КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Кафедра английского языка
для естественнонаучных специальностей

ENGLISH for MASTERS of COMPUTING

Учебное пособие для студентов-магистров
ИВМиИТ- ВМК

Изд-во КФУ, 2013

Печатается по решению Учебно-методической комиссии и Ученого совета Института Языка ФГАОУВПО «Казанский (Приволжский) федеральный университет»

Кафедра английского языка для естественнонаучных специальностей Института языка КФУ
Протокол № __ от _ ___ 2013 г.

Ученый совет Института языка КФУ
Протокол № __ от ____ 2013 г.

Составители:
к.филол.н., доцент каф. англ. языка для ест.спец. Д.Ф.Хакимзянова
к. филол.н., ст.преп. кафедры англ. языка для ест.спец. Ф.Б.Ситдикова
преп. кафедры англ. языка для ест.спец. Р.Н.Сабирова

Научный редактор:
зав.кафедрой англ. языка для естественнонаучных специальностей,
кандидат педагогических наук, доцент
И.Г.Кондратьева

Рецензенты:
к. пед. н., доц., зав.кафедрой англ.языка КНИТУ-КАИ
Н.А.Константинова
к.филол. н., доц. кафедры английского языка Ф.Х.Исмаева

К А З А Н С К И Й Ф Е Д Е Р А Л Ь Н Ы Й У Н И В Е Р С И Т Е Т, 2013

От составителей

Данное учебное пособие предназначено для студентов-магистров Института вычислительной математики и информационных технологий (ВМиИТ-ВМК), имеющих уровень А2/В1, и представляет собой сборник аутентичных текстов с разработанными к ним упражнениями.

Пособие состоит из 17 уроков (юнитов) и нескольких приложений: правил чтения математических символов и формул (с транскрипцией и примерами), выражений для аннотирования статей, текстов для чтения и аннотирования и библиографии.

Тексты, на которых основано пособие, заимствованы из оригинальных источников и охватывают различные области прикладной математики. К ним относятся отрывки из широко известных книг "Operating systems" А.С.Таненбаума и А.С.Вудхалла, "Introduction to Computing" Д.Эванса, “Prime numbers” Р.Крэндалла и С.Померанца, “Introduction to grid computing”, а также тексты из современных Интернет-источников. Это позволит магистрам расширить словарный запас и набрать необходимую профессиональную лексику, а также будет способствовать поддержанию их интереса к изучению английского языка. Пособие рассчитано как для аудиторной, так и для самостоятельной работы.

CONTENTS

Unit 1

Programming and problems with natural languages

I. Before you read answer the following questions:

What do you think is the main difference of the computer from other machines?

What qualities should an ideal programmer have?

Why do you think we cannot use natural languages for writing computer programs?

II. Read the text and compare your answers with the information in the text.

What distinguishes a computer from other machines is its *programmability*. Without a program, a computer is an overpriced door stopper. With the right program, though, a computer can be a tool for communicating across the continent, discovering a new molecule that can cure cancer, composing a symphony, or managing the logistics of a retail empire.

Programming is the act of writing instructions that make the computer do something useful. It is an intensely creative activity, involving aspects of art, engineering and science. Good programs are written to be executed efficiently by computers, but also to be read and understood by humans. The best programs are delightful in ways similar to the best architecture, elegant in both form and function.

The ideal programmer would have the vision of Isaac Newton, the intellect of Albert Einstein, the creativity of Miles Davis, the aesthetic sense of Maya Lin, the wisdom of Benjamin Franklin, the literary talent of William Shakespeare, the oratorical skills of Martin Luther King, the audacity of John Roebling, and the self-confidence of Grace Hopper.

Fortunately, it is not necessary to possess all of those rare qualities to be a good programmer! Indeed, anyone who is able to master the intellectual challenge of learning a language <…> can become a good programmer. <…>

Natural languages <…> are not well suited for human-computer or computer-computer communication. Why can't we use natural languages to program computers? We survey several of the reasons for this. We use specifics from English, although all natural languages suffer from these problems to varying degrees.

Complexity. Although English may seem simple to you now, it took many years of intense effort (most of it subconscious) for you to learn it. Despite using it for most of their waking hours for many years, native English speakers know a small fraction of the entire language. The Oxford English Dictionary contains 615,000 words, of which a typical native English speaker knows about 40,000.

Ambiguity. Not only do natural languages have huge number of words, most words have many different meanings. Understanding the intended meaning of an utterance requires knowing the context, and sometimes pure guesswork.

For example, what does it mean to be paid *biweekly*? According to the American Heritage Dictionary, *biweekly* has two definitions:

1. *Happening every two weeks.*
2. *Happening twice a week, semi-weekly.*

<…> So, depending on which definition is intended, someone who is paid biweekly, could either be paid once or four times every two weeks! The behavior of a pay-roll management program better not depend on how biweekly is interpreted.

Even if we can agree on the definition of every word, the meaning of a sentence is often ambiguous. This particularly difficult example is taken from the instructions with a shipment of ballistic missiles from the British Admiralty:

> *It is necessary for technical reasons that these warheads be stored upside down, that is, with the top at the bottom and the bottom at the top. In order that there be no doubt as to which is the bottom and which is the top, for storage purposes, it will be seen that the bottom of each warhead has been labeled "TOP".*

Irregularity. Because natural languages evolve over time as different cultures interact and speakers misspeak and listeners mishear, natural languages

end up a morass of irregularity. Nearly all grammar rules have exceptions. For example, English has a rule that we can make a word plural by appending an *s*. <…> This rule works for most words. <…> It does not work for *all* words, however. The plural of *goose* is *geese*, the plural of *deer* is *deer*, and the plural of *beer* is controversial <…>. These irregularities can be charming for a natural language, but they are a constant source of problems for non-native speakers, attempting to learn a language. <…>

Uneconomic. It requires a lot of space to express a complex idea in a natural language. Many superfluous words are needed for grammatical correctness, even though they do not contribute to the desired meaning. Since natural languages evolved for everyday communication, they are not well-suited to describe the precise steps and decisions needed in a computer program. As an example, consider a procedure for finding the maximum of two numbers. In English, we could describe it like this:

> *To find the maximum of two numbers, compare them. If the first number is greater than the second number, the maximum is the first number. Otherwise, the maximum is the second number.*

Perhaps shorter descriptions are possible, but any much shorter description probably assumes the reader already knows a lot. By contrast, we can express the same steps in the Scheme programming language in very concise way (don't worry if this doesn't make sense yet — it should by the end of this chapter):

(**define** (bigger a b)(**if** (> ab) ab))

Limited means of abstraction. Natural languages provide small, fixed sets of pronouns to use as means of abstraction, and the rules for binding pronouns to meanings are often unclear. Since programming often involves using simple names to refer to complex things, we need more powerful means of abstraction than natural languages provide.

(http://computingbook.org/Programming.pdf)

III. Close the text and tell whether the following sentences are true or false, correct the false statements:

1. The best use of the computer is being a door stopper.
2. Programming is an extremely creative activity, demanding a lot of qualities from a person.
3. According to the text, no person in the world can become an ideal programmer because the requirements are too high.
4. Anyone who has learnt at least one language can become a good programmer.
5. Natural languages can successfully be used for communicating with the computer.
6. A typical native English speaker knows about one fourth of the whole language stock which is contained in the Oxford English Dictionary.
7. According to the text, a programming language must not be ambiguous.
8. Irregularities are really charming for both natural and programming languages.
9. Natural languages can be well suited to describe the precise steps and decisions needed in a computer program.
10. Programming languages need more powerful means of abstraction than natural languages provide.

IV. Match the words in two columns so that they should form word-combinations from the text.

1. to cure	a. communication
2. well	b. an utterance
3. human-computer	c. sense
4. self-	d. suited
5. meaning of	e. cancer
6. in concise	f. confidence
7. to make	g. way

V. Find derivatives in the text for the following words. Explain their meaning or translate into Russian:

1. to program	*adjective*	*2 nouns*
2. to describe		*noun*
3. regular	*adjective*	*noun*
4. to create	*adjective*	*noun*
5. to suite	*adjective*	
6. audacious		*noun*
7. to utter		*noun*

VI. Match the first half of each sentence with the most appropriate second half

1. What distinguishes a computer from other machines	a. . it is not necessary to possess all of those rare qualities.
2. Programming is the act of writing instructions	b. for human-computer or computer-computer communication.
3. To be a good programmer	c. it took many years of intense effort for you to learn it.
4. Natural languages are not well suited	d. requires knowing the context
5. Although English may seem very simple,	e. is its programmability.
6. Understanding the intended meaning of an utterance	f. to describe the precise steps and decisions needed in a computer program.
7. Natural languages are not well-suited	g. that make the computer do something useful.

VII. Answer the following questions:

1. What can computer do with the right program?
2. Can you explain what programmability is?
3. What is programming and why good programs can be compared with masterpieces of arts?
4. What kind of person can become a good programmer?
5. What percent of the English word stock does an average Englishman know?

6. Can you give your own examples of natural languages ambiguity?

7. What does the irregularity of natural languages mean? Try to explain on yourself.

8. Why are natural languages not able to describe precise steps and decisions needed in a computer program?

9. Can you summarize all reasons why natural languages are not well suited for programming?

VIII. Translate sentences from Russian into English:

1. Компьютерное программирование – это деятельность, результатом которой является последовательность инструкций для компьютера.

2. К счастью, чтобы быть хорошим программистом, не обязательно обладать этими редкими качествами.

3. Существует несколько причин, по которым естественные языки не подходят для общения между человеком и компьютером.

4. Изучение естественного языка может отнять несколько лет интенсивных усилий.

5. Неоднозначность естественных языков также не позволяет использовать их для программирования.

6. К сожалению, во всех естественных языках правила грамматики имеют исключения, которые нужно заучивать.

7. Естественные языки обладают тенденцией к избыточности, что не позволяет назвать их экономичным средством выражения мыслей.

8. Эффективный язык программирования нуждается в средствах абстракции, с помощью которых сложные объекты можно называть и обращаться с ними как с единым целым.

9. Характеристика, которая отличает компьютер от других устройств – это программируемость.

IX. Make up a plan to the text and try to write a short summary of the text, using one sentence for each item of the plan.

Unit 2

Programming Languages Requirements

I. Before you read answer the following questions:

Can you give the definition of a programming language?

What requirements should a well-suited programming language meet?

Are you able to explain the difference between a compiler and an interpreter?

Who is known to develop the first compilers?

II. Read the text and compare your answers with the information in the text.

For programming computers, we want simple, unambiguous, regular, and economical languages with powerful means of abstraction. A *programming language* is a language that is designed to be read and written by humans to create programs that can be executed by computers. Programming languages come in many flavors. It is difficult to simultaneously satisfy all desired properties since simplicity is often at odds with economy. Every feature that is added to a language to increase its expressiveness incurs a cost in reducing simplicity and regularity. For the first two parts of this book, we use the Scheme programming language which was designed primarily for simplicity. For the later parts of the book, we use the Python programming language, which provides more expressiveness but at the cost of some added complexity.

Another reason there are many different programming languages is that they are at different levels of abstraction. Some languages provide programmers with detailed control over machine resources, such as selecting a particular location in memory where a value is stored. Other languages hide most of the details of the machine operation from the programmer, allowing them to focus on higher-level actions.

Ultimately, we want a program the computer can execute. This means at the lowest level we need languages the computer can understand directly. At this level, the program is just a sequence of bits encoding machine instructions. Code at this level is not easy for humans to understand or write, but it is easy for a processor to execute quickly. The machine code encodes instructions that direct the processor to take simple actions like moving data from one place to another, performing simple arithmetic, and jumping around to find the next instruction to execute.

For example, the bit sequence 1110101111111110 encodes an instruction in the Intel x86 instruction set (used on most PCs) that instructs the processor to jump Backwards two locations. Since the instruction itself requires two locations of space, jumping back two locations actually jumps back to the beginning of this instruction. Hence, the processor gets stuck running forever without making any progress.

The computer's processor is designed to execute very simple instructions like jumping, adding two small numbers, or comparing two values. This means each instruction can be executed very quickly. A typical modern processor can execute billions of instructions within a second.[1] The problem with instructions at this level is that they are not easy for humans to write and understand, and you need many simple instructions before you have a useful program.

A *compiler* is a computer program that generates other programs. It translates an input program written in a high-level language that is easier for humans to create into a program in a machine-level language that can be executed by the computer. Admiral Grace Hopper developed the first compilers in the 1950s. An alternative to a compiler is an interpreter. An *interpreter* is a tool that translates between a higher-level language and a lower-level language, but where a compiler translates an entire program at once and produces a machine language

[1] *5A "2GHz processor" executes 2 billion cycles per second. This does not map directly to the number of instructions it can execute in a second, though, since some instructions take several cycles to execute. Until the early 1950s, all programming was done at the level of simple instructions.*

program that can be executed directly, an interpreter interprets the program a small piece at a time while it is running. This has the advantage that we do not have to run a separate tool to compile a program before running it; we can simply enter our program into the interpreter and run it right away. This makes it easy to make small changes to a program and try it again, and to observe the state of our program as it is running.

One disadvantage of using an interpreter instead of a compiler is that because the translation is happening while the program is running, the program executes slower than a compiled program. Another advantage of compilers over interpreters is that since the compiler translates the entire program it can analyze the program for consistency and detect certain types of programming mistakes automatically instead of encountering them when the program is running (or worse, not detecting them at all and producing unintended results). This is especially important when writing critical programs such at flight control software – we want to detect as many problems as problems in the flight control software before the plane is flying!

Since we are more concerned with interactive exploration than with performance and detecting errors early, we use an interpreter instead of a compiler.

(http://computingbook.org/Programming.pdf)

IV. Find the synonyms from the text meaning the same as the words or phrases on the left:

1. having a clear meaning	a.
2. to contradict, not to match smth.	b.
3. first of all	c.
4. finally, eventually, in the long run	d.
5. to implement, to carry out	e.
6. to come to standstill	f.
7. to create, to produce	g.
8. to discover, to reveal, to disclose	h.
9. consequently, therefore	i.

V. Give definitions to the following terms:

1) programming language; 2) program; 3) instruction; 4) compiler; 5) interpreter.

VI. Find derivatives in the text for the following words. Explain their meaning or translate into Russian:

1. economy	*2 adjectives*	*noun*
2. code	*verb*	–
3. consistent	*adjective*	*noun*
4. to intend	*2 adjectives*	*noun*
5. to translate	*2 nouns*	–
6. active	*adjective*	*noun*
7. direct	*adverb*	*noun*

VII. Match the first half of each sentence with the most appropriate second half

1. A programming language is designed to be used by humans	a. just a sequence of bits encoding machine instructions.
2. Different programming languages	b. into machine-level language
3. At the lowest level a program is	c. but it works in a different way from a compiler.
4. A compiler translates an input program written in a high-level language	d. to create programs that can be executed by computers.
5. The first compilers were developed	e. that it can analyze the program for consistency and detect certain types of programming mistakes automatically.
6. An interpreter translates between a higher-level language and a lower-level language,	f. provide different levels of abstraction.
7. An important advantage of compilers over interpreters is	g. by an American computer scientist Grace Hopper whose nick was “Amazing Grace”.

VIII. Answer the following questions:

1. What does a program represents at the lowest level?
2. Why do computers use codes consisting of 1s and 0s?
3. What kind of simple actions can the processor perform?
4. How many instructions a second can a typical modern processor execute?
5. What kind of problem do humans have with instructions at the lowest level?
6. Who invented the first compilers?
7. Do you know any other achievements and inventions by Grace Hopper?
8. What is the difference between a compiler and an interpreter?
9. What are advantages and disadvantages of compilers and interpreters?

IX. Translate sentences from Russian into English:

1. Языки программирования – это искусственно созданные языки, предназначенные для написания компьютерных программ.
2. Все языки программирования делятся на две группы: языки низкого уровня и языки высокого уровня, близкие к человеческому языку.
3. Одним из первых языков программирования стал FORTRAN (от FORmula TRANslator – переводчик формул), разработанный в 1957г.
4. Недостаточно создать язык программирования, для каждого языка нужен свой переводчик. Такими программами являются компиляторы и трансляторы.
5. Компилятор – это программа, предназначенная для перевода программы, написанной на каком-либо языке, в программу в машинных кодах.
6. Интерпретатор – это программа, предназначенная для построчной (line-by-line) трансляции и выполнения исходной программы.
7. Интерпретатор сообщает о найденных им ошибках после трансляции каждой строки программы. Это значительно облегчает процесс поиска и исправления ошибок в программе, однако существенно увеличивает время трансляции.

8. Компилятор транслирует программу намного быстрее, чем интерпретатор, но сообщает о найденных им ошибках после завершения компиляции всей программы. Найти и исправить ошибки в этом случае труднее.

9. Интерпретаторы рассчитаны, в основном, на языки, предназначенные для обучения программированию, и используются начинающими программистами.

10. Большинство современных языков предназначены для разработки сложных пакетов программ и рассчитаны на компиляцию.

X. Make up a plan to the text and try to write a short summary of the text, using one sentence for each item of the plan.

Unit 3
Programming languages classification

I. Before you read answer the following questions:

1. Do you know anything about the types or categories of programming languages?
2. How many generations of programming languages are there nowadays?

II. Read the text and compare your answers with the information in the text.

Programming is a way of sending instructions to the computer. To write these instructions, programmers use programming languages to create source code, and the source code is then converted into machine (or object) code, the only language that a computer understands. People, however, have difficulty understanding machine code. As a result, first assembly languages and then higher-level languages were developed. Programming languages require that information be provided in a certain order and structure, that symbols be used and sometimes even that punctuation be used. These rules are called the syntax of the programming language, and they vary a great deal from one language to another.

Categories of languages. Based on evolutionary history, programming languages fall into one of the following three broad categories:

Machine languages. Machine languages consist of the 0s and 1s of the binary number system and are defined by hardware design. A computer understands only its machine language – the commands in its instruction set that order the computer to perform elementary operations such as loading, storing, adding and subtracting.

Assembly languages. These languages were developed by using Englishlike mnemonics. Programmers worked in text editors to create their source files. To convert the source file into object code, researchers created translator programs called assemblers. Assembly languages are still much easier to use than machine languages.

High-level languages. These languages use syntax that is close to human language, they use familiar words instead of communicating in digits. To express computer operations, they use operators, such as the plus or minus sign, that are the familiar components of mathematics. As a result, reading, writing and understanding computer programs is easier.

Machine languages are considered first-generation languages, and assembly languages are considered second-generation languages. The higher-level languages started with the third generation. Third-generation languages (3GLs) can support structured programming, use true English-like phrasing, and make it easier for programmers to share in the development of program. Besides, they are portable, that is, you can put the source code and a compiler or interpreter on practically every computer can create working object code. Some of the third-generation languages include the following: FORTRAN, COBOL, BASIC, Pascal, C, C++, Java, ActiveX.

Fourth-generation languages (4GLs) use either a text environment, much like a 3GL, or a visual environment. In the text environment the programmer uses English-like words when generating source code. In a 4GL visual environment the programmer uses a toolbar to drag and drop various items like buttons, labels, and

text boxes to create a visual definition of an application. Many 4GLs are database-aware; that is, you can build programs with a 4GL that works as front end (an interface that hides much of the program from the user) to databases. Programmers can also use 4GLs to develop prototypes of an application quickly. Some of the fourth generation languages are Visual Basic and Visual Age.

A 5GL would use artificial intelligence to create software based on your description of what the software should do.

(http://lingualeo.ru/jungle/21386)

III. Close the text and tell whether the following sentences are true or false, correct the false statements:

1. The only language that a computer understands is the source code.
2. The syntax of programming languages varies a great deal from one language to another.
3. Programming languages are divided into 2 broad categories: low-level and high level languages.
4. Machine languages consist of the 0s and 1s and are defined by hardware design.
5. Assembly languages were developed by using Russian-like mnemonics.
6. Assemblers are translator programs created for converting object code into source code.
7. A high-level programming language is characterized by strong abstraction from the details of the computer, making the process of developing a program simpler compared to a lower-level language.
8. The higher-level languages started with the fourth generation.
9. Some of the fourth generation languages are ALGOL and PL1.
10. A 5GL would use artificial intelligence to create software based on user's description of the software tasks.

IV. Match English and Russian equivalents in two columns:

1. portable	ассоциативное запоминание
2. text editor	пользовательский интерфейс
3. text box	компонующая программа, программа сборки
4. mnemonics	визуальная среда
5. assembler	переносимый с одной машины на другую
6. front end	программа редактирования текстов
7. visual environment	текстовое окно, поле текста

V. Match the words in two columns so that they should form word-combinations from the text.

1. source	a. end
2. low-level	b. operations
3. to convert into	c. code
4. translator	d. language
5. to perform	e. environment
6. front	f. machine code
7. visual	g. programs

VI. Complete the gaps in the following sentences:

1. The … code is then converted into machine (or object) code.

2. High level languages were … because people had difficulties in understanding … code.

3. The rules about symbols and punctuation to be used are called the …of the programming language.

4. All programming languages … … one of the following three broad categories.

5. Instruction are commands for the computer to … elementary operations such as loading, storing, adding and subtracting.

