LECTURE 1:

MODULE OVERVIEW

COMP1002J: Introduction to Programming 2

Dr. Brett Becker (brett.becker@ucd.ie)
Beijing Dublin International College

A little about Brett

- Brett will be here in week 3

  - His research area is Computer Science Education
  - More on Brett: www.brettbecker.com
- I will give next week's lecture.
- We'll talk about labs later.

Module Timetable

- **Lectures:**
  - Weeks 1-12, 14-15
  - **Mondays** @ 13:30-15:05, Room 102, Teaching Building 4
- **Labs**
  - Weeks 3*-15
  - **Wednesdays** @ 13:30-15:05, Room 102, Teaching Building 4

- *Yes, labs start in week 3!

Course Outcomes

- On completing this module, the students will be able to:
  - understand the fundamental concepts of programming such as arrays, structures, pointers, functions, etc.;
  - be able to program in the C programming language;
  - demonstrate an ability to produce solutions to common programming problems.

Course Material

- Slides & Examples:
  - All slides and examples will be available via the UCD Computer Science Moodle site.
  - The CP2 page is at https://csmoodle.ucd.ie/moodle/course/view.php?id=772
  - You will need the enrolment key (Comp1002Jxyz)
  - **Everyone should register to this moodle this week!**
    - **Please tell others that aren't here.**

- But, a lecture is not just the PowerPoint slides. It also includes:
  - Code examples I create in class.
  - Things I write on the blackboard.
  - Things that I say.
  - Lab exercises and discussions.
  - All material on moodle (unless explicitly labelled as optional)
  - Basically, everything!

Course Material

- You should bring a notebook to class and take notes on each lecture.
  - Your understanding of things will be written in your own words.
  - Much easier to study later.

- Textbook: The C Programming Language (Kernighan & Richie)
  - Downloadable from Moodle

Course Assessment

- Continuous Assessment: **40%**
  - Worksheets/quizzes/etc. in the lab. **More details in week 3.**
  - Labs start week 3.
  - **Note:** For this module, attendance in labs will be recorded.
- End of semester written examination (2 hours): **60%**

Worksheets and lab work

- The lab exercises, quizzes, etc. are **individual** assignments (not group assignments).

- This means that you must do **your own work**.

  If you submit somebody else's work and pretend that you wrote it, this is called **plagiarism**.

- Plagiarism is a **very serious** academic offence. If you have questions or are in doubt, ask!

Plagiarism & UCD Computer Science

- **Plagiarism is a serious academic offence**
  - [Student Code, section 6.2] or [UCD Registry Plagiarism Policy] or [CS Plagiarism policy and procedures]
- Our staff and demonstrators are **proactive** in looking for possible plagiarism in all submitted work
- Suspected plagiarism is reported to the CS Plagiarism subcommittee for investigation
  - Usually includes an interview with student(s) involved
  - 1st offence: **usually** 0 or NG in the affected components
  - 2nd offence: referred to the **University disciplinary committee**
- Student who enables plagiarism is equally responsible

http://www.ucd.ie/registry/academicsecretariat/docs/plagiarism_po.pdf

http://www.ucd.ie/registry/academicsecretariat/docs/student_code.pdf

http://libguides.ucd.ie/academicintegrity

How to avoid plagiarism: Helping without copying.

- If you are trying to help a classmate with a programming assignment, there are two golden rules:

1. **Never, ever** give your code to somebody else.
   - You don't know what they will do with it or who they will give it to.
   - If somebody else submits code that is the same as yours, **you will be questioned also.**
2. **Don't touch their keyboard**.
   - Don't type solutions for them! It will end up looking a lot like your code. Also, they don't learn anything.

How to avoid plagiarism: Helping without copying.

- Here are some other ways you can help a friend with an assignment, without risking plagiarism:
  - If their code doesn't work, it's OK to explain what is wrong with it.
  - If they don't understand a concept, draw a picture to explain.
  - Tell them about useful functions that can help achieve their goals.
  - Describe an algorithm that will help.
    - Describe it in **words** or **pictures**, not in code!
    - Example: "You could try saving the first value as a variable. Then you could use a loop to keep reading new values from the file until you reach the end of the file."

