ECE 2574 Data Structures and Algorithms

Fall 2017
CRN 82739 MWF 2:30-3:20 in WLH 320
`https://filebox.ece.vt.edu/~ECE2574`

Instructor: Chris Wyatt, clwyatt@vt.edu

Today's Schedule:

- Course objectives and content
- Course administration
- Quick Review of ECE 1574

Introductions

Please take the ungraded survey on Canvas (Quizzes - Surveys - Introduction Survey)

- Tell me about yourself
- Help me select my office hours

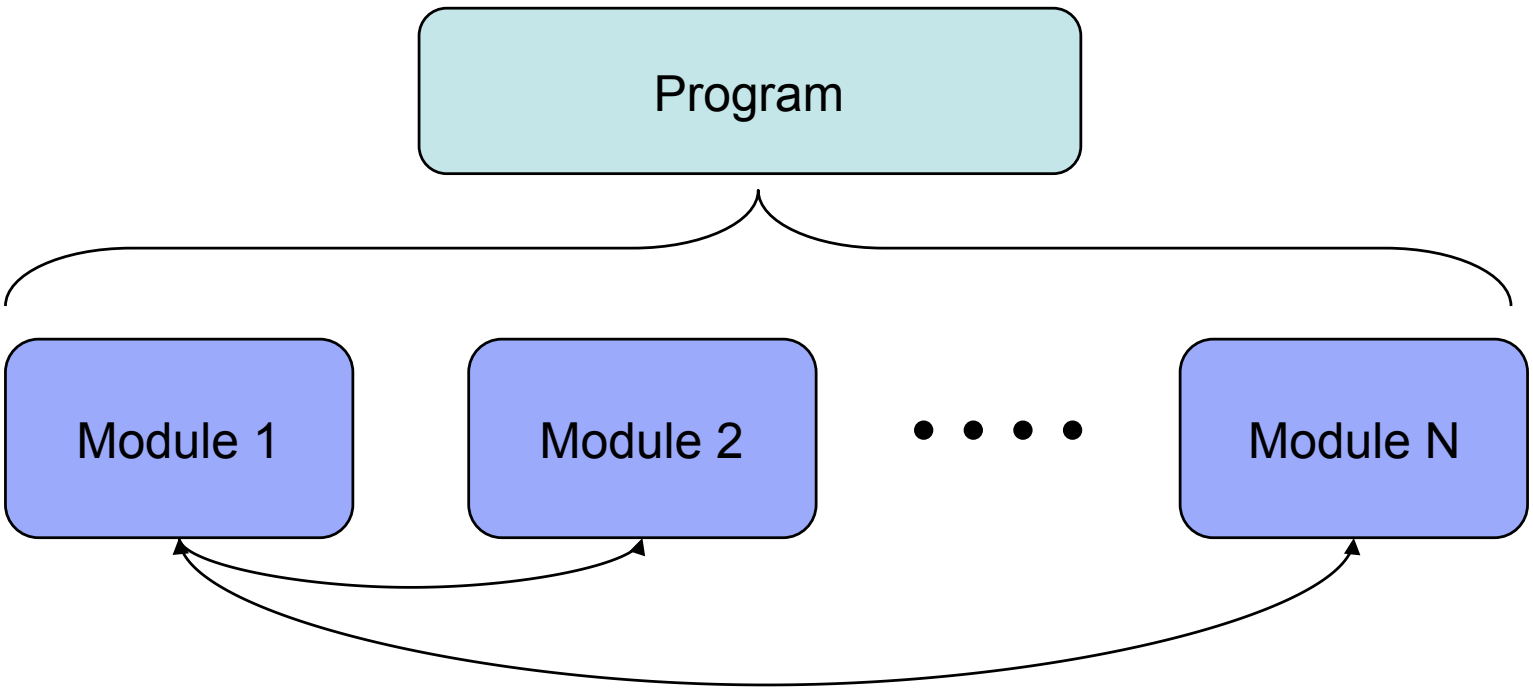
An expression of a computational method in a computer language is a program.

```
// simple C++ program
#include <iostream>
#include <cstdlib>

int main()
{
  std::cout << "Hello World!" << std::endl;
  return EXIT_SUCCESS;
};
```

Programs are made up of modules.

