ECE 449/590 – OOP and Machine Learning
Lecture 01 Introduction

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

August 22, 2022

Outline

Reading Assignment

- This lecture: Course Syllabus
- Next lecture: Python Introduction

Outline

Instructor

- Professor Jia Wang
- E-Mail: jwang34@iit.edu
- Office hours: TBD

Lectures

- Mon./Wed. 3:15 PM – 4:30 PM
- Robert A. Pritzker Science Ctr 111
- Course website:
  `http://www.ece.iit.edu/~jwang/ece449-2022f`

Textbooks

- Required Textbook
  - "Accelerated C++: Practical Programming by Example", Koenig and Moo, Addison-Wesley, 2000.
  - "Deep Learning" `http://www.deeplearningbook.org` I. Goodfellow et al., MIT Press, 2016.
- Recommended Textbooks:
  - "A Tour of C++" `https://isocpp.org/tour` B. Stroustrup, Addison-Wesley, 2013.
  - "The C++ Programming Language: 4th Edition", Stroustrup, Addison-Wesley, 2013.
  - "Design Patterns: Elements of Reusable Object-Oriented Software", Gamma et al., Addison-Wesley, 1994.

Useful Websites

- `https://leetcode.com`
  - Practice your programming skills.
- `https://www.youtube.com`
  - Find tutorials.
- `http://stackoverflow.com`
  - Learn how to communicate with professionals.
- `http://www.cplusplus.com/reference`
  - Standard C++ Library reference

Prerequisite

- Computer programming: branch and loop, function, class, searching, sorting.
- Computer organization: memory and pointer, call stack and debugging.
- Linear algebra: vector and matrix algebra.
- Probability: random variables, joint and conditional distributions.
- If you have no recent programming experience:
  - "Programming – Principles and Practice Using C++", Stroustrup, Addison-Wesley, 2014.

Course Outline

Software development for machine learning

- Programming techniques
  - Work with existing code.
  - Python and C++ programming: core language and standard library
  - OOD/OOP: class design, polymorphism, design patterns.
- Software engineering practices: Agile software development
  - Iterative and incremental development (IID)
  - Test-driven development (TDD)
  - Continuous integration (CI)
- Machine and deep learning basics

Course Objectives (ABET)

After completing this course, the student should be able to do the following:

1. Identify objects and their interactions for machine learning applications.
2. Utilize object lifetime for resource management considering object composition, inheritance, and exception handling.
3. Understand typical machine and deep learning algorithms.
4. Reuse existing class libraries to improve code quality and productivity.
5. Utilize class invariants to design class types. Document and validate pre-conditions and post-conditions via assertions.
6. Construct reusable class libraries using polymorphism.
7. Utilize design patterns when designing and reusing class libraries.
8. Implement a machine learning library following test-driven and iterative/incremental software engineering practices.

Homeworks/Projects

- 4 Homeworks
  - Submit online in Blackboard only.
- 6 Projects
  - Project Report: submit online in Blackboard only
  - Source code: via Git only.

Project Workload

- This is a project based course and you are required to design and implement the software system.
  - Projects will be discussed in lectures but we will not provide step-by-step instructions.
  - Expect to practice your problem solving and troubleshooting skills, in particular what you have gained in a programming /design course like CS 115/116/201 and ECE 218/242.
- Do NOT expect to finish a project during the weekend right before its deadline.
- Project 1: Python and course project introduction.
  - Expect to spend hours to understand the basic concepts.
- Project 2: work with the existing code base.
  - Expect to spend 20-40 hours to understand and to modify existing code in addition to learning lecture materials.
- Project 3-6: more functionalities.
  - You should be more comfortable with the projects now. You may still need to spend spend 20+ hours per project.

