CSCI 3901: Software Development Concepts

Mike McAllister
Fall 2019

DALHOUSIE UNIVERSITY
Inspiring Minds

CSCI 3901

- Ensure that you are able to implement key elements of
  - Csci 2110 – data structures & algorithms
  - Csci 2132 – software development
  - Csci 2141 – introduction to database systems
  - Csci 3130 – software engineering
- Know how to implement and design from the basics

Learning Outcomes

- Data Structures and Algorithms
  - Use abstract data types (ADTs), including lists, stacks, queues, maps, dictionaries.
  - Implement fundamental data structures, such as linked lists, trees, graphs, and hash tables.
  - Implement traversals, recursive search and state-space exploration algorithms.
  - Implement simple iterative and recursive algorithms to solve moderately simple tasks.
  - Select the appropriate data structure to implement a given ADT under a given set of constraints.
  - Select and use appropriate abstract data types, data structures, and algorithms to solve real-world problems.

Learning Outcomes

- Databases
  - Describe the properties of multiuser database transactions (ACID).
  - Describe the purpose, function, evolution, classification, and building blocks of data models and data modeling.
  - Describe the basic components of a relational model and how relations are implemented.
  - Derive business rules from requirements' specifications and translate these rules into database table and relationship designs.
  - Use SQL data definition and manipulation operations.
  - Construct an entity relationship diagram (ERD).
  - Describe normalization and denormalization, and their role in database design.
  - Perform normalization and denormalization on a database.

Learning Outcomes

- Software Engineering
  - Design a software system and prepare detailed design documentation.
  - Implement moderate-sized programs individually and as a team.
  - Use general data representations like XML
  - Apply standard software processes for build and deployment management.
  - Apply standard software processes for risk management.
  - Apply standard software processes for version control.
  - Apply concepts of software engineering to plan, execute and manage a small software project.
  - Create a Test Plan for a software development project.
  - Effectively debug a program.

Grading

- Class participation – 5%
- Assignments – 40% (7 assignments, equally weighed)
- Midterm – 20%
- Final exam – 35%
- No late assignments accepted
- Use the university's policy on self-declaration of short-term illness
  - Can be used twice in the term for up to 3 days of illness

Important Dates

- October 16
  - Midterm in class – check _now_ for conflicts with other courses
- November 11-15
  - Fall study break
- December 5-15
  - Final exam period
- September 18
  - Last day to add or drop courses without penalty
- October 31
  - Last day to drop courses without academic penalty

DALHOUSIE UNIVERSITY Inspiring Minds

Tentative Schedule

Date	Topic	Lab	Assignment
Sept 2	Administration, problem solving		
Sept 9	ADT, data structures	Problem solving	A1 due
Sept 16	Graphs, interfaces, recursion	Debugging	A2 due
Sept 23	Call stack, exceptions, object-oriented design	Version control	A3 due
Sept 30	Software engineering	Build tools	
Oct 7	Software engineering, database basics	Java – database connection	A4 due
Oct 14	No class Monday, midterm Wednesday		
Oct 21	Software architecture, database insert and query	TBD	A5 due
Oct 28	Database query (continued), join, subqueries	Database queries	
Nov 4	Database CRUD, views, transactions, ACID properties	TBD	A6 due
Nov 11	Study week	No lab	
Nov 18	Database design and normalization	Database design	
Nov 25	Networks	Networks	A7 due
Dec 3	Review	No lab	

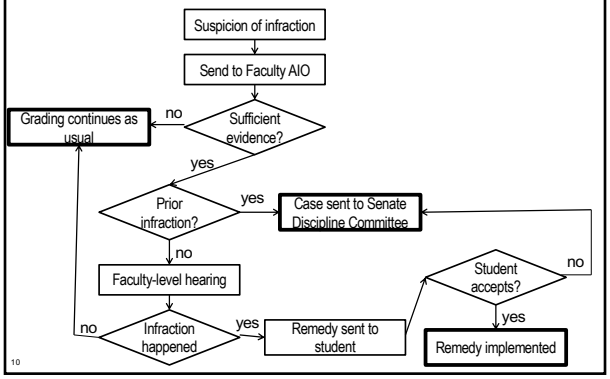
DALHOUSIE UNIVERSITY Inspiring Minds

Homework

- Get Java working on your computer
- Select and install an IDE for Java
- Install a secure shell program and a file transfer program
- Write a "hello world" program using the IDE that can also work on bluenose.cs.dal.ca
  - Compile foo.java on bluenose with "javac foo.java"
  - Run foo.class on bluenose with "java foo"
- Write a program that uses one loop to print the integers from 1 to 10 on individual lines and with a # symbol before the numbers 5 and 6
- Read up on / review linked lists, stacks, queues, and abstract data types

