Course Title: Construction Programming

Course Code: 202-1-5941

Credit Hours: 4.0

Instructor: Dr. Han Attinger

Prerequisites:
- 201-1-0201 Introduction to Logic and Set Theory
- 202-1-0203 Computer Programming
- Fundamentals of Programming Languages

The significant challenge that developers of software face is the need to find ways to formulate algorithms without errors and to develop clean, bug-free programs. This goal is not simple to achieve, and achieving it is particularly important in systems whose malfunction could endanger human life, cause environmental damage, or lead to business failure.

When we focus on the functional correctness of the program (such as for every legal input, the program produces the desired output), we can rely on software testing (testing) or on a process of verification. However, testing does not guarantee the absence of errors, due to the practical difficulty (and sometimes the inability) to check correctness for every possible input.

In contrast, verifying the program after writing, by proving its correctness at the end, may be effective to find bugs, as intended. But this is a complex process that requires understanding the way programmers think and following the assumptions that were taken during development and are not documented. A third way to convince of functional correctness is to state the correctness proof while building the program.

This course teaches students how to plan and build algorithms and programs whose correctness is guaranteed. The course describes a well-defined method for developing code in a step-by-step manner, directly from the specification (specification). The specification itself includes conditions for legal input (preconditions) and desired results for the output (postconditions). Algorithm refinement techniques are presented in the course, using them to state loop invariants and termination conditions. Consequently, by developing the code systematically from the specification, the correctness of loops and recursive functions is guaranteed in a consistent manner.

A student who has completed this course will be able to use the software development methodology to implement algorithms and programs whose correctness is guaranteed.
A major challenge software developers are facing is how to design correct algorithms and develop computer programs without bugs. This is certainly not a trivial goal, and finding ways to achieving it is highly important, especially in the development of life-critical, safety-critical, mission-critical, or security critical systems.

When focusing on a program’s functional correctness (such that for any valid input the program generates the expected output), one could resort to common practices such as testing or verification. However, successful testing does not guarantee the absence of errors, whereas verification is difficult to perform. A third option is to construct the proof of correctness alongside the development of code.

This is an advanced course teaching how to design algorithms and programs that are guaranteed to meet their specification. Starting with a mathematical description of the program’s requirements, the course presents a formal method for turning such specifications into actual code, in a stepwise approach known as refinement. Techniques of algorithm refinement are presented, for the derivation of loops and recursive procedures from invariants.

The developed algorithms are typically very short, but challenging, as we aim to construct correct and efficient code. The taught material is mainly based on the textbook "Programming from Specifications" by Carroll Morgan. The programming throughout this course is done using a development environment that enables the annotation of programs with their specifications. The environment includes an automatic verifier, such that the functional correctness of a program (with respect to its specification) can be established ahead of the generation of an executable file.

At the end of the course students are expected to be able to construct correct programs. More concretely, you will be able to:

- Specify program requirements abstractly.
- Perform rigorous and formal derivations of efficient programs from their abstract specifications.
- Understand the criteria for algorithm refinement.
The following topics will be covered, along with a range of examples and case studies:

- Program specification using predicates and assertions: predicate notation, preconditions and postconditions, specification statements.
- A language based on guarded commands (featuring an assignment statement, sequential composition and conditional statements, blocks, local variables, and arrays) with proof rules for each program construct.
- Basic techniques for finding invariants.
- Constructed types: from sets, bags, and lists to functions and relations.
- Procedures, parameter passing, and reuse.
- Recursive procedures: rigorous derivation of sorting and search algorithms.
- Recursive types: linked lists and binary trees.
Students taking this course will be required to have good programming skills, yet they will not be expected to possess the logical-reasoning skills needed for specifying the algorithms, as those will be acquired throughout the course.

The final grade will be determined by two homework assignments (30%) and a must-pass final examination (70%). The assignments are to be prepared in (small) teams.

The main textbook of this course is:

Carroll Morgan. *Programming from Specifications (2nd edition)*

Additional textbooks:

Anne Kaldewaij. *Programming: The Derivation of Algorithms*

Roland Backhouse. *Program Construction: Calculating Implementations from Specifications*

David Gries. *The Science of Programming*

Edsger W. Dijkstra. *A Discipline of Programming*