Modules are self contained units of code that solve sub-problems of the larger programming task.



This course is about Abstract Data Types and Algorithms

Abstraction separates the purpose of program modules from the implementation details.

An algorithm is a finite set of rules that give a sequence of operations for solving a specific type of problem.

Abstraction separates the purpose of program modules from the implementation details.

One way to represent a list in C++ (initially empty)

```
typedef itemtype char;

const unsigned int LEN = 256;

itemtype *mylist;

unsigned int mylistlen= 0;
mylist= new itemtype[LEN+1];
*mylist = 0;
```

Abstraction separates the purpose of program modules from the implementation details.

One way to append to that list

```
typedef itemtype char;
const unsigned int LEN = 256;
itemtype *mylist;

unsigned int mylistlen= 0;
mylist= new itemtype[LEN+1];
*mylist = 0;

_____

assert(mylistlen < LEN);

*(mylist + mylistlen) = A;
*(mylist + mylistlen + 1) = 0;
mystringlen++;
```

We can define a list based solely on its behavior.

A *list* is a sequence of (possibly ordered) items with a beginning and end. The operation *append* places an item in the list.

There is no mention of how the list is implemented. This is an **abstract data type** (ADT).

```
List<char> MyList;
MyList.append(A);
```

So what's the Big Deal?

```
typedef itemtype char;
const unsigned int LEN = 256;
itemtype *mylist;

unsigned int mylistlen= 0;
mylist= new itemtype[LEN+1];
*mylist = 0;

assert(mylistlen < LEN);
*(mylist + mylistlen) = A;
*(mylist + mylistlen + 1) = 0;
mystringlen++;
```

```
List<char> MyList;
MyList.append(A);
```

Which one is easier to write, read, debug, and maintain?

Abstraction is needed because of the limitations of the human mind.

Abstraction is needed because of the limitations of the human mind.

- 1 5 7 4 0 2 4 8 5 1

Abstraction is needed because of the limitations of the human mind.

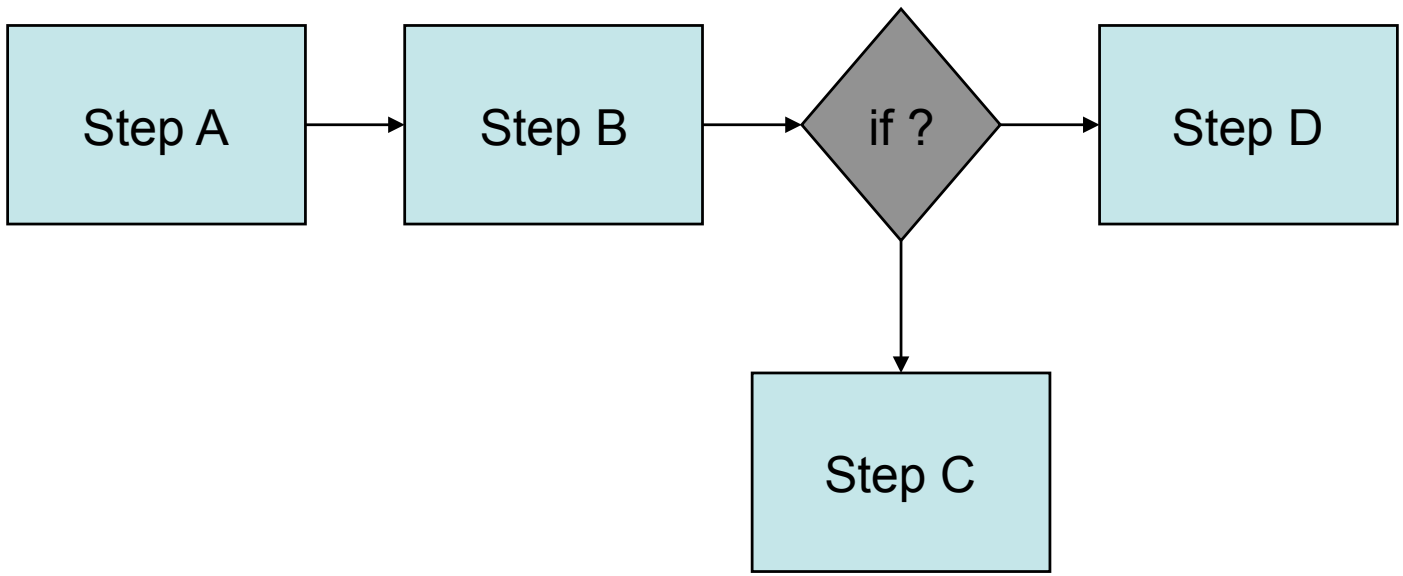
- 1 5 7 4 0 2 4 8 5 1
- 5 4 0 2 3 1 6 6 5 8

