Overview

Data structures with C++ (CSC 240)

Marcus Birkenkrahe

August 13, 2024

Contents

Course Overview

Syllabus available on Canvas and on GitHub. This file was rendered as HTML presentation from the Org-mode original. Exporting into many different file formats is one of the strengths of Org-mode. Your handout is a PDF, this is HTML, on GitHub you find Markdown or raw Org-mode. The Reveal-based HTML is amazing.

Instructor Introduction

- Background and experience
- Course objectives and expectations
- I've not taught data structures or algorithms before.
- It's been a bit of a struggle because it's complex.
- There's not one textbook so I had to read 12 or more.
- Lots of decisions to be made along the way like C/C++.
- Can be taught entirely theoretically (not very fun).
- My approach is exactly as in CSC 100: learning by doing.

General Course Information

- Meeting Times: Tuesday & Thursday at 9:30-10.45 AM (75 min)
- Meeting place: Derby Science Building computer lab room 239
- Professor's Office: Lyon Building computer lab room 104
- Phone: (870) 307-7254 (office) / (501) 422-4725 (private)
- Office hours: by appointment MWF 4pm, Tue 3pm, Thu 11 am & 3 pm

Please book through my Google appointment calendar if you can: `https://tinyurl.com/fall24-office-hours`

Books that I used

- C Programming - A Modern Approach by King (W W Norton, 2008)
- Mastering algorithms with C by Loudon (O'Reilly, 1999)
- Algorithmic Thinking (2e) by Zingaro (NoStarch,2024)
- Little Book of Pointers by Collingworth (bitwisebooks, 2019) You find plenty of other sources in the handout.

  1. Secondary sources (recommended):
     - Learn C the Hard Way by Shaw (Addison-Wesley, 2015)
     - 100-page C++ language reference: Rook's Guide to C++
     - 800-page C++ language reference: C++ Crash Course
  2. 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)
  3. Recommended Videos:
     - C Programming for Beginners @Udemy
     - Advanced pointers @Udemy
     - "C++ Data Structures & Algorithms @Udemy
     - Helfrich's YouTube videos on C++ Data Structures
     - freeCodeCamp course on Data Structures with C++

Course Objectives

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

- Help you understand how data structures help you solve programming problems.

Three parts:

1. Extension of some known topics: arrays, functions, pointers
2. Dynamic memory allocation and strings in C and C++
3. Different data structures and their interfaces
4. Solving algorithmic problems with different data structures

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
- Those interested in system and network programming (cybersecurity)
- Those interested in moving on to C++/C# (game programming)

This is part I of a two-part course on algorithms and on smart programming. You need this for any software engineering related job interview, and it will be useful in any IT related job.

Student Learning Outcomes

- Understand major data structures and their applications
- Implement various data structures in C (with C++ / Python / R / SQL / bash).
- Analyze the efficiency of data structures and algorithms (without Big-O)
- Apply algorithmic thinking to understand and solve complex problems
- Design and develop efficient software solutions in a realistic infrastructure setting

"Big-O" refers to a way of measuring the efficiency of an algorithm by estimating how many operations it will take to complete. Some courses start out with discussing this but I might park it until CSC 265.

Course Requirements

- Prerequisites: Introduction to Programming (CSC 100 or CSC 115 or CSC 109, and MTH 101) - know the basics & understand numbers.
- Willingness to engage in problem-solving and critical thinking
- Willingness to put in substantial time to program on your own

Discrete maths might also include a discussion of graph theory, combinatorics, logic and proof techniques, and basic probability.

Grading (see also Rubric)

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

There will be 1 review test (open book & remote) per week, 2 mandatory program assignments, and other assignments for those who are bored. There is also now a student club for programming, whose adviser I am where you can take your programming prowess and get your fill. The final exam will consist of test questions you've seen, and you can use it to improve your grade.

Learning Management System

- Use of Lyon's Canvas installation (use the calendar to keep time!)
- Course materials, Assignments, Gradebook, Zoom recordings
- Course links, Syllabus, Announcements, Course Links

Do you know how to use/read/link the calender to your Google Calendar?

