Introduction to the course and development tools

CSC 240 - Data structures with C++ - Lyon College, FA24

Marcus Birkenkrahe

July 29, 2024

Contents

1 Course Overview

The syllabus is available on Canvas and on GitHub.

1. **Instructor Introduction**
    - Background and experience
    - Course objectives and expectations
2. **General Course Information**
    - Meeting Times: Tue-Thu 9:30-10.45 AM (75 min)
    - Meeting place: Derby Science Building computer lab room 239
    - Professor's Office: Derby Science Building 210
    - Phone: (870) 307-7254 (office) / (501) 422-4725 (private)

- Office hours: by appointment MWF 4pm, Tue 3pm, Thu 11 am & 3 pm

3. **Materials and Multimedia**

   Primary sources:

   - C Programming by King (W W Norton, 2008)
   - Learn C the Hard Way by Shaw (Addison-Wesley, 2015)
   - Algorithmic Thinking (2e) by Zingaro (NoStarch,2024)
   - Mastering algorithms with C by Loudon (O'Reilly, 1999)

   Secondary sources:

   - 100-page C++ language reference: Rook's Guide to C++
   - 800-page C++ language reference: C++ Crash Course
   - Optional Textbooks:
     - Helfrich, C++ Data Structures (KendallHunt, 2020)
     - Kanetkar Y, Data Structures Through C++ 5th ed (bpb, 2024)
     - Malik, C++ Programming - Program Design Including Data Structures (Cengage 2015)
     - La Rocca, Grokking Data Structures (Manning 2023) - with Python
     - Morin, Open Data Structures (in C++) (OpenText, 2013)
   - Recommended Videos:
     - "C++ Data Structures & Algorithms @Udemy
     - Helfrich's YouTube videos on C++ Data Structures
     - freeCodeCamp course on Data Structures with C++

4. **Course Objectives**

   - Explore theory and application of data structures using mainly C
   - Equip students with knowledge to store, process, and retrieve data efficiently
   - Emphasize algorithmic thinking and optimization techniques

5. **Target Audience**

   - Students with basic understanding of programming concepts

- Those interested in preparing for deeper study of algorithms
- Those interested in advancing their knowledge in data science

6. **Student Learning Outcomes**

   - Understand major data structures and their applications
   - Implement various data structures in C and C++
   - Analyze efficiency of data structures and algorithms
   - Apply algorithmic thinking to solve complex problems
   - Design and develop efficient software solutions
   - Design UML class diagrams and BMPN models

7. **Course Requirements**

   - Prerequisites: Introduction to Programming (CSC100 or CSC115 or CSC109, and MTH101)
   - Willingness to engage in problem-solving and critical thinking
   - Willingness to put in substantial time to program on your own

8. **Grading** (see also Rubric)

   - Weekly programming assignments: 50%
   - Weekly multiple-choice tests: 25%
   - Final exam (optional): 25%

9. **Learning Management System**

   - Use of Lyon's Canvas installation (use the calendar!)
   - Course materials, assignments, grades, recorded sessions

10. **GitHub, Linux & Emacs & Org-mode**

    - Course materials in public GitHub repository: github.com/birkenkrahe/alg1
    - Free subscription to GitHub Codespaces with AI coding assistant Copilot
    - Emacs Power User Editor & Org-mode in a Linux programming environment

2 Questions with Answers

1. What if you don't know C or C++?

   You should have taken introductory programming in Java or Python or C/C++: Both Java and Python are high-level languages, but C++ is quite different (and quite a bit more complicated, too).

   Make sure you take the (optional) entry quiz and complete the (optional) entry programming assignment if you didn't do it yet. If you meet the August 19, 11:50 pm deadline, you get some bonus points, too.

   If you have any weaknesses in the basics (or if you haven't studied C++ yet at all), read "The Rook's Guide to C++" for free and answer the review questions. Sample solutions available online in GitHub.

