Unit 3

Text A
Computer Software

Computer software, consisting of programs, enables a computer to perform specific tasks, as opposed to its physical components (hardware) which can only do the tasks they are mechanically designed for.[1] The term includes application software such as word processors which perform productive tasks for users, system software such as operating systems, which interface with hardware to run the necessary services for user-interfaces and applications, and middleware which controls and coordinates distributed systems.

1. Relationship to computer hardware

Computer software is so called in contrast to computer hardware, which encompasses the physical interconnections and devices required to store and run the software.[2] In computers, software is loaded into RAM and executed in the central processing unit. At the lowest level, software consists of a machine language specific to an individual processor. A machine language consists of groups of binary values signifying processor instructions (object codes), which change the state of the computer from its preceding state. Software is an ordered sequence of instructions for changing the state of the computer hardware in a particular sequence. It is usually written in high-level programming languages that are easier and more efficient for humans to use (closer to natural language) than machine language. High-level languages are compiled or interpreted into machine language object code. Software may also be written in an assembly language, essentially, a mnemonic representation of a machine language using a natural language alphabet. Assembly language must be assembled into object code via an assembler.

2. Types

Practical computer systems divide software systems into three major classes: system software, programming software and application software, although the distinction is arbitrary, and often blurred.

2.1 System software

System software helps run the computer hardware and computer system. It includes operating systems, device drivers, diagnostic tools, servers, windowing systems, utilities and more. The purpose of

systems software is to insulate the applications programmer as much as possible from the details of the particular computer complex being used, especially memory and other hardware features, and such accessory devices as communications, printers, readers, displays, keyboards, etc.

2.2 Programming software

Programming software usually provides tools to assist a programmer in writing computer programs and software using different programming languages in a more convenient way. The tools include text editors, compilers, interpreters, linkers, debuggers, and so on. An integrated development environment (IDE) merges those tools into a software bundle, and a programmer may not need to type multiple commands for compiling, interpreter, debugging, tracing, and etc., because the IDE usually has an advanced graphical user interface, or GUI.

2.3 Application software

Application software allows end users to accomplish one or more specific (non-computer related) tasks. Typical applications include industrial automation, business software, educational software, medical software, databases, and computer games. Businesses are probably the biggest users of application software, but almost every field of human activity now uses some form of application software. It is used to automate all sorts of functions.

3. Program and library

A program may not be sufficiently complete for execution by a computer. In particular, it may require additional software from a software library in order to be complete. Such a library may include software components used by stand-alone programs, but which cannot work on their own. Thus, programs may include standard routines that are common to many programs, extracted from these libraries. Libraries may also include "stand-alone" programs which are activated by some computer event and/or perform some function (e.g. of computer "housekeeping") but do not return data to their calling program. Programs may be called by one to many other programs.

4. Three layers

Users often see things differently than programmers. People who use modern general purpose computers (as opposed to embedded systems, analog computers, supercomputers, etc.) usually see three layers of software performing a variety of tasks: platform, application, and user software. [3]

4.1 Platform software

Platform includes the firmware, device drivers, an operating system, and typically a graphical user interface which, in total, allow a user to interact with the computer and its peripherals (associated equipment). Platform software often comes bundled with the computer, and users may not realize that it exists or that they have a choice to use different platform software.

4.2 Application software

Application software or applications are what most people think of when they think of software. Typical examples include Office suites and computer games. Application software is often purchased separately from computer hardware. Sometimes applications are bundled with the computer, but that does not change the fact that they run as independent applications. Applications are almost always independent programs from the operating system, though they are often tailored for specific platforms. Most users think of compilers, databases, and other "system software" as applications.

4.3 User-written software

User software tailors systems to meet the users specific needs. User software include spreadsheet templates, word processor macros, scientific simulations, and scripts for graphics and animations. Even E-mail filters are a kind of user software. Users create this software themselves and often overlook how important it is. Depending on how competently the user-written software has been integrated into purchased application packages, many users may not be aware of the distinction between the purchased packages, and what has been added by fellow co-workers. [4]

5. Operation

Computer software has to be "loaded" into the computer's memory. Once the software is loaded, the computer is able to execute the software. Computers operate by executing the computer program. This involves passing instructions from the application software, through the system software, to the hardware which ultimately receives the instruction as machine code. Each instruction causes the computer to carry out an operation — moving data, carrying out a computation, or altering the control flow of instructions.

Data movement is typically from one place in memory to another. Sometimes it involves moving data between memories and registers which enable high-speed data access in the CPU. Moving data, especially large amounts of it, can be costly. So, this is sometimes avoided by using "pointers" to data instead. Computations include simple operations such as incrementing the value of a variable data element. More complex computations may involve many operations and data elements together.

Instructions may be performed sequentially, conditionally, or iteratively. Sequential instructions are those operations that are performed one after another. Conditional instructions are performed such that different sets of instructions execute depending on the value(s) of some data. In some languages this is known as an "if" statement. Iterative instructions are performed repetitively and may depend on some data value. This is sometimes called a "loop". Often, one instruction may "call" another set of instructions that are defined in some other program or module. When more than one computer processor is used, instructions may be executed simultaneously.

