Syllabus (Fall 2018)

1 Official Yale course listing

CPSC 427 01 (10690) /	Fall 2018
CPSC 527 01 (12215)	Final exam 12/17/2018 at 7pm
Object-Oriented Programming	
Michael Fischer	
MW 4.00–5.15 WLH 119	

Object-oriented programming as a means to efficient, reliable, modular, reusable code. Use of classes, derivation, templates, name-hiding, exceptions, polymorphic functions, and other features of C++.

After CPSC 223.

2 Course Description

2.1 Industrial Strength Programming

Programming, broadly defined, is the activity of getting a computer to perform a desired task. It generally takes the form of constructing an artifact, called a *program*, which in turn instructs the computer in the sequence of steps needed to carry out the task. The program is expressed in a *programming language*.

The primary requirement of any program is that it *correctly* performs the desired task. For many small applications, the task is easily described, and the goal of the programmer is to get the job done as quickly and easily as possible. This is typical of homework assignments in beginning programming courses. The specification is provided by the instructor, and the overriding concern of the student is to have something to turn in by the deadline. Small throwaway jobs occur in the real world, too, where the primary goal is to get answers quickly to one-off problems. Succinct interpretive languages such as Python and JavaScript are often well-suited to such applications.

By way of contrast, large software systems typically have lifetimes measured in decades. They are built over time by teams of programmers. Programmers come and go. The people who create these systems are often not the same as the people who maintain them over the years. The systems must run correctly in a variety of hardware and software environments. They are not static but are continually evolving to meet changing needs.

They need to do much more than just produce correct outputs. They must also be:

1. **Convincingly correct**. It's not enough that the program *be* correct; there must also be confidence that it *is* correct. Such confidence is achieved through a combination of proper use of language constructs such as type systems to enforce constraints, formal verification tools, human understanding of the code, and thorough testing.
2. **Maintainable**. The program must be understandable by humans other than the original developers. This requires that the program be modular, clean, elegant, and well documented. As in any design process, style and judgment are important to the finished product.

3. **Efficient**. The program must perform its task within the available resources such as time, memory, communication bandwidth, etc.
4. **Reliable**. The program must be robust in the face of unexpected program inputs, runtime errors, and changes in requirements during a program's lifecycle. It must handle known risks predictabily and unknown risks gracefully. It is rarely acceptable for a large system to crash in response to error conditions.
5. **Secure**. The program must resist a malicious adversary who deliberately attempts to cause it to fail or misbehave or to otherwise violate its intended properties.
6. **Deployable**. The program's dependencies on its environment must be clearly specified and easily satisfied on target systems. Maintaining backwards compatibility as new features are added aids in achieving this goal.
7. **Free of code replication**. Code that is needed in more than one part of a program should appear only once and be referenced appropriately where needed. Replicated code sections are particularly difficult to debug and maintain. A bug that is fixed in one copy will remain unfixed in undiscovered replicas.

Since good software is expensive to construct and maintain, an additional goal is that it be *reusable* in other applications.

Experience shows that a disciplined software development process is needed in order to produce software that achieves these goals throughout its lifecycle. How to economically develop and maintain large systems that are useful in real-world applications is the realm of *software engineering*, covered by Yale course CPSC 439.

This course focuses on *software design and construction,* namely, the process of defining and implementing the architecture, components, interfaces, and other characteristics of the software artifact that implements the design specification.

2.2 Topics and Emphasis

This course will use the version of the C++ programming language known as C++17. C++ is large and complicated. It contains many features that *allow* it to be used to satisfy the requirements stated above, but they don't *ensure* good code. These same features can be misused to create code that is opaque and indecipherable. A good understanding of the language is necessary in order to reap its benefits.

The course will cover many C++ concepts, explaining for each what the purpose is, some of its intended uses, and some of the pitfalls it presents. Topics include design principles and standards, the C/C++ memory model, objects and classes, constructors and destructors, types, casts and conversions, operator definitions, name-hiding, restrictions on data modification, derivation and inheritance, abstract classes, polymorphism, virtual functions, multiple inheritance, templates, exceptions, and the C++ Standard Library. The course may also touch on some other major class libraries such as Boost, GTK+, and gtkmm. Other topics as time permits include advanced design patterns, programming for efficiency and testability, performance measurement, and debugging.

These topics will be reinforced by frequent programming assignments. The goal of the assignments will be to learn how to apply design principles to actual code. Submissions will be judged on their design, style, cleanliness, and understandability as well as on their functionality and on their adherence to specific requirements of the assignments.

