***Data Structures and Algorithms***

|  |  |
| --- | --- |
| **Course No.:** | **PBM763** (Undergraduate; 2nd year Computer Science) |
| **Program:** | BITL |
| **Instructors:** | Kalvis Apsītis; Jānis Lazovskis |
| **Class Days and Time:** | 5 cr.  **Fall 2021 Semester:** Mon, Wed, Thu 14 weeks: 2021-08-30 to 2021-12-03  Three 90-minute sessions; also about 8 hrs of independent study time per week). |
| **Office Location and Hours:** | Please view the official schedule and epidemiological updates. |
| **Phone and WhatsApp:** | **+371 29112997** |
| **EMail:** | [kalvis.apsitis@rbs.lv](mailto:kalvis.apsitis@rbs.lv); [jlazovskis@gmail.com](mailto:jlazovskis@gmail.com) |
| **Prerequisites:** | Computer Science (L.Selavo, G.Dēķena);  Recurrent sequences, some asymptotic complexity (Big-O Notation), discrete probabilities, trees and graphs. |

**TEXTBOOK**

Drozdek, A. (2013). Data Structures and Algorithms. *Cengage Learning.*

**Website:** <https://bit.ly/3fGkUC6>   
The course assignments and other necessary information is in ORTUS. The website is just a supplement to browse course topics, slides, test samples for programming tasks.

**COURSE OBJECTIVES**

The course discusses algorithm creation considerations, in particular for large input data. We study creation of new algorithms as pseudocode and C++ programs, also some classic data structures (arrays, vectors, lists, stacks, queues, trees, hash tables) and their operations. In practical programming our focus is console applications processing numbers and text – using either STL or hand-crafted data structures with pointers and manual memory management.  
Algorithms are classified by their time and space complexity, the algorithm design paradigms (such as exhaustive search, greedy, divide-and-conquer, dynamic programming).

**COURSE OVERVIEW**

This course discusses programming tasks at several levels:

* Convert informal task descriptions into algorithms operating on data structures.
* Formalize data structures as abstract data types (ADTs) and discuss their implementation tradeoffs.
* Write maintainable C++ code to run algorithms and data structures.

We first introduce some data structures as built-in libraries such as STL or Boost. Later we have tasks relying on self-implemented data structures; in such tasks using any C++ libraries apart from standard textual input/output are not permitted.

The following activities are planned:

**(1) Theory Sessions (Mondays and Wednesdays):**

**Objectives:** Discuss algorithm analysis amd related concepts, introduce data structures and their implementation choices. In some early classes we do walk-throughs – step-by-step guides how to complete some typical programming tasks. **What is graded (Quizzes):** Short-answer quizzes may be offered during classroom sessions in Google Forms. If any questions were first covered on the day of the quiz, there is no penalty for wrong answers, just the participation counts. **Resolving Issues:** Attending class sessions is strongly recommended – and there are no makeup quizzes after the day of the theory session where they appeared. Remediation measures might be needed for those who do not get credit for multiple assignments (quizzes, written assignments, programming tasks)AND miss several classes in a row.

**(2) Lab Sessions (Thursdays):**

**Objectives:** Short problems to test algorithmic knowledge and programming skills may simulate the execution of algorithms, analyze their time complexity, draw data structures, write pseudocode or complete snippets of C++ on paper or in computer. Such problems are also popular in recreational programming and job interviews. We may ask you to do short (5-10 minute) presentations in front of the class – such as analyzing some algorithm – that you have prepared in advance.  
**What is graded (Written Assignments):** During the lab sessions you might need to solve problems on topics covered during that week. Some problems are parametrized using the 3 unique Student ID digits. The assignments usually happen during a class and may take 10-15 minutes. Some coding exercises can also be completed at home and submitted during the following day. They are short (max 20 lines of C++), they usually introduce new concepts and may be useful for your upcoming programming tasks.   
**Resolving Issues:** If you miss any of the written assignments, there are no makeup assignments and you may lose the credit for that particular assignment. Regardless of your status that everyone has prepared before taking the respective midterm (or final).