A simple example of the way software operates is what happens when a user selects an entry such as "Copy" from a menu. In this case, a conditional instruction is executed to copy text from data in a "document" area residing in memory, perhaps, to an intermediate storage area known as a "clipboard" data area. [5] If a different menu entry such as "Paste" is chosen, the software may execute the instructions to copy the text from the clipboard data area to a specific location in the same or another

document in memory.

Depending on the application, even the example above could become complicated. The field of software engineering endeavors to manage the complexity of how software operates. This is especially true for software that operates in the context of a large or powerful computer system.

Currently, almost the only limitations on the use of computer software in applications is the ingenuity of the designer/programmer. Consequently, large areas of activities (such as playing grand master level chess) formerly assumed to be incapable of software simulation are now routinely programmed. The only area that has so far proved reasonably secure from software simulation is the realm of human art, especially pleasing music and literature.

Kinds of software by operation: computer program as executable, source code or script, configuration.

New Words

enable	[iˈneibl]	*vt.* 使能够
opposed	[əˈpəuzd]	*adj.* 相反的，相对的
mechanically	[miˈkænikəli]	*adv.* 机械地
middleware	[ˈmidlwɛə]	*n.* 中间设备，中间件
coordinate	[kəuˈɔːdineit]	*vt.* 调整，整理
contrast	[ˈkɔntræst]	*vt.* 使与……对比，使与……对照
encompass	[inˈkʌmpəs]	*vt.* 包括，包含
load	[ləud]	*n.* 负荷，负载，加载 *vt.* 装载，装入
individual	[ˌindiˈvidjuəl]	*n.* 个人，个体 *adj.* 个别的，单独的
value	[ˈvæljuː]	*n.* 值，数值
signify	[ˈsignifai]	*v.* 表示，象征
state	[steit]	*n.* 状态，状况，情形
particular	[pəˈtikjulə]	*n.* 细节，详细 *adj.* 特殊的，特别的，独特的；详细的
efficient	[iˈfiʃənt]	*adj.* 有效的
compile	[kəmˈpail]	*vt.* 编译
interpret	[inˈtəːprit]	*v.* 解释
essentially	[iˈsenʃəli]	*adv.* 本质上，基本上
mnemonic	[niˈmɔnik]	*adj.* 记忆的，记忆术的
alphabet	[ˈæːlfəbit]	*n.* 字母表
assembler	[əˈsemblə]	*n.* 汇编程序
practical	[ˈpræktikəl]	*adj.* 实际的，实践的，实用的，应用的
divide	[diˈvaid]	*v.* 分，划分，分开，隔开
distinction	[disˈtiŋkʃən]	*n.* 区别，差别

arbitrary	['ɑ:bitrəri]	*adj.* 武断的，专断的
diagnostic	[ˌdaiəg'nɔstik]	*adj.* 诊断的
utility	[ju:'tiliti]	*n.* 实用程序
insulate	['insjuleit]	*vt.* 使绝缘，隔离
accessory	[æk'sesəri]	*n.* 附件
		adj. 附属的，补充的
communication	[kəˌmju:ni'keiʃən]	*n.* 通信
programmer	['prəugræmə]	*n.* 程序设计员
convenient	[kən'vi:njənt]	*adj.* 便利的，方便的
linker	['liŋkə]	*n.* 连接程序
debugger	[di:'bʌgə]	*n.* 调试程序
merge	[mə:dʒ]	*v.* 合并，并入，融合
bundle	['bʌndl]	*n.* 捆，束，包
accomplish	[ə'kɔmpliʃ]	*vt.* 完成，达到，实现
automation	[ɔ:tə'meiʃən]	*n.* 自动控制，自动操作
database	['deitəbeis]	*n.* 数据库
library	['laibrəri]	*n.* 库
routine	[ru:'ti:n]	*n.* 程序，常规，日常事务
event	[i'vent]	*n.* 事件，活动
housekeeping	['hauski:piŋ]	*n.* 家务管理
calling	['kɔ:liŋ]	*n.* 职业，行业
supercomputer	[ˌsu:pəkəm'pju:tə]	*n.* 巨型计算机
platform	['plætfɔ:m]	*n.* 平台
suite	[sju:t]	*n.* 套件
separately	['sepərətli]	*adv.* 分离地，分别地
independent	[indi'pendənt]	*adj.* 独立的
tailored	['teiləd]	*adj.* 定制的
spreadsheet	['spredʃi:t]	*n.* 电子制表软件，电子数据表
template	['templit]	*n.* 模板
macro	['mækrəu]	*n.* 宏
simulation	[ˌsimju'leiʃən]	*n.* 仿真，模拟
script	[skript]	*n.* 脚本
animation	[æni'meiʃən]	*n.* 动画
E-mail	['i:'meil]	*n.* 电子邮件
filter	['filtə]	*n.* 筛选，过滤器，滤波器
		vt. 过滤
create	[kri:'eit]	*vt.* 创造，创作，创建
package	['pækidʒ]	*n.* 包
execute	['eksikju:t]	*vt.* 执行，实行
avoid	[ə'vɔid]	*vt.* 避免，消除