   Otherwise we'll be reviewing (and adding to) our knowledge of C/C++ in the first 4 weeks of the course (including some OOP aspects).

2. Will I be able to work on this course from home?

   Absolutely. You'll be using a personal virtual Linux box on our dedicated Lyon computer and data science server, and you can login to this system from anywhere as long as you've got an Internet connection.

   If that doesn't work for you for whatever reason (connection, convenience etc.) then you can easily install Linux via dual boot, as a virtual machine on your PC, or on Windows with the WSL system. On MacOS, you have a form of Unix already, so you're set.

   In this case, you can put your GitHub skills to good use: create a repository for this course where you keep the latest version of all your files, and clone it to your local PC. This is what I do, too: I'm working on my files in different locations, and I use Git to maintain the latest version on any machine at any time.

3. Why did you choose C/C++ for "data structures"?

I did not choose it, the catalog (and wiser, older professors before me) did. But I'm on board with it for a number of reasons:

- C/C++ is **compiled** (unlike Python) and offers high performance, which you need for big data and **complex** algorithms (like machine learning).
- C/C++ offers explicit **control** over memory allocation and deallocation, which is essential to know how data structures work under the hood.
- C++ in particular has a Standard Template Library (STL) which offers template classes for **common** data structures (vectors, lists, queues, stacks) and algorithms, which saves time and helps you learn.
- C/C++ is used in industry for performance-**critical** applications (like large-scale multi-player network games, network communication, or space exploration and communications).

4. But all the programs are in C not in C++ - why is that?

   After working out many lectures in C++ only, I realized that Object-orientation and the extra layers of abstraction that C++ brings, are actually in the way of understanding the essence of data structures.

   This essence is not "what can this language do?" but "how can you stroe, organize, and access data to solve a given problem most efficiently." This question is largely language-independent, and so we want that the language does not get in the way to our solving problems algorithmically.

   The arguments given by Zingaro in favor of C in his 2024 book "Algorithmic Thinking" resonated with me, too. Once you understand this stuff in C, you can take your knowledge anywhere, to any other language, wherever problems can be solved computationally.

5. What are your favorite programming languages?

   Though I have (passing) knowledge of many programming languages, C/C++, Python, SQL and Lisp are becoming

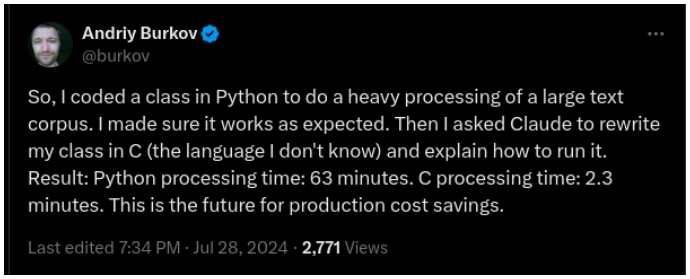
my main go-to languages for performance (C++), data science (Python), database applications (SQL), and Emacs customization (Lisp).

R is great for visualization and stats, and was another one of my favorite languages but it is slowly falling out of my quality world because of the dominance of the "Tidyverse" (see here). The data science intro courses are however still in R (with some Python).

I learnt these languages at different times in my life: Lisp because of Emacs (late 1980s), C++ in the 1990s, SQL in 2005, R in 2019, and Python in 2022. C++ has developed massively since then though and I am more comfortable with its subset C than with many of the modern developments of the language - but I'm keen to learn!

6. Why did you not choose Python given that it's the most popular programming language?

   Python has some serious deficiencies compared to C/C++, most importantly speed. Here is an interesting paper that shows that Python consumes 76 times more energy and is 72 times slower than C.



   But more importantly for the purpose of learning about data structures: Python abstracts away many details such as memory management, which are crucial to understanding why to choose one data structure over another. C gives you much more control over system resources. The only real reason to pick Python is that it's easier to learn and use.

