National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"

Department of Computer Engineering

Programming Fundamentals. Part 2

Syllabus

Details of the academic discipline

Level of Higher Education	***First level of higher education (Bachelor's degree)***
Field of Study	*12 Information technologies*
Speciality	*121 Software engineering*
Education Program	*Computer Ssystems Software Sengineering*
Type of Course	*Normative*
Mode of Studies	*full-time*
Year of studies, semester	*1st year, 2nd semester*
ECTS workload	*5.5 ECTS credits (165 hours). Lectures 36 hours, Laboratory work (computer workshop) - 54 hours, Self-study work - 75 hours*
Testing and assessment	*Exam*
Course Schedule	*http://roz.kpi.ua/*
Language of Instruction	*English*
***Head of the course* / Course Instructors**	*Head of the course: head of the department, D.sc., Prof. Stirenko S. G. , sergii.stirenko@gmail.com* *Shemsedinov T.G., senior lecturer of the Department of Computer Engineering. timur _ shemsedinov @ gmail . com* *Laboratory: assistant of the department of Computer Engineering* *Kuhar V.V. kukhar . vitalii @ gmail . com*
Placement of the course	https://github.com/HowProgrammingWorks

Outline of the Course

1. Description of the academic discipline, its purpose, subject of study and learning outcomes

The goal of the educational discipline is to develop students' programming abilities (competencies), fluency in programming syntax and methodology, understanding of basic data structures and paradigms. According to the results of studying the discipline, the student should be able to solve professional tasks and possess the following competencies:

- Ability to abstract thinking, analysis and synthesis (GC01)
- Ability to search, process and analyze information from various sources (GC06)
- Ability to identify, classify and formulate Software requirements (PC01)
  Ability to participate in Software Design, including Modeling (formal description) its Structure, Behavior, and Operating Processes (PC02)
- Ability to develop Architectures, Modules and Program System Components (PC03)

- Knowledge of Information Data Models, ability to create Software for storing, extracting and processing Data (PC07)
- Ability to use Fundamental and Interdisciplinary Knowledge to successfully solve Software Engineering problems (PC08)
- Ability to accumulate, process and systematize Professional Knowledge about the creation and maintenance of Software and recognition of the importance of lifelong learning (PC10)
- Ability to Algorithmic and Logic thinking (FC14)

After mastering the academic discipline, students must demonstrate the following learning outcomes:

- Analyze, purposefully search and select the Information and Reference Resources and Knowledge necessary for solving Professional Tasks, taking into account the Modern Achievements of Science and Technology (PLO01)
- Know the basic Processes, Phases, and Iterations of the Software Lifecycle (PLO03)
- Know and apply in practice the Fundamental Concepts, Paradigms and Basic Principles of functioning of Language, Instrumental and Computational Means of Software Engineering (PLO07)
- Know and apply methods for developing Algorithms, Software Design and Data and Knowledge Structures (PLO13)
- Know and be able to apply Information Technologies for Data Processing, Storage and Transmission (PLO18)

2. Pre-requisites and post-requisites of the discipline (place in the structural and logical scheme of training according to the relevant educational program)

Preceding disciplines: Programming Fundamentals. Part 1

Disciplines for which this course prepares: Software Engineering Components. Parts 1 and 2, Fundamentals of Software Development on the Node Js, Agile Programming Techniques, System programming, Software modeling.

3. Content of the academic discipline

Topic 1. State of applications, data structures and collections
Topic 2. Approaches to working with state: statefulandstateless
Topic 3. Structures and records
Topic 4. Stack, queue, dec
Topic 5. Trees and graphs
Topic 6. Projections and display of data sets
Topic 7. Estimation of computational complexity
Topic 8. Structure of the application: files, modules, components
Topic 9. Object, prototype and class
Topic 10. Dependencies and libraries
Topic 11. Regular expressions

Topic 12. Factories and pools
Topic 13. I/O (input-output) and files
Topic 14. Monomorphic and polymorphic code, inline cache, hidden classes
Topic 15. Code performance measurement and optimization
Topic 16. Asynchronous programming on callbacks
Topic 17. Asynchronous programming on promises
Topic 18. Asynchronous functions, async/await, thenable, error handling
Topic 19. Immutable data structures
Topic 20. Automatic programming: finite state machines (state machines)
Topic 21. JavaScript Singleton template
Topic 22. Functional objects, functors and monads
Topic 23. Asynchronous generators and asynchronous iterators
Topic 24. Enumerated type (enum)

4. Educational materials and resources

Basic:

1. Shemsedinov T.G., Nechay D.O., Kuhar V.V., Orlenko O.A., Golikov O.G., Bilochub M.M., Dukhin V., Ivanova L.A., Chornenkyi A.Yu. . and other. Code examples and project examples [Electronic resource] are available at: https://github.com/HowProgramming Works/
2. Refactoring: Improving the Design of Existing Code // MartinFowler
3. Clean Code: A Handbook of Agile Software Craftsmanship // Robert C. Martin
4. Introduction to Algorithms, 3rd Edition // Thomas H. Cormen
5. Design Patterns // Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides

Additional:

1. Shemsedinov T.G., Nechay D.O., Kuhar V.V., Orlenko O.A., Golikov O.G., Bilochub M.M., Dukhin V., Ivanova L.A., Chornenkyi A.Yu. . and other. The Metarhia technology stack [Electronic resource] is located at: https://github.com/metarhia/
2. Algorithms Unlocked // Thomas H. Cormen
3. The Art of Computer Programming // DonaldKnuth
4. Code Complete // Steve McConnell
5. Designing Object Oriented C++ Applications Using The Booch Method // Robert C. Martin
6. Extreme Programming Explained // Kent Beck
7. Analysis Patterns: Reusable Object Models // Martin Fowler

Educational content

5. Methodology

The main tasks of the cycle of laboratory classes (computer workshop) - acquisition of practical skills in using data structures and collections, writing code, creating multi-module libraries and programs, optimizing code, debugging and decomposing code, group work and using version control systems.

