Engineering 3891

Learning outcomes:

- problem-solving approach
- practical language skills

In this course, we'll consider both the intellectual and the practical sides of computer programming.

Programming is _____. It's the application of an intellectual approach to practical problem-solving (which is actually a pretty serviceable definition of _____!).

We also, as developers of real software, need to be familiar with _____ _____. We'll do this too. As a side benefit, note that a solid grasp of C++ is a highly marketable skill.

Problem solving

- Intellectual content
- Language-agnostic
- Broadly applicable

- explain and differentiate fundamental abstractions and types,
- read and understand well-written programs,
- evaluate the correctness of pseudocode and C++,
- diagnose common errors in memory management and simple algorithms,
- design simple object-oriented systems; and
- synthesize correct and idiomatic C++ with simple STL containers.

All of the important ideas in this course can apply to many programming languages — or none at all! In fact, an awful lot of the material in this programming course doesn't have to have be applied to computers. A study of abstractions, algorithms and types can be _____ _____ and _____ to logistics, writing procedures and manuals, etc.

However, we will focus on computer programming. This isn't a C++ course: it's a foundational *programming* course. This slide highlights the aspects of the course outcomes that aren't necessarily C++-specific, and it's most of them.

Practical skills

- C++
- Guided exploration: labs
- Practice: problems, assignments

- explain and differentiate fundamental abstractions and types,
- read and understand well-written programs,
- evaluate the correctness of pseudocode and C++,
- diagnose common errors in memory management and simple algorithms,
- design simple object-oriented systems; and
- synthesize correct and idiomatic C++ with simple STL containers.

However, we can't teach ideas in a vacuum. Often, the best way to _____ is to _____, and in this course we will apply ideas using the C++ programming language.

We will also have some practical, hands-on labs to guide you through the exploration of real running software and important software development tools.

Throughout the course I will provide problems and exercises for your own personal practice. The assignments are meant to do the same thing: encourage practice. I would strongly suggest that you take any practice opportunity you can get.

How to succeed

Ask questions

Practice!

- exercises from lectures and tutorials
- practice problems from textbook(s)
- problems inspired by other courses

If you're here, _____. Getting through Engineering One means that you have all the ability you need to pass this course.

However, _____. I will do my part: I will attempt to make the material as _____ as possible and give you the _____ you need to succeed. However, you must take _____ of your progress.

I have found that the students who practice programming tend to succeed. This doesn't have to require a lot of time. I challenge you to spend _____ working through exercises and problems. It's only when you _____ knowledge that you find out whether or not you really _____ it.

How to succeed

Components

- Lectures
- Labs
- Tutorials

Labs

Unlike ENGI 1020, our labs have nothing to do with assignments. Instead, they're more like traditional Engineering labs: _____ of course material and exposure to _____.

Tutorials

We'll use the tutorial slot for different kinds of things in different weeks: set exercises, unstructured question-asking, etc. Worth your while.

In the first week, I'll host a _____ to help you get a decent C++ compiler running on your own computer. It's not *required* for the course, but it's *highly recommended*.

How to succeed

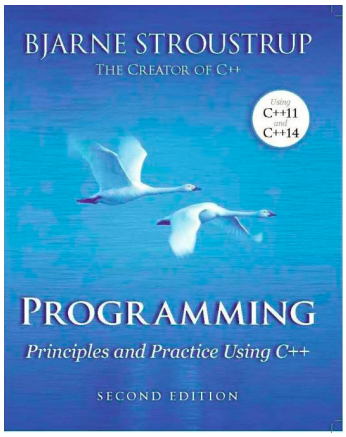
Communication:

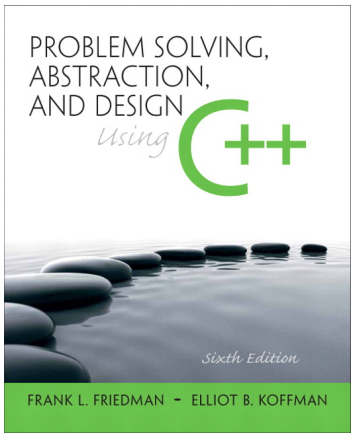
- Office hours (10-11am, Monday and Friday)
- Appointments
- D2L forums
- Course website

Office policy: my door is usually open — if it is, come knock!

Textbooks

- None *required*, but useful references:





This course doesn't follow a textbook: it's about preparing you for work terms and further courses in the ECE program. However, I do recommend buying a textbook as a reference and source of further example problems.

