

目 录

第一章 计算机组成原理.....	1
考点一 进制转换与机器数.....	1
考点二 主存容量计算.....	2
考点三 Cache-主存地址映射	3
考点四 总线结构.....	5
第二章 计算机网络	7
考点一 OSI 模型和 TCP/IP 模型	7
考点二 IP 地址和子网掩码	9
考点三 域名.....	10
考点四 防火墙和数据备份.....	11
考点五 计算机病毒.....	12
第三章 操作系统.....	14
考点一 单道、多道批处理系统、分时系统和实时系统.....	14
考点二 进程.....	16
考点三 进程调度方式和进程调度算法.....	17
考点四 死锁.....	18
第四章 数据结构与算法.....	20
考点一 常见的算法.....	20
考点二 线性表的顺序、链式存储方式.....	21
考点三 线性链表.....	22
考点四 队列、栈、二叉树和图.....	23
第五章 数据库	26
考点一 三级模式和二级映像.....	26
考点二 关系运算.....	27
考点三 关系的三类完整性约束.....	29

考点四 数据类型、数据定义语言和数据操纵语言.....	30
考点五 数据查询.....	34
考点六 事务与并发操作问题.....	37
第六章 软件工程.....	39
考点一 软件的生存周期和开发模型.....	39
考点二 软件测试.....	40
考点三 软件维护.....	42
第七章 C++.....	44
考点一 类、标识符、数据类型和存储类型.....	44
考点二 运算符.....	46
考点三 函数.....	49
考点四 指针数组、符号常量、数组做函数参数.....	51

第一章 计算机组成原理

考点一 进制转换与机器数

进制转换方法	非十进制转换成十进制	<u>按权展开，然后相加求和。</u>
	十进制转换成非十进制	<u>整数部分：除 N 倒序取余，除到商为 0 停止；</u> 小数部分：乘 N 正序取整，乘到小数点后为 0 停止。
	二进制转换成八进制	<u>三位二进制数为一组相当于一位八进制数（421）；</u> <u>整数部分，从右向左数三位；</u> 小数部分，从左向右数三位； 不够三位时，补 0。
	二进制转换成十六进制	<u>四位二进制数为一组相当于一位十六进制数（8421）；</u> <u>整数部分，从右向左数四位；</u> 小数部分，从左向右数四位； 不够四位时，补 0。
机器数	真值	带“+”或“-”符号的数。
	原码	原码由 <u>符号位和数值位</u> 组成； <u>符号位为 0 表示正数，符号位为 1 表示负数；</u> 数值位即真值的绝对值转换成的二进制数。
	反码	对于正数，反码=原码； 对于 <u>负数</u> ， <u>反码是对原码除符号位以外的每一位取反。</u>
	补码	在计算机系统中， <u>数值一律用补码来表示和存储；</u> 对于正数，补码=原码； 对于负数，补码是在反码的基础上末位加 1 得到的。

备注：

数制：数制也叫计数制，使用一组固定的符号和统一的规则来表示数值的方法。目前我们用到的进制有二进制、八进制、十进制和十六进制。

数制组成：十进制数由 0~9 十个数字组成，二进制数由 0 和 1 两个数字组成；八进制数由 0~7 八个数字组成；十六进制数由 0~9 十个数字和 A~F（a~f）六个字母组成。

~ 冲刺模拟 ~

- 1.将十进制数 24 转换为二进制数是 ()。
 A. 11000 B. 11100 C. 11001 D. 10001
- 2.将二进制数 1100101 转换为十六进制数是 ()。
 A. 56 B. AB C. 65 D. C4
- 3.二进制数-1011010 的补码是 ()。
 A. 1011010 B. 11011010 C. 10100101 D. 10100110

考点二 主存容量计算

公 式	<u>主存容量=存储单元个数×存储字长(位)</u>
各 部 分 含 义	<p>(1) <u>主存容量是指主存中存放二进制代码的总位数。</u></p> <p>(2) <u>存储单元的个数对应 MAR 的位数(即地址线的根数)</u>, MAR 即存储器地址寄存器,用来存放欲访问的存储单元的地址。存储单元个数=2^n (n 为地址线的根数或者 MAR 的位数)。如 MAR 有 10 位,对应地址线有 10 根,则有 $2^{10}=1024$ 个存储单元,记为 1K。</p> <p>(3) <u>存储字长对应 MDR 的位数(即数据线的根数)</u>, MDR 即存储器数据寄存器,用来存放从存储体某单元取出的代码或者准备往某存储单元存入的代码。如 MDR 有 16 位,对应数据线有 16 根,则存储字长为 16 位。</p>

~ 冲刺模拟 ~

- 1.一个 16K×8 位的存储器,其地址线和数据线的总和是 ()。
 A. 48 B. 46 C. 17 D. 22
- 2.已知存储体内有 65536 个存储单元,对应的数据总线为 32 根,那么此主存储器容量为 ()。
 A. 128KB B. 64KB C. 512KB D. 256KB

考点三 Cache-主存地址映射

定义	地址映射是指由主存地址映射到 Cache 地址	
地址映射的方式	直接映射 (固定的映射关系)	<p>(1) <u>每个主存块只与一个缓存块相对应，映射结果表明每个缓存块对应若干个主存块。</u>例如，主存的第 0 块、第 16 块、第 32 块……，只能映射到 Cache 的第 0 块；而主存的第 1 块、第 17 块、第 33 块，只能映射到 Cache 的第 1 块……。</p> <p><u>直接映像的关系如下：</u></p> <p><u>$K = I \bmod C$;</u></p> <p><u>式中 K 为 Cache 的块号，I 为内存的块号，C 为 Cache 块号的总数；</u></p> <p><u>表示第 I 号内存块映射到第 K 号 Cache 块上。</u></p> <p>(2) 优点：实现简单，只需利用主存地址的某些位直接判断，即可以确定所需字块是否在缓存中。</p> <p>(3) 缺点：不够灵活，因为每个主存块只能固定地对应某个缓存块，即使缓存内还空着许多位置也不能占用，使缓存的存储空间得不到充分的利用。如果程序恰好要重复访问对应同一缓存位置的不同主存块，就要不停地进行替换，从而降低命中率。</p>

	<p>(1) <u>全相联映射允许主存中每一字块映射到 Cache 中的任何一块位置上。</u></p> <p>(2) 优点：方式灵活，命中率高，缩小了块冲突率。</p> <p>(3) 缺点：由于 Cache 比较电路的设计和实现比较困难，所以这种方式只适合于小容量 Cache 采用。</p>
组相联映射	<div data-bbox="540 415 1235 895" data-label="Diagram"> </div> <p>(1) 组相联映射是上述两种映射的折中，<u>主存和 Cache 都分组，主存中一个组内的块数与 Cache 中的分组数相同，组间采用直接映射，组内采用全相联映射。</u>也就是说，将 Cache 分成 u 组，每组 v 块，主存块存放到哪个组是固定的，至于存到该组哪一块则是灵活的。例如，主存分为 256 组，每组 8 块，Cache 分为 8 组，每组 2 块。主存中的各块与 Cache 的组号之间有固定的映射关系，但可自由映射到对应 Cache 组中的任何一块。例如，主存中的第 0 块、第 8 块……均映射于 Cache 的第 0 组，但可映射到 Cache 第 0 组中的第 0 块或第 1 块；主存的第 1 块、第 9 块……均映射于 Cache 的第 1 组，但可映射到 Cache 第 1 组中的第 2 块或第 3 块；……。</p> <p><u>把 Cache 分为 Q 组，每组有 S 块，组相联映射的关系如下：</u></p> <p><u>$I = J \bmod Q$</u></p> <p><u>式中，I 为缓存的组号，J 为主存的块号；</u></p> <p><u>表示第 J 号主存块映射到第 I 号缓存组里。</u></p> <p>(2) 假设 Cache 共有 32 块，共分为 16 组，每组内包含 2 块。组内 2 块的组相联映射又称为二路组相联。</p>

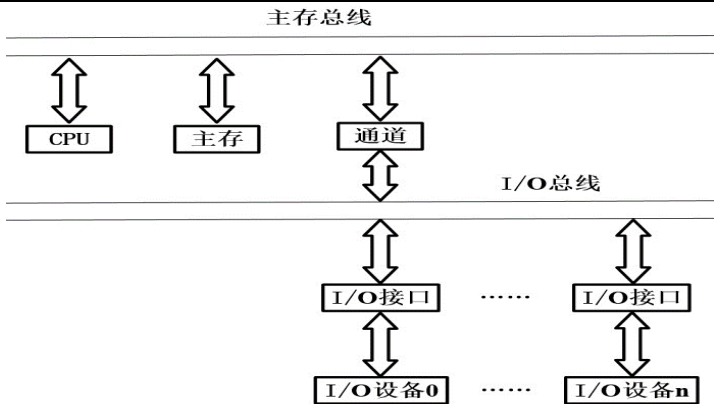
❧ 冲刺模拟 ❧

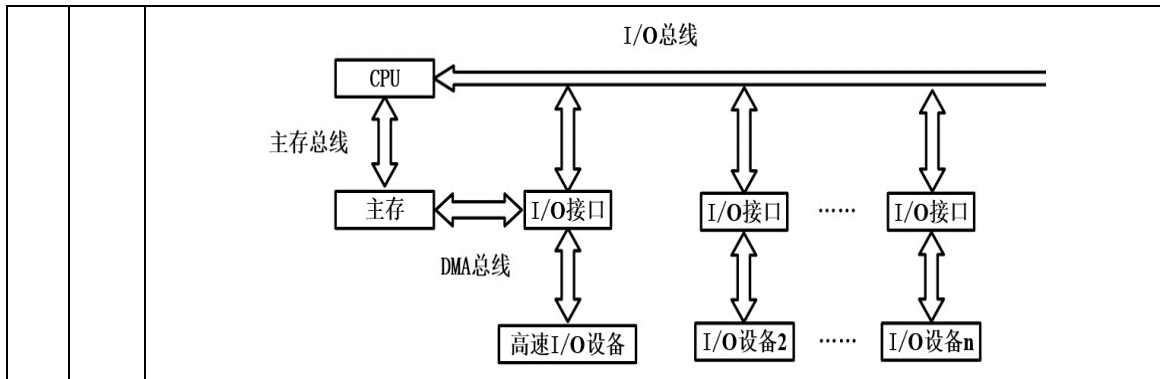
1. Cache 地址映象中，主存中的任一块均可映射到 Cache 内任一块的位置上，称作()。
- A.直接映象 B.全相联映象 C.组相连映像 D.快速映象

2.假设主存容量为 512KB，Cache 容量为 4KB，每个字块为 16 个字，每个字 32 位。在直接映射方式下，主存的第（ ）块映射到 Cache 中的第 6 块（设起始块为第 1 块）。

- A. 5 B. 7 C. 8 D. 70

考点四 总线结构

总线结构	单总线结构	<p>1.<u>单总线结构，是将 CPU、主存、I/O 设备都挂在一组总线上</u>，允许 I/O 设备之间、I/O 设备与 CPU 之间或 I/O 设备与主存之间直接交换信息。</p> <p>2.特点：结构简单，便于扩充，所有的传送都通过这组共享总线，<u>不允许两个以上的部件在同一时刻向总线传输信息</u>，影响系统工作效率的提高。</p>
	双总线结构	<p>1.双总线结构如下图所示，图中<u>通道是一个具有特殊功能的处理器，即输入/输出处理器。CPU 将一部分功能下放给通道，使其对 I/O 设备具有统一管理的功能</u>，以完成外部设备与主存储器之间的数据传送，其系统的吞吐能力可以相当大，这种结构大多用于大、中型计算机系统。</p> <p>2.特点：将速度较低的 I/O 设备从单总线上分离出来，形成主存总线与 I/O 总线分开的结构。</p> 
	三总线结构	<p>1.<u>主存总线</u>用于 <u>CPU 与主存</u>之间的传输；<u>I/O 总线</u>供 <u>CPU 与各类 I/O 设备</u>之间传递信息；<u>DMA 总线，即直接内存存取总线</u>，用于<u>高速 I/O 设备</u>（磁盘、磁带等）<u>与主存</u>之间直接交换信息。</p> <p>2.<u>在三总线结构中，任一时刻只能使用一种总线。</u></p> <p>3.主存总线与 DMA 总线不能同时对主存进行存取，I/O 总线只有在 CPU 执行 I/O 指令时才能用到。</p>



🌊 冲刺模拟 🌊

- 1.所谓计算机三总线结构的“三总线”分别指（ ）。
A.地址线、数据线和控制线三组传输线
B. I/O 总线、主存总线 和 DMA 总线三组传输线
C. I/O 总线、主存总线和系统总线三组传输线
D.设备总线、主存总线和控制总线三组传输线
- 2.双总线结构中的主存总线和 I/O 总线通过（ ）进行连接。
A.通道 B. CPU C.主存 D. I/O 接口