Google Chat Space

- Much preferred over email for questions
- I'm online and working most of the time
- Use Google chat to share your stuff, too

I don't mind if you contact me at impossible hours via Google chat because I often work at impossible hours. Don't be cross if you don't get an answer right away. Tell me which course you're referring to and attach enough information so that I don't have to keep asking too many questions, and/or just talk to me during office hours or after class.

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

We'll up our GitHub game this term, as well as our Linux and Emacs game. More details later.

Questions with Answers

When I wrote this, I had several questions that I needed to answer myself - here they are - the long answers are in the PDF. Please ask your own questions now!

What if you don't know (or remember C or C++?

- Any introductory programming class is OK
- Complete the entry quiz (Aug 19, 11:50pm)
- Check out: freeCodeCamp (video, 3 hrs.), and The Rook's Guide to C++ (book, 100 pg.)

You should have taken introductory programming in Java or Python or C/C++: Both Java and Python are high-level languages, but C/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/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 few weeks of the course. If you had Java, you already know a lot about Object-Oriented Programming (OOP).

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

- You can use your personal virtual Linux box from anywhere using `ssh` and graphics
- You can install Linux via dual boot / VirtualBox or (on Windows) with WSL
- Use GitHub as a central repo and clone to your local machine(s) as I do it.

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.

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

- C/C++ is compiled & much, much faster than Python or R
- C/C++ offers memory allocation and deallocation control
- C/C++ have large Standard (Template) Libraries available
- C/C++ used in industry for performance-critical applications (games, space, robotics)

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).

But most programs are in C not in C++ - why is that?

- Object-orientation adds (unnecessary) layers of abstraction
- The programming language should not get in the way of problem solving
- Once you understand it in C, you can take your knowledge anywhere
- See also: Bad Boy Mower story, August 2024

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 store, 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.

Another data point: tell Wyatt's story (2014 metal sheet cutter) - A couple of days ago, a former student of mine showed me around the factory where he develops new high performance lawn mowers (that's big business in the US, believe it or not). He showed me a table where he cuts metal sheets so that he can build new things. The machine was from 2014, and the computer and the software for it were insulated from the Internet and any other network because the firmware had not been updated since 2014 - he said that was quite common for many other large machines that they have. So any update has to be developed by them and transferred to the machine manually with a thumb drive - and often they don't do it or badly because it's so hard to understand (& maintain & improve) the original programs that came with the machine. This struck me as another nice application for literate programming.

What are my favorite programming languages?

- Lisp (Emacs - 1980s)
- C/C++ (computational physics - 1990s)
- SQL (databases - 2000s)
- R (data analytics - 2010s)
- Python (machine learning - 2020s)

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).

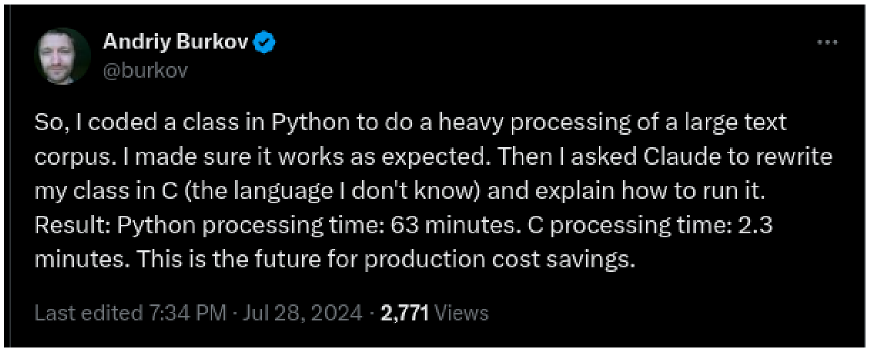
I learnt these languages at different times in my life: Lisp because of Emacs (late 1980s), C++ in the 1990s, SQL in 2005, R in 2019, and Python in 2022. C++ has developed massively since then though and I am more comfortable with its subset C than with many of the modern developments of the language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

How well do I know them? I think: Lisp (5%), C/C++ (30%), SQL (50%), R (50%), Python (25%) - but these estimates may be way off, too!

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

- Python consumes 76 x more energy than C
- Python is 72 x slower than C
- Python abstracts memory and performance management away

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.