You might already have Friedman & Koffman, but I really do recommend Stroustrup for its use of modern C++ features. It's also one of the cheapest textbooks you'll ever buy --- you can get it in paperback for $65 or Kindle Edition for $47. There's no reason not to get this book.

Academic Integrity

Intellectual independence

Collaboration

"I did this"

The University has standards of honesty and integrity for all its students, but you are also _____ with ethical obligations. When you complete a piece of evaluation, you must be confident that it was done ethically: it must be _____ _____.

Helping each other learn is important. Help each other understand the material and the assignments, but when you put your name on a submission, you are saying, _____. Think about _____: if you come to me with questions, I will:

1. talk through the problem with you,
2. ensure that you understand the relevant concepts,
3. ask you questions about the assignment and your approach and even
4. ask you, "what next? and then?".

That process is intended to be helpful, but when you leave my office, I will have helped you *find* the solution, not *given* you the solution. I will certainly _____ let you look at my solution! Your interactions with each other over labs and assignments should work like that: _____ _____.

Topic 1:

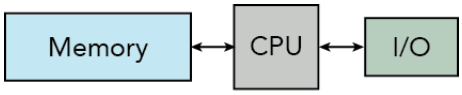
Abstractions and programming paradigms

Abstractions

Two views:

- Mathematical
- Abstract machine

$$f(x) = \begin{cases} x, & x \geq 0 \\ y, & x < 0 \end{cases}$$




```
a = true
b = 42 > x
z = x + y
c = a or b
goto z if c
```

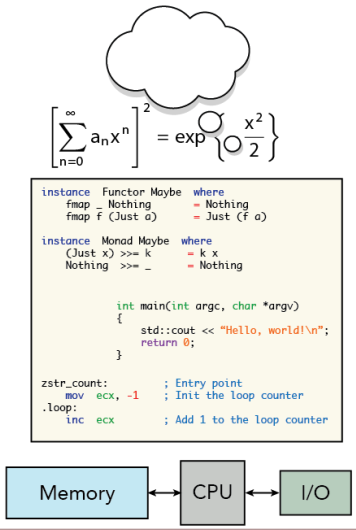

The mathematical view of the world is extremely *abstract*: it _____ everything until we reach a problem that can be solved mathematically. If you've heard the joke about _____ _____ you'll know what I mean by generalization.

Another view is the classic Engineer's view: "give me a machine that I can tinker with". In this view, we have bits that are organized in memory and we have systems for manipulating them and interacting with peripheral components. This is the foundation for computation, but we're so deep in detail, it's easy to _____.

So, which approach is the best? It depends on the problem! That is, it depends on _____ _____ we need to be. Right now, I need to understand that each of you is a _____ _____. When you come to my office hour, however, I need to understand more about you as an individual: what you're good at, struggling with, etc. That level of detail would be _____ in a large lecture like this.

Fortunately, we don't have to choose between just two approaches...

Abstraction spectrum



Abstractions can be found on a spectrum, from the very abstract at the top (of this slide) to the very practical at the bottom. Sometimes, the best answer is an equation. Sometimes, it's a circuit. In between, we have many different kinds of programming. Programming can be done quite abstractly (almost pure math) or in a very detail-oriented way (turning individual bits on and off). One of the reasons we're using C++ is because _____.

Programming languages

- To communicate with computers
- To communicate with people
  - Other developers
  - Test/verification teams
  - Financial traders
  - Yourself!

Most people understand that programming languages are used for communicating with computers: for giving them instructions. What often isn't appreciated is that programming languages are also about communicating with... _____!

People will spend more time reading your code than compilers. Other developers will need to understand what you've done and _____. Other software development professionals will need to be able to understand what your code does and what it was supposed to do.

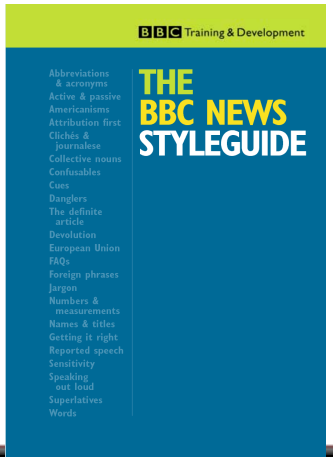
Seriously! In 2010, the US SEC (Securities and Exchange Commission) proposed that vendors of Asset-Backed Securities (think of subprime mortgages) have to produce a Python program that describes their crazy-complex financial instruments:

> *We are proposing to require that most ABS issuers file a computer program that gives effect to the flow of funds, or "waterfall," provisions of the transaction. We are proposing that the computer program be filed on EDGAR in the form of* ***downloadable source code in Python****. … (page 205)*
>
> *Under the proposed requirement, the filed source code, when downloaded and run by an investor, must provide the user with the ability to* ***programmatically input the user's own assumptions*** *regarding the future performance and cash flows from the pool assets, including but not limited to assumptions about future interest rates, default rates, prepayment speeds, … (page 210)*

(see https://jrvarma.wordpress.com/2010/04/16/the-sec-and-the-python)

Programming style

Not "style", *style*:



When we talk about programming "style", we're (hopefully) not worried about fashion. Rather, we're talking about conventions that promote _____.

