# DMA（Direct Memory Access）

直接内存存取，是所有现代电脑的特色，它允许不同速度的硬件装置来沟通，而不需要依赖于CPU的大量终端负载。否则，CPU需要从来源把每一片段的资料复制到暂存器，然后把他们再次写回到新的地方。在这个时间中，CPU对于其他的工作来说就无法使用。

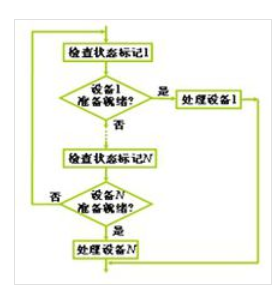
# 程序查询方式

程序查询方式是一种程序直接控制方式，这是主机与外设间进行信息交换的最简单的方式，输入和输出完全是通过CPU执行程序来完成的。

一旦某一外设被选中并启动后，主机将查询这个外设额某些状态，看其是否准备就绪？若外设未准备就绪，主机将再次查询；若外设已准备就绪，则执行一次I/O操作。

这种控制方式很简单，但外设和主机不能同时工作，各外设之间也不能同时工作，系统效率很低，因此，仅适用于外设的数目不多，对I/O处理的实时要求不怎么高，CPU的操作任务比较单一，并不很忙的情况。

多个设备工作流程如下：



# 程序中断方式

程序中断是指计算机执行现行程序的过程中，出现某些急需处理的异常情况和特殊请求，CPU暂时终止现行程序，而转去对随机发生的更紧迫的事件进行处理，在处理完毕后，CPU将自动返回原来的程序继续执行。

当主机启动外设后，无需等待查询，而是继续执行原来的程序，外设在做好输入输出准备时，向主机放出中断请求，主机借到请求后就暂时终止原来执行的程序，转去执行中断服务程序对外部请求进行处理，在中断处理完毕后返回原来的程序继续执行。显然，程序中断不仅适用于外部设备的输入输出操作，也适用于对外界发生的随机事件的处理。

程序中断在信息交换方式中处于最重要的地位，它不仅允许主机和外设同时并行工作，并且允许一台主机管理多台外设，使他们同时工作。但是完成一次程序中断还需要许多辅助操作，当外设数目较多时，中断请求过分频繁可能使CPU应接不暇；另外，对于一些高速外设，由于信息交换是成批的，如果处理不及时，可能会造成信息丢失，因此，它主要适用于中、低速外设。

程序中断与调用子程序的区别：

子程序的执行是由程序员实现安排好的，而中断服务程序的执行则是由随机中断事件引起的；

子程序的执行受到主程序或上层子程序的控制，而中断服务程序一般与被中断的现行程序毫无关系；

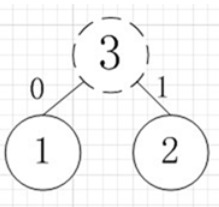
不存在同时调用多个子程序的情况，而有可能发生多个外设同时请求CPU为自己服务的情况。

# 哈夫曼编码（霍夫曼编码）Huffman Coding

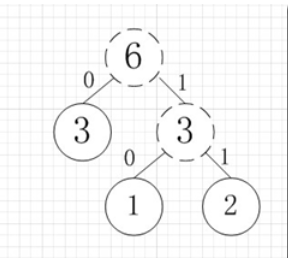
哈夫曼编码（Huffman coding），又称霍夫曼编码，是一种编码方式，可变长编码（VLC）的一种。Huffman于1952年提出的一种编码方法，该方法完全依据字符出现概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做Huffman编码）。

哈夫曼编码，主要目的是根据使用频率来最大化节省字符（编码）的存储空间。

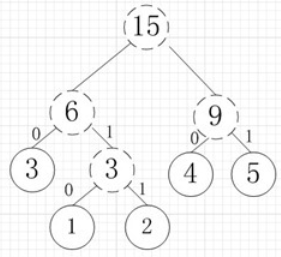
简易的理解就是，假如我有ABCDE五个字符，出现的频率（即权值）分别为：5，4，3，2，1，那么我们第一步先取两个最小权值作为左右子树构造一个新树，即取1，2构成新树，其节点为1+2=3，如图：