Andriy Burkov
@burkov

So, I coded a class in Python to do a heavy processing of a large text corpus. I made sure it works as expected. Then I asked Claude to rewrite my class in C (the language I don't know) and explain how to run it. Result: Python processing time: 63 minutes. C processing time: 2.3 minutes. This is the future for production cost savings.

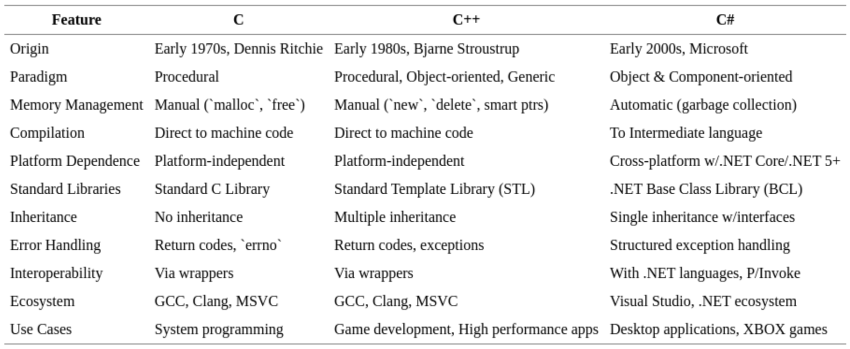
Last edited 7:34 PM · Jul 28, 2024 · 2,771 Views

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 - which is critical for some of the most popular applications like AI and Large Language Models. The only real reason to pick Python is that it's easier to learn and use. But that's also a reason to pick C because so many more people are fluent in Python than in C.

Here is ChatGPT's estimate: C (1.5-2 mio), C++ (4-5 mio), C# (6-7 mio), Python (10-12 mio). So with C you're 5-6 times more valuable but in the end it's also much more difficult to become a master at C.

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.

What's the difference between C, C++ and C#?



Feature	C	C++	C#
Origin	Early 1970s, Dennis Ritchie	Early 1980s, Bjarne Stroustrup	Early 2000s, Microsoft
Paradigm	Procedural	Procedural, Object-oriented, Generic	Object & Component-oriented
Memory Management	Manual (`malloc`, `free`)	Manual (`new`, `delete`, smart ptrs)	Automatic (garbage collection)
Compilation	Direct to machine code	Direct to machine code	To Intermediate language
Platform Dependence	Platform-independent	Platform-independent	Cross-platform w/.NET Core/.NET 5+
Standard Libraries	Standard C Library	Standard Template Library (STL)	.NET Base Class Library (BCL)
Inheritance	No inheritance	Multiple inheritance	Single inheritance w/interfaces
Error Handling	Return codes, `errno`	Return codes, exceptions	Structured exception handling
Interoperability	Via wrappers	Via wrappers	With .NET languages, P/Invoke
Ecosystem	GCC, Clang, MSVC	GCC, Clang, MSVC	Visual Studio, .NET ecosystem
Use Cases	System programming	Game development, High performance apps	Desktop applications, XBOX games

The list shows the simplicity of C vs. C++ vs. C#, and the independence on complicated paradigms and commercial solutions - which translates into deeper understanding and freedom from fads.

Here is a short video that summarizes C vs. C++ vs. C# (conaticus, 2022). And here is another one by a YouTuber who specializes in coding interview preparation, and who prefers C because it is devoid of external dependencies - essentially useful on its own (NeedCode, 2024).

Can this course help me break into Game Programming?

- Data structures and algorithms enhance computing performance
- Speed is critical for (multiplayer) games
- C/C# are more directly relevant

Only in so far as data structures and algorithms are performance enhancing choices, which are critical when programming games. Other than that, C++ and C# are more directly relevant for game development.

Can this course help me break into Cybersecurity?

- Emacs = the ultimate hacker editor
- Linux = the dominant server OS
- C = the system programming language

Absolutely: the mixture of Emacs + Linux + C is the winning solution for cybersecurity.

How should you study for this course?

- Code every day and participate actively
- Complete assignments early
- Make up your own exercises and programs
- Focus on fundamentals, simple examples
- Drill yourself with tests & memorize
- Build a code & notebook repo on GitHub

1. Code every day no matter how little.
2. Review lecture notes and notebooks.
3. Participate actively in the class.
4. Form study groups and/or join Lyon's Programming Club.
5. Complete assignments well before the deadline.
6. Practice literate programming by documenting your process.
7. Focus on fundamentals, simple examples, solid understanding.
8. Drill yourself using the weekly quizzes, make small examples.
9. Seek help when needed, on the chat or during office hours.
10. Build a code & notebook repository at GitHub for your resume.