No. z/p	Name of laboratory work (computer workshop)	Number of aud. hours
1	Implementation of linked lists	9
2	Building modules and dependencies, application structure	9
3	Working with files: descriptors and file streams	9
4	Traversal of trees and graphs	9
5	Code optimization for V8 virtual machine	9
6	Using reflection and introspection	9
	Total:	54

6. Self-study

In the process of completing individual tasks, students must process the knowledge gained during lectures and independent work, independently study specific topics, deepen their knowledge for further study. Students' independent work consists of the following:

- preparation for lecture classes on the study of previous lecture material;
- review and modification of code examples and projects from git repositories provided by the teacher;
- performance of laboratory work with the study of theory and implementation of the given topic in program code;
- preparation and participation in the discussion of topics at seminars;
- peer review of fellow students;
- preparation and protection of the code for the teacher's code review.

No. z/p	The name of the topic submitted for self-study	Number of hours
1	Contribution to Open-Source projects	12
2	Setting up the test system and IDE development environment	12
3	Create and configure a Github profile	12

Policy and Assessment

7. Course policy

During classes in an academic discipline, students must adhere to certain disciplinary rules:
1) use chat groups, repositories and the execution environment in such a way as not to create problems with unnecessary notifications to other participants of the educational process and the teacher ;
2) perform work on time and commit everything to the version control system every day, do not create many commits in one pull request, do not create large commits, divide everything into separate thematic commits, clearly and clearly describe commits and pull requests, add tags for linking issues , PR, and commits in repositories ;
3) it is not allowed to use pirated copies of development environments, operating systems or other development and deployment tools both on one's computing devices and on cloud ones ;
4) spam, post too many memes and stickers in groups and repositories;
5) if a question arises, you must first search in the course materials, then on the Internet, then ask assistants and other students, and only then, only when the solution is not found or is unclear, bother the teacher;

Labs are submitted only through Github in open source (students add the MIT license to the code), and the repository must have a development history so that the instructor can trace the sequence of code writing and authorship . Development takes place in git feature branches, after which the code is checked through PR and includes a review that the teacher makes in the pull request . The review history remains in the student's Github account.

8. Types of control and rating system for evaluating learning outcomes (RSO)

Types of control from the educational discipline "Fundamentals of programming 2. Programming methodologies" include:

Laboratory work
Independent performance of 6 laboratory works is planned.
The topics of laboratory works are coordinated in time and content with the topics of lectures

Current control :
There are 2 preliminary reviews for the course, which fully cover the subjects of the academic discipline discussed in the lectures. To the code review, the teacher can add theoretical questions that reveal the student's understanding of the topic.

Semester control
The final review of the code is done in several approaches, but at the deadline, the pull-request is closed, and all comments, whether corrected or not, are recorded.

Exam
conducted in the form of an interview with the student to objectively determine the level of knowledge, skills and practical skills acquired during the semester or pair live coding with a

teacher or assistant, pair coding with another student or return to the code written during the semester and eliminate its shortcomings or discuss its features .

The rating of the student from the credit module consists of the points he receives for the types of work according to table 4.

Table 4

Assessment of individual types of student's academic work

Section 1.2		Section 3.4	
Kind educational work	Maximum number points	Kind educational work	Maximum number points
Performance and protection of laboratory work no 1	5	Performance and protection of laboratory work no 4	5
Performance and protection of laboratory work no 2	5	Performance and protection of laboratory work No. 5	5
Performance and protection of laboratory work no 3	5	Performance and protection of laboratory work No. 6	5
Current code review #1	5	Current code review #2	5
		Final code review	1 0
		Exam	50
		In just one semester	100

Total for laboratory works (maximum number of points) – 30

The student's individual semester rating (final semester rating **RD**) is the sum of points received by the student during the semester by participating in seminars, code reviews, and discussions .

All students, regardless of whether they have met all the conditions for admission to the semester certification of the credit module and have a rating of at least 60 points, undergo an interview with the teacher.

A necessary condition for a student's admission to the exam is his individual semester rating (**RD**) of not less than 30% of the maximum points , i.e. 30 points, 4 laboratory tests passed and one positive certification in the semester. If at least one of the mentioned conditions is not met, the student will not be admitted to the exam.

The sum of the final semester (**RD**) and examination rating grades in points constitutes the final semester rating grade, which is converted into grades according to the national scale and the ECTS scale (Table 5).

Table 5

Correspondence of rating points to grades on the university scale

Rating	*Grade*
100-95	Excelent
94-85	Very good
84-75	Good
74-65	Satisfactorily
64-60	Sufficient
Less than 60	Fail
Admission conditions not met	Not Graded

Working program of the academic discipline (syllabus):

designed by Shemsedinov T.G., a senior lecturer at the Department of Computer Engineering

adopted by the Department of Computer Engineering (Protocol No. 10 dated 05/25/2022)

agreed by the Methodical Commission of the faculty (protocol No. 10 dated 06/09/2022)