***Data Structures and Algorithms***

|  |  |
| --- | --- |
| **COURSE No.:** | **CSE 250 (?)** |
| **PROGRAM:** | BITL |
| **INSTRUCTOR:** | Kalvis Apsītis |
| **CLASS DAYS & TIME:** | 5 cr.  **Fall 2020 Semester:** Mon, Wed, Thu 2020-09-07 to 2020-12-18  Three 90-minute sessions; about 8 hrs of independent study time per week). |
| **OFFICE LOCATION & HOURS:** | TBA |
| **CONTACT PHONE:** | **+371 29112997** |
| **E-MAIL:** | kalvis.apsitis@rbs.lv |
| **PREREQUISITES:** | Ievads Datorzinībās – I, Ievads Datorzinībās – II. (This refers to the Python and Scala programming course by Leo and Gundega in Year 1.) |

**TEXTBOOK**

**The Textbook:** TBA (To be announced).

**Class Website:** TBA (hosting may change – the permanent URL will be announced to the students).   
It contains presentations and descriptions of assignments.   
**Reading Assignments:** Class Website may contain task analysis (all the skills that are in the scope of this course).

**COURSE OBJECTIVES**

The course covers design, implementation, and properties of data structures and related algorithms. They are also described in C++, a high-level programming language.  
Theory topics include time-space analysis and tradeoffs in algorithm design, some patterns of algorithm design and object-oriented development. Basic data structures – arrays, vectors, lists, stacks, queues, and heaps are included. Tree and graph algorithms and traversals, hashing, sorting, and data structures on secondary storage. The course shows the importance of choosing appropriate data structures for real-world programming projects.

**COURSE OVERVIEW**

This course uses the interplay between various levels of abstraction:

* Describe how certain families of real-world algorithmic tasks (such as sorting, searching, string processing, graph-related, combinatorial, geometric or numerical problems) translate into algorithms using certain data structures. For example, one can translate a programming task from a human language into a pseudocode of some algorithm in order to reason about its correctness and efficiency.
* Define data structures as abstract data types with a limited number of public interfaces; use object orientation to hide the implementation details from the caller of that data structure. Once the interfaces are known, map the abstract data types into lower-level constructs. For example, dictionary (as an abstract data type) can be implemented as a hash table or as a binary search tree and also in some other ways.
* Use constructs from C++ to develop a collection of software artefacts that can implement all the above, using the computer hardware efficiently. For this reason, we need to learn some C++ and to use object-orientation for the data types.

In order to acquire the skills on all the interrelated topics, the following activities are planned:

