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 also see a `for` loop
- This is a type of *control flow statement*
- This allows us to iterate a particular operation using an index

```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++

- We define an index `i` (for the purposes of this example, read `size_t` as `int`)
- The index begins at `0`, and the second argument provides the total number of points - here, the loop will iterate from `0` to `values.size() - 1`
- `i++` indicates at the end of each time the statement in `{ }` is executed, `i` is increased by 1, i.e. `i = i+1`
- The statement in `{ }` will be executed until the max value of `i` is reached, then the loop will be exited

```
for (size_t i = 0; i < values.size(); i++) {
    sum += values[i];
}
```

Advanced Programming

Review of C++

- A more concise method is to use a range-based for loop
- This functionality has been available since C++11
- Here, `double value : values` indicates that the `for` loop should be executed for every `value` within the `values` vector

```
for (double value : values) {
    sum += value;
}
```

Advanced Programming

Review of C++

- We can now determine exactly what this function does
  - It is passed the reference to a vector of values
  - It defines a `double sum = 0.0`
  - It iterates through vector, adding the values together
  - The average value is returned

```
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++

```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;
    }
```

Advanced Programming

Review of C++

- The file is read in the `main()` function
- First, `std::ifstream file("data.csv")` opens file (input file stream)
- Second part is an error message - these are very important for smooth debugging

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

Advanced Programming

Review of C++

- `if` is a conditional statement
  - Checks whether a given statement is true
  - If yes, executes the commands in `{ }`
- Often combined with an `else` statement with what to do if the statement is *not* true
- `if (!file)` checks if the file was *not* found, `{ }` outputs an error message

```
    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 another type of control flow statement, a `while` loop
- These loop through a block of code contained in `{ }` as long as the condition specified is `true`

```
while (std::getline(file, line)) {
```

- Here, we use the `std::getline` function (https://cplusplus.com/reference/string/string/getline/) to extract characters from `file` and store them as a string in `line`
- `while` loop is executed while there is still a row of data in the file

Advanced Programming

Review of C++

- `std::istringstream` reads a string to a stream
- `std::getline` is also used with a comma delimiter to read the value of the second column in each row
- Values are appended onto the vector `columnData` using `push_back`
- File is closed with `file.close()`

Practical Task: Compile and run the `data_reader.cpp` example

Advanced Programming

Review of C++

- We have now been introduced to some key C++ concepts
  - Defining additional functions
  - Passing by value/reference
  - Control statements: `if, else, for, while`
  - Opening and reading data files
  - Manipulating strings and streams
  - Using vectors
  - Writing error messages

Advanced Programming

Review of C++

- What would we do to `data_reader.cpp` if we wanted to calculate the same average, but for a different data file?
- Using what we currently know, we would either have to:
  - Write all the same code out again with a different file name
  - Recompile the code with a different file name and run again
  - Change the name of our data file to fit with the name in the code
- Might work for a few files, but not straightforward with many files
- A more efficient option is to use a more advanced, central C++ concept: a *class* (a user-defined data type)
- We can define a class that takes a file name as a *parameter*

Advanced Programming

Review of C++

- Looking at the new `main()` function in `data_reader_class.cpp`, we see an unfamiliar data type - `CSVReader`

- This is a user-defined *class*

```
int main() {
    std::string filename = "data.csv"; // Replace "data.csv" with the path t

    CSVReader reader(filename);
    if (!reader.readData()) {
        return 1;
    }

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

    return 0;
}
```

```cpp
class CSVReader {
private:
    std::string filename;
    std::vector<double> columnData;

public:
    CSVReader(const std::string& filename) : filename(filename) {}

    bool readData() {
        std::ifstream file(filename);
        if (!file) {
            std::cerr << "Failed to open the file." << std::endl;
            return false;
        }

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

            // Skip the first column
            if (std::getline(ss, cell, ',') && std::getline(ss, cell, ',')) {
                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 false;
        }

        return true;
```

Advanced Programming

Review of C++

- First, focus on the section underneath keyword `public`
- This defines functions that we can access from outside the class - here, we have `readData()` and `calculateAverage()`
- These are called in `main()` by:

