***Data Structures and Algorithms***

|  |  |
| --- | --- |
| **Course No.:** | **PBM763** (Undergraduate; 2nd year) |
| **Program:** | BITL |
| **Instructors:** | Kalvis Apsītis; Jānis Lazovskis |
| **Class Days and Time:** | 5 cr.  **Fall 2022 Semester:** TBD, TBD, TBD 14 weeks: 2022-MM-DD to 2022-MM-DD  Three 90-minute sessions; also 4-8 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) , [janis.lazovskis@rbs.lv](mailto:janis.lazovskis@rbs.lv) |
| **Prerequisites:** | Computer Science (L.Selavo, G.Dēķena);  Recurrent sequences, some asymptotic complexity, Big-O Notation, discrete probabilities, trees and graphs. |

**TEXTBOOK**

Open Question

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

**COURSE OBJECTIVES**

The course discusses algorithm creation methods that are efficient for large input data. New algorithms are written as pseudocode and C++ programs; classic data structures (arrays, vectors, lists, stacks, queues, trees, hash tables) are needed as well. In programming our focus is console applications that manipulate text symbols – they build on top of the Standard Template Library (STL) or hand-crafted data structures with pointers and manual memory management.  
Algorithms are classified by their time and space complexity, their topic and their design paradigm (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 using certain data structures.
* Formalize data structures as abstract data types (ADTs) and discuss implementation tradeoffs of basic operations in ADTs.
* Write maintainable C++ code to run algorithms and data structures.

In several tasks only standard textual input/output libraries are permitted to get first-hand experience implementing your own data structures.

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. Attending class sessions is strongly recommended. Remediation measures might be needed for those who miss several classes in a row and have not submitted some major tasks.

**Written Assignments** can be offered during theory classes. They take 10-15 minutes. Typically, these tasks cover topics from the current or the previous week – there may be different variants of tasks for different people; some problems can be parametrized using the 3 unique Student ID digits. You are expected to prepare for the written assignments before the class and solve similar samples at home (there are handouts showing how to solve sample problems).  
You can expect written assignments approximately twice per week, but they could be skipped, e.g. if some midterm exam or major lab task is submitted at that time. If you miss any of the written assignments, there are no makeup assignments and you may lose the credit for that particular assignment.

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

**Objectives:** Short problems simulate the execution of algorithms on paper, analyze their time complexity, draw data structures, write pseudocode.

**Designing and presenting algorithms** can happen on lab sessions (in some cases they can be scheduled individually – during office hours). One of your 7 programming tasks is an individual task – and before coding it you would need to analyze the algorithms and data structures that you would use to solve it. Your presentation should contain object oriented design (UML class diagrams, design patterns, if applicable), it should also contain pseudocode for major functionality and the analysis of its correctness and time complexity. The presentations are scheduled and prepared in advance. **Written Assignments** can also be offered during lab sessions (see description above).

**(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 choice of algorithm and the data structures is up to you. You may need to observe certain guidelines – some external libraries may be disallowed; limits on RAM or speed may be introduced. Programming tasks will include sample input/output to avoid ambiguities.  
Some programming tasks are submitted in two 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 testcases are public (included in the task description and known to everyone in advance) and some are private (they comply with the programming task description, but are not communicated in advance). Your program is not responsible for handling all sorts of wrongly formatted input, unless the error conditions are explicitly mentioned in the task description.

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 tag the files you include with your submission (tagging guidelines are included in the labs).

**(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). 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.

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 400 points, and the expected minimum is at least 200 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) Written Assignments (about 15) | 150 |
| (2) Programming Tasks (about 7) | 400 |
| (3) Exams (4 – three midterms and a final) | 400 |
| (4) Designing and presenting algorithms (1) | 50 |

**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.

**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:

* **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:** Written parts of exams will be proctored and are closed book and closed notes; you can use only paper and pen. Calculators are not allowed.   
  C++ programming parts allow the use of computers with C++ IDE and the Internet.

**CLASS SCHEDULE**

The schedule of topics for every week. In the schedule we mark programming tasks becoming due and the time for exams. *In-class assignments* refer to demos and written assignments offered during the lab sessions. *Walk-throughs* show how to complete typical development routines, *objectives* – skills used in assignments or exams. Some schedule changes during the semester are possible.