When a news organization produces a style guide, they're not telling their journalists how to dress, they're laying down rules for how "we" use punctuation and grammar, how "we" attribute sources, how "we" use controversial terms. That is, when you read the CBC, the Guardian or the New York Times, you should see internally-consistent meanings of terms like "activist" or "private military contractor". This makes the content _____.

Programming Paradigms

- Procedural
- Functional
- Object-oriented
- Logic

There are many ways to think about computer programming, but we're going to look briefly at four of the main ones and then spend most of the term on the first three.

Procedural programming

```
void drive(double distance)
{
  int speed = 40;  // or maybe less if we aren't going far?
  double time = distance / speed * MILLISECONDS;

  set_motors(speed, speed);
  delay_ms(time);
  set_motors(0, 0);
}
```

im·per·a·tive /əmˈperədiv/

adjective

1. of vital importance; crucial.
2. giving an **authoritative command**; peremptory.

Procedural programming should already feel familiar to you: it's what you did in ENGI 1020 or your first tinker with Basic or JavaScript. Procudural programming is about _____ _____ and then _____ _____.

For example, the `drive` function on this slide might make a robot drive a fixed distance forwards or backwards. If you wanted to write a procedure for solving a maze with a robot, you wouldn't want to repeat the "drive forward" procedure over and over — instead you'd use the procedure that's already been given. This is also true of non-computer programming, like an instruction manual.

One aspect of this programming style is that it's *imperative*: it consists of lots of commands that _____.

Functional programming

- Using functions as variables:

```
def triple(x: Int) = 3 * x

val some_numbers = List(1,2,3)
val bigger_numbers = x.map(triple)
```

- Popular in 1970s (e.g., Lisp), coming back:
  - Haskell, Scala, OCaml
  - C++!

Functional programming is more about _____ than the details of _____.

C++ acquired some really nice functional primitives in the 2011 version of the standard — we'll come back to these towards the end of the course.

Part of C++'s strength as a lanuage is that it is *multi-paradigm*: it's so flexible that you can do just about any kind of programming in it. The downside of this flexibility is that people *do* try to do every kind of programming in C++, which can make for difficult-to-read code. A critical part of any company or project style guide is telling developers what language features are or are not used.

Object-oriented programming

Wrap code and data together:

```
class Shape {
public:
  virtual double area() = 0;
};

Shape *a = new Circle(4.0);
Share *b = new Rectangle(0, 0, 10, 10);
double total_area = a->area() + b->area();
```

Object-oriented programming is centred around *objects*, which are bundles of _____ _____.

For instance, in this example we can have different type of `Shape` object, each of which knows how to calculate its own area. A `Circle` object will contain a radius and some code that knows how to calculate the area from it ($2\pi r$), whereas a `Rectangle` object will contain a length and width, together with its own area computation code ($l \times h$).

You've already seen one example of an object type in ENGI 1020: `std::string`. However, they can get much more interesting and powerful than that! We will spend a lot of our time this term on object-oriented programming.

Logic programming

Declare things to be true:

```
arc(a,b).           arc(b,c).
arc(a,c).           arc(a,d).
arc(b,e).           arc(e,f).
arc(b,f).           arc(f,g).
```

Query the program for facts:

```
?- arc(a,X). /* arcs from node a got to node X */
X = b
X = c
X = d
```

Used in AI, theorem proving

Finally, logic programming is a way of reasoning about true statements. In logic programming, we don't tell the computer *how* to figure out whether or not there is an arc from `a` to any other point, we just program it with facts and enough information about relationships that it can go off and search all of its knowledge for a solution.

In the 1970s-1990s, AI researchers got really excited (and maybe still are?) about this kind of thing, convinced that we'd replace doctors with computers. They're partially right (people do use WebMD), but not as much as they'd hoped. It turns out that people are still useful and hard to replicate with computer programs.

Summary

- Abstraction spectrum
- Programming languages
- Programming paradigms