|  |  |
| --- | --- |
| **Description of the activity** | **How is it affected by the epidemiological situation?** |
| **(1) Classroom or Remote Sessions:**  **Objectives:** Listen and tell stories about data-structures and algorithms. Clarify concepts and misunderstandings. Help with the other activities, get the instant feedback from the students regarding their understanding.  **Challenges:** Instructor should limit the urge to spend the whole class time talking; topics should be well aligned to the current assignments. Students could also become engaged and use the class time to make their independent coding experience more effective and meaningful.  **Techniques:** Attending class sessions is mandatory (should we enforce mandatory office hours, if somebody skips too many classes?) Three 90-minute sessions per week are planned. The topics to be covered are known at least 1 week in advance; students are encouraged to read ahead using their textbook or other reference material. Small assignments should take no more than 10-15 minutes.  **Evaluations:** To encourage active participation, informal polls, multiple-choice or short answer quizzes or short written assignments would be offered in most classroom sessions. Some can be graded by computer; some need human grader. | Blended classrooms (where some participants are in the classroom, but some are self-isolating) might be necessary. The instructor can repeat the questions asked by the classroom (no need for “Catchbox” equipment that might be unsafe). But instructor’s voice when speaking though a face-mask is muffled and not very clear. Could microphones and voice amplification help?  Adjustments for the remote participants:   * **Computer graded tests:** Use polling and short answer tests (ORTUS or Socrative). Same experience for remote students. * **Paper tests:** Typical in-class writing assignment (on paper) could take about 7-10 minutes. * **Both tests:** Use parametrized exercises (numbers from their student ID or randomly generated)   More things to consider:   * Encourage the participants to ask questions, to discuss, to type their input in chat? * Splitting the class in teams? |
| **(2) Coding exercises in C++:**  **Objectives:** New programming language (in particular, one like C++) should have regular practice; one cannot start directly with large programming projects.  **Challenges:** There are large differences in programming experience and consequently – unpredictable drain on students’ free time. The class is (mostly) not about learning C++.  **Techniques:** About once every week we start a short coding exercise in C++ in a class session. It can involve small programs from the ground up, refactoring an existing code or error correction or learning how to use a specific language feature – of the sort “Add your code here”. These coding exercises are assumed to be short (up to 30 lines of code).  **Evaluations:** Some I/O behavior checks plus pattern matching for the source code (to detect expected or forbidden language features) should allow automated or semi-automated grading. | We expect that 2nd year BITL students can work on small coding exercises on their own. The necessary support depends on the student.  Adjustments for the remote participants:   * Help with the configuration issues is going to be hard; cannot look over anyone’s shoulder. Problems should be resolved over email. Create a mailing group where people can post their problems and can get answers from the instructor and from each other. * Introduce step-by-step guides how to set up the development environment and how to submit the exercises – to avoid common pitfalls.   More things to consider:   * Walk-throughs for key technical skills (using a debugger? using a Makefile configuration?) – where every participant follows some procedure and we ensure that everyone can perform some activity? |
| **(3) Programming Problems:**  **Objectives:** In programming problems the participants can go through all the steps of the algorithmic problem solving – starting with a problem that describes a real-world situation and ending with software that solves it correctly and efficiently.  **Challenges:** Larger programming exercises may create stressful situations just before the deadline.  **Techniques:** Every problem starts as a reading comprehension exercise – checking the ability to elicit information that is needed for the algorithmic model. Samples of input and output data should ensure that there is no ambiguity regarding the expected result.  For some programming problems two-fold test cases and also two-fold submission dates may be introduced. The first one is meant to achieve the basic functionality; the second one – to ensure the algorithm that has robust behavior for large input data (and uses algorithms with reasonably good asymptotic complexity).   * 1 STL problem: Use the built-in STL library and its data structures. * 3 problems from the scratch: Use only the I/O libraries and implement all the needed data structures by yourselves. * 1 team problem: Convert a user story into an algorithmic problem; create a precise specification and solve it using all previous techniques.   **Evaluation:** Public test cases (available before the turn-in date) and private test cases are used for grading. Before submitting one can verify, if the code compiles and runs the public tests. | Adjustments for the remote participants:   * Schedule the office hours in the RBS building (outside the normal class hours in small teams) * Plan some virtual office hours; arrange weekdays and hours.   More things to consider:   * Many programming assignments so far involved mutual consultations between the students themselves. Not clear how this can happen in remote settings. * Some higher-level problems involving data structures could be covered on a “conceptual level” without much coding at all: For example the participants analyze a user story to make it into an algorithmically solvable problem and select the appropriate data structures. |
| **(4) Midterm and Final exams:**  **Objectives:** The conceptual material about the algorithms and data structures (operations of abstract data types, their implementations, amortized time complexity, time-space tradeoffs, hash functions and many other things) can be tested in written exams. Some of this is covered by the in-class assignments. | Adjustments for the remote participants:   * Participants can write the exam remotely (with Webcam switched on) * If exams are offered remotely, they are open book and open notes.   More things to consider:   * A short oral part of exam for remote participants – to discuss their thinking process; gain a more detailed feedback of what they have learned. |

**COURSE REQUIREMENTS**

In order to get a satisfactory grade, there are some mandatory activities that you need to complete.