Asking for help

- If you find things difficult, help is available.
  - There is a lecturer (me!) and many TAs in every lab.
  - You can ask a question after class.
  - You can email a TA with a question outside class.
  - You can email me with a question outside class.
  - You can get help from your classmates.
    - Getting help to understand something is OK. Simply copying a solution is not ok!

Asking for Help

- The best way to get useful answers is to ask good questions.
- **Don't** just send a photo of your computer screen and ask "Why does this not work?".
- **Do:**
  - Send your C file(s) as an attachment. We can't run code that's in a photograph to test it out!
  - Say what error message you got when you tried to run the code (if any).
  - Say what the code did that you did not expect.
  - Say what the code did not do that you did expect.

Asking for Help

- You should always ask a TA first.
- If you ever need to email me you **must** include:
  - Name
  - Student Number
  - Class (Comp1002J)
  - It is **highly** recommended that you use your UCD email address.
    - It is not uncommon for emails from qq and other domains to go to spam or not get delivered.

Labs

- We will be learning C this semester, continuing on from what you learned last semester.

- It is your responsibility to have a working C editing environment and compiler on your computer by week 3!
  - **I recommend that you use Notepad++ and MinGW**
    - **www.notepad-plus-plus.org**
    - **www.mingw.org**

Reminder

- The Lab exercises/quizzes/worksheets make up 40% of the final grade for this module.
- Not all weeks will be graded. I will tell you which weeks will be graded.
- You may also have quizzes and other types of assessment
- Worksheets are *individual* pieces of work. You should only submit work that *you* have done.
  - Plagiarism is a serious offence. Ask if you are in doubt!

Attendance At Labs

- Everybody must attend their lab sessions.
  - When your timetable says you should be in a lab, you should be in a lab!
  - Lab sessions take priority over everything else: meetings with other teachers (including English teachers) must happen at another time.
- Practicing is the **only** way to get good at programming. If you do not practice writing programs during this module, next year will be very difficult.
- There are many Teaching Assistants available to help explain anything you do not understand.

Grading Policy

- Every worksheet has a date to say when your answers are due.

- I will accept work up until **exactly one week** past this deadline, but a penalty is applied for late submissions.
  - The moodle will not allow you to upload later than this.

- My grading policy is:
  - Worksheets submitted less than 1 week after deadline: 10% subtracted from grade.
  - Worksheets more than 1 week late will not be accepted.

Grading of Worksheets

- Grading of worksheet solutions is not only based on whether your code works.

- The *quality* of the code is assessed too.
  - Do you use variable names that describe what they store?
  - Is your code readable, and properly indented?
  - Do you include comments that help the reader understand what you're doing?
  - Does your code achieve its goals quickly, or do you have a lot of extra code that is not needed?
  - Did you follow the instructions? (e.g. if you were asked to use pointer arithmetic then you cannot get full marks if you did it some other way).

Grading of Worksheets

- In other words, is there careful thought behind the code?

- Is the code elegant?
  - I do not expect super-elegant, perfect code
  - I do expect code that is written by someone who is learning to write elegant code
  - Progress is key, along with careful attention!

Elegance

- Something to think about…

- Programming can be fun, and very satisfying when you come up with an “elegant” solution to a problem.
- Elegant code is **simple**, gives you some **new insight** and is generally **composable** and **modular**. These qualities, although they may look almost arbitrary, are actually deeply related, practically different facets of the same underlying idea.
  - http://www.forbes.com/sites/quora/2014/06/02/what-does-one-mean-by-elegant-code/#aa9d10c64884

Elegance

- **Elegant:** Combining simplicity, power, and a certain ineffable grace of design. Higher praise than "clever", "winning" or even cuspy.
  - The French aviator, adventurer, and author Antoine de Saint-Exup'ery, probably best known for his classic children's book "The Little Prince", was also an aircraft designer. He gave us perhaps the best definition of engineering elegance when he said "A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away.“
  - http://foldoc.org/elegant
    - By the way, FOLDOC (the Free OnLine Dictionary of Computing – www.foldoc.org, is a great resource!

More on C

- The C language has a rich history that is intertwined with the history of computers in general – operating systems, and many other programming languages either use C, are based on C, or wouldn't even have been possible without
- On moodle there is a great article on the history of C: "The Development of C" by Dennis Ritchie
  - I recommend you read this – it is not long.

Have a great semester!

- Please let me know if you have any questions.