第二章 计算机网络

考点一 OSI 模型和 TCP/IP 模型

OSI 七层参考模型	层次名	基本功能	数据名称	设备
	应用层	面向应用	数据	计算机或终端设备 (如网关)
	表示层	数据加密、数据格式修改等		
	会话层	会话建立		
	传输层	<u>数据分段、端到端传输</u>	<u>数据段</u>	<u>路由器</u> 、防火墙
	网络层	<u>路径选择和寻址、点到点传输</u>	<u>数据报</u>	
	数据链路层	封装成帧、流量控制、链路管理	<u>数据帧</u>	
	物理层	比特流传输	<u>比特</u>	中继器、 <u>集线器</u>

TCP/IP 模型	层次名	功能
	应用层	<u>向用户提供一组常用的应用程序，比如电子邮件、文件传输访问、远程登录。</u>
	传输层	<u>提供应用程序间的通信。其功能包括：格式化信息流、提供可靠传输。</u>
	网络层	<u>负责相邻计算机之间的通信，处理数据报、路径、流控、拥塞。</u>
	网络接口层	定义物理介质的各种特性，处理数据帧。

OSI 和 TCP/IP 的 对应关系	OSI	TCP/IP	协议	说明
	应用层	应用层	HTTP (超文本传输协议)	支持 WWW 应用
			FTP (文件传输协议)	负责文件的上传和下载
	表示层		TELNET (远程登录协议)	本地计算机通过网络访问远 程计算机上的资源的过程
			SMTP (简单邮件传输协议)	负责邮件的发送
	会话层		POP3 (邮局协议)	负责邮件的接收
			DNS	负责将域名转换成 IP 地址

			(域名解析协议)	
	传输层	传输层	TCP (传输控制协议)	面向连接、可靠、重质量
			UDP (用户数据报协议)	面向非连接、不可靠、重速度
			OSPF (路由选择协议)	在单一自治系统内决策路由
	网络层	网络层	IP (网际协议)	对数据包进行寻址和路由
			ARP (地址解析协议)	将 IP 地址转换成 MAC 地址
			RARP (逆向地址解析协议)	将 MAC 地址转化成 IP 地址
	数据链路层	网络接口层	各种通信网络接口 (物理网络)	
	物理层			

备注:

1977 年, ISO (国际标准组织) 为了使不同厂家的网络产品实现互连和通信提出了开放系统互联参考模型 OSI。

美国国防部赞助的 ARPANET 提出的 TCP/IP 参考模型, 这个模型中包含了很多协议, 但 TCP 协议和 IP 协议是最重要的协议, 所以叫 TCP/IP 模型, 构成了现在 Internet 网的体系结构。

冲刺模拟

1. IP 协议工作在 TCP/IP 模型的 ()。

- A.网络层 B.传输层 C.应用层 D.表示层

2. OSI 参考模型中物理层传输的数据单位是 ()。

- A.比特 B.帧 C.数据包 D.数据段

考点二 IP 地址和子网掩码

IP 地址	IPV4	定义	IP 地址是用来唯一标识计算机的网络地址。IPV4 由 32 个二进制数组成，采用“点分十进制”表示法，将 32 位二进制数分为四组，每组 8 位， <u>每组二进制数用十进制数表示</u> ，每组的取值范围为 0~255。一般 IP 地址由网络号和主机号两部分构成。	
		分类	A 类	IP 地址地址范围 1.0.0.0 至 127.255.255.255
			B 类	IP 地址地址范围 128.0.0.0 至 191.255.255.255
			C 类	IP 地址地址范围 192.0.0.0 至 223.255.255.255
IPV6	定义	IPV6 由 128 位二进制数组成，采用“冒分十六进制”表示法，将 128 位二进制数分为八组，每组 16 位二进制数， <u>每组二进制数用十六进制数表示</u> ，每组的取值范围为 0000~FFFF。		
	IPV4 转换为 IPV6	在 IPV6 中，有两类地址用来表示 IPV4 地址： <u>一类是兼容的，它是 96 位 0 和 32 位的 IPV4 地址；</u> 例如：IPV4 地址为 192.168.1.1，十进制表示为 192.168.1.1，二进制表示为 11000000 10101000 00000001 00000001，十六进制表示为 C0 A8 01 01，那么以兼容方式转换为 IPV6 地址为 0:0:0:0:0:C0A8:0101。 <u>另一类是映射的，它是由 80 位的 0、16 位的 1、32 位的 IPV4 地址来表示。</u> 例如：地址 0000:0000:0000:0000:0000:FFFF:1234:5678 属于映射方式表示的 IPV4 地址。		
子网掩码	定义	子网掩码是一个 32 位地址，用“点分十进制”表示，用于屏蔽 IPV4 地址的一部分以区别网络标识和主机标识。IP 地址网络部分对应的掩码部分全为“1”，主机部分对应的掩码全为“0”。子网掩码不能单独存在，它必须结合 IP 地址一起使用，作用是将某个 IP 地址划分成网络地址和主机地址两部分，即划分逻辑网段。		
	分类	A 类网络	子网掩码为 255.0.0.0	
		B 类网络	子网掩码为 255.255.0.0	
		C 类网络	子网掩码为 255.255.255.0	

备注：

IPv4 地址分为 A、B、C、D、E 五类，用户使用的是 A、B、C 三类，称为基本类，D 类地址为多播地址，E 类地址为保留今后使用的地址。

在已知 IP 地址和缺省子网掩码的情况下，求网络号和主机号的方法：首先将 IP 地址和子网掩码分别转换为二进制数，然后，将两个二进制数进行逻辑与运算，得到的结果即为网络号；

将子网掩码取反再与 IP 地址进行逻辑与运算，得到的结果即为主机号。

~ 冲刺模拟 ~

1. 具有将某个 IP 地址划分成网络地址和主机地址两部分的功能是指 ()。

A. IP 地址
B. 冒分十六进制

C. 点分十进制
D. 子网掩码

2. 192.168.0.1 是 ()。

A. IP 地址
B. 子网掩码
C. A 类 IP 地址
D. B 类 IP 地址

考点三 域名

域名定义	域名是由一串用点分隔的名字组成的 Internet 上某一台计算机或计算机组的名称，用于标识计算机的地理位置。通过 <u>域名系统 (DNS) 将域名和 IP 地址相互映射</u> 。	
域名组成	域名的结构由标号序列组成，各标号之间用点隔开：...三级域名.二级域名.顶级域名，各标号分别代表不同级别的域名，域名的组成：一般分 4 级，每级 1-4 个字符，中间用小数点间隔。 <u>级别低的域名写在左边，级别高的域名写在右边。</u>	
域名解析	<u>递归查询</u>	<u>主机向本地域名服务器的查询一般都是采用递归查询。在该模式下 DNS 服务器接收到客户机请求，必须使用一个准确的查询结果回复客户机。如果主机所询问的本地域名服务器不知道被查询域名的 IP 地址，那么本地域名服务器就以 DNS 客户的身份，向其他根域名服务器继续发出查询请求报文。</u>
	<u>迭代查询</u>	<u>本地域名服务器向根域名服务器的查询通常是采用迭代查询。当根域名服务器收到本地域名服务器的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地域名服务器，“你下一步应当向哪一个域名服务器进行查询”。然后让本地域名服务器进行后续的查询。</u>

备注：

1. IP 地址和域名的关系为一对多的关系。
2. 常见域名如下：

域名	意义	域名	意义
com	商业组织	cn	中国
edu	学校教育	tw	台湾

gov	政府部门	hk	香港
int	国际组织	us	美国
mil	军事组织	jp	日本
net	网络中心	org	非盈利组织

冲刺模拟

1. 以下属于美国的域名的是 ()。

A. www.asD. com

B. www.qwe.com.cn

C. www.ito.com.us

D. www.gol.com.usa

2. 若要对 Internet 上任何一个域名进行解析, 只要自己无法解析, 应首先求助于 ()。

A. 根域名服务器

B. 顶级域名服务器

C. 权限域名服务器

D. 本地域名服务器

考点四 防火墙和数据备份

防火墙	定义	防火墙是指一个由软件和硬件设备组合而成、在内部网和外部网之间、专用网与公共网之间的界面上构造的 <u>保护屏障</u> 。 <u>它能够阻止对信息资源的非法访问、控制进出两个方向的通信、并且只允许本地安全策略授权的通信信息通过。</u>	
	功能	<p>(1) <u>服务控制</u>: 确定可以访问的网络服务类型。防火墙可以在 IP 地址和 TCP 端口号的基础上过滤通信量。</p> <p>(2) <u>方向控制</u>: 确定特定的服务请求可以发起和允许通过防火墙的方向。</p> <p>(3) <u>用户控制</u>: <u>根据哪个用户尝试访问服务来控制对于一个服务的访问。这个特征典型地应用于防火墙边界以内的用户 (本地用户)。它也可能应用于来自外部用户的进入通信量。</u></p> <p>(4) <u>行为控制</u>: 控制怎样使用特定的服务。例如: 防火墙可以过滤电子邮件来消除垃圾邮件, 或者可以使得外部只能访问一个本地 Web 服务器的一部分信息。</p>	
	分类	<u>根据防火墙组成组件划分</u>	将防火墙分为 <u>软件防火墙和硬件防火墙</u> 。软件防火墙以纯软件的方式表现, 安装在边界计算机或服务器上就可以实现防火墙的各种功能; 硬件防火墙以专用硬件设备形式出现, 一般情况下, 硬件防火墙都是以软件和硬件相结合的方式实现。
		<u>根据防火墙使用的主要技术</u>	防火墙分为基于 <u>包过滤的防火墙、应用层代理</u> 、电路级网关、地址翻译防火墙和状态检查防火墙等。

		<u>划分</u>	<u>最主要的防火墙是包过滤型防火墙</u> ，它往往用一台 <u>过滤路由器</u> 来实现对所接收的每个数据包做允许或拒绝的决定。 <u>路由器审查每个数据包，以便确定其是否与某一条包过滤规则匹配。</u> 过滤规则基于 IP 包头信息。包头信息中包括 IP 源地址、IP 目标端地址、内装协议、TCP、UDP 目标端口等。
数据备份	定义	<u>数据备份是容灾的基础</u> ，是指为防止系统出现操作失误或系统故障导致数据丢失，而将全部或部分数据的集合，从应用主机的硬盘复制到其它的存储介质的过程。	
	分类	<u>完全备份</u>	完全备份就是用一盘磁带 <u>对整个系统进行完全备份</u> ，即备份全部选中的文件夹，包括系统和数据。
		<u>增量备份</u>	增量备份指的是在第一次完全备份或最后一次备份的基础上， <u>以后每次的备份只需备份与前一次相比增加或者被修改的文件。</u>
		<u>差分备份</u>	<u>差分备份是指每次备份的数据是相对于上一次完全备份之后新增加的和修改过的数据。</u>

~ 冲刺模拟 ~

1. 防火墙所不具有的功能是（ ）。
 A. 服务控制 B. 用户控制 C. 行为控制 D. 查杀病毒
2. 常用的数据备份策略有完全备份、增量备份和（ ）。
 A. 差分备份 B. 冷备份 C. 磁盘备份 D. 热备份

考点五 计算机病毒

定义	计算机病毒是 <u>编制者</u> 在计算机程序中插入的破坏计算机功能或者数据的代码，并能进行 <u>自我复制</u> 的一组 <u>计算机指令或者程序代码</u> 。	
分类	根据破坏性划分	<u>良性病毒、恶性病毒</u>
	根据传染方式划分	引导区型病毒、 <u>文件型病毒</u> 、混合型病毒、 <u>宏病毒</u>
	根据病毒存在的媒体划分	网络病毒、文件病毒、引导型病毒
特征	<u>寄生性、传染性、潜伏性、隐蔽性、破坏性、可触发性</u>	
预防方法	1. 及时更新系统漏洞补丁，很多计算机病毒都是利用操作系统的漏洞进行感染和传播的； 2. 打开防火墙并安装杀毒软件，及时更新杀毒软件病毒库；	

	<p>3.不要轻易打开陌生的电子邮件附件；</p> <p>4.从网上下载的任何文件，一定要先扫描杀毒后再运行；</p> <p>5.对重要的文件要做备份，以免遭到病毒侵害时不能立即恢复，造成不必要的损失。</p>
--	---

