Practical 0 – Starting Eclipse

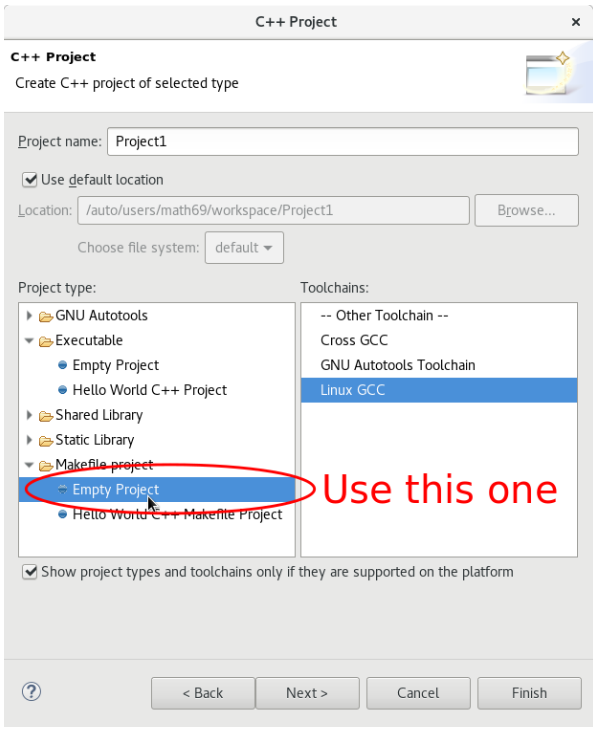
You can do all the practicals for this course with a simple editor and the command-line interface if you wish. However, in future programming projects you may find that you get better productivity from using an integrated development environment such as Eclipse or Visual Studio.

1 Launch the Eclipse IDE

Use the program launcher ("Activities") to find Eclipse or start it from the command-line with `eclipse &`. The first time you start Eclipse you'll need to choose a workspace (such as the default which is `$HOME/workspace`). Then on the Welcome screen you will need to click the arrow at the top right to go to the Workbench.

2 Create a project

From the File menu, choose New, then Other/C++ Project. On some versions of Eclipse you may need to select `New -> Project -> C/C++ -> C++ Project`. If you see the "Templates for New C/C++ Project" window then select "C++ Managed Build". Give the project a name (for example "Project1") and under Project types, choose `Makefile Project -> Empty Project` with Toolchain `Linux GCC`. Click Finish. You should switch to the C++ perspective as prompted.


C++ Project

C++ Project
Create C++ project of selected type

Project name: Project1

[✓] Use default location
Location: /auto/users/math69/workspace/Project1
Choose file system: default

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World C++ Project
- Shared Library
- Static Library
- Makefile project
  - Empty Project [✓]
  - Hello World C++ Makefile Project

Toolchains:
- -- Other Toolchain --
- Cross GCC
- GNU Autotools Toolchain
- Linux GCC [✓]

[✓] Show project types and toolchains only if they are supported on the platform

(It's important for Practical 3 onward that you select the correct project type, so I've included a screenshot from a recent version of Eclipse.)

3 Open a C++ code window and start programming

From the File menu, choose New, then Source File. Choose a name such as `prog.cpp`, and then click the Open button. Type your program into the central, blank window.

```
#include <iostream>
#include <string>
int main()
{
  std::string name;
  std::cout<<"Enter name\n";
  std::cin>>name;
  std::cout<<"Hello "<< name<<"\n";
  return 0;
}
```

4 Compiling

From the File menu (or right-click the project name), choose New, then File (or File from Template). Call your new file `Makefile` and put **only** these contents into it.

```
all:  prog
```

The idea here is that build system will look for a file called `Makefile` with instructions for building a target called 'all'. Your instructions say that one of the components of 'all' is an executable called 'prog'. The build system will figure out that `prog.cpp` needs to be compiled. From the Project menu select Build Project, to test this is working.