element	[ˈelimənt]	*n.* 要素，元素，成分，元件
conditionally	[kənˈdiʃnəli]	*adv.* 有条件地
iterative	[ˈitərətiv]	*adj.* 迭代的，反复的，重复的
set	[set]	*n.* 集合
call	[kɔːl]	*v.* 调用
module	[ˈmɔdjuːl]	*n.* 模块
simultaneously	[siməlˈteiniəsly]	*adv.* 同时地
select	[siˈlekt]	*vt.* 选择，挑选
copy	[ˈkɔpi]	*v.* 复制，拷贝
menu	[ˈmenjuː]	*n.* 菜单
document	[ˈdɔkjumənt]	*n.* 文档
clipboard	[ˈklipbɔːd]	*n.* 剪贴板
paste	[peist]	*n.* 粘贴
endeavor	[inˈdevə]	*n. & vi.* 尽力，努力
ingenuity	[ˌindʒiˈnjuːiti]	*n.* 独创性，灵活性
configuration	[kənˌfigjuˈreiʃən]	*n.* 配置，构造，结构

Phrases

word processor	字处理程序
operating system	操作系统，缩写为 OS
machine language	机器语言
object code	目标码
high-level programming language	高级编程语言
assembly language	汇编语言
device drivers	设备驱动程序
text editor	文本编辑程序
end user	终端用户
analog computer	模拟计算机
a variety of	多种的
a kind of	有一点，有几分，稍稍
carry out	执行，完成，实现，贯彻
software engineering	软件工程

Abbreviations

IDE	集成开发环境（Integrated Development Environment）

Notes

[1] Computer software, consisting of programs, enables a computer to perform specific tasks, as opposed to its physical components (hardware) which can only do the tasks they are mechanically designed for.

本句中，consisting of programs 是一个现在分词短语，作 Computer software 的非限定性定语，对其作补充说明。as opposed to 的意思是“ 与……不同；与……相反 ”，its 指 a computer’s。which can only do the tasks they are mechanically designed for 是一个定语从句，修饰和限定 its physical components。在该从句中，they are mechanically designed for 也是一个定语从句，修饰和限定 the tasks。

[2] Computer software is so called in contrast to computer hardware, which encompasses the physical interconnections and devices required to store and run the software.

本句中，which encompasses the physical interconnections and devices required to store and run the software 是一个非定语从句，对 computer hardware 作补充说明。在该从句中，required to store and run the software 是一个过去分词短语，作定语，修饰和限定 devices。in contrast to 的意思是“ 和……形成对比 ”。

[3] People who use modern general-purpose computers (as opposed to embedded systems, analog computers, supercomputers, etc.) usually see three layers of software performing a variety of tasks: platform, application, and user software.

本句中，who use modern general purpose computers 是一个定语从句，修饰和限定 People。as opposed to embedded systems, analog computers, supercomputers, etc.对 modern general-purpose computers 作补充说明。

[4] Depending on how competently the user-written software has been integrated into purchased application packages, many users may not be aware of the distinction between the purchased packages, and what has been added by fellow co-workers.

本句中，Depending on how competently the user-written software has been integrated into purchased application packages 是一个现在分词短语，作条件状语，Depending on 的意思是“ 根据 ”。be aware of 的意思是 “ 了解，知道 ”。

[5] In this case, a conditional instruction is executed to copy text from data in a "document" area residing in memory, perhaps to an intermediate storage area known as a "clipboard" data area.

本句中，residing in memory 是一个现在分词短语，作定语，修饰和限定 a "document" area，known as a "clipboard" data area 是一个过去分词短语，作定语，修饰和限定 an intermediate storage area。in this case 的意思是 “ 在这种情况下 ”。

Exercises

〖Ex. 1〗根据课文内容回答问题

(1) What does computer software consist of? What does it enable a computer to do? What does the term include?

(2) What does a machine language consist of?
(3) What does system software do? What does it include?
(4) What does programming software do? What do the tools include?
(5) What does application software do? What do the typical applications include?
(6) What are the three layers of software people who use modern general purpose computers (as opposed to embedded systems, analog computers, supercomputers, etc.) usually see?
(7) What does platform software include?
(8) What are the typical examples of application software?
(9) What does user software include?
(10) How may instructions be performed? What are sequential instructions, conditional instructions and iterative instructions?

〖Ex. 2〗根据下面的英文解释，写出相应的英文词汇

(1) _____: To transfer (data) from a storage device into a computer's memory.
(2) _____: To translate (a program) into machine language.
(3) _____: An assigned or calculated numerical quantity.
(4) _____: A machine code telling a computer to perform a particular operation.
(5) _____: To translate a statement or instruction into executable from and then execute it.
(6) _____: The letters of a language, arranged in the order fixed by custom.
(7) _____: A program operating on symbolic input data to produce the equivalent executable machine code.
(8) _____: A program designed to perform a particular function; the term usually refers to software that solves narrowly focused problems or those related to computer system management.
(9) _____: The technology employed in transmitting messages.
(10) _____: One who writes computer programs.

〖Ex. 3〗把下列句子翻译为中文

(1) Machine language consists of the raw numbers that can be directly understood by a particular processor.
(2) The actual effect of any particular virus depends on how it was programmed by the person who wrote the virus.
(3) An anti-virus program that hasn't been updated for several months will not provide much protection against current viruses.
(4) Memory is typically measured in kilobytes or megabytes, and disk space is typically measured in megabytes or gigabytes.
(5) The hypertext transfer protocol is the native protocol of browsers and is most typically used to transfer HTML formatted files.
(6) LAN is the extent of network which connects computers that are physically close together, typically within a single room or building.
(7) For mobile users, firewalls allow remote access in to the private network by the use of secure logon

procedures and authentication certificates.