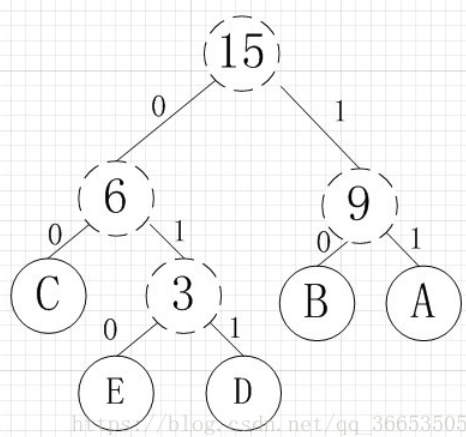
虚线为新生节点，第二步再把新生成的权值为3的节点放到剩下的集合中，所以集合变成{5，4，3，3}，在根据第一步，取最小的两个权值构成新树，如图：



再依次建立哈夫曼树，如下图：



其中各个权值替换对应的字符即为下图：



所以各字符对应的编码为：A->11,B->10,C->00,D->011,E->010

霍夫曼编码是一种无前缀编码。解码时不会混淆。其主要应用在数据压缩，加密解密等场合。

如果考虑到进一步节省存储空间，就应该将出现概率大（占比多）的字符用尽量少的0-1进行编码，也就是更靠近根（节点少），这也就是最优二叉树-哈夫曼树

# 邮件服务

25端口（SMTP）：25端口为SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）服务所开放的，是用于发送邮件。如今绝大多数邮件服务器都使用该协议。当你给别人发送邮件时，你的机器的某个动态端口（大于1024）就会与邮件服务器的25号端口建立一个连接，你发送的邮件就会通过这个连接传送邮件到邮件服务器上，保存起来。

109端口（POP2）：109端口是为POP2（Post Office Protocol Version 2，邮局协议2）服务开放的，是用于接收邮件的。

110端口（POP3）：110端口是为POP3（Post Office Protocol Version 3，邮局协议3）服务开放的，是用于接收邮件的。

143端口（IMAP）：143端口是为IMAP（Internet Message Access Protocol）服务开放的，是用于接收邮件的。

目前POP3使用的比POP2广的多，POP2几乎被淘汰，也有某些服务器同时支持POP2和POP3协议。客户端可以使用POP3协议来访问服务端的邮件服务，如今ISP的绝大多数邮件服务器都是使用POP3协议（极少用POP2协议）。在使用邮件客户端程序的时候，会要求输入POP3服务器地址，默认情况下使用的就是110端口。当你使用邮件客户端（比如:Thunderbird,foxmail，MS Outlook Express以及各类 邮件精灵）登录时，你的机器就会自动用某一个动态端口（大于1024）连接邮件服务器的110端口，服务器就把别人发给你的邮件（之前保存在邮件服务器上），发送到你的机器，这样牛就可以看到新邮件了。

IMAP协议，和POP3协议一样是用来接收邮件的，但是他又他的特别和新颖之处，它是面向用户的，它和POP3协议的主要区别是：用户可以不把所有的邮件内容全部下载，而是只下载邮件标题和发件人等基本信息，用户可以由标题等基本信息，去决定是否下载邮件全文，用户可以通过客户端的浏览器直接对服务器上的邮件进行操作（比如：打开阅读全文、丢经垃圾箱 、永久删除、整理到某文件夹下、归档）。再简单来说就是：浏览器用的IMAP协议（143端口）来为你接收邮件以及让你很方便的操作服务器上的邮件。邮件客户端用的POP3协议（110端口）来为你接收邮件的全部信息和全文内容保存到你本地机器成为一个副本，你对邮件客户端上的副本邮件的任意操作都是在副本上，不干涉邮件服务器上为你保存的邮件原本。

SMTP，POP2，POP3，IMAP协议都是不安全的协议。因考虑到网络安全的因素，下面介绍基于SSL（Secure Sockets Layer安全套接层）协议的安全的邮件收发协议。你的邮件在传输过程中可能被网络黑客截取邮件内容，如果你的邮件机密性非常强，不想被收件人以外的任何人和任何黑客截取，或者是涉及国家机密安全的，等待。那么你的邮件就不该用上述的协议进行收发。

465端口（SMTPS）：465端口是为SMTPS（SMTP-over-SSL）协议服务开放的，这是SMTP协议基于SSL安全协议之上的一种变种协议，他继承了SSL非对称加密的高度安全可靠性，可防止邮件泄露。SMTPS的SMTP协议一样，也是用来发送邮件的，只是更安全些，防止邮件被黑客截取泄露，还可实现邮件发送者抗抵赖功能。防止发送者发送之后删除已发邮件，拒不承认发送过这样一份邮件。