1. **Class Sessions (about 40) and In-Class Assignments (at least once per week):** We expect that you attend all scheduled classes (missing several classes in a row because of health reasons might imply some ). Attendance is 50 points and in-class assignments during the class (polls, short-answer tests or written tests) is also 50 points.
2. **Coding Exercises (at least once per week):** They cover one particular aspect of C++ language. They are typically started in a class, and finished by the students a few days later. They should be submitted by a deadline and just their conformance to the formal (both functional and non-functional) requirements is evaluated.
3. **Programming Problems (5)** are longer programming assignments that involve analysis and implementation of a problem (sometimes using built-in data-structure libraries such as STL, sometimes doing the data structures “from the scratch”). They have to be submitted by a deadline and private test cases (i.e. tests not known to the student) may be used during the grading.   
   Some problems are submitted in two parts – the first part aims at getting the correct functionality for certain typical inputs; the second part can
4. **Midterm and Final Exams (1 and 1).** Books and notes are not allowed during the exams (unless they are done remotely). 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. The only time a make-up exam can be taken is during the week following the date of the exam. If a student does not notify the instructor of an absence or misses the make-up exam deadline, the exam will not be included in the final grade.

**GRADING**

Grading for the course is as follows:

Final grades are calculated on the following basis.

> 949 = 10

850-949 = 9

750-849 = 8

650-749 = 7

550-649 = 6

450-549 = 5

350-449 = 4

< 350 = failing

Your grades will be updated regularly on the online assistant. Please check to see that your grades are recorded correctly.

|  |  |
| --- | --- |
| ***Assignments*** | **Points** |
| Class sessions and in-class assignments | 100 |
| Coding exercises | 100 |
| Programming Problems | 500 |
| Written Exams (Midterm and Final) | 300 |
| TOTAL | 1000 |

**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/>

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

* **In-class Assignments:** They may be either supervised or unsupervised (remote students). You are on your honor to complete them independently, using your notes, Internet and the computational devices, but no human assistance. Failure to do so may result in poor performance during the midterm and final exams.
* **Coding exercises and Programming problems:** You should be capable to explain the code you have written. You are encouraged to discuss your assignments and code snippets on the whiteboard, but you should never exchange your solutions in any form or copy the answers from the Internet (even when you can find them). Your code should be created and typed by you.
* **Exams:** Exams will be proctored and are closed book and closed notes; you can use only paper and pen. It will be open book and open notes, if it is done remotely. During the exam you cannot communicate with others regarding your solutions.

An excerpt from the Syllabus of Buffalo CSE 250:

* *No tolerance on plagiarism:*
  + *0 on the particular assignment/exam for first attempt*
  + *Fail the course on the second*
* *Group study/discussion is encouraged, but the submission must be your own work*
* *On the Programming Assignments: discussions of ideas are welcome, but NO exchanges of source codes, please.*
* ***I will take cheating VERY VERY seriously***

<https://cse.buffalo.edu/~hungngo/classes/2013/Fall/250/syllabus.html>

**CLASS SCHEDULE**

A specific schedule of topics is listed below. Topics and submission dates may change to synchronize with the course load in other subjects. The activities listed here are just examples or suggestions. They may change later as we adjust to the speed of the class.