**(3) Programming Tasks:**

**Objectives:** A programming task is defined as a real-world problem. For a few initial exercises the implementation details might be suggested, but the algorithm and the data structures are your choice. You may need to observe certain guidelines – regarding the use of external libraries or the speed of execution. Each programming task will include sample input and output to avoid any ambiguity.  
Some programming tasks are submitted in several parts – during the first week you implement and test some basic functionality; during the next week – a more advanced one. Finally, you may need to ensure the efficiency for large inputs, avoid memory leaks, or test the code on a certain platform.

**What is graded:** Typically, your grade is proportional the count of the passed testcases. Some initial testcases are public (included in the task description and known to everyone in advance) and private (they comply with the limitations in your programming task description, but not communicated in advance). Your program is not responsible for handling all sorts of wrongly formatted input, except error conditions explicitly mentioned in the task description.

**Resolving Issues:** We recommend you tostart the programming tasks well in advance to leave time to resolve risks and open questions. Code submission is done through GitHub. For some labs you might run the instructors’ grading script on a remote server to see how many testcases you have passed, including the private testcases.

To avoid surprises with wrong code version being graded you should always tag the files you include with your submission (appropriate tagging guidelines are included in the lab).

**(4) Four Exams (three midterms and a final)**

**Objectives:** Conceptual material about the algorithms and data structures and also basic patterns of C++ language are all tested during written exams. Exam can last from about 60 minutes (the midterms) to 120 minutes (the final exam).  
**What is graded:** The scope and the style of written exams are similar to the written assignments (see above). The exam allows asking a few more questions, and their scope is typically broader.  
**Resolving Issues:** Make-up exams for the midterm and the final are generally not given. If there are extenuating circumstances and you must miss an exam, the instructor must be notified ahead of the exam time. If a student does not notify the instructor in advance or misses the make-up exam deadline, the exam receives no credit.

**COURSE REQUIREMENTS**

To get a passing grade you are expected to submit substantial amount of C++ code to pass at least 50% of the testcases (all programming tasks counted together). The maximum grade for programming tasks is 500 points, and the expected minimum is at least 250 points. Also your overall score should be at least 350 points to get a passing final grade.

**GRADING**

Grading for the course is as follows:

Final grades are calculated as follows:

> 949 = 10  
850-949 = 9  
750-849 = 8  
650-749 = 7  
550-649 = 6  
450-549 = 5  
350-449 = 4  
< 350 = failing

Please check to see that your intermediate points in ORTUS are recorded correctly.

|  |  |
| --- | --- |
| ***Assignments*** | **Points** |
| (1) In-class quizzes (15-30 small quizzes) | 50 |
| (2) Written Assignments (about 15) | 150 |
| (3) Programming Tasks (about 10) | 500 |
| (4) Exams (three midterms and a final) | 300 |

**ACADEMIC INTEGRITY**

To strengthen ethics within Riga Business School and the business community the RBS policy is to take steps to avoid cases of academic fraud. Be aware that any student who turns in computer code that is not their own will be subject to the RBS sanctions policy on Academic Fraud (see the policy on the online assistant). The same consequences apply to academic dishonesty on in-class assignments and exams.

To find information on what plagiarism is and how to avoid it please visit the links at:  
<http://www.uottawa.ca/plagiarism.pdf>  
<http://www.socialsciences.uottawa.ca/pdf/plagiarism2.pdf>  
<http://www.sass.uottawa.ca/writing/plagiarism.pdf>

This short presentation on research and plagiarism will also help learn to correctly reference sources and provides good advice on research:  
<http://library.acadiau.ca/tutorials/plagiarism/>

**How does it apply to our class?**

During the Data Structures and Algorithms class, you should be aware of the following integrity guidelines in particular:

* **In-class quizzes and written assignments:** You are on your honor to complete them independently, using your notes, Internet and the computational devices, but no assistance from other humans. Failure to do so may result in poor performance during the midterm and final exams.
* **Programming Tasks:** Group study and discussions are encouraged, but you should never exchange finalized solutions or obtain them anywhere on the Web. Copying any snippets of C++ from the Internet can be done only if all the conditions hold:   
  (A) the snippet is very short (1-3 lines)  
  (B) you immediately analyze what you have copied and make sense of it  
  (C) you know the names of C++ concepts involved (for example, Google-search “how to iterate over an STL Queue” and then copy the result)  
  There is a vast difference between a competent programmer who consults Internet to find samples; and someone who does not see how the copied pieces fit together.
* **Exams:** Exams will be proctored and are closed book and closed notes; you can use only paper and pen. Calculators are not allowed. (The exams could be open book and open notes and using your computer with C++ IDE and other software tools, only if the class does it remotely for epidemiological reasons or similar.)