| Fall 2021 | Topic | Readings, preparation | Deliverables and Activities |
| --- | --- | --- | --- |
| Week 1 2022-MM-DD to  2022-MM-DD | C++ constructs apart from object orientation: running simple programs, variables and expressions, control statements, arrays and structs.  *Walk-through 1:* Programming in VS Code; single-file programs.  *Objectives:* Compute expressions and understand their side-effects; implement conditionals/loops appropriate for various tasks; understand the layout of arrays and structs in the memory. | Class slides. Pages 121-231 from a free book: Eckel, B. *Thinking in C++, 2nd edition.* Pearson Higher Education. | *In-class assignments:* Simpleconsole applications in C++ using textual I/O patterns: <https://www.hackerrank.com/> and similar. |
| Week 2 2021-09-06 to  2021-09-10 | Classes, inheritance, virtual functions and polymorphism. Templates, operator overloading. STL library and vector data-structure.  *Walk-through 2:* Set up a virtual Linux guest, test C++ code, use command-line git.  *Objectives:* Create classes to encapsulate real-world entities, operate with pointers and references, use template functions, template classes and STL vectors. Manipulate strings and char\*. | Drozdek2013, Chapter 1.1, ..., 1.9 (pp.1–35): *Object-Oriented Programming Using C++*. | *Programming Task 1.1:* C-style single-file programs to manipulate integer and string data. Test and submit on Linux.  *In-class assignments:* Define classes and class hierarchies, define inherited functions and declare them virtual. Other basic OO design patterns (singleton, polym |
| Week 3 2021-09-13 to  2021-09-17 | Find a Big-Theta notation for functions, analyze time complexity of algorithms, show how to improve the running time.  *Walk-through 3:* Implement an UML class and unit-test with Catch2. (Can also test the time.)  *Objectives:* Write recurrences for algorithm running time, solve them and find their Big-O notation. | Drozdek2013, Chapter 2.1, ..., 2.9 (pp.51–64): *Complexity Analysis*. | *Programming Task 1.2:* Use classes. Peaks, element with given rank. Catch2 unit-tests and Test-Driven Development.  *In-class assignments:* Analyze functions as Big-Theta, Big-O, Big-Omega notation. Optimize some algorithms (finding permutation with certain properties, peak finding) in terms of complexity. |
| Week 4 2021-09-20 to  2021-09-24 | Pointer-based structs in C++. Class attributes as pointers to the same class. Define linked lists, doubly linked lists and circular lists. Use List class in STL.  *Walk-through 4:* Use cmake as a cross-platform build tool; build self-defined shared libraries (\*.dll or \*.so files) and use them in another C++ project.  *Objectives:* Use pointers with user-defined structures, write node traversals, inserts, deletes as pointer operations. | Drozdek2013, Chapter 3.1, ... 3.3 (pp.75–96): *Linked Lists*.  Drozdek2013, Chapter 3.7 (pp.109–113): *Lists in STL*. | *In-class assignments:* Problems that manipulate “Nodes” (containing integers or other predefined types) connected with pointers. Deleting explicitly reserved memory.  *Midterm 1:* Basic C++ features, OO concepts, time complexity of algorithms, problems using pointers. |
| Week 5 2021-09-27 to  2021-10-01 | Singly linked, doubly linked lists, pointer operations needing them.  *Walk-through 5:* Including external libraries, use Google Test in C++ projects.  *Objectives:* Implement stacks, queues or circular queues; use them to evaluate expressions, new/delete on pointers. | Drozdek2013, Chapter 4.1, ..., 4.7 (pp.131–158): *Stacks and Queues*. | *Programming task 2.1:* Array implementation of a complete binary tree.  *In-class assignments:* Stack and queue implementations using class templates. Running stack and queue operations on paper. |
| Week 6 2021-10-04 to  2021-10-08 | Tree concepts, traversals, representing trees with node structures or arrays.  *Walk-through 6:* Use Valgrind to find memory leaks; also run Clang compiler to monitor certain operations.  *Objectives:* Draw an array representation of a binary tree. Draw and manipulate priority queues and heaps. | Drozdek2013, Chapter 6.1, ..., 6.6 (pp.214–249): *Binary Trees.*  Drozdek2013, Chapter 6.9 (pp.268–276): *Heaps*. | *Programming task 2.2:* Using STL datatypes (or your own) implementation for some queueing problem.  *In-class assignments:* Insertion and deletion of nodes in a binary search tree.Preorder, inorder, postorder traversals. Combining trees and stacks. Priority queues with heaps. |
| Week 7 2021-10-11 to  2021-10-15 | Construct and manipulate balanced Binary Search Trees (BSTs).  *Walk-through 7:* Compile C++ to use in wasm (WebAssembly). Run it with JavaScript to draw images in a browser screen.  *Objectives:* Compute operation time estimates in balanced (AVL or red-black trees). | Drozdek2013, Chapter 6.7, ..., 6.8 (pp.250–268): *Balancing a Tree*.  Chapter 7.1.8 (pp.337–353): *2-4 Trees.* | *Programming Task 3.1:* Construct a full binary tree from an unordered list of parent-child relations.  *In-class assignments:* Node manipulations in AVL or red-black trees. |