冲刺模拟

1.计算机病毒具有破坏性、()、潜伏性和传染性等特点。

- A.不可识别性 B.再生性 C.隐蔽性 D.永久性

2.计算机病毒侵入到计算机后，不会立即发作，而是到条件成熟时才发作，这属于计算机病毒的()特征。

- A.潜伏性 B.隐藏性 C.传染性 D.破坏性

第三章 操作系统

考点一 单道、多道批处理系统、分时系统和实时系统

单道批处理系统	定义	系统对作业的处理是成批进行的，但 <u>内存中始终保持一道作业</u> 。	
	特点	内存中仅有一道程序运行，即监督程序每次从磁带上只调入一道程序进入内存运行，当该程序完成或发生异常情况时，才换入其后继程序进入内存运行。	
多道批处理系统	背景	在单道批处理系统中，内存中仅有一道作业，它无法充分利用系统中的所有资源，致使系统性能较差。 <u>为了进一步提高资源的利用率和系统吞吐量</u> ，在 20 世纪 60 年代又引入了多道程序设计技术，由此而形成了多道批处理系统。	
	定义	<u>多道程序设计技术是指多个程序同时进入内存并运行，即同时把多个程序放入内存，并允许它们交替在 CPU 中运行，它们共享系统中的各种软、硬件资源</u> 。当一道程序因 I/O 请求而暂停运行时，CPU 便立即转去运行另一道程序。	
	特征	<u>多道</u>	计算机内存中同时存放多道相互独立的程序。
		<u>宏观上并行</u>	<u>同时进入系统的多道程序都处于运行过程中</u> ，即它们先后开始了各自的运行，但都未运行完毕。
		<u>微观上串行</u>	<u>内存中的多道程序轮流占有 CPU，交替执行</u> 。
		优点	<u>资源利用率高</u> 多道程序共享计算机资源，从而使各种资源得到充分利用。
			<u>系统吞吐量大</u> CPU 和其它资源保持“忙碌”状态。
		缺点	用户响应的时间较长、且 <u>不提供人机交互能力</u> 。
分时系统	背景	为了提高人机交互能力，诞生了分时系统。	
	定义	在操作系统中采用分时技术形成的分时系统。 <u>分时系统是指在一台主机上连接了多个带有显示器和键盘的终端，同时允许多个用户通过自己的终端，以交互方式使用计算机，共享主机中的资源</u> 。	
	特点	<u>同时性</u>	<u>同时性也称多路性，指允许多个终端用户同时使用一台计算机，即一台计算机与若干台终端相连接，终端上的这些用户可以同时或基本同时使用计算机</u> 。
		<u>交互性</u>	用户能够方便地与系统进行人-机对话，即 <u>用户通过终端采用人机对话的方式直接控制程序运行</u> 。
		<u>独立性</u>	<u>系统中多个用户可以彼此独立地进行操作，互不干扰</u> ，单个用户感觉不到别人也在使用这台计算机，好像只有自己单独使用这台计算机。

			机一样。
		及时性	<u>用户请求能在很短时间获得响应</u> 。分时系统采用时间片轮转方式使一台计算机同时为多个终端服务，使用户能够对系统的及时响应感到满意。
实时系统	背景	<u>为了能在某个时间限制内完成某些紧急任务而不需时间片排队，诞生了实时操作系统。</u>	
	定义	在实时操作系统的控制下，计算机系统接收到外部信号后及时进行处理，并且要在严格的时限内处理完接收的事件。 <u>实时操作系统的主要特点是及时性和可靠性（实时系统的特征和分时系统的特征类似）。</u>	

操作系统的特征	并发性	<u>并发性是指两个或多个事件在同一时间间隔内发生。</u> （和并发相区别的是并行， <u>并行是指两个或多个事件在同一时刻发生。</u> ）
	共享性	<u>共享性是指系统中的资源可供内存中多个并发执行的进程共同使用</u> ，相应地，把这种资源共同使用的情况称为资源共享，或称为资源复用。
	虚拟性	<u>虚拟性是指通过某种技术把一个物理实体变为若干个逻辑上的对应物。</u> 物理实体（前者）是实的，即实际存在的，而后者是虚的，仅是用户感觉上的东西。
	异步性	由于资源等因素的限制，使进程的执行通常都不是“一气呵成”，而是以“停停走走”的方式运行。内存中的每个进程在何时能获得处理机运行，何时又因提出某种资源请求而暂停，以及进程以怎样的速度向前推进，每道程序总共需要多少时间才能完成，等等，这些都是不可预知的。或者说， <u>进程是以人们不可预知的速度向前推进，此即进程的异步性。</u>

❧ 冲刺模拟 ❧

- 下列关于操作系统的叙述中，正确的是（ ）。
 - 单道批处理系统，内存中仅有一道程序运行
 - 分时系统不一定都具有人机交互的功能
 - 从响应时间的角度来看，实时系统与分时系统的要求差不多
 - 由于采用了分时系统，用户可以独占计算机的文件系统
- 下面系统中，必须是实时操作系统的是（ ）。
 - 计算机辅助设计系统
 - 航空订票系统
 - 办公自动化系统
 - 计算机激光照排系统

考点二 进程

定义	<p>(1) <u>进程是程序的一次执行；</u></p> <p>(2) <u>进程是一个程序及其数据在处理器上顺序执行时所发生的活动；</u></p> <p>(3) <u>进程是程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位。</u></p>		
特征	结构特征	<p><u>进程实体包括程序段、相关的数据段和进程控制块三部分。</u></p> <p>进程控制块 (PCB) 是指为使程序 (含数据) 能独立运行的程序。</p>	
	<u>动态性</u>	<p><u>动态性是指进程实体的一次执行过程。</u></p> <p>动态性表现在：“它由创建而产生，由调度而执行，由撤消而消亡”。</p>	
	<u>并发性</u>	<p><u>并发性是指多个进程实体共同存在内存中，且能在一段时间内同时运行。</u></p> <p>引入进程的的目的就是为了使进程实体能和其它进程实体并发执行；而<u>程序</u> (没有建立 PCB) 是不能并发执行的。</p>	
	<u>独立性</u>	<p>独立性是指进程实体是一个能独立运行、独立分配资源和独立接受调度的基本单位。</p>	
	<u>异步性</u>	<p>异步性是指进程按各自独立的、不可预知的速度向前推进，或者说进程实体按异步方式运行。</p>	
基本状态	分类	<u>就绪状态</u>	就绪状态的进程除了未分配到 CPU 外，其他资源均已具备。
		<u>执行状态</u>	进程获得 CPU，其程序正在执行。
		<u>阻塞状态</u>	<u>正在执行的进程由于发生某个事件暂时无法继续执行而放弃处理机处于暂停状态</u> ，即进程的执行受到阻塞，把这种暂停状态称为阻塞状态、等待状态或封锁状态。(致使进程阻塞的典型事件有：请求 I/O、申请缓冲空间等。)
	<u>状态转换</u>	<p><u>就绪态→运行态</u></p> <p><u>运行态→就绪态</u></p> <p><u>运行态→阻塞态</u></p> <p><u>阻塞态→就绪态</u></p>	

冲刺模拟

1.当进程已分配到除 CPU 以外的所有必要资源，只要再获得 CPU，便可立即执行，此时的进程所处的状态为 ()。

- A.执行状态 B.阻塞状态 C.封锁状态 D.就绪状态

2.以下选项当中是错误的进程状态转换的是 ()。

A.就绪态→运行态

B.运行态→阻塞态

C.阻塞态→运行态

D.运行态→就绪态

考点三 进程调度方式和进程调度算法

进程调度方式	非抢占式	在采用这种调度方式时， <u>一旦把处理机分配给某进程后，不管它要运行多长时间，都一直让它运行下去</u> ，决不会因为时钟中断等原因而抢占正在运行进程的处理机，也不允许其它进程抢占已经分配给它的处理机。	
	抢占式	<u>允许调度程序根据某种原则去暂停某个正在执行的进程，将已分配给该进程的处理机重新分配给另一进程。</u>	
进程调度算法	先来先服务	先来先服务(FCFS) <u>总是把当前处于就绪队列之首的那个进程调度到运行状态</u> 。当前作业或进程占用 CPU 后，直到执行完或阻塞时，才出让 CPU。先来先服务是一种非抢占式策略。	
	短作业优先	短作业优先是对预计执行时间短的作业（进程）优先分派处理机。 <u>优点是缩短作业的等待时间；提高系统的吞吐量。</u> <u>缺点是对长作业非常不利，可能长时间得不到执行。</u> 目标是减少平均周转时间。	
	高优先权优先	非抢占式	在这种方式下，系统一旦把处理机分配给就绪队列中优先权最高的进程后，该进程便一直执行下去，直至完成；或因发生某事件使该进程放弃处理机时，系统方可再将处理机重新分配给另一优先权最高的进程。
		抢占式	在这种方式下， <u>系统把处理机分配给优先权最高的进程，使之执行。但在其执行期间，只要又出现了另一个其优先权更高的进程，进程调度程序就立即停止当前进程（原优先权最高的进程）的执行，重新将处理机分配给新到的优先权最高的进程。</u>
	时间片轮转	在时间片轮转法中，系统将所有的就绪进程按先来先服务的原则排成一个队列， <u>每次调度时，把 CPU 分配给队首进程，并令其执行一个时间片，即该进程允许运行的时间，如果在时间片结束时进程还在运行，则 CPU 将被剥夺并分配给另一个进程。</u> （时间片略大于一次典型的交互所需要的时间，保证大多数进程在一个时间片内完成。）	

冲刺模拟

1.先来先服务的进程调度策略适合于（ ）。

A.短作业进程

B.长作业进程

C.低优先作业进程

D.高优先作业进程

2.下面选项当中，()算法可以缩短作业的等待时间，提高系统的吞吐量，有利于短作业的执行，但不利于长作业的执行。

A.短作业优先

B.先来先服务

C.高优先权优先

D.时间片轮转

考点四 死锁

产生死锁的原因	<u>竞争资源</u>	当系统中供多个进程共享的资源（如打印机），其 <u>数目不足以满足多个进程的需要</u> 时，会引起多个进程对资源的竞争而产生死锁。
	<u>进程间推进顺序非法</u>	<u>进程在运行过程中，请求和释放资源的顺序不当</u> ，也同样会导致产生进程死锁。
产生死锁的必要条件	<u>互斥条件</u>	指 <u>进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用</u> 。如果此时还有其它进程请求该资源，则请求者只能等待，直至占有该资源的进程用完释放。
	<u>请求和保持条件</u>	指 <u>进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源又已被其它进程占有</u> ，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。
	<u>不剥夺条件</u>	指 <u>进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。</u>
	<u>环路等待条件</u>	指 <u>在发生死锁时，必然存在一个进程——资源的环形链。</u>
处理死锁的基本方法	<u>预防死锁</u>	为保证系统中进程的正常运行，应事先采取必要的措施，来预防发生死锁。 <u>通过设置某些限制条件，去破坏产生死锁的四个必要条件中的一个或几个条件，来预防发生死锁：</u> <u>(1) 摒弃“请求和保持”条件；</u> <u>(2) 摒弃“不剥夺”条件；</u> <u>(3) 摒弃“环路等待”条件。</u>
	<u>避免死锁</u>	该方法是在资源的动态分配过程中，用某种方法去防止系统进入不安全状态，从而避免发生死锁。 <u>避免死锁的方法：银行家算法。</u>

	<p><u>解除死锁</u></p> <p><u>在系统中已经出现死锁后，则应及时检测到死锁的发生，并采取适当措施来解除死锁。</u>当发现有进程死锁时，便应立即把它们从死锁状态中解脱出来。<u>常采用的解除死锁的方法是：</u></p> <p>(1) <u>剥夺资源</u>：从其它进程剥夺足够数量的资源给死锁进程，以解除死锁状态。</p> <p>(2) <u>撤消进程</u>：最简单的撤消进程的方法是使全部死锁进程都夭折掉；稍微温和一点的方法是按照某种顺序逐个地撤消进程，直至有足够的资源可用，使死锁状态消除为止。</p>
--	--

❧ 冲刺模拟 ❧

1. 解决死锁的途径是 ()。
 - A. 立即关机再重新开机
 - B. 立即关机排除故障
 - C. 不要共享资源，增加独占资源
 - D. 设计预防死锁，运行检测并恢复

2. (多选题) 系统产生死锁必定同时保持的必要条件是 ()。
 - A. 互斥条件
 - B. 占有和等待条件
 - C. 不剥夺条件
 - D. 循环等待条件

第四章 数据结构与算法

考点一 常见的算法

<u>递归法</u>	递归法是指一种通过 <u>重复将问题分解为同类的子问题而解决问题的方法</u> 。
<u>穷举法(暴力破解法)</u>	对于要解决的问题， <u>列举出它的所有可能的情况，逐个判断有哪些是符合问题所要求的条件，从而得到问题的全部解答</u> 。
<u>贪心算法</u>	一步一步地进行， <u>常以当前情况为基础根据某个优化测度作最优选择，而不考虑各种可能的整体情况</u> ，采用自顶向下，以迭代的方法做出相继的贪心选择， <u>每一步的贪心选择，都可得到问题的一个最优解</u> 。
<u>分治法</u>	<u>把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题，循环往复，直到最后子问题可以简单的直接求解</u> ，原问题的解即子问题的解的合并。
<u>动态规划法</u>	<u>将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解</u> 。
<u>回溯法</u>	一种选优搜索法， <u>按选优条件向前搜索，当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择</u> ，这种走不通就退回再走的技术为回溯法。

冲刺模拟

1.现有 16 枚外形相同的硬币，其中有一枚比真币的重量轻的假币，若采用分治法找出这枚假币，至少比较（ ）次才能够找出该假币。

- A. 3 B. 4 C. 5 D. 6

2.下面选项当中，（ ）是一种选优搜索法，按选优条件向前搜索，当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择。