**CLASS SCHEDULE**

The schedule of topics discussed every week is listed below. Some reordering might happen during the semester. Regarding the Pre-Class Preparation see ORTUS or the Webpage.

In this schedule “*Programming”* means programming tasks becoming due, “*Exam”* means exam on that week, “*Walk-throughs”* are routines to complete some typical development tasks, “*Objectives”* are general question types that typically appear in class assignments or exams (see Webpage for a full list of objectives).

| Fall 2021 | Topic | Pre-class preparation | Deliverables and Activities (may change) |
| --- | --- | --- | --- |
| Week 1 2021-08-30 to  2021-09-03 | Console programs with “cin”, “cout”. Expressions, control structures and function calls.  Variable declarations, definitions and scope. Uninitialized variables.  Solve “single-pass” tasks (like finding a maximum).  Store input in arrays to revisit.  **Objective:** Given a flowchart, write its control structure.  **Objective:** Given an initial state of an array, predict its contents after a conditional/loop. |  | Console application patterns: write text/variables to STDOUT, read a fixed/variable number of items from STDIN.  **Walk-Through #1:** “Programming in VS Code”.  Set up an IDE on Windows (VS Code and Microsoft compiler). Run executables from command line; redirect input and output streams to files. Display return value of main().  Also show how to submit a coding exercise. |
| Week 2 2021-09-06 to  2021-09-10 | Perform searches, filters, mappings and similar data manipulations using just loops and arrays.  **Objective:** Given int or char variables, perform bit operations (shifts, etc.), and simple (bitwise or short-circuit) Boolean logic.  **Objective:** Initialize multi-dimensional arrays as arrays of pointers. |  | Two-dimensional arrays, processing input from a textfile, read input line-by-line and then read all items on every line.  **Walk-Through #2:**  “Running C++ on Virtual Linux”. Set up a VirtualBox guest Xubuntu. “git” command-line. Run makefiles. Executables from command line. Copying to/from Linux with WinSCP. “diff” utility with options for whitespace. Windows and Linux line-endings. |
| Week 3 2021-09-13 to  2021-09-17 | Implement multiple classes in several CPP and header files.  Parameter passing by value or by reference, implicit returns from a function and other side-effects.  “const” modifier in functions. Pass arrays or large objects as parameters.  Write recursive and mutually-recursive functions. Declaration vs. definition.  Abstraction and separation of concerns (1 class = 1 concern). Introduce encapsulation as an OO concept.  Methods of std::string. Manipulate char and char\*; compare chars to identify digits, letters, spaces, etc.  **Objective:** Given ASCII table, figure out which comparisons are needed to sort out certain characters.  **Objective:** Find if a string has some interesting snippet and cut it. |  | Use collections from the STL library. Implement recursive (and mutually recursive) functions.  **Walk-through #3:**  “Test-driven development with C++ objects”.  Starting files from a given UML diagram, using method stubs. Catch2 for unit testing framework. Add testcases. Unit-test functions with expected exception conditions. Add testsuite “setup” and “teardown” routines. Create Catch2 test reports. |
| Week 4 2021-09-20 to  2021-09-24 | Constructors and destructors revisited. Order how object is initialized; implicit constructor and methods.  C++ inheritance, virtual methods.  C++ exceptions.  Define operations in Abstract Data Types (ADTs); use “Stack” as the first sample ADT. |  | Code throwing/catching exceptions. Functions with same names - overriding vs. overloading functions.  **Walk-through #4:**  “Building libraries and CMake”. Build and run C++ code with CMake on Windows and Linux. Build DLLs (and also static libs) and use them in other projects. |
| Week 5 2021-09-27 to  2021-10-01 | Introduce polymorphism as an OO concept along with some design patterns.  C++ template functions and template classes.  Operator overloading.  Time complexity of algorithms; Big-O revisited. |  | **Midterm #1:** Basic C++ features, OO concepts.  **Coding:** Array implementation of a complete binary tree.  **Walk-through #5:** “External libraries and Google Test”Use 3rd party libraries such as Google Test. Include headers and use DLLs (or static LIBs). |
