POZNAN UNIVERSITY OF TECHNOLOGY

EUROPEAN CREDIT TRANSFER AND ACCUMULATION SYSTEM (ECTS)

COURSE DESCRIPTION CARD - SYLLABUS

Course name
Declarative programming [S1Inf1>PDEK]

Course

Field of study Computing	Year/Semester 1/2
Area of study (specialization) –	Profile of study general academic
Level of study first-cycle	Course offered in Polish
Form of study full-time	Requirements compulsory

Number of hours

Lecture 16	Laboratory classes 16	Other (e.g. online) 0
Tutorials 0	Projects/seminars 0	

Number of credit points

2,00

Coordinators	Lecturers
dr inż. Artur Michalski artur.michalski@put.poznan.pl	

Prerequisites

The student starting this course should have basic knowledge of computational logic and set theory. He should have the ability to solve basic algorithmic problems described by means of a formal mathematical apparatus and the ability to obtain information on how to solve them from the indicated sources.

Course objective

Provide students with basic knowledge about the declarative programming paradigm on the example of the Prolog language. Developing students" skills in solving algorithmic problems using a high-level programming language based on recursion mechanisms and searching solution space. Shaping students" awareness of the differences between the declarative paradigm and other programming paradigms.

Course-related learning outcomes

Knowledge:
1. The Student has a structured and theoretically founded general knowledge of programming languages and paradigms as well as procedural and declarative interpretation of the program code.
2. The student has a basic knowledge of the functioning of automatic inference mechanisms in declarative languages based on computational logic.

3. The student knows the basic methods, techniques and tools used in solving computer tasks using recursive data structures and their implementation in declarative programming languages.
4. The student has elementary knowledge of methods of metaprogramming, processing symbolic representation of data and knowledge.

Skills:
1. The student can assess the usefulness of languages, methods and tools for solving tasks typical for computer science, and indicate the appropriate areas of application of the methods and programming tools of the declarative paradigm.
2. The student has the ability to formulate algorithms and implement it in the field of symbolic processing and textual tasks.
3. The student can develop and implement a solution to a programming problem in terms of the declarative paradigm using simple and complex (recursive) data structures.

Social competences:
1. The student understands the need to constantly expand their knowledge and improve skills in the field of programming tools and developing programming paradigms.
2. The student is able to think and act creatively, remaining open to non-IT aspects of the activity of an IT engineer related to software development.

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning outcomes presented above are verified as follows:
Verification of assumed learning objectives related to lectures: evaluation of knowledge by test in a form of 20 questions worth of 20 points in total. Positive grade is obtained by acquiring more than 10 points.
Verification of assumed learning objectives related to laboratory classes by a test at the end of the semester, consisting in solving a specific programming problem; the task being the subject of the test is a multi-stage algorithm, and obtaining a positive grade is possible after correctly implementing more than half of all its steps that make up the complete final solution and demonstrating the ability to construct one's own (genuine) components of the solution; some of the steps of the correct solution include the methods presented during the semester, however, the student must first of all demonstrate originality in solving the task in order to obtain the final credit; this criterion is considered the key in the evaluation of the final work.

Programme content

Declarative and procedural interpretation of the Prolog program
Recursive definition of the rule
Data representation in Prolog
Term unification
Relations between Prolog and formal logic
Representation of lists in the Prolog
Arithmetic operations in the Prolog
Processing flow control in Prolog
Negation by failure
Meta-predicates in Prolog
Efficiency of Prolog programs

Course topics

Lecture 1: Introduction to the Prolog language
- An example of a simple program in Prolog: Family relationships
- Extending the program by introducing rules
- Recursive definition of the rule
- Generating answers to the Prolog questions
- Declarative and procedural interpretation of the Prolog program
Lecture 2: Syntax structure and interpretation of Prolog programs
- Data representation in Prolog

- Term unification
- Formal declarative interpretation of the Prolog program
- Formal procedural interpretation of the Prolog program
- Program interpretation example: The monkey and banana problem
- Order of Prolog clauses and goals
- Relations between Prolog and formal logic

Lecture 3: Lists, operators and arithmetic operations
- Representation of lists in the Prolog
- Selected operations on lists in the Prolog
- Operator notation
- Arithmetic operations in the Prolog

Lecture 4: Processing flow control in Prolog
- Cuts in Prolog
- Examples of using cuts in a Prolog program
- Negation by failure
- Problems with cuts and negation in the Prolog

Lecture 5: Predefined Prolog predicates - meta-predicates
- Checking the type of term
- Term composition and decomposition: = .., functor, arg, name
- Different types of equality operations in the Prolog
- Database manipulation in Prolog
- Manipulating the control flow in the Prolog
- Predicates: bagof, setof and findall

Lecture 6: Input / output operations in Prolog
- File operations in Prolog
- Processing files of terms
- Manipulating character data
- Composition and decomposition of atoms
- Loading Prolog programs: consult and reconsult

Lecture 7: Style and programming techniques in Prolog
- General rules of correct programming in Prolog
- How to interpret Prolog programs?
- Programming style in Prolog
- Efficiency of Prolog programs

The program of the laboratory classes corresponds to the program of the lecture with the exception of the input/output operations on files.

Teaching methods

1. Lecture: multimedia presentation, including examples of problem solutions
2. Laboratory classes: implementation of practical programming tasks of increasing difficulty

Bibliography

Basic
1. Prolog. Programowanie, W.F. Clocksin, C.S. Mellish, Helion, Gliwice, 2003
2. Logika w rozwiązywaniu zadań, R.A. Kowalski, WNT, Warszawa, 1989

Additional
1. Prolog - programming for AI, I. Bratko, Addison-Wesley, 1990
2. Micro-Prolog, K.L. Clark, F.G. McGabe, WNT, Warszawa, 1985
3. Prolog, F. Kluźniak, S. Szpakowicz, WNT, Warszawa, 1983

Breakdown of average student's workload

	Hours	ECTS
Total workload	60	2,00
Classes requiring direct contact with the teacher	32	1,00
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	28	1,00