6. Assembly languages are … second-generation languages.

7. Third-generation languages can support … programming.

8. Third-generation languages are …, that is, you can put the source code and a compiler or interpreter on every computer.

9. Fourth-generation languages use either a text …, or a visual … .

10. Many 4GLs are database- … .

VII. Answer the following questions:

1. What languages are called first generation languages?
2. What for were assembler programs created?
3. What generation do assembly languages belong to?
4. Can you name any early high-level languages?
5. What features do third-generation languages (3GLs) have?
6. What is special for fourth-generation languages (4GLs)?
7. What would 5GLs use to create software?

VIII. Translate sentences from Russian into English:

1. В развитии языков программирования рассматривают 5 поколений.

2. Языки программирования первого поколения представляли собой набор машинных команд в двоичном коде, который определялся архитектурой конкретной ЭВМ.

3. Второе поколение ЯП характеризуется созданием языков ассемблерного типа , позволяющих вместо команд использовать их мнемонические символьные обозначения.

4. Третье поколение ЯП начинается с появления в 1956 г. первого языка высокого уровня - Fortran, разработанного под руководством Дж. Бэкуса в фирме IBM.

5. Вскоре после языка Fortran появились такие ныне широко известные языки, как Algol, Cobol, Basic, PL/1, Pascal, APL, ADA, C, Forth, Lisp, Modula и др.

6. Основная отличительная особенность языка третьего поколения: ориентирование на алгоритм (алгоритмические языки).

7. С начала 70-х годов по настоящее время продолжается период языков четвертого поколения.

8. Эти языки обычно ориентированы на специализированные области применения, где хороших результатов можно добиться, используя не универсальные, а проблемно-ориентированные языки, оперирующие конкретными понятиями узкой предметной области.

9. Рождение языков пятого поколения произошло в середине 90-х годов. К ним относятся также системы автоматического создания прикладных программ с помощью визуальных средств разработки, без знания программирования.

IX. Make up a plan to the text and try to write a short summary of the text, using one sentence for each item of the plan.

Unit 4

Russell's Paradox.

I. Before you read the text answer the following question:

What is paradox? Give some examples.

Are they useful anyway?

II. Read the statements. Do you think they are true or false?

1. Gottlob Frege tried to create an axiomatic system for all of mathematics built from simple logic.
2. Frege's system was perfect.
3. There are three sensible answers to the paradoxical question.
4. No one tried to resolve the paradox by constructing a system to make it possible to define the set R.

5. Russell and Whitehead published Principia Mathematica to solve a problem with Frege's system.

III. Read the text to check your answers.

Towards the end of the 19th century, many mathematicians sought to systematize mathematics by developing a consistent axiomatic system that is complete for some area of mathematics. One notable attempt was Gottlob Frege's Grundgestze der Arithmetik (1893) which attempted to develop an axiomatic system for all of mathematics built from simple logic.

Bertrand Russell discovered a problem with Frege's system, which is now known as Russell's paradox. Suppose R is defined as the set containing all sets that do not contain themselves as members. For example, the set of all prime numbers does not contain itself as a member, so it is a member of R. On the other hand, the set of all entities that are not prime numbers is a member of R. This set contains all sets, since a set is not a prime number, so it must contain itself.

The paradoxical question is: is the set R a member of R? There are two possible answers to consider but neither makes sense:

Yes: R is a member of R

We defined the set R as the set of all sets that do not contain themselves as member. Hence, R cannot be a member of itself, and the statement that R is a member of R must be false.

No: R is not a member of R

If R is not a member of R, then R does not contain itself and, by definition, must be a member of set R. This is a contradiction, so the statement that R is not a member of R must be false.

The question is a perfectly clear and precise binary question, but neither the "yes" nor the "no" answer makes any sense. Symbolically, we summarize the paradox: for any set s, s ∈ R if and only if s /2 s. Selecting s = R leads to the contradiction: R 2 R if and only if R /2 R.

Whitehead and Russell attempted to resolve this paradox by constructing their system to make it impossible to define the set R. Their solution was to

introduce types. Each set has an associated type, and a set cannot contain members of its own type. The set types are defined recursively:

• A type zero set is a set that contains only non-set objects.

• A type-n set can only contain sets of type n - 1 and below.

This definition avoids the paradox: the definition of R must now define R as a set of type k set containing all sets of type k - 1 and below that do not contain themselves as members. Since R is a type k set, it cannot contain itself, since it cannot contain any type k sets.

In 1913, Whitehead and Russell published Principia Mathematica, a bold attempt to mechanize mathematical reasoning that stretched to over 2000 pages. Whitehead and Russell attempted to derive all true mathematical statements about numbers and sets starting from a set of axioms and formal inference rules. They employed the type restriction to eliminate the particular paradox caused by set inclusion, but it does not eliminate all self-referential paradoxes.

For example, consider this paradox named for the Cretan philosopher Epimenides who was purported to have said "All Cretans are liars". If the statement is true, then Epimenides, a Cretan, is not a liar and the statement that all Cretans are liars is false. Another version is the self-referential sentence: this statement is false. If the statement is true, then it is true that the statement is false (a contradiction). If the statement is false, then it is a true statement (also a contradiction). It was not clear until Gödel, however, if such statements could be stated in the Principia Mathematica system.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011//http://computingbook.org)

IV. Fill in the gaps with the words given:

Avoid, contradiction, be defined, follow, hierarchy, members, significance, states, specify, set theory, variable, ordinals.

1. Cesare Burali-Forti discovered a similar …….. in 1897 when he noticed that since the set of …….. is well- ordered, it too must have an ordinal.

2. Russell's paradox arises within naïve … … by considering the set of all sets that are not …… of themselves.
3. Before a function can ……, one must first ……. exactly those objects to which the function will apply (the function's domain).
4. The ……. Of Russell's paradox can be seen once it is realized that all sentences …… from a contradiction.
5. Comprehension (or Abstraction) axiom in effect …… that any propositional function, P(*x*) containing *x* as a free …… can be used to determine a set.
6. Russell's basic idea is that we can ….. commitment to R (the set of all sets that are not members of themselves) by arranging all sentences (all propositional functions) into …….

V. Match a word in A with its synonym in B.

A. attempt, contradiction, define, member, set, since, select, inclusion.

B. because, choose, characterize, group of items, incorporation, opposition, part, try

VI. Match the words in two columns so that they should form word-combinations from the text.

1. to develop	a. the set of elements
2. to lead to	b. sense
3. to define	c. an axiomatic system
4. to make	d. the type restriction
5. to eliminate	e. the contradiction
6. to employ	f. the paradox

VII. Match the first half of each sentence with the most appropriate second half

1. Many mathematicians sought to systematize mathematics	a. is the set R a member of R?
2. Bertrand Russell discovered a problem with Frege's system,	b. but neither the "yes" nor the "no" answer makes any sense.
3. The paradoxical question is:	c. by developing a consistent axiomatic system
4. The question is a perfectly clear and precise binary question,	d. who stated: "All Cretans are liars".
5. Whitehead and Russell attempted to resolve this paradox	e. which is now known as Russell's paradox
6. Consider this paradox named for the Cretan philosopher Epimenides	f. by constructing their system to make it impossible to define the set R.

VIII. Translate from Russian into Eglish:

1. Самым знаменитым из открытых в нашем веке парадоксов является парадокс Рассела, обнаруженный Б. Расселом и описанный им в письме к Г. Ферге.

2. Пусть R – множество всех множеств, которые не содержат себя в качестве своего элемента. Содержит ли оно само себя в качестве элемента? Если предположить, что содержит, то мы получаем противоречие с "не содержат себя в качестве своего элемента". Если предположить, что не содержит себя, как элемент, то вновь возникает противоречие, ведь — множество всех множеств, которые не содержат себя в качестве своего элемента, а значит, должно содержать все возможные элементы, включая и себя.

3. Существует много популярных формулировок парадокса Рассела. Наиболее древняя из них приписывается критскому философу Эпимениду и звучит следующим образом: "Критянин сказал, что все критяне лжецы". Сказал ли он правду?

IX. Make up a plan to the text and try to write a short summary of the text, using one sentence for each item of the plan.

X. Use additional resources. Find an example of a paradox, present and consider it.

Unit 5

Coin-flip protocol

I. Look at the title of the text. What are you going to read about?

II. Read the text and answer the following questions:

1. What is a protocol?
2. Should the coin be tossed doubly?
3. How do Alice and Bob flip a fair coin?
4. Are there any variants of coin-flip protocol?

In cryptography, a protocol is essentially an algorithm specifying—in a certain order—the steps that involved parties must take. We have seen key-exchange and related protocols already. Here we investigate an intriguing cultural application of number-theoretical protocols. How can one toss a coin, fairly, over the telephone? Or play poker among n individuals, playing "blind" on a network? We assume the worst: That no party trusts any other, yet a decision has to be reached, as one would so reach it via a coin toss, with one party calling heads or tails. It turns out that such a remote tossing is indeed possible, using properties of certain congruencies.

Incidentally, the motivation for even having a coin-flip protocol is obvious, when one imagines a telephone conversation—say between two hostile parties involved in a lawsuit—in which some important result accrues on the basis of a coin flip, meaning a random bit whose statistics cannot be biased by either party. Having one party claim they just flipped a head, and therefore won the toss, is clearly not good enough. Everyone must be kept honest, and this can be done via adroit application of congruences involving primes or certain composites. Here is one way to proceed, where we have adapted some ideas from [Bressoud and Wagon 2000] on simple protocols:

Algorithm 8.1.11 (Coin-flip protocol). Alice and Bob wish to "flip a fair coin," using only a communication channel. They have agreed that if Bob guesses correctly, below, then Bob wins, otherwise Alice wins.

1. [Alice selects primes]

Alice chooses two large primes p < q, forms the number n = pq, and chooses a random prime r such that (n/r)= −1;

2. [Alice sends Bob partial information]

Alice sends Bob n and r;

3. [Bob chooses]

Bob makes a choice between "the smaller prime factor of n is a quadratic residue mod r" and "the larger prime factor of n is a quadratic residue mod r" and sends this choice to Alice;

4. [Alice announces winner]

Alice announces whether Bob is correct or not, and sends him the primes p, q so that Bob can see for himself that she is not cheating;

It is interesting to investigate the cryptographic integrity of this algorithm. Though we have cast the above algorithm in terms of winner and loser, it is clear that Alice and Bob could use the same method just to establish a random bit, say "0" if Alice wins and "1" if Bob wins. There are many variants to this kind of coin-flip protocol. For example, there is a protocol in [Schneier 1996] in which four square roots of a number $n = pq$ are generated by Alice and sent to Bob, with Bob having generated a random square modulo n. This scenario is not as simple as Algorithm 8.1.11, but it is replete with interesting issues; e.g., one can extend it to handle the peculiar Micali scenario in which Bob intentionally loses [Schroeder 1999]. There are also algorithms based on Blum integers and, generally, the fact of a product pq allowing multiple roots (see Exercise 8.7). These ideas can be extended in a natural way to a poker-playing protocol in which a number of players claim what poker hands they possess, and so on [Goldwasser and Micali 1982].

(Crandall R., Pomerance C. Prime Numbers/A Computational Perspective.-Springer Science+ Business Media, Inc. 2005,-598pp)

III. In pairs, look at the given words. Try to guess what they mean from the context. Then check with your dictionary or the teacher.

Hostile parties, toss a coin, play ‘blind’, random bit, to be biased, to flip a head, to make a choice

IV. Match the words in A with their synonyms in B.

A. accrue, adroit, announce, issue, random, remote, to be replete with

B. accidental, compile, declare, distant, dexterous, to be full with, problem

V. Match the phrases (1-6) with their equivalents (a-f).

1.	Another lay treatment about	**a.**	В последней работе
2.	Answer roughly the question	**b.**	Другое описание о…
3.	The answer is relevant	**c.**	Приблизительно ответить на вопрос
4.	In the latter exposition	**d.**	Стандартный пример
5.	To be not uncommon	**e.**	Ответ на этот важный вопрос
6.	A typical instance	**f.**	Являться широкодоступным

VI. Make the sentences of your own using the phrases from exercise V.

VII. Make written translation of the second paragraph.

VIII. Translate from Russian into English.

1. Протоколом является алгоритм.
2. Алгоритм указывает в определенном порядке те шаги, которые нужно предпринять.
3. Необходимо контролировать честность каждого участника.
4. Боб делает выбор и отправляет его Алисе.

5. Протокол подбрасывания монет имеет множество вариантов.

IX. Find 3-4 key words in every paragraph. Write down the sentences of your own using these words. They should give the main idea of each paragraph.

X. Make up a plan of the text and summarize the text in brief.

Unit 6

Random-number generation

I. Before you read the text look at the words given and try to predict what the text is going to be about:

- Randomness
- Probability theory
- Deterministic generator
- To decrypt
- Unbreakable

II. Read the text to check your ideas.

The problem of generating random numbers goes back, of course, to the dawn (1940s, say) of the computer age. It has been said that to generate random numbers via machine arithmetic is to live, in the words of J. von Neumann, "in a state of sin." Though machines can ensure nearly random statistics in many senses, there is the problem that conventional machine computation is deterministic, so the very notion of randomness is suspect in the world of Turing machines and serial programs. If the reader wonders what kind of technology could do better in the matter of randomness (though still not "purely" random in the sense of probability theory), here is one exotic example: Aim a microwave receiving dish at the remote heavens, listening to the black-body "fossil" radiation from the early cosmos, and

digitize that signal to create a random bitstream. We are not claiming the cosmos is truly "random," but one does expect that a signal from remote regions is as "unknowable" as can be.

In modern times, the question of true randomness has more import than ever, as cryptographic systems in particular often require numbers that are as random, or as seemingly random, as can be. A deterministic generator that generates what looks to an eavesdropper like random numbers can be used to build a simple cryptosystem. Create a random bitstream. To encrypt a message, take the logical exclusive- or of bits of the message with bits of the random bitstream. To decrypt, do the exclusive-or operation again, against the same random bitstream. This cryptosystem is unbreakable, unless certain weaknesses are present—such as, the message is longer than the random stream, or the same random stream is reused on other messages, or the eavesdropper has special knowledge of the generator, and so on. In spite of such practical pitfalls, the scheme illustrates a fundamental credo of cryptography: Somehow, use something an eavesdropper does not know.

It seems that just as often as a new random-number generator is developed, so, too, is some older scheme shown to be nonrandom enough to be, say, "insecure," or yield misleading results in Monte Carlo simulations. We shall give a brief tour of random number generation, with a view, as usual, to the involvement of prime numbers.

(Crandall R., Pomerance C. Prime Numbers/A Computational Perspective.-Springer Science+ Business Media, Inc. 2005,-598pp)

III. Say whether these sentences are true or false. Try not to refer to the text.

1. The problem of random-number generation is a new one.
2. To generate random number with the help of machine arithmetic is to live in heavens.
3. Nowadays the question of true randomness is insignificant.
4. A deterministic generator cannot be used to build a simple crypto-system.

5. To encrypt and to decrypt you have to do the same operations.

IV. Explain the meaning of the words given or find synonyms. Consult your dictionary or teacher.

Generate, ensure, conventional, suspect, digitize, unknowable, eavesdropper, decrypt, pitfall, illustrate, microwave receiving dish, fossil radiation, nonrandom.

V. Match the phrases (1-6) with their equivalents (a-f).

1.	To put it roughly...	**A.**	Помимо современных достижений
2.	The relevance is…	**B.**	Хотя современные технологии
3.	Similarly …	**C.**	Грубо говоря
4.	It is amusing …	**D.**	Существенным моментом является
5.	While modern technology…	**E.**	Аналогичным образом…
6.	Alongside these modern achievements…	**F.**	Удивительно, что…

VI. Translate from Russian into English.

1. Компьютеры в некотором смысле могут гарантировать почти случайное распределение.
2. Известные способы компьютерных вычислений обладают детерминированностью.
3. Мы предполагаем, что сигнал, излучаемый отдаленными галактиками настолько непредсказуем, насколько это возможно.
4. Сегодня моделирование реальной случайности важно как никогда.
5. Такая криптосистема в принципе защищена от взлома.

Unit 7

Numbers

Text 1

I. Skim the text quickly. What is the main idea of this text?

We have indicated that prime numbers figure into modern cryptography—the science of encrypting and decrypting secret messages. Because many cryptographic systems depend on prime-number studies, factoring, and related number-theoretical problems, technological and algorithmic advancement have become paramount. Our ability to uncover large primes and prove them prime has outstripped our ability to factor, a situation that gives some comfort to cryptographers. As of this writing, the largest number ever to have been proved prime is the gargantuan Mersenne prime $225964951 - 1$, which can be thought of, roughly speaking, as a "thick book" full of decimal digits. The kinds of algorithms that make it possible to do speedy arithmetic with such giant numbers is discussed in Chapter 8.8. But again, alongside such algorithmic enhancements come machine improvements. To convey an idea of scale, the current hardware and algorithm marriage that found each of the most recent "largest known primes" performed thus: The primality proof/disproof for a single candidate $2q - 1$ required in 2004 about one CPUweek, on a typical modern PC (see continually updating website [Woltman 2000]). By contrast, a number of order 220000000 would have required, just a decade earlier, perhaps a decade of a typical PC's CPU time! Of course, both machine and algorithm advances are responsible for this performance offset. To convey again an idea of scale: At the start of the 21st century, a typical workstation equipped with the right software can multiply together two numbers, each with a million decimal digits, in a fraction of a second. As explained at the end of Section 9.5.2, appropriate cluster hardware can now multiply two numbers each of a billion digits in roughly one minute.

(Crandall R., Pomerance C. Prime Numbers/A Computational Perspective.-Springer Science+ Business Media, Inc. 2005,-598pp)

II. Write down a short summary of the text (5-6 sentences).

III. Give the title to the text.

IV. Make a written translation of the text.

Text 2. Smooth numbers

I. Before you read the text answer the following questions.

What kind of numbers do you know? Give examples.

What are smooth numbers?

II. Read the text to check your answers.

Smooth numbers are extremely important for our computational interests, notably in factoring tasks. And there are some fascinating theoretical applications of smooth numbers, just one example being applications to a celebrated problem upon which we just touched, namely the Waring problem [Vaughan 1989]. We begin with a fundamental definition:

Definition 1.4.8. A positive integer is said to be y-smooth if it does not have any prime factor exceeding y.

What is behind the usefulness of smooth numbers? Basically, it is that for y not too large, the y-smooth numbers have a simple multiplicative structure, yet they are surprisingly numerous. For example, though only a vanishingly small fraction of the primes in [1, x] are in the interval [1,√ x], nevertheless more than 30% of the numbers in [1, x] are √ x-smooth (for x sufficiently large). Another example illustrating this surprisingly high frequency of smooth numbers: The number of (ln2 x)-smooth numbers up to x exceeds √ x for all sufficiently large numbers x. These examples suggest that it is interesting to study the counting function for smooth numbers. Let

$$\psi(x, y) = \#\{1 \le n \le x : n \text{ is } y\text{-smooth}\}. \qquad (1.42)$$

Part of the basic landscape is the Dickman theorem from 1930:

Theorem 1.4.9 (Dickman). For each fixed real number u > 0, there is a real number ρ(u) > 0 such that

$$\psi(x, x1/u) \sim \rho(u)x.$$

Moreover, Dickman described the function ρ(u) as the solution of a certain differential equation: It is the unique continuous function on [0,∞) that satisfies (A) (u) = 1 for 0 ≤ u ≤ 1 and (B) for u > 1, ρ′(u) = −ρ(u − 1)/u. In particular, ρ(u) = 1−ln u for 1 ≤ u ≤ 2, but there is no known closed form (using elementary functions) for ρ(u) for u > 2. The function ρ(u) can be approximated numerically (cf. Exercise 3.5), and it becomes quickly evident that it decays to zero rapidly. In fact, it decays somewhat faster than u−u, though this simple expression can stand in as a reasonable estimate for ρ(u)in various complexity studies. Indeed, we have

$$\ln \rho(u) \sim -u \ln u. \qquad (1.43)$$

Theorem 1.4.9 is fine for estimating ψ(x, y) when x, y tend to infinity with u = lnx/ ln y fixed or bounded. But how can we estimate ψ x, x1/ ln ln x or ψ x, e √ ln x or ψ x, ln2 x? Estimates for these and similar expressions became crucial around 1980 when subexponential factoring algorithms were first being studied theoretically (see Chapter 6). Filling this gap, it was shown in [Canfield et al. 1983] that

$$\psi(x, x1/u) = xu - u + o(u) \qquad (1.44)$$

uniformly as $u \to \infty$ and u < (1−ϵ) ln*x*/ ln ln *x*. Note that this is the expected estimate, since by (1.43) we have that $\rho(u) = u - u + o(u)$. Thus we have a reasonable estimate for $\psi(x,y)$ when $y > \ln 1 + x$ and *x* is large.