| Week 6 2021-10-04 to  2021-10-08 | Use STL (standard template library) for queues, vectors, maps.  Impose sorting order or sort by passing a pointer to a comparator function. |  | **Walk-through #6:** “Measuring efficiency and memory leaks”.Add time measurement to the executable code in C++ environment. Run Clang compiler and get LLVM reports for the C++ code. Configure Valgrind report on memory leaks. |
| Week 7 2021-10-11 to  2021-10-15 | Do pointer-based constructs in C++. Create class definitions that recursively point to themselves.  Construct and manipulate list-like data structures.  **Objective:** Given an initial state of a stack or a queue, perform some sequence of actions.  **Objective:** Draw and manipulate priority queues and heaps. |  | **Walk-through #7:**  “WebAssembly – efficient C++ in browsers”. Compile code with Clang to use in Webassembly. Run it together with some JavaScript to draw images. |
| Week 8 2021-10-18 to  2021-10-22 | Construct and traverse tree-like data structures.  Construct and manipulate Binary Search Trees (BSTs). |  | **Midterm #2:** Polymorphism, templates, STL datastructures. Also list-like ADTs (stacks, queues, vectors). |
| Week 9 2021-10-25 to  2021-10-29 | Analyze and compare sorting algorithms.  **Objective:** Given an (arbitrary) binary tree, draw an array representation.  **Objective:** Given heap implementation as an array, perform heap operations.  **Objective:** Perform 1-2 steps of QuickSort (or other efficient algorithm). |  | **Coding:** Pointer exercise on balancing and tree rotation.  **In-class:** |
| Week 10 2021-11-01 to  2021-11-05 | Use hashing functions (including your own) in custom structures and STL.  Construct and manipulate set-like structures (also maps, dictionaries, hashtables).  **Objective:** Given an initial state, perform some operations on AVL tree.  **Objective:** Given an initial state, perform some operations on a red-black tree.  **Objective:** Given a hash function, predict, if it creates an efficient hash table. |  |  |
| Week 11 2021-11-08 to  2021-11-12 | Describe graph-related structures and their traversals.  Analyze Shortest Paths and MST algorithms.  Use weighted graphs or some other augmented structure. |  | Midterm #3: Trees, balanced trees, sorting, set and dictionary ADTs. |
| Week 12 2021-11-15 to  2021-11-19 | Graph algorithm applications. DFS applications.  **Objective:** Do a topological sort of a directed acyclic graph.  **Objective:** Identify strongly connected components in a directed graph.  Celebrate November 18 and 19. |  |  |
| Week 13 2021-11-22 to  2021-09-26 | Describe algorithms by type (numeric, operating on collections, graphs, etc.)  Illustrate exhaustive/brute-force paradigm; describe descrease-and-conquer paradigm. Review algorithms for those paradigms.  **Objective:** Run Dijkstra’s shortest path algorithm on a weighted directed graph.  **Objective:** Run Bellman-Ford shortest path algorithm on a weighted directed graph.  **Objective:** Run Prim’s MST algortihm on a weighted undirected graph. |  |  |
| Week 14 2021-11-29 to  2021-12-03 | Illustrate divide-and-conquer paradigm, dynamic programming paradigm, greedy paradigm. Review algorithms for those paradigms.  All kinds of randomness in algorithms: Probabilistic algorithms, algorithms allowing a probability of error, measuring amortized complexity for an ADT implementation, algorithm behavior for “likely” vs. “worst-case” operation/input sequences. Hashing and pseudorandom numbers.  **Objective:** Given a hash table, find the expected number of collisions.  **Objective:** Given a Bloom filter (a fixed collection of “independent” hash functions), estimate the probability of a false positive.  **Objective:** Given a Miller-Rabin primality test, interpret the results for a given number of rounds and witnesses. Estimate the rounds needed for given certainty.  **Objective:** Given probabilities of insert/lookup/delete operations, find the amortized time complexity for a data structure such as hash table. |  |  |
| Consultation Week  2021-12-06 to  2021-12-10 | Reviewing all the topics, solving sample questions. Analyzing some elements of labs submitted previously. | N/A | Reviewing your programming problems; list of preparation topics – what question types will be covered during the final exam. |
| Exam Session 2021-12-13 to  2021-12-17 | Final Exam (TBA) | N/A | Final Exam: Various topics; in particular graph algorithms and general algorithm paradigms. |