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

Course content

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

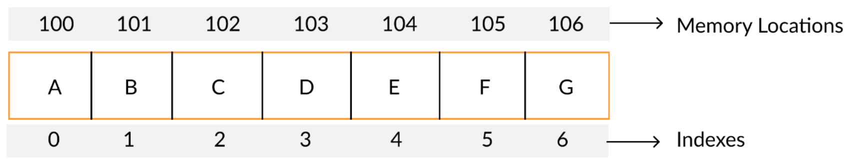
Data types classify the type of data a variable can hold, and the operations that can be performed on it.

Primitive Data Types

- `int`: Integer type
- `float` / `double`: Floating-point types
- `char`: Character type
- `void`: Special purpose type

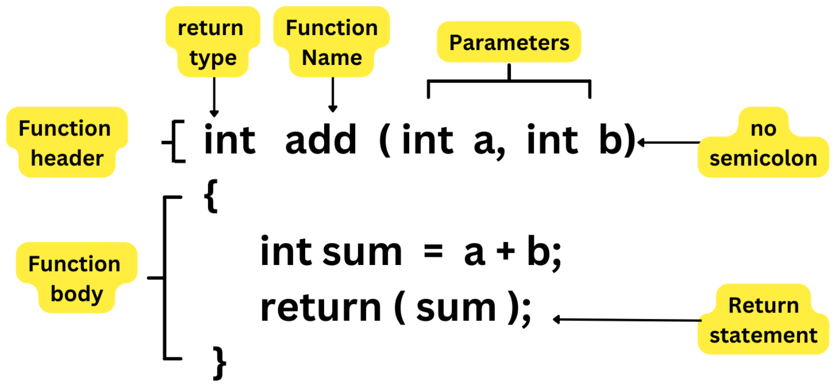
Derived Data Types

Array: Collection of elements of one type



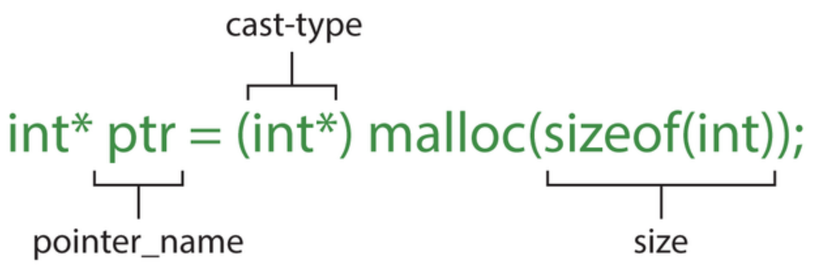
```
int arr[10];
```

Function: Does a job



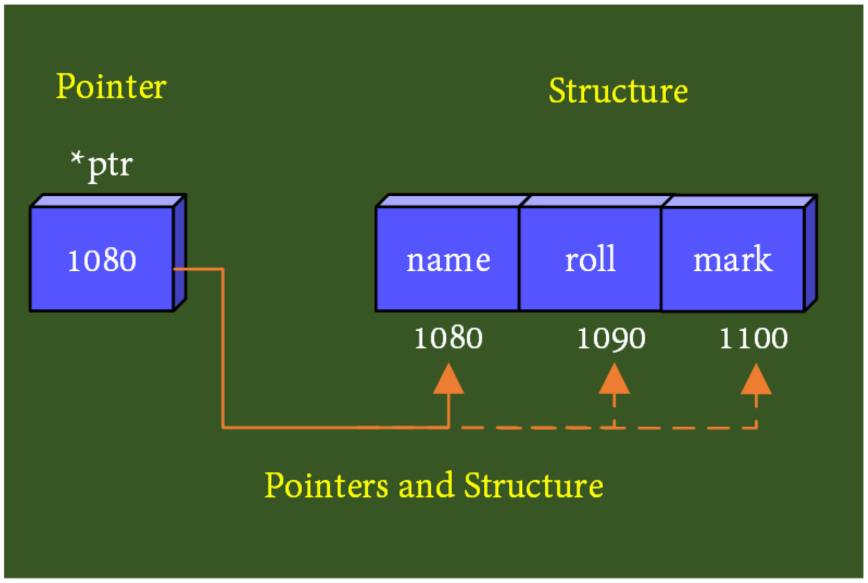
```
int func(int, float);
```

Pointer: Stores memory address



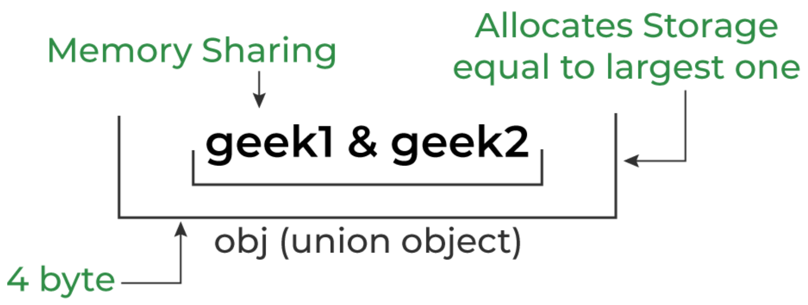
