UNIVERSITY
OF LONDON

london.ac.uk

Senate House, Malet Street,
London WC1E 7HU

UNIVERSITY OF LONDON

# Object Orientated Programming

## File I/O, exception handling and algorithms

Date: 9th May 2025

london.ac.uk

# File I/O, exception handling and algorithms

- Introduction
- Algorithms & Pseudocode
- File I/O
- Errors & exception handling
- Questions
- Wrap



UNIVERSITY OF LONDON

# Introduction: About Me

Name: Toby Brodie

Where I am based: I am based in Southend-on-Sea, England.

Background: 14 years at London Universities lecturing in Computer Studies

What my roles are: Supporting you. Marking. Webinars

Contact: Through *new posts* on the Group 5 forum

# Introduction: Webinar plan

- **Algorithms & Pseudocode**

  We will look at how to turn a simple program specification into a step-by-step algorithm, written in pseudocode, which allows us, in turn, to create a program in C++

- **File I/O**

  Our program will require both reading and writing to text files.

  We will consider different modes that should be used

- **Errors & exception handling**

  We will consider possible faults in the program design

  and how to handle user errors

# Algorithm & Pseudocode definitions

- **Algorithm**

  An algorithm is a step-by-step procedure developed to solve a problem.

- **Pseudocode**

  Pseudocode is a textual representation of an algorithm in a human-readable form.

  **Algorithms provide a systematic and logical approach to solving problems, whereas pseudocode serves as a method to express algorithms in a simplified manner.**

Algorithm vs. Pseudocode: What's the Difference? www.difference.wiki/algorithm-vs-pseudocode/

# Pseudocode – a reminder

- Pseudocode serves as design notation that is independent of any specific programming language.
- It is **not** a natural language such as written English, nor does it represent formal code utilized in high-level programming languages.
- This notation enables programmers to use indentation to effectively illustrate programming constructs, including iteration (loops) and selection statements (IF or CASE).
- Generally, we create pseudocode from a specification derived from analysing the problem that needs to be addressed.

# Pseudocode – do it right!

- Define the steps of the main algorithm
- Refine the main steps where possible (not all main steps may need refining)
- There is no need to declare variables
- Indentation should be used to help identify the use of loops and selection statements
- **Do not** write steps using a formal language such as C++,  Java, Python, etc.
- Pseudocode should be able to be converted to most coding languages.

# Our example program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

# Functional requirements

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – step 1

1. Initialise



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – step 2/3

1. Initialise
2. Start fixed loop for twenty pupils
3. End fixed loop



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – 2.1

1. Initialise
2. Start fixed loop for twenty pupils
    21. Get forename and surname for user
3. End fixed loop



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – step 2.2

1. Initialise
2. Start fixed loop for twenty pupils
    21. Get forename and surname for user
    22. Get valid age for the user
3. End fixed loop



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – step 2.3

1. Initialise
2. Start fixed loop for twenty pupils
    21. Get forename and surname for user
    22. Get valid age for the user
    23. Generate a username
3. End fixed loop



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – step 2.4

1. Initialise
2. Start fixed loop for twenty pupils
    21. Get forename and surname for user
    22. Get valid age for the user
    23. Generate a username
    24. Display "Username", index, "is" username
3. End fixed loop



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – complete

1. Initialise
2. Start fixed loop for twenty pupils
    21. Get forename and surname for user
    22. Get valid age for the user
    23. Generate a username
    24. Display "Username", index, "is" username
3. End fixed loop

# Refinement

- In what ways could the pseudocode be refined?



UNIVERSITY OF LONDON

# Program specification

- A program is to be developed to create usernames for a class of twenty pupils
- The program will ask a teacher to enter the first name, surname, and age of each pupil
- The age entered must be between five and eighteen
- The program should output a list of usernames for the teacher

| Inputs | Processes | Outputs |
|---|---|---|
| Pupil first name | Validate age | List of usernames |
| Pupil surname | Create username | |
| Pupil age | | |

# Pseudocode – complete

1. Initialise
2. Start fixed loop for twenty pupils
    21. Get forename and surname for user
    22. Get valid age for the user
    23. Generate a username
    24. Display "Username", index, "is" username
3. End fixed loop

# Refinement

- In what ways could the pseudocode be refined?
- Additional steps for the validation of age is one possibility
- Username generation steps is another