DALHOUSIE UNIVERSITY Inspiring Minds

Academic Integrity Officer (AIO) Process



Concluding Reminders

- Accessibility
- Responsible computing
- Code of Conduct
- Culture of Respect
- Scent-free university
- Mi'kmaq Territory
  - We are all part of the peace and friendship treaties dating back to 1725

DALHOUSIE UNIVERSITY Inspiring Minds

Lessons from last time

- Understand the problem early
  - Ask early about parts that seem unclear
- Test your program before submitting it
- Document every program
  - Internal comments, not just one per method
  - External documentation that doesn't just duplicate the assignment text
- When submitting work
  - Do not upload your entire development directory
  - Ensure that you provide correct permissions on the files when we submit through version control

DALHOUSIE UNIVERSITY Inspiring Minds

Lessons learned from last time

- Best practice coding
  - Do not write huge methods
  - Avoid hard-coding constants
    - Create static final variables to hold the values then use the variable names in the code
  - Use braces around the body of all if and loop statements
  - Aim to have classes in their own files
    - One class per file unless the class is a private support class
- Adhere to the input and output specifications
  - Keep debugging print statements inactive
  - Don't "improve" on the interface unless the assignment gives you that leeway

Postage problem

- Write a program that accepts the destination country (Canada or US) and the weight of a standard envelope and returns the needed postage.

Weight	Canada	US
Up to 30g	$0.85	$1.20
Over 30g and up to 50g	$1.20	$1.80
Up to 100g	$1.80	$2.95
Over 100g and up to 200g	$2.95	$5.15
Over 200g and up to 300g	$4.10	$10.30
Over 300g and up to 400g	$4.70	$10.30
Over 400g and up to 500g	$5.05	$10.30

Postage data from Canada Post at https://www.canadapost.ca/cpo/mc/personal/ratesprices/postalprices.is on Sept. 2015

Problem Solving – Starting the process

- What comes in to the program?
  - Do different data or modes need to be handled differently?
- What transformations do I need to make to the data?
  - Are there sub-problems or patterns that I can use?
- What part of the data is processed right away?
- What part of the data do I need to keep longer?
  - What tasks do I need to do to with that longer-term data?
    - How do I organize or store the data to make those tasks easy?
- What goes out of the program?
  - Do different data or modes need to be handled differently?

Problem Solving – Starting the process

- What assumptions can I make?
  - Are any given?
  - Can I reasonably make any of my own?
- What constraints exist?
- Are there strange cases to handle?
- What is important for the solution to do?

Postage problem

- What comes in to the program?
  - Country and weight from the keyboard
  - The table of postage rates (never changes, so can be part of program itself).
- What transformations do I need to make to the data?
  - None
- What part of the data is processed right away?
  - Answer as soon as we get input
- What part of the data do I need to keep longer?
  - Nothing stored long-term
- What goes out of the program?
  - The postage rate

Problem Solving – Starting the process

- What assumptions can I make?
  - The country and weight are given as integers
- What constraints exist?
  - None
- Are there strange cases to handle?
  - Country or weight outside the table
- What is important for the solution to do?
  - Nothing beyond the given output constraint

Assignment 1

- Start with the sections
  - Goal
    - Tells you why you are doing this work
    - Signals what is important to focus on
  - Background
    - General context to help you understand how or why we might want to write this program
  - Problem
    - Gives you the specifics of what needs to be done. This part is where you get the key information on what to do
    - This part is where you are likely to spend most of your time to understand what to do

Assignment 1

- Start with the sections
  - Inputs
    - Gives you an example of input and/or how that input will be represented.
    - Likely some of the first code you design/write should be using this part
  - Outputs (none on this assignment)
    - As with inputs. Also something that you are likely to address first
  - Assumptions
    - Help you to simplify your design by having some situations that you may not need to cover in your code

Assignment 1

- Start with the sections
  - Constraints
    - Tells you things that you must consider in your design and implementation
    - Review these as and after you design your code and before you start to implement
  - Notes
    - Suggestions on what might make an easier path to a solution
    - You don't need to follow the suggestions
  - Marking Scheme
    - Look over where to prioritize
    - Spend more time or effort on items worth more marks