| Fall 2020 | Topic | Pre-class preparation | Deliverables and Activities |
| --- | --- | --- | --- |
| Week1 07.09 - 11.09 | * Hello World programs in C++, Makefiles, Input/Output for CLI programs, fundamental types, submitting and grading your work. Data in machine’s memory. Two’s complement and float types. Reading and writing bytes and chars. * Algorithms: characteristics, topics. ADTs and data structures. | Textbook chunks and/or YouTube videos – selections for every week. | **Coding:** Hello World.  **Coding:** Expressions and bit operations.  **In-class:** Stack as an array. |
| Week2 14.09 - 18.09 | * Control structures, declarations and definitions, values and references, enums and structures, pointers and arrays. C++ classes and header files. Function calls by value and by reference. Compilation and linking. Function pointers. * Analysis of algorithm feasibility and efficiency. What is revealed by code inspection, testing, debugging. How to find mistakes. |  | **Coding:** Implementing recursive (and mutually recursive) functions.  **In-class:** Flowcharts vs. Control structures.  **In-class:** Doubly-ended queue as an array. |
| Week 3 21.09 - 25.09 | * Object orientation (abstraction, encapsulation, inheritance), namespaces, source organization and naming conventions, C++ templates, operator overloading, STL to implement vectors, stacks, maps. * Algorithm paradigms, overview. Simple sorting and searching in arrays. |  | **Coding:** Playing with the STL library  **In-class:** Finding Big-O notation and/or solving sums and recurrences (time analysis). |
| Week 4 28.09 - 02.10 | * Variable scopes, C++ memory model, stack and heap. Polymorphism calls and sorting. * Brute force paradigm. * Review of lists, stacks, queues and deques as ADTs * Lists and doubly linked lists as pointer structures. |  | **Coding:** Calls of some library sorting methods (passing the custom comparison function)  **In-class:** Drawing list-like data structures as pointer diagrams.  **Problem 1:** An ICPC-style problem with STL. |
| Week 5 05.10 - 09.10 | * Method calls with inheritance. Abstract classes and virtual functions. * Divide and Conquer paradigm * Binary Search Trees and other Trees as pointer-based data structures. * Trees as ADTs * Some tree traversals * Fast matrix multiplication |  | **Coding:** Array implementation of a complete binary tree.  **In-class:** Drawing an array representation of an (arbitrary) binary tree. |
| Week 6 12.10 - 16.10 | * Decrease and conquer paradigm * Priority queues as ADTs. * Priority queue as a heap. * Skew binomial heap * Leftist binomial heap * Efficient sorting – 1 (HeapSort) |  | **Problem 2:** Another ICPC-style problem using lists etc.  **In-class:** Dooing some heap operations on paper. |
| Week 7 19.10 - 23.10 | * Greedy paradigm * Optimization tasks as algorithms * Efficient sorting – 2 (MergeSort, QuickSort) |  | **In-class:** Doing 1-2 steps of some efficient sorting on paper. |
| Week 8 26.10 - 30.10 | * Time and space trade-off * More on space complexity * Horner’s scheme for polynomials |  | **Problem 3:** Use search trees, maps, dictionaries |
| Week 9 02.11 – 06.11 | * Tree balancing * AVL trees * Red-black trees * Sets and iterators in C++ * Implementing rooted N-ary tree as a binary tree. |  | **Coding:** Pointer exercise on balancing and tree rotation.  **In-class:** Drawing an operation on AVL tree. |
| Week 10 09.11 - 13.11 | * Hash functions and hash tables. * Hash collisions * Hash tables and chaining * Dictionaries as ADTs (comparison between hash and tree implementation). * Bloom filters; probabilistic dictionary implementations |  | **Problem 4:** Use weighted graphs or some other augmented structure. |
| Week 11 16.11 - 20.11 | * Transform and conquer paradigm * Linear equations * Some linear optimization |  | **Coding:** An easy numeric method (iterative linear equations or similar) |
| Week 12 23.11 - 27.11 | * Memoization * Dynamic programming. * Graph problems * Combinatorial problems |  | **Problem 5:** Converting user story into an algorithmic problem. |
| Week 13 30.11 - 04.12 | * NP-complete problems and their approximation. * Reductions to SAT or 3-SAT problem to show NP-completeness |  | **Problem 5:** Back-end implementation |
| Week 14 07.12 - 11.12 | * Trees on secondary storage, B-trees. * Augmented data structures. Some computational geometry algorithms. |  | **Problem 5:** Integrating the back-end with some application. |
| Consultation Week 14.12 - 18.12 | Reviewing all the topics | Reviewing your programming problems and other assignments; also the list of preparation topics to be released before the final exam. | None |
| Exam Session 21.12-23.12 | Final Exam (TBA) | N/A | Final Exam |