- A.递归法 B.回溯法 C.穷举法 D.分治法

考点二 线性表的顺序、链式存储方式

线性表	定义	线性表是数据结构的一种，一个线性表是 n 个具有相同特性的数据元素的有限序列，表的长度就是数据元素的个数。		
	存储方式	顺序存储结构	定义	<u>顺序存储是指用一段地址连续的存储单元依次存储线性表的数据元素。</u> 例如：顺序表即采用顺序存储结构的线性表。
			特点	<p>(1) <u>顺序存储结构是一种随机存取的存储结构；</u></p> <p>(2) <u>在做插入、删除操作时需移动大量元素。</u></p> <p>例如：长度为 n 线性表，在位置 $i-1$ 和 i 之间插入一个元素，则线性表的长度变为 $n+1$，需将第 n 至第 i (共 $n-i+1$) 个元素向后移动一个位置。</p>
		链式存储结构	定义	<u>链式存储是指用一组任意的存储单元存储线性表的数据元素，这组存储单元可以是连续的也可以是不连续的。</u>
			特点	<u>插入或删除结点灵活（不必移动结点，只要改变结点中的指针即可）。</u>
			存储形式	<p>链式存储数据时，不仅要<u>存储数据元素本身的信息</u>，还要<u>存储其直接后继元素的存储位置（地址）</u>，来表示元素与其后继元素的关系，<u>由这两部分信息组成某个数据元素的存储映像，即结点。</u></p> <p><u>结点的组成：</u></p> <p>(1) <u>数据域：存储数据元素信息的域。</u></p> <p>(2) <u>指针域：存储直接后继存储位置的域。</u></p> <p>指针域中存储的信息称为指针或链。</p>

冲刺模拟

1. 线性表若采用链式存储结构时，要求内存中可用存储单元的地址（ ）。

- A. 部分地址必须是连续的 B. 必须是连续的
C. 一定是不连续的 D. 连续或不连续都可以

2. 用链表示线性表的优点是（ ）。

- A. 便于随机存取
B. 花费的存储空间较顺序存储少
C. 便于插入和删除操作
D. 数据元素的物理顺序与逻辑顺序相同

考点三 线性链表

单链表	定义	<u>每个结点中只包含一个指针域的链表</u> ，并且，线性链表中最后一个结点的指针为“空”(NULL)，即 <u>最后一个数据元素没有直接后继</u> 。
	头指针	指示链表中第一个结点的存储位置。
	<u>结点的插入</u>	在线性表的两个数据元素 a 和 b 之间插入一个数据元素 x，已知 p 为其单链表存储结构中指向结点 a 的指针，s 为指向结点 x 的指针。则插入语句可以是： <u>$s \rightarrow next = p \rightarrow next; p \rightarrow next = s;$</u>
	<u>结点的删除</u>	单链表中元素 a, b, c 相邻，p 为指向结点 a 的指针，在此链表中删除元素 b。那么，要删除 b，只需修改结点 a 中的指针域即可，不需移动元素。则删除语句可以是： <u>$p \rightarrow next = p \rightarrow next \rightarrow next;$</u>
双链表	定义	<u>在双链表的结点中有两个指针域，一个指向直接后继，另一个指向直接前趋</u> 。 若 d 为指向表中的某一个结点的指针，则 $d \rightarrow next \rightarrow prior = d \rightarrow prior \rightarrow next = d$ 。
循环链表	定义	<u>表中最后一个结点的指针域指向头结点</u> ，整个链表形成一个环。循环链表是链式存储结构的另一种形式， <u>从表中任一点出发均可找到表中其它结点</u> 。
	<u>双循环链表的插入结点</u>	在循环双链表的 p 结点之后插入 s 结点的操作可以是： <u>$s \rightarrow prior = p; s \rightarrow next = p \rightarrow next; p \rightarrow next \rightarrow prior = s; p \rightarrow next = s;$</u>
	<u>双循环链表的删除结点</u>	在循环双链表上删除结点 p 的直接后继结点 q 的操作可以是： <u>$q = p \rightarrow next; p \rightarrow next = q \rightarrow next; q \rightarrow next \rightarrow prior = p; free(q);$</u>

冲刺模拟

1. 已知 P 结点是某双向链表的中间结点，要删除 P 结点的直接后继结点的语句序列是()。

- A. $q = p \rightarrow next; p \rightarrow next = p \rightarrow next \rightarrow next; p \rightarrow next \rightarrow next \rightarrow prior = p; free(q);$
- B. $p \rightarrow next \rightarrow next \rightarrow prior = p; p \rightarrow next = p \rightarrow next \rightarrow next; q = p \rightarrow next; free(q);$
- C. $q = p \rightarrow next; p \rightarrow next \rightarrow prior = p; p \rightarrow next = p \rightarrow next \rightarrow next; free(q);$
- D. $q = p \rightarrow next; p \rightarrow next = p \rightarrow next \rightarrow next; p \rightarrow next \rightarrow prior = p; free(q);$

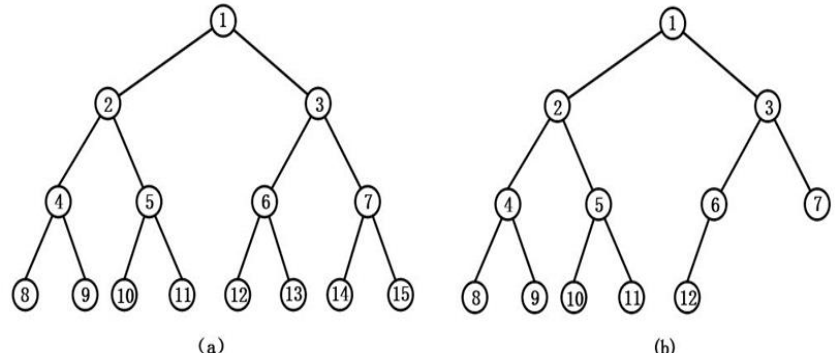
2. 已知 P 为单链表中的非首尾结点，删除 P 结点的后继结点 Q 的语句为 ()。

- A. $Q \rightarrow next = p; free(Q)$
- B. $P \rightarrow next = Q \rightarrow next; free(Q)$

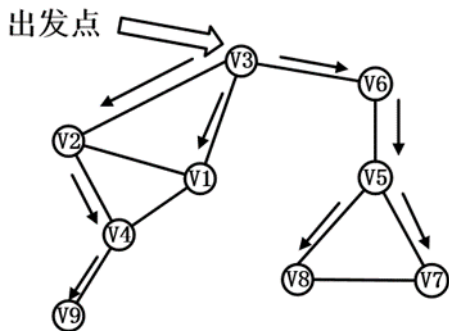
C. $Q \rightarrow next = P \rightarrow next$; $free(Q)$

D. $P \rightarrow next = S$; $S \rightarrow next = P$; $free(Q)$

考点四 队列、栈、二叉树和图

队列	定义	一种 <u>先进先出 (FIFO)</u> 的线性表。	
	特点	允许在表的一端进行插入，而在另一端删除元素。 <u>允许删除的一端叫做队头，允许插入的一端叫做队尾。</u> 队列是操作受限的线性表。	
栈	定义	限定 <u>仅在表尾（即栈顶）进行插入或删除操作</u> 的线性表。	
	特点	栈是操作受限的线性表，也称为限定性的数据结构。 空栈：不含元素的空表。 <u>栈是一种先进后出的线性表。</u>	
二叉树	定义	二叉树是 <u>每个结点最多有两个子树</u> 的树形存储结构。通常子树被称作“左子树”和“右子树”。	
	分类	满二叉树	一个二叉树，如果每一个层的结点数都达到最大值，并且除最后一层无任何子节点，则这个二叉树就是满二叉树。 <u>满二叉树是一棵深度为 k 且有 $2^k - 1$ 个结点的二叉树。</u>
		完全二叉树	深度为 k ，有 n 个结点的二叉树，当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应时，称之为完全二叉树。
			 <p>(a) (b)</p>
	性质	<u>性质 1：在二叉树的第 i 层至多有 2^{i-1} 个结点 ($i \geq 1$)。</u>	
		<u>性质 2：深度为 k 的二叉树至多有 $2^k - 1$ 个结点 ($k \geq 1$)。</u>	
		<u>性质 3：对任何一棵二叉树 T，如果其终端结点数为 n_0，度为 2 的结点数为 n_2，则 $n_0 = n_2 + 1$。</u>	

		性质 4: 具有 n ($n \geq 0$) 个结点的完全二叉树的深度为 $\lceil \log_2 n \rceil + 1$ 。	
	遍历二叉树	定义: 按某条搜索路径访问树中每个结点, 使得 <u>每个结点均被访问一次, 而且仅被访问一次。</u>	
		先序遍历	<u>①访问根结点;</u> <u>②遍历左子树;</u> <u>③遍历右子树。</u>
		中序遍历	<u>①遍历左子树;</u> <u>②访问根结点;</u> <u>③遍历右子树。</u>
		后序遍历	<u>①遍历左子树;</u> <u>②遍历右子树;</u> <u>③访问根结点。</u>
图	定义	图是一种比线性表和树更为复杂的数据结构, 结点之间的关系是任意的, 图中任意两个数据元素之间都可能相关。	
	图的遍历	深度优先搜索	<p>深度优先搜索 (DFS) 是对每一个可能的分支路径深入到不能再深入为止, 而且每个结点只能访问一次。它<u>类似于树的先序遍历</u>。</p> <p>遍历过程如下:</p> <ol style="list-style-type: none"> (1) 从图 G 中选某个顶点 v 作为出发点; (2) 访问 v; (3) 依次从 v 的未被访问的邻接点出发, 深度优先搜索遍历图 G, 直至与 v 相通的顶点都被访问完; (4) 如果此时图中还有顶点未曾被访问, 则从这些未被访问的顶点中再选出一个顶点, 重复 (2), 继续遍历; 否则, 遍历结束。
			<div style="text-align: center;"> <p>出发点 \Rightarrow V1</p> <p>深度优先搜索遍历的访问序列为: V1 V2 V5 V6 V7 V3 V4。</p> </div>

		广度优先搜索	<p>广度优先搜索 (BFS) 是指从上往下对每一层依次访问, 在每一层中, 从左往右 (也可以从右往左) 访问结点, 访问完一层就进入下一层, 直到没有结点可以访问为止。它是<u>树的层次遍历</u>的推广。</p> <p>遍历过程如下:</p> <p>(1) 从图 G 的某个顶点 v 出发, 访问 v;</p> <p>(2) 依次访问 v 的未被访问的邻接点;</p> <p>(3) 再按照“先被访问顶点的邻接点先访问”的次序, 依次访问这些邻接点的邻接点, 直至图中所有已被访问的顶点的邻接点都被访问到;</p> <p>(4) 若此时图中还有顶点未曾被访问, 则另选一个未被访问的顶点 v 作为出发点, 重复上述过程, 直至图中所有的顶点都被访问完。</p> <div style="text-align: center;">  <p>广度优先搜索遍历的访问序列: $V_3 V_2 V_1 V_6 V_4 V_5 V_9 V_8 V_7$。</p> </div>
--	--	--------	--

备注:

图的遍历是指从图中某一顶点出发访遍图中其余顶点, 且使每一个顶点仅被访问一次。

🌊 冲刺模拟 🌊

- 若进栈序列为 1、2、3、4, 则不可能得到的出栈序列为 ()。

A. 3、2、4、1
B. 3、2、1、4

C. 4、2、3、1
D. 2、3、4、1
- 假设在一颗 2 叉树中, 双分支结点数为 15 个, 单分支结点为 30 个, 则叶子结点数为 () 个。

A. 15
B. 16
C. 17
D. 18

第五章 数据库

考点一 三级模式和二级映像

三级模式	特点	三级模式能有效地组织和管理数据，并且三级模式之间提供了两层映像：外模式/模式映像和模式/内模式映像。正是 <u>三级模式间的两层映像保证了数据库系统中的数据具有较高的数据独立性。</u>	
	外模式	定义	<u>外模式，也称子模式或用户模式</u> ，是数据库用户（应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是 <u>数据库用户的数据视图</u> ，外模式是与某一应用有关的数据的逻辑表示。
		本质	外模式通常是模式的子集， <u>一个数据库可以有多个外模式。</u>
	模式	定义	<u>模式，也称逻辑模式</u> ，是数据库中全体数据的逻辑结构和特征的描述，是 <u>所有用户的公共数据视图</u> 。例如：模式是一个部门或者是公司的整体数据视图。
		本质	<u>模式实际上是数据库数据在逻辑上的视图，一个数据库只有一个模式。</u>
	内模式	定义	<u>内模式，也称存储模式</u> ，是数据物理结构和存储方式的描述，是数据在数据内部的表示方式。
		本质	<u>一个数据库只有一个内模式。</u>
二级映像	外模式/模式映像	概述	模式描述的是数据的 <u>全局逻辑结构</u> ，外模式描述的是数据的 <u>局部逻辑结构</u> 。 <u>对于每一个外模式，数据库系统都有一个外模式/模式映像。</u>
		作用	定义了该外模式与模式之间的对应关系。
		特征	<u>如果模式的结构发生改变，如添加字段，修改字段的类型等等，但这些模式的改变不一定会影响外模式。保持了逻辑的独立性。</u>
	模式/内模式映像	概述	数据库中只有一个模式，也只有一个内模式，则 <u>模式/内模式映像是唯一</u> 的。
		作用	定义了数据全局逻辑结构与存储结构之间的对应关系。
		特征	<u>如果数据库的存储结构发生了变化，也就是说如果改变了存储结构的定义，那么模式/内模式映像必须进行相应的更改，以使模式保持不变。内模式改变可以不改变模式。保持了物理的独立性。</u>