```
int *ptr;
int **ptrPtr;
int ***ptrPtrPtr;
```

Structure: Groups variables of different types



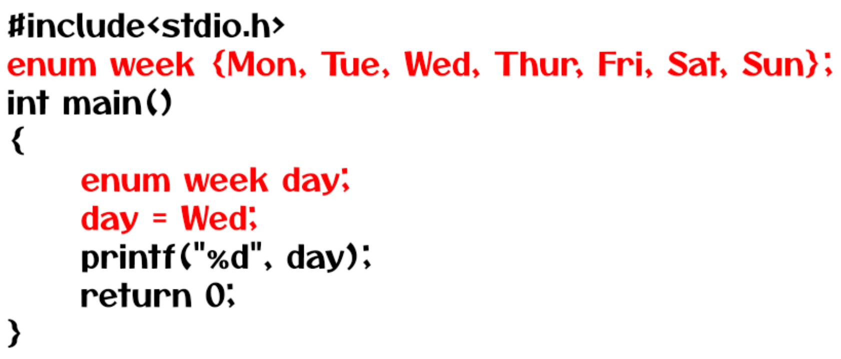
```
struct Person {
  char name[50];
  int roll;
  bool mark;
};
```

Union: Shared memory structures



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

Enumeration: Collection of constants



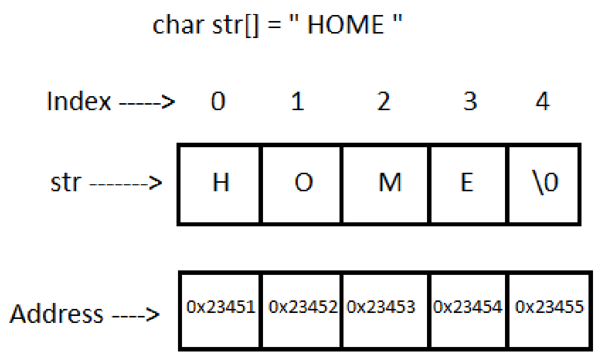
```
#include<stdio.h>
enum week {Mon, Tue, Wed, Thur, Fri, Sat, Sun};
int main()
{
    enum week day;
    day = Wed;
    printf("%d", day);
    return 0;
}
```

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

Data Structures

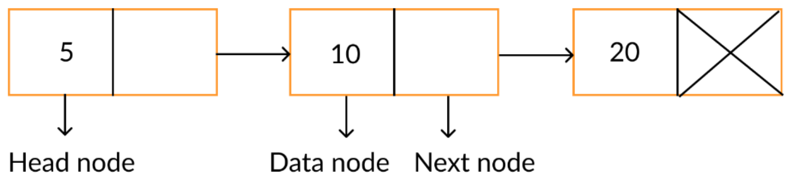
Data structures organize and store data for efficient access and modification.