| Week 8 2021-10-18 to  2021-10-22 | Describe graph-related structures and their traversals. Using BFS and DFS in other algorithms.  *Objectives:* Use graph representations as Create topological sort, identify (strongly) connected components. | Drozdek2013, Chapter 8.1, 8.2 (pp.391–398): *Graphs*.  Drozdek2013, Chapter 8.6, 8.7 (pp.415–422): *Connectivity, Topological Sort*. | *Midterm 2:* ADTs for stacks, queues, vectors; implementing with pointers, or arrays, or STL. Trees, balanced trees, ordered maps and dictionaries.  *In-class assignments:* Graph traversals (DFS, BFS). Use them for topological sort, etc. |
| Week 9 2021-10-25 to  2021-10-29 | Algorithms on augmented (weighted etc.) graphs. Cut and cycle properties in Minimum Spanning Trees (MSTs).  *Objectives:* Analyze shortest path algorithms (Dijkstra, Bellman-Ford), run MST algorithms (Prim, Kruskal). | Drozdek2013, Chapter 8.3, ..., 8.5 (pp.398–415): *Shortest Paths, Spanning Trees*. | *Programming Task 3.2:* Implement node insertion and deletion, visualize a sequence of tree modifications.  *In-class assignments:* Properties of cuts and cycles w.r.t. Mimum Spanning Trees. Visualize shortest paths and MST algorithms, run them on paper. |
| Week 10 2021-11-01 to  2021-11-05 | Estimates for sorting complexity. Shell sort, Heap sort, Quicksort, Mergesort, Radix sort, Counting sort.  *Objectives:* Visualize sorting algorithms (quicksort, etc.). Sorting algorithms in STL and sorted maps. | Drozdek2013, Chapter 9.1, ..., 9.4 (pp.491–532): *Sorting*. | *Programming Task 4.1:* Implement some variant of shortest paths algorithm in a weighted graph.  *In-class assignments:* Minimizing comparisons (coin weighting), tradeoffs between sorting algorithms. |
| Week 11 2021-11-08 to  2021-11-12 | Hashtables, hash collision resolution.  *Objectives:* Understand implications of selecting a hash function, configure hashes in STL classes. | Drozdek2013, Chapter 10.1, ..., 10.3 (pp.548–562): *Hash functions, Collisions, Deletion.* | *Midterm 3:* Graph traversals, MST and Shortest Path algorithms, sorting.  *In-class assignments:* Numerical algorithms, large numbers, GCDs, inverses, primality tests. |
| Week 12 2021-11-15 to  2021-11-19 | Show how hashing can implement Map and Dictionary ADTs.  *Objectives:* Analyze event probabilities (collisions, rehashing costs) in hashtables. Select optimal parameters for classical hashtables and Bloom filters.  *Celebrate:* November 18. | Drozdek2013, Chapter 10.4, ..., 10.6 (pp.562–576): *Perfect Hashing Functions, Rehashing*. | *Programming Task 4.2:* Apply the shortest paths implemented earlier to a real-world data.  *In-class assignments:* Compare ordered and unordered maps. Unusual effects with particular hashing functions. Secure hashes (SHA-256 etc.). |
| Week 13 2021-11-22 to  2021-09-26 | Introduce string concepts and string-search algorithms.  *Objectives:* Analyze “naive” and efficient string algorithms (Knuth-Morris-Pratt, Boyer-Moore); use hashing for string search (Rabin-Karp) and plagiarism detection. | Drozdek2013, Chapter 13.1.1, ..., 13.1.4 (pp.674–696): *Exact String Matching, KMP, BM Algorithms.* | *Programming Task 5.1:* Implement the Rabin-Karp algorithm with rolling hash.  *In-class assignments:* Build data structures for KMP or BM algorithms, run them on paper. Experiment with rolling-hashes. |
| Week 14 2021-11-29 to  2021-12-03 | Regular expression searching; using suffix trees to index texts and find keywords. Introduce Ukkonen’s algorithm.  *Objectives:* Learn string indexing. Revisit algorithms with probability assumptions (Quicksort; non-balanced search trees), amortized complexity (hashtables, arraylists), probabilistic behavior (Rabin-Miller) or probabilistic data structures (Bloom filters). | Drozdek2013, Chapter 13.1.6, ... 13.1.9 (pp.700–718): *Matching Sets of Words*. | *Programming Task 5.2:* Use hashing as in a Bloom filter, count words from some glossary.  *In-class assignments:* Implementing automata, using suffix trees, consider time-complexity models – deterministic or probabilistic. |
| Consultation Week  2021-12-06 to  2021-12-10 | Review earlier topics, solve sample questions. Analyze, discuss and fix programming tasks submitted previously. | N/A | Review previous tasks, do refactoring, optimization and eliminate memory leaks. No much coding, but reviewing the tasks from the perspective of style and maintainability.  *Final Review Topics:* Revisit question types that will be covered during the final exam. |
| Exam Session 2021-12-13 to  2021-12-17 | Final Exam (TBA) | N/A | *Final Exam:* Covers various topics from the course, in particular, graphs and string search. |