995端口（POP3S）：995端口是为POP3S（POP3-over-SSL）协议服务开放的，这是POP3协议基于SSL安全协议之上的一种变种协议，它继承了SSL安全协议的非对称加密的高度安全可靠性，可防止邮件泄露。POP3S的POP3协议一样，也是用来发送邮件的，只是更安全些，防止邮件被黑客截取泄露，还可实现邮件发送者抗抵赖功能。防止发送者发送之后删除已发邮件，拒不承认发送过这样一份邮件。

993端口（IMAPS）：993端口是为IMAPS（IMAP-over-SSL）协议服务开放的，这是IMAP协议基于SSL安全协议之上的一种变种协议，它继承了SSL安全协议的非对称加密的高度安全可靠性，可防止邮件泄露。IMAPS的IMAP协议一样，也是用来发送邮件的，只是更安全些，防止邮件被黑客截取泄露，还可实现邮件发送者抗抵赖功能。防止发送者发送之后删除已发邮件，拒不承认发送过这样一份邮件。

# TCP和 UDP的优缺点

Tcp的优点：可靠、稳定，TCP的可靠体现在TCP传递数据之前会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。

TCP的缺点：慢、效率低、占用系统资源高，易被攻击。TCP在传输数据之前，要先建立连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的CPU、内存等硬件资源。而且，因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击。

UDP优点：快，比TCP稍安全，UDP没有TCP的握手、确认、窗口、重传、拥塞控制等机制，UDP是一个无状态的传输协议，所以他在传递数据时非常快。没有TCP的这些机制，UDP较TCP被攻击者利用的漏洞就要少一些。但UDP也是无法避免攻击的，比如：UDP Flood攻击…

UDP的缺点：不可靠，不稳定，因为UDP没有TCP那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。

# 网络攻击

网络攻击（Cyberattack，也译为赛博攻击）是指针对计算机信息系统、基础设施、计算机网络或个人计算机设备的，任何类型的进攻动作。

于计算机和计算机网络中，破坏、揭露、修改、使软件或服务失去功能、在没有得到授权的情况下偷取或访问任何一台计算机的数据，都会被视为于计算机和计算机网络中的攻击。

主动攻击：主动攻击会导致某些数据流的篡改和虚假数据流的产生。这类攻击可分为篡改、伪造消息数据和中断（拒绝服务）。

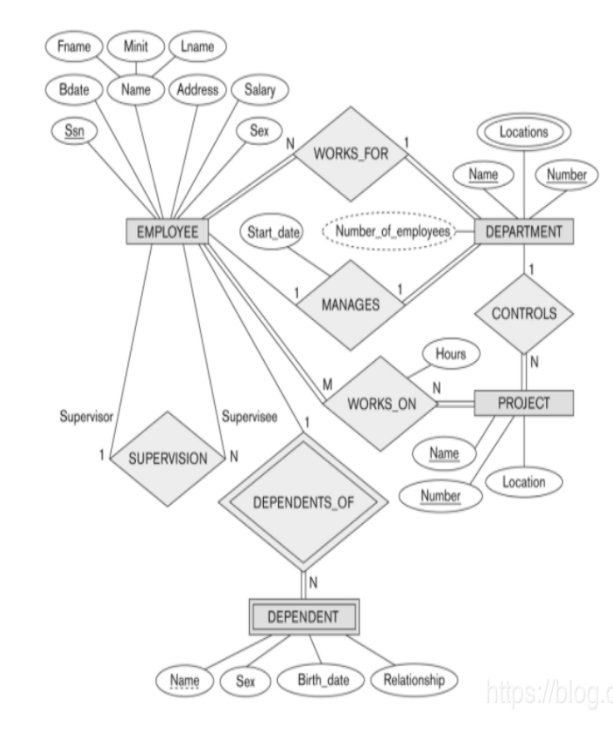
被动攻击：被动攻击中攻击者不对数据做任何修改，窃取/窃听是指在未经用户同意和认可的情况下攻击者获得了信息或相关数据。通常包括窃听、流量分析、破解弱加密的数据流等攻击方式。

# DFD（数据流图）

数据流图（Data Flow Diagram）；简称DFD，他从数据传递和加工角度，以图形方式来表达系统的逻辑功能、数据在系统内部的逻辑流向和逻辑变化过程，是结构化系统分析方法的主要表达工具及用于软件模型的一种图示方法。