Arrays: Collection of same type elements



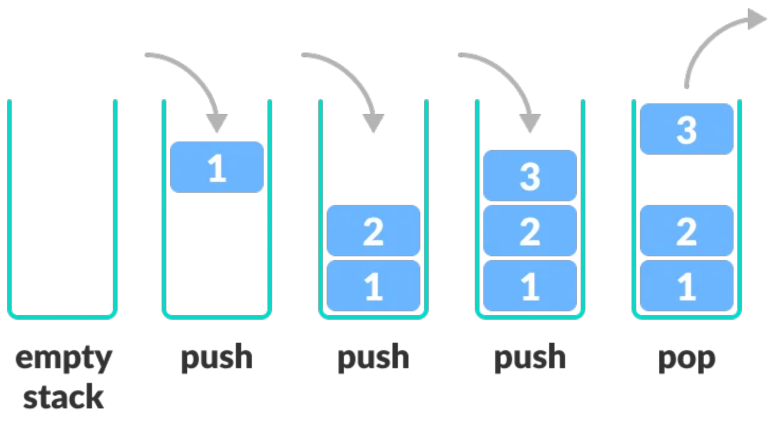
```
int arr[10];
```

Linked Lists: Each element points to the next element



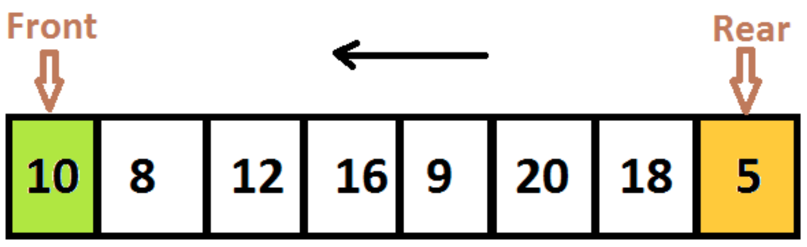
```
struct Node {
  int data;
  struct Node *next;
};
```

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



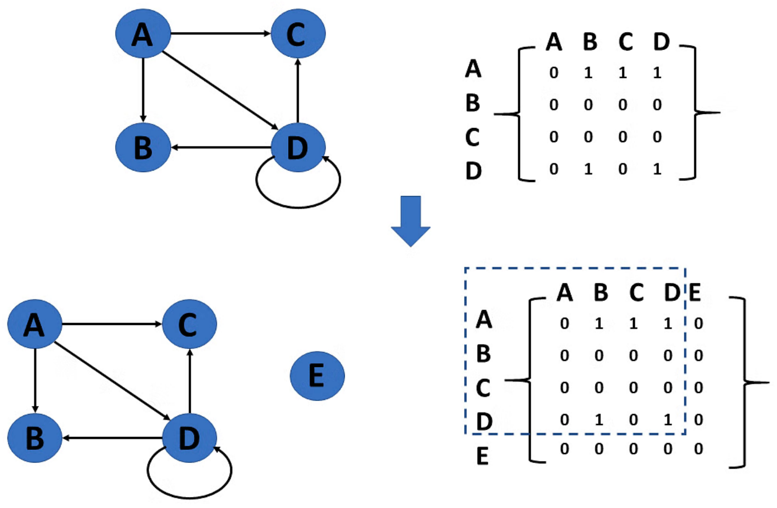
```
#define MAX 100
int queue[MAX];
int front = -1, rear = -1;
```

Trees: Hierarchical data structure

../img/tree.gif

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

Graphs: Collection of nodes and edges

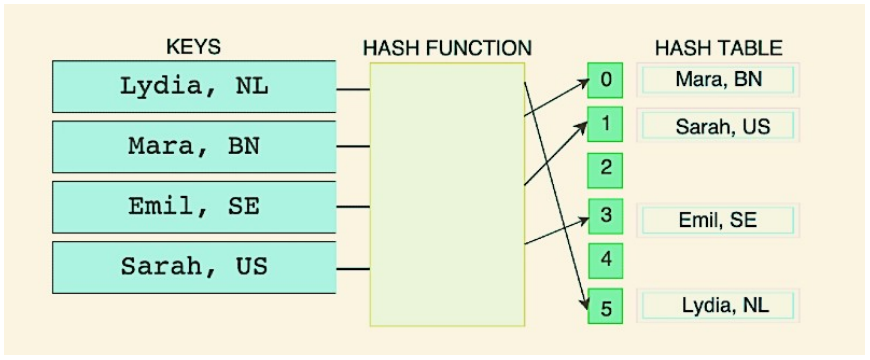


$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 1 & 1 \\ B & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 \\ D & 0 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 1 & 0 \\ B & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 1 & 0 & 1 & 0 \\ E & 0 & 0 & 0 & 0 & 0 \end{array}$$