Late and Resubmission/Regrading Policy

- Software developers should learn how to manage deadlines, especially with all the available tools.
- Late homeworks and projects will NOT be graded, unless
  - A request to extend the deadline is received by email 48 hours BEFORE the deadline.
  - With 48 hours of the deadline or after, the request should be accompanyed by extraordinary reasons with documented proof like docter's notes, or it will be rejected.
- Extraordinary reasons do NOT include
  - Lost of code due to software, hardware, networking failures.
  - Forget account password
  - You should push your code to our Git repository frequently (preferably daily).
- Resubmission/regrading of projects are not allowed.
  - You should utilize our CI system to access your project grading reports and to correct any errors BEFORE the deadlines.

Project Setup

- A computer desktop or laptop that is able to run VirtualBox is required, with the following recommendations.
  - Solid-state drive(s).
  - At least 16GB of memory.
  - At least 4 physical processor cores.
- Course Virtual Machine
  - A full-featured Linux installation with all the necessary tools.
  - Run on your own computer.
- ECE UNIX Network (require Internet and IIT VPN access)
  - Provide a Git-based code repository as the interface to our continuous integration (CI) system.
  - Available remotely: 24×7, mandatory for all students.
  - Please wait for my email regarding your ECE account.
- Read "Guide to System Setup and Work Flow"
  - Follow the links to learn basic Linux and Git usage.
  - Learn how to start your project and use our CI system.

How to ~~survive~~ succeed in this course?

- Read: all instructions are in written.
  - Tutorials, source code, documents, and don't overlook command outputs.
- Communicate: we are very happy to solve any issue you may meet but you need to let us know what's wrong.
  - `https://stackoverflow.com/help/how-to-ask`
- Tools
  - Commit and push your code using Git frequently in case your computer fails.
  - Use the CI system to know your project grade and correct any error before the deadlines.
- Feel free to explore new computer hardware and software but make sure they do not interfere with your schedule to meet deadlines.

Ethics (Very Seriously)

- Read "IIT Code of Academic Honesty" and "IEEE Code of Conduct" (posted on the course website).
  - Projects/homeworks should be done individually.
  - Discussions on homeworks/projects are encouraged.
  - Source code from the lectures and instructions in this course can be used directly.
  - Source code from other places not directly related to this course may be used with proper references.
  - All writings including figures, tables, and screenshots should be BY YOURSELF.
  - NEVER POST YOUR PROJECT CODE OR ASK FOR HELP DIRECTLY ONLINE!
- Please review our Academic Honesty Guidelines.
  `https://web.iit.edu/ugaa/academic-honesty`

ECE 449 Grading

- Percentage
  - Homeworks: 2.5%*4=10%
  - Projects: 20%*6=120% (30% extra)
- Letter grade
  - A: 90
  - B: 80
  - C: 60
  - D: 55

ECE 590 Grading

- Percentage
  - Homeworks: 2.5%*4=10%
  - Projects: 10%+15%*2+20%*3=100% (10% extra)
- Letter grade
  - A: 90
  - B: 80
  - C: 60

Outline

Opportunities

- Our modern society depends increasingly more on computer systems.
  - Internet of Things/Big Data/Machine Learning
- As electrical or computer engineers, we either
  - Develop software by ourselves, or
  - Work closely with software developers.
- A lot of career and business opportunities with a good understanding of how software packages are developed.

Economics of Software Development

- Alternatives: hardware design, in particular FPGA and ASIC.
- Trade-offs
  - Time-to-market and non-recurring engineering (NRE) cost
  - Functionality: more features or less bugs?
  - Performance: optimize for now or future?
- Software is more flexible compared to hardware.
  - For many software systems, performance is not a concern, especially at the very beginning.
  - Software systems can be updated in field, making it possible to improve both functionality and performance later.
  - Usually shorter time-to-market and much less NRE cost.
- Software innovations are less risky than hardware innovations.
  - You can have a demo much sooner to attract clients, collaborators, and investors.

Challenges

- Software systems are complicated.
  - Complex business logic.
  - Different platforms for deployment.
  - Need to allow future improvement.
- It is necessary for a lot of people to collaborate over a long period of time.
  - People are directly involved in the project, or
  - Contribute indirectly via OS and libraries.
  - Very risky!
- It is desirable to have mechanisms that
  - Facilitate reasoning of complex systems, e.g. to decompose them into manageable pieces.
  - Facilitate collaboration between developers and management of development progress to mitigate risks.