It is also possible to prove explicit inequalities for $\psi(x, y)$. For example, in [Konyagin and Pomerance 1997] it is shown that for all x ≥ 4 and 2 ≤ x1/u ≤ x,

$$\psi x, x1/u \geq x \ln u x. \qquad (1.45)$$

The implicit estimate here is reasonably good when x1/u = lnc x, with c > 1 fixed.

As mentioned above, smooth numbers arise in various factoring algorithms, and in this context they are discussed later in this book. The computational problem of recognizing the smooth numbers in a given set of integers is discussed

in Chapter 3. For much more on smooth numbers see the new survey article [Granville 2004b].

(Crandall R., Pomerance C.Prime Numbers/A Computational Perspective.-Springer Science+ Business Media, Inc. 2005,-598pp)

III. Give Russian equivalents to the given phrases.

Extremely important, multiplicative structure, vanishingly small, decay to zero, tend to infinity, reasonable estimate, factoring algorithms.

IV. Fill in the gaps with the words given.

1. Computational number theory abounds with examples of sequences *N* of integers from which we need to extract y-smooth numbers.
2. These hypotheses appear to be fairly ad hoc, tied in to the algorithms.
3. Such a range is certainly dependent on the coefficients of *f*, but we need results in which this dependence is simply stated and easily applicable.
4. There are very few results in the literature with precise inequalities where every constant is explicit.
5. In general, if we wish to determine the number of lattice points inside an *n*-dimensional tetrahedron, then we can get good estimates using a little geometry.

V. Write down a short summary of the text (5-6 sentences).

Unit 8

Quasi-Monte Carlo (qMC) methods

I. Read the quotation and comment on it.

Nothing in Nature is random… A thing appears random only through the incompleteness of our knowledge.

Spinoza, Ethics I

II. Before you read the text say what you know about Monte Carlo methods.

Who would have guessed, back in the times of Gauss, Euler, Legendre, say, that primes would attain some practical value in the financial-market analysis of the latter twentieth century? We refer here not to cryptographic uses— which certainly do emerge whenever money is involved—but quasi-Monte Carlo science which, loosely speaking, is a specific form of Monte Carlo (i.e., statistically motivated) analysis. Monte Carlo calculations pervade the fields of applied science.

The essential idea behind Monte Carlo calculation is to sample some large continuous (or even discrete, if need be) space—in doing a multidimensional integral, say—with random samples. Then one hopes that the "average" result is close to the true result one would obtain with the uncountable samples theoretically at hand. It is intriguing that number theory—in particular prime number study—can be brought to bear on the science of quasi-Monte Carlo (qMC). The techniques of qMC differ from traditional Monte Carlo in that one does not seek expressly random sequences of samples. Instead, one attempts to provide quasirandom sequences that do not, in fact, obey the strict statistical rules of randomness, but instead have certain uniformity features attendant on the problem at hand.

Although it is perhaps overly simplistic, a clear way to envision the difference between random and qMC is this: Random points when dropped can be expected to exhibit "clumps" and "gaps," whereas qMC points generally avoid each other to minimize clumping and tend to occupy previous gaps. For these reasons qMC points can be—depending on the spatial dimension and precise posing of the problem—superior for certain tasks such as numerical integration, min–max problems, and statistical estimation in general.

(Crandall R., Pomerance C.Prime Numbers/A Computational Perspective.-Springer Science+ Business Media, Inc. 2005,-598pp)

III. Find 3-4 key words in every paragraph. Write down the sentences of your own using these words. They should give the main idea of each paragraph.

IV. Make up a plan of the text and summarize the text in brief.

V. Fill in the gaps with the words given.

Analysis, applicability, applied, computer age, definition, numerical method, practical implementation, random.

1. The Monte Carlo method (or "method of statistical trials") may be described in simple terms as a …….. based on random sampling.
2. Statisticians have intuitively used its principle long before 1940s, but it was only the …… that could turn into a systematic and viable technique.
3. The main reason for the popularity of the Monte Carlo method is its …… to a never ending variety of problems in numerical analysis, statistics, …… mathematics, particle physics, engineering, systems ……., and so on.
4. For the ……. ……. of the Monte Carlo method, the fundamental question is, of course, how to produce a ……. sample.
5. There is no ready- made answer since no satisfactory ……. of randomness exists.

VI. Make a written translation of the sentences in exercise V.

Unit 9.
Operating systems. Introduction.

I. Before you read the text answer the following questions:

What is an operating system, what is its main function?

What are the most popular kinds of operating systems nowadays?

Can you name their advantages and disadvantages?

II. Read the text and compare your answers with the information in the text.

Without its software, a computer is basically a useless lump of metal. With its software, a computer can store, process, and retrieve information; play music and videos; send e-mail, search the Internet; and engage in many other valuable activities to earn its keep. Computer software can be divided roughly into two kinds: system programs, which manage the operation of the computer itself, and application programs, which perform the actual work the user wants. The most fundamental system program is the **operating system**, whose job is to control all the computer's resources and provide a base upon which the application programs can be written. <…>

A modern computer system consists of one or more processors, some main memory, disks, printers, a keyboard, a display, network interfaces, and other input/output devices.<…> Many years ago it became abundantly clear that some way had to be found to shield programmers from the complexity of the hardware. The way that has evolved gradually is to put a layer of software on top of the bare hardware, to manage all parts of the system, and present the user with an interface or **virtual machine** that is easier to understand and program. This layer of software is the operating system.

The placement of the operating system is shown in Fig. 1-1. At the bottom is the hardware, which, in many cases, is itself composed of two or more levels (or layers). The lowest level contains physical devices, consisting of integrated circuit chips, wires, power supplies, cathode ray tubes, and similar physical devices. How these are constructed and how they work is the province of the electrical engineer.

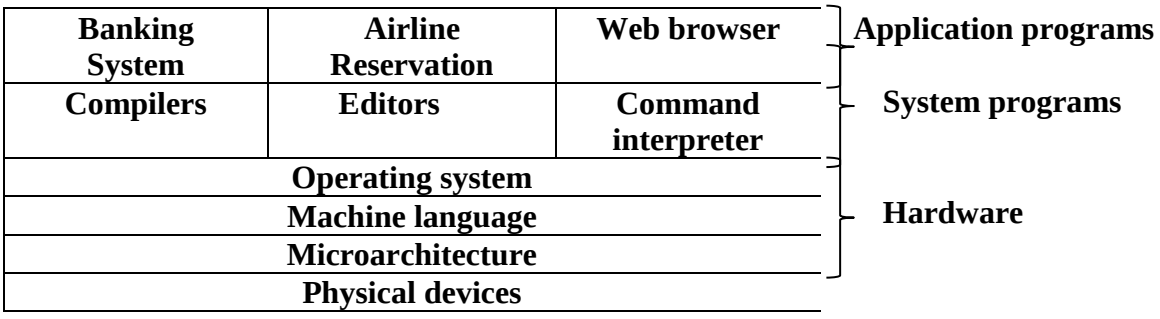


Fig. 1-1. *A computer system consists of hardware, system programs, and application programs.*

Next comes the **microarchitecture level**, in which the physical devices are grouped together to form functional units. Typically this level contains some registers internal to the CPU (Central Processing Unit) and a data path containing an arithmetic logic unit. In each clock cycle, one or two operands are fetched from the registers and combined in the arithmetic logic unit (for example, by addition or Boolean AND). The result is stored in one or more registers. On some machines, the operation of the data path is controlled by software, called the **microprogram**. On other machines, it is controlled directly by hardware circuits.

The purpose of the data path is to execute some set of instructions. Some of these can be carried out in one data path cycle; others may require multiple data path cycles. These instructions may use registers or other hardware facilities. Together, the hardware and instructions visible to an assembly language programmer form the **ISA (Instruction Set Architecture)** This level is often called **machine language**.

The machine language typically has between 50 and 300 instructions, mostly for moving data around the machine, doing arithmetic, and comparing values. In this level, the input/output devices are controlled by loading values into special **device registers.** For example, a disk can be commanded to read by loading the values of the disk address, main memory address, byte count, and direction (read or write) into its registers. In practice, many more parameters are needed, and the status returned by the drive after an operation may be complex. Furthermore, for many I/O (Input/Output) devices, timing plays an important role in the programming.

The major function of the operating system is to hide all this complexity and give the programmer a more convenient set of instructions to work with. For example, *read block from file* is conceptually much simpler than having to worry about the details of moving disk heads, waiting for them to settle down, and so on.

On top of the operating system is the rest of the system software. Here we find the command interpreter (shell), window systems, compilers, editors, and similar application independent programs. It is important to realize that these programs are definitely not part of the operating system, even though they are typically supplied preinstalled by the computer manufacturer, or in a package with the operating system if it is installed after purchase. This is a crucial, but subtle, point. The operating system is (usually) that portion of the software that runs in **kernel mode** or **supervisor mode**. It is protected from user tampering by the hardware (ignoring for the moment some older or low-end microprocessors that do not have hardware protection at all). Compilers and editors run in **user mode**. If a user does not like a particular compiler, he is free to write his own if he so chooses; he is not free to write his own clock interrupt handler, which is part of the operating system and is normally protected by hardware against attempts by users to modify it.

This distinction, however, is sometimes blurred in embedded systems (which may not have kernel mode) or interpreted systems (such as Java-based systems that use interpretation, not hardware, to separate the components). Still, for traditional computers, the operating system is what runs in kernel mode.

<…> Finally, above the system programs come the application programs. These programs are purchased (or written by) the users to solve their particular problems, such as word processing, spreadsheets, engineering calculations, or storing information in a database.

(Modern Operating Systems, Second edition by A.S.Tanenbaum p.28-30)
http://it.tdt.edu.vn/~tttin/giangday/HDH/Modern%20Operating%20Systems.pdf

III. Match the terms and their definitions:

1. hardware	a. software that communicates with hardware enabling the applications to run
2. input/output	b. program-addressable storage that is directly controlled

device	by and generally contained in the CPU
3. operating system	c. a program used for particular application, not a system program
4. main memory	d. mechanical, magnetic and electronical devices comprising a computer system
5. application program	e. any of various devices used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a human operator
6. machine language	f. programs that are used to direct the operation of a computer, as well as documentation giving instructions on how to use them
7. software	g. a coding system built into the hardware of a computer, requiring no translation before being run

IV. Find derivatives for the following words. Explain their meaning or translate into Russian:

1. to instruct	*adjective*	*2 nouns*
2. editor	*verb*	*noun*
3. to apply	–	*2 nouns*
4. to define	*adverb*	*noun*
5. bed	*adjective*	verb
6. to process	*2 nouns*	–
7. concept	*adjective*	*adverb*

V. Complete the gaps in the following sentences:

1. Computer … can be divided roughly into two kinds: system programs, which manage the operation of the computer itself, and … programs.

2. A modern computer system consists of one or more …, some main memory, disks, printers, a keyboard, a display, network interfaces, and other … devices.

3. The lowest level contains … devices, consisting of integrated circuit chips, wires, power supplies and similar physical devices.

4. In the … level the physical devices are grouped together to form functional units.

5. Together, the hardware and instructions visible to an assembly language programmer form the Instruction … Architecture.

6. The major function of the … system is to hide all this complexity and give the programmer a more convenient set of instructions to work with.

7. The operating system is usually that portion of the software that runs in … mode or … mode.

8. … programs are purchased or written by users to solve their particular problems, such as word processing, spreadsheets, engineering calculations, or storing information in a database.

VI. Match the first half of each sentence with the most appropriate second half

1. Without its software,	a. is the province of the electrical engineer.
2. The most fundamental system program is the operating system	b. from the registers and combined in the arithmetic logic unit.
3. A computer can store, process, and retrieve information; play music and videos; send e-mail, search the Internet	c. and similar application independent programs are definitely not part of the operating system.
4. How physical devices are constructed and how they work	d. a computer is basically a useless lump of metal.
5. In each clock cycle, one or two operands are fetched	e. whose job is to control all the computer's resources and provide a base upon which the application programs can be written.
6. The machine language typically has between 50 and 300 instructions, mostly for	f. due to its software.
7. It is important to realize that such programs as the command interpreter, window systems, compilers, editors,	g. moving data around the machine, doing arithmetic, and comparing values.

VII. Answer the following questions:

1. What kinds of things can computer do with its software?
2. What two main parts can all computer software divided into?
3. What is the main function of the operating system?
4. How many layers are in the diagram of the computer system offered in the text?
5. What is the lowest layer of a computer system?
6. What is the purpose of the data path?
7. How many instructions does the machine language typically have?
8. Can we consider the command interpreter (shell), window systems, compilers, editors, and similar programs as a part of the operating system?.
9. Which layer comes above the system programs?

VIII. Translate sentences from Russian into English:

1. Все программное обеспечение принято делить на две части: *прикладное и системное*.
2. Под системным программным обеспечением обычно понимают программы, способствующие функционированию и разработке прикладных программ.
3. Уже много лет назад стало ясно, что нужно было найти какой-то выход, чтобы оградить программиста от сложностей аппаратного обеспечения.
4. Операцио́нная систе́ма – это комплекс программ, которые, с одной стороны, выступают как интерфейс между устройствами вычислительной системы и прикладными программами, а с другой стороны — предназначены для управления устройствами, вычислительными процессами, а также для эффективного распределения вычислительных ресурсов.
5. С 1990-х годов наиболее распространёнными операционными системами являются системы семейства Windows и системы класса UNIX (особенно Linux и Mac OS).
7. С точки зрения (in terms of longevity) долголетия ни одна операционная система для микрокомпьютеров не может приблизиться к DOS.

8. С момента появления в 1981 году DOS распространилась настолько широко, что завоевала право считаться самой популярной в мире ОС.

9. Несмотря на некоторые свои недостатки, DOS продолжает существовать и развиваться.

10. Всего за несколько лет система MS DOS прошла путь от простого загрузчика до универсальной сложившейся операционной системы для персональных компьютеров, построенных на базе микропроцессоров Intel 8086.

IX. Make up a plan to the text and try to write a short summary of the text, using one sentence for each item of the plan.

Unit 10.
The Operating System Functions

I. Before you read answer the following questions:

1. Without having a look at the text, can you name two most important functions of the operating system?

2. Can you agree with the statement that operating system is a kind of bridge between the programmer (user) and the computer? If not, what other comparisons can you think of?

II. Read the text and compare your answers with the information in the text.

Most computer users have had some experience with an operating system, but it is difficult to pin down precisely what an operating system is. Part of the problem is that operating systems perform two basically unrelated functions, extending the machine and managing resources. Let us now look at both.

The Operating System as an Extended Machine. As mentioned earlier, the **architecture** (instruction set, memory organization, I/O, and bus structure) of most computers at the machine language level is primitive and awkward to program, especially for input/output. <…>

Without going into the real details, it should be clear that the average programmer probably does not want to get too intimately involved with the programming of floppy disks (or hard disks, which are just as complex and quite different). Instead, what the programmer wants is a simple, high level abstraction to deal with. In the case of disks, a typical abstraction would be that the disk contains a collection of named files. Each file can be opened for reading or writing, then read or written, and finally closed. Details such as whether or not recording should use modified frequency modulation and what the current state of the motor is should not appear in the abstraction presented to the user.

The program that hides the truth about the hardware from the programmer and presents a nice, simple view of named files that can be read and written is, of course, the operating system. Just as the operating system shields the programmer from the disk hardware and presents a simple file-oriented interface, it also conceals a lot of unpleasant business concerning interrupts, timers, memory management, and other low-level features. In each case, the abstraction offered by the operating system is simpler and easier to use than that offered by the underlying hardware.

In this view, the function of the operating system is to present the user with the equivalent of an **extended machine** or **virtual machine** that is easier to program than the underlying hardware. How the operating system achieves this goal is a long story, which we will study in detail throughout this book. To summarize it in a nutshell, the operating system provides a variety of services that programs can obtain using special instructions called system calls.

The Operating System as a Resource Manager. Modern computers consist of processors, memories, timers, disks, mice, network interfaces, printers, and a wide variety of other devices. The job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs competing for them.

Imagine what would happen if three programs running on some computer all tried to print their output simultaneously on the same printer. The first few lines of

printout might be from program 1, the next few from program 2, then some from program 3, and so forth. The result would be chaos. The operating system can bring order to the potential chaos by buffering all the output destined for the printer on the disk. When one program is finished, the operating system can then copy its output from the disk file where it has been stored to the printer, while at the same time the other program can continue generating more output, oblivious to the fact that the output is not really going to the printer (yet).

When a computer (or network) has multiple users, the need for managing and protecting the memory, I/O devices, and other resources is even greater, since the users might otherwise interfere with one another. In addition, users often need to share not only hardware, but information (files, databases, etc.) as well. In short, this view of the operating system holds that its primary task is to keep track of who is using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

Resource management includes multiplexing (sharing) resources in two ways: in time and in space. When a resource is time multiplexed, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on. For example, with only one CPU and multiple programs that want to run on it, the operating system first allocates the CPU to one program, then after it has run long enough, another one gets to use the CPU, then another, and then eventually the first one again. Determining how the resource is time multiplexed who goes next and for how long is the task of the operating system. Another example of time multiplexing is sharing the printer. When multiple print jobs are queued up for printing on a single printer, a decision has to be made about which one is to be printed next.

The other kind of multiplexing is space multiplexing. Instead of the customers taking turns, each one gets part of the resource. For example, main memory is normally divided up among several running programs, so each one can be resident at the same time (for example, in order to take turns using the CPU). Assuming there is enough memory to hold multiple programs, it is more efficient

to hold several programs in memory at once rather than give one of them all of it, especially if it only needs a small fraction of the total. Of course, this raises issues of fairness, protection, and so on, and it is up to the operating system to solve them. Another resource that is space multiplexed is the (hard) disk. In many systems a single disk can hold files from many users at the same time. Allocating disk space and keeping track of who is using which disk blocks is a typical operating system resource management task.

(Modern Operating Systems, Second edition by A.S.Tanenbaum p.31-33)
http://it.tdt.edu.vn/~tttin/giangday/HDH/Modern%20Operating%20Systems.pdf

III. Close the text and tell whether the following sentences are true or false, correct the false statements:

1. Every user deals with the operating system when working at their computer.
2. All programmers are eager to get intimately involved into all details of the computer hardware.
3. The operating system shields the programmer from the programming work itself.
4. The operating system offers a variety of services that can be obtained using special instructions called system requests.
5. It is the operation system that is responsible for all input/output operations.
6. Multiple users never interfere with one another when they share a network.
7. Determining how the resource is time multiplexed who goes next and for how long is the task of the system engineer.
8. When several programs are held in memory at once, it is up to the operating system to solve issues of fairness, protection, and so on.
9. In a typical system a single disk can hold files from only one user at the same time.

IV. Match the terms and their definitions:

1. processor	a. a disk made of rigid magnetizable material that is

(or CPU)	used to store data for computers
2. bus	b. a collection of related data or program records stored on some input/output or auxiliary medium
3. hard disk	c. an output device that produces a paper copy of alphanumeric or graphic data
4. file	d. the part of a computer that performs logical and arithmetical operations on the data as specified in the instructions
5. memory resident	e. a communication system that transfers data between components inside a computer or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocol.
6. printer	f. a kind op programs that the operating system is not permitted to swap out to a storage device, they will always remain in memory

V. Find derivatives for the following words. Explain their meaning or translate into Russian:

1. precise	*adverb*	*noun*
2. to relate	*2 adjectives*	*noun*
3. to modulate	*adjective*	*noun*
4. to compete	*adjective*	*2 nouns*
5. to manage	*adjective*	*2 nouns*
6. to protect	*adjective*	*2 nouns*
7. to allocate	*adjective*	*2 nouns*

VI. Complete the gaps in the following sentences:

1. It is difficult to … … precisely what an operating system is.

2. The operating system performs two basically … functions: extending the machine and managing resources.

3. The program that hides the truth about the … from the programmer and presents a nice, simple view of named files that can be read and written is, of course, the operating system.

4. The function of the operating system is to present the user with the equivalent of an … machine or … machine that is easier to program than the underlying hardware.

5. The job of the operating system is to provide for an orderly and controlled … of the processors, memories, and I/O devices among the various programs … for them.

6. When a computer (or network) has … users, the need for managing and protecting the memory, I/O devices, and other resources is even greater

7. When a resource is time … , different programs or users take turns using it.

8. The other kind of multiplexing is … multiplexing.

9. For example, main memory is normally divided up among several running programs, so each one can be … at the same time

VII. Match the first half of each sentence with the most appropriate second half

1.The architecture of most computers at the machine language level is	a. simple, high level abstraction to deal with.
2. The average programmer probably does not want to get too intimately	b. using special instructions called ystem calls.
3. What the programmer wants is a	c. primitive and awkward to program.
4. The operating system shields the programmer from the disk	d. (files, databases, etc.) as well.
5. The operating system provides a variety of services that programs can obtain	e. in two ways: in time and in space.
6. Users often need to share not only hardware, but information	f. involved with the programming of hardware details.