   But at the end of the day, it's a personal decision. In my case, I'm also better at C than Python, and I like the control it gives.

7. How should you study for this course?

    (a) Code every day no matter how little.
    (b) Review lecture notes and notebooks.
    (c) Participate actively in the class.
    (d) Form study groups and/or join Lyon's Programming Club.
    (e) Complete assignments well before the deadline.
    (f) Practice literate programming by documenting your process.
    (g) Focus on fundamentals, simple examples, solid understanding.
    (h) Drill yourself using the weekly quizzes, make small examples.
    (i) Seek help when needed, on the chat or during office hours.
    (j) Build a code & notebook repository at GitHub for your resume.

[See also the "New FAQ" for fall '24 courses available on GitHub.]

3 Course content - Data Types and Data Structures

This is a vast topic. If you're anything like me, you like a clear roadmap with code examples. So here's all of what we'll cover in the briefest form possible, including definitions + code.

Data Types

Data types classify the type of data a variable can hold, and the operations that can be performed on it.

Primitive Data Types

- `int`: Integer type
- `float`: Floating-point type
- `double`: Double-precision floating-point type

- `char`: Character type
- `void`: Special purpose type

Derived Data Types

- **Array**: Collection of elements of the same type

  ```
  int arr[10];
  ```

- **Function**: Represents a function

  ```
  int func(int, float);
  ```

- **Pointer**: Stores the address of another variable

  ```
  int *ptr;
  ```

- **Structure**: Groups variables of different types

  ```
  struct Person {
    char name[50];
    int age;
  };
  ```

- **Union**: Similar to structures, but with shared memory

  ```
  union Data {
    int i;
    float f;
    char str[20];
  };
  ```

- **Enumeration**: Consists of named integer constants

  ```
  enum Color {RED, GREEN, BLUE};
  ```

Data Structures

Data structures organize and store data for efficient access and modification.

Common Data Structures

- **Arrays**: Collection of elements of the same type

```
int arr[10];
```

- **Linked Lists**: Each element points to the next element

```
struct Node {
  int data;
  struct Node *next;
};
```

- **Stacks**: Follows the LIFO (Last In First Out) principle

```
#define MAX 100
int stack[MAX];
int top = -1;
```

- **Queues**: Follows the FIFO (First In First Out) principle

```
#define MAX 100
int queue[MAX];
int front = -1, rear = -1;
```

- **Trees**: Hierarchical data structure

```
struct TreeNode {
  int data;
  struct TreeNode *left;
  struct TreeNode *right;
};
```

- **Graphs**: Collection of nodes and edges

```
struct Graph {
  int numVertices;
  int *adjMatrix;
};
```

- **Hash Tables**: Implements an associative array

```
struct HashTable {
  int size;
  int *table;
};
```

4 Development tools

This section is shared across my courses. Apologies if you have to hear/do it twice but remember: imitation/repetition creates mastery!

Pep talk for developers

- Computer and data science courses are "maker spaces": you're supposed to make stuff rather than only listen and be passive. Your "making" is "developing software", and flex your programming muscles. It's like weight training or running: 15 minutes per day beats 1 hour once a week.

- In computer and data science, your professional development is only as good as your toolkit. Your toolkit for this course includes:

  1. The Linux operating system and the shell
  2. The Emacs editor with the Org-mode package
  3. The GitHub software engineering platform

Linux - the world's premier OS

- A summer research grant 2024, and the gracious professional support of Lyon's IT Services enabled us to establish a Linux server for exclusive use in computer and data science courses.

- If you completed my Operating Systems course or if you were exposed to Linux in some other way (e.g. via WSL, the Windows Subsystem for Linux), you know more than enough Linux to feel comfortable.

- For the duration of the term, you get your own virtual Linux PC running Ubuntu LTS 22.04. This is where we will do all our work. Emacs, R, Python, C++, SQLite are pre-installed on your PC.