Elements of Software Development

- Programming paradigms
  - Define abstraction levels for developers to reason about complex computing systems.
  - Various trade-offs lead to different language designs.
- Software engineering
  - Define methodologies that developers can follow to reduce risk.
  - Effective methodologies vary for different software projects and as workforce changes.
  - Will be discussed in more detail later.

Programming Paradigms I: Imperative Programming

- Imperative: computation as state and state transitions
- Depend on the abstraction level that state and state transitions are defined,
  - Synchronous sequential logic: flip-flops, combinational gates
  - Assembly: registers, opcodes
  - Procedural and structured programming: variables, control structures and functions
  - Object-oriented programming (OOP): objects, member functions
  - Event-driven programming: actors, messages
- A popular choice
  - Intuitive for most people.
  - Very good performance due to the close relationship to hardware implementations.

Programming Paradigms II: Functional Programming

- Functional: computation as functions and their compositions
  - Function (as defined in mathematics): relation between inputs and outputs
- Effective to specify
  - Computations that have well-defined mathematical counterparts, e.g. basic data structure and algorithms.
  - Computations that are easily specified as input/output relations.
- As old as imperative programming and is gaining momentum
  - Performance is no longer a major concern.
  - Correctness has become more important where the relation to math could help.
- Most modern imperative programming languages support certain level of functional programming.

Static vs. Dynamic Typing

- Each variable has a type defining what operations are legal and their meanings.
  - This type system is used to guard against misunderstandings of system components and against careless typos.
- Static typing: the type cannot be changed, e.g. C++/Java
  - Type information is available at compile time – easier for others to reason about the program, and for the compilers to detect type violations no matter how large the program is.
  - However, to decide and to declare the type when writing the code is sometimes cumbersome.
- Dynamic typing: the type can be changed, e.g. Python
  - Allow to write code without knowing much about/being aware of the type system – easier for beginners or for small programs.
  - Type violations can only be detected at execution time – need a lot of tests to guarantee quality, especially for large programs.

Resource/Memory Management

- Resource management, especially memory management, is essential for any program.
  - Also has a great impact on performance.
- Manual management: developers are responsible to allocate and free memory pieces correctly.
  - Typical when performance was the major concern, e.g. C.
  - However, that's impossible for the majority of the programmers as software systems become more and more complicated.
  - Too many bugs related to memory management simply lead to failure of software projects.
- Automatic management
  - Garbage collection (GC): fully automatic, usually for memory management only, trade-off certain amount of performance, almost all languages designed since '90s have this feature.
  - Resource Acquisition Is Initialization (RAII): deterministic, mostly from C++, partially adopted by major languages for exception-safe resource management in general.

Why C++?

- Multi-paradigm
  - Introduce OOP to C (initial objective).
  - Many functional programming features are added later.
  - Able to choose the best paradigm to build each component in a large system.
- Flexible resource management correlated with OOP
  - Low-level memory management compatible with C leads to very efficient memory usage – desirable for systems with limited resources or for applications that need to store large amount of data in memory.
  - RAII-based automatic management can be used to build GC that can greatly simplify memory management – critical to avoid bugs in large systems.
- Static typing: compiler helps to detect errors.
- Share many similarity with other languages.
  - C-like syntax
  - Support of basic data structures and algorithms via containers and iterators

Why Python?

- The rise of big data and machine learning demands programming skills from more professionals.
  - To customize software tools to explore data better.
  - Languages like C++/Java have a steep learning curve, while languages easier to learn like Matlab are proprietary.
- Python as a glue language
  - Easy to learn: dynamic typing, GC
  - Very expressive: multi-paradigm
  - Achieve rich functionality and high performance in small programs by reusing prebuilt external libraries.
- Over the years, the community has built a huge machine and deep learning ecosystem on top of Python.
  - Backed by many libraries written in C/C++/Fortran etc.

Summary

- Software development as a career option.
- C++ and Python.