5 Running

Select `Run -> Run`. Choose C/C++ Local Application and press the 'New' button. You may need to enter the name of the exectuable file ('prog') in the C/C++ Local Application box. You should be able to launch the program.
If this is your first time coding in C++ then make sure you can use Eclipse to compile code or that you can compile using a text editor and shell in Linux. If you want to work in another operating system then you might experiment with Visual Studio (Windows) or Code::Blocks (multiplatform).

6 Extras

Eclipse needs a lot of screen space. It's a good idea to run it full-screen and maximise the size of the editing window.

It's easier for demonstrators to talk about your code if you switch on line numbers. Right-click in the left margin of the editing window and you should find a "Show line numbers" option.

Practical 1

If this is your first time coding in C++ then make sure you can use Eclipse to compile code or that you can compile using a text editor and shell in Linux. If you want to work in another operating system then you might experiment with Visual Studio (Windows) or Code::Blocks (multiplatform). Note that when you are working on small, similar questions then you might need to consider how to organise your work. You might prefer

- one Eclipse project per question
- one file per question each with its own `main()` function. You may need to tweak the build rules so that each file produces an executable
- one function per question, with the `main()` function only calling the function you are testing (when you are used to writing functions)
- delete (or comment out) answers to previous questions as you go along

If you feel that exercises in any of these sheets are getting repetitive and that you "get the point" then it's fine to skip questions and move on. If you need to check or visualize your results then feel free to use MATLAB or any scripting tools that you know.

1. Write code that asks a user to enter two positive integers from the keyboard and writes the product of these integers to the screen.
2. Write code that calculates the product of more than two positive integers. The user should be asked how many integers they want to enter, and then to enter the integers. The code should then print the output to the screen. Note that you are not required to *store* the input numbers but only calculate their product.
3. Modify the code in the previous exercise so that the user doesn't have to specify in advance how many integers there will be. Instead the user enters "-1" to signify the end of the string of integers.
4. Modify the code in the previous exercise so that the program terminates if the product of the integers exceeds 1000.
5. Write code to implement the backward Euler method to solve the ODE

$$\frac{\mathrm{d}y}{\mathrm{d}x} = -y \qquad y(0) = 1.$$

   on the interval $[0, 1]$. The user should be asked how many grid points they want to use—use an `assert` statement to ensure that the number of grid points is greater than 1. Your code should print a file called `xy.dat` that has two columns: the calculated values of x; and the calculated values of y. Read this file into MATLAB and plot the solution.

   [*The backward Euler method for this problem results in the difference relation*

$$\frac{y_{n+1} - y_n}{h} = -y_{n+1}$$

   where h is step size]

Practical 1 continues on next page

6. Write code that reads the file `xy.dat` created in the previous exercise, and computes the error at each point (the true solution is $y = \mathrm{e}^{-x}$). Print to the screen the maximum error.

7. Write code to calculate the scalar (dot) product of two vectors of length 3.

8. Write code to multiply two 3×3 matrices.

9. Write a file called `numbers.dat` that contains 2 rows and 2 columns of numbers.

   Run the following code:

```
#include <iostream>
#include <fstream>
#include <cassert>
int main()
{
  std::ifstream input("numbers.dat");
  assert(input.is_open());
  double x, y;
  while (!input.eof())
    {
      input >> x >> y;
      std::cout << x << "  " << y << "\n";
    }
  return 0;
}
```

   Did the code give the expected results? Why were 3 rows of output printed out?[1].

10. Write code for solving the equation $A\mathbf{x} = \mathbf{b}$, where is A a 3×3 matrix, and $\mathbf{b}$ is a vector of length 3.

    Have a think about how you would write code to solve the equation $A\mathbf{x} = \mathbf{b}$ using Gaussian elimination, where is A a $n \times n$ matrix, and $\mathbf{b}$ is a vector of length n.

[1]Hint—Investigate the use of `input.fail()` or `input.good()`

Practical 2

Use dynamic allocation of memory for vectors and matrices in this practical: i.e. allocate memory for vectors as shown on **Slide 64**, and memory for matrices as shown on **Slide 67**.

