

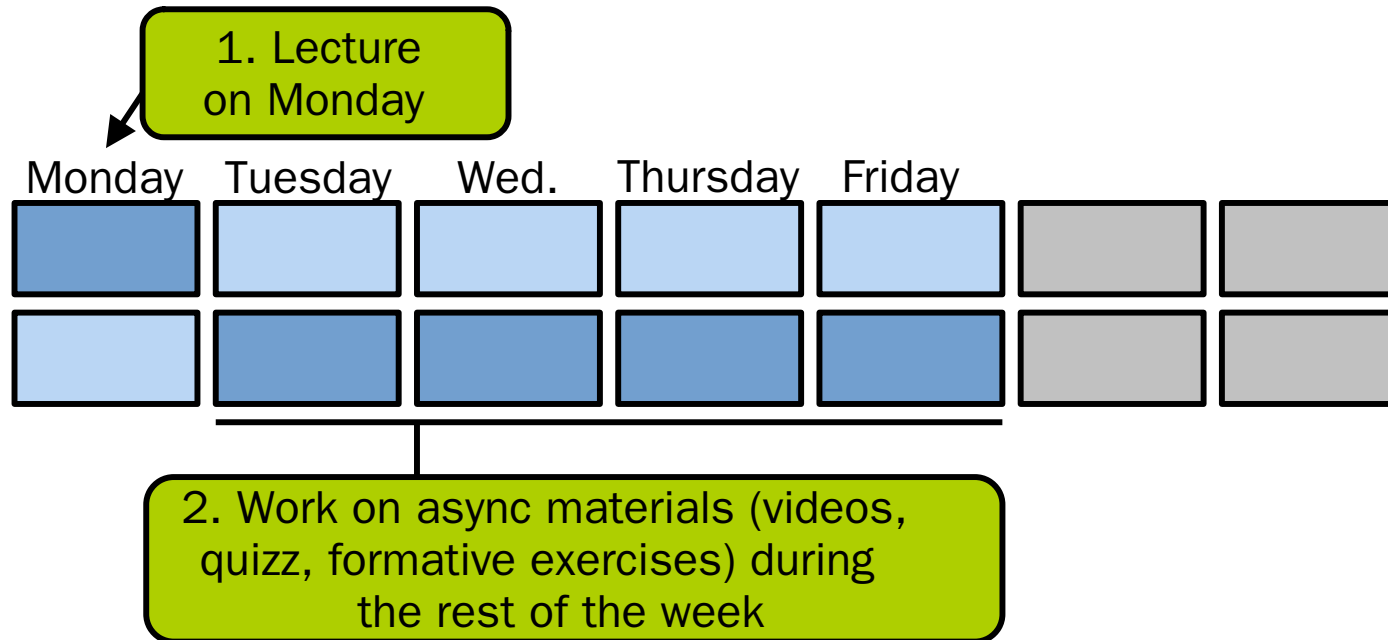
# COMP26020 Programming Languages and Paradigms Part 1: C Programming

---

## Logistics

# Part 1/C Organisation

**Blended approach:** live sessions + asynchronous videos & exercises



# Part 1/C Organisation

## 1. Lecture materials:

- Live lecture every Monday 12pm-1pm, located in *Simon Building Lecture Theatre B*
- A set of videos to watch every week **after the lecture**

# Part 1/C Organisation

## 1. Lecture materials:

- Live lecture every Monday 12pm-1pm, located in *Simon Building Lecture Theatre B*
- A set of videos to watch every week **after the lecture**

## 2. **Formative assessment** (not graded), every week:

- A quiz (theory)
- Autocorrected programming exercises (practice)

# Part 1/C Organisation

## 1. Lecture materials:

- Live lecture every Monday 12pm-1pm, located in *Simon Building Lecture Theatre B*
- A set of videos to watch every week **after the lecture**

## 2. **Formative assessment** (not graded), every week:

- A quiz (theory)
- Autocorrected programming exercises (practice)

## 3. **Summative assessment** (marked): 1 lab exercise, to be done on your free time at home and/or on the Department's machines

# Part 1/C Organisation

## 1. Lecture materials:

- Live lecture every Monday 12pm-1pm, located in *Simon Building Lecture Theatre B*
- A set of videos to watch every week **after the lecture**

## 2. Formative assessment (not graded), every week:

- A quiz (theory)
- Autocorrected programming exercises (practice)

## 3. Summative assessment (marked): 1 lab exercise, to be done on your free time at home and/or on the Department's machines

## 4. Support sessions


- Come if you have any questions about the lecture materials and formative/summative exercises
- Every other week starting week of 30/09, Wednesday 11am-12pm and Thursday 9am-10am in *Kilburn 1.8*