冲刺模拟

1.数据库系统依靠（ ）支持了数据独立性。

- A. DDL 语言和 DML 语言互相独立
- B.具有封装机制
- C.定义完整性约束条件
- D.模式分级、各级之间的映像机制

2.数据库管理系统 DBMS 中用来定义模式、内模式和外模式的语言为（ ）。

- A. C
- B. Basic
- C. DDL
- D. DML

考点二 关系运算

<u>并</u>	设有两个关系 R 和 S，它们具有相同的结构。 <u>R 和 S 的并是由属于 R 和属于 S 的元组组成的集合，运算符为 \cup。记为 $T=R \cup S$。</u>
<u>差</u>	<u>R 和 S 的差是由属于 R 但不属于 S 的元组组成的集合，运算符为 $-$。记为 $T=R-S$。</u>
<u>交</u>	<u>R 和 S 的交是由既属于 R 又属于 S 的元组组成的集合，运算符为 \cap。记为 $T=R \cap S$。 $R \cap S = R - (R - S)$。</u>
<u>选择运算 (σ)</u>	<p><u>从关系中找出满足给定条件的那些元组称为选择</u>。其中的条件是以逻辑表达式给出的，值为真的元组将被选取。这种运算是从水平方向抽取元组。</p> <p>设有一个学生-课程数据库，包括学生关系 Student，第一列是学号 Sno，第二列是姓名 Sname，第三列是性别 Ssex，第四列是年龄 Sage，第五列是系 Sdept……。</p> <p>如查询年龄小于 20 岁的学生： $\sigma \text{ Sage} < 20 (\text{Student})$ 或 $\sigma 4 < 20 (\text{Student})$。</p>
<u>连接运算 (\bowtie)</u>	<p>连接也称 θ 连接，<u>连接运算是从两个关系的笛卡尔积中选择属性间满足一定条件的元组</u>。这个条件为 θ。连接运算其实就是在笛卡尔积运算的基础上限定了条件（某列大于、小于、等于某列），只匹配和条件相符合的，从而得出结果。</p> <div style="text-align: center;"> $R \bowtie_{A\theta B} S$ </div> <p>其中 A 和 B 分别为 R 和 S 上度数相等且可比的属性组，θ 是比较运算符，连接运算从 R 和 S 的笛卡尔积 ($R \times S$) 中选取 R 关系在 A</p>

	<p>属性组上的值域、S 关系在 B 属性组上的值域满足比较关系 θ 的元组。</p> <p>R, S 如图 求 $R \bowtie_{C>D} S$</p> <div><div>关系 R</div><table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>a1</td><td>b1</td><td>3</td></tr><tr><td>a1</td><td>b2</td><td>6</td></tr><tr><td>a2</td><td>b2</td><td>5</td></tr><tr><td>a3</td><td>b3</td><td>11</td></tr></table></div> <div><div>关系 S</div><table><tr><th>D</th><th>E</th></tr><tr><td>4</td><td>e1</td></tr><tr><td>7</td><td>e2</td></tr><tr><td>15</td><td>e3</td></tr></table></div> <p>连接结果 $R \bowtie_{C>D} S$ 关系</p> <table><tr><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th></tr><tr><td>a1</td><td>b2</td><td>6</td><td>4</td><td>e1</td></tr><tr><td>a2</td><td>b2</td><td>5</td><td>4</td><td>e1</td></tr><tr><td>a3</td><td>b3</td><td>11</td><td>4</td><td>e1</td></tr><tr><td>a3</td><td>b3</td><td>11</td><td>7</td><td>e2</td></tr></table>	A	B	C	a1	b1	3	a1	b2	6	a2	b2	5	a3	b3	11	D	E	4	e1	7	e2	15	e3	A	B	C	D	E	a1	b2	6	4	e1	a2	b2	5	4	e1	a3	b3	11	4	e1	a3	b3	11	7	e2
A	B	C																																															
a1	b1	3																																															
a1	b2	6																																															
a2	b2	5																																															
a3	b3	11																																															
D	E																																																
4	e1																																																
7	e2																																																
15	e3																																																
A	B	C	D	E																																													
a1	b2	6	4	e1																																													
a2	b2	5	4	e1																																													
a3	b3	11	4	e1																																													
a3	b3	11	7	e2																																													
<u>投影运算 (π)</u>	<p><u>从关系模式中挑选若干属性组成新的关系称为投影。</u>这是从列的角度进行的运算，相当于对关系进行垂直分解。</p> <p>设有一个学生-课程数据库，包括学生关系 Student，第一列是学号 Sno，第二列是姓名 Sname，第三列是性别 Ssex，第四列是年龄 Sage，第五列是系 Sdept……。</p> <p>查询学生的姓名和所在系，即求 Student 关系上学生姓名和所在系；两个属性上的投影：$\pi_{Sname, Sdept}(Student)$ 或 $\pi_{2, 5}(Student)$。</p>																																																
<u>笛卡尔积 (\times)</u>	<p><u>两个分别为 n 目和 m 目的关系 R 和 S 的笛卡尔积是一个 (n+m) 列的元组的集合。</u>元组的前 n 列是关系 R 的一个元组，后 m 列是关系 S 的一个元组。若 R 有 k1 个元组、b1 个属性，S 有 k2 个元组、b2 个属性，则关系 R 和关系 S 的笛卡尔积有 $k1 \times k2$ 个元组，$b1+b2$ 个属性。记作 $R \times S$。</p> <div><div>关系 R</div><table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table></div> <div><div>关系 S</div><table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>2</td><td>4</td><td>6</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table></div> <p>$R \times S$</p> <table><tr><th>R.A</th><th>R.B</th><th>R.C</th><th>S.A</th><th>S.B</th><th>S.C</th></tr><tr><td>1</td><td>2</td><td>3</td><td>2</td><td>4</td><td>6</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr><tr><td>4</td><td>5</td><td>6</td><td>2</td><td>4</td><td>6</td></tr><tr><td>4</td><td>5</td><td>6</td><td>4</td><td>5</td><td>6</td></tr></table>	A	B	C	1	2	3	4	5	6	A	B	C	2	4	6	4	5	6	R.A	R.B	R.C	S.A	S.B	S.C	1	2	3	2	4	6	1	2	3	4	5	6	4	5	6	2	4	6	4	5	6	4	5	6
A	B	C																																															
1	2	3																																															
4	5	6																																															
A	B	C																																															
2	4	6																																															
4	5	6																																															
R.A	R.B	R.C	S.A	S.B	S.C																																												
1	2	3	2	4	6																																												
1	2	3	4	5	6																																												
4	5	6	2	4	6																																												
4	5	6	4	5	6																																												

冲刺模拟

1. 关系模式 $R(a,b,c,d)$ 中关系代数表达式 $\sigma_{3<4}(R)$ 等价于 SQL 语句 ()。
- A. `select * from R where c < '4'`
 B. `select * from R where d < 3`
 C. `select c from R where c < 3`
 D. `select c,d from R having d < 2`
2. 下面选项的集合运算, () 可以让属于 R 但不属于 S 的元组组成集合。
- A. 并 B. 交 C. 投影 D. 差

考点三 关系的三类完整性约束

实体完整性	定义	实体完整性要求每一个表中的 <u>主键字段都不能为空或者重复的值</u> 。
	规则说明	一个关系对应现实世界中一个实体集。现实世界中的实体是可以相互区分、识别的, 即它们应具有某种惟一性标识。 <u>在关系模式中, 以主关键字作为惟一性标识, 而主关键字中的属性(称为主属性)不能取空值</u> , 否则, 表明关系模式中存在着不可标识的实体(因空值是“不确定”的), 这与现实世界的实际情况相矛盾, 那么这样的实体就不是一个完整实体。
参照完整性	定义	参照完整性是指要求通过定义的 <u>主关键字与外部关键字之间的引用规则来约束两个关系之间的联系</u> 。
	规则说明	关系数据库中通常都包含多个存在相互联系的关系, 关系与关系之间的联系是通过公共属性来实现的。所谓公共属性, 它是一个关系 R (称为被参照关系) 的主关键字, 同时又是另一关系 K (称为参照关系) 的外部关键字。 <u>参照关系 K 中外部关键字的取值, 要么取空值, 要么与被参照关系 R 中某元组主关键字的值相同</u> , 那么, 在这两个关系间建立关联的主关键字和外部关键字引用, 符合参照完整性规则要求。
	示例	<p>学生实体和专业实体可以用下面的关系来表示, 其中主键用下划线标识:</p> <p>学生 (<u>学号</u>, 姓名, 性别, 专业号, 年龄);</p> <p>专业 (<u>专业号</u>, 专业名)。</p> <p>分析: 学生关系中的“专业号”属性与专业关系中的主键“专业号”相对应, 因此“专业号”属性是学生关系的外键。这里专业关系是被参照关系, 学生关系为参照关系。</p> <p>那么, 学生关系中每个元组的“专业号”属性只能取下面两类值:</p> <p>(1) 空值, 表示尚未给该学生分配专业;</p> <p>(2) 非空值, 此时该值必须是专业关系中某个元组的“专业号”值, 表示该学生不可能分配到一个不存在的专业。即被参照关系“专业”中一定</p>

		存在一个元组，它的主键等于该参照关系“学生”中的外键。
用户定义完整性	定义	用户定义完整性则是 <u>根据应用环境的要求和实际需要，对某一具体应用所涉及的数据提出约束性条件。</u>
	规则说明	用户定义完整性主要包括 <u>字段有效性约束和记录有效性。</u>
	示例	<u>某个属性必须取唯一值，某个非主属性也不能取空值，某个属性的取值范围在 0-100 之间等。</u>

🌊 冲刺模拟 🌊

- 1.在关系数据库中建立数据库表时，将年龄字段值限制在 12~40 岁之间的这种约束属于（ ）。

A.用户定义完整性约束
B.视图完整性约束

C.参照完整性约束
D.实体完整性约束
- 2.如果某一个字段被定义为主键，那么该字段（ ）。

A.不能为空且不能重复
B.可以为空

C.可以重复
D.能够支持所用情况

考点四 数据类型、数据定义语言和数据操纵语言

数据类型	CHAR(n)	长度为 n 的定长字符串	
	VARCHAR(n)	最大为 n 的变长字符串	
	INT	长整数（INTEGER）	
	SMALLINT	短整数	
	DOUBLE PRECISION	取决于机器精度的双精度浮点数	
	FLOAT(n)	浮点数，精度至少为 n 位数字	
数据定义语言	创建表	格式	<u>CREATE TABLE <表名>(<列名> <数据类型> [完整性约束条件]……);</u>

		示例	<p>建立一个学生表 Student。</p> <pre>CREATE TABLE Student (Sno CHAR(9) PRIMARY KEY, //完整性约束条件, Sno 是主键 Sname CHAR(20) UNIQUE, Ssex CHAR(2), Sage SMALLINT, Sdept CHAR(20));</pre> <p>说明：在数据库中建立一个新的空的学生表 Student，并将有关学生表的定义及有关约束条件存放在数据字典中。</p>
		修改表	<p><u>ALTER TABLE <表名></u> <u>[ADD <新列名> <数据类型> [完整性约束]]</u> <u>[DROP <完整性约束名>]</u> <u>[ALTER COLUMN <列名> <数据类型>]</u></p> <p>说明：</p> <p>(1) <表名>是要修改的基本表；</p> <p>(2) <u>ADD 子句：增加新列和新的完整性约束条件；</u></p> <p>(3) <u>DROP 子句：删除指定的完整性约束条件；</u></p> <p>(4) <u>ALTER COLUMN 子句：修改原有的列定义（列名和数据类型）。</u></p>
		示例	<p>将学生表中年龄的数据类型由字符型改为整数。</p> <pre>ALTER TABLE Student ALTER COLUMN Sage INT;</pre>
		删除表	<p><u>DROP TABLE <表名> [RESTRICT CASCADE];</u></p> <p>说明：</p> <p>(1) <u>选择 RESTRICT，删除表有限制条件：删除的基本表不能被其他表的约束所引用（如 CHECK，FOREIGN KEY 等约束），不能有视图、触发器、存储过程或函数等，若存在这些依赖该表的对象，则表不能被删除。</u></p> <p>(2) <u>选择 CASCADE：删除表时没有限制条件，删除表以及相关的依赖对象（如视图）。</u></p> <p>(3) 缺省情况是 RESTRICT。</p>