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

Hash Tables: Implements an associative array



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

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 & data science = "maker spaces"
- 15 minutes per day beats 1 hour per week
- Professionals rely on good toolkits
- 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

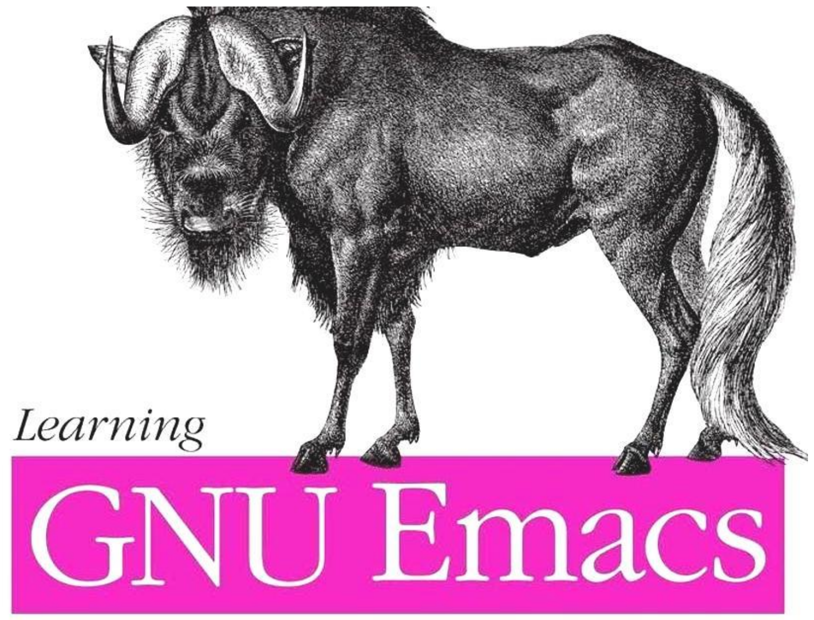
Linux - the world's premier Operating System

- Linux server for exclusive use by CS and DS
- If you completed CSC 420 you're well prepared
- You get your own Ubuntu LTS 22.04 system
- Access from anywhere with Internet
- Daily use = edge in job market
- 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 best 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:

What is special about Emacs?

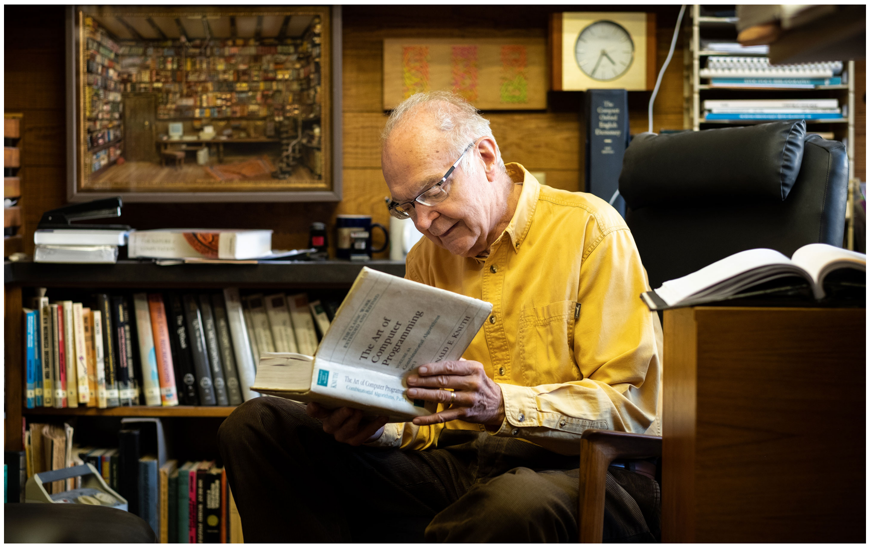


Emacs is a self-extensible, free, open source editor written in a Lisp dialect, and first published in the 1980s by Richard Stallman. It is a hacker- and power-user tool because of its customizability and openess. The vanilla version can be downloaded from gnu.org/software/emacs. Famous applications: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

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.