1. Write code that sends the address of an integer to a function that prints out the value of the integer.
2. Write code that sends the address of an integer to a function that changes the value of the integer.
3. Write a function that accepts two floating point numbers, and swaps the values of these numbers.
   (a) Write this function using pointers as function arguments:
   `void swap(double* x, double* y)`
   (b) Write this function using references as the function arguments
4. Write a function that returns the scalar (dot) product of two vectors of a given length.
5. Write a function that can be used to calculate the mean and standard deviation of an array of double precision numbers. Note that the standard deviation σ of a collection of numbers x_j, $j = 1, 2, \ldots, N$ is given by

   $$\sigma = \sqrt{\frac{\sum_{j=1}^{N}(x_j - \bar{x})^2}{N-1}}$$

   where $\bar{x}$ is the mean of the numbers.
6. Write a function `multiply` that may be used to multiply two matrices given the size of both matrices.
7. Overload the function `multiply` written in the previous example so that it may be used to multiply:
   (a) a vector and a matrix of given sizes;
   (b) a matrix and a vector of given sizes;
   (c) a scalar and a matrix of a given size; and
   (d) a matrix of a given size and a scalar.
8. The $p-$norm of a vector $\mathbf{v}$ of length n is given by

   $$||\mathbf{v}||_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p}$$

   where p is a positive integer. Write a function to calculate the $p-$norm of a given array, where p takes the default value 2.
9. Write a module for solving the 3×3 linear system $A\mathbf{x} = \mathbf{b}$ where A is non-singular.
10. Write a module for solving the $n \times n$ linear system $A\mathbf{x} = \mathbf{b}$ using Gaussian elimination with pivoting, where A is non-singular.

Practical 3

This practical is based on the material in Lectures 7–9. In all cases test your code using suitable examples.

In order to help you with this practical, I have prepared a small zip file containing `student.h`, `student.cpp`, `use_student.cpp` and a `Makefile`. This is available on weblearn and at
`http://www.cs.ox.ac.uk/people/joe.pitt-francis/C++ScientificComputing/Prac3.zip`

If you are using `make` on the command line (under Linux) then you will find the `Makefile` handy since it will re-build automatically as you change things.

1. Write a class of students at the University of Oxford that has the following public members
   - A string for the student's name
   - A string for the student's college
   - A string for the student's status, i.e. graduate or undergraduate
   - A string for the degree that the student is registered for, e.g. BA, MSc, DPhil, PGCE
   - A double precision variable that stores library fines owed by the student
   - A double precision variable that stores the tuition fees owed by the student
   - A function that returns the total money owed by the student
   - A few constructors that take different arguments
2. The string representing the student's status can be only "graduate" or "undergraduate". Enforce this by making a student's status a private member of the class. Write one function that allows the user to set this variable only to one of these values, and another function that can be used to access this private variable.
3. (a) A graduate student at the University of Oxford *is a* student at the University of Oxford. Derive a class of graduate students from the class of students that you have already written. Note that in order for the graduate student class to have access to the status member of student it should now be *protected* rather than private.

   (b) Graduate students do not pay tuition fees. Use polymorphism to write a function that calculates the total money owed by a graduate student.
4. A student enrolled in the DTC *is a* graduate student at the University of Oxford. Derive this class from the class written in the previous question.[2]
5. Write a function that accepts a pointer to an instance of the class of students and use the `->` notation shown on Slide 134.

[2]Hint: think about the access privileges for the `status` member of the base class—look at Slide 119 and Slide 139

Practical 4

The `Exception` class and `Vector` class designed in Lectures 10–12 and a Makefile are available on weblearn and also at
`http://www.cs.ox.ac.uk/people/joe.pitt-francis/C++ScientificComputing/Prac4.zip`

This class of vectors includes the following:

- some constructors;
- a destructor;
- overloading of the binary operators +, –, $*$ and /, along with the unary operator –;
- overloading of the operators = and ( ); and
- the functions `norm` and `length`.