- You can access this PC from anywhere with an Internet connection.

- Having used Linux daily will give you an edge in the job market (esp for server-related jobs), and you should mention it on your resume.

Emacs - the world's most customizable hacking power tool

- You've most likely worked with Emacs + Org-mode and the literate programming approach in my other data science or computing courses.
- Test your memory with a few review questions:

1. What is special about Emacs?

   Emacs is a self-extensible, free, open source editor written in a Lisp dialect, and first published in the 1980s by Richard Stallman. It is a hacker- and power-user tool because of its customizability and openess. The vanilla version can be downloaded from gnu.org/software/emacs. Famous applications: Org-mode (for literate programming) and magit (for Git). Additional resource: Emacs Reference Card.

2. What is Org-mode?

   Org-mode is an Emacs mode (plugin or extension package) for plain-text note-taking, task management, documentation. It was first released in 2003 by Carsten Dominik. More information at orgmode.org. Famous application: literate programming, spreadsheets. Additional resource: Org-mode Reference Card.

3. What is Literate Programming?

   Literate Programming is a programming paradigm introduced by Donald Knuth in the 1970. It emphasises writing code and documentation together to make programs better structured and more enjoyable to read and understand by humans. More information at literateprogramming.com. Famous application: The TEX typesetting system.

4. How can you run the "Hello World" program in C++ inside Emacs?

   ```
   cout << "Hello, World!" << endl;
   ```

   ```
   Hello, World!
   ```

   This code chunk is a souped-up (by way of header arguments) version of this complete program:

```
// include input/output library
#include <iostream>
// use standard namespace for cout, endl
using namespace std;
int main() { // begin of main function
  // stream string to standard output then print newline
  cout << "Hello, world!" << endl;
  // return 0 when program ran successfully
  return 0;
} // end of main function
```

```
Hello, world!
```

The only header arguments left are: `C++` for the language, `:results output` to stream output to the screen, and `:exports both` to export both source code and output (e.g. to LaTeX, Markdown or HTML).

5. How can you run "Hello World" in C inside Emacs?

```
puts("Hello, World!");
```

```
Hello, World!
```

This code chunk works only because of the `#+property:` header at the top of the file:

```
#+property: header-args:C :main yes :includes <stdio.h> :results output
```

To make use of it, you only need to open the file or run it with `C-c C-c`. If you work with code that includes functions outside of `main`, you need to change the header arguments. Example:

```
void hello(void); // prototype definition

int main(void)
{
  hello();
  return 0;
}

void hello(void)
{
```

```
  puts("Hello, World!");
}

Hello, World!
```

Nt

6. What are the differences between the C and the C++ versions?

C++	C
`<iostream>`	`<stdio.h>`
`using namespace std`	[1]
`cout << ... << endl;`	`puts("...");`
header-args:C++	header-args:C

   What does the `<>` mean in the `#include` statements?[2]

7. Why are we using Emacs + Org-mode instead of the VSCode or Code::Blocks IDEs?

   While VSCode and Code::Blocks are excellent IDEs with their own strengths, Emacs + Org-mode provides a unique combination of customizability, integrated task management, support for literate programming, and a powerful text editing environment that can lead to a more efficient and personalized workflow, and that teaches you important file management and productivity techniques - with a much higher transfer value than other tools.

- Having mastered and used Emacs daily will give you an edge in the job market (esp for programming jobs) and you should mention it as "Literate Programming with Emacs/Org-mode" on your resume.

GitHub - the world's largest development platform

[If you are in more than one of my courses, do this only once.]

- GitHub is the premier software engineering platform. An early user (since 2010) I use it for course materials, but most users are software developers and teams who use Git for version control.

