Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Домашняя работа №1

По предмету: «Иностранный язык»

Студент: Гасанзаде М.А.,
Группа: ИУ7-66

Москва, 2020 г.

Contents

1. TEST TASK

1st Fill in blanks with the appropriate words below:

1. apply
2. input
3. output
4. trained
5. predict
6. consumption
7. requirement
8. relationship
9. feature
10. therefore

2nd Choose the correct option:

1. b
2. b
3. b
4. c
5. a
6. b
7. c
8. c
9. b
10. a

2. ARTICLE BY SPECIALTY

What Makes Great Programmers Different?

The keys to being a good programmer are well known. Greatness, however, requires something else altogether.

A recent meme running on programming sites across the Web is a generally humorous discussion of the attributes of bad programmers. If we put aside the humor — everyone has a war story here — in my experience, bad programmers fall into two main categories: the dim and the reckless. The dim ones generally have a small skill and are unaware of its limitations. They generally avoid deep technical conversations, but when they do speak, they often greatly overreach in the estimation of what they're capable of doing or what they know. Curiously, I regularly receive article manuscripts from these folks ("Want an article on building your own DBMS in less than 100 lines of JavaScript?") and it's difficult to respond courteously to the proposals, so I often beg off by simply citing that the pieces are not what we're looking for.

The other group, the reckless, consists of those who have the skills but don't have the discipline. They are the cowboy programmers of yore. They code according to their own desires, do nothing to integrate their work with that of others, scrimp on basic discipline, and cause work for other team members. Fortunately, this model is dying out (somewhat too slowly) as IT becomes aware that their cost is rarely ever offset by their benefits.

I guess you could add a third grouping of folks, the jerks, who wield their roles so poorly that they drag down the entire team. They often come from one of the other two previous groups, but have been given enough responsibility that they can make life miserable for others. The only solution when faced with them is to hope that they eventually are let go or to leave yourself.

The Good Guys

To happier topics: Good programmers are, I believe, the core of the profession. They are competent, desire the right outcomes, and strive to contribute to the team. They frequently have well-defined skills, a good understanding of development disciplines, and are open to new approaches as long as the material can be integrated with what they already know. They write solid code, although only occasionally does it show a truly imaginative solution. More commonly, when faced with a difficult problem, they will lapse into quick and dirty hacking. They block themselves from greatness by not having the curiosity or know-how at their command to be more than they are. That is, they refine their skills principally by continued application, not by learning new technologies — unless required to do so by job demands. Such programmers are at risk of slipping into the lower grouping

by letting their skills atrophy. I discussed several examples of this in my previous editorial on coding from within the echo chamber.

Really good programmers have another dimension to them: They carry an abiding passion for programming. They like to solve challenging problems and they like to solve them well. They are not satisfied with writing one more CRUD app. They want the magical work that is hard and requires extended effort to bring to fruition. If they can't find this satisfaction in the workplace, they find it in personal projects or by contributing to open-source projects. They frequently test new technologies, try out new languages, explore new tools, and read about programming. *Dr. Dobb's* readers are found in great numbers in this group, I believe. *Dr. Dobb's* has no exclusive franchise here. These developers are interested, consume programming books, and hide out in developer forums as well. They revel in challenge and have a constant sense of searching. They're looking for the best answer to a problem, or the most elegant.

The Greats

The next tier up — the final tier — consists of great programmers who have supernormal gifts that enable them to do easily what good programmers find difficult. Due to my work in the press, I've met more than my share of these talents, although I've not had the opportunity to work with many of them. To my eye, the traits that most stand out are three in number: an excellent memory, a conspicuously wide knowledge of programming techniques and the judgment to know when to apply which, and a deep ability to relate details to the larger picture. The last trait — being capable of quickly shifting registers from the large picture to small details and back again — relies on the strong memory and operates on an almost automatic basis. There is an effortlessness to it, which makes them particularly good architects.

There's one discipline they all share as well, which appears only in varying degrees in the earlier levels: Without exception, they possess a very deep and intimate knowledge of their tools. Be it the editor, the compiler, or the framework, they know the ins and outs of its features and they navigate efficiently. They use a much wider array of features because they know exactly how the tools work.