		示例	<p>若表上建有视图，选择 RESTRICT 时不能删除表，CASCADE 时可以删除表以及视图。</p> <pre>CREATE VIEW IS_Student //Student 表上建立视图 AS SELECT Sno,Sname,Sage FROM Student WHERE Sdept='IS'; DROP TABLE Student RESTRICT; //删除 Student 表 --ERROR: cannot drop table Student because other objects depend on it //系统返回错误信息，存在该依赖表的对象，不能删除。 DROP TABLE Student CASCADE; //删除 Student 表 --NOTICE: drop cascades to view IS_Student //系统返回提示，此表以及其上的视图一并被删除了。</pre>
数据操纵语言	插入元组	格式	<p><u>INSERT INTO <表名> [(<属性列 1> [, <属性列 2>] ...)]</u> <u>VALUES(<常量 1> [, <常量 2>] ...);</u></p> <p>说明：</p> <p>(1) 功能：将新元组插到指定表中。</p> <p>(2) <u>新元组的属性列 1 的值为常量 1</u>，属性列 2 的值为常量 2，…。</p> <p>(3) INTO 子句中没有出现的属性列，其值为空值，但若<u>在表定义时说明了 NOT NULL 的属性列不能取空值</u>，否则出错。</p> <p>(4) <u>若 INTO 子句中没有指明任何属性列名，则新插入的元组必须在每个属性列上均有值。</u></p>
		示例	<p>将一个新学生元组（学号：201515128；姓名：左耳；性别：男；所在系：MATH；年龄：18 岁）插入到 Student 表中。</p> <pre>INSERT INTO Student(Sno,Sname,Ssex,Sdept,Sage) VALUES ('201515128' , '左耳' , '男' , 'MATH' , 18);</pre>
	修改数据	格式	<p><u>UPDATE <表名> SET <列名>=<表达式>……[WHERE <条件>];</u></p> <p>功能：修改指定表中满足 WHERE 子句条件的元组。</p> <p><u>SET 子句给出的 <表达式> 的值用于取代相应的属性列的值，若省略了 WHERE 子句，则表示要修改表中的所有元组。</u></p>

	示例	例：在 Student 表中，将学号为 201515121 学生的年龄修改为 22 岁。 UPDATE Student SET Sage=22 WHERE Sno='201515121';
	删除数据	<u>DELETE FROM <表名> [WHERE <条件>];</u> 功能：从指定表中删除满足 WHERE 子句条件的所有元组。 <u>若省略 WHERE 子句，表示删除表中全部元组</u> ，DELETE 语句删除的是表中的数据，而不是关于表的定义。
	示例	例：在 Student 表中，删除学号为 201515128 的学生记录。 DELETE FROM Student WHERE Sno='201515128';

❧ 冲刺模拟 ❧

1.使用 SQL 语句向学生表 S(SNO, SN, AGE, SEX)中添加一条新记录，字段学号(SNO)、姓名(SN)、性别(SEX)、年龄(AGE)的值分别为 0401、王芳、女、18，正确的命令是()。

- A. APPEND S VALUES ('0401','王芳', 18,'女')
- B. APPEND INTO S (SNO,SN,SEX,AGE) VALUES ('0401','王芳','女',18)
- C. INSERT S VALUES ('0401','王芳',18,'女')
- D. INSERT INTO S (SNO,SN,SEX,AGE) VALUES ('0401','王芳','女',18)

2.下列四组 SQL 命令，全部属于数据定义语句的命令是()。

- A. CREATE, DROP, GRANT
- B. CREATE, DROP, UPDATE
- C. CREATE, DROP, SELECT
- D. CREATE, DROP, ALTER

考点五 数据查询

数据查询一般格式	<p><u>SELECT [ALL DISTINCT] <目标列表表达式>[,<目标列表表达式>]...</u></p> <p><u>FROM <表名或视图名>[,<表名或视图名>]...</u></p> <p><u>[WHERE <条件表达式>]</u></p> <p><u>[GROUP BY <列名 1> [HAVING <条件表达式>]]</u></p> <p><u>[ORDER BY <列名 2> [ASC DESC]]</u></p> <p>说明:</p> <p>(1) 整个 SELECT 语句的定义: 根据 WHERE 子句的条件表达式, 从 FROM 子句指定的基本表或视图中找出满足条件的元组, 再按 SELECT 子句中的目标列表表达式, 选出元组中的属性形成结果表。</p> <p>(2) <u>在 GROUP BY 子句中, <列名 1>表示将结果按它的值进行分组, 该属性列值相等的元组为一个组。</u>通常会在每组中作用聚集函数。<u>如果 GROUP BY 子句带 HAVING 短语, 则只有满足指定条件的组才予以输出。</u></p> <p>(3) <u>在 ORDER BY 子句中, <列名 2>表示将结果按它的值升序或降序排序。</u></p>
----------	---

查询满足条件的元组	方法	我们首先要确定查询条件的范围, 使用谓词 <u>BETWEEN...AND...和 NOT BETWEEN...AND...</u> 来查找属性值在或不在指定范围内的元组, 其中 <u>BETWEEN</u> 后是范围的下限 (低值), <u>AND</u> 后是范围的上限 (高值)。
	示例	在学生表 Student 中, 查询年龄在 20~23 岁 (包括 20 岁和 23 岁) 之间的学生的姓名、系和年龄。 SELECT Sname,Sdept,Sage FROM Student WHERE Sage BETWEEN 20 AND 23;
模糊查询的字符匹配	方法	使用谓词 <u>LIKE</u> 可以用来进行字符匹配; 一般格式为: <u>[NOT] LIKE '<匹配串>'</u> 说明: 查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串或含有通配符 '%' 和 '_' 的字符串。 (1) <u>%: 百分号代表任意长度的字符串 (可匹配 0 个)。</u> 如 a%b 表示以 a 开头, 以 b 结尾的任意长度的字符串, 如通过 a%b 可以找到 acb, ab 等。 (2) <u>_: 下划线代表任意单个字符。</u> 如 a_b 表示以 a 开头, 以 b 结尾长度为 3 的任意字符串, 如通过 a_b 可以找到 acb, afb 等。
	示例	在学生表 Student 中, 查询名字中第 2 个字为“阳”字的学生姓名和学号。 SELECT Sname,Sno FROM Student WHERE Sname LIKE ' __阳%';

涉及空值的查询	方法	<u>判断一个属性的值是否为空值，用 IS NULL 或 IS NOT NULL 来表示。</u>
	示例	<p>某些学生选修课程后没有参加考试，所以这个学生有选课记录，但没有考试成绩。在学生选课表 SC 中，查询没有考试成绩的学生的学号和相应的课程号。</p> <p>SELECT Sno,Cno FROM SC WHERE Grade IS NULL; //分数 Grade 是空值</p> <p>注：这里的“IS”不能用“=”代替。</p>
ORDER BY 子句	方法	<u>用 ORDER BY 子句对查询结果按照一个或多个属性列的升序（ASC）或降序（DESC）排列，缺省值为 ASC。</u>
	示例	<p>在学生选课表 SC 中，查询选修了 3 号课程的学生的学号及其成绩，查询结果按分数的降序排列。</p> <p>SELECT Sno,Grade FROM SC WHERE Cno='3' ORDER BY Grade DESC;</p>
GROUP BY 子句	方法	<p>1.<u>GROUP BY 子句将查询结果按某一列或多列的值分组，值相等的为一组。</u></p> <p>2.分组的目的：细化聚集函数的作用对象。分组后聚集函数将作用于每一个组（每一组都有一个函数值）。</p>
	示例	<p>在学生选课表 SC 中，查询选修了 3 门以上课程的学生学号。</p> <p>SELECT Sno FROM SC GROUP BY Sno HAVING COUNT(*)>3;</p> <p>说明：先用 GROUP BY 子句按 Sno 进行分组，再用聚集函数 COUNT 对每一组计数。<u>HAVING 短语后跟选组的条件</u>，即只有满足条件（元组个数>3，表示此学生选修的课程超过 3 门）的组才会被选出来。</p>
	区别	<p><u>(1) WHERE 子句作用于基本表或视图，从中选择满足条件的行；</u></p> <p><u>(2) HAVING 短语作用于组，从中选择满足条件的组。</u></p>
带有 EXISTS 谓词的子查询	方法	<p>1. EXISTS 谓词：<u>带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值（true）或逻辑假值（false）。</u></p> <p>2.<u>使用 EXISTS 的查询：若内层查询结果非空，则外层的 WHERE 子句返回真值，否则返回假值。</u></p> <p>3. EXISTS 引出的子查询的目标列表表达式通常都用“*”，因为带 EXISTS 的子查询只返回真值或假值，给出列名无实际意义。</p>

	示例	<p>(1) 通过学生表 (Student) 和选课表 (SC), 查询所有选修了 1 号课程的学生姓名。</p> <pre>SELECT Sname FROM Student WHERE EXISTS(SELECT * FROM SC WHERE SC. Sno=Student.Sno AND Cno='1');</pre> <p>说明: 首先取外层查询中 (Student) 表的第一个元组, 根据它与内层查询相关的属性值 (Sno 值) 处理内层查询, 若 WHERE 子句返回值为真, 则取外层查询中该元组的 Sname 放入结果表, 然后再取 (Student) 表的下一个元组, 重复这一过程, 直至外层 (Student) 表全部检查完为止。</p> <p>(2) 查询没有选修 1 号课程的学生姓名。</p> <pre>SELECT Sname FROM Student WHERE NOT EXISTS(SELECT * FROM SC WHERE Sno=Student.Sno AND Cno='1');</pre>	
集合查询	查询原则	多个 SELECT 语句的结果可以进行集合操作, 但是, 参加集合操作的各查询结果的列数必须相同, 对应项的数据类型也必须相同。	
	并	定义	<u>并的关键字是 UNION, 是指合并两个或多个 SELECT 语句的结果集, 并且去除重复数据。</u> 若要保留重复元组则用 UNION ALL 操作符。
		示例	<p>TABLE: A = (1,2,3)</p> <p>TABLE: B = (2,4,5)</p> <pre>SELECT * FROM A UNION SELECT * FROM B</pre> <p>RESULT: 1,2,3,4,5</p>
	交	定义	<u>交的关键字是 INTERSECT, 是指在两个集合中都存在的数据。</u>
		示例	<p>TABLE: A = (1,2,3)</p> <p>TABLE: B = (2,4,5)</p> <pre>SELECT * FROM A INTERSECT SELECT * FROM B</pre> <p>RESULT: 2</p>
	差	定义	<u>差的关键字是 EXCEPT, 是指在第一个集合中存在, 但是不存在于第二个集合中的数据。</u>
		示例	<p>TABLE: A = (1,2,3)</p> <p>TABLE: B = (2,4,5)</p> <pre>SELECT * FROM A EXCEPT SELECT * FROM B</pre> <p>RESULT: 1,3</p>

冲刺模拟

1. 通过 SQL, 如何从“Persons”表中选取“FirstName”列的值以“a”开头的所有记录? ()
- A. SELECT * FROM Persons WHERE FirstName LIKE 'a%'
- B. SELECT * FROM Persons WHERE FirstName='a'
- C. SELECT * FROM Persons WHERE FirstName LIKE '%a'
- D. SELECT * FROM Persons WHERE FirstName='%a%'
2. 下列哪个查询所返回的结果集中的行只在表 T1 中存在, 而在表 T2 中不存在? ()
- A. SELECT * FROM T1 MINUS SELECT * FROM T2
- B. SELECT * FROM T2 UNION EXCEPT SELECT * FROM T1
- C. SELECT * FROM T1 INTERSECT SELECT * FROM T2
- D. SELECT * FROM T1 NOT EXISTS SELECT * FROM T2

考点六 事务与并发操作问题

事务	定义	事务是用户定义的一个数据库操作序列, 一般 <u>一个程序中包含多个事务</u> 。 例如: 在关系数据库中, 一个事务可以是一条 SQL 语句、一组 SQL 语句或整个程序。	
	特性	原子性	事务是数据库的逻辑工作单位, 事务中包括的 <u>操作要么都做, 要么都不做</u> 。
		一致性	事务执行的结果必须是使 <u>数据库从一个一致性状态变到另一个一致性状态</u> 。如果数据库中只包含成功事务提交的结果, 就说数据库处于一致性状态。
		隔离性	<u>一个事务的执行不能被其他事务干扰</u> 。即一个事务内部的操作及使用的数据对其他并发事务是隔离的, 并发执行的各个事务之间不能互相干扰。
		持续性	持续性也称 <u>永久性</u> , 指 <u>一个事务一旦提交, 对数据库中数据的改变就应该是永久的</u> , 接下来的 <u>其他操作或故障不应该对其执行结果有任何影响</u> 。
并发操作问题	丢失更新问题	定义	在多用户环境中, 同一时间可能会有多个用户更新相同的记录, 这会产生冲突, 而这就是并发操作的问题。并发操作带来的丢失更新问题指的是当两个或多个事务读取同一数据并修改, <u>一个事务的更新覆盖了另一个事务的更新</u> 。