Assignment 1

- Start with the sections
  - Test Cases
    - Read at the start to help understand any borderline cases to consider that must be handled in your design
    - Run the tests at the end!
    - Will be phasing out this section as the course advances, but you should then write these yourself

Assignment 1

- Look for common patterns to re-do
  - Doing file management in most languages follows the same typical steps
- Look for similar operations that you might solve with one "generic" method
  - "top", "print", and "write" all do basically the same work – can you write one method for all of them and just differ by parameter values?
- See if you can break the big problem into subproblems that are easier to manage
- Look to the Java libraries for helpful structures to use

Transformation example – Grocery list where I pay half

Name	BasePrice	Quantity
Flour	6	2
Oil	10	1
Bottled water	3	3

Name	BasePrice	Quantity	Total	MyShare
Flour	6	2	0	0
Oil	10	1	0	0
Bottled water	3	3	0	0

Name	BasePrice	Quantity	Total	MyShare
Flour	6	2	12	0
Oil	10	1	10	0
Bottled water	3	3	9	0

Name	BasePrice	Quantity	Total	MyShare
Flour	6	2	12	6
Oil	10	1	10	5
Bottled water	3	3	9	5

25

Problem Solving – Assignment 1

- What comes in to the program?
  - To your code: the given method interfaces
  - To the internals: the data you have stored and a tab-separated content from a file
- What transformations do I need to make to the data?
  - Create new columns
  - Copy one column or value to another
  - Store the results of one binary operator into a column
- What part of the data is processed right away?
  - The method call data
- What part of the data do I need to keep longer?
  - The matrix of data
  - Which data structure?
- What goes out of the program?
  - Writing the matix to a file as tab-separated content

26

Problem Solving – Starting the process

- What assumptions can I make?
  - See the assumption section of the assignment
  - No limit on the number of rows, so I should select a data structure that lets me grow the number of rows
  - Limit on input columns but not on newColumn calls, so I probably need to store columns independently so that more can be added
    - Initial start of planning for a maximum number of columns will make the design easier and is likely to incur only a small mark deduction, so I would start with that
- What constraints exist?
  - See the constraint section of the assignment
  - No method throws an exception, so you need to handle them yourself

27

Problem Solving – Starting the process

- Are there strange cases to handle?
  - Watch for bad parameters being passed
  - Watch for column names that don't exist yet
  - Printing when I have no data in the matrix or fewer rows in the matrix than "top" expects
  - What do we do when we have more than 11 columns and want to write to a file?
- What is important for the solution to do?
  - Have the operations act robustly
  - Conform to the given method interface
  - Fit documentation, maintainability, and robustness constraints

28

Postage Problem

- How many different solution styles can you create?

29

Evolution of solving problems

- Often follow a sequence of solutions
  - Can a computer solve the problem at all?
    - There are some problems that computers cannot solve
  - What is _a_ solution?
  - What is an efficient solution?
  - What is a practical solution?
  - What is a simple and practical solution?
  - What is an optimal solution?
  - What is a simple and optimal solution?
- Experience lets you start at different points in the sequence

30

Postage Problem

- The code must know that cases exist
  - Decide whether the cases appear in the code itself or in data structures that the code navigates.
    - Cases in the code: often easier to follow and ensure
    - Cases in data structures: easier to change or expand; more likely to treat the testing of all cases the same way

Encode Cases in the Code

- One independent "if" statement for each case
- Set of "if" statements and exploit previous failed tests using "else" clauses
  - "if" statements could be nested or not

Part data structure, part code

- Encode the boundaries in an array, search for the position in the array for the weight, and encode the answer for that solution into code

Data structures

- Use a data structure (two-dimensional array is enough) to store all of the rates.

Independent "if"

Get the country and weight
If (country is Canada and weight <= 30) report $0.85;
If (country is Canada and 30 < weight <= 50) report $1.20;
If (country is Canada and 50 < weight <= 100) report $1.80;
If (country is Canada and 100 < weight <= 200) report $2.95;
If (country is Canada and 200 < weight <= 300) report $4.10;
If (country is Canada and 300 < weight <= 400) report $4.70;
If (country is Canada and 400 < weight <= 500) report $5.05;
If (country is US and weight <= 30) report $1.20;
If (country is US and 30 < weight <= 50) report $1.80;
…