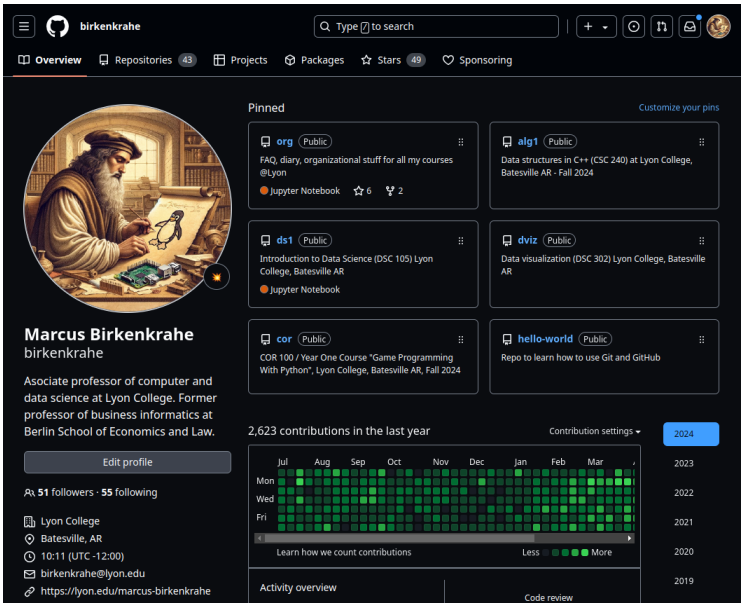
[1] C does not have namespaces like C++ to encapsulate names and organize code, instead it has prefixes, `static`, and `struct` objects to avoid name conflicts and group related functions and variables.

[2] It means that the header file or library is in the computers `PATH`, the set of locations where the computer looks for programs. Alternatively, you need to put the complete (absolute) path to the file in between `"..."`.

- Besides software development, you can also use it to share code snippets called "gists" (cp. my gists), and (as students or paying pro subscribers) for AI-assisted code development ("codespaces").

- My course materials were always on GitHub (so that I can develop them on different computers and use it as a central Git-controlled hub). From fall'24, you'll take another step towards software engineering. You must:

  1. Register with GitHub (use your Lyon email address/Google) at github.com/join. You can use this registration with many coding-related sites as login.
  2. Complete (or re-do) the GitHub Hello World exercise in class. The repo that you create should be called `hello-world`, include a `README` file, a `.gitignore` C++ template, and the `GNU General Public License v3.0`. Writing the `README` will teach you to write Markdown and use the markdown editor. When you are done with the description, you "Commit changes..." to save the file.
  3. Fork my course materials from github.com/birkenkrahe/alg1. You find the "fork" option at the top of the dashboard. The owner of the forked repo will be notified. Whenever he makes changes, you can update your fork (that'll be almost daily in my case, at least during the term).
  4. Create a (public) repo for the code you generate in the course, change your avatar, edit your profile, customize your pins
  5. Submit a screenshot of your GitHub profile that shows the forked course repo, the hello-world repo, your (professional) profile pic and description, like this (`hello-world` is pinned).



6. Org-mode files from my repos are automatically rendered as markdown in GitHub on the desktop. Footnotes and special layout does not render well, and the mobile GitHub app does not render Org-mode at all.
7. What the Hello World exercise does not show you is how you use GitHub for your own code across **remote** locations as I do it. For that you need to **clone** your own repo to a remote computer. This is explained in the freeCodeCamp tutorial (tinyurl.com/guide-to-git). There is also a short course (4 hr) "Introduction to Git"[3] and a short tutorial.
8. How are you going to use GitHub?
   - You'll regularly update your fork of my course repo.
   - In this way you have automatic access to the latest materials.
   - If you have a PC, you can install `git` and `clone` the repo.
   - If you have Emacs, you can install `magit` and use Git that way.
9. Having used GitHub like a professional daily will give you an edge in the job market (esp. for software engineering jobs) and you should mention it (as "GitHub/Git") on your resume.

[3] Lyon College has a classroom at DataCamp, and if you're enrolled, you can ask me to get access to this online platform. DataCamp is not for C++ but for SQL, Python and R, as well as for tools like Git and the shell.