# Course Website

On Blackboard: <https://bit.ly/3B70LCE>

- **Everything regarding part 1/C can be accessed from there:**
  - Schedule (what to do each week)
  - Videos, including live sessions recordings
  - Lecture notes and lecture slides
  - Summative lab exercise brief
  - Formative autocorrected programming exercises
  - Formative quizzes
  - Discussion board
  - Reading list


# Required Software

All programming exercises should be done in a Linux  x86-64 (i.e. Intel CPU) environment with GCC 10/11/12. Several solutions:

- **Use the Department's machines**
- **Run Linux in an x86-64 VM**
  - Grab a VirtualBox VM image here: <https://bit.ly/3UGN45i>




# Required Software

All programming exercises should be done in a Linux  x86-64 (i.e. Intel CPU) environment with GCC 10/11/12. Several solutions:


- **Use the Department's machines**
- **Run Linux in an x86-64 VM**
  - Grab a VirtualBox VM image here: <https://bit.ly/3UGN45i>
- **Install Linux natively on your x86-64 machine**
  - If you know what you are doing
  - Ubuntu 22.04 recommended

# Required Software

All programming exercises should be done in a Linux  x86-64 (i.e. Intel CPU) environment with GCC 10/11/12. Several solutions:

- **Use the Department's machines**
- **Run Linux in an x86-64 VM**
  - Grab a VirtualBox VM image here: <https://bit.ly/3UGN45i>
- **Install Linux natively on your x86-64 machine**
  - If you know what you are doing
  - Ubuntu 22.04 recommended
- **Do NOT use native Windows/Mac or WSL environments**
  - Won't provide support on these and you may lose marks because of toolchain difference

# Required Software

All programming exercises should be done in a Linux  x86-64 (i.e. Intel CPU) environment with GCC 10/11/12. Several solutions:

- **Use the Department's machines**
- **Run Linux in an x86-64 VM**
  - Grab a VirtualBox VM image here: <https://bit.ly/3UGN45i>
- **Install Linux natively on your x86-64 machine**
  - If you know what you are doing
  - Ubuntu 22.04 recommended
- **Do NOT use native Windows/Mac or WSL environments**
  - Won't provide support on these and you may lose marks because of toolchain difference
- **MacBooks with non-Intel (i.e. M1) CPUs:** some solutions given next but the general advice is to come on campus and use lab machines



# Lecture Videos

- Videos and slides available on Blackboard: <https://bit.ly/3XVrzyV>

# Lecture Videos

- Videos and slides available on Blackboard: <https://bit.ly/3XVrzyV>
- When watching videos/attending lecture, **access the slides for code examples you can run in your browser:**
  - <https://olivierpierre.github.io/comp26020>
  - You will also find **lecture notes** there

# Lecture Videos

- Videos and slides available on Blackboard: <https://bit.ly/3XVrzyV>
- When watching videos/attending lecture, **access the slides for code examples you can run in your browser:**
  - <https://olivierpierre.github.io/comp26020>
  - You will also find **lecture notes** there
- Using the bottom-right links you can:
  - Download the file: click on the blue path
  - Run it in your browser in a Programiz sandbox 
  - Get instructions on how to load and run it in a container that should run on Mac/Windows by clicking on the GitHub logo 

```
#include <stdio.h>

int main() {
    printf("hello, world!\n");
    return 0;
}
```

[00-logistics/sample-code.c](#)  

# Feedback on Lectures

- Each slides deck ends with a link to a small feedback form
- It's fully optional and anonymous, feel free to use it to tell what you think about the lecture in question
  - Your feedback will help to make the course better!

## Picking a Programming Paradigm

- Some paradigms are (much) better suited than others to solve a **given kind of software engineering problem**
- Choosing a paradigm and a language to solve a given software engineering problem has **huge consequences in terms of the design and behaviour of the solution**, impacting among others:
  - Code structure and modularity, clarity/complexity to understand, amount of LoC

# Lab Assignment (marked)

- Subject is already on Blackboard: <https://bit.ly/3ZsgPZH>
- Development of a matrix processing library in C
- Weight: 6.5% of the final COMP26020 mark
  - **50/50 coursework/exam weight split for the C part**



# Autocorrected Programming Exercises (not marked)

- A few exercises per week:
  - Divided into **essential** and **additional** exercises
  - You should really try at least to do the essentials

