CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 1
August 29, 2018

About This Course

Where to find information

Information about this course is posted on the course website:

https://zoo.cs.yale.edu/classes/cs427/2018f/

- Syllabus.
- One of the online textbooks, Exploring C++ by Alice Fischer.
- Lecture notes.
- Code samples.
- Homework assignments.

The course uses Canvas for assignments and announcements. It also contains some links to the main course website on the Zoo.

The syllabus contains important additional information. Read it!

Course mechanics

You will need a Zoo course account. It should be created automatically when you register for this course as a Shopper or Student. The login credentials will be your standard Yale NetID and password. Test it now!

Assignments will be submitted on Canvas using the Assignments tool. Detailed instructions will be provided.

Course Requirements: Homework assignments (∼40%), midterm exam (∼20%), final exam (∼40%).

Course goals

Learn how to answer the following questions:

1. *Who* programs and why?
2. *How long* does a program last?
3. *What* are the characteristics of a good program?
4. *When* do good programs matter?
5. *How* does C++ help one write good programs?

Discussion.

Who programs and why?

People program for different reasons.

1. To get answers to particular problems of interest.
2. To avoid repetitive work when solving several instances of the same problem.
3. To provide tools that others can use.
4. To produce software of commercial value.
5. To provide a mission-critical service.

How long does a program last?

Three facetious answers:

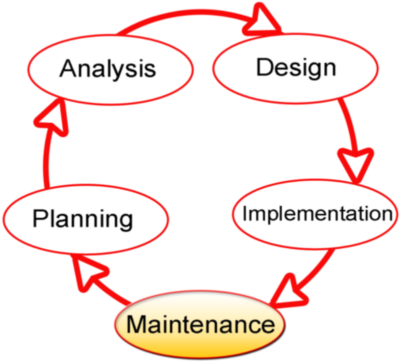
1. Until it stops being useful.
2. Until nobody maintains it.
3. Far longer than was originally anticipated.

What are the characteristics of a good program?

1. **Correctness:** Does what is intended.
2. **Robustness:** Handles bad input gracefully.
3. **Security:** Resists malicious exploits.
4. **Efficiency:** Makes cost-effective use of computer resources.
5. **Isolation:** Prevents unintended interactions within itself and with its hardware and software environment.
6. **Cleanliness:** Embodies a direct connection between the task and the solution.
7. **Clarity:** Can be comprehended rapidly by humans.
8. **Maintainability:** Has complete test suite. Modifications cause expected changes to behavior.

When do good programs matter?

The program development life cycle is a continuous circular process:



By Dzonatas (Own work) [CC BY-SA 3.0 or GFDL], via Wikimedia Commons

Important imperatives for life cycle management

1. Modularity – group related parts together at each level
2. Non-interference – protect unrelated parts from each other
3. Produce clean, simple, straightforward, understandable code
4. Avoid replicated code fragments
5. Avoid unnecessary hardware and OS dependencies
6. Follow recognized style guidelines
7. Document your work approrpiately

How does C++ help one write good programs?

1. Language and core library are standardized and documented.
2. Classes, functions and templates support modularity.
3. Privacy and const attributes protect and isolate code.
4. Constructors/destructors ensure object coherence.
5. Inheritance and templates help one avoid replicated code.
6. Exceptions separate error handling from normal program flow.
7. Operator extensions and qualified names improve readability.
8. Inline functions, const, reference types, move semantics, stack-allocated objects, and static type checking permit better code efficiency.

Topics to be Covered

Major Areas

1. Foundations of C++ (basics of objects and classes).
2. Software toolset.
3. C++ storage model: paradigms for object creation and deletion, pointers, references, lvalues and rvalues, move semantics.
4. Software design process
5. Programming for reliability, testing, debugging.
6. Programming for efficiency.

Course goals - practical

- Learn how to follow instructions, and how to question them if you think they are wrong.
- Learn how to get a big job done one module at a time.
- Learn how to use a reference manual.
- Learn how to design for efficiency and reliability.
- Learn how to test, analyze, and debug code.
- Learn how to present your work in a professional manner.
- Become proficient at C++ programming, starting with a knowledge of C.

