**Course Title**: Construction Programming - by Correct

**Course Code**: 202-1-5941

**Type of Course**: Elective

**Credit**: 4.0

**Instructor**: Dr. Ron Atinnger

**Prerequisites**: Automata 202-1-2011, Systems Programming 202-1-2031, Principles of Programming Languages 202-1-2051

**Course Outline**

One of the significant challenges faced by software developers is the need to find ways to write algorithms without errors and build programs that are free of bugs. The goal is to achieve reliable, effective, and efficient implementations.

While testing and verification (testing, verification) are not guaranteed to prevent bugs, especially in complex systems, they are necessary for ensuring the correctness of the software. In contrast, proving the correctness of the program after writing it, especially by verifying its correctness, can ensure the absence of bugs, as desired. However, this is a complex process that requires understanding the thought processes of the developers and tracking the assumptions made during development and are not documented.

A third way to convince oneself of the functional correctness of the program is to write the correctness at the time of building the program.

This elective course teaches techniques for designing and building 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 feature (specification), through refinement (preconditions) and verification (testing).

Techniques for algorithm refinement (algorithm refinement) are presented in the course, which involves writing loop invariants and stopping conditions, for example, which are derived from the preconditions and postconditions that make up the feature.

In addition, advanced techniques for data refinement (data refinement) are taught in the course, allowing improvements to be made to the generated code without compromising its correctness.

The algorithms developed in the course are very short, although the development process is not trivial. The course content is based mainly on Carole Morgan's book "Programming from Specifications". The course is based on the Dafny programming language and takes place in the Visual Studio environment of Microsoft. The environment also includes automated verification of the functional correctness of the programs once they have been written.
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, or mission-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. Advanced techniques of data refinement are explored too, targeting the efficient implementation of abstract data types (along with the code for manipulating them).

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 in the language Dafny, using its integration into Microsoft Visual Studio. This environment enables the annotation of programs with their specification. Moreover, it includes an automatic verifier, such that a program can be executed only after its functional correctness has been established.

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 and data 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.
- The language of guarded commands (the assignment statement, sequential composition and conditional statements, blocks, local variables, and arrays) with proof rules for each program construct and the corresponding syntax in the programming language Dafny (see http://research.microsoft.com/en-us/projects/dafny/, http://rise4fun.com/Dafny).
- Basic techniques for finding invariants.
- Constructed types: from sets, bags, and lists to functions and relations.
- Procedures and parameter passing.
- Recursive procedures: rigorous derivation of sorting and search algorithms.
- Modules, encapsulation, and the refinement of modules.
- State transformation and data refinement.
- Recursive types: linked lists and binary trees; pointer algorithms.

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 (10% each), a must-pass midterm examination (20%), and a final project (60%). The assignments and project are to be prepared in pairs or individually.

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
Edsger W. Dijkstra. A Discipline of Programming