Numele și prenumele (cu inițiala / inițialele tatălui, așa cum apar în catalog)

____

CNP ____

Facultatea de____

Specializarea / secția ____

Anul de studiu : _____________ **Grupa de studiu :** _____________

Restanță (se va completa anul universitar pentru care se dă restanța, dacă este cazul – de exemplu 2017-2018) : ___________________

Această secțiune se completează de către examinator :

	Ascultare	Citire	Scriere	Vorbire
Nota obținută (în cifre)				
Nivel C.E.F.R. echivalent				

ENGLISH PROFICIENCY (FINAL) TEST

Solve the following tasks and then send me an e-mail containing ONLY your solutions/answers.

I. LISTENING COMPREHENSION 10 points

Listen to the attached audio files (AF1, AF2, AF3 and AF4), then choose ONE and take/write your notes on the main/essential ideas.

Finally, briefly express your opinion on it (about 30 words).

II. READING COMPREHENSION 10 points

1. Read the following texts (A1, A2 and A3), then choose ONE and work out a suitable title for each paragraph (1, 2 and 3).

1. TEXT A1

1. The sharply increasing data deluge in the big data era brings huge challenges on data acquisition, storage, management and analysis. Traditional data management and analytics systems are based on the relational database management system (RDBMS). However, such RDBMSs only apply to structured data, other than semi-structured or unstructured data. In addition, RDBMSs are increasingly utilizing more and more expensive hardware. Apparently the traditional RDBMSs cannot handle the huge volume and heterogeneity of big data.

2. The research community has proposed some solutions from different perspectives. For example, cloud computing is utilized to meet the requirements on infrastructure for big data, e.g., cost efficiency, elasticity, and smooth upgrading/downgrading. For solutions of permanent storage and management of large-scale disordered datasets, distributed file systems and NoSQL databases are good choices. Such programming frameworks have achieved great success in processing clustered tasks, especially for webpage ranking. Various big data applications can be developed based on these innovative technologies or platforms. Moreover, it is non-trivial to deploy the big data analytics systems.

3. One source of all of this data is the Internet of Things. Not only do people send messages: now cars, phones, and machines communicate with each other. This enables much closer monitoring of patient health, to include little wristbands to monitor the wearer's pulse, temperature, blood pressure forwarded on to the patient's physician. This does indicate the tons of data in which a miniscule bit of important data exists. Of course, signals are sent only when critical limits are reached, just as vending machines can send signals to replenish stock at their critical limits. Monitors in homes can reduce electricity use, thus saving the globe from excessive warming. Cars can send signals to dealers about engine problems, so that they might send a tow truck to the location provided by the car's GPS. Insurance already advertise their ability to attach devices to cars to identify good drivers, a euphemism for detection of bad driving so that they can cancel policies more likely to call for claims.

2. TEXT A2

1. We're programmers. Programmers are, in their hearts, architects, and the first thing they want to do when they get to a site is to bulldoze the place flat and build something grand. We're not excited by incremental renovation: tinkering, improving, planting flower beds.

There's a subtle reason that programmers always want to throw away the code and start over. The reason is that they think the old code is a mess. And here is the interesting observation: they are probably wrong. The reason that they think the old code is a mess is because of a cardinal, fundamental law of programming:

It's harder to read code than to write it.

2. This is why code reuse is so hard. This is why everybody on your team has a different function they like to use for splitting strings into arrays of strings. They write their own function because it's easier and more fun than figuring out how the old function works.

The idea that new code is better than old is patently absurd. Old code has been used. It has been tested. Lots of bugs have been found, and they've been fixed. There's nothing wrong with it. It doesn't acquire bugs just by sitting around on your hard drive.

3. Each of these bugs took weeks of real-world usage before they were found. The programmer might have spent a couple of days reproducing the bug in the lab and fixing it. If it's like a lot of bugs, the fix might be one line of code, or it might even be a couple of characters, but a lot of work and time went into those two characters.

When you throw away code and start from scratch, you are throwing away all that knowledge. All those collected bug fixes. Years of programming work.