Knowledge of tools, coupled with an extensive, tested palette of programming techniques, and the ability to remember large amounts of the code base, while relating low-level details to the whole with unconscious ease — these are the traits I see most often in great programmers. And they're the skills I continue to aspire to in my work.

Jobs in Computing.

There is a wide range of computer related specialists. Computing is an area where competition between companies is considerable and technology is moving very

quickly. With the increasing number of computers used in the office and at home and as part of industrial and communications equipment there`s a growing need for engineers to design these as well as to service them.

Read the text and complete this table.

1. job title
2. nature of work
3. formal qualifications
4. personal qualities
5. technical skills
6. how to get started
7. how to make progress

How to Become A Programming Expert

The primary requirements for being a good programmer are nothing more than a good memory, an attention to detail, a logical mind and the ability to work through a problem in a methodical manner breaking tasks down into smaller, more manageable pieces.

However, it's not enough just to turn up for a job interview with a logical mind as your sole qualification. An employer will want to see some sort of formal qualification and a proven track record. But if you can show someone an impressive piece of software with your name on it, it will count for a lot more than a string of academic qualifications.

So what specific skills are employers looking for? The Windows market is booming and there's a demand for good C, C++, Delphi, Java and Visual Basic developers. Avoid older languages such as FORTRAN and COBOL unless you want to work as a contract programmer.

For someone starting out, my best advice would be to subscribe to the programming magazines such as Microsoft Systems Journal. Get one or two of the low-cost 'student' editions of C++, Visual Basic and Delphi. Get a decent book on Windows programming. If you decide programming is really for you, spend more money on a training course.

Questions:

1.What is your future profession?

My future profession is a programmer.

2.What qualifications are you going to have?

I wanna be an application developer, but now I need to study for a bachelor

3.What courses have you done?

Programming in high-level languages, also low-level languages. Now we learning to work with Linux & Unix OS kernels, write some modules by themselves.

4.What personal qualities are needed to become a good specialist in your field?

The primary requirements for being a good programmer are nothing more than a good memory, an attention to detail, a logical mind and the ability to work through a problem in a methodical manner breaking tasks down into smaller, more manageable pieces.

5.What technical skills do you expect to gain during your studies at university?

Knowledge of tools, coupled with an extensive, tested palette of programming techniques, and the ability to remember large amounts of the code base, while relating low-level details to the whole with unconscious ease — these are the traits I see most often in great programmers.

6.What future possibilities do you see of upgrading your skill set?

The other group, the reckless, consists of those who have the skills but don't have the discipline. They are the cowboy programmers of yore. They code according to their own desires, do nothing to integrate their work with that of others, scrimp on basic discipline, and cause work for other team members.

7.What type of practical tasks have you dealt with or would like to deal with?

I wrote CW via backward ray tracing. Now I wanna create a website forum, but have not decided on the topic yet.

3. EXAM TOPIC

Operating Systems. Linux

An operating system is the core set of software on a device that keeps everything together. Operating systems communicate with the device's hardware. They handle everything from your keyboard and mice to the Wi-Fi radio, storage devices, and display. In other words, an operating system handles input and output devices. Operating systems use device drivers written by hardware creators to communicate with their devices.

The operating system sits in between the applications you run and the hardware, using the hardware drivers as the interface between the two. For example, when an application wants to print something, it hands that task off to the operating system. The operating system sends the instructions to the printer, using the printer's drivers to send the correct signals. The application that's printing doesn't have to care about what printer you have or understand how it works. The OS handles the details.

Linux® is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work. Think about an OS like a car engine. An engine can run on its own, but it becomes a functional car when it's connected with a transmission, axles, and wheels. Without the engine running properly, the rest of the car won't work.

What does Linux OS include?

- Kernel - the base component of the OS. Without it, the OS doesn't work. The kernel manages the system's resources and communicates with the hardware. It's responsible for memory, process, and file management.
- **System user space** - the administrative layer for system level tasks like configuration and software install. This includes the shell, or command line, daemons, processes that run in the background, and the desktop environment, the interface the users interacts with.
- **Apps** - a type of software that lets you perform a task. Apps include everything from desktop tools and programming languages to multiuser business suites. Most Linux distributions offer a central database to search for and download additional apps.

Topic plan:

1. About OS
2. The history of Linux
3. Linux, what it can do
4. What does Linux OS include?
5. Conclusion