3 Course materials

This course has no required textbook. Rather, supplementary material is either publicly available on the internet, isposted to the CPSC 427 Zoo web site for use by this class, or is a licensed online book for which free access is available to the Yale community through a licensing arrangement. See information about off campus access to learn how to access licensed materials when you are not on the Yale network.

Primary References:

1. Alice E. Fischer, David W. Eggert, and Michael J. Fischer, *Applied C and C++ Programming*, manuscript, 2018.
   This covers basic material on C/C++ program that is typically covered in introductory and intermediate programming courses and in data structures. If you don't know C already or would like a deeper understanding of it, then this is a good book for getting up to speed.

2. A. Fischer, *Exploring C++*, manuscript, 2009.
   This book gives a comprehensive overview of the tools that C++ provides for writing industrial strength programs. It's primary drawback is that it is already nine years old, so it doesn't cover important new features added to the language by the C++11, C++14, and C++17, standards.

3. Internet website `http://cplusplus.com` is a good source of introductory, tutorial, and reference materials on C++. It is generally up to date and accurate, and I encourage you to become familiar with it.
   Registration is not required to view the site, and because the site does not support secure web connections (`https://`), I would not recommend registration on an insecure site that requires sending passwords over the internet.

4. Internet website `https://cppreference.com` is a comprehensive reference site for C++. I find it more precise but less readable than `cplusplus.com`, but both are definitive places to learn about fine points of C++ and are highly recommended.

Other References: The following books are licensed by Yale for use by the Yale community. The links below will give free access to the books when you are on the Yale network. To access from off-campus locations, you will need to use the Yale VPN. To know that you have licensed access, you should see "Yale University" in the blue menu bar next to the "Personal Sign In" menu. If you don't see it, click on "Sign In", then on the "Start Using Safari" under the Academic License, and then try the original link again.

1. Scott Meyers, Effective STL, Addison-Wesley Professional, 2001, ISBN-13: 978-0-321-54518-3.
   This book is rather out of date but gives some advice that is still relevant.

2. Scott Meyers, *Effective Modern C++*. O'Reilly Media, Inc., 2014, ISBN-13: 978-1-4919-0399-5.
   This book explains many of the important new features of C++11 and C++14. It will be mainly useful during the latter part of the term.

Beware of the many web sites out there that have C++ information that is outdated, incomplete, or just plain wrong. Not only are many people misinformed about how the language actually works, but some important things have changed with the advent of C++17, so you should also be careful about trusting out-of-date information.

4 Course Websites:

This class will use two websites:

- Canvas: `https://yale.instructure.com/courses/39412`
- Zoo website: `https://zoo.cs.yale.edu/classes/cs427/2018f/index.html`

Canvas will be used for homework assignments and submissions, grading feedback, and emailed announcements. The Zoo website will be used for the syllabus, handouts, lecture notes, general announcements, and other course-related materials.

5 Course Mechanics

Prerequisite: The prerequisite for this course is CPSC 223 (Data Structures) or equivalent. *The ability to write a significant program in C, C++, or Java is required.* This course also assumes a familiarity with basic computer science concepts such as are covered in CPSC 201 and CPSC 202. Graduate students should have an equivalent background.

Requirements: Course requirements include programming assignments and/or written problem sets (∼40%), a midterm exam (∼20%), and a final exam (∼40%). The approximate weights of each in determining the course grade are subject to change depending on the number and difficulty of the assignments actually given. Graduate students taking the course will be expected to perform at a higher level than undergraduates and may be required to do additional work.

Assignments and other announcements: Written problem sets and programming assignments will be posted from time to time on the Zoo handouts page and will be announced using Canvas. Other course announcements will be posted on the Zoo website home page. It is your responsibility to check these pages frequently.

Help with technical questions: The graduate teaching fellows and undergraduate learning assistants will be holding scheduled office hours during the term as posted on the course web site. You are encouraged to meet with them with questions about the lectures, textbook, problem sets, and C++ programming generally. You may also send questions by email. Please copy the instructor on all such emails. This will often result in a quicker response, since whoever is available at the time can decide to answer it. The response will also go both to the TAs and to the instructor so everyone will know that the questions have been addressed.

Other Questions: All questions about assessment and grading should be taken first to the graduate TF. If the TF is unable to resolve your questions to your satisfaction, or if you wish to talk to me privately about any matter, you are always welcome to contact me, either by email or in person. Email is the preferred way to arrange an appointment with me.

6 Policies

Late policy: Assignments will be due at 11:55 pm on the night of the stated due date. Late work will generally be subject to a penalty of 5% per day late unless accompanied by a Dean's excuse. A 2-hour grace period following the *original* due date will be granted during which no late penalty will be assessed. However, there will be no grace period in counting the number of days late for assignments turned in after the grace period. Work more than 4 days late will not be accepted, but alternative means for making up missed work may be arranged on an individual basis with a Dean's excuse.