3. TEXT A3

1. Mathematics, like other human endeavors, has both a “what” and a “how.” The “what” is the subject matter of mathematics, ranging from numbers to geometry to calculus and beyond. The “how” depends upon who is doing the mathematics. At the elementary school level, we deal with everything very concretely. At the high school level, when we learn algebra and geometry, things get more abstract. We prove some things, for example in geometry, and do others computationally, for example algebra. To a mathematician, by contrast, there is no split between how we do algebra and how we do geometry: everything is developed axiomatically, and all facts are proved rigorously.

2. Euclid used an axiomatic system - which is needed for proofs - in the field of geometry. Today, virtually all branches of pure mathematics are based on axiomatic systems, and work in pure mathematics involves the construction of rigorous proofs for new theorems. Much of the great mathematics of the past has been recast with a precision missing from its original treatment. Abstract algebra, for example, which received its modern form only in the last one hundred years, reconstructs the elementary algebra studied in high school in a rigorous, axiomatic fashion. A lot of applied mathematics today also has rigorous foundations (though the work of applied mathematicians, while no less challenging than pure mathematics, is not always oriented toward proofs).

3. Theorems are not proved in a vacuum. To prove one theorem, we usually need to use various relevant definitions, and theorems that have already been proved. If we do not want to keep going backwards infinitely, we need to start with some objects that we use without definition, as well as some facts about these objects that are assumed without proof. Such facts are called axioms, and a body of knowledge that can be derived from a set of axioms is called an axiomatic system. In modern abstract mathematics, we take set theory as our basis for all arguments. In each branch of mathematics, we then give specific axioms for the objects being studied.

Mathematical correctness is certainly the ultimate test of the validity of a proof, but to allow us to judge mathematical correctness, however, a number of important factors in the proper writing of mathematics are needed.

2. Read the following texts (B1, B2 and B3), then choose ONE and briefly enumerate its main ideas.

1. TEXT B1

Technology, especially the computer, is rapidly changing the world. The ubiquitous nature of the computer is probably not even realized by most people. We see them in our homes, in schools, and in libraries, but computer technology can be found in cars, cell phones, and even appliances like washing machines. With the increased reliance on technology, some people are wary of the changes and wonder if society is moving too quickly. Other people embrace the changes and look forward to the benefits of each new innovation.

One concern deals with privacy. Many people today enjoy the ease of shopping, banking, and paying bills online. However, if your personal information is not securely encrypted, problems can arise. Without encoding private information, unscrupulous people can access credit card numbers, bank accounts, or other personal information. Your money can easily be stolen but, even worse, so can your identity. If this happens, the criminal can use your name to commit crimes from theft to murder. It can take years and loads of paperwork to get your good name back. Another area that worries some people is the idea of embedding computer chips in clothing and possibly in a person’s hand or brain.

Researchers are looking at attaching global positioning systems (GPS) to jackets and putting miniature cameras into necklaces. A person could simply push buttons on one’s sleeve to listen to music or text a message. One may even be able to swipe a hand over a scanner to pay for a bill instead of using a credit card. The question is whether the benefits of having less to carry outweigh the possible loss of privacy. Some people can be considered paranoid in their concern that someone is constantly watching them; on the other hand, George Orwell’s idea of Big Brother, as presented in his novel 1984, could become a reality.

2. TEXT B2

Programming is fun. Where else can you spend your days devising machines of your own imagination, build them without ever touching a hammer or staining your clothes, make them run as by magic, and get paid — not too bad, thanks for asking — at the end of the month?

Programming is tough. Where else do products from the most prestigious companies fail even in ordinary use? Where else does one find so many users complaining so loudly? Where else do engineers routinely work for hours or days trying to understand why something that should work doesn’t?

Get ready for the mastery of programming and its professional form, software engineering; get ready for both the toughness and the fun.

By going into computing science you have chosen one of the most exciting and fast-moving topics in science and technology. Fifty years ago it was not even recognized as a scientific subject; today hardly a university in the world is without a CS department. Thousands of books, journals, magazines and conferences cover the field. The global revenues of its industry — called information technology or IT — are in the trillions. No other field, in the history of technology, has undergone growth of either such magnitude or such speed.

