МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

SELF-ACCESS GUIDE FOR MASTERS OF APPLIED MATHEMATICS AND MECHANICS

Учебно-методическое пособие для вузов

Составитель:
Н. М. Шишкина

Воронеж
Издательский дом ВГУ
2015

Введение

Предлагаемое учебно-методическое пособие предназначено для магистров, обучающихся на факультете прикладной математики, информатики и механики.

Пособие состоит из четырех разделов: Reading and Speaking; Reading and Summarizing Information; Writing Research Papers; Grammar Exercises и приложения. В разделе Reading and Speaking предлагается аутентичный учебный материал, направленный на развитие навыков устной речи, совершенствование умений высказывать свое мнение в форме минимонолога и диалога-обмена мнениями. Второй раздел Reading and Summarizing Information включает в себя тексты научного характера и предусматривает развитие навыков чтения научной и научно-технической литературы с целью извлечения основной информации по определенному алгоритму и последующего ее обобщения в устной или письменной реферативной форме. Раздел Writing Research Papers имеет своей целью развитие навыков письменной научной коммуникации и включает справочный материал по написанию и оформлению научных статей на английском языке. Раздел Grammar Exercises содержит грамматические упражнения. В приложении дается справочный материал по образованию грамматических форм английского глагола в активном и пассивном залогах.

Пособие предназначено для практических занятий по английскому языку и может быть использовано как на аудиторных занятиях, так и в ходе самостоятельной работы студентов.

4. Wiener wrote a book about
   a) himself,
   b) childhood,
   c) philosophy.
5. According to the text, most scientists
   a) know a lot about many different subjects,
   b) are familiar with applied science,
   c) deal with certain fields only.

Norbert Wiener

Norbert Wiener, the famous applied mathematician, was born in 1894 in the USA and died in Stockholm, Sweden, in 1964. His father was a professor of Slavonic languages at Harvard. Norbert was a very intelligent child and his father was determined to make him a famous scholar. This is indeed what he became, being awarded a PhD by Harvard at the age of 18. He also studied Philosophy, Logic and Mathematics at Cambridge and Gottingen.

His first important position was that of Instructor of Mathematics at MIT (Massachusetts Institute of Technology) in 1919, followed by that of Assistant Professor in 1929 and of Professor in 1931. Two years later, in 1933, he was elected to the National Academy of Sciences (USA), from which he resigned in 1941. In 1940 he started to work on a research project at MIT on anti-aircraft devices, a project which played an important part in his development of the science of cybernetics.

The idea of cybernetics came to Wiener when he began to consider the ways in which machines and human minds work. This led to the development of the idea of cybernetics, which is the study of the ways humans and machines process information, in order to understand their differences. It often refers to machines that imitate human behaviour. The term was coined from the Greek *kubernetike* which means *the art of the steersman* (the skill of a captain when controlling the ship). This idea made it possible to turn early computers into machines that imitate human ways of thinking, particularly in terms of control (via negative feedback) and communication (via the transmission of information).

Norbert Wiener was also deeply attracted to mathematical physics. This interest originated in the collaborative work that he did with Max Born in 1926 on quantum mechanics. But Wiener's interests were not limited to logic, mathematics, cybernetics or mathematical physics alone, as he was also familiar with every aspect of philosophy. In fact, he was awarded his doctorate for a study on mathematical logic that was based on his studies in philosophy. In addition to that, in a very different field, he wrote two short stories and a novel. Wiener also published an autobiography in two parts: *Ex-Prodigy: My Childhood and Youth* and *I am a Mathematician*.

Norbert Wiener was an amazing mathematician, who was gifted with philosophical insight. In an age when scientists tended, and still tend, to specialise in their own very specific fields, this man was interested and involved in many different disciplines. Due to this, he was able to draw on many resources in his varied research, thus making him an incredibly successful *applied* scientist. Wiener was one of the most original and significant contemporary scientists and his reputation was securely established in the new sciences such as cybernetics, theory of information and biophysics.

3. Discuss these questions with your partners.

- What do scientists do?
- Do you think it is difficult to be a scientist? Why? Why not?
- Do you think a scientist should have an all-round education?
- Do you think it is a good idea to award prizes to scientists for their work? Why? Why not?
- How do scientists in your country get support to conduct their research?
- Would you like to be a scientist? Why? Why not?

Unit II. Reading and Summarizing Information

Useful Information

To summarize the article you look at a whole text and reduce it to a few sentences retaining the main points. The first sentence of the summary should express the overall message of the text. The remaining sentences should present the most important ideas in the text. A good summary does not need to include details or supporting evidences for the main ideas.

1a. Read the title of the text to know what it deals with.

1b. Read the text carefully to know its content in more detail and complete the tasks that follow.

Learning C++

The most important thing to do when learning C++ is to focus on concepts and not get lost in language-technical details. The purpose of learning a programming language is to become a better programmer; that is, to become more effective at designing and implementing new systems and at maintaining old ones. For this, an appreciation of programming and design techniques is far more important than an understanding of details; that understanding comes with time and practice.

C++ supports a variety of programming styles. All are based on strong static type checking, and most aim at achieving a high level of abstraction and a direct representation of the programmer's ideas. Each style can achieve its aims

effectively while maintaining run-time and space efficiency. A programmer coming from a different language (say C, Fortran, Smalltalk, Lisp, ML, Ada, Eiffel, Pascal, or Modula-2) should realize that to gain the benefits of C++, they must spend time learning and internalizing programming styles and techniques suitable to C++. The same applies to programmers used to an earlier and less expressive version of C++.

Thoughtlessly applying techniques effective in one language to another typically leads to awkward, poorly performing, and hard-to-maintain code. Such code is also most frustrating to write because every line of code and every compiler error message reminds the programmer that the language used differs from "the old language". You can write in the style of Fortran, C, Smalltalk, etc., in any language, but doing so is neither pleasant nor economical in a language with a different philosophy. Every language can be a fertile source of ideas of how to write C++ programs.

However, ideas must be transformed into something that fits with the general structure and type system of C++ in order to be effective in the different context. Over the basic type system of a language, only Pyrrhic victories are possible.

C++ supports a gradual approach to learning. How you approach learning a new programming language depends on what you already know and what you aim to learn. There is no one approach that suits everyone. My assumption is that you are learning C++ to become a better programmer and designer. That is, I assume that your purpose in learning C++ is not simply to learn a new syntax for doing things the way you used to, but to learn new and better ways of building systems. This has to be done gradually because acquiring any significant new skill takes time and requires practice. Consider how long it would take to learn a new natural language well or to learn to play a new musical instrument well. Becoming a better system designer is easier and faster, but not as much easier and faster as most people would like it to be.

It follows that you will be using C++ – often building real systems – before understanding every language feature and technique. By supporting several programming paradigms, C++ supports productive programming at several levels of expertise. Each new style of programming adds another tool to your toolbox, but each is effective on its own and each adds to your effectiveness as a programmer. C++ is organized so that you can learn its concepts in a roughly linear order and gain practical benefits along the way. This is important because it allows you to gain benefits roughly in proportion to the effort expended.

In the continuing debate on whether one needs to learn C before C++, I am firmly convinced that it is best to go directly to C++. C++ is safer, more expressive, and reduces the need to focus on low-level techniques. It is easier for you to learn the trickier parts of C that are needed to compensate for its lack of