# C++ (no age validation)

```cpp
#include <iostream>
#include <string>

int main() {
   std::cout << "Pupil Username Program" << std::endl;
   std::cout << "=======================" << std::endl;
   int student_counter = 1;
   while (student_counter <= 20) {
        std::string forename;
        std::string surname;
        int age;
        std::cout << "Enter pupil " << student_counter << " firstname: ";
        std::cin >> forename;
        std::cout << "Enter pupil " << student_counter << " lastname: ";
        std::cin >> surname;
        std::cout << "Enter pupil " << student_counter << " age: ";
       std::cin >> age;
       std::string username = "User" + std::to_string(student_counter);
        std::cout << "Pupil " << student_counter << " username: " << username << std::endl;
        ++student_counter;
   }
   return 0;
}
```

# C++ age validation function

```
static int get_age() {
    int age = 0; // Initialise the variable to avoid uninitialised memory usage
    std::cout << "Enter pupil age: ";
    bool valid_input = false;
    while (!valid_input) {
        std::cin >> age; // Read input from the user
        if (age < 5 || age > 18) {
            std::cout << "Age must be between 5 and 18. Enter a valid age: ";
        }
       else
       {
           valid_input = true; // Valid input received
       }
    }
    return age;
}
```



UNIVERSITY OF LONDON

# Programming considerations

Consider the previous program – what are some possible issues?

# Programming considerations

Consider the previous program – what are some possible issues?

1. What if the users are not all added at the same time?
2. What happens if the forename or surname is left blank?
3. What if the user enters a non-integer value for age?

# Adding persistence to our program

- We can use data files to add persistence to our program
- We can write each user's data to a file, including the counter number
- When the program starts, we can read the last line of the file to determine the last counter value
- We use the standard library's fstream to access files

```
#include <fstream>
```

- We can use the ifstream class for reading from files
- We can use the ofstream class for writing to files



UNIVERSITY OF LONDON

# Open files for reading - 1

- Include fstream at the top of your file

```
#include <fstream>
```

- Create an input file stream object

```
std::ifstream userChosenStreamName{}
```

- You need to use the correct constructor

std::basic_ifstream<CharT,Traits>::basic_ifstream - cppreference.com

- Just adding the filename will add the default mode (in), so the following will work

```
std::ifstream userChosenStreamName{"filename.ext"}
```

# Open files for reading 2

- Make sure you use the correct path to the file
- Always check that the file has successfully opened

```
if (userChosenStreamName.is_open())
```

- Consider what to do if opening fails with an else block!
- Close the file once you have finished accessing it – this will save on memory.
- Other considerations include whether the file exists and whether file permissions allow access.

# Reading lines

- To read a line of a file, we need a variable to store that line so declare this first:

```
std::string line_read;
```

- We use the `getline()` function to read the line and store it into the variable as below:

```
std::getline(userChosenStreamName, line_read)
```

- We can read the entire file line by line by using a while loop
- Reading files line by line is optimal, as reading an entire file's contents could cause memory issues.
- `while (std::getline(userChosenStreamName, line_read))`

# Reading the last line

- We read the entire file line by line by using a while loop

```
while (std::getline(userChosenStreamName, line_read))
```

- The last iteration of the while loop will be the last line
- The variable `line_read` variable will store the content of that line
- If using this technique, it is best to check that each line is not empty before overwriting the data in the `line` variable

# Open files for writing 1

- Include fstream at the top of your file

```
#include <fstream>
```

- Create an output file stream object

```
std::ofstream userChosenStreamName{}
```

- You need to use the correct constructor

std::basic_ofstream<CharT,Traits>::basic_ofstream - cppreference.com

- Just adding the filename will add the default mode (out)

```
std::ofstream userChosenStreamName{"filename.ext"}
```

- This mode will overwrite any file content or create a new file if one does not exist

# Open files for writing – appending data to a file

- You need to use the correct constructor

  std::basic_ofstream<CharT,Traits>::basic_ofstream - cppreference.com

- To append to a file, we add a second argument to the constructor, which instructs it to change the mode to append

```
std::ofstream chosenStreamName{ "filename.ext", std::ios_base::app };
```

- To write content, we use the following syntax:

```
chosenStreamName << content_in_variable ;
```

- This mode will append to the selected file or create a new file if one does not exist

# Back to our program

- When the program starts, see if a file exists with pupil data
- If the file  exists, open the file for reading
- Read the last line and retrieve the counter value
- Close the file
- If no value is retrieved, start the counter at 1; else, use the value
- While inside a loop,
  - Open the file to write data (*on the first iteration, the program will create a file if it does not exist*)
  - Write each pupil's data to the file
  - Close the file
  - Allow the user to exit the program using a sentinel value