# 实体关系图（ERD）

我们来看一个公司数据库的例子



实体关系模型，也称为实体关系（ER）图，是实体（将称为您的表）及相互关系的图形表示。数据库建模是创建数据模型的过程。

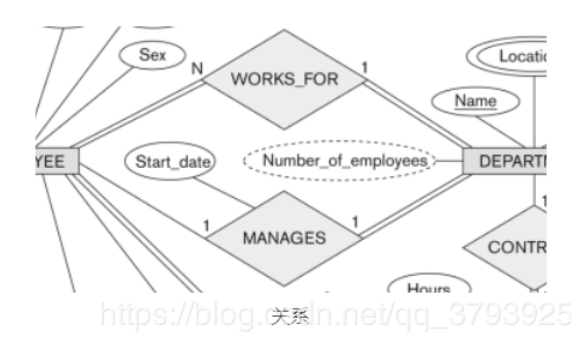
## 实体

一个简单的矩形代表一张表。



## 关系

它使用钻石形状勾勒出来



您可以绘制关系的类型，无论是使用1-M的一对多，还是使用M-N的多对多，或使用1-1的一对一。

连接具有关系的实体的线是单线还是双线引用另一个约束，称为存在依赖性约束（也称为“参与约束”）

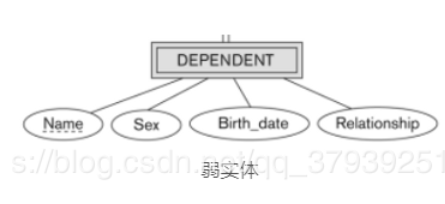
因此，如果一名员工必须为一个部门工作，这由双线勾画，并称为“全部或强制参与”。如果一名员工可能会或可能不会管理某个部门，则可以通过单行描述并称之为“部分或可选参与”。

## 递归关系

这是与同一实体的关系。例如，员工可以担任一个或多个员工的主管，而员工可以由员工监督（这是一对多的关系）。

## 弱实体

弱实体只是一个实体，他的存在依赖另一个实体。在缺少employee表的情况下，您无法在逻辑上拥有依赖（儿子、女儿…）他的轮廓与普通实体相同，但有双线。



# 需求分析阶段的任务

对现实世界要处理的对象（组织、部门、企业等）进行详细调查，在了解现行系统的概况、确定新系统功能的过程中，确定系统边界、收集支持系统目标的基础数据及处理方法。

# 逻辑设计阶段的任务

逻辑设计阶段的任务之一是对关系模式进一步的规范化处理。因为生成的初始关系模式并不能完全符合要求，会有数据冗余、更新异常存在，这就需要根据规范化理论对关系模式进行分解，以消除冗余和更新异常。不过有时根据处理要求，可能还需要增加部分冗余以满足处理要求。逻辑设计阶段的任务就需要做部分关系模式的处理，分解、合并或增加冗余属性，提高存储效率和处理效率。

# 主属性定义

“包含在任何一个候选码中的属性叫做主属性（Prime Attribute），否则叫做非主属性（Nonprime attribute）。

# BIOS

Bios（基本输入输出系统）是一个程序，是固化在主板上的ROM芯片里面的，它相当于硬件底层的一个操作系统，控制和保存着硬件的输入输出工作。

# 奇偶校验

奇校验：如果以二进制数据中的1的个数是奇数为依据，则是奇校验

偶校验：如果以二进制数据中的1的个数是偶数为依据，则是偶校验

如果传输过程中包括校验位在内的奇数个数据位发生改变，那么奇校验位将出错表示传输过程有错误发生，但是由于没有办法确定哪一位出错，所以不能进行错误校正。

# 浮点数对阶

两个浮点数对阶的时候要把阶码小的数的尾数右移n位，与阶码大的对齐。

# 上下文无关文法

形式语言理论中一种重要的变换文法，用来描述上下文无关语言，在乔姆斯基分层中称为2型文法。由于程序设计语言的语法基本上都是上下文无关文法，因此应用十分广泛。

# 超长指令字

VLIW（very long instruction word）超长指令字，一种非常长的指令组合，他把许多指令连在一起，增加了运算的速度。

# 解释方式和编译方式

解释程序也称为解释器，它或者直接解释执行源程序，或者将源程序翻译成某种中间表示形式后再加以执行。

编译程序（编译器）则是将源程序翻译成目标语言程序，然后在计算机上运行目标程序。

两种语言处理程序的根本区别是：在编译方式下，机器上运行的是与源程序等价的目标程序，源程序和编译程序都不再参与目标程序的执行过程，而在解释方式下，解释程序和源程序（或某种等价表示）要参与到程序的运行过程中，运行程序的控制权在解释程序。