Course goals - conceptual

- Learn what object-oriented programming is – and isn't.
- Learn what constitutes good object oriented design.
- Learn how C++ differs in syntax and semantics from standard ISO C
- Learn how C++ provides better support OO-programming than other object-oriented languages such as Python, Ruby, and Java.
- Learn about classes, objects, type hierarchies, virtual functions, templates, and their implementations in C++.
- Learn the principles behind the exception handler and when and how to use it.
- Learn how to use the standard C++ class libraries.

Kinds of Programming

Problem solving

Desired properties of programs for solving problems:

- Correct outputs from correct inputs
- Succinct expression of algorithm
- Simple development cycle

Beginning programming courses tend to focus on programs to solve small problems.

Industrial-Strength Sofware

- Thousands of lines of code
- Written by many programmers
- Over a large span of time
- Deployed on a large number of computers
- Used by many people
- With different architectures and operating systems
- Interacts with foreign code and devices
- Evolves over time

Software Construction

Desired properties of **industrial-strength** software:

- Correct outputs from correct inputs
- Robust in face of bad inputs; stable; resilient
- Economical in resource usage (time and space)
- Understandable and verifiable code
- Secure
- Easily repurposed
- Easily deployed
- Maintainable

Why C++?

C/C++ are popular

According to the TIOBE Index[1] for August 2018, C and C++ are the 2nd and 3rd most popular programming languages, behind only Java.

[1] See TIOBE Index

C/C++ is flexible

A typical software system is built in layers on top of the raw hardware:

5 Application
4 Application support (libraries, databases)
3 Virtual machine [optional]
2 Operating system
1 System kernel
0 Hardware

C/C++ are almost universally used to implement code at levels 1-4. Java is popular for levels 5, but recent additions to C++ make it increasingly attractive for level 5 applications as well.

Advantages and disadvantages of C++

- C++ allows one to construct stable, reliable, industrial-strength software.
- Many programming errors are detected by the compiler, resulting in reduced debugging time after the first successful compile.
- C++ is “closer” to the machine, making it possible to have better control over resource usage.

Downsides of C++

- C++ is a big powerful tool that can easily be misused.
- The C++ programmer must pay attention to how memory is managed. Mistakes in memory management can lead to catastrophic failures and security holes.
- C++ programs may be longer than other languages because the programmer learns to describe her program more fully.

C++ Programming Standards

Five commandments for this course

From Chapter 1 of Exploring C++ and elsewhere:

1. Use C++ input and output, not C I/O, for all assigned work.
2. Don't use global variables – you will lose points. If you think you need one, ask for help. Your class design is probably defective.
3. Don't use getter and setter functions. Rather, provide a public interface with semantically meaningful functions for querying and updating the state of an object.
4. Don't believe a lot of the rules of thumb you may have learned in a Java course or that you read on the internet. Java is different from C++ in many important ways, and many Java books do not focus on industrial strength programming.

Can is not the same as should!

From Chapter 1 of Exploring C++:

- C++ is a very powerful language, which, if used badly can produce projects that are badly designed, badly constructed, and impossible to debug or maintain.
- Your goal is to learn to use the language well, and with good style.
- Please read *and follow* the style guidelines in Section 1.2.
- Download the two tools files from the website.
- Read Section 1.3, about the tools library, and use this information to customize your own copy of the tools.

Rules for preparing your work

1. Every code file you submit must contain a comment at the top giving the name of the file, your name and netID, the course number, and the assignment number.
2. If your work is based on someone else's work, you *must* cite them at the top of the file and describe what part(s) of the code are theirs.
3. If you have started from a file that you obtained from someone else and it contains authorship/copyright information, you must leave that information in place.
4. If you have any doubts about the proper way to cite your sources, *ask*, don't just guess. Stay out of trouble.

Rules for submitting your work

1. All submissions must be done on Canvas.
2. Test every line of code you write. It is your job to verify that your entire program works. If you submit a program without a test plan and test output, the grader will assume that it does not compile and will grade it accordingly.
3. Compile and test your program on the Zoo before submission.
4. Supply a `Makefile` with your code so that a grader can type `make` and your code will compile and be ready to run.
5. Supply a `README` file that contains instructions to the grader on how to run and test your code.
6. Submit all files needed to compile your program, including copies files that have been provided for your use.