(8) From the Internet users' point of view, to download a file is to request it from another computer (or from a Web page on another computer) and to receive it.

(9) After the Soviet launch of the Sputnik satellite in 1957, the US military set up the Advanced Research Projects Agency to fund research in things sometimes only vaguely related to military matters.

(10) Often a chip set will fit on one chip.

〖Ex. 4〗选择适当的答案填空（真题再现　2006 年 11 月程序员专业英语试题）

(1) _____: An error can be caused by attempting to divide by 0.

A．Interrupt　　B．Default　　C．Underflow　　D．Overflow

(2) _____: The process of identifying and correcting errors in a program.

A．Debug　　B．Bug　　C．Fault　　D．Default

(3) _____: A collection of related information, organized for easy retrieval.

A．Data　　B．Database　　C．Buffer　　D．Stack

(4) _____: A location where data can be temporarily stored.

A．Area　　B．Disk　　C．Buffer　　D．File

(5) _____: A graphical bar with buttons that perform some of the most common commands.

A．Title bar　　B．Tool bar　　C．Status bar　　D．Scroll bar

(6) Every valid character in a computer that uses even _____ must always have an even number of 1bits.

A．parity　　B．check　　C．test　　D．compare

(7) The maximum number of data that can be expressed by 8 bits is _____.

A．64　　B．128　　C．255　　D．256

(8) Integration _____ is the process of verifying that the components of a system work together as described in the program design and system design specifications.

A．trying　　B．checking　　C．testing　　D．coding

(9) GIF files are limited to a maximum of 8 bits/pixel, it simply means that no more than 256 colors are allowed in _____.

A．an image　　B．a file　　C．a window　　D．a page

(10) Computer ____ is a complex consisting of two or more connected computing units, it is used for the purpose of data communication and resource sharing.

A．storage　　B．device　　C．network　　D．processor

〖Ex. 5〗将下列词填入适当的位置（每词只用一次）

customers	directory	related	emphasis	accessible
method	fields	implies	mainframe	information

In data processing, using an office metaphor, a file is a ___(1)___ collection of records. For example, you might put the records you have on each of your ___(2)___ in a file. In turn, each record would consist of ___(3)___ for individual data items, such as customer name, customer number, customer address, and so forth. By providing the same ___(4)___ in the same fields in each record (so

that all records are consistent), your file will be easily ___(5)___ for analysis and manipulation by a computer program. This use of the term has become somewhat less important with the advent of the database and its ___(6)___ on the table as a way of collecting record and field data. In ___(7)___ systems, the term data set is generally synonymous with file but ___(8)___ a specific form of organization recognized by a particular access ___(9)___. Depending on the operating system, files (and data sets) are contained within a catalog, ___(10)___, or folder.

Text B
Computer Programming

Computer programming, often shortened to programming or coding, is the process of writing, testing, and maintaining the source code of computer programs. The source code is written in a programming language. This code may be a modification of existing source or something completely new, the purpose being to create a program that exhibits the desired behavior. The process of writing source code requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.

Within software engineering, programming is regarded as one phase in a software development process.

In some specialist applications or extreme situations a program may be written or modified (known as patching) by directly storing the numeric values of the machine code instructions to be executed into memory.

There is an ongoing debate on the extent to which the writing of programs is an art, a craft or an engineering discipline. Good programming is generally considered to be the measured application of all three: expert knowledge informing an elegant, efficient, and maintainable software solution. The discipline differs from many other technical professions in that programmers generally do not need to be licensed or pass any standardized, or governmentally regulated, certification tests in order to call themselves "programmers" or even "software engineers".

Another ongoing debate is the extent to which the programming language used in writing programs affects the form that the final program takes. This debate is analogous to that surrounding the Sapir-Whorf hypothesis in linguistics.

1. Programmers

Computer programmers are those who write computer software. Their job usually involves:

- Requirements analysis
- Specification
- Software architecture
- Coding
- Compilation
- Software testing

- Documentation
- Integration
- Maintenance

2. Programming languages

Different programming languages support different styles of programming, called programming paradigms. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Ideally the programming language best suited for the task at hand will be selected. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.

3. Modern programming

3.1 Algorithmic complexity

The academic field and engineering practice of computer programming are largely concerned with discovering and implementing the most efficient algorithms for a given class of problem. For this purpose, algorithms are classified into orders using so-called Big O notation, O(n), which expresses execution time, memory consumption, or another parameter in terms of the size of an input. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities, and use this knowledge to consider design trade-offs between, for example, memory consumption and performance.

Research in computer programming includes investigation into the unsolved proposition that P, the class of algorithms which can be deterministically solved in polynomial time with respect to an input, is not equal to NP, the class of algorithms for which no polynomial-time solutions are known. Work has shown that many NP algorithms can be transformed, in polynomial time, into others, such as the Travelling salesman problem, thus establishing a large class of "hard" problems which are for the purposes of analysis, equivalent.

