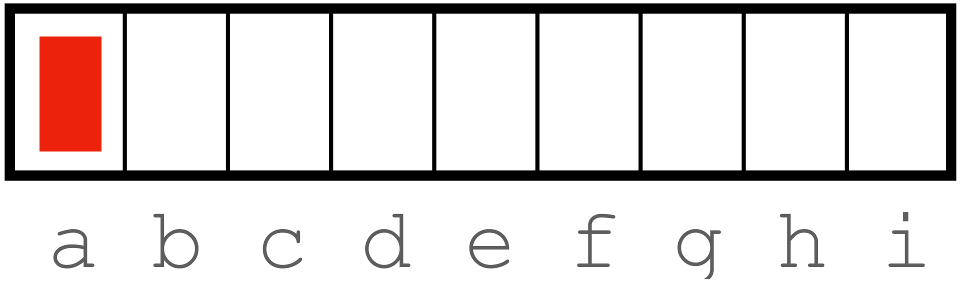
Additionally, `&` can be used to pass references to variables

eg. `int& red_box = black_box;`

- We read this as, ‘the address in memory that points to `red_box` is equal to the address in memory that points to `black_box`’ (which we saw earlier was `a`)

- The variable stored in the location `a` will be effectively aliased by a new variable, pointed to by same memory location

- When passed as a function parameter, this is known as ‘passing by reference’

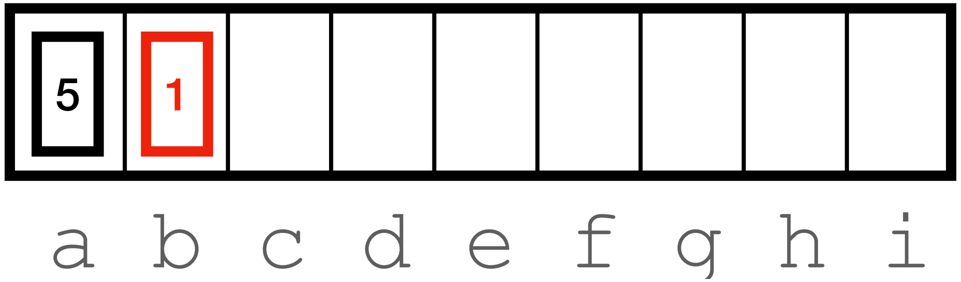


Advanced Programming

Review of C++

Example: what is returned here?

```
void swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
}
int main() {
    int black_box = 5;
    int red_box = 1;
    swap(black_box, red_box);
    return 0;
}
```

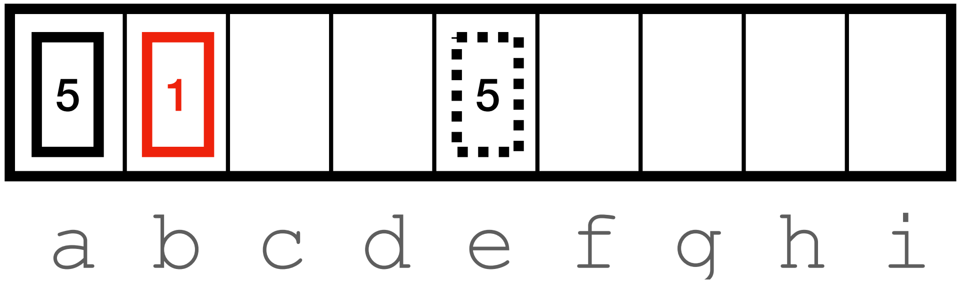


Advanced Programming

Review of C++

Example: what is returned here?

```
void swap(int& x, int& y) {
    int temp = x;
    x = y;         //Read x, y as ‘the variable
    y = temp;        pointed to by this address’
}
int main() {
    int black_box = 5;
    int red_box = 1;
    swap(black_box, red_box);
    return 0;
}
```

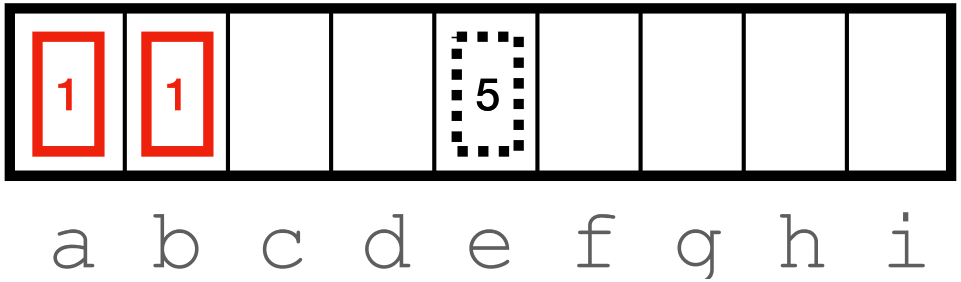


Advanced Programming

Review of C++

Example: what is returned here?

```
void swap(int& x, int& y) {
    int temp = x;
    x = y;            //Read x, y as ‘the variable
    y = temp;           pointed to by this address’
}
int main() {
    int black_box = 5;
    int red_box = 1;
    swap(black_box, red_box);
    return 0;
}
```