# Autocorrected Programming Exercises (not marked)

- A few exercises per week:
  - Divided into **essential** and **additional** exercises
  - You should really try at least to do the essentials
- Autocorrected with `check50`
- To trigger autocorrection, have your source code in the local folder and type on the command line:

```
check50 -l --ansi-log <exercise URL>
```

- URL is per-exercise, given on the page of each exercise

# Autocorrected Programming Exercises (not marked)

- A few exercises per week:
  - Divided into **essential** and **additional** exercises
  - You should really try at least to do the essentials
- Autocorrected with `check50`
- To trigger autocorrection, have your source code in the local folder and type on the command line:

```
check50 -l --ansi-log <exercise URL>
```

- URL is per-exercise, given on the page of each exercise
- Accessible from the website, along with instructions on how to install `check50`: <https://olivierpierre.github.io/comp26020/exercise-set-1.html>

# Quizzes

- A small quiz to complete each week, after having seen all the lecture materials
  - On Blackboard: <https://bit.ly/47qB4sV>
  - Formative (unmarked)

submissions: (78%)		submissions: (70%)	
16-50		17-51	
105-134		119-146	
59-84		61-88	
8-13		8-13	
141-153		152-164	

```

int main(int argc, char **argv){
    //Check number of command line arguments
    if(argc != 4){
        printf("ERROR: incorrect number of arguments\n");
        printf("Please use: %s <M> <N> <seed>\n", argv[0]);
        return -1;
    }
    //Check it is a digit
    if(!is_number(argv[1]) || !is_number(argv[2]) || !is_number(argv[3])){
        printf("ERROR: invalid argument\n");
        printf("Please use: %s <M> <N> <seed>\n", argv[0]);
        return -1;
    }
    //Check it is valid number
    if((atoi(argv[1])<=0) || (atoi(argv[2])<=0) || (atoi(argv[3])<0)){
        printf("ERROR: invalid value\n");
        printf("Please use: %s <M> <N> <seed>\n", argv[0]);
        return -1;
    }

    // get the value of command line arguments
    int m = atoi(argv[1]), n = atoi(argv[2]), seed = atoi(argv[3]);

    //initialise
    int **matrix_1;
    int **matrix_2;
    int **result;

    //generate the matrix
    matrix_1 = matrix_generate(m, n);
    matrix_2 = matrix_generate(n, m);
    result = matrix_generate(m, m);

    //check if fail allocation
    //if (matrix_1 == NULL) return -1;
    //if (matrix_2 == NULL) return -1;
    //if (result == NULL) return -1;

    //fill matrix with random number
    srand(seed);

    fill_random_matrix(m, n, matrix_1);
    fill_random_matrix(n, m, matrix_2);

    //setup time
    struct timeval start, stop, elapsed;

```

```

int main(int argc, char **argv){
    //Check number of argument present
    if(argc != 4){
        printf("ERROR: incorrect number of arguments\n");
        printf("Please use: %s <M> <N> <seed>\n", argv[0]);
        return -1;
    }
    //Check it is a digit
    if(!check_digit(argv[1]) || !check_digit(argv[2]) || !check_digit(argv[3])){
        printf("ERROR: invalid argument\n");
        printf("Please use: %s <M> <N> <seed>\n", argv[0]);
        return -1;
    }
    //Check it is valid number
    // cannot have negative int
    if((atoi(argv[1])<=0) || (atoi(argv[2])<=0) || (atoi(argv[3])<0)){
        printf("ERROR: invalid value\n");
        printf("Please use: %s <M> <N> <seed>\n", argv[0]);
        return -1;
    }

    // get the value of command line arguments
    // save to variable
    int m = atoi(argv[1]), n = atoi(argv[2]), seed = atoi(argv[3]);

    //initialise
    int **matrix_1;
    int **matrix_2;
    int **result;

    //generate the matrix
    matrix_1 = generate_matrix(m, n);
    matrix_2 = generate_matrix(n, m);
    result = generate_matrix(m, m);

    //check if fail allocation
    if (matrix_1 == NULL) return -1;
    if (matrix_2 == NULL) return -1;
    if (result == NULL) return -1;

    //fill matrix with random number
    srand(seed);
    fill_matrix(m, n, matrix_1);

    fill_matrix(n, m, matrix_2);
    fill_matrix_zero(m, m, result);

    //setup time
    struct timeval start, stop, elapsed;

```

# How to Get Help?

- Any question? in increasing order or urgency:
  - Step 1: come to the next support session
  - Step 2: discussion boards on Blackboard
    - <https://bit.ly/3TuORsC>
    - **Do not post answers there!**
  - Step 3 or if urgent: [pierre.olivier@manchester.ac.uk](mailto:pierre.olivier@manchester.ac.uk)