VIII. Answer the following questions:

1. Why is it difficult to pin down precisely what an operating system is?

2. Why does the operating system hide the truth about the hardware from the programmer?

3. What kind of physical devices do modern computers systems consist of?

4. What is the second function of the operating system?

5. What would happen if three programs running on some computer all tried to print their output simultaneously on the same printer?

6. What information does the operating system need to keep track of when the computer (or network) has multiple users?

7. What is space multiplexing?

8. What kind of problems might arise when there are several programs in memory at once?

9. Can you give an example of a typical operating system resource management task?

IX. Translate sentences from Russian into English:

1. Операционная система – это комплекс системных программ, которые предназначены организовать взаимодействие пользователя с компьютером и выполнение всех других программ.

2. Первая задача ОС – организация связи, общения пользователя с компьютером в целом и его отдельными устройствами.

3. Операционная система представляется пользователю виртуальной машиной, с которой намного проще иметь дело, чем непосредственно с оборудованием компьютера.

4. Операционная система действует как менеджер ресурсов, осуществляет распределение процессоров, памяти и других ресурсов между различными программами, их использующими.

5. В задачи операционной системы также входит агрузка программ в память и обеспечение их выполнения.

6. Файлы, составляющие ОС, хранятся на диске, поэтому система называется дисковой операционной (ДОС).

7. ОС можно назвать программным продолжением устройства управления компьютера.

8. По (according to) числу одновременно выполняемых задач операционные системы могут быть разделены на два класса:

- многозадачные (Unix, OS/2, Windows).
- однозадачные (например, MS-DOS) и

9. По числу одновременно работающих пользователей ОС можно разделить на:

- однопользовательские (например, MS-DOS, Windows 3.x);
- многопользовательские (Windows NT, Unix).

X. Make up a plan to the text and try to write a short summary of the text, using one sentence for each item of the plan.

Unit 11.

Cloud Computing

I. Look at the following quotes about cloud computing. Do you agree with them? Why (not)?

A lot of people are jumping on the [cloud] bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of "the cloud."

Andy Isherwood, quoted in ZDnet News, December 11, 2008

The interesting thing about Cloud Computing is that we've redefined Cloud Computing to include everything that we already do. . . . I don't understand what

we would do differently in the light of Cloud Computing other than change the wording of some of our ads.

Larry Ellison, quoted in the Wall Street Journal, September 26, 2008

II. Study the text and answer the question "Why is cloud computing an old idea?".

To Cloud Computing: An Old Idea Whose Time Has (Finally) Come

Cloud Computing is a new term for a long held dream of computing as a utility, which has recently emerged as a commercial reality.

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud.

When a Cloud is made available in a pay-as-you-go manner to the public, we call it a Public Cloud; the service being sold is Utility Computing. Current examples of public Utility Computing include Amazon Web Services, Google AppEngine, and Microsoft Azure. We use the term Private Cloud to refer to internal datacenters of a business or other organization that are not made available to the public. Thus, Cloud Computing is the sum of SaaS and Utility Computing, but does not normally include Private Clouds. We'll generally use Cloud Computing, replacing it with one of the other terms only when clarity demands it.

The advantages of SaaS to both end users and service providers are well understood. Service providers enjoy greatly simplified software installation and maintenance and centralized control over versioning; end users can access the service "anytime, anywhere", share data and collaborate more easily, and keep their data stored safely in the infrastructure. Cloud Computing does not change these arguments, but it does give more application providers the choice of deploying their product as SaaS without provisioning a datacenter: just as the emergence of semiconductor foundries gave chip companies the opportunity to

design and sell chips without owning a fab, Cloud Computing allows deploying SaaS—and scaling on demand—without building or provisioning a datacenter. Analogously to how SaaS allows the user to offload some problems to the SaaS provider, the SaaS provider can now offload some of his problems to the Cloud Computing provider.

From a hardware point of view, three aspects are new in Cloud Computing:

1. The illusion of infinite computing resources available on demand, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning;

2. The elimination of an up-front commitment by Cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs; and

3. The ability to pay for use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

While the attraction to Cloud Computing users (SaaS providers) is clear, who would become a Cloud Computing provider, and why? To begin with, realizing the economies of scale afforded by statistical multiplexing and bulk purchasing requires the construction of extremely large datacenters.

Building, provisioning, and launching such a facility is a hundred-million-dollar undertaking. However, because of the phenomenal growth of Web services through the early 2000's, many large Internet companies, including Amazon, eBay, Google, Microsoft and others, were already doing so. Equally important, these companies also had to develop scalable software infrastructure (such as MapReduce, the Google File System, BigTable, and Dynamo [16, 20, 14, 17]) and the operational expertise to armor their datacenters against potential physical and electronic attacks.

Therefore, a necessary but not sufficient condition for a company to become a Cloud Computing provider is that it must have existing investments not only in

very large datacenters, but also in large-scale software infrastructure and operational expertise required to run them.

The long dreamed vision of computing as a utility is finally emerging. The elasticity of a utility matches the need of businesses providing services directly to customers over the Internet, as workloads can grow (and shrink) far faster than 20 years ago. It used to take years to grow a business to several million customers – now it can happen in months.

From the cloud provider's view, the construction of very large datacenters at low cost sites using commodity computing, storage, and networking uncovered the possibility of selling those resources on a pay-as-you-go model below the costs of many medium-sized datacenters, while making a profit by statistically multiplexing among a large group of customers. From the cloud user's view, it would be as startling for a new software startup to build its own datacenter as it would for a hardware startup to build its own fabrication line. In addition to startups, many other established organizations take advantage of the elasticity of Cloud Computing regularly, including newspapers like the Washington Post, movie companies like Pixar, and many universities.

(Abridged from "Above the clouds: a Berkeley view of cloud computing" http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf)

III. Look through the text again and give Russian equivalents to the following words.

Utility, scale (v), Public cloud, Utility computing, semiconductor, application, Software as a Service (SaaS), on demand, service provider, multiplex, maintenance, offload, Private cloud, version (v), bulk purchase.

Now make up 3 sentences of your own using these words.

IV. Using external resources (e.g. wordspy.com) find 5-7 terms related to cloud computing and give their definitions.

V. Make pairs of words with similar meaning.

Commitment, short-term, current, appear, thereby, eliminate, enough, spread out, limitless, old desire, available, inner, reinforce, require, accessible, actual, undertaking, expertise, remove, proficiency, sufficient, emerge, by means of that, demand, up-front, deploy, definite, obligation, infinite, short-run, internal, long-held dream, reward, project.

VI. Find sentences in the text beginning with the following words and phrases, translate them into Russian.

To begin with... .

Analogously... .

In addition to ... .

From a hardware point of view ... .

Therefore ... .

Now using these words and phrases make written translation of the sentences given below.

1. Аналогично, облачные версии деловых и коммуникационных приложений позволят даже малым компаниям воспользоваться инструментами, больше характерными для корпораций.
2. Таким образом, облачные вычисления – это комбинация давно существующих идей и решений, обеспечивающая им новое качество.
3. Для начала создадим Частное облако, настроим сетевую адресацию и запустим виртуальный сервер, который будет выполнять роль маршрутизатора.
4. С точки зрения поставщика, благодаря объединению ресурсов и непостоянному характеру потребления со стороны потребителей, облачные вычисления позволяют экономить на масштабах, используя меньшие аппаратные ресурсы, чем требовались бы при выделенных аппаратных мощностях для каждого потребителя, а за счёт автоматизации

процедур модификации выделения ресурсов существенно снижаются затраты на абонентское обслуживание.

VII. Focus on grammar. Look at the sentences from the text and translate them paying attention to the words in bold. Comment on the use and meaning of these words.

- Computing does not change these arguments, but it **does** give more application providers the choice of deploying their product as SaaS without provisioning a datacenter... .
- ... thereby eliminating the need for Cloud Computing users to plan **far** ahead for provisioning. ... workloads can grow (and shrink) **far** faster than 20 years ago.

Complete the sentences.

1. – You said you would phone me! – I ... you! But you didnt answer. – Ok, I didnt phone you, but I ... send you a text message.
2. She ... look good in that dress.
3. This book is ... more interesting than that one.
4. I never ... understand what he saw in her.
5. He is ... taller than his brother.

VIII. Answer the questions about cloud computing basing on the text and your knowledge.

1. Name the main features of cloud services.
2. How many types of deployment models are used in the cloud?
3. What is the difference between Private cloud and Public cloud?
4. What are the advantages of SaaS for end users?
5. What are the advantages of SaaS for service providers?
6. Explain the essence of SaaS, PaaS and IaaS.

7. What is the biggest obstacle and opportunity for growth of cloud computing?
8. What prevented from realization of the idea of cloud computing before?
9. In what way do you use cloud computing?
10. Do you use cloud storage? If yes, which providers do you prefer? Why?

IX. Choose the correct answer.

1. Amazon Web Services is which type of cloud computing distribution model?

A.Software as a Service

B.Platform as a Service

C. Infrastructure as a Service

2. What is private cloud?

A. A standard cloud service offered via the Internet

B. A cloud architecture maintained within an enterprise data center

C.A cloud service inaccessible to anyone but the cultural elite

3. Which of the following isn't an advantage of cloud?

A.No worries about running out of storage

B.Easier to maintain a cloud network

C.Immediate access to computing resources

D.Paying only for what you use

4. Google Docs is a type of cloud computing.

A.True

B.False

5. "Cloud" in cloud computing represents what?

A.Wireless

B.Hard drives

C.People

D.Internet

X. Discuss in groups "Is cloud computing in Russia on the same level as in other developed countries?".

Unit 12.
Grid Ccomputing

I. Before reading the text try to explain what grid computing is.

II. Read the text and see if your definition of grid computing was right.

Clouds, Grids, and Distributed Systems

Grid computing provides key infrastructure for distributed problem solving in dynamic virtual organizations. It has been adopted by many scientific projects, and industrial interest is rising rapidly. However, Grids are still the domain of a few highly trained programmers with expertise in networking, high-performance computing, and operating systems. In particular, the number of users lags behind the initial forecasts laid out by proponents of grid technologies. This underachievement may have led to claims that the grid concept as a whole is on its way to being replaced by Cloud computing and various X-as-a-Service approaches.

With cloud computing, companies can scale up to massive capacities in an instant without having to invest in new infrastructure, train new personnel, or license new software. Cloud computing is of particular benefit to small and medium-sized businesses who wish to completely outsource their data-center infrastructure, or large companies who wish to get peak load capacity without incurring the higher cost of building larger data centers internally. The consumer does not own the infrastructure, software, or platform in the cloud. He has lower upfront costs, capital expenses, and operating expenses. He does not care about how servers and networks are maintained in the cloud.

Cloud computing evolves from grid computing and provides on-demand resource provisioning. Grid computing may or may not be in the cloud depending on what type of users are using it.

Grid computing requires the use of software that can divide and farm out pieces of a program as one large system image to several thousand computers. One concern about grid is that if one piece of the software on a node fails, other pieces of the software on other nodes may fail. This is alleviated if that component has a failover component on another node, but problems can still arise if components rely on other pieces of software to accomplish one or more grid computing tasks. Large system images and associated hardware to operate and maintain them can contribute to large capital and operating expenses.

Similarities and differences

Cloud computing and grid computing are scalable. Scalability is accomplished through load balancing of application instances running separately on a variety of operating systems and connected through Web services. CPU and network bandwidth is allocated and de-allocated on demand. The system's storage capacity goes up and down depending on the number of users, instances, and the amount of data transferred at a given time.

Both computing types involve multitenancy and multitask, meaning that many customers can perform different tasks, accessing a single or multiple application instances. Sharing resources among a large pool of users assists in reducing infrastructure costs and peak load capacity. Cloud and grid computing provide service-level agreements (SLAs) for guaranteed uptime availability of, say, 99 percent. If the service slides below the level of the guaranteed uptime service, the consumer will get service credit for receiving data late.

The storage computing in the grid is well suited for data-intensive storage, however it is not economically suited for storing objects as small as 1 byte. In a data grid, the amounts of distributed data must be large for maximum benefit. A computational grid focuses on computationally intensive operations.

Clouds and Grids share a lot commonality in their vision, architecture and technology, but they also differ in various aspects such as security, programming model, business model, compute model, data model, applications, and abstractions.

(Abridged from http://www.ibeehosting.com/blog/what-is-the-difference-between-cloud-computing-and-grid-computing.html)

III. Find words in the text that have similar meaning.

Focus; quickly; an addressable point on a network; execute; be unable to keep up; project; prediction; redundancy within computer network; supporter; cause; upon request; arrange for work to be done by others; the difference between the highest and lowest frequencies of a transmission channel; facilitated; amounts paid for goods and services that may be currently tax deductible; techniques which aim to spread tasks among the processors; first; ability to adjust configuration and size to fit new conditions; proficiency; principle in software architecture where a single instance of the software runs on a server, serving multiple client-organization; immediately; profit.

IV. Match the beginnings (1-9) to the endings (a-i) to make definitions of the words in bold.

1) A **proponent** is
2) To **replace**
3) When a company **outsources**
4) **Infrastructure** is
5) When a company becomes liable for something unpleasant or undesirable,
6) **Upfront** payment is
7) **Uptime** is
8) If a company distributes something according to a plan or set apart for a special purpose,
9) When you **evolve**,
    a) a person who favors a particular idea or proposal.

b) means to substitute a person or a thing for.
c) it turns to outside suppliers or manufacturers; contract workers from outside of a company to perform specific tasks instead of using company employees.
d) foundation, basis, substructure, underlying features of an operation.
e) it **incurs** troubles.
f) made in advance, ahead of time.
g) a period when something is functional and available for use.
h) it **allocates** it.
i) you develop, gradually change or mature over time.

V. Underline the key words in the sentences (in exercise 4) which summarize the meaning of the prefixes (they are underlined).

Look at the prefixes below and try to explain their meanings.

Pseudo-, anti-, mal-, trans-, per-, de-, bi-, dia-, em-, hemi-, co-/com-/con-, uni-, ambi-, circum-, fore-, intra-, inter-, un-.

Choose 3-5 prefixes that you would like to remember and make at least five words with each of them.

VI. Complete the words in the following sentences with an appropriate prefix from exercise 5.

- In addition, ...municating ...encrypted over the in-band network leaves the management ...actions open and ...secure.
- Last month computer ...time cost the company over €10000 in lost production.
- Business clients based on these processors are built for the needs of business and engineered to protect their data with new levels of ...formance and ...precedented ...bedded security technologies ...bined into a single high-...formance, secure business tool.

- Software applications may not be ...patible with all operating systems.
- Many companies distribute internal documents on their own ...net.

VII. Complete the text with the following words.

Performance, peer-to-peer (2), tools, complementary, multiple, physical (3), variety, solution, calculation, resources, virtual, clusters (3), single (2), scheduling, grid.

What is Grid Computing?

The idea behind grid is to make ... machines that may be in different ... locations, behave like they are one large ... machine. A ... of technologies are used to make this happen. ... of machines can be used to increase the ... available at one ... location but to go beyond that requires using ... communications ... and the internet to allow ... of machines at different ... locations to work together. Grid computing is precisely that, you have a ... ... process that uses ... communication to control multiple ... of machines at different locations. A compute cluster refers to a technology that allows a calculation to be done using ... CPUs at a ... site. This is normally done to improve ... by making more CPUs available for doing a ... . The clustering technology can be used independently or it can be used as a component part of a ... . The technologies are ... , a good starting point for someone who wants to use these technologies is to focus on using the clustering technology first and then to migrate to a distributed ... at a later point.

VIII. While working in ICT sphere you come across many abbreviations. Check if you remember some of them from the previous two units.

SLA, SaaS, PaaS, IaaS, CPU.

Try to guess what these abbreviations stand for.

API, PPP, MIDI, IEEE, VPN, DLL, URL, MMOS, VCL, P2P, OLE, CCU

Using dictionary find ten more IT abbreviations.

IX. Discuss the questions below.

1. What will be the best definition of grid computing?
2. Did grid computing evolve from cloud computing?
3. In which situations is grid computing more preferable over cloud computing? What is the target application for both of them?
4. Provide examples of companies using cloud computing and those using grid computing.
5. What components are necessary to form a grid?

X. Make a written translation of the text paying special attention to the terms related to grid computing.

In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the selected machine. More advanced grid systems would include a job scheduler of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid. The term scheduling is not to be confused with reservation of resources in advance to improve the quality of service. Sometimes the term resource broker is used in place of scheduler, but this term implies that some sort of bartering capability is factored into scheduling. In a scavenging grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job whose requirements are satisfied by the machine's resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-grid work, the grid job is usually suspended or delayed. This situation creates somewhat unpredictable completion times for grid jobs, although it is not disruptive to those machines donating resources to the grid. Grid applications that run in scavenging mode often mark themselves at the operating system's lowest priority level. In this way, they only run when no other work is pending. Due to the performance of modern day

processors and operating system scheduling algor, the grid application can run for as short as a few milliseconds, even between a user's keystrokes.

Taken from 'Introduction to grid computing' (ibm.com/redbooks)

Unit 13.

Open Source vs. Closed Source

I. Before you read answer the questions:

1. What was first to appear: open source or closed source?
2. Do you use more open or closed source software?

II. Read the text to learn more about Open Source and Closed Source.

Open source vs. closed source

Reviewing literature on open source and closed source security reveals that the discussion is often determined by biased attitudes toward one of these development styles.

Over the last few decades we have got used to acquiring software by procuring licenses for a proprietary, or binary-only, immaterial "object". We have come to regard software as a good we have to pay for just as we would pay for material objects, such as electronic devices, or food. However, in more recent years, this widely cultivated habit has begun to be accompanied by a new model, which is characterized by software that comes with a compilable source code (open source code). Often, such a source code is free of charge and may be modified and/or redistributed. The software type is referred to by the umbrella term "open source software". When discussing this alleged innovation in software distribution, we are reminded by (Glass, 2004) that, essentially, free and open source software dates right back to the origins of computing, as far back in fact as the 1950s, when all software was free, and most of it open. (Schwarz and Takhteyev, 2008) provide

detailed insights into the history and the diffusion of open source software. The application fields of open source software are manifold.

Obviously, its increasing availability and deployment makes it appealing for hackers and others who are interested in exploiting software vulnerabilities, which become even more dangerous when software is not applied in a closed context, but interconnected with other systems and the Internet (this argument is valid for closed source software as well).

Generally, the availability of source code to the public is a precondition for software being denoted as "open source software". Beyond this requirement, the Open Source Initiative (OSI) has defined a set of criteria that software has to comply with (OSI, 2006). The definition includes the permission to modify the code and to redistribute it. However, it does not govern the software development process in terms of who is eligible to modify the original version. When what is called "bazaar style" by (Raymond, 2001) is in place, any volunteer can provide source code submissions. Software development is then often based on informal communication between the coders (Gonzalez-Barahona, 2000). In a more closed environment, software is crafted by individual wizards and the development process is characterized by a relatively strong control of design and implementation. This style is referred to as "cathedral style" (Raymond, E.S. (2001) The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly, Beijing, China). The implementation of this modification procedure might have an impact on the security of software, so that a detailed discussion of open source security would need to take this into account. A plethora of OSD-compliant licenses have come into operation, such as the Apache License, BSD license, and GNU General Public License (GPL), which is maintained by the Free Software Foundation (FSF). The FSF provides a definition of " „free software" [as] a matter of liberty, not price." (FSF, 2007). In contrast to the OSD definition, the FSF definition explicitly focuses on the option of releasing improved versions to the public (freedom 3), thereby rejecting the strong

supervision of the modification process. Software is usually regarded as being "closed", if the source code is not available to the public.

Vulnerabilities. When software is executed in a way different from what the original software designers intended, this misbehaviour is rooted in software bugs. (Anderson, 2001) assumes the ratio between bugs and software lines of code (SLOC) to be about 1:35, i.e. Windows 2000 with its 35 Mio. SLOC would then have included one million bugs. The portion of bugs that are security critical ("vulnerabilities") is assumed to be 1% (Anderson, 2001), resulting in an amazingly high figure of 350,000 vulnerabilities in Windows 2000. Detected vulnerabilities can further be divided into those being published and those that remain unpublished.