	不可重复读问题	示例	用户 A 把值从 6 改为 2，用户 B 把值从 2 改为 6，则用户 A 丢失了他的更新。
		定义	前一事务读取某个数据后，后一事务执行了对该数据的更新，前一事务再次读取该数据时（希望与第一次读取的是相同的值），得到的数据与前一次的不一樣，这是由于前一事务第一次读取数据后，后一事务对其做了修改，导致 <u>前一事务再次读取数据时与第一次读取的数据不相同</u> 。
	读「脏」数据	示例	用户 A 读取到某一数值 3，用户 B 读取它并把它修改为 5，A 为了对读取值进行检验而再次读取该数据，不再是 3 而是 5，得到了不同的结果。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为不可重复读，即原始读取不可重复。
		定义	当一个事务修改某个数据后，另一事务对该数据进行了读取，由于某种原因前一事务 <u>撤销了对数据的修改，将修改过的数据恢复原值</u> ，而 <u>后一事务读出的数据与数据当前值不一致</u> ，因此，后一事务 <u>读取到的是一个无效的数据</u> ，称之为 <u>读“脏”数据</u> 。
		示例	用户 A 把 2 改为 6，此时，用 B 进行读取数据，用户 A 和用户 B 看到的值都是 6，然后用户 A 撤销操作，值改回为 2，此时，用户 B 读到的值仍为 6，这就是无效的数据，即“脏”数据。

❧ 冲刺模拟 ❧

- （多选题）以下选项当中，属于数据库的并发操作带来的问题的是（ ）。

A.不断更新
B.不可重复读
C.读“脏”数据
D.数据回滚
- 当两个或多个事务读取同一数据并修改，其中一个事务的更新覆盖了另一个事务的更新，这种情况属于数据库并发操作的（ ）问题。

A.读“错”数据
B.“脏”读数据

C.数据冗余
D.丢失更新

第六章 软件工程

考点一 软件的生存周期和开发模型

软件的生存周期	定义	软件的生存周期是指 <u>从形成开发软件概念起，所开发的软件使用以后，直到失去使用价值直至消亡的整个过程。</u>	
	三个时期	<u>定义时期</u>	<u>可行性研究、需求分析</u>
		<u>开发时期</u>	<u>概要设计、详细设计、编码和测试</u>
		<u>运行时期</u>	<u>运行、维护、更新</u>
	六个阶段	<u>(1) 软件计划与可行性研究；(2) 需求分析；(3) 软件设计；(4) 编码；(5) 测试；(6) 运行与维护。</u>	
开发模型	瀑布模型	<p>(1) 各个阶段的顺序性和依赖性。</p> <p>(2) 每个阶段必须完成规定的文档，通过复审及早发现其中问题，及早解决。</p> <p>(3) <u>文档驱动，系统可能不满足客户的需求。</u></p>	
	快速原型模型	<p>(1) <u>第一步是建造一个快速原型</u>，实现客户或未来的用户与系统的交互，用户或客户对原型进行评价，进一步细化待开发软件的需求；第二步则在第一步的基础上开发客户满意的软件产品。</p> <p>(2) 关键在于<u>尽可能快速地建造出软件原型，一旦确定了客户的真正需求，所建造的原型将被丢弃。</u></p> <p>(3) 特点：<u>关注满足客户需求，但可能导致系统设计差、效率低，难于维护。</u></p>	
	增量模型	<p>(1) <u>从部分需求出发，先建立一个不完全的系统</u>，通过测试运行该系统取得经验和信息反馈，加深对软件需求的理解，<u>进一步使系统扩充和完善</u>。如此反复，直至软件人员和用户对所设计完成的软件系统满意为止。</p> <p>(2) <u>在渐增型开发下的软件是随软件开发的过程而逐渐形成的。</u></p> <p>(3) 渐增型开发方法适合于知识型软件的开发，设计系统时<u>对用户需求的认识开始不是很清楚的</u>，需要在开发过程中不断认识、不断获得新的知识去丰富和完善系统。多数研究性质的试验软件，一般采用此方法。</p> <p>(4) 特点：<u>开发早期反馈及时</u>，易于维护，需要开放式体系结构，<u>但可能会导致效率低下。</u></p>	

	螺旋模型	<p>(1) 将瀑布模型和快速原型模型结合起来，<u>强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。</u></p> <p>(2) 由风险驱动，强调可选方案和约束条件从而支持软件的重用，有助于将软件质量作为特殊目标融入产品开发之中。</p> <p>(3) 特点：<u>风险驱动</u>，风险分析人员需要有经验且经过充分训练。</p>
--	------	--

❧ 冲刺模拟 ❧

- 通常在软件的（ ）活动中无需用户参与。
A.需求分析 B.维护 C.编码 D.测试
- 原型化方法是用户和设计者之间执行的一种交互构成，适用于（ ）。
A.需求确定的 B.需求不确定性高的
C.管理信息 D.实时

考点二 软件测试

定义	在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程。	
目的	<u>为了发现程序中的错误。</u>	
测试对象	<u>程序、数据、文档</u> 。其中最主要的还是程序。	
分类	从是否执行程序的角度划分	<p>(1) 静态测试：指不运行被测程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性。</p> <p>(2) 动态测试：指通过运行软件来检验软件的动态行为和运行结果的正确性。</p>

从是否关心软件内部结构和具体实现的角度划分	白盒测试法	定义	白盒测试又称 <u>结构测试、透明盒测试</u> 、逻辑驱动测试或 <u>基于代码的测试</u> ，白盒法 <u>全面了解程序内部逻辑结构</u> ，对所有逻辑路径进行测试。
		覆盖标准	白盒测试法的覆盖标准有逻辑覆盖、循环覆盖和基本路径测试。其中 <u>逻辑覆盖包括语句覆盖</u> （每条语句至少执行一次）、 <u>判定覆盖</u> （每个判定的每个分支至少执行一次）、 <u>条件覆盖</u> （每个判定的每个条件应取到各种可能的值）、判定/条件覆盖、条件组合覆盖和路径覆盖，六种覆盖标准发现错误的能力呈由弱到强的变化。
	黑盒测试法	定义	<u>黑盒测试也称功能测试</u> ，它是通过测试来检测每个功能是否都能正常使用。 <u>黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。</u>
		测试分类	<u>等价类划分法</u> 是指 <u>把程序的输入域划分成若干部分</u> ，然后从每个部分中选取少数代表性数据作为测试用例。 <u>每一类的代表性数据在测试中的作用等价于这一类中的其它值。</u>
			<u>边界值分析法</u> 是指通过 <u>选择等价类边界的测试用例</u> 。边界值分析法不仅重视输入条件边界，而且也必须考虑输出域边界。它是对等价类划分方法的补充。
			<u>错误推测法</u> 是指在测试程序时，人们可以根据经验或直觉推测程序中可能存在的各种错误，从而有针对性地编写检查这些错误的测试用例的方法。
			<u>因果图法</u> 是一种描述对于多种输入条件组合的测试方法， <u>根据输入条件的组合、约束关系和输出条件的因果关系，分析输入条件的各种组合情况，从而设计测试用例的方法</u> ，它适合于检查程序输入条件涉及的各种组合情况。
	灰盒测试法	定义	灰盒测试是 <u>介于白盒测试与黑盒测试之间的一种测试</u> ，灰盒测试多用于集成测试阶段，不仅关注输入、输出的正确性，同时也关注程序内部的情况。
		特点	灰盒测试不像白盒那样详细、完整，但又比黑盒测试更关注程序的内部逻辑，常常是通过一些表征性的现象、事件、标志来判断内部的运行状态。

测试过程	单元测试	<u>单元测试，又称模块测试</u> ，是针对软件设计的最小单位——程序模块，进行正确性检验的测试工作。 <u>单元测试使用白盒测试技术</u> ，从程序的内部结构出发设计测试用例， <u>发现各模块内部可能存在的各种差错</u> 。
	集成测试	<u>集成测试，也称组装测试或联合测试</u> ，在单元测试的基础上， <u>把已测试过的模块组装成一个系统进行测试</u> ，主要 <u>对与设计相关的软件体系结构的构造进行测试</u> 。
	确认测试	<u>确认测试，又称有效性测试</u> ，是检查已实现的软件 <u>是否满足需求规格说明中确定的各种需求</u> ，以及软件配置是否完全、正确。
	系统测试	<u>系统测试</u> 是将通过确认测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等 <u>其它系统元素结合在一起</u> ，在实际运行环境下，对计算机系统进行一系列的组装测试和确认测试。

备注：

测试只能证明程序中存在错误，但不能证明程序中不存在错误。

🌊 冲刺模拟 🌊

1.下列测试方法属于白盒测试方法的是（ ）。

- A.等价划分和错误推测法
- B.判定覆盖和边界值分析法
- C.路径覆盖和判定/条件覆盖法
- D.条件覆盖和错误推测法

2.对管理信息系统程序检查功能、性能要求是否达到，文档资料是否正确完整，以及其他要求等是否满足的测试属于（ ）。

- A.系统测试
- B.组装测试
- C.单元测试
- D.确认测试

考点三 软件维护

定义	<u>软件维护是指软件系统交付使用以后，为了改正软件运行错误，或者因满足新的需求而加入新功能的修改软件的过程。</u>
特点	<ul style="list-style-type: none"> (1) <u>软件维护是软件生存周期中延续时间最长、工作量最大的一个阶段。</u> (2) <u>维护的代价高。</u> (3) <u>维护不当，极易产生很多副作用。</u> (4) 维护的问题比较多。

	(5) 有关维护方面的文献、技术和手段比较缺乏，故维护难度比较大。	
类型	<u>改正性维护</u>	改正性维护是指 <u>改正在系统开发阶段已发生而系统测试阶段尚未发现的错误</u> 。
	<u>适应性维护</u>	适应性维护是指 <u>使软件适应信息技术变化和管理需求变化而进行的修改</u> 。
	<u>完善性维护</u>	完善性维护指 <u>为扩充功能和改善性能而进行的修改</u> ，主要是指对已有的软件系统增加一些在系统分析和设计阶段中 <u>没有规定的功能与性能特征</u> 。 <u>完善性维护占整个维护工作量的 50%。</u>
	<u>预防性维护</u>	预防性维护是指为了改进应用软件的可靠性和可维护性， <u>为了适应未来的软硬件环境的变化，应主动增加预防性的新的功能</u> ，以使应用系统适应各类变化而不会被淘汰。

❧ 冲刺模拟 ❧

- 软件生命周期中所花费最多的阶段是（ ）。
 A. 详细设计 B. 软件编码 C. 软件测试 D. 软件维护
- 产生软件维护的副作用，是指（ ）。
 A. 开发时的错误 B. 隐含的错误
 C. 因修改软件而造成的错误 D. 运行时误操作

第七章 C++

考点一 类、标识符、数据类型和存储类型

类	定义	<u>类是具有相同属性和服务的一组对象的集合</u> ，它为属于该类的全部对象提供了 <u>抽象的描述</u> ，其内容包括数据项和行为两个主要部分（一个属于某类的对象称为该类的一个实例）。							
	基本特征	<u>封装</u>	封装是把对象的属性和服务结合成一个独立的系统单位，并尽可能隐藏对象的内部细节。 （1）把对象的全部属性和全部服务结合在一起，形成一个不可分割的独立单位。 （2）信息隐藏，即尽可能隐蔽对象的内部细节，对外形成一个边界（或者说一道屏障），只保留有限对外接口使之与外部发生联系。						
		<u>继承</u>	特殊类的对象拥有其一般类的全部属性与服务，称作特殊类对一般类的继承。						
		<u>多态</u>	多态性是指在一般类中定义的属性或行为，被特殊类继承之后，可以具 <u>有不同的数据类型或表现出不同的行为</u> 。						
标识符	标识符的 <u>命名规则</u> 如下： （1） <u>以大写字母、小写字母或下划线（-）开始；</u> （2） <u>可以由大写字母、小写字母、下划线（-）或数字 0~9 组成；</u> （3） <u>大小写敏感，即大写字母和小写字母代表不同的标识符；</u> （4） <u>不能是 C++关键字。</u>								
数据类型	<u>类型名</u>	bool	char	short int	int	long int	float	double	
	<u>长度（B）</u>	1	1	2	4	4	4	8	
	注意事项	float 和 double 除了取值范围不同，精度也不一样， <u>float 可以保存 7 位有效数字，double 可以保存 15 位有效数字</u> 。							
存储类型	<u>自动存储类型（auto）</u>	变量的存储类型决定其存储方式。 <u>自动存储类采用堆栈方式分配内存空间，属于暂时性存储</u> ，其存储空间可以被若干变量多次覆盖使用。 <u>函数中的局部变量，不加特殊声明，都是 auto 变量，但是关键字“auto”可以被省略。这些变量在函数被调用时分配存储方式，函数调用结束后这些存储空间就被释放。</u>							

<u>寄存器存储类型</u> <u>(register)</u>	当 <u>某些变量被频繁使用</u> ，就有可能将这样的变量保存在 CPU 的寄存器中，以加快其存取速度。
<u>静态存储类型</u> <u>(static)</u>	在内存中是以固定地址存放的， <u>在整个程序运行期间都有效</u> 。
<u>外部存储类型</u> <u>(extern)</u>	<u>全局变量（外部变量）是在函数的外部定义的，它的作用域为从变量的定义处开始，到本程序文件的末尾。</u> 在此作用域内，全局变量可以为本文件中各个函数所引用。还可以把在另一文件中定义的外部变量的作用域扩展到本文件，在本文件中可以合法地引用该外部变量。

❧ 冲刺模拟 ❧

1. 以下程序的运行结果是（ ）。

```
int main(){
    int k=4,m=1,p;
    p=func(k,m);
    cout<<p<<endl;
    p=func(k,m);
    cout<<p<<endl;
}
func(int a,int b)
{ static int m=0,i=2;
  i+=m+1;
  m=i+a+b;
  return(m);
}
```