3.2 Methodologies

The first step in every software development project should be requirements analysis, followed by modeling, implementation, and failure elimination or debugging.

There exist a lot of differing approaches for each of those tasks. One approach popular for requirements analysis is Use Case analysis.

Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). The Unified Modeling Language (UML) is a notation used for both OOAD and MDA.

A similar technique used for database design is Entity-Relationship Modeling (ER Modeling).

Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.

Debugging is most often done with IDEs like Visual Studio, and Eclipse. Separate debuggers like gdb are also used.

3.3 Measuring language usage

It is very difficult to determine what the most popular of modern programming languages are. Some languages are very popular for particular kinds of applications (e.g., COBOL is still strong in the corporate data center, often on large mainframes, FORTRAN in engineering applications, and C in embedded applications), while some languages are regularly used to write many different kinds of applications.

Methods of measuring language popularity include: counting the number of job advertisements that mention the language, the number of books teaching the language that are sold (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL).

3.4 Debugging

Debugging is a very important task for every programmer, because an erroneous program is often useless. Languages like C++ and Assembler are very challenging even to expert programmers because of failure modes like Buffer overruns, bad pointers or uninitialized memory. A buffer overrun can damage adjacent memory regions and cause a failure in a totally different program line. Because of those memory issues tools like Valgrind, Purify or Boundschecker are virtually a necessity for modern software development in the C++ language. Languages such as Java, PHP and Python protect the programmer from most of these runtime failure modes, but this may come at the price of a dramatically lower execution speed of the resulting program. This is acceptable for applications where execution speed is determined by other considerations such as database access or file I/O. The exact cost will depend upon specific implementation details. Modern Java virtual machines, for example, use a variety of sophisticated optimizations, including runtime conversion of interpreted instructions to native machine code.

New Words

shorten	['ʃɔ:tn]	*v.* 缩短，（使）变短
test	['test]	*n.* & *v.* 测试
maintain	[mein'tein]	*vt.* 维护，保持，维持
modification	[ˌmɔdifi'keiʃən]	*n.* 更改，修改，修正
exhibit	[ig'zibit]	*vt.* 表现，展现
behavior	[bi'heivjə]	*n.* 举止，行为
expertise	[ekspə:'ti:z]	*n.* 专门技能，专门知识
algorithm	['ælgəriðəm]	*n.* 算法
phase	[feiz]	*n.* 阶段，状态，相，相位
extreme	[iks'tri:m]	*adj.* 极端的，极度的

situation	[ˌsitju'eiʃən]	*n.* 情形，境遇
patch	[pætʃ]	*n.* 补丁
		vt. 修补，补缀
debate	[di'beit]	*v. & n.* 争论，辩论
craft	[krɑ:ft]	*n.* 工艺，手艺
solution	[sə'lju:ʃən]	*n.* 解答，解决办法
licensed	['laisənst]	*adj.* 得到许可的
certification	[sə:tifi'keiʃən]	*n.* 证明，证明书
architecture	['ɑ:kitektʃə]	*n.* 体系机构
style	[stail]	*n.* 风格，类型，字体
paradigm	['pærədaim]	*n.* 范例
consideration	[kənsidə'reiʃən]	*n.* 深思，考虑
preference	['prefərəns]	*n.* 偏爱，优先选择
complexity	[kəm'pleksiti]	*n.* 复杂性
notation	[nəu'teiʃən]	*n.* 记号，符号；注释
parameter	[pə'ræmitə]	*n.* 参数，参量
polynomial	[pɔli'nəumiəl]	*n.* 多项式
equivalent	[i'kwivələnt]	*adj.* 相等的，相当的，同意义的
methodology	[meθə'dɔlədʒi]	*n.* 方法学，方法论
elimination	[ilimi'neiʃən]	*n.* 消除，排除
regularly	['regjuləli]	*adv.* 有规律地，有规则地
overestimate	[ˌəuvə'estimeit]	*vt.* 评价过高
erroneous	[i'rəunjəs]	*adj.* 错误的，不正确的
useless	['ju:slis]	*adj.* 无用的，无效的
overrun	[əuvə'rʌn]	*vt.* 溢出
pointer	['pɔintə]	*n.* 指针
optimization	[ˌɔptimai'zeiʃən]	*n.* 最佳化，最优化
conversion	[kən'və:ʃən]	*n.* 变换，转化

Phrases

source code	源代码
formal logic	形式逻辑
requirements analysis	需求分析
third-party package	第三方软件包
Use Case	用例
Java virtual machine	Java 虚拟机，缩写为 JVM

Abbreviations

OOAD	面向对象分析与设计（Object Oriented Analysis and Design）
MDA	模型驱动体系（Model Driven Architecture）
UML	统一建模语言（Unified Modeling Language）
ER	实体—关系（Entity Relationship）
PHP	超级文本预处理语言（Hypertext Preprocessor）

〖Ex. 6〗根据课文内容，判断正误

(1) Computer programming is often shortened to programming or coding.

(2) The source code is written in a programming language.

(3) In any applications, a program may be written or modified (known as patching) by directly storing the numeric values of the machine code instructions to be executed into memory.

(4) Good programming is generally considered to be the measured application of any of the three: expert knowledge informing an elegant, efficient, and maintainable software solution.

(5) Computer programmers are those who write computer software.

