COURSE DESCRIPTION CARD - SYLLABUS

Course name
Computer architecture with low-level programming [S1SI1E>ASK]

Course

Field of study
Artificial Intelligence

Year/Semester
1/2

Area of study (specialization)
–

Profile of study
general academic

Level of study
first-cycle

Course offered in
English

Form of study
full-time

Requirements
compulsory

Number of hours

Lecture
15

Laboratory classes
15

Other
0

Tutorials
0

Projects/seminars
0

Number of credit points

3,00

Coordinators

dr hab. inż. Tomasz Żok prof. PP
tomasz.zok@put.poznan.pl

Lecturers

Prerequisites

The student should have the ability to obtain information from indicated sources and show willingness to work in a team.

Course objective

To provide knowledge about low-level aspects of programming in C. Developing students" awareness of challenges and potential difficulties while designing low-level applications. Familiarize students with the x86 CPU architecture and extensions.

Course-related learning outcomes

Knowledge:
1. has a well structured knowledge of programming in C.
2. is familiar with the most common errors and problems associated with designing low-level applications.
3. is familiar with x86 computer system architecture and extensions.

Skills:

1. is able to design and implement a C-language application that solves problems such as processing text or binary data.
2. is able to use a debugger and memory leakage analysis programs to solve the most common problems related to low-level application development.
3. is able to create an x86 assembly application including optimizing the use of SIMD operations.

Social competences:
1. is aware that knowledge of computer system architecture and the ability to create low-level applications translates into a fuller understanding of any type of IT solutions.
2. understands that low-level solutions are crucial from the point of view of IT systems security.

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 the knowledge acquired in the course of the lecture is done by means of a written assessment containing open or multiple choice questions. The pass threshold is 50%.
Skills acquired during the laboratory classes are verified by evaluating several projects or practical tasks. Students can work in pairs. Each project is evaluated separately. In order to pass the laboratory classes it is required to receive at least a 3.0 grade from each project.

Programme content

Lecture:
1. C, Basics, Types, Literals, Expressions, Statements
2. Functions, Arrays, Pointers
3. Structures, Unions and Bit-Fields
4. Dynamic Memory Management, Input and Output
5. Multithreading, Floating-point Numbers
6. Computer Architecture
7. x86 Assembly
8. Test

Laboratory classes:
1. Practical guide to C programming and debugging
2-3. Project 1: Text File Parsing
4-5. Project 2: Binary File Parsing
6-7. Project 3: x86 Assembly
8. Late Project Reporting

Course topics

The lectures provide a comprehensive introduction to computer science fundamentals. Topics include the history and evolution of computing, the significance of abstraction in computer design, and Moore's Law and its future implications. The lectures also explore computer architecture, covering memory hierarchies, pipelining, prediction, and system dependability. An introduction to the C programming language addresses data types, variables, and control structures.

In addition, the lectures delve into computer programming essentials, focusing on instruction sets, assembly language, operations, and high-level languages like C and Java. They explain data structures, data transfer instructions, and the stored-program concept.

Specific to C programming, the lectures cover literals, expressions, control structures, and statements. Topics include integer, floating-point, character, and string literals, operators, precedence, type conversions, if-else statements, switch cases, and various loops. They also discuss assignment, increment and decrement operators, and jump statements such as break, continue, and goto, alongside function calls and returns.

Further lectures explain function definitions and calls, array initialization and access, and pointer usage,

including memory manipulation and pointers to functions. The `restrict` keyword's role in pointer operations is also covered.

Arithmetic operations in computing, including integer addition and subtraction, multiplication and division, and floating-point representation, are explained. The lectures address computer number limitations like overflow and underflow and introduce C structures, unions, and bit-fields along with their applications.

Advanced C programming topics include algorithms, date and time functions, memory management, I/O operations, and error handling. The use of standard library functions for sorting (`qsort`) and searching (`bsearch`), date and time manipulation, and memory allocation (`malloc`, `calloc`, `free`, `realloc`) are detailed. I/O operations using `printf`, `scanf`, `fread`, `fwrite`, and random file access methods (e.g., `ftell`, `fgetpos`, `fsetpos`, `fseek`, `rewind`) are also covered, with examples and code snippets provided.

Lastly, the lectures discuss the memory hierarchy and virtualization. They explain the economic benefits of leveraging locality and cost-performance trade-offs, the design of the memory hierarchy, the processor-memory performance gap, power consumption, and cache operations. Cache miss rates, optimizations, virtual memory, and virtual machine monitors (VMMs) are also examined to enhance system dependability, isolation, and security.

In the laboratory sessions, students will master the fundamentals of C programming and x86 assembly. They will undertake various projects that apply the concepts discussed in the lectures.

Teaching methods

Lecture: multimedia presentation
Laboratory exercises: multimedia presentation, developing examples at the board, working in pairs

Bibliography

Basic
1. Peter Prinz, Tony Crawford „C in a nutshell”
2. David Patterson, John Hennessy „Computer organization and design”
Additional
1. Gynvael Coldwind „Zrozumieć programowanie”

Breakdown of average student's workload

	Hours	ECTS
Total workload	75	3,00
Classes requiring direct contact with the teacher	30	1,50
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	45	1,50