```
CSVReader reader(filename);
if (!reader.readData()) {
    return 1;
}

double average = reader.calculateAverage();
```

- Any function defined within a class is called a *member function*

Advanced Programming

Review of C++

- At the top of the section `public`, we also have

```
public:
    CSVReader(const std::string& filename) : filename(filename) {}
```

- This is called the *constructor*
- The name of the constructor, `CSVReader`, must match the name of the class itself
- `(const std::string& filename)` specifies the parameters that the constructor takes

Advanced Programming

Review of C++

- At the top of the section `public`, we also have

```
public:
    CSVReader(const std::string& filename) : filename(filename) {}
```

- This is called the *constructor*
- The name of the constructor, `CSVReader`, must match the name of the class itself
- `(const std::string& filename)` specifies the parameters that the constructor takes
- The member variables must be initialised - this is done here by `: filename(filename)`
- Eg. Here, if we did not initialise, `reader.readData()` would return an error

Advanced Programming

Review of C++

- In the section underneath keyword `private`, we see the declaration of two variables

```
class CSVReader {
private:
    std::string filename;
    std::vector<double> columnData;
```

- These are used by the functions in `public`, but *cannot* be accessed outside the class - eg. `reader.columnData` is not valid

- Any variable defined within a class is called a *member variable*

Advanced Programming

Review of C++

- Another useful property of classes is *inheritance*
- This allows a class to *inherit* (access) public members from a base class

```
class CSVReader {
public:
    // Public members and functions of the CSVReader
};

class HigherLevelReader : public CSVReader {
    // Additional members and functions specific to t
};

int main() {
    HigherLevelReader obj;

    // Accessing public members from the base class
    obj.somePublicMemberOfCSVReader;
    obj.somePublicFunctionOfCSVReader();

    return 0;
}
```

Advanced Programming

Review of C++

- One specific kind of class is a `friend class`
- These can access *private and protected* members of other classes in which it is declared as a friend
- This is not possible via inheritance

Advanced Programming

Review of C++

```cpp
class CSVReader {
private:
    int privateData;

public:
    CSVReader(int data) : privateData(data) {}

    friend class CSVReaderFriend;
};

class CSVReaderFriend {
public:
    void accessPrivateData(const CSVReader& obj) {
        int data = obj.privateData;  // Friend class
        // Perform operations using private data
    }
};

int main() {
    CSVReader obj(42);
    CSVReaderFriend friendObj;

    friendObj.accessPrivateData(obj);  // Friend clas

    return 0;
}
```

Advanced Programming

Review of C++

- Note: there is another user-defined data type - `struct`
- All members are `public` by default, vs `private` by default with classes
- They are defined similarly to classes, eg.

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

// Define a struct for representing a student
struct Student {
    int id;
    string name;
    int age;
};
```

- In practise, classes are used much more often

Advanced Programming

Review of C++

- Another key concept in C++ is *templating*
- This allows us to define a function *without having to specify the type of the variables it acts on*
- Some example syntax:

```
// Templated function that swaps two values of any type
template <typename T>
void swapValues(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}
```

- Note that this is the same syntax used for `vector` - these are class templates

Advanced Programming

Review of C++

Some miscellaneous advanced features:

- When defining the `main()` function, we will often see the definition

  ```
  int main(int argc, char *argv[]) {}
  ```

- This is the method used to pass *command line arguments* to `main()` - whatever you put on the command line will become an input
- `argc` - argument count (number of strings pointed to by `argv`)
- `argv` - argument vector (these names are convention, can be given other names)
- Run programs like this using eg. `./Hello_World param1 param2`
- `::` denotes the *scope* (as we have seen before, in i.e. the namespace)
- This also applies to *classes*, eg. `classname::functionname`

Advanced Programming

Review of C++

Now that we understand a lot of C++ key features, how do we navigate a large, complex research code?

- Will use the example of adaptive mesh refinement numerical relativity code, GRChombo (https://github.com/GRChombo/GRChombo)
- Other codes large codes include eg. CosmoLattice, SOFTSUSY, N-Body code GADGET
- In short:
  - Read any documentation
  - Make use of example code
  - Find `main()` and follow functions from there