解释器翻译源程序时不生成独立的目标程序，而编译器则将源程序翻译成独立的目标程序。

# 词法分析

词法分析是编译或解释用户源程序过程中唯一与源程序打交道的阶段，其主要功能是按顺序分析出源程序的记号。

# 主要的程序语言

Fortran语言：第一个高级程序设计语言，科学计算，执行效率高

Pascal语言：结构化程序设计语言，表达能力强，Delphi

C语言：通用、结构化程序设计语言，指针操作能力强，高效

Lisp语言：函数式程序语言，符号处理，人工智能

C++语言：C语言基础上增加了类机制，面向对象，高效

Java语言：面向对象，中间代码，跨平台，通用的程序设计语言

Python语言：面向对象，解释型程序设计语言，通用的脚本语言

PHP：服务端脚本语言，制作动态页面

Ruby：简单快捷，面向对象，脚本语言

Delphi：快速应用程序开发工具，可视化编程环境

Cobol：数据处理领域最为广泛的程序设计语言，高级编程语言

Prolog：逻辑式语言，建造专家系统、自然语言理解、智能知识库等

# 常用的中间代码

常用的中间代码的表达形式有语法树、后缀式、三地址代码。

# 指令集

1. 指令系统：RISC设计者把主要精力放在那些经常使用的指令上，尽量使他们具有简单高效的特色。对不经常用的功能，常通过组合指令来完成。而CISC计算机的指令系统比较丰富，有专用的指令来完成特定的功能。因此，处理特殊任务效率较高。
2. 存储器操作：RISC对存储器操作有限制，使控制简单化；而CISC机器的存储器操作指令多，操作直接。
3. 程序：RISC汇编语言程序一般需要较大的内存空间，实现特殊功能时程序复杂，不易设计；而CISC汇编语言程序编程相对简单，科学计算及复杂操作的程序设计相对容易，效率较高。
4. 设计周期：RISC微处理器结构简单，布局紧凑，设计周期短，且易于采用最新技术；CISC微处理器结构复杂设计周期长。
5. 应用范围：由于RISC指令系统的确定与特定应用领域有关，故RISC机器更适合于专用机；而CISC机器则更适合于通用机。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令系统类型 | 指令 | 寻址方式 | 实现方式 | 其他 |
| CISC | 数量多，使用频率差别大，可变长格式 | 支持多种 | 微程序控制技术 |  |
| RISC | 数量少 ，使用频率接近，定长格式，大部分为单周期指令，操作寄存器，只有load/store操作内存 | 支持方式少 | 增加了通用寄存器，硬布线逻辑控制为主，采用流水线 | 优化编译，有效支持高级语言 |

# 表示媒体

表示媒体指的是为了传输感觉媒体而认为研究出来的媒体，借助于此种媒体，能有效地存储感觉媒体或将感觉媒体从一个地方传送到另一个地方。如：语言编码、电报码、条形码等。

# 表现媒体

是指进行信息输入和输出的媒体，如：键盘、鼠标、扫描仪、话筒和摄像机等输入媒体以及显示器、打印机和扬声器等输出媒体。

# 感觉媒体

是指直接作用于人的感觉器官，使人产生直接感觉的媒体，如：引起听觉反应得声音，引起视觉反应得文本、图形和图像等。

# 存储媒体

存储媒体是指用于存储表示媒体得物理介质，如：硬盘、软盘、光盘和胶卷等。

# 传输媒体

是指用于传输表示媒体得物理介质，如：电缆和光缆等。

# 光盘存储技术

DVD-RAM和DVD-RW是DVD技术所支持的两种不同的可多次插除重写的DVD光盘 格式，CD-R指一次性可写（刻录）CD光盘，而CD-RW指多次可擦除、重写的CD光盘。

# 程序语义

分为静态语义和动态语义，其中静态语义分析方法是语法制导翻译，其基本思想是将语言结构的语义以属性的形式赋予代表此结构的文法符号，而属性的计算以语义规则的形式赋予文法的产生式。

# 色彩

亮度是指发光体（反光体）表面发光（反光）强弱的物理量。

色调指的是一幅画中画面色彩的总体倾向，是大的色彩效果。在大自然中，我们经常见到这样一种现象：不同颜色的物体或被笼罩在一片金色的阳光之中，或被笼罩在一片轻纱薄雾似的、淡蓝色的月色之中；或被秋天迷人的金黄色所笼罩；或被统一在冬季银白色的世界之中。这种在不同颜色的物体上，笼罩者某一种色彩，使不同颜色的物体都带有同一种色彩的倾向，这样的色彩现象就是色调。