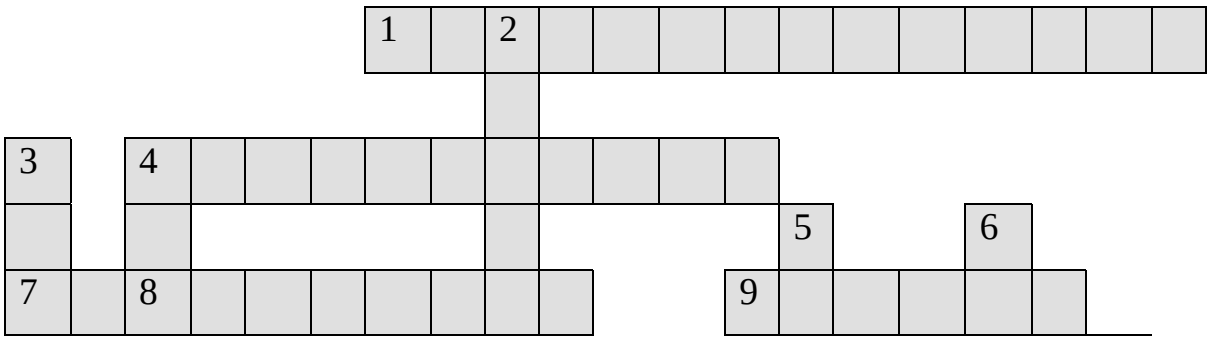
Vulnerabilities are (software) product-related weaknesses, for which publicly accessible databases are available. They are the root for concrete security incidents (breaches), which are system-related and cause the actual harm. Breaches are much more difficult to investigate, because data is scarcer.

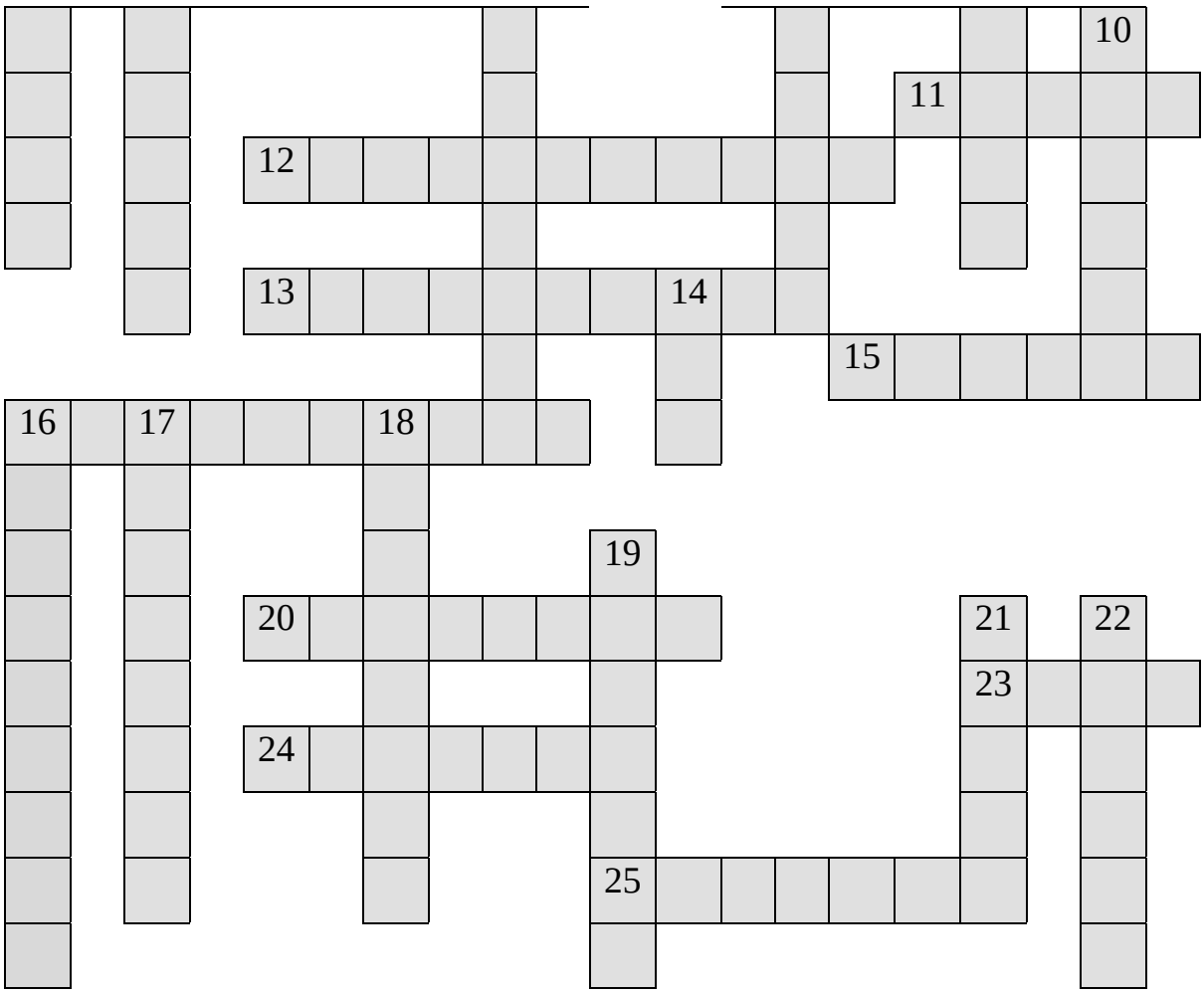
(Taken from "Security of open source and closed source software: An empirical comparison of published vulnerabilities", Guido Schryen International Computer Science Institute, Berkeley)

III. Find derivatives for the following words from the text.

Vulnerability, exploit, publicly, submission, requirement, appealing, weakness, coder, development, available, implementation, valid, modification, misbehaviour, deployment, accessible, procedure, compilable.

IV. Solve the crossword puzzle.





Horizontal

1. The act of accomplishing some aim or executing some order; or providing a practical means for accomplishing something.
4. Owned by a private individual or corporation under a trademark or patent.
7. Means the same as 'entry', 'giving in', 'filing'.
9. Of or relating to a system of numeration having 2 as its base.
11. A relationship between two quantities, normally expressed as the quotient of one divided by the other.
12. Something obligatory, a necessity.
13. Collectable, able to be gathered together.
15. A period of ten years.
16. The distribution of forces in preparation for work.
20. Qualified or entitled to be chosen.
23. To come into existence; originate.

24. To get by special effort; obtain or acquire.
25. To employ to the greatest possible advantage.

Vertical

2. Something that is necessary to a subsequent result.
3. The act or outcome of grasping the inward or hidden nature of things or of perceiving in an intuitive manner.
5. A certificate, tag, document, etc., giving official permission to do something.
6. A failure to perform some promised act or obligation.
8. Prejudiced.
10. A skilled or clever person.
14. A defect in the code or routine of a program.
16. Spreading, distribution.
17. A superabundance; an excess.
18. Of many kinds; multiple.
19. Represented as existing or as being as described but not so proved; supposed.
21. To make with great skill and care.
22. To change in form or character; alter.

V. Close the text and tell whether the following sentences are true, false or there is no such information in the text.

1. Closed source is more secure than open source.
2. In "bazaar style" any volunteer can provide source code submissions.
3. Software is usually regarded as being "closed", if the source code is available to the public.
4. The portion of breaches of software is assumed to be 1%.
5. Open source means the same as free (without payment).

VI. Work in pairs. One of you makes a list of advantages and disadvantages of open source, while another one works over the same list of closed source. Then swap the lists. Do you agree with it? Can you add anything?

VII. Answer the questions on the text.

1. Which of the open source definitions given in the text do you find the most accurate?
2. What is the portion of security critical bugs?
3. What are the characteristics of "cathedral style"?
4. Why not all vulnerabilities are published?
5. Which do you think is more popular in Russia: open or closed source software? In your opinion, does it reflect preferences in the whole world? Why (not)?
6. Will the ratio of open source software to closed source software in the market change in the nearest future?
7. As a programmer would you like your product to be open source or closed source? Explain your choice.

VIII. Comment on the following quotes. Do you agree with them?

- “If you control the code, you control the world. This is the future that awaits us.” – Marc Goodman.
- "To be able to choose between proprietary software packages is to be able to choose your master. Freedom means not having a master. And in the area of computing, freedom means not using proprietary software." – Richard M. Stallman.
- “It just makes it even harder for people to even approach the (open source) side, when they then end up having to worry about ... public humiliation.” – Linus Torvalds.

IX. Read the information below.

Proponents of open source software stress the strength of the resulting review process and argue in the sense of Raymond, that, “Given enough eyeballs, bugs are shallow.”, while some opponents follow the argument of Levy, who remarks “Sure, the source code is available. But is anyone reading it?"
Which side do you take? Discuss it as a group. You might find useful the list of advantages and disadvantages of open and closed source software that you made in exercise 6.

X. Sum up all the ideas and write an essay "Open source software: pros and cons". Use appropriate linkers. (First/ firstly ..., secondly ..., thirdly ... . On the other hand, there are also (some) disadvantages ... . For instance/ for example ... . In conclusion/ To sum up, I think ...).

Unit 14.
Future Human Computer Interaction

I. Before you read the text work in pairs. Find out what input-output device you use more often. Compare your answers with other pairs, are you surprised with the results?

II. Scan the text and complete it with the words:
video input and output, human computer interaction, gesture and speech recognition, speech detection, touch screens and multi-touch pads.
Then read the text carefully to know more about future human computer interaction techniques.

As humans are used to handle things with their hands the technology of multi-touch displays or touchpad’s brought much more convenience for use in daily life. Thereby the ..... take an important role as these are the main communication methods between humans and how they could disrupt the keyboard

or mouse as we know it today. Thus it will surely take not much time that sophisticated techniques will enhance these techniques for human ..... .

Hewett, et al defined that ”Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.”[1] So since the invention of the Human Computer Interface in the 1970s at Xerox Park, we are used to have a mouse and a keyboard to interact with the computer and to have the screen as a simple output device. With upcoming new technologies these devices are more and more converting with each other or sophisticated methods are replacing them.

Nowadays the usage of ..... seems to be really common and that this is going to be the future of human computer interaction, but there is for sure more enhancement which can be seen in many approaches. In the field of multi-touch products the trend to bigger touch pads in terms of multi-touch screens can be seen. Therefore the technique of a single-touch touchpad as it is known from former notebooks is enhanced and more fingers offering natural human hand gestures can be used. Thus the user can use up to 10 fingers to fully control things with both hands as with the 10/GUI system which R. Clayton Miller [2] introduced in 2009. With another upcoming tool even the usage of any surface for such a touch screen can be used in future, like for the example Displax ™ Multitouch Technology from Displax ™ Interactive Systems [4]. From these examples it can be clearly seen that new high potential techniques are pushing into the market and are going to replace the Apple’s iPad and Microsoft’s Surface.

Great efforts have been made also in the section of ..... devices like for example SixthSense that Pranav Mistry, et al. [5] invented in 2009. In their approach they are using a wearable gesture interface, where they are taking the HCI to the next level where communication is done with hands without any handheld tools; while researchers at Carnegie Mellon University and Microsoft have developed another highly topical approach by using the humans arm as an input surface, called Skinput. As SixthSense and Skinput are mainly designed for

mobil usage, Oblong Industries invented with g-speak a full gesture input/output device with a more sophisticated 3D user interface,which is designed for a usage with big screens that are using a lot more space.

..... is always mentioned as the most common straight-forwarded way, after the gestural motion, of how people interact between each other. The major thing with speech detection is to create a good algorithm not only to select the noise from the actual voice but rather detecting what humans are actually saying. First main algorithm is based on improving spoken natural language, the second one is connected with Microsoft Speech Technologies like the Speech Application Programming Interface SAPI. With this API Microsoft provides on the one hand a converter for full human spoken audio input to readable text. On the other hand it can be also used to convert written text into human speech with a synthetic voice generator.

There are a lot approaches of new future ..... methods and also devices or prototypes in which these techniques are already in use. We are tending towards to disrupt the usage of the mouse and the keyboard as we are used to use them as a computer input device for the last 3 decades. Many new methods are going into the sector of using human hand gestures and even multi-modal methods to interact with the computer. With the tools described we have seen that with the released iPad and Microsoft's Surface some of these methods are already included. For sure there are more sophisticated methods which will push into the market soon.

(From "Future Human Computer Interaction with special focus on input and output techniques" by Thomas Hahn)

III. Put words in the correct order and you will get some additional information about the devices mentioned in the text. The first word is underlined.

the tied projected any <u>In</u> a certain against the output which place is you directly surface with a like devices which simple is clearly to SixthSense advantage other small main are to projector.

very the beside scratches With damage is durable fact weight of a on just 300g surface from placed transportable protect Displax **TM** is tool that it well as the film is the also back of the Multitouch the to it and the other.
this as tracking in both a 10/GUI human can interact with the with a maybe With and hands use it and computer also beings as opportunity keyboard system device.

IV. Find derivatives of the following words in the text and explain their meaning.

Nouns: convenient, enhance, evaluate, implement, detect.

Adjective: sophisticate, wear, read, concern, compute.

What noun suffixes are used in the words? Can you think of other words with these suffixes?

What adjective suffixes are used in the words? Can you think of other words with these suffixes?

V. Make up combinations using words form A and B.

A: User, signal, natural, interact, upcoming, major, the convenience of, algorithm, provide, handheld, handle, improve, sophisticated, software, disrupt.

B: Device, an installment plan, enhancement, access, change, deliveries, with audience, method, phenomenon, advantage, pronunciation, validation, interface, detection, the case.

VI. Open the brackets and comment on the use of the verb "use" in the sentences.

1) As humans *(use)* to handle things with their hands ...

2) With another upcoming tool even the usage of any surface for such a touch screen can *(use)*in future, ...

3) On the other hand it can also *(use)* to convert written text ...

4) ... as we *(use) (use)* them as a computer input device for the last 3 decades.

VII. While working with scientific literature pay attention to the layout. Try to answer the questions below without referring to the text.

- How does the author refer to the work of other scientists?
- What does *et al.* mean?
- What is the singular form of the word "*phenomena*"?

You can often see the following Latin abbreviations in the articles. Do you remember what they mean?

NB, etc., e.g., i.e., per cent, vs., viz., A.D., am, pm, ibid.

Give plural form of the following nouns of Latin and Greek origin. Use the dictionary if necessary.

Medium, abacus, analysis, phenomenon, axis, thesis, nucleus, criterion, matrix.

VIII. Answer the following questions on the text.

1. What input and output devices are mentioned in the text?
2. Using additional information compare new input and output techniques, outline advantages and disadvantages of each of them.
3. Most people use mouse as their pointing device. Discuss the reasons why someone might prefer to use one of the other pointing devices.
4. Do you think that voice input is practical for your own use?
5. In your opinion, what is the future of input and output techniques?
6. To answer these questions use your own knowledge from reading or experience or imagine the possibilities. What kind of input device might be convenient for this type of jobs or situations?
   a) A restaurant in which customers place their own orders from the table.
   b) A psychologist who wants o give a new client a standard test.
   c) A retailer who wants to move customers quickly through the checkout lines.
   d) A small business owner who wants to keep track of employee work hours.

IX. Translate the text to English (or give its main idea).

Бюро по патентам и товарным знакам США выдало Apple патент 8,228,305 под названием "Методы ввода информации человеком в вычислительное устройство", в котором описано множество способов взаимодействия с электронным устройством, лежащими за пределами возможностей обычной периферии, сообщает Patently Apple.

Так, например, в патенте предлагается использовать практически любую поверхность для ввода данных с помощью прикосновений. При этом фиксироваться может не только положение пальцев, но и величина их давления на поверхность. Такую поверхность, например, можно превратить в холст, и рисовать на ней; толщина линий будет завсеть от силы нажатия.

В патенте также упоминаются перчатки для взаимодействия с объектами в виртуальном пространстве, которое способна симулировать вычислительная система.

Для считывания данных в виртуальном пространстве Apple предлагает использовать технологию "5D". Данная технология подразумевает, что вычислительная система будет не только фиксировать положение, например, ладоней человека по трем осям, но и направление их перемещения и скорость (то есть всего 5 параметров). Все это будет определять степень воздействия на виртуальный предмет.

В патенте также описывается система, которая бы позволяла управлять игровым персонажем с помощью перемещения пользователя и частей его тела в пространстве (напоминает реализованный коммерческий продукт Microsoft Kinect для игровой приставки Xbox 360).

Согласно описанию, в качестве входных в вычислительную систему данных могут выступать: изменение электромагнитного поля, перемещение потоков воздуха в помещении и даже изменение температуры.

Как уточняет Patently Apple, автором данного документа является не сама Apple - компания приобрела наработки у канадского изобретателя Тимоти Прайора (Timothy Pryor). Заявки на получение связанных патентов

датируются 1999-2006 г. В июле 2009 г. Apple подала заявку на регистрацию патента в его окончательном виде.

В Apple предполагают, что в будущем подобные методы ввода данных могут заменить традиционные клавиатуру, мышь и даже сенсорные экраны. Отметим, что некий подобный взгляд на будущее информационных технологий разделяет основатель Microsoft Билл Гейтс (Bill Gates).

X. Using additional resources make presentations about recent human computer interaction techniques.

Unit 15.
Robots

I. Read the following text. Do you agree that "nowadays robots are in the infancy stage of their evolution"?

Why build robots? Robots are indispensable in many manufacturing industries. The reason is that the cost per hour to operate a robot is a fraction of the cost of the human labor needed to perform the same function. However nowadays robots are in the infancy stage of their evolution. As robots evolve, they will become more versatile, emulating the human capacity and ability to switch job tasks easily, since robots require a combination of elements to be effective: sophistication of intelligence, movement, mobility, navigation, and purpose.

II. Study the information about robots in the text.

Industrial robots. Robots are indispensable in many manufacturing industries. For instance, robot welders are commonly used in automobile manufacturing. Other robots are equipped with spray painters and paint components. The semiconductor industry uses robots to solder (spot weld) microwires to semiconductor chips. Other robots (called “pick and place”) insert integrated circuits (ICs) onto printed circuit boards, a process known as “stuffing

the board." These particular robots perform the same repetitive and precise movements day in and day out. Robots improve the quality and profit margin (competitiveness) of manufacturing companies.

Design and prototyping. Some robots are useful for more than repetitive work. Manufacturing companies commonly use computer-aided design (CAD), computer-aided manufacturing (CAM), and computer numerical control (CNC) machines to produce designs, manufacture components, and assemble machines. These technologies allow an engineer to design a component using CAD and quickly manufacture the design of the board using computer-controlled equipment.

Hazardous duty. Without risking human life or limb, robots can replace humans in some hazardous duty service. Take for example bomb disposal. Typically these robots resemble small armored tanks and are guided remotely by personnel using video cameras (basic telepresence system) attached to the front of the robot. Robotic arms can grab a suspected bomb and place it in an explosion-proof safe box for detonation and/or disposal. Similar robots can help clean up toxic waste. Robots can work in all types of polluted environments, chemical as well as nuclear.

Maintenance robots specially designed to travel through pipes, sewers, air conditioning ducts, and other systems can assist in assessment and repair. A video camera mounted on the robot can transmit video pictures back to an inspecting technician. Where there is damage, the technician can use the robot to facilitate small repairs quickly and efficiently. Fire-fighting robots better than a home fire extinguisher, how about a home fire-fighting robot? This robot will detect a fire anywhere in the house, travel to the location, and put out the fire.

Medical robots fall into three general categories: diagnostic testing (e.g. Papnet, a neural network tool that helps cytologists detect cervical cancer quickly and more accurately), telepresence surgery (here a surgeon is able to operate on a patient remotely using a specially developed medical robot), enhanced manipulation (the surgeon performs delicate and microscopic surgical procedures on a patient through a robot).

Nanotechnology is the control and manipulation of matter at the atomic and molecular level. Tiny (nano) components can be assembled to make machines and equipment the size of bacteria. IBM has already created transistors, wires, gears, and levers out of atoms. Nanotechnology can also be used to create small and microscopic robots. Imagine robots so small they can be injected into a patient's bloodstream. The robots travel to the heart and begin removing the fatty deposits, restoring circulation. Or the robots travel to a tumor where they selectively destroy all cancerous cells.

War robots are becoming increasingly more important in modern warfare. Drone aircraft can track enemy movements and keep the enemy under surveillance. The Israeli military used an unmanned drone in an interesting way. The drone was created to be a large radar target. It was flown into enemy airspace. The enemy switched on its targeting radar, allowing the Israelis to get a fix on the radar position. The radar installation was destroyed, making it safe for fighter jets to follow through. Smart bombs and cruise missiles are other examples of "smart" weaponry. As much as I appreciate Asimov's Three Laws of Robotics, which principally state that a robot should never intentionally harm a human being, war bots are here to stay.

(Abridged from "Robots, androids and animatrons" by John Lovine)

III. Give Russian equivalents to the words and word combinations below and explain their meaning in your own words.

Indispensable, enhanced, disposal, deposit, limb, margin, surveillance, hazardous, facilitate, remotely, wire, assessment, capacity, precise, resemble, missile, accurate, versatile, day in and day out, appreciate, grab, target, detect, switch, put out, emulate, solder, installation, lever, surgery.

Choose 3-5 words that you find hard to memorize and use them in sentences of your own.