(6) Different programming languages support different styles of programming.

(7) Expert programmers are familiar with only a few well-established algorithms.

(8) In every software development project, requirements analysis should be the first thing.

(9) COBOL is still strong in the corporate data center, often on large mainframes, FORTRAN in embedded applications, and C in engineering applications.

(10) Debugging is a very important task for every programmer, because an erroneous program is often useless.

Reading Material
Computer Programming Basic

1. Compile

To transform a program written in a high-level programming language from source code[1] into object code[2]. Programmers write programs in a form called source code. Source code must go through several steps before it becomes an executable program. The first step is to pass the source code through a compiler, which translates the high-level language instructions into object code.

The final step in producing an executable program — after the compiler has produced object code

[1] 源代码。The source code consists of instructions in a particular language, like C or FORTRAN.

[2] 目标代码

— is to pass the object code through a linker[3]. The linker combines modules and gives real values to all symbolic addresses[4], thereby producing machine code[5].

2. Interpreter

A program that executes instructions written in a high-level language. There are two ways to run programs written in a high-level language. The most common is to compile the program; the other method is to pass the program through an interpreter.

An interpreter translates high-level instructions into an intermediate form, which it then executes. In contrast, a compiler translates high-level instructions directly into machine language. Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, is that it does not need to go through the compilation stage[6] during which machine instructions are generated. This process can be time-consuming[7] if the program is long. The interpreter, on the other hand, can immediately execute high-level programs. For this reason, interpreters are sometimes used during the development of a program, when a programmer wants to add small sections at a time and test them quickly. In addition, interpreters are often used in education because they allow students to program interactively[8].

Both interpreters and compilers are available for most high-level languages. However, BASIC and LISP are especially designed to be executed by an interpreter. In addition, page description languages[9], such as PostScript, use an interpreter. Every PostScript printer, for example, has a built-in interpreter that executes PostScript instructions.

3. Assembler

A programming language that is once removed from a computer's machine language. Machine languages consist entirely of numbers and are almost impossible[10] for humans to read and write. Assembly languages have the same structure and set of commands as machine languages, but they enable a programmer to use names instead of numbers.

Each type of CPU has its own machine language and assembly language, so an assembly language program written for one type of CPU won't run on another. In the early days of programming, all programs were written in assembly language. Now, most programs are written in a high-level language such as FORTRAN or C. Programmers still use assembly language when speed is essential or when they need to perform an operation that isn't possible in a high-level language.

---

[3] linker ['liŋkə]n. （目标代码）连接器

[4] 符号地址，可变地址

[5] 机器代码

[6] stage [steidʒ] n.进程，阶段，时期

[7] time-consuming ['taimkən,sju:miŋ] adj.耗时的

[8] interactive [ˌintər'æktiv] adj.交互式的

[9] 页面描述语言。Abbreviated as PDL, a language for describing the layout and contents of a printed page. The best-known PDLs are Adobe PostScript and Hewlett-Packard PCL (Printer Control Language), both of which are used to control laser printers.

[10] impossible [im'pɔsəbl] adj.不可能的，不会发生的，难以忍受的

4．High level language

A programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages. In contrast, assembly languages are considered low-level because they are very close to machine languages.

The main advantage of high-level languages over low-level languages is that they are easier to read, write, and maintain. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter.

The first high-level programming languages were designed in the 1950s. Now there are dozens of[11] different languages, including Ada, Algol, BASIC, COBOL, C, C++, FORTRAN, LISP, Pascal, and Prolog.

5．Low level languages

A machine language or an assembly language. Low-level languages are closer to the hardware than are high-level programming languages, which are closer to human languages.

6．Middleware

In the computer industry, middleware is a general term for any programming that serves to "glue together" or mediate between two separate and often already existing programs. A common application of middleware is to allow programs written for access to a particular database to access other databases.

Typically, middleware programs provide messaging[12] services so that different applications can communicate. The systematic tying together of disparate applications, often through the use of middleware, is known as enterprise application integration (EAI)[13].

7．Debugger

A debugger or debugging tool is a computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an instruction set simulator (ISS)[14], a technique that allows great power in its ability to halt when specific conditions are encountered[15] but which will typically be somewhat slower than executing the code directly on the appropriate (or the same) processor. Some debuggers offer two modes of operation —— full or partial

---

[11] 许多的

[12] In programming, messaging is the exchange of messages (specially-formatted data describing events, requests, and replies) to a messaging server , which acts as a message exchange program for client programs（客户程序）.

[13] 企业应用系统整合，企业应用系统集成。EAI is a business computing term for the plans, methods, and tools aimed at modernizing, consolidating （巩固）, and coordinating the computer applications in an enterprise.

[14] 指令集仿真器。An instruction set simulator (ISS) is a simulation model, usually coded in a high-level programming language, which mimics the behavior of a mainframe or microprocessor by "reading" instructions and maintaining internal variables which represent the processor's registers.

[15] encounter [in'kauntə] v.遭遇，遇到，相遇

simulation, to limit this impact.