饱和度是指色彩的鲜艳程度，也称色彩的纯度。

# 海明码

海明码是利用奇偶性来检错和纠错的校验方法，码距最小为2n+1。

# 主存与Cache地址映射

全相联地址映射：主存的任意一块可以映射到cache中的任意一块。

直接相联映射：主存中的一块只能映像到cache的一个特定块中。

组相联映射：各区中的某一块只能存入缓存的同组号的空间内，但组内各块地址之间则可以任意存放。即从主存的组到cache的组之间采用直接映射，在两个对应的组内部采用全相联映射。

# 累加寄存器AC

通用寄存器，为ALU（算术逻辑单元）提供一个工作区，用来暂存数据。

# CPU

指令寄存器（IR）用来存放从内存中读取的指令；累加器（累加寄存器AC），通用寄存器，可用于传送和暂存数据，也可参与算术逻辑运算，并保存运算结果；程序计数器（PC）是用于存放下一条指令所在单元的地址，当执行一条指令时，首先需要根据PC中存放的指令地址，将指令由内存取到指令寄存器中，即 将程序计数器PC中的内容送到地址总线上，此过程称为“取指令”，与此同时，PC中的地址或自动加1或由转移指针给出下一条指令的地址，此后经过分析指令，执行指令，完成第一条指令的执行，而后根据PC取出第二条指令的地址，如此循环，执行每一条指令；状态寄存器（PSW）用于存放状态标志与控制标志如：中断标志，溢出标志等。

# 编译过程

一般分为5个阶段：词法分析、语法分析、语义分析、优化、目标代码生成。

1. 词法分析：也就是从左到右一个一个读入源程序，识别一个单词或符号，并进行归类
2. 语法分析：在词法分析的基础上，将单词序列分解成各类语法短语，如：程序、语句、表达式等
3. 语义分析：审查源程序是否由语义错误，当不符合语言规范的时候，程序就会报错
4. 代码优化：这个阶段是对前阶段的中间代码进行变换或改造，目的是使生成的目标代码更为高效，即节省时间和空间
5. 目标代码生成：也就是把优化后 的中间代码变化成指令代码或汇编代码

# 死循环

死循环错误属于典型的语义错误，但静态的语义错误可被编译器发现，到程序真正陷入死循环说明编译器并未发现，所以属于动态语义错误。

# 各种码制下带符号的数的表示范围

|  |  |  |
| --- | --- | --- |
| 码制 | 定点整数 | 定点小数 |
| 原码 | -((2^(n-1))-1)~(2^(n-1))-1 | -(1-2^(-(n-1)))~(1-2^(-(n-1))) |
| 反码 | -((2^(n-1))-1)~(2^(n-1))-1 | -(1-2^(-(n-1)))~(1-2^(-(n-1))) |
| 补码 | -2^(n-1)~(2^(n-1))+1 | -1~(1+2^(-(n-1))) |
| 移码 | -2^(n-1)~(2^(n-1))-1 | -1~(1+2^(-(n-1))) |

# 内存分配

一个由c/c++编译的程序占用内存分为以下几个部分：

1. 栈区（stack）：由编译器自动分配释放，存放函数的参数值 ，局部变量的值等。其操作方式类似于数据结构中的栈。
2. 堆区（heap）：一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。注意它与数据结构中的堆是两回事，链表的数据空间必须采用堆存储分配策略。
3. 全局区（静态区）（static）：全局变量和静态变量的存储是放在这一块的。初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域，程序结束后由系统释放。
4. 文字常量区：常量字符串就是放在这里的，程序结束后由系统释放。
5. 程序代码区：存放函数体的二进制代码。

# 存储器

SRAM静态的随机存储器：特点是工作速度快，只要电源不撤除，写入SRAM的信息就不会消失，不需要刷新电路，同时在读出时不破坏原来存放的信息，一经写入可多次读出，但集成度较低，功耗较大。SRAM一般用来作为计算机中的高速缓冲存储器（Cache）。

DRAM只能将数据保持很短的时间。为了保持数据，DRAM使用电容存储，所以必须隔一段时间刷新（refresh）一次，如果存储单元没有被刷新，存储的信息就会丢失。关机就会丢失数据。

# 遗传算法

遗传算法（Genetic Algorithm）是模拟达尔文生物进化论的自然选择仿遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群（population）开始的，而一个种群则由经过基因（gene）编码的一定数目的个体组成。每个个体实际上是染色体带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，他决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。