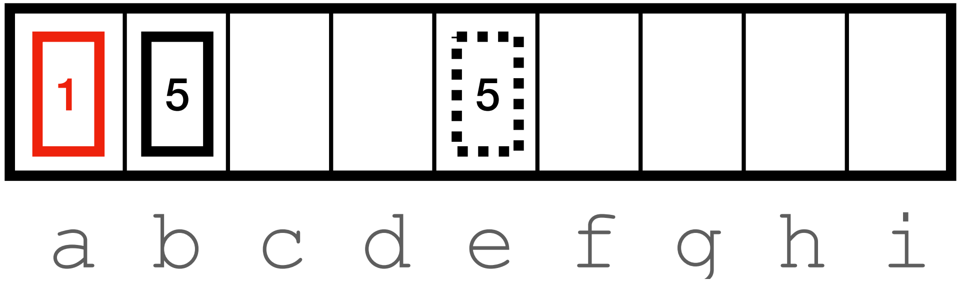


Advanced Programming

Review of C++

Example: what is returned here?

```
void swap(int& x, int& y) {
    int temp = x;
    x = y;         //Read x, y as ‘the variable
    y = temp;        pointed to by this address’
}
int main() {
    int black_box = 5;
    int red_box = 1;
    swap(black_box, red_box);
    return 0;
}
```



Advanced Programming

Review of C++

- Pointers and references in particular can take a lot of repetition to get your head around
- Advise keeping good notes that you can refer to while you are getting familiar with the concept (and for when you forget)
- Passing by reference vs by value becomes *very important* with large datasets
- If you have gigabytes or terabytes of data eg. on a time-evolving 3D simulation grid, you do NOT want to be copying unnecessarily
- Passing by reference is the most memory- and time-efficient, especially for large simulations

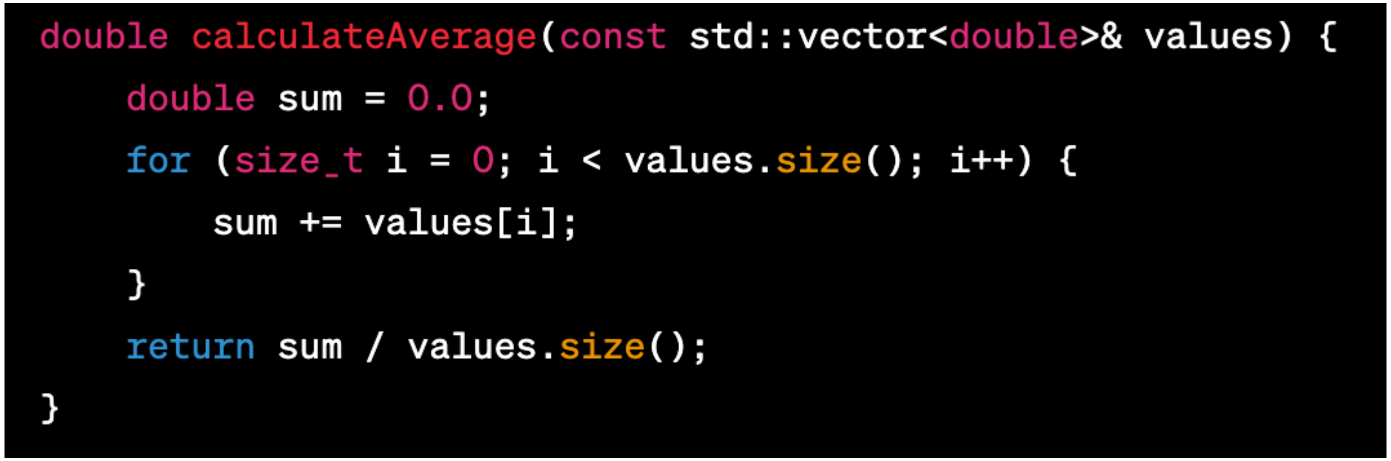
Advanced Programming

Review of C++

- In our example, we see that the function `calculateAverage()` is passed a `vector` *by reference*

- This means that no copies of `values` will be made in memory

- We have also used `const`, which additionally ensures that the values will not be changed



```cpp
double calculateAverage(const std::vector<double>& values) {
    double sum = 0.0;
    for (size_t i = 0; i < values.size(); i++) {
        sum += values[i];
    }
    return sum / values.size();
}
```

Advanced Programming

Review of C++

- `<vector>` is included as an external library (https://en.cppreference.com/w/cpp/container/vector)

- This is a *class template* (see later) for sequence containers that allow for *dynamic size arrays*

- Vectors are useful if you want to store variables, but you do not know how many there will be before you run the code

- Eg. output from a simulation that depends on runtime parameters

- Some particularly useful functions are `size` (in example, this measures the size of the vector), `push_back` (add an element to the end of the vector), `insert`