Exceptions have been used to check that the vectors used are of the correct size.

The aim of this practical is to write a class of matrices. Examples of features that should definitely be included are

1. A constructor that dynamically allocates memory
2. A copy constructor
3. A destructor that frees memory
4. Overloading of the binary operators + and – to define addition and subtraction of matrices
5. Overloading of the = assignment operator
6. Overloading of the unary – operator

Examples of other features you may wish to include are

1. Overloading of the ( ) operator
2. Overloading of the $*$ operator to define multiplication of matrices by scalars, vectors and other matrices[3]
3. The function `size` that returns the size of a matrix (note that since size has two components it should return an array or a vector)
4. Overloading of the / operator to define division of a matrix by a scalar and a vector – i.e. if `A*x=b` then `x = b/A`. Use Gaussian elimination for solving `A*x=b`
5. Simple MATLAB functions such as `size`, `eye` and `diag`
6. A more complex MATLAB function such as `cgs` or `gmres`

[3]Hint: At this point you may find that your `Matrix` and `Vector` classes need to know about each other. If you find that your `#include`s have circular dependency then investigate "forward declaration".

More practicals (STL, Mex, finite difference)

1. Use templates to write a single function that may be used to calculate the absolute value of an integer or a double precision floating point number.

2. Use the data on `https://en.wikipedia.org/wiki/List_of_MPs_who_stood_down_at_the_2019_United_Kingdom_general_election` to construct two STL multimaps: one keyed on year of entry and one keyed on political party. The idea is to enumerate all the MPs who entered Parliament in 1970 (or whenever) without having to search through the origin list.

   ```
   std::multimap<int, std::string> map_by_entry; // Key=year Value=surname
   ...
   ```

3. Using the two multimaps, form the set intersection of all the MPs in the list who entered in 1997 with party "Labour", and write out their names.

4. Copy the example file which has a `mexFunction` definition (slide 157) and save it in a `.cpp` file. Compile it with the `mex` compiler (or `mkoctfile --mex` for GNU Octave). Note that you will need to do this on a machine which has a valid version of MATLAB (or the `octave-headers` package). Run the function from the MATLAB interface (or from Octave).

5. Write a `mexFunction` which takes a vector as input and outputs the 2-norm of the vector. Extend the function so that it can optionally take another argument in order to compute the $p-$norm.

6. Write a `mexFunction` which multiplies two MATLAB matrices. Check your answer against MATLAB.

Extra practicals continue on next page

7. Solve the heat equation in 1D using an explicit finite difference method for $u(x,t)$ in $[0,1]$. The equation is
$$u_t = u_{xx},$$
with Neumann boundary conditions at the end of the domain
$$u_x(0,t) = 0, \quad u_x(1,t) = 0$$
and an initial heat distribution given by:
$$u(x,0) = \cos(2x).$$
This example is discussed at
`http://commons.wikimedia.org/wiki/Image:Heatequation_exampleB.gif`
The explicit finite difference scheme is
$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2},$$
where Δt is the and is known to be convergent for step sizes $\frac{\Delta t}{h^2} \leq \frac{1}{2}$. Use a first difference to impose the boundary conditions:
$$\frac{u_0^{n+1} - u_0^n}{\Delta t} = 2\frac{u_1^n - u_0^n}{h^2}.$$
In order for the bookkeeping to work properly you won't want u_j to be updated before it's used in the calculation of u_{j+1}. Write your answer into another vector and then copy other the original at the end of a time-step.

8. Re-factor the code so that this copying does not take place. Use two vectors at each time-step (one for the input and one for the output). Swap them over at the end of each time-step.

9. Re-factor the code to use only one vector with the value of u_j being held to be re-used in the next iteration of the loop.

10. Consider (and implement if you have time) using an implicit (backward Euler) solver for this problem. You will need to be able to solve linear algebra system involving a tri-diagonal matrix.