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, which dominates technical and scientific publishing.

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

```
#+begin_src C++ :main yes :includes <iostream> :namespaces std :results output
  cout << "Hello, World!" << endl;
#+end_src
```

This is what you're saving:

```
// 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).

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

```
#+begin_src C
  puts("Hello, World!");
#+end_src

#+RESULTS:
: Hello, World!
```

Works because of the `#+property:` file header:

```
#+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!
```

What are the differences between C and C++ blocks?

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

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

Why are we using Emacs + Org-mode (not VSCode or Code::Blocks IDEs)?

- Customizability and extensibility
- Integrated file management and shell
- Literate programming support
- Powerful text editing environment
- 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.

- Premier software engineering platform
- I used it since 2010 for development
- Uses Git for development version control
- Used for code sharing and AI assisted coding

- 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.
- 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").

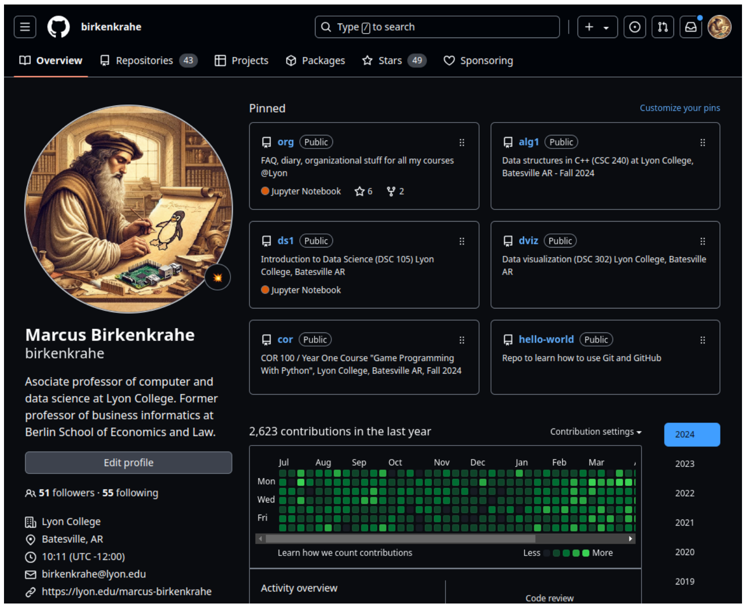
GitHub Todo Now

1. Register with GitHub
2. Complete Hello World Exercise
3. Fork my course materials
4. Create public repo for your code
5. Submit screenshot of forked repo

- 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).



Pinned

org Public
FAQ, diary, organizational stuff for all my courses @Lyon
Jupyter Notebook 6 2

alg1 Public
Data structures in C++ (CSC 240) at Lyon College, Batesville AR - Fall 2024

ds1 Public
Introduction to Data Science (DSC 105) Lyon College, Batesville AR
Jupyter Notebook

dviz Public
Data visualization (DSC 302) Lyon College, Batesville AR

cor Public
COR 100 / Year One Course "Game Programming With Python", Lyon College, Batesville AR, Fall 2024

hello-world Public
Repo to learn how to use Git and GitHub

Marcus Birkenkrahe
birkenkrahe

Asociate professor of computer and data science at Lyon College. Former professor of business informatics at Berlin School of Economics and Law.

51 followers · 55 following

Lyon College
Batesville, AR
10:11 (UTC -12:00)
birkenkrahe@lyon.edu
https://lyon.edu/marcus-birkenkrahe

2,623 contributions in the last year

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.

1. 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.

2. 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" and a short tutorial.

3. 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.

4. 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.

GitHub Hello World Exercise

1. Create hello-world repo and set it up
2. Create a branch
3. Make a commit changes
4. Open a pull request
5. Merge your pull request

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

1. 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.
- 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`.

2. 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.
   - 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.

3. 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. You can make as many changes and commits as you like.
   - Your two branches, `main` and `readme-edits` have now diverged.

4. 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, a line-by-line comparison.

5. 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.

Summary I

- 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

Summary II

- 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.