Please contact the instructor as soon as you find out that you are unable to submit work on time or to attend a scheduled exam so that suitable makeup arrangements can be made.

Policy on Working Together: This course follows the Yale College Undergraduate Regulations and the Yale Graduate School Professional Ethics and Regulations policies regarding cheating, plagiarism, and documentation, with which you should familiarize yourself. Briefly, if you use someone else's work, you must acknowledge it. If it's a piece of code, place the acknowledgment in your source file and explain clearly what parts are not your own. Similarly, if it's in a paper or written assignment, the acknowledgment belongs in the paper itself. All work not so acknowledged must be your own.

You may of course discuss the lectures and readings with your classmates in order to improve your understanding of the subject matter. Helping each other learn to use the tools in the Zoo is also okay. However, the design and implementation of all programs and all submitted work must be your own except where other sources are explicitly noted.

You must never let another student see your work, either before or after the due date of the assignment. Sometimes you may be tempted to "help" your friends by letting them see your solution. Don't! This doesn't help them. To the contrary, it allows them to avoid the hard work of learning the material and deprives them of the educational experience they came to Yale to get.

You are always free (and encouraged) to come in and ask the TAs or instructor for help about anything concerning the course. Please talk to the instructor if you have any questions about this policy.

Avoiding Plagiarism: You may neither copy from another student nor permit your own work to be copied, unless explicit permission is given for such collaborations. If your work is found in the possession of another student, you and the other student are equally guilty of plagiarism. To avoid unintended involvement in plagiarism, *your work should never be in the possession of another student*. Do not ask someone else to deliver or pick up your work. Do not let another student "borrow" your code to compare with theirs. Keep your files protected so that others cannot read them and carefully guard your password. Do not leave printed work in public areas such as the Zoo or in accessible wastebaskets. If you think your password may have been compromised, you must change it immediately and notify the instructor.

Policy on Computer Problems: The Yale College policy on "Use of Computers and Postponement of Work" in the Yale College Programs of Study, Academic Regulations, applies to this course. It is reproduced below.

> "Problems that may arise from the use of computers, software, and printers normally are not considered legitimate reasons for the postponement of work. A student who uses computers is responsible for operating them properly and completing work on

> time. (It is expected that a student will exercise reasonable prudence to safeguard materials, including backing up data in multiple locations and at frequent intervals and making duplicate copies of work files.) Any computer work should be completed well in advance of the deadline in order to avoid last-minute technical problems as well as delays caused by heavy demand on shared computer resources in Yale College."

Particularly relevant for this course are the cautions against leaving a programming assignment to the last minute when machines might be busy, printers broken, and so forth, and about safeguarding your data.

Policy on Technology in the Classroom: Cell phones are not to be used in class. Tablets and laptops are allowed only for course-related activities such as note-taking, reading slides and other materials from the course website, and quick internet searches on topics relevant to the lecture. Their use must limited so as to not distract you from paying attention in class. If in doubt, ask the instructor or TF first. Games, instant-messaging, reading email, and other diversions are not permitted. You may be asked to leave the class if these rules are not followed.

7 Computing Facilities

The Zoo: This course will use the Computer Science Department's educational computing facility, affectionately known as the Zoo. This facility contains modern workstations running Fedora Linux 28. You will need to use these machines to prepare coursework. Look at

```
https://zoo.cs.yale.edu/help/
```

for information on getting started if you are new to the Zoo. A Zoo account will be automatically created for you if you don't already have one when you register as a shopper for this course.

These days, most of you have your own laptops and may be wondering why you should be bothered with using a new computer system. The answer is because code development software is still not completely compatible across multiple platforms. If it works on your Mac or Windows PC but fails when the graders run it on the Zoo, you will lose points. If you ask for help with compiler errors on your personal machine, we might not be in a position to answer your questions. If you lack needed software that has been installed on the Zoo for your use, you're on your own. In short, develop your code on the Zoo! Regardless of where the code is developed, *your assignments will be graded according to how well they work on the Zoo*. Submission of assignments will be through Canvas.

The Zoo machines support remote access via the SSH and VNC protocols. These enable you to do your work remotely when it is inconvenient to go in person to the Zoo. Instructions on how to configure your machine for remote access will be posted to the course web site.

Course directory: The shared course directory, `/c/cs427`, is located on the Zoo server. You can access it from your Zoo course account. It will contain any software needed for this course and miscellaneous documentation and files. Public files there can be also be accessed via the web.