A "crash[16]" happens when the program cannot normally continue because of a programming bug[17]. For example, the program might have tried to use an instruction not available on the current version of the CPU or attempted to access unavailable or protected memory. When the program "crashes" or reaches a preset[18] condition, the debugger typically shows the position in the original code if it is a source-level debugger or symbolic debugger, commonly now seen in integrated development environments. If it is a low-level debugger or a machine-language debugger it shows the line in the disassembly[19] (unless it also has online access to the original source code and can display the appropriate section of code from the assembly or compilation).

Typically, debuggers also offer more sophisticated functions such as running a program step by step (single-stepping[20] or program animation), stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint[21], and tracking the values of some variables. Some debuggers have the ability to modify the state of the program while it is running, rather than merely to observe it. It may also be possible to continue execution at a different location in the program to bypass a crash or logical error.

The importance of a good debugger cannot be overstated[22]. Indeed, the existence and quality of such a tool for a given language and platform can often be the deciding factor in its use, even if another language/platform is better-suited to the task. The absence of a debugger, having once been accustomed to using one, has been said to "make you feel like a blind man in a dark room looking for a black cat that isn't there". However, software can (and often does) behave differently running under a debugger than normally, due to the inevitable changes the presence of a debugger will make to a software program's internal timing. As a result, even with a good debugging tool, it is often very difficult to track down runtime problems in complex multi-threaded[23] or distributed systems.

The same functionality which makes a debugger useful for eliminating bugs allows it to be used as a software cracking tool to evade[24] copy protection, digital rights management, and other software protection features. It often also makes it useful as a general testing verification tool test coverage[25] and performance analyzer, especially if instruction path lengths are shown.

---

[16] A crash (or system crash) in computing is a condition where a computer or a program, either an application or part of the operating system, ceases to function properly, often exiting after encountering errors.

[17] A programming bug is often referred to as a software bug. A software bug is nothing like a pesky worm or virus. Instead, it's an error or mistake that causes a computer program to misbehave. These bugs are generally the result of mistakes made by the programmer either in the design or the source code. Some are caused by compilers that generated invalid code.

[18] preset ['pri:'set] vt.事先设定

[19] disassembly [ˌdisə'sembli] n.反汇编。A disassembler is a computer program that translates machine language into assembly language—the inverse operation to that of an assembler.

[20] 单步执行

[21] breakpoint ['breikpɔint] n.断点。In software development, a breakpoint is an intentional stopping or pausing place in a program, put in place for debugging purposes. It is also sometimes simply referred to as a pause.

[22] overstate ['əuvə'steit] vt.夸大，夸张

[23] 多线程的

[24] evade [i'veid] v.规避，逃避，躲避

[25] 测试覆盖

8. Platform

In computers, a platform is an underlying computer system on which application programs can run. On personal computers, Windows 7 and the Mac OS X are examples of two different platforms. On enterprise servers or mainframes, IBM's S/390 is an example of a platform.

A platform consists of an operating system, the computer system's coordinating program, which in turn is built on the instruction set[26] for a processor or microprocessor, the hardware that performs logic operations and manages data movement in the computer. The operating system must be designed to work with the particular processor's set of instructions. As an example, Microsoft's Windows 7 is built to work with a series of microprocessors from the Intel Corporation that share the same or similar sets of instructions. There are usually other implied parts in any computer platform such as a motherboard and a data bus, but these parts have increasingly become modularized[27] and standardized[28].

Historically, most application programs have had to be written to run on a particular platform. Each platform provided a different application program interface for different system services. Thus, a PC program would have to be written to run on the Windows platform and then again to run on the Mac OS X platform. Although these platform differences continue to exist and there will probably always be proprietary differences between them, new open or standards-conforming interfaces now allow many programs to run on different platforms or to interoperate[29] with different platforms through mediating or "broker[30]" programs.

Reference Translation

Text A 计算机软件

计算机软件由程序组成，使计算机能够执行特定的任务，与之相对的物理部件（硬件）只能机械地执行设计好的任务。这个术语包括如字处理程序这样的应用软件（为用户执行生产性任务）、操作系统这样的系统软件（提供运行用户接口服务和应用软件所必需的硬件接口）以及中间件（管理和协调分布式系统）。

1. 与计算机硬件的关系

计算机软件如此称呼是与计算机硬件相对应的，硬件包括外部连接和存储与运行软件所需的设备。在计算机中，软件被装入 RAM 并在中央处理器中执行。在最低的层次上，软件由一个独立处理器所使用的特定机器语言组成。一个机器语言由表示处理器指令（目标代码）的多组二进制值组成，它们可以改变计算机以前的状态。软件是特定序列的指令，用来按特定顺序改变计算

[26] 指令集

[27] modularize ['mɔdjuləraiz] v.模块化

[28] standardize ['stændədaiz] vt.使标准化

[29] interoperate ['intə'ɔpəreit] v.互通，互用，互操作

[30] broker ['brəukə] n.掮客，经纪人

机硬件的状态。它通常用高级语言编写，人们使用高级语言比机器语言更容易和更有效（因为它更接近自然语言）。高级语言被编译或解释为机器语言目标代码。软件也可用汇编语言编写，从本质上来说，汇编语言是使用自然语言字母的机器语言便于记忆的表示法。汇编语言必须用编译程序编译为目标代码。