IV. Match words with their definitions.

Robot	characterized by perfect conformity to fact or truth ; strictly correct
Evolution	uplifting enlightenment; being expert or having knowledge of some technical subject
Circuit	a mechanical device for performing a task which might otherwise be done by a human
Semiconductor	conforming exactly or almost exactly to fact or to a standard or performing
Sophistication	make better or more attractive
Enhance	a complete path through which an electric current can flow
Precise	a material, typically crystaline, which allows current to flow under certain circumstances.
Accurate	a process in which something passes by degrees to a different stage

V. Fill in the gaps using the words from the previous exercises.

1. The law in this point is not ... .(Bacon)
2. Her .... beauty was the result of a good night's sleep rather than makeup.
3. Playing a guitar is ... different from playing the drums.
4. Highly pure tellurium is used in ... .
5. Technological ... spelled superiority in the military sphere.

6. Transported and enamel ... belong to brand new production not in Russian, but also in foreign market.
7. And if you cannot think yourself, let your ... do the thinking.
8. Mary ... her mother in looks.
9. It has been raining all day! How can we go out when it rains ... ?
10. Skydiving is a ... sport.

VI. Decide if the following sentences are true or false. Explain your choice.

1. The performance of tasks by robots is based on preprogrammed algorithms.
2. Today's most advanced industrial robots will soon become "dinosaurs".
3. Medical robots are used for diagnostic testing and enhanced manipulating.
4. Robots in industry are used only for repetitive work.
5. Most robots are made and used in Japan.

VII. Discuss the questions below in the class.

1. Is there a difference between a machine, like a microwave, and a robot?
2. Where did the word 'robot' come from?
3. How did first robot appear?
4. Do you know Isaac Asimov? What did he do?
5. In what sphere (from the ones mentioned in the text) do you think the usage of robots is of special importance?
6. Can you think of any sphere (e.g. nanotechnology) that could exist nowadyas without robots?
7. Have you seen usage robots in industry/ design/ maintenance/ medical spheres? What were your impressions? Were you satisfied with their work?
8. Can a robot completely replace a human in some work?
9. Do you think robots will cause unemployment in the future or make more work?
10. Is there a strong robotics industry in your country?

VIII. Look at the suffixes below, find words with them in the text and divide the suffixes in the box into those which usually indicate a noun form and those which are used in adjective forms. There is one that you don't need.

-ence/-ance, -ed, -able, -ity, -ate, -ics, -on, -or/-er, -ness, -ment, -ar, -ic, -fare, -al, -proof, -ry/-y, -ly, -ing, -able, -ous, -ion, -ive.

Explain why you did not need one suffix.

Make 2-3 words using each suffix.

IX. Find information about recent usage of robots in manufacturing industry. Discuss new technologies and techniques of their usage in the class.

X. Make a plan of the text, find key words in each part and make a summary of the text "Robots" adding extra information that you discussed in the exercises above.

Unit 16.
Artificial intelligence and robots

I. Before you read answer the questions in pairs.

1. Spend one minute writing down all of the different words you associate with the word “robot”. Share your words with your partner(s) and talk about them. Together, put the words into different categories.
2. What is your favourite movie that feature robots?

II. Read the text to know more about AI.

Artificial intelligence and robots

What is **artificial intelligence**? This is a legitimate question. We most certainly will develop neural networks that are intelligent before we develop nets that become conscious. So in attempting to create neural networks that are

intelligent or demonstrate intelligence, what criteria should one use to determine if this goal has been achieved? Alan Turing, a British mathematician, devised an interesting procedural test that is generally accepted as a valid way to determine if a machine has intelligence. The test is conducted as follows: A person and the machine hold a conversation by typing messages to one another via a teletype. If the machine can carry on a conversation without the person being able to determine whether a machine or person exists at the other teletype, the machine can be classified as intelligent. This is called the Turing test and is one criteria used to determine AI. Although the Turing test is well accepted, it isn't a definitive test for AI. There are a number of "completely dumb" language processing programs that come close to passing the Turing test. The most famous program is named ELIZA, developed by Joseph Weizenbaum at the Massachusetts Institute of Technology (MIT). ELIZA simulates a psychologist, and you are able to conduct a conversation with ELIZA. For instance, if you typed to ELIZA that you missed your father, ELIZA might respond with "Why do you miss your father?" or "Tell me more about your father." These responses may lead you to believe that ELIZA understands what you have said. It doesn't. The responses are clever programming tricks constructed from your statements. Therefore, if we like, we could do away with the Turing test and consider a different criterion. Perhaps consciousness or selfawareness would be a better signpost of intelligence. A self-aware machine would certainly know that it is intelligent. Another criterion, more simple and direct is the ability to learn from experience. Of course, we could abandon logical approximations entirely and state that intelligence is achieved in systems that develop a sense of humor. As far as I know humans are the only animals that laugh. Perhaps humor and emotion will end up being the truest test of all.

As for intelligence packed in a robot, it takes one or two forms: rule-based (expert) or neural. It's possible for both forms of intelligence to work in tandem. This synthesis of intelligence will be commonly used in robotics to create a robust intelligence system. Expert (rule-based) intelligence programs are familiar to most people; these are programs written in high-level or low-level languages like C ,

BASIC, and assembly. Neural systems on the other hand do not have a central processing unit (CPU), rather they function on a neural stimulus-response mechanism. The robotic stimulus-response mechanism goes by a number of names, including neural network, behavioral-based robotics, subsumption architecture, and nervous network. William Grey Walter pioneered behavioral-based robotics in the late 1940s. Independent of Walter's work, neural-based robotic response was academically explored and developed in the 1980s by Valentino Braitenberg in his book Vehicles: Experiments in Synthetic Psychology. Rodney Brooks at the Massachusetts Institute of Technology (MIT), inspired by work accomplished by Walter, developed his own derivative of stimulus responses he calls "subsumption architecture." Mark Tilden, inspired by work done by Rodney Brooks, founded BEAM robotics, which uses "nervous nets." Behavioral-based robotics is a hot topic and one that will continue to get hotter in the future. In these architectural schemes the stimulus-response mechanisms can be layered on top of one another. A multilayer stimulus-response mechanism can exhibit what appears as strikingly intelligent behavior. One approach is called "topdown intelligence" and the other is called "bottom-up intelligence. The top-down approach attempts to create an expert system or program to perform a controlled search and discover. The bottom-up approach creates "artificial" behavior in the robot and then causes it to explore and discover. At first glance you may not see much of a difference in either approach, but there is one and it's quite significant. If the expert system approaches a situation (or terrain) it hasn't been programmed to handle, it will falter. The behavior system on the other hand isn't looking for any template "programmed" situation to calculate procedures and couldn't care less about the situation; it just goes on exploring. Robotists have found over the last 30 years of experimentation that bottom-up programming (behavioral-based) is successful many times where top-down programming fails.

(Abridged from "Robots, androids and animatrons" by John Lovine)

III. Match the following phrases from the article.

neural	system
legitimate	in tandem
self-aware	mechanism
bottom-up	behavioral-based
subsumption	question
to work	a conversation
stimulus-response	network
robotics	approach
robust	architecture
to conduct	machine

IV. Spell the <u>jumbled</u> words (from the text) correctly.

Another <u>irenitco</u> (n.)

a <u>vitidifene</u> test (adj.)

perhaps <u>sssscoounnie</u> (n.)... would be ...

<u>fescusclus</u> (adj.) many times

<u>gkitylsrin</u> (adv.) intelligent behaviour

V. Design project.

Your teacher is a rich investor. He/she has sponsored a robotics design competition. The team who comes up the best idea for a new robot will be awarded 10 million dollars to have their product developed. You and your parner have entered the contest. You have five minutes to think of a new robotics product. When fininshed, try to sell your idea to your teacher.

VI. Role play

*(*note: each student reads his/her role only)

Student A:	You believe that space exploration is the key to the survival of the human race. Take a minute to think of other reasons why space exploration is important. When ready, ask your partner what she/he thinks of the new mission to Mars.
Student B:	Governments worldwide spend billions of dollars on space exploration. You believe it's a waste of time and money. The Earth has many problems. We need to fix our own problems instead of flying off into space. Think of other reasons to support your argument. Your partner will start the conversation.

VII. Discuss the questions below in the class.

1. Do you feel comfortable with the idea of AI (that robots can think)?
2. Do you think we create a new problem with each invention? Think of examples.
3. Is there anything that does not yet exist that you would like to see invented? What is it?
4. Imagine the job you wish to hold when you get older – could a robot be programmed to do that job as well as you?
5. A film like The Matrix has explored the idea that we might be living in virtual reality. But what evidence is there for or against this hypothesis? And what are its implications?
6. What name would you give to your robot?
7. Do you agree that robots will always be too expensive for most people?

VIII. Magazine article: Write a magazine article about robots. Include an imaginary interview with a robot. Read what you wrote to your classmates in the next lesson. Give each other feedback on your articles.

IX. In pairs / groups, write down questions about robots and their roles in our future.

- Ask other classmates your questions and note down their answers.
- Go back to your original partner / group and compare your findings.
- Make mini-presentations to other groups on your findings.

X. Translate the text below to English.

Хорошим примером искусственных агентов служат интеллектуальные роботы. В первую очередь, подобные роботы имеют широкий ассортимент искусственных органов чувств (сенсорные датчики) и искусственных эффекторов (манипуляторы). Их мобильность достигается благодаря колесным, гусеничным, шагающим и прочим системам перемещения. Активность и автономность роботов тесно связаны с наличием средств целеполагания и планирования действий, систем поддержки решения задач, а интеллектуализация, помимо обладания системой обработки знаний, предполагает развитые средства коммуникации различных уровней, вплоть до средств естественно-языкового общения.

Набор сенсорных датчиков внешней и внутренней информации включает: зрительные и звуковые (ультразвуковые), тактильные и кинестетические датчики. Кроме того, часто применяются датчики для измерения температуры, давления, влажности, радиоактивности, магнитного поля и других физических величин.

При зрительном очувствлении роботов источниками информации служат теле- и видеокамеры, оптические и голографические датчики. К звуковым датчикам относятся микрофоны, эхолокационные датчики и пр. При взаимодействии робота с внешней средой ключевую роль играют тактильные датчики, позволяющие реагировать на прикосновение и измерять давление в местах контакта с объектом среды. Эти датчики служат для обнаружения различных объектов, распознавания внешней обстановки путем

ощупывания предметов, а в конечном счете, для получения обратных связей по усилиям.

Неотъемлемым атрибутом интеллектуальных роботов является наличие специальной подсистемы планирования, составляющей программу действий робота в реальных условиях окружающей среды, которые определяются рецепторами робота. Для планирования деятельности робот должен иметь знания о свойствах окружающей среды и путях достижения целей в этой среде.

Unit 17.
Computational linguistics.

I. Comment on the following statements:

Friendly software should listen and speak.
Machines can also help people communicate with each other.
Language is the fabric of the web.

II. Before you read the text say what computational linguistics deals with. Read the text to check your answers.

Computational linguistics (CL) is a discipline between linguistics and computer science which is concerned with the computational aspects of the human language faculty. It belongs to the cognitive sciences and overlaps with the field of artificial intelligence (AI), a branch of computer science aiming at computational models of human cognition. Computational linguistics has applied and theoretical components. Human language is the most exciting and demanding puzzle. Theoretical CL takes up issues in theoretical linguistics and cognitive science. It deals with formal theories about the linguistic knowledge that a human needs for generating and understanding language. Today these theories have reached a degree of complexity that can only be managed by employing computers.

Computational linguists develop formal models simulating aspects of the human language faculty and implement them as computer programs. These programs constitute the basis for the evaluation and further development of the theories. In addition to linguistic theories, findings from cognitive psychology play a major role in simulating linguistic competence. Within psychology, it is mainly the area of psycholinguistics that examines the cognitive processes constituting human language use. The relevance of computational modeling for psycholinguistic research is reflected in the emergence of a new subdiscipline: computational psycholinguistics. We teach computers to communicate with people.

Applied CL focuses on the practical outcome of modelling human language use. The methods, techniques, tools and applications in this area are often subsumed under the term language engineering or (human) language technology. Although existing CL systems are far from achieving human ability, they have numerous possible applications. The goal is to create software products that have some knowledge of human language. Such products are going to change our lives. They are urgently needed for improving human-machine interaction since the main obstacle in the interaction between human and computer is a communication problem. Today's computers do not understand our language but computer languages are difficult to learn and do not correspond to the structure of human thought. Even if the language the machine understands and its domain of discourse are very restricted, the use of human language can increase the acceptance of software and the productivity of its users.

Natural language interfaces enable the user to communicate with the computer in French, English, German, or another human language. Some applications of such interfaces are database queries, information retrieval from texts, so-called expert systems, and robot control. Current advances in the recognition of spoken language improve the usability of many types of natural language systems. Communication with computers using spoken language will have a lasting impact upon the work environment; completely new areas of application for information technology will open up. However, spoken language

needs to be combined with other modes of communication such as pointing with mouse or finger. If such multimodal communication is finally embedded in an effective general model of cooperation, we have succeeded in turning the machine into a partner.

Much older than communication problems between human beings and machines are those between people with different mother tongues. One of the original aims of applied computational linguistics has always been fully automatic translation between human languages. From bitter experience scientists have realized that they are still far away from achieving the ambitious goal of translating unrestricted texts. Nevertheless computational linguists have created software systems that simplify the work of human translators and clearly improve their productivity. Less than perfect automatic translations can also be of great help to information seekers who have to search through large amounts of texts in foreign languages.

The rapid growth of the Internet/WWW and the emergence of the information society pose exciting new challenges to language technology. Although the new media combine text, graphics, sound and movies, the whole world of multimedia information can only be structured, indexed and navigated through language. For browsing, navigating, filtering and processing the information on the web, we need software that can get at the contents of documents. Language technology for content management is a necessary precondition for turning the wealth of digital information into collective knowledge. The increasing multilinguality of the web constitutes an additional challenge for our discipline. The global web can only be mastered with the help of multilingual tools for indexing and navigating. Systems for crosslingual information and knowledge management will surmount language barriers for e-commerce, education and international cooperation.

We still do not know very well how people produce and comprehend language. Yet our understanding of the intricate mechanisms, that underlay human language processing, keeps growing. Modelling such mechanisms on a computer

also helps us to discover and formally describe hidden properties of human language that are relevant for any kind of language processing including many useful software applications. Our long term goal is the deep understanding of human language and powerful intelligent linguistic applications. However, even today's language technologies full of clever short cuts and shallow processing techniques can be turned into badly needed software products.

For many students and practitioners of computational linguistics the special attraction of the discipline is the combination of expertise from the humanities, natural and behavioral sciences, and engineering. Scientific approaches and practical techniques come from linguistics, computer science, psychology, and mathematics. At some universities the subject is taught in computer science at others it belongs to linguistics or cognitive science. In addition there is a small but growing number of programs and departments dedicated solely to computational linguistics.

(http://hanz.uzkoreit.net)

III. Answer the following questions.

1. What does theoretical CL deal with?

2. What does applied CL focus on?

3. What do natural language interfaces help to do?

4. How can CL simplify communication between people?

5. Why is language technology for content management a necessary precondition?

6. What is the special attraction of CL?

IV. Match the words in A with their synonyms in B.

A. cognition, embedded, emergence, to enable, interaction, obstacle, to overlap, precondition, recognition, to surmount

B. acknowledgment, to allow, appearance, awareness, communication, to cover a part of, impediment, inserted, to overcome, prerequisite

V. Match the words in two columns so that they should form word-combinations from the text.

Achieve	Being
Badly	Cuts
Collective	Complexity
Degree of	The goal
Human	Growing
Human-machine	Outcome
Keep	Needed
Practical	Interaction
Short	Needed
Urgently	knowledge

VI. Match the first half of each sentence with the most appropriate second half.

1.	Computational linguistics as a field predates artificial intelligence,	A.	that would allow them the same remarkable capacity to process language.
2.	Computational linguistics originated with efforts in the United States in the 1950s	B.	dealing with human-level comprehension and production of natural languages.
3.	Computers had proven their ability	C.	into existence in the 1960s.
4.	It was thought to be only a short matter of time before the technical details could be taken care of	D.	devoted to developing algorithms and software for intelligently processing language data.

5.	Computational linguistics was born as the name of the new field of study	E.	to have computers automatically translate texts in foreign languages into English.
6.	Artificial intelligence came	F.	a field under which it is often grouped.
7.	the field of computational linguistics became that sub-division of artificial intelligence	G.	ability to do complex mathematics much faster and more accurately than humans.

VII. Find 3-4 key words in every paragraph. Write down the sentences of your own using these words. They should give the main idea of each paragraph.

VIII. Make a summary of the text using words and phrases from Appendix 2.

Appendix 1

Mathematical and scientific symbols

Common pronunciations of mathematical and scientific symbols are given in the list below.

Symbols

+	Plus	/'plʌs/
-	Minus	/'maɪnəs/
±	plus or minus	/'plʌs ɔ: 'maɪnəs/
x	multiplied by	/'mʌltɪplaɪd baɪ/
/	over; divided by	/'əʊvə/ /dɪ'vaɪdəd/
÷	divided	/dɪ'vaɪdəd/
=	equals	/'ɪ:kwəlz/

≈	approximately, similar	/ə'prɒksɪmətlɪ/ /'sɪmɪlə tʊ/
≡	equivalent to; identical	/ɪk'wɪvələnt tʊ/ /aɪ'dentɪkl tʊ/
≠	not equal to	/'nɒt 'i:kwəl tʊ/
>	greater than	/'greɪtə ðən/
<	less than	/'les ðən/
≥	greater than or equal to	/'greɪtə ðən ər 'i:kwəl tʊ/
≤	less than or equal to	/'les ðən ər' i:kwəl tʊ/
≯	not greater than	/'nɒt 'greɪtə ðən/
≮	not less than	/'nɒt 'les ðən/
≫	much greater than	/'mʌʧ 'greɪtə ðən/
≪	much less than	/'mʌʧ 'les ðən/
⊥	perpendicular to	/pɜ:pən'dɪkjʊlə tʊ/
∥	parallel to	/'pærəlel tʊ/
≢	not equivalent to, not identical to	/'nɒt ɪk'wɪvələnt tʊ/ /'nɒt aɪ'dentɪkl tʊ/
≠≠	not similar to	/'nɒt 'sɪmɪlə tʊ/
^{2}	squared	/'skweəd/
^{3}	cubed	/'kju:bd/
^{4}	to the fourth; to the power four	/tə ðə 'fɔ:θ/ /te ðə 'pɑʊə fɔ:/
^{n}	to the n; to the nth; to the power n	/tə ðɪ en; tə dɪ enθ; tə ðə pɑʊər en/
√	root; square root	/ru:t/ /skweə ru:t/
∛	cube root	/kju:b ru:t/
∜	fourth root	/fɔ:θ ru:t/
!	factorial	/fæk'tɔ:rɪəl/
%	percent	/pə'sent/
∞	infinity	/ɪn'fɪnətɪ/
∝	varies as; proportional to	/'vɛərɪz/ /prə'pɔ:ʃənəl/
˙	dot	/dɒt/
¨	double dot	/dʌbl dɒt/
:	is to, ratio of	/reɪʃɪəʊ/
f(x) fx	f; function	/ef/ /'fʌŋkʃən/
f'(x)	f dash; derivative	/dæʃ/ /dɪ'rɪvətɪv/

f"x	f double-dash; second derivative	/'dʌbl dæʃ/ /'sekənd dɪ'rɪvətɪv/
f'''(x)	f triple-dash; f treble-dash; third derivative	/'trɪpl dæʃ/ / trebl dæʃ/ /θɜ:d dɪ'rɪvətɪv/
$f^{(4)}$	f four; fourth derivative	/fɔ:θ dɪ'rɪvətɪv/
∂	partial derivative, delta	/pa:ʃəl dɪ'rɪvətɪv/ /deltə/
$\int$	integral	/'ɪntɪgrəl/
$\sum$	sum	/sʌm/
w.r.t.	with respect to	/wɪð 'rɪspekt/
log	log	/lɒg/
$\log_2 x$	log to the base 2 of x	/lɒg tə ðə beɪs tu: əv eks/
$\therefore$	therefore	/'ðɛəfɔ:/
$\because$	because	/bɪ'kɒz/
$\rightarrow$	gives, leads to, approaches	/gɪvz/ /li:dz tʊ/ /əprəʊʧəz/
/	per	/pɜ:/
$\in$	belongs to; a member of; an element of	/bɪ'lɒŋz/ /'membə/ /'elɪmənt/
$\notin$	does not belong to; is not a member of; is not an element of	/nɒt bɪ'lɒŋ/ /nɒt ə 'membə/ /nɒt ən 'elɪmənt/
$\subset$	contained in; a proper subset of	/kən'teɪnd ɪn/ /'prɒpə 'sʌbset/
$\subseteq$	contained in; subset	/'sʌbset/
$\cap$	intersection	/'ɪntəsekʃən/
$\cup$	union	/'ju:nɪən/
$\forall$	for all	/fə rɔ:l/
cos x	cos x; cosine x	/kɒz/
sin x	sine x	/saɪn/
tan x	tangent x	/tan/
cosec x	cosec x	/'kəʊsek/
sinh x	shine x	/'ʃaɪn/
cosh x	cosh x	/'kɒʃ/
tanh x	than x	/θæn/
|x|	mod x; modulus x	/mɒd/ /'mɒdjʊləs/
℃	degrees Centigrade	/dɪ'gri:z 'sentɪgreɪd/
℉	degrees Fahrenheit	/dɪ'gri:z 'færənhaɪt/
°K	degrees Kelvin	/dɪ'gri:z 'kelvɪn/