GitHub Hello World Exercise

You've got to be registered at GitHub (github.com/join). Open github.com in your browser (Google Chrome is best) and login.

- Step 1: Create hello-world repo and set it up
  - Click on your profile image in the upper right corner, and pick "Your repositories".
  - Click on the green "New" button to create a new repo
  - In the form, choose `hello-world` as repo name
  - Write a minimal description (`"Hello World exercise for CSC 240"`)
  - Check `"Public"` (everybody can see this)
  - Check `"Add a README file"`
  - Choose the `.gitignore template: C++`
  - Choose the `"GNU General Public License v3.0"`
  - Click `"Create repository"` at the bottom of the page.
  - If you checked "Add a README file", the Markdown editor will open: enter a description (plain-text), then click on `Preview` to see how it will look like[4].
  - Click on the green `"Commit changes..."` button. A second window appears - make sure you check `"Commit directly to the ~main~ branch"`, and click on `"Commit changes"`.
  - Your repo appears with three auto-created files, `.gitignore`, `LICENSE`, and `README.md`[5].
- Step 2: Create a branch
  - Go back to the exercise and continue with step 2. Make sure you read the explanation on what a "branch" exactly is. If this is Chinese to you (and you're not Chinese), check out the freeCodeCamp tutorial at tinyurl.com/guide-to-git.

[4] Markdown is simple layout language: for example # gives you a headline, ‘ (backtick) choose code font, * gives you bold face, and _ gives you italics.

[5] Explanation: `.gitignore` contains file types that Git will ignore - especially those created in the process of compiling C++ code - you can add other file types here, e.g. `*.*~` for backup, or `*.html` for HTML files; the LICENSE specifies the copyright for everything in your repo; the `README.md` file is markdown with a description of the repository.

  - Essentially, you're posing as a developer who creates a new branch called `readme-edits` besides the `main` branch. Once you've added the branch, you'll see both branches in the `Code` dashboard of your repo.

- Step 3: Make and commit changes

  - You're now asked to make a change to your code base in the new branch using the `README.md` file (a change to any file would be equivalent).
  - Once you made the change, you commit it[6]. You can make as many changes and commits as you like.
  - Your two branches, `main` and `readme-edits` have now diverged.

- Step 4: Open a pull request

  - A "pull request" is a request for the maintainer of `main` to consider using your changes in the `main` code base. Follow the steps of the exercise.
  - After creating a `New pull request`, you can check out the changes in the well-known Linux "diff" format[7], a line-by-line comparison.

- Step 5: merge your pull request

  - The GitHub dashboards seem a little crowded. When you `"View the pull request"`, you find the `"Merge pull request"` button, and since the branches do not report a "conflict", you can go ahead and merge.
  - The pull request is now closed. You can delete the `readme-edits` branch (e.g. by clicking on the branch symbol next to the branch name in the repo dashboard).
  - Now go back to your profile, find `Customize your pins` and pin `hello-world` to the profile as you see it on my GitHub profile.

[6] On the shell, the equivalent `git` command is `git commit -m "[message]"` following a `git add .` for all changed files. To push a commit to the remote code base (`origin`), use `git push`.

[7] That's not a coincidence: both Linux and Git were created by Linus Torvalds, a Finnish programmer and (still) leader of the global Linux kernel development

5 Summary

- You only need basic programming skills to succeed in this course.
- There will be 2 programming assignments and 1 quiz per week, with an optional final exam.
- We will make excessive use of Linux, Emacs + Org-mode, and GitHub.
- We will mostly use the C programming language
- We will review several derived data types: pointers, functions, structures, unions and enums
- You will learn common data structures: arrays, linked lists, stacks, queues, trees, graphs and hash tables.
- You don't need to buy a textbook for this course but the book by King (C Programming 2e, W W Norton 2008) is worth having anyway.