2．类型

实用的计算机系统可以把软件系统分为三大主要类型：系统软件、编程软件和应用软件，尽管这样的划分是武断的并且通常是含混不清的。

2.1 系统软件

系统软件可以帮助计算机硬件和计算机系统的运行。它包括操作系统、设备驱动程序、诊断工具、服务器、窗口系统、实用软件等。系统软件的目的是使应用程序员尽可能不考虑特定计算机所使用的细节，特别是内存和其他硬件及附属设备，如通信设备、打印机、阅读机、显示器、键盘等的特点。

2.2 编程软件

通常提供工具，帮助程序员以更方便的方法、采用不同编程语言编写计算机程序及软件。这些工具包括文本编辑程序、编译程序、解释程序、连接程序、调试程序等。集成开发环境把这些工具合成为一个软件包，程序员不再需要为编译、解释、调试、跟踪等任务输入多个命令，因为 IDE 通常有一个高级的图形用户界面，即 GUI。

2.3 应用软件

应用软件允许终端用户完成一个或多个特定的（与计算机无关的）任务。典型的应用包括工业自动化、商业软件、教育软件、医用软件、数据库和计算机游戏。商业大概是应用软件最大的用户群。但几乎人类活动的每个领域现在都使用应用软件。它用于各类功能的自动化。

3．程序和库

一个程序也许不能让计算机十分圆满地执行任务。在特殊情况下，要完全执行也许还需要来自软件库中的附加软件。这样的一个库可能包括独立程序所用的软件部件，但这些不能单独工作。这样，程序可以包含从该库提取的许多程序公用的标准例程。库也可以包含“独立”程序，这些程序可以被某一计算机事件激活或/并执行某一功能（例如计算机的“家务管理”功能），但并不把数据返回给它们的调用程序。程序也可以被一个程序调用给其他许多程序。

4．三个层次

用户看事情的方式通常不同于程序员。使用当今普通计算机（相对于嵌入式系统、模拟计算机、超级计算机等）的人，通常把执行各种任务的软件分为三个层次：平台软件、应用软件和用户软件。

4.1 平台软件

平台包括固件、设备驱动程序、操作系统和特定的图形用户界面。总体来说，图形用户界面

让用户与计算机及其外设（附件）交互。平台软件通常与计算机绑定，用户也许没有感觉它的存在，或者也没有意识到他们可以选择使用其他平台软件。

4.2 应用软件

许多人想到软件时所想的是应用软件或应用。典型的例子包括 Office 套件和计算机游戏。应用软件通常可以与硬件分开购买。有时与计算机绑定，但这并不改变它们作为独立软件运行的事实。应用软件几乎总是独立于操作系统，尽管它们经常是为特定平台而定制的。大部分用户认为编译程序、数据库和其他“系统软件”是应用软件。

4.3 用户编写的软件

用户软件定制系统用于满足用户的特定需要。用户软件包括电子表格模板、字处理程序宏、科学仿真和图形与动画脚本。甚至电子邮件过滤程序也是一种用户软件。用户自己建立这些软件，并往往忽略它们的重要性。就用户把自己编写的软件整合到所购买的软件包中的能力而言，许多用户可能不了解软件包之间的差异，也不知道合作伙伴已经增加了什么东西。

5．运行

计算机软件必须“装入”计算机内存中。一旦软件被装入，计算机就可以执行该软件。计算机通过执行计算机程序来运行。这涉及把来自应用软件的指令经过操作系统传达给最终以机器代码形式接受该指令的硬件。每个指令都会使计算机执行一个操作 —— 移动数据、执行计算或改变指令的控制流。

数据移动是从内存中的一个位置移动到另一位置。有时也在内存和寄存器之间进行数据移动，寄存器在 CPU 中，可以高速访问数据。移动数据，特别是大量的数据，要花费大的代价。因此，有时也使用“指针”来代替数据。计算包括简单的操作，如增加一个可变数据元素的值。更复杂的计算也许涉及许多操作和数据元素。

指令也可按顺序、有条件地或循环地执行。顺序指令是一个接一个执行的指令。条件指令执行时根据某些数值执行不同的指令集。在某些语言中这被称为“if”语句。循环指令反复执行，也可以取决于某些数值。这有时称作“loop”。通常，一个指令可以“调用”在其他程序或模块中定义的指令集。当使用多个计算机处理器时，指令可以同时执行。

软件运行的一个简单例子是当用户从菜单中选择“复制”时所发生的事情。在这种情况下，条件指令被执行，也许把内存“文档”存储区中的数据文本复制到称作“剪贴板”数据区的中间存储区。如果选择另外的菜单项如“粘贴”，软件也许执行指令，把剪贴板数据区中的文本复制到内存中本文档或其他文档的特定位置。

根据应用软件的不同，以上例子也可能变得更复杂。软件工程行业努力管理软件运行的复杂性。在大型或功能强大的计算机系统中，软件的运行尤其如此。

当前，在应用软件中，使用计算机软件的唯一限制几乎就是设计人员/编程人员的独创性。因此，从前被认为不能进行软件模拟的大的活动领域（如进行大师级象棋比赛）现在可以程序化了。迄今为止，业已证明软件仿真比较可靠的唯一领域是人类艺术方面，特别是在演奏动听的音乐和创作文学作品方面。

软件按照运行可以分为以下几类：可执行的计算机程序、源代码或脚本、配置程序。