0°K, –273.15 °C	absolute zero	/absəluːt ziːrəʊ/
mm	millimetre	/'mɪlɪmiːtə/
cm	centimetre	/'sentɪmiːtə/
cc, cm³	cubic centimetre, centimetre cubed	/'kjuːbɪk 'sentɪmiːtə/ /'sentɪmiːtə 'kjuːbd/
m	metre	/'miːtə/
km	kilometre	/kɪ'lɒmɪtə/
mg	milligram	/'mɪlɪgræm/
g	gram	/græm/
kg	kilogram	/'kɪləgræm/
AC	A.C.	/eɪ siː/
DC	D.C.	/diː siː/

^

Examples

x + 1	x plus one
x -1	x minus one
x ± 1	x plus or minus one
xy	x y; x times y; x multiplied by y
(x — y)(x + y)	x minus y, x plus y
x/y	x over y; x divided by y;
x ÷ y	x divided by y
x = 5	x equals 5; x is equal to 5
x ≈ y	x is approximately equal to y
x ≡ y	x is equivalent to y; x is identical with y
x ≠ y	x is not equal to y
x > y	x is greater than y
x < y	x is less than y
x ≥ y	x is greater than or equal to y
x ≤ y	x is less than or equal to y
0 < x < 1	zero is less than x is less than 1; x is greater than zero and less than 1
0 ≤ x ≤ 1	zero is less than or equal to x is less than or equal to 1; x is greater than

	or equal to zero and less than or equal to 1
x^2	x squared
x^3	x cubed
x^4	x to the fourth; x to the power four
x^n	x to the n; x to the nth; x to the power n
x^{-n}	x to the minus n; x to the power of minus n
$\sqrt{}$	root x; square root x; the square root of x
$\sqrt[3]{}$	the cube root of x
$\sqrt[4]{}$	the fourth root of x
$\sqrt[n]{}$	the nth root of x
$(x + y)^2$	x plus y all squared
$(x/y)^2$	x over y all squared
n!	n factorial; factorial n
x%	x percent
∞	infinity
$x \propto y$	x varies as y; x is (directly) proportional to y
$x \propto 1/y$	x varies as one over y; x is indirectly proportional to y
$\dot{x}$	x dot
$\ddot{x}$	x double dot
f(x) fx	f of x; the function of x
f'(x)	f dash x; the (first) derivative of with respect to x
f"x	f double-dash x; the second derivative of f with respect to x
f"'(x)	f triple-dash x; f treble-dash x; the third derivative of f with respect to x
$f^{(4)}$	f four x; the fourth derivative of f with respect to x
∂v	the partial derivative of v
$\frac{\partial v}{\partial \theta}$	delta v by delta theta, the partial derivative of v with respect to θ
$\frac{\partial^2 v}{\partial \theta^2}$	delta two v by delta theta squared; the second partial derivative of v with respect to θ
dv	the derivative of v
$\frac{dv}{d\theta}$	d v by d theta, the derivative of v with respect to theta
$\frac{d^2 v}{d\theta^2}$	d 2 v by d theta squared, the second derivative of v with respect to theta,

$\int$	integral
$\int_0^\infty$	integral from zero to infinity
$\sum$	sum
$\sum_{i=1}^{n}$	the sum from i equals 1 to n
w.r.t.	with respect to
$\log_e y$	log to the base e of y; log y to the base e; natural log (of) y
$\therefore$	therefore
$\because$	because
$\rightarrow$	gives, approaches
$\Delta x \rightarrow 0$	delta x approaches zero
$\lim_{\Delta x \rightarrow 0}$	the limit as delta x approaches zero, the limit as delta x tends to zero
$\mathrm{Lt}_{\Delta x \rightarrow 0}$	the limit as delta x approaches zero, the limit as delta x tends to zero
m/sec	metres per second
$x \in A$	x belongs to A; x is a member of A; x is an element of A
$x \notin A$	x does not belong to A; x is not a member of A; x is not an element of A
$A \subset B$	A is contained in B; A is a proper subset of B
$A \subseteq B$	A is contained in B; A is a subset of B
$A \cap B$	A intersection B
$A \cup B$	A union B
cos x	cos x; cosine x
sin x	sine x
tan x	tangent x, tan x
cosec x	cosec x
sinh x	shine x
cosh x	cosh x
tanh x	than x
$\|x\|$	mod x; modulus x
18 ℃	eighteen degrees Centigrade
70 ℉	seventy degrees Fahrenheit

Greek alphabet

Α	α	alpha	/'ælfə/
Β	β	beta	/'biːtə/
Γ	γ	gamma	/'gæmə/
Δ	δ	delta	/'deltə/
Ε	ε	epsilon	/'epsilən/
Ζ	ζ	zeta	/'ziːtə/
Η	η	eta	/'iːtə/
Θ	θ	theta	/'θiːtə/
Ι	ι	iota	/aɪ'əʊtə/
Κ	κ	kappa	/'kæpə/
Λ	λ	lamda	/'læmdə/
Μ	μ	mu	/'mjuː/
Ν	ν	nu	/'njuː/
Ξ	ξ	xi	/'ksaɪ/
Ο	ο	omicron	/'əʊmɪkrən/
Π	π	pi	/'paɪ/
Ρ	ρς	rho	/'rəʊ/
Σ	σ	sigma	/'sɪgmə/
Τ	τ	tau	/'tɑʊ/
Υ	υ	upsilon	/'jʊpsɪlən/
Φ	φ	phi	/'faɪ/
Χ	χ	chi	/'kaɪ/
Ψ	ψ	psi	/'psaɪ/
Ω	ω	omega	/'əʊmɪgə/

Roman alphabet

A	a	/'eɪ/
B	b	/'biː/
C	c	/'siː/
D	d	/'diː/
E	e	/'iː/
F	f	/'ef/
G	g	/'ʤiː/
H	h	/'eɪʧ/

I	i	/'aɪ/
J	j	/'ʤeɪ/
K	k	/'keɪ/
L	l	/'el/
M	m	/'em/
N	n	/'en/
O	o	/'əʊ/
P	p	/'piː/
Q	q	/'kjuː/
R	r	/'ɑː/
S	s	/'es/
T	t	/'tiː/
U	u	/'juː/
V	v	/'viː/
W	w	/'dʌbljuː/
X	x	/'eks/
Y	y	/'waɪ/
Z	z	/'zed/

Fractions

½	a half	/ə 'hɑːf/
¼	a quarter	/ə 'kwɔːtə/
¾	three quarters	/θriː 'kwɔːtəz/
⅓	a third	/ə 'θɜːd/
⅔	two thirds	/tuː 'θɜːdz/
⅕	a fifth	/ə 'fɪfθ/
⅖	two fifths	/tuː 'fɪfθs/
⅗	three fifths	/θriː 'fɪfθs/
⅘	four fifths	/fɔː 'fɪfθs/
⅙	a sixth	/ə 'sɪksθ/
⅚	five sixths	/faɪv 'sɪksθs/
⅛	an eighth	/ən 'eɪtθ/

⅜	three eighths	/θri: 'eɪtθs/
⅝	five eighths	/faɪv 'eɪtθs/
⅞	seven eighths	/sevən 'eɪtθs/

Decimal Fractions

0.1	nought point one	/nɔ:t pɔɪnt wʌn/
0.01	nought point oh one	/nɔ:t pɔɪnt əʊ wʌn/
0.0001	nought point oh oh oh one	/ten pɔɪnt əʊ əʊ əʊ wʌn/
1.1	one point one	/wʌn pɔɪnt wʌn/
1.2	one point two	/wʌn pɔɪnt tu:/
1.23	one point two three	/wʌn pɔɪnt tu: θri:/
1.0123	one point oh one two three	/wʌn pɔɪnt əʊ wʌn tu: θri:/
10.01	ten point oh one	/ten pɔɪnt əʊ wʌn/
21.57	twenty-one point five seven	/'twentɪ wʌn pɔɪnt faɪv 'sevən/
2.6666666666....	two point six recurring	/tu: pɔɪnt sɪks rɪ'kɜ:rɪŋ/
2.612361236123...	two point six one two three recurring	/tu: pɔɪnt sɪks wʌn tu: θri: rɪ'kɜ:rɪŋ/
2.5 million	two point five million	/tu: pɔɪnt faɪv 'mɪljən/

SI Units: Prefixes

10^{-24}	yocto	y	/'jɒktəʊ/
10^{-21}	zepto	z	/'zeptəʊ/
10^{-18}	atto	a	/'atəʊ/
10^{-15}	femto	f	/'femtəʊ/
10^{-12}	pico	p	/'pi:kəʊ/
10^{-9}	nano	n	/'nanəʊ/
10^{-6}	micro	µ	/'maɪkrəʊ/
10^{-3}	milli	m	/'mɪlɪ/
10^{-2}	centi	c	/'sentɪ/
10^{-1}	deci	d	/'desɪ/
10^{3}	kilo	k	/'kɪləʊ/
10^{6}	mega	M	/'megə/

10^{9}	giga	G	/'gɪgə/
10^{12}	tera	T	/'terə/
10^{15}	peta	P	/'petə/
10^{18}	exa	E	/'eksə/
10^{21}	zetta	Z	/'zetə/
10^{24}	yotta	Y	/'jɒtə/
10^{27}	xona	X	/'zəʊnə/
10^{30}	weka	W	/'wekə/
10^{33}	vunda	V	/'vɒndə/

APPENDIX 2

HOW TO RENDER THE ARTICLE

1. Headline / Title of the article	The article is headlined... The headline of the article is... The article goes under the headline... The article under the headline... has the subhead... The title of the article is... The article is entitled...
2. Place of origin	The article is (was) printed / published in... The article is from a newspaper under the nameplate...
3. Time of origin	The publication date of the article is... The article is dated the first of October 2008. The article is printed on the second of October, 2008.
4. Author	The article is written by…

	The author of the article is... The article is written by a group of authors. They are…
5. Theme / Topic	The article is about... The article is devoted to... The article deals with the topic... The basic subject matter of the script is... The article touches upon the topic of... The article addresses the problem of... The article raises/brings up the problem... The article describes the situation... The article assesses the situation... The article informs us about… / comments on… The headline of the article corresponds to the topic.
6. Main idea / Aim of the article	The main idea of the article is… The purpose of the article / author is to give the reader some information on... The aim of the article / author is – to provide the reader with some information about...; – to provide the reader with some material / data on... – to inform about...; – to compare / determine...;
7. Contents of the article (a short summary of 3 or 4 sentences) + important FACTS, NAMES, FIGURES.	The article can be divided into some parts. The first part deals with... The second covers the events... The third touches upon the problem of... The fourth part includes some interviews, dialogues, pictures, reviews, references, quotations, figures. The article is written in the form of the monologue, from the first / third person narration. The author starts by telling the reader that... (writes, states, stresses, depicts, says, informs, underlines, confirms, emphasizes, puts an accent on, accepts / denies the fact, reports, resorts to, hints on, inclines to, points out... and so on) Later the article / the author describes...

	The article / the author goes on to say that... According to the text... In conclusion... The author comes to the conclusion / concludes that… The key sentence / words of the article (is / are) the following...
8. Vocabulary of the article – the topical vocabulary – the author's vocabulary	While reading I've come across some topical words and expressions like.../ A great number of words belong to the topic… The author's vocabulary is rather vivid, poor, rich… The author resorts to colourful general phrases/ clichés / stable statements / understatements / exaggerations / words with negative / positive connotation / fine words / descriptive adjectives / comparisons (to create a vivid picture, a humorous effect / to enforce the influence on the reader). We see the author's mastery in conveying the main idea to the reader with the help of the phrases / parenthesis / sayings / proverbs
9. Personal opinion / impression of the article	I found the article interesting / important / useful / dull / of no value / (too) hard to understand and assess (Why?) I appreciate the author's word-painting as / superb / ordinary / exaggerated. I think / believe that... My point is that... In my opinion... To my mind...
10. Personal view on the topic / idea / problem	The message of the writer is clear to understand... I share the author's view... I see the problem in a different way... I don't quite agree with the fact (that)...

APPENDIX 3
Texts for translation and rendering

Text 1

Types of data

All data in a program has an associated type. Internally, all data is stored just as a sequence of bits, so the type of the data is important to understand what it means. We have seen several different types of data already: Numbers, Booleans, and Procedures (we use initial capital letters to signify a datatype).

A datatype defines a set (often infinite) of possible values. The Boolean datatype contains the two Boolean values, true and false. The Number type includes the infinite set of all whole numbers (it also includes negative numbers

and rational numbers). We think of the set of possible Numbers as infinite, even though on any particular computer there is some limit to the amount of memory available, and hence, some largest number that can be represented. On any real computer, the number of possible values of any data type is always finite. But, we can imagine a computer large enough to represent any given number.

The type of a value determines what can be done with it. For example, a Number can be used as one of the inputs to the primitive procedures +,*, and =. A Boolean can be used as the first subexpression of an if expression and as the input to the not procedure (—not— can also take a Number as its input, but for all Number value inputs the output is false), but cannot be used as the input to +, *, or =.

A Procedure can be the first subexpression in an application expression. There are infinitely many different types of Procedures, since the type of a Procedure depends on its input and output types. For example, recall bigger procedure from Chapter 3:

(**define** (*bigger a b)* **(if** *(> a b) a b))*

It takes two Numbers as input and produces a Number as output. We denote this type as:

Number Number Number

The inputs to the procedure are shown on the left side of the arrow. The type of each input is shown in order, separated by the _ symbol. The output type is given on the right side of the arrow.

From its definition, it is clear that the bigger procedure takes two inputs from its parameter list. How do we know the inputs must be Numbers and the output is a Number?

The body of the bigger procedure is an if expression with the predicate expression (> a b). This applies the > primitive procedure to the two inputs. The type of the > procedure is Number _ Number ! Boolean. So, for the predicate expression to be valid, its inputs must both be Numbers. This means the input values to bigger must both be Numbers. We know the output of the bigger

procedure will be a Number by analyzing the consequent and alternate subexpressions: each evaluates to one of the input values, which must be a Number. Starting with the primitive Boolean, Number, and Procedure types, we can build arbitrarily complex datatypes.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 2

History of Computing Machines

The goal of early machines was to carry out some physical process with less effort than would be required by a human. These machines took physical things as inputs, performed physical actions on those things, and produced some physical output. For instance, a cotton gin takes as input raw cotton, mechanically separates the cotton seed and lint, and produces the separated products as output.

The first big leap toward computing machines was the development of machines whose purpose is abstract rather than physical. Instead of producing physical things, these machines used physical things to represent information. The output of the machine is valuable because it can be interpreted as information, not for its direct physical effect.

Our first example is not a machine, but using fingers to count. The base ten number system used by most human cultures reflects using our ten fingers for counting. Successful shepherds needed to find ways to count higher than ten. Shepherds used stones to represent numbers, making the cognitive leap of using a physical stone to represent some quantity of sheep. A shepherd would count sheep by holding stones in his hand that represent the number of sheep.

More complex societies required more counting and more advanced calculating. The Inca civilization in Peru used knots in collections of strings known as *khipu* to keep track of thousands of items for a hierarchical system of taxation. Many cultures developed forms of abaci, including the ancient Mesopotamians and Romans. An abacus performs calculations by moving beads on rods. The Chinese suan pan (“calculating plate”) is an abacus with a beam subdividing Suan Pan the

rods, typically with two beads above the bar (each representing 5), and five beads below the beam (each representing 1). An operator can perform addition, subtraction, multiplication, and division by following mechanical processes using an abacus.

All of these machines require humans to move parts to perform calculations. As machine technology improved, automatic calculating machines were built where the operator only needed to set up the inputs and then turn a crank or use some external power source to perform the calculation. The first automatic calculating machine to be widely demonstrated was the Pascaline, built by then nineteen-year old French mathematician Blaise Pascal (also responsible for Pascal's triangle from Exploration 5.1) to replace the tedious calculations he had to do to manage his father's accounts. The Pascaline had five wheels, each representing one digit of a number, linked by gears to perform addition with carries. Gottfried Wilhelm von Leibniz built the first machine capable of performing all four basic arithmetic operations (addition, subtraction, multiplication, and division) fully mechanically in 1694.

Over the following centuries, more sophisticated mechanical calculating machines were developed but these machines could still only perform one operation at a time. Performing a series of calculations was a tedious and error-prone process in which a human operator had to set up the machine for each arithmetic operation, record the result, and reset the machine for the next calculation.

The big breakthrough was the conceptual leap of programmability. A machine is programmable if its inputs not only control the values it operates on, but the operations it performs.

The first programmable computing machine was envisioned (but never successfully built) in the 1830s by Charles Babbage. Babbage was born in London in 1791 and studied mathematics at Cambridge. In the 1800s, calculations were done by looking up values in large books of mathematical and astronomical tables. These tables were computed by hand, and often contained errors. The calculations

were especially important for astronomical navigation, and when the values were incorrect a ship would miscalculate its position at sea (sometimes with tragic consequences).

Babbage sought to develop a machine to mechanize the calculations to compute these tables. Starting in 1822, he designed a steam-powered machine known as the Difference Engine to compute polynomials needed for astronomical calculations using Newton's method of successive differences (a generalization of Heron's method from Exploration 4.1). The Difference Engine was never fully completed, but led Babbage to envision a more general calculating machine.

This new machine, the Analytical Engine, designed between 1833 and 1844, was the first general-purpose computer envisioned. It was designed so that it could be programmed to perform any calculation. One breakthrough in Babbage's design was to feed the machine's outputs back into its inputs. This meant the engine could perform calculations with an arbitrary number of steps by cycling outputs back through the machine.

The Analytical Engine was programmed using punch cards, based on the cards that were used by Jacquard looms. Each card could describe an instruction such as loading a number into a variable in the store, moving values, performing arithmetic operations on the values in the store, and, most interestingly, jumping forward and backwards in the instruction cards. The Analytical Engine supported conditional jumps where the jump would be taken depending on the state of a lever in the machine (this is essentially a simple form of the if expression).

In 1842, Charles Babbage visited Italy and described the Analytical Engine to Luigi Menabrea, an Italian engineer, military officer, and mathematician who would later become Prime Minister of Italy. Menabrea published a description of Babbage's lectures in French. Ada Augusta Byron King (also known as Ada, Countess of Lovelace) translated the article into English.

In addition to the translation, Ada added a series of notes to the article. The notes included a program to compute Bernoulli numbers, the first detailed program for the Analytical Engine. Ada was the first to realize the importance and interest

in creating the programs themselves, and envisioned how programs could be used to do much more than just calculate mathematical functions. This was the first computer program ever described, and Ada is recognized as the first computer programmer.

Despite Babbage's design, and Ada's vision, the Analytical Engine was never completed. It is unclear whether the main reason for the failure to build a working Analytical Engine was due to limitations of the mechanical components available at the time, or due to Babbage's inability to work with his engineer collaborator or to secure continued funding.

The first working programmable computers would not appear for nearly a hundred years. Advances in electronics enabled more reliable and faster components than the mechanical components used by Babbage, and the desperation brought on by World War II spurred the funding and efforts that led to working general-purpose computing machines.

The remaining conceptual leap is to treat the program itself as data. In Babbage's Analytical Engine, the program is a stack of cards and the data are numbers stored in the machine. The machine cannot alter its own program.

The idea of treating the program as just another kind of data the machine can process was developed in theory by Alan Turing in the 1930s, and first implemented by the Manchester Small-Scale Experimental Machine (built by a team at Victoria University in Manchester) in 1948.

This computer (and all general-purpose computers in use today) stores the program itself in the machine's memory. Thus, the computer can create new programs by writing into its own memory. This power to change its own program is what makes stored-program computers so versatile.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 3

Running Time

We want a measure of the running time of a procedure that satisfies two properties: (1) it should be robust to ephemeral properties of a particular execution or computer, and (2) it should provide insights into how long it takes evaluate the procedure on a wide range of inputs.

To estimate the running time of an evaluation, we use the number of steps required to perform the evaluation. The actual number of steps depends on the details of how much work can be done on each step. For any particular processor, both the time it takes to perform a step and the amount of work that can be done in one step varies. When we analyze procedures, however, we usually don't want to deal with these details. Instead, what we care about is how the running time changes as the input size increases. This means we can count anything we want as a "step" as long as each step is the approximately same size and the time a step requires does not depend on the size of the input.

The clearest and simplest definition of a step is to use one Turing Machine step. We have a precise definition of exactly what a Turing Machine can do in one step: it can read the symbol in the current square, write a symbol into that square, transition its internal state number, and move one square to the left or right. Counting Turing Machine steps is very precise, but difficult because we do not usually start with a Turing Machine description of a procedure and creating one tedious.