Does our program work? Test cases

- Independent of implementation (blackbox)
  - Case boundaries
    - Try values on either side of each weight boundary
    - Try with each country
  - Input boundaries
    - Country: each country, invalid country numbers
    - Weight: negative, zero, 1-500, 501 or more
    - Non-integer values when integers are expected
  - Output cases
    - Value < $1, value with one integer digit, value with two integer digits

Does our program work? Test cases

- Dependent on implementation (whitebox)
  - Multiple "if" statements
    - Try each case and ensure appropriate output
  - Linear search in array
    - Search for first, middle, and last entry
    - Search for entry not in the array
  - Binary search in array
    - Search for element requiring
      - Left – left search
      - Left – right search
      - Right – left search
      - Right – right search

Blackbox test cases

Weight	Canada	US
1	0.85	1.20
30	0.85	1.20
31	1.20	1.80
50	1.20	1.80
51	1.80	2.95
100	1.80	2.95
101	2.95	5.15
200	2.95	5.15
201	4.10	10.30
300	4.10	10.30
301	4.70	10.30
400	4.70	10.30
401	5.05	10.30
500	5.05	10.30

Weight	Canada	US
-1	?	?
0	?	?
501	?	?
String	?	?

Country:
0 -- ?
3 -- ?
String -- ?

The problem doesn't specify what to do in the bad cases, so document your assumption and ensure that your code does it.

In general, report an error condition.

"if - else"

```
Get the country and weight
If (country is Canada and weight <= 30) report $0.85;
Else if (country is Canada and weight <= 50) report $1.20;
Else if (country is Canada and weight <= 100) report $1.80;
Else if (country is Canada and weight <= 200) report $2.95;
Else if (country is Canada and weight <= 300) report $4.10;
Else if (country is Canada and weight <= 400) report $4.70;
Else if (country is Canada and weight <= 500) report $5.05;
Else if (country is US and weight <= 30) report $1.20;
Else if (country is US weight <= 50) report $1.80;
…
```


"if - else" nesting

```
Get the country and weight
If (country is Canada) {
        If (weight <= 30) report $0.85;
        Else if (weight <= 50) report $1.20;
        Else if (weight <= 100) report $1.80;
        Else if (weight <= 200) report $2.95;
        Else if (weight <= 300) report $4.10;
        Else if (weight <= 400) report $4.70;
        Else if (weight <= 500) report $5.05;
} else if (country is US) {
        Else if (weight <= 30) report $1.20;
        Else if (weight <= 50) report $1.80;
        …
}
```


"if - else" deeper nesting

```
Get the country and weight
If (country is Canada) {
        If (weight <= 200) {                                    /* Canada and weight <= 200 */
                if (weight <= 50) {                             /* Canada and weight <= 50 */
                        if (weight <= 30) report $0.85          /* Canada and weight <= 30 */
                        else report $1.20                       /* Canada and 30 < weight <= 50 */
                } else {                                        /* Canada and 50 < weight <= 200 */
                        if (weight <= 100) report $1.80         /* Canada and 50 < weight <= 100 */
                        else report $2.95                       /* Canada and 100 < weight <= 200 */
                }
        } else {                                                /* Canada and weight > 200 */
                if (weight <= 400) {                            /* Canada and 200 < weight <= 400 */
                        if (weight <= 300) report $4.10         /* Canada and 200 < weight <= 300 */
                        else report $4.70                       /* Canada and 300 < weight <= 400 */
                } else report $5.05                             /* Canada and 400 < weight */
        }
} else if (country is US) {
        …
}
```


2d-array for rate classes

Get the country and weight

boundaries = array with values 30, 50, 100, 200, 300, 500
Find index i such that boundaries[i-1] < weight <= boundaries[i]

rates = 2d array:
0.85, 1.20, 1.80, 2.95, 4.10, 4.70, 5.05
1.20, 1.80, 2.95, 5.15, 10.30, 10.30, 10.30

Report rates[country][i]

Switch solution

```
Get the country and weight
boundaries = array with values 0, 30, 50, 100, 200, 300, 500
Find index i such that boundaries[i] < weight <= boundaries[i+1]
/* We know that there are at most 7 rates, so combine the country and weight into one
integer: 10's digit is country, unit digit is weight category. */
Class = country * 10 + i
Switch (class) {
        10: report $0.85
        11: report $1.20
        12: report $1.80
        …
        20: report $1.20
        21: report $1.80
        22: report $2.95
        …
}
```

DALHOUSIE UNIVERSITY
Inspiring Minds

43