# Understanding Errors

- **Syntax errors** – where the coder incorrectly writes code (such as forgetting to add parentheses when they are needed)
- **Logical errors** – where the coder incorrectly uses the wrong formula (such as using an addition instead of a multiplication symbol in a calculation)
- **Linker errors** – where there is an issue with files not being compiled correctly
- **Runtime errors** - where an error occurs during the execution of a program after it has successfully compiled

# Understanding Errors

- **Syntax errors** are typically identified by the IDE; these errors prevent the code from compiling. There are rare instances where the code compiles but behaves unexpectedly during execution
- **Logical errors** are often obvious, but expected results should be compared to the output of a program. Unit testing is applied to check for these errors
- **Linker errors** will be made apparent when compiling
- **Runtime errors** can occur when dealing with any external data, including reading from files and user input. Additional code should be written to validate data and to 'gracefully' deal with bad data. Testing is paramount

# Back to our program

- Data comes in from the user via the keyboard, What data input could cause runtime errors?

- Could entering bad data into the forename variable cause runtime errors?
- Could entering bad data into the surname variable cause runtime errors?
- Could entering bad data into the age variable cause runtime errors?
- Can leaving any of the above empty lead to errors?

# Back to our program

- Could entering bad data into the forename variable cause runtime errors?

  **No,** but we should probably trim the data.

- Could entering bad data into the surname variable cause runtime errors?

  **No,** but we should probably trim the data.

- Could entering bad data into the age variable cause runtime errors?

  **Yes,** the variable age expects an integer value

- Can leaving any of the above empty lead to errors?

  **No**, this is not possible

# Trinm function example

```
// Function to trim leading and trailing whitespace from a string

std::string CSVReader::trim(std::string str)
{
    int first = str.find_first_not_of(' ');
    int last = str.find_last_not_of(' ');
    return str.substr(first, (last - first + 1));
}
```

# Checking for an integer

- There is a built-in method for the cin object, fail() which can be used to check that the last input succeeded;

- Declare an integer variable with a default value: `int input = 0;`
- Declare a Boolean and set it to false: `bool valid = false;`
- Start a while loop using (not) `valid` as the condition: `while (!valid)`
  - Request input from the user
  - **if:** the user enters invalid input (e.g., a non-integer value), **std::cin.fail()** will return **true**.
    - Display an error message
    - **std::cin.clear()** is called to reset the error flags on the input stream.
    - **std::cin.ignore(INT_MAX, '\n')** discards any remaining invalid input in the stream (up to the next newline).
    - The loop continues, prompting the user again.
  - **else:**
    - A valid input was entered, the Boolean value is updated to true, and the loop exits

# Checking for an integer – a c++ example

```
int get_input(int counter) {
  int input = 0;
  bool valid = false;
  while (!valid) { // repeat as long as the input is not valid
    std::cout << "Enter pupil " << counter << " age: ";
    std::cin >> input;
    if (std::cin.fail()) {
        std::cout << "Wrong input! " << std::endl;
        std::cin.clear(); // clear error flags
        std::cin.ignore(INT_MAX, '\n'); // Wrong input remains on the stream, so get rid
        continue;
     }
     valid = true;
  }
  return input;
}
```



UNIVERSITY OF LONDON

# Exception handling

- Exceptions are unexpected problems or errors that occur while a program is running
- The process of dealing with exceptions is known as exception handling
- We can code our programs to handle these exceptions gracefully, so they keep running without errors
- C++ provides an inbuilt feature for handling exceptions using try and catch blocks
- The code that may cause an exception is placed inside the try block and the code that handles the exception is placed inside the catch block.

# Exception handling example

```
#include <iostream>
#include <string>

int main()
{
    std::string number = "Ten";
    try {
      double d_num = std::stod(number);
    }
    catch (const std::exception& e) {
      std::cout << "Exception: " << e.what() << std::endl;
    }
}
```



UNIVERSITY OF LONDON

# Catching the exception

Within the catch block, there are several options:

- Ignore the error and move on
- Add a default value
- Let the program crash – this can be useful during development, but not so much in a release
- Throw the error up to a higher call, where it can be caught again and dealt with

```
throw std::runtime_error("Error thrown!");
```

- Whichever method is used, whilst developing, it is a good idea to output the method and class name where the exception happened

```
std::cout << "***** Class name::method ******" << std::endl;
```

# Exception examples

- See examples in Visual Studio

# Question time

- Any questions?
- I will pass these questions along and post answers in the group forum

# Wrap up

- You should now know:
  - About me and how to contact me
  - How to turn pseudocode into code
  - How to read from files
  - How to append data to files
  - A little about error handling and dealing with exceptions

UNIVERSITY
OF LONDON

london.ac.uk

Senate House, Malet Street,
London WC1E 7HU