And we have made a difference. Without software there would be no large-scale plane travel, and in fact no modern planes (or modern cars, or high-speed trains) since their design requires sophisticated “Computer-Aided Design” software. To pay its workers, any large corporation would employ hundreds of people just to write the paychecks. A phone would still be a device tied to the wall by a cable. After taking a picture, you still could not see the result until the roll of film came back from processing. There would be no video games, no camcorders, no iPods and no iPhones, no Skype, no GPS to guide you to your destination even when there is no one around to ask. To produce a report you would still hand-write a draft, give it to a typist, and go through rounds of correction requests. A sudden itch to know the name of the captain in The Grand Illusion, or the population of Cape Town, or the author of a familiar citation, would require (rather than typing a couple of search words and getting the answer in a blink) a trip to the library. The list goes on; at the heart of countless practices that now pervade our daily life lie programs — increasingly sophisticated programs.

All this does not happen by itself. While computers may have become a commodity, programs — without which computers would be useless — definitely are not. Programming, the task of constructing new programs or improving existing ones, is a challenging intellectual pursuit that requires programmers possessing creativity and experience.

Although more and more people are acquiring basic computing proficiency, being able to program at a professional level is another matter, and is what a curriculum in computing science will bring you.

For comparison, consider mathematics. A few centuries ago, just being able to add and subtract 5-digit numbers required a university education, and in return provided qualifications for such good jobs as accountant. Nowadays these skills are taught in grade school; if you want to become an engineer or a physicist, or just a stock trader, you need to study more advanced mathematical topics, such as calculus, in a university. The boundary between basic training and university-level education has moved up.

Computing is following the same evolution, only much faster — the scale is in decades, not centuries. Not so long ago, being able somehow to program a computer was enough to land a job. Do not expect this today; an employer will not be much more impressed if your résumé states “I have written programs” than if you say you can add numbers.

What increasingly counts is the difference between having some basic programming experience and being a software engineer. The former skill will soon be available to anyone who has gone through a basic education; but the latter is a professional qualification, just like advanced mathematics.

3. TEXT B3

By any measure, C++ has been a tremendous success, but even its most ardent proponents won't deny that it's a complicated beast. This complexity influenced the design of C++'s most widely used successors, Java and C#. Both strove to avoid C++'s complexity — to provide most of its functionality in an easier-to-use package.

Complexity reduction took two basic forms. One was elimination of "complicated" language features. C++'s need for manual memory management, for example, was obviated by garbage collection. Templates were deemed to fail the cost/benefit test, so the initial versions of these languages chose to exclude anything akin to C++'s support for generics.

The other form of complexity reduction involved replacing "complicated" C++ features with similar, but less demanding, constructs. C++'s multiple inheritance morphed into single inheritance augmented with interfaces. Current versions of Java and C# support templatesque generics, but they're simpler than C++'s templates.

These successor languages aspired to far more than simply doing what C++ did with reduced complexity. Both defined virtual machines, added support for runtime reflection, and provided extensive libraries that allow many programmers to shift their focus from creating new code to gluing existing components together. The result can be thought of as C-based "productivity languages." If you want to quickly create software that more or less corresponds to combinations of existing components — and much software falls into this category — Java and C# are better choices than C++.

But C++ isn't a productivity language; it's a systems programming language. It was designed to rival C in its ability to communicate with hardware (e.g., in drivers and embedded systems), to work with C-based libraries and data structures without adaptation (e.g., in legacy systems), to squeeze the last drop of performance out of the hardware it runs on. It's not really an irony that the performance-critical components of the virtual machines beneath Java and C# are written in C++. The high-performance implementation of virtual machines is a job for a systems language, not a productivity language.

III. WRITING 10 points

Write an essay of about 200 words on ONE of the following topics:

1. The job of your dreams
2. A university course that has had an important impact on your formation
3. An interesting subject in the field of Computer Science or Mathematics