COURSE DESCRIPTION CARD - SYLLABUS

Course name
Low-level programming in C [N1Inf1>PLOW]

Course

Field of study Computing	Year/Semester 2/3
Area of study (specialization) –	Profile of study general academic
Level of study first-cycle	Course offered in polish
Form of study part-time	Requirements compulsory

Number of hours

Lecture 12	Laboratory classes 12	Other (e.g. online) 0
Tutorials 0	Projects/seminars 0	

Number of credit points

2,00

Coordinators

dr inż. Wojciech Complak
wojciech.complak@put.poznan.pl

Lecturers

Prerequisites

Students starting this course should have a basic knowledge of algorithms and programming in imperative programming languages. Students should be able to solve basic problems in the range of design, checking the correctness and implementing algorithms in the C programming language and the ability to acquire information from the indicated sources. Students should also understand the necessity to broaden own competences/be ready to cooperate within the team. In addition, in the field of social competence, the student must present such attitudes as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture, respect for other people.

Course objective

1. Teach students basic knowledge in the field of low-level programming based on the C and assembly programming languages. Presentation of the intrinsic functions of the compiler, the X86-32 architecture from the point of view of the application programmer, the basics of assembly language in the range of C language inline assembly and elements of the binary application interface. 2. Develop the ability to use an integrated programming environment in terms of support for low-level programming. 3. Presentation of specific hardware mechanisms and ways of using them with the help of intrinsic functions and inlined assembly language.

Course-related learning outcomes

Knowledge:
1. the student has a structured and theoretically founded general knowledge of key IT issues, and detailed knowledge of computer systems architecture and imperative programming languages
2. has basic knowledge of the life cycle of IT systems, both hardware and software, in particular about the system feasibility analysis and detailed design
3. knows the basic techniques, methods and tools used in the process of solving IT tasks in the field of algorithm analysis, computer systems architecture and algorithm implementation

Skills:
1. the student can - in accordance with the given specification - design, formulate a functional specification in the form of use cases at the level of formal description, formulate non-functional requirements for the time characteristic and implement software using an imperative programming language with the help of appropriate techniques and tools
2. has the ability to formulate algorithms and their implementation using imperative programming language and an integrated programming environment

Social competences:
1. the student understands that in computer knowledge and skills very quickly become obsolete
2. is aware of the importance of knowledge in solving engineering problems and knows examples and understands the reasons for malfunctioning IT systems that led to serious financial and social losses or to serious health conditions or even to death

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning outcomes presented above are verified as follows:
Formative assessment:
a) lectures:
- based on answers to questions related to subjects covered during former lectures,
b) laboratory classes:
- based on assessment of progress of implementation of assigned tasks,
Total assessment:
Verification of assumed learning objectives is based on:
- assessment of skills related to solving laboratory exercises,
- continual assessment, during each class (initial tests) - rewarding the progress in the ability to use the principles and methods learnt,
- assessment of knowledge and skills related to the implementation of project/laboratory tasks through tests during the third (C programming language) and the fifth (assembly language, assembly + C languages) laboratory class; tasks are both constructional (i.e. write a program) and analytical (i.e. determine the result of a given program).
Additional elements cover:
- discussing additional aspects of the class topic,
- ability of using the acquired knowledge while solving a given problem
- remarks related to improving teaching materials,
- pointing out perceptual difficulties enabling ongoing improvement of the teaching process

Programme content

The first lecture is devoted to presentation of the organisation of classes and a discussion/recall of the basic constructions of the C programming language. During the laboratory classes, the environment is prepared, and the principles of using, creating and launching programs in the C programming language are discussed.
The second lecture is devoted to more advanced mechanisms of the C language (C11 standard revision) such as structures, unions, bit fields and machine representation. Compiler internals, both cross-platform and specific to the X86 architecture, are also presented. During the laboratory classes, students create complex programs in the C programming language by analysing the data representation and the features of internal functions.
The third lecture is devoted to the presentation of the X86-32 architecture, the basics of assembly programming language, arithmetic, logical and control flow instructions as well as the principles of

inlining assembly language into the C language. During the laboratory classes, students begin to write simple inlined assembly code.
The fourth lecture deals with the stack mechanism, the convention of subroutines calling, passing parameters and the rules of creating subroutines in assembly language. During the laboratory classes, students create subroutines in assembly language.
The fifth lecture is a presentation of advanced processing mechanisms (FPU, multimedia and vector instructions) available on the IA-32 (X86-32) platform and using them with the intrinsic functions of the C language compiler and assembly language instructions. Examples are shown how to simplify code and improve performance. The last lecture is devoted to the use of the binary application interface to invoke system services and to summary of the course.

Teaching methods

1. lecture: multimedia presentation, presentation illustrated with examples shown on the blackboard, solving tasks, programming tools demonstration,
2. laboratory classes: solving tasks, discussion

Bibliography

Basic
1. Język ANSI C, B. W. Kernighan, D. M. Ritchie, Wydawnictwa Naukowo-Techniczne, dowolne wydanie
2. Mikroprocesory 80286, 80386 i i486, R. Goczyński, M. Tuszyński, Komputerowa Oficyna Wydawnicza HELP, 1991
3. Koprocesory arytmetyczne 80287 i 80387 oraz jednostka arytmetyki zmiennoprzecinkowej mikroprocesora i486, M. Tuszyński, R. Goczyński. Komputerowa Oficyna Wydawnicza HELP, 1992
4. MSDN, https://msdn.microsoft.com/
Additional
1. Język C. Szkoła programowania, Wydanie VI, Prata S., Helion, 2016
2. Asembler w koprocesorze, S. Kruk, Wydawnictwo MIKOM, 2003
3. Mikroprocesor 8086, Z. Mroziński, Wydawnictwa Naukowo-Techniczne, 1992
4. Koprocesor arytmetyczny 8087, Z. Mroziński, Wydawnictwa Naukowo-Techniczne, 1992

Breakdown of average student's workload

	Hours	ECTS
Total workload	50	2,00
Classes requiring direct contact with the teacher	24	1,00
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	26	1,00