Instead, we usually reason directly from a Scheme procedure (or any precise description of a procedure) using larger steps. As long as we can claim that whatever we consider a step could be simulated using a constant number of steps on a Turing Machine, our larger steps will produce the same answer within the asymptotic operators. One possibility is to count the number of times an evaluation rule is used in an evaluation of an application of the procedure. The amount of work in each evaluation rule may vary slightly (for example, the evaluation rule for an if expression seems more complex than the rule for a primitive) but does not depend on the input size.

Hence, it is reasonable to assume all the evaluation rules to take constant time. This does not include any additional evaluation rules that are needed to apply one rule. For example, the evaluation rule for application expressions includes evaluating every subexpression. Evaluating an application constitutes one work unit for the application rule itself, plus all the work required to evaluate the subexpressions. In cases where the bigger steps are unclear, we can always return to our precise definition of a step as one step of a Turing Machine.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 4

Binary Search

If the data to search is structured, it may be possible to find an element that satisfies some property without examining all elements. Suppose the input data is a sorted binary tree, as introduced in Section 8.1.4. Then, with a single comparison we can determine if the element we are searching for would be in the left or right subtree. Instead of eliminating just one element with each application of the matching function as was the case with *list-search*, with a sorted binary tree a single application of the comparison function is enough to exclude approximately half the elements.

The *binary-tree-search* procedure takes a sorted binary tree and two procedures as its inputs. The first procedure determines when a satisfying element has been found (we call this the *ef* procedure, suggesting equality). The second procedure, *cf* , determines whether to search the left or right subtree. Since *cf* is used to traverse the tree, the input tree must be sorted by *cf* .

```
(define (binary-tree-search ef cf tree) ; requires: tree is sorted by cf
  (if (null? tree) false
      (if (ef (tree-element tree)) (tree-element tree)
          (if (cf (tree-element tree))
              (binary-tree-search ef cf (tree-left tree))
              (binary-tree-search ef cf (tree-right tree))))))
```

For example, we can search for a number in a sorted binary tree using = as the equality function and < as the comparison function:

```
(define (binary-tree-number-search tree target)
  (binary-tree-search (lambda (el) (= target el))
                      (lambda (el) (< target el))
                      tree))
```

To analyze the running time of *binary-tree-search*, we need to determine the number of recursive calls. Like our analysis of *list-sort-tree*, we assume the input tree is well-balanced. If not, all the elements could be in the right branch, for example, and *binary-tree-search* becomes like *list-search* in the pathological case.

If the tree is well-balanced, each recursive call approximately halves the number of elements in the input tree since it passed in either the left or right subtree. Hence, the number of calls needed to reach a null tree is in Θ(log n) where n is the number of elements in the input tree. This is the depth of the tree: binary-tree-search traverses one path from the root through the tree until either reaching an element that satisfies the *ef* function, or reaching a *null* node.

Assuming the procedures passed as ef and cf have constant running time, the work for each call is constant except for the recursive call. Hence, the total running time for *binary-tree-search* is in Θ(log n) where n is the number of elements in the input tree. This is a huge improvement over linear searching: with linear search, doubling the number of elements in the input doubles the search time; with binary search, doubling the input size only increases the search time by a constant.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 5

Python programming language

We could implement a Charme interpreter using Scheme or any other universal programming language, but implement it using the programming

language Python. Python is a popular programming language initially designed by Guido van Rossum in 1991.Python is freely available from http://www.python.org.

We use Python instead of Scheme to implement our Charme interpreter for a few reasons. The first reason is pedagogical: it is instructive to learn new languages. As Dijkstra's quote at the beginning of this chapter observes, the languages we use have a profound effect on how we think. This is true for natural languages, but also true for programming languages. Different languages make different styles of programming more convenient, and it is important for every programmer to be familiar with several different styles of programming. All of the major concepts we have covered so far apply to Python nearly identically to how they apply to Scheme, but seeing them in the context of a different language should make it clearer what the fundamental concepts are and what are artifacts of a particular programming language.

Another reason for using Python is that it provides some features that enhance expressiveness that are not available in Scheme. These include built-in support for objects and imperative control structures. Python is also well-supported by most web servers (including Apache), and is widely used to develop dynamic web applications.

The grammar for Python is quite different from the Scheme grammar, so Python programs look very different from Scheme programs. The evaluation rules, however, are quite similar to the evaluation rules for Scheme. This chapter does not describe the entire Python language, but introduces the grammar rules and evaluation rules for the most important Python constructs as we use them to implement the Charme interpreter.

Like Scheme, Python is a *universal programming language*. Both languages can express all mechanical computations. For any computation we can express in Scheme, there is a Python program that defines the same computation. Conversely, every Python program has an equivalent Scheme program.

One piece of evidence that every Scheme program has an equivalent Python program is the interpreter we develop in this chapter. Since we can implement an

interpreter for a Scheme-like language in Python, we know we can express every computation that can be expressed by a program in that language with an equivalent Python program: the Charme interpreter with the Charme program as its input.

Tokenizing. We introduce Python using one of the procedures in our interpreter implementation. We divide the job of parsing into two procedures that are combined to solve the problem of transforming an input string into a list describing the input program's structure. The first part is the tokenizer. It takes as input a string representing a Charme program, and outputs a list of the tokens in that string.

A token is an indivisible syntactic unit. For example, the Charme expression, (define square (lambda (x) (* x x))), contains 15 tokens: (, define, square, (,lambda, (, x,), (, *, x, x,),), and). Tokens are separated by whitespace (spaces, tabs, and newlines). Punctuation marks such as the left and right parentheses are tokens by themselves.

The tokenize procedure below takes as input a string s in the Charme target language, and produces as output a list of the tokens in s. We describe the Python language constructs it uses next.

```
def tokenize(s): #                  # starts a comment until the end of the line
  current = '' #       initialize current to the empty string (two single quotes)
  tokens = [] #                          initialize tokens to the empty list
  for c in s:  #                        for each character, c, in the string s
    if c.isspace():  #                                  if c is a whitespace
      if len(current) > 0: #                 if the current token is non-empty
      tokens.append(current) #                                add it to the list
      current = '' #                        reset current token to empty string
    elif c in '()': #                        otherwise, if c is a parenthesis
      if len(current) > 0: #                                end the current token
        tokens.append(current) #                    add it to the tokens list
        current = '' #                and reset current to the empty string
```

```
            tokens.append(c) #                add the parenthesis to the token list
        else: #                               otherwise (it is an alphanumeric)
            current = current + c #           add the character to the current token
    # end of the for loop                     reached the end of s
    if len(current) > 0: #                    if there is a current token
        tokens.append(current) #              add it to the token list
    return tokens #                           the result is the list of tokens
```

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 6

Gődel's Incompleteness Theorem

Kurt Gödel was born in Brno (then in Austria-Hungary, now in the Czech Republic) in 1906. Gödel proved that the axiomatic system in Principia Mathematica could not be complete and consistent. More generally, Gödel showed that no powerful axiomatic system could be both complete and consistent: no matter what the axiomatic system is, if it is powerful enough to express a notion of proof, it must also be the case that there exist statements that can be expressed in the system but cannot be proven either true or false within the system.

Gödel's proof used construction: to prove that Principia Mathematica contains statements which cannot be proven either true or false, it is enough to find one such statement. The statement Gödel found:

Gpm: Statement Gpm does not have any proof in the system of Principia Mathematica.

Similarly to Russel's Paradox, this statement leads to a contradiction. It makes no sense for Gpm to be either true or false:

Statement Gmp is provable in the system.

If Gmp is proven, then it means Gpm does have a proof, but Gpm stated that Gpm has no proof. The system is inconsistent: it can be used to prove a statement that is not true.

Statement Gpm is not provable in the system.

Since Gpm cannot be proven in the system, Gpm is a true statement. The system is incomplete: we have a true statement that is not provable in the system.

The proof generalizes to any axiomatic system, powerful enough to express a corresponding statement G:

G: Statement G does not have any proof in the system.

For the proof to be valid, it is necessary to show that statement G can be expressed in the system.

To express G formally, we need to consider what it means for a statement to not have any proof in the system. A proof of the statement G is a sequence of steps, T0, T1, T2, . . ., TN. Each step is the set of all statements that have been proven true so far. Initially, T0 is the set of axioms in the system. To be a proof of G, TN must contain G. To be a valid proof, each step should be producible from the previous step by applying one of the inference rules to statements from the previous step.

To express statement G an axiomatic system needs to be powerful enough to express the notion that a valid proof does not exist. Gödel showed that such a statement could be constructed using the Principia Mathematica system, and using any system powerful enough to be able to express interesting properties. That is, in order for an axiomatic system to be complete and consistent, it must be so weak that it is not possible to express this statement has no proof in the system.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 7

Ontogeny Recapitulates Phytogeny

After Charles Darwin's book The Origin of the Species was published, the German zoologist Ernst Haeckel stated that "Ontogeny Recapitulates Phylogeny." By this he meant that the development of an embryo (ontogeny) repeats (i.e., recapitu1ates) the evolution of the species (phy1ogeny). In other words, after

fertilization, a human egg goes through stages of being a fish, a pig, and so on before turning into a human baby. Modern biologists regard this as a gross simplification, but it still has a kernel of truth in it.

Something analogous has happened in the computer industry. Each new species (mainframe, minicomputer, personal computer, embedded computer, smart card, etc.) seems to go through the development that its ancestors did. The first mainframes were programmed entirely in assembly language. Even complex programs, like compilers and operating systems, were written in assembler. By the time minicomputers appeared on the scene, FORTRAN, COBOL, and other high-level languages were common on mainframes, but the new minicomputers were nevertheless programmed in assembler (for lack of memory). When microcomputers (early personal computers) were invented, they, too, were programmed in assembler, even though by then minicomputers were also programmed in high-level languages. Palmtop computers also started with assembly code but quickly moved on to high-level languages (mostly because the development work was done on bigger machines). The same is true for smart cards.

Now let us look at operating systems. The first mainframes initially had no protection that handled one manually-loaded program at a time. Later they acquired the hardware and operating system support to handle multiple programs at once, and then full timesharing capabilities.

When minicomputers first appeared, they also had no protection hardware and ran one manually-loaded program at a time, even though multiprogramming was well established in the mainframe world by then. Gradually, they acquired protection hardware and the ability to run two or more programs at once. The first microcomputers were also capable of running only one program at a time, but later acquired the ability to multiprogram. Palmtops and smart cards went the same route.

Disks first appeared on large mainframes, then on minicomputers, microcomputers, and so on down the line. Even now, smart cards do not have hard

disks, but with the advent of flash ROM, they will soon have the equivalent of it. When disks first appeared, primitive file systems sprung up. On the CDC 6600, easily the most powerful mainframe in the world during much of the 1960s, the file system consisted of users having the ability to create a file and then declare it to be permanent, meaning it stayed on the disk even after the creating program exited. To access such a file later, a program had to attach it with a special command and give its password (supplied when the file was made permanent). In effect, there was a single directory shared by all users. It was up to the users to avoid file name conflicts. Early minicomputer file systems had a single directory shared by all users and so did early microcomputer file systems.

Virtual memory (the ability to run programs larger than the physical memory) had a similar development. It first appeared in mainframes, minicomputers, microcomputers and gradually worked its way down to smaller and smaller systems. Networking had a similar history.

In all cases, the software development was dictated by the technology. The first microcomputers, for example, had something like 4 KB of memory and no protection hardware. High-level languages and multiprogramming were simply too much for such a tiny system to handle. As the microcomputers evolved into modern personal computers, they acquired the necessary hardware and then the necessary software to handle more advanced features. It is likely that this development will continue for years to come. Other fields may also have this wheel of reincarnation, but in the computer industry it seems to spin faster.

1.7.1 Monolithic Systems

By far the most common organization, this approach might well be subtitled "The Big Mess." The structure is that there is no structure. The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs.

To construct the actual object program of the operating system when this approach is used, one first compiles all the individual procedures, or files containing the procedures, and then binds them all together into a single object file using the system linker. In terms of information hiding, there is essentially none—every procedure is visible to every other procedure (as opposed to a structure containing modules or packages, in which much of the information is hidden away inside modules, and only the officially designated entry points can be called from outside the module).

Even in monolithic systems, however, it is possible to have at least a little structure. The services (system calls) provided by the operating system are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction. This instruction switches the machine from user mode to kernel mode and transfers control to the operating system. The operating system then fetches the parameters and determines which system call is to be carried out. After that, it indexes into a table that contains in slot k a pointer to the procedure that carries out system call k.

This organization suggests a basic structure for the operating system:

1. A main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

In this model, for each system call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs.

(Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011)

Text 8

Layered Systems

A generalization of the approach of Fig. 1-24 is to organize the operating system as a hierarchy of layers, each one constructed upon the one below it. The

first system constructed in this way was the THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E. W. Dijkstra (1968) and his students. The THE system was a simple batch system for a Dutch computer, the Electrologica X8, which had 32K of 27-bit words (bits were expensive back then).

The system had 6 layers, as shown in Fig. 1-25. Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired. Above layer 0, the system consisted of sequential processes, each of which could tie programmed without having to worry about the fact that multiple processes were running on a single processor. In other words, layer 0 provided the basic multiprogramming of the CPU.

Layer Function

5

The operator

4

User programs

3

Input/output management

2

Operator-process communication

1

Memory and drum management

0

Processor allocation and multiprogramming

Figure 1-25. *Structure of the THE operating system*.

Layer 1 did the memory management. It allocated space for processes in main memory and on a 512K word drum used for holding parts of processes (pages) for which there was no room in main memory. Above layer 1, processes did not have to worry about whether they were in memory or on the drum; the layer 1 software took care of making sure pages were brought into memory whenever they were needed.

Layer 2 handled communication between each process and the operator console. Above layer 1, processes did not have to worry about whether they were in memory or on the drum; the layer 1 software took care of making sure pages were brought into memory whenever they were needed. Layer 2 handled communication between each process and the operator console. Above this layer each process effectively had its own operator console. Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Above layer 3 each process could deal with abstract I/O devices with nice properties, instead of real devices with many peculiarities. Layer 4 was where the user programs were found. They did not have to worry about process, memory, console, or I/O management. The system operator process was located in layer 5.

A further generalization of the layering concept was present in the MULTICS system. Instead of layers, MULTICS was described as having a series of concentric rings, with the inner ones being more privileged than the outer ones (which is effectively the same thing). When a procedure in an outer ring wanted to call a procedure in an inner ring, it had to make the equivalent of a system call, that is, a TRAP instruction whose parameters were carefully checked for validity before allowing the call to proceed. Although the entire operating system was part of the address space of each user process in MULTICS, the hardware made it possible to designate individual procedures (memory segments, actually) as protected against reading, writing, or executing.

Whereas the THE layering scheme was really only a design aid, because all the parts of the system were ultimately linked together into a single object program, in MULTICS, the ring mechanism was very much present at run time and enforced by the hardware. The advantage of the ring mechanism is that it can easily be extended to structure user subsystems. For example, a professor could write a program to test and grade student programs and run this program in ring n , with the student programs running in ring n +1 so that they could not change their grades.

(Tanenbaum Andrew S. Modern Operating Systems/ Prentice Hall PTR. – 976 pp.)

Text 9

Lipschitzian Optimization without the Lipschitz Constant

D. R. Jones, C. D. Pertunnen, B. E. Stuckman

From a theoretical point of view, the Lipschitzian approach to global optimization has always been attractive. By assuming knowledge of a Lipschitz constant (i.e., a bound on the rate of change of the objective function), global search algorithms can be developed and convergence theorems easily proved. Since Lipschitzian methods are deterministic, there is no need for multiple runs. Lipschitzian methods also have few parameters to be specified (besides the Lipschitz constant), and so the need for parameter finite-tuning is minimized. Finally, Lipschitzian methods can place bounds on how far they are from the optimum function value, and hence can use stopping criteria that are more meaningful than a simple iteration limit.

In practice, however, Lipschitzian optimization has three major problems: (i) specifying the Lipschitz constant; (ii) speed of convergence; and (iii) computational complexity in higher dimensions. This paper shows how these problems can be eliminated by modifying the standard approach.

Specifying a Lipschitz constant is a practical problem because a Lipschitz constant may not exist or be easily computed. For example, in optimizing a nonlinear control system, the objective function may be based on a time-consuming simulation or, perhaps, an experiment on the real system. Similarly, in mechanical engineering applications, designs are often evaluated by a lengthy finite-element analysis. In these cases, no closed-form expression for the objective function is available, and so computing a Lipschitz constant is usually difficult or impossible. The new algorithm eliminates the need to specify the Lipschitz constant by carrying out simultaneous searches using all possible constants from zero to infinity. The exact sense in which this is done will become clear later.

The second problem--speed of convergence--is closely related to the first. As we describe later, the Lipschitz constant can be viewed as a weighting parameter

that indicates how much weight to place on global versus local exploration. In standard Lipschitzian methods, this constant is usually large because it must equal (or exceed) the maximum rate of change of the objective function. As a result, these methods place a high emphasis on global search and exhibit slow convergence. In contrast, the new algorithm uses all possible constants, and therefore operates at both the global and local level. Once the global part of the algorithm finds the basin of convergence of the optimum, the local part of the algorithm quickly and automatically exploits it. This is why the new algorithm can converge more quickly than the standard approach.

The third and final problem has to do with computational complexity. When optimizing a function of n variables subject to simple bounds, the search space is a hyperrectangle in n-dimensional Euclidean space. Most previous Lipschitzian algorithms partition this search space into smaller hyperrectangles whose vertices are sampled points. Horst and Tuy review several such methods. To initialize the search, these algorithms must evaluate all 2^n vertices of the search space. The new algorithm cuts through this computational complexity by sampling the midpoint of each hyperrectangle as opposed its vertices. Whatever the number of dimensions, a rectangle can have only one midpoint.

As mentioned above, the new algorithm does not need a Lipschitz constant to determine where to search. But knowledge of a Lipschitz constant can be helpful in determining when to stop searching (e.g., stop when one is certain to be within e of the optimum function value). When a Lipschitz constant is not known, the algorithm stops after a prespecified number of iterations.

The new algorithm has only one parameter that must be specified in addition to the iteration limit. Empirical results suggest that the algorithm is fairly insensitive to this parameter, which can be varied by several orders of magnitude without substantially affecting performance. In contrast, many global search methods have several algorithmic parameters that must be carefully adjusted to ensure good results. One of our goals in developing the new algorithm was to eliminate the need to experiment with such algorithmic parameters.

We call the new algorithm DIRECT. This captures the fact that it is a direct search technique and also is an acronym for dividing rectangles, a key step in the algorithm. We will introduce DIRECT as a modification and extension of a one-dimensional Lipschitzian algorithm due to Shubert. We begin in Section 2 by reviewing Shubert's method and discussing why it is hard to extend it to more than one dimension. Section 3 then modifies Shubert's method to make it tractable in higher dimensions and to eliminate the need to specify a Lipschitz constant. This gives us the one-dimensional DIRECT algorithm. Section 4 extends this one-dimensional algorithm to several dimensions. Section 5 proves convergence. Section 6 compares the performance of DIRECT to other algorithms, and Section 7 summarizes our results.

(Journal of optimization theory and application: Vol. 79, No. 1, October 1993)

Bibliography

1. Above the Clouds: A Berkeley View of Cloud Computing. Michael Armbrust, et al // University of California, Berkeley Technical Report No. UCB/EECS-2009-28 February 10, 2009. [Электронный ресурс] URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf (дата обращения: 22.10.2013)
2. Crandall R., Pomerance C. Prime Numbers. A Computational Perspective . – Springer Science+ Business Media, Inc. – 2005. – 598 pp.
3. Encyclopaedia Britannika [Электронный ресурс] URL: http://www.britannica.com (дата обращения: 22.10.2013)
4. Evans D. Introduction to Computing/ Explorations in Language, Logic, and Machines. 2011 [Электронный ресурс] URL: http://computingbook.org (дата обращения: 22.10.2013)
5. Introduction to grid computing by B.Jacob, et al. IBM Redbooks, 2005. [Электронный ресурс] URL: http://www.redbooks.ibm.com/redbooks/pdfs/sg246778.pdf (дата обращения: 22.10.2013)
6. Oxford Dictionaries [Электронный ресурс] URL: http://www.oxforddictionaries.com/ (дата обращения: 22.10.2013)

7. Tanenbaum Andrew S. Modern Operating Systems/ Prentice Hall PTR. – 976 pp.

8. Tanenbaum Andrew S., Woodhull Albert S. Operating Systems/Design and Implementation/ Prentice Hall PTR. – 1080 pp.

9. Thomas Hahn. Future human computer interaction with special focus on input and output techniques. 2010. [Электронный ресурс] URL: http://www.olafurandri.com/nyti/papers2010/FutureHumanComputerInteraction.pdf (дата обращения: 22.10.2013)