Abstraction is needed because of the limitations of the human mind.

- 1 5 7 4 0 2 4 8 5 1
- 5 4 0 2 3 1 6 6 5 8
- 1 2 3 1 2 1 2 3 1 2

An algorithm is a finite set of rules that give a sequence of operations for solving a specific type of problem.

Informally, a recipe, process, method, procedure, or routine



In a more formal sense, an algorithm has five essential features.

1. Finiteness, an algorithm should terminate after a finite number of steps.

A procedure that has all the other characteristics of an algorithm except finiteness is more accurately called a computational method.

In a more formal sense, an algorithm has five essential features.

2. Definiteness, each step of an algorithm must be precisely defined.

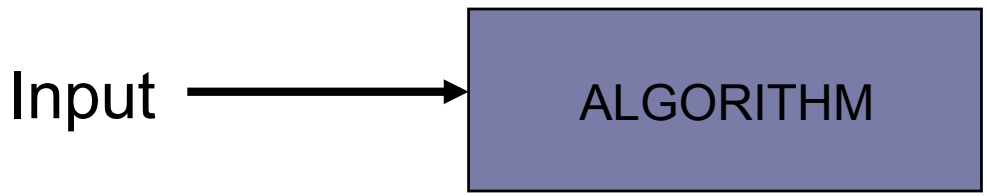
Ambiguous
Compute the average value in a list of integers.

Precise
Compute the unbiased sample mean, m, of a list of n integers $i_1, i_2, \ldots i_n$ as

$$m = \frac{1}{n-1} \sum_{l=1}^{n} i_l$$

In a more formal sense, an algorithm has five essential features.

3. Input, an algorithm has *zero or more* quantities given to it initially or as it runs.



In a more formal sense, an algorithm has five essential features.

4. Output, an algorithm has *one or more* outputs specifically related to the inputs.

In a more formal sense, an algorithm has five essential features.

5. Effectiveness, an algorithms operations should be sufficiently basic to be done exactly in a finite length of time.

The pen and paper test.

An example: Euclids greatest common divisor (GCD) algorithm

Algorithm *Euclid*. Given two positive integers m and n, find their greatest common divisor, that is, the largest positive integer that evenly divides both m and n.
A.0 If $m < n$, swap m and n
A.1 Divide m by n and let r be the remainder.
A.2 If $r = 0$, terminate; n is the answer.
A.3 Set m to n, n to r, and go back to step A.1

- Finite?
- Definite?
- Inputs?
- Outputs?
- Effective?

Course Objectives

- design algorithms for solving problems that use data structures
- implement algorithms in C++ using good programming style.

Course Topics

Abstract Data Types (ADTs):

- Bag
- Lists (std::vector, std::deque, std::list)
- Stacks (std::stack)
- Queues (std::queue, std::priority_queue)
- Dictionaries (std::map, std::unordered_map)
- Trees
- Graphs

Algorithms:

- data structure operations
- sorting and searching

Course Topics

Software Design:

- Interface Design
- ADT implementations
- Testing
- Templates (generics)

Complexity:

- Recursion Relations
- O-notation
- memory and computational complexity
- best, worst, and average complexity
- classification of problems by their complexity: P, NP, and NP-Complete

The only prerequisite is ECE 1574 – Object-Oriented Engineering Problem Solving With C++

You are expected to be competent in the basics of programming with C++.