遗传算法的基本运算过程如下：

1. 初始化
2. 个体评价
3. 选择运算
4. 交叉运算
5. 变异运算

# 队列和栈的应用

栈：表达式求值、括号匹配、递归

队列：打印队列

# 链表

单向链表仅设头指针时，在表尾插入节点时需要遍历整个链表，时间复杂度为O（n），仅设表尾指针时，在表尾插入节点的时间复杂度为O（1），但是不能访问除了尾节点之外的所有其他节点。

单向循环链表仅设头指针时，在表尾插入节点时需要遍历整个链表，时间复杂度为O（n），仅设表尾指针时，在表尾插入节点的时间复杂度为O（1），同时达到表头节点的时间复杂度为O（1）。

# 二叉树

二叉树采用二叉链表存储时空指针个数为节点数+1，采用 三叉链表存储时空指针个数为节点数+2。

# 定长编码

例如有6个字母：a,b,c,d,e,f ，因为2^3=8，所以需要3位编码。

# 计数排序

计数排序是一个非基于比较的排序算法。他的优势在于对一定范围内的整数排序时，他的复杂度为O（n+k）（其中k是整数的范围）。快于任何比较排序算法。

# 耦合

无直接耦合：指两个模块之间没有直接的关系，他们分别从属于不同的模块的控制与调用，他们之间不传递任何信息。因此，模块之间耦合性最弱，模块独立性最高。

数据耦合：指两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言中的值传递。

标记耦合：指两个模块之间传递的是数据结构。

控制耦合：指一个模块调用另一个模块时，传递的是控制变量，被调用模块通过该控制变量的值有选择地执行模块内的某一功能。因此，被调用模块应具有多个功能，哪个功能起作用受调用模块控制。

外部耦合：模块间通过软件之外的环境连结（如I/O将模块耦合到特定的设备、格式、通信协议上）时称为外部耦合。

公共耦合：指通过一个公共数据环境相互作用的那些模块间的耦合。

内容耦合：当一个模块直接使用另一个模块的内部数据，或通过非正常入口转入另一个模块的内部时，这种模块之间的耦合称为内容耦合。

# 多态

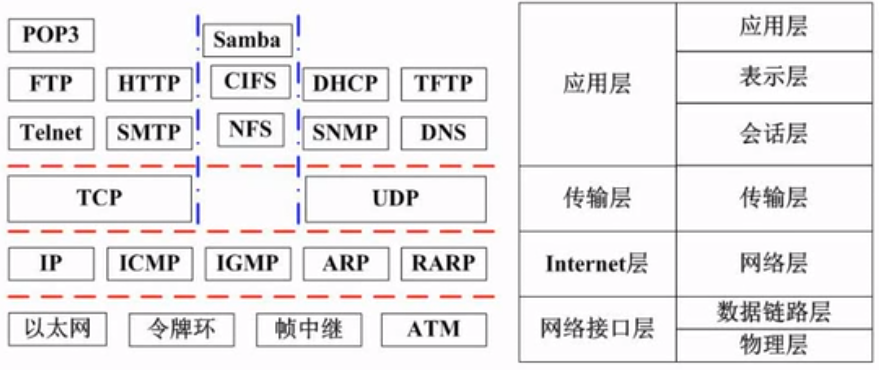
参数多态：应用最广泛、最纯的多态。

包含多态：同样的操作可用于一个类型及其子类型。包含多态一般需要进行运行时的类型检查。

过载（overload）多态：同一个名 （操作符、函数名）在不同的上下文中有不同的类型。

强制多态：编译程序通过语义操作，把操作对象的类型强制加以变换，以符合函数或操作符的要求。

# 网络



# 结构化分析工具

数据流图（DFD）

数据字典（Data Dictionary，DD）

结构化语言

判定表

判定树

# ISO/IEC软件质量模型

由三个层次组成：

第一层是质量特性；

第二层是质量子特性；

第三层是度量指标。

功能性是与一组功能及其指定的性质的存在有关的一组属性，其子特性包括适应性、准确定、互用性、依从性和安全性。

软件质量是软件特性的总和，是软件产品满足规定或潜在用户需求的能力。2001年，软件产品质量国际标准ISO/IEC9126定义的软件质量包括“内部质量”、“外部质量”和“使用质量”3部分。也就是说，“软件满足规定或潜在用户需求的能力”要从软件在内部、外部和使用中的表现来衡量。

|  |  |
| --- | --- |
| 质量特性 | 质量子特性 |
| 功能性 | 适合性 |
| 准确性 |
| 互用性 |
| 依从性 |
| 安全性 |
| 可靠性 | 成熟性 |
| 容错性 |
| 易恢复性 |
| 易使用性 | 易理解性 |
| 易学性 |
| 易操作性 |
| 效率 | 时间特性 |
| 资源特性 |
| 可维护性 | 易分析性 |
| 易改变性 |
| 稳定性 |
| 易测试性 |
| 可移植性 | 适应性 |
| 易安装性 |
| 一致性 |
| 易替换性 |