A. 8,17 B. 8,16 C. 8,20 D. 8,8

2. 以下选项中，合法的用户标识符是（ ）。

A. long B. _2Test C. 3Dmax D. A. dat

考点二 运算符

基本运算符	类型	+（加或正号）、-（减或负号）、*（乘）、/（除）、%（取余）。			
	注意事项	<p>（1）<u>当“/”用于两个整型数据相除时，其结果为商的整数部分，小数部分自动舍弃，如 1/2=0。</u></p> <p>（2）<u>“%”只能用于整型操作数，表达式 a%b 的结果为 a 被 b 除的余数。</u>其优先级与“/”相同。</p>			
自增运算符（++）与自减运算符（--）	定义	自增运算符（++）和自减运算符（--）的定义是将操作数的值增 1（减 1）后重新写回该操作数在内存中原有的位置。			
	两种形式	每种运算符都有两种使用形式：前置和后置。 如：自增的前置用法为++i，后置用法为 i++；自减的前置用法为--i，后置用法为 i--。 前置与后置的区别： <u>前置是先自增（自减）后再使用，后置是先使用后自增（自减）。</u>			
	示例	当 i=5 时，下列两条语句的执行结果是不一样的。 <pre>cout<<i++;</pre> <pre>cout<<++i;</pre>			
逗号运算符	形式	表达式 1，表达式 2，……，表达式 n。 求解顺序为：先求解表达式 1，再求解表达式 2，……，最后求解表达式 n， <u>最终整个逗号运算符构成的表达式结果为：表达式 n 的值。</u>			
逻辑运算符	非 (!)	“!”是一元运算符，作用是 <u>对操作数取反</u> 。			
	与 (&&)	“&&”是二元运算符，作用是求两个操作数的逻辑“与”， <u>只有当两个操作数的值都为 true 时，“与”的运算结果才为 true。</u>			
	或 ()	“ ”是二元运算符，作用是求两个操作数的逻辑“或”， <u>只有当两个操作的值都为 false 时，“或”运算的结果才为 false。</u>			
	注意事项	<p>（1）优先级从上到下逐渐递减；</p> <p>（2）<u>默认非 0 值为真，0 值为假。</u></p>			
	运算规则	<u>&&</u>		<u> </u>	
	<u>0&&0=0</u>	<u>0&&1=0</u>	<u>0 0=0</u>	<u>0 1=1</u>	
	<u>1&&0=0</u>	<u>1&&1=1</u>	<u>1 0=1</u>	<u>1 1=1</u>	

条件运算符	表达式的格式	形式为： <u>表达式 1? 表达式 2: 表达式 3</u> 求解顺序为： (1) <u>先求解表达式 1;</u> (2) <u>若表达式 1 的值为 true，则求解表达式 2，表达式 2 的值为最终结果;</u> (3) <u>若表达式 1 的值为 false，则求解表达式 3，表达式 3 的值为最终结果。</u>		
	注意事项	<u>条件运算符优先级高于赋值运算符，低于逻辑运算符。</u>		
sizeof 运算符	语法格式	<u>sizeof (类型名或表达式)</u> <u>运算结果为“类型名”所指定的类型或“表达式”的结果类型所占的字节数。</u>		
	sizeof 与 strlen 的区别	<u>sizeof(), 计算的是所占空间的大小，包括'\0';</u> <u>strlen(), 计算的是字符串的长度，到'\0'截止，不包括'\0';</u>		
与地址相关的运算符 “*” 和 “&”	“*”运算符的两种定义	指针类型说明符	定义	<u>当“*”出现在声明语句中，放在被声明的变量名之前，表示当前声明的变量是指针变量。</u>
			示例	int *p; //声明 p 是一个整型指针变量
		指针运算符	定义	<u>当“*”出现在执行语句中或声明语句的初始化表达式中时，表示获取指针所指向的变量的值，即访问指针所指对象的内容。</u>
			示例	cout<<*p; //输出指针 p 所指向的内容
	“&”运算符的两种定义	引用运算符	定义	<u>当“&”出现在变量声明语句中，放在被声明变量的左边，表示声明的变量是引用变量。引用的定义是给已定义的变量起别名。通过引用名与通过被引用的变量名访问变量的效果是一样的。</u>

			注意事项	<p>(1) <u>声明一个引用时，必须同时对它进行初始化</u>，使它指向一个已存在的对象。</p> <p>(2) <u>一旦一个引用被初始化后，就不能改为指向其他对象。</u>也就是说，一个引用从它诞生之时起，就必须确定是哪个变量的别名，而且始终只能作为这个变量的别名。</p>
			示例	<pre>int i,j=10; int &a=i; a=j; //相当于 i=j //建立一个 int 型的引用 a，并将其初始化为变量 i 的一个别名，a 变量引用了 i 变量，可知 a 和 i 的值相等，均共用同一块内存地址。</pre>
		取地址运算符	定义	<u>在给变量赋初值时，“&”出现在等号右边或在执行语句中出现时，表示用来获得一个对象的地址。</u>
			示例	例：&i 表示获得变量 i 的存储单元地址。

~ 冲刺模拟 ~

1. while(!x)中的(!x)与下面 () 条件等价。

- A. x==1 B. x!=1 C. x!=0 D. x==0

2. 阅读下列程序，正确的选项为 ()。

```
#include<iostream>
using namespace std;
int main(){
int x=4;
do{
cout<<(x-=3)<<endl;
}while(--x);
}
```

- A. 1 B. 1 和 -3 C. 2 和 0 D. 死循环

考点三 函数

内联函数	定义原则	把一些 <u>功能简单、规模较小并且使用频繁的函数</u> ，设计为内联函数。 <u>复杂的函数，一般不设置成内联函数</u> ，规模复杂的函数如果定义为内联函数会造成代码膨胀，增大开销。比如：递归函数。
	执行过程	内联函数在调用时不会发生控制转移，而是 <u>在编译时将内联函数的函数体嵌入在每一个调用处</u> 。
	特点	提高开发效率，增强程序的可靠性，节省参数传递和控制转移的开销。
	语法形式	<u>内联函数的定义使用关键字 inline</u> ，形式如下： <pre>inline 类型说明符 函数名（含类型说明的形参表） { 语句序列 }</pre>
带默认形参值的函数	调用原则	<u>函数在定义时可以预先声明默认的形参值。调用时如果给出实参，则用实参初始化形参，如果没有给出实参，则采用预先声明的默认形参值。</u>
	示例	<pre>int add(int x=5, int y=6){ //声明默认形参值 return x+y;} int main(){ add(10, 20); add(10); add();}</pre>
	注意事项	<p>（1）<u>有默认值的形参必须在形参列表的最后，即在有默认值的形参右面，不能出现无默认值的形参。</u></p> <p>例：</p> <pre>int add(int x, int y=5, int z=6); //正确 int add(int x=1, int y=5, int z); //错误</pre> <p>（2）<u>对于同一个函数的多个声明来说，禁止对同一个参数的默认值进行重复定义，即使前后定义的值相同也不行。</u></p> <p>例：</p> <pre>int add(int x=5, int y=6); //默认形参值在函数原型声明中给出 int main(){ add(); return 0; }</pre> <pre>int add(int x =5, int y =6){ //错误，这里不能再出现默认形参值的定义 return x+y;</pre>

		}
重载函数	定义	两个以上的函数， <u>具有相同的函数名，但是形参的个数或者类型不同</u> ，编译器根据实参和形参类型及个数的最佳匹配，自动确定调用哪一个函数。
	示例	<p>(1) <u>形参类型不同</u></p> <pre>int add(int x, int y); int add(float x, float y);</pre> <p>(2) <u>形参个数不同</u></p> <pre>int add(int x, int y); int add(int x, int y, int z);</pre>
构造函数	对象的初始化	对象初始化是指在定义对象的时候进行的数据成员设置。
	作用	在对象被创建时，利用特定的值构造对象，将对象初始化为一个特定的状态。
	特点	<p>(1) <u>构造函数是类的一个成员函数；</u></p> <p>(2) <u>构造函数的函数名与类名相同，而且没有返回值；</u></p> <p>(3) <u>构造函数通常被声明为公有函数；</u></p> <p>(4) <u>构造函数在对象被创建的时候将被自动调用。</u></p>
	默认构造函数	<u>如果类中没有写构造函数，编译器会自动生成一个隐含的默认构造函数，该构造函数的参数列表和函数体皆为空。若类中已声明构造函数，编译器则不再自动生成默认构造函数。</u>
析构函数	作用	<p>(1) 析构函数与构造函数的作用几乎正好相反，它<u>用来完成对象被删除前的一些清理工作；</u></p> <p>(2) 析构函数是<u>在对象的生存期即将结束的时刻被自动调用的</u>。析构函数调用完成之后，<u>对象会消失，相应的内存空间会被释放。</u></p>
	特点	<p>(1) 析构函数也是类的一个公有函数成员，它的<u>名称是由类名前面加“~”</u>构成，没有返回值。与构造函数不同的是<u>析构函数不接收任何参数；</u></p> <p>(2) <u>若没有对析构函数做显式说明，系统会自动生成一个函数体为空的隐含析构函数。</u></p>

❧ 冲刺模拟 ❧

- 1.在 C++语言中函数返回值的类型是由（ ）决定的。
 - A.调用该函数时系统临时
 - B. return 语句中的表达式类型
 - C.定义该函数时所指定的函数类型

D.调用该函数时的主调函数类型

2.在 C++ 语言中，对函数参数默认值描述正确的是（ ）。

- A.函数参数的默认值只能设定一个
- B.一个函数的参数若有多个，则参数默认值的设定可以不连续
- C.函数必须设定默认值
- D.在设定参数的默认值后，该参数后面定义的所有参数都必须设定默认值

考点四 指针数组、符号常量、数组做函数参数

指针数组	定义	<u>数组中的每个元素保存的都是指针</u> ，那么就称它为指针数组。指针数组的每个元素都必须是同一类型的指针。	
	语法形式	<u>声明一维指针数组的语法形式为：</u> <u>数据类型 *数组名[下标表达式]</u>	
	示例	int *pa[3]; 说明：声明一个整型的指针数组 pa，它有 3 个数组元素，每个元素都是一个指向整型数据的指针。	
符号常量	定义	<u>常量（const）是指在程序运行的整个过程中其值始终不可改变的量</u> 。符号常量就是为常量命名，符号常量在使用之前 <u>一定要先声明，且在声明时赋初值</u> 。	
	声明形式	<u>const 数据类型说明符 常量名=常量值；</u> <u>或</u> <u>数据类型说明符 const 常量名=常量值；</u>	
	示例	const float PI=3.1415926; 说明：声明一个代表圆周率的符号常量，并赋初值。	
	<u>常量指针</u>	定义	<u>即指向常量的指针。</u>
		注意事项	<u>不能通过指针来改变所指对象的值，但是指针本身可以改变，可以指向另外的对象，即该指针所指向的地址可发生改变。</u>
		示例	const char *p= "John"; //p 是指向常量的指针 char s[]="abc"; p=s; //正确，p 本身的值可以改变 *p= 'l'; //编译时出错，不能通过 p 改变所指的对象
	<u>指针常量</u>	定义	<u>即指针本身是一个常量。</u>

数组作为函数参数		注意 事项	<u>指针本身不能被改变，也就是说指针所指向的那个对象的地址不能改变。</u>
		示例	int a,c; int * const b=&a; //b 是指针常量 b=&c; //错误，指向的地址不可以重新赋值
	<u>值传递</u>	定义	用数据元素作为调用函数时的实参，这与普通变量做实参含义相同。在发生函数调用时，把作为实参的数组元素的值传送给形参，实现单向的值传递。
		注意 事项	<u>如果在被调函数中对形参值进行改变，主调函数中实参的值不会改变。</u>
	<u>地址传递</u>	定义	<u>使用数组名做函数的参数，则实参和形参都应该是数组名，且类型要相同，此外，数组名作为实参，传递的是地址。</u>
		传递 过程	把实参数组的首地址传给形参数组，形参数组和实参数组的首地址重合，即实参数组名和形参数组名共同指向数组的第一个元素，对应元素使用相同的数据存储地址。
		注意 事项	<u>如果在被调函数中对形参数组元素值进行改变，主调函数中实参数组的相应元素值也会改变。</u>

~ 冲刺模拟 ~

1.设有如下定义：

```
char *aa[2]={ "abcd","ABCD"};
```

则以下说法中正确的是（ ）。

- A. aa 数组中的元素值分别为“abcd”和“ABCD”
- B. aa 是指针变量，它指向含有两个数组元素的字符型一维数组
- C. aa 数组的两个元素分别存放的是含有 4 个字符的一维字符数组的首地址
- D. aa 数组的两个元素中各自存放了字符‘a’和‘A’的地址

2.已知有定义

```
const int d=5;
```

```
int i=1;
```

```
double f=0.32;
```

```
char c="c";
```

下面哪个命令是错误的？（ ）

- A. ++i;
- B. d--;
- C. c++;
- D. --f;