- Basics of computer organization
- Data types and expressions
- Functions
- Iteration and conditionals
- Design and implementation of complete programs
- Use of arrays
- Use of classes and basic class design
- Formatted I/O and the use of files
- Pointers and run-time memory allocation, basic memory management

The first assignment, program 0, will help you review.

Software

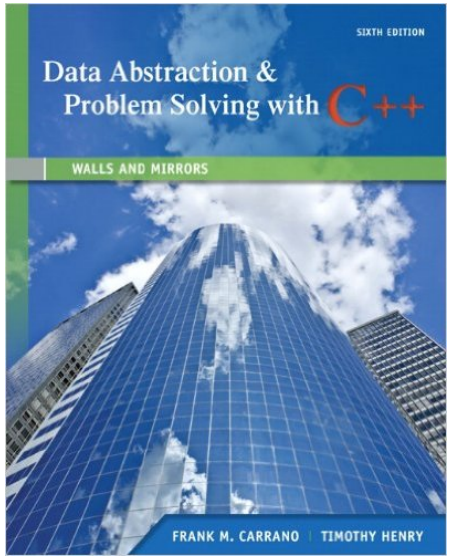
A modern C++ compiler with sufficient C++11 support is required.

- GCC >= 4.8
- Clang >= 3.3
- VC++ >= 18 (Visual Studio 2013 or 2015)

We will use the open source CMake application for managing the build process (www.cmake.org).
You can use any integrated development environment (IDE) supported by CMake, including Visual Studio on Windows and XCode on the Mac.
Further information about the software development environment for this course, including installation instructions, is on the website.
`https://filebox.ece.vt.edu/~ECE2574/devenv.html`

Texts and Resources

(CH) Carrano and Henry, Data Abstraction and Problem Solving with C++: Walls and Mirrors, 6th ed.



Additional References:

- Sedgewick and Wayne, Algorithms
- Martin, Clean Code
- Hunt and Thomas, The Pragmatic Programmer

Additional Resources:

- C++ Reference: `http://en.cppreference.com`

Course Activities

- Readings - Each meeting has a section of the text you are responsible for
- Warmups - due before (usually) every lecture, test basic understanding of assigned reading
- Lectures - Go over the warmup and we write some code
- Exercises - Gain some experience with the material for that day (often in class)
- Projects - core of your learning, implementing and using ADTs

Communication

Course Website: `https://filebox.ece.vt.edu/~ECE2574`

- syllabus, schedule, notes, etc.
- primary way materials are distributed.

Canvas: `https://vt.instructure.com`

- take warmups
- submit assignments
- feedback and grades posted

Piazza: `https://piazza.com/`

- forum/wiki like software for QA, polls, announcements
- replaces email listserv, but has a configurable email digest
- good mobile apps, embedded in Canvas
- use it to ask (and answer) questions

The Auto-Grader

- `https://grader.ece.vt.edu`
- Service to you so that you have a good idea how correct your code is.
- You upload your code as a zip file, the auto-grader runs your tests and my tests, giving hints where your code fails my tests.
- **You still need to submit through Canvas.**

Grading

Warmups:	5% (Extra Credit)
Exercises:	10%
Projects:	60%
Midterm:	10%
Final:	20%

Collaboration and Late Policy

- All assignments must be turned in via Canvas by due date and time
- No late assignments will be accepted, *with the following exception*: you get five free late days (24 hour periods) on projects during the semester to accommodate emergencies. To use one or more of these, just submit the project via Canvas as normal.
- All graded work is expected to be the original work of the individual student unless otherwise directed.

How to Get Help

Some recommendations

- Use Piazza, it is probably the fastest way to get a question answered
- Software Engineering Lab (SWEL): 332 Whittemore
- Don't bang your head against a wall - ask.
- Don't be afraid to come to my office hours/help sessions.

Questions ?

Program 0

https://filebox.ece.vt.edu/~ECE2574/programs/P0/index.html

Due: 9/8 by 11:55 pm via Canvas.

Next Actions

- setup your development environment
- Start P0 - this should be a review of the prerequisite material
- Read CH 1.5 , review Appendix A and B
- Take warmup before noon on Wednesday 8/30.