# 开发方法

原型法适合需求不清晰且多变的情况；

结构化开发方法适用于系统规模不太大且不太复杂，需求变化也不大的情况；

Jackson是一种面向数据结构的方法，以数据结构为驱动，适合于小规模的项目。

# 软件测试

语句覆盖要求被测试程序中的每一条语句至少执行一次，这种覆盖对程序逻辑执行逻辑的覆盖很低。

条件覆盖要求每一判定语句中每个逻辑条件的各种可能值至少满足一次。

判定/条件覆盖要求判定中每个条件的所有可能取值（真/假）至少出现一次，并使每个判定本身的判定结果（真/假）也至少出现一次。

路径覆盖则要求覆盖被测试程序中所有可能的路径。

# 状态图和活动图的区别

活动图主要描述行为的动作

状态图主要描述行为的结果

# 设计模式

## 创建型模式

|  |  |
| --- | --- |
| 设计模式名称 | 简要说明 |
| Abstract Factory抽象工厂模式 | 提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定他们具体的类 |
| Builder构建器模式 | 将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示 |
| Factory Method工厂方法模式 | 定义一个创建对象的接口，但由子类决定需要实例化哪一个类。工厂方法使得子类实例化的过程推迟 |
| Prototype原型模式 | 用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象 |
| Singleton单例模式 | 保证一个类只有一个实例，并提供一个访问他的全局访问点 |

## 结构性模式

|  |  |  |
| --- | --- | --- |
| 设计模式名称 | 简要说明 | 速记关键字 |
| Adapter适配器模式 | 将一个类的接口转换成用户希望得到的另一种接口。它使版本不相容的接口得以协同工作 | 转换接口 |
| Bridge桥接模式 | 将类的抽象部分和实现部分分离开来，使他们可以独立地变化 | 继承树拆分 |
| Composite组合模式 | 将对象组合成树形结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性 | 树形目录结构 |
| Decorator装饰模式 | 动态的给一个对象添加一些额外的职责。他提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活 | 附加职责 |
| Facade外观模式 | 定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用 | 对外统一接口 |
| Flyweight享元模式 | 提供支持大量细粒度对象共享的有效方法 |  |
| Proxy代理模式 | 为其他对象提供一种代理以控制这个对象的访问 |  |

## 行为型模式

|  |  |  |
| --- | --- | --- |
| 设计模式名称 | 简要说明 | 速记关键字 |
| Chain of Responsibility职责链模式 | 通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，直到有一个对象处理这个请求 | 传递职责 |
| Command命令模式 | 将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作 | 日志记录，可撤销 |
| Interpreter解释器模式 | 给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子 |  |
| Iterator迭代器模式 | 提供一种方法来顺序访问一个聚合对象中的各个元素，而不需要暴露该对象的内部表示 |  |
| Mediator中介者模式 | 用一个中介对象来封装一系列的对象交互。它使对象不需要显式地相互调用，从而达到低耦合，还可以独立地改变对象间的交互 | 不直接引用 |
| Memento备忘录模式 | 在不破坏封装性的前提下，捕获一个对象的内部状态，并在对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态 |  |
| Observer观察者模式 | 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新 |  |
| State状态模式 | 允许一个对象在其内部状态改变时改变它的行为 | 状态变成类 |
| Strategy策略模式 | 定义一系列算法，把他们一个个封装起来，并且使他们之间可互相替换，从而让算法可以独立于使用他们的用户而变化 | 多方案切换 |
| Template Method模板方法模式 | 定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法结构即可重新定义算法的某些特定步骤 |  |
| Visitor访问者模式 | 表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作 |  |

# 原型

原型可以划分为不同的种类。从原型是否实现功能来分，可以分为水平原型和垂直原型；从原型最终结果来分，可以分为抛弃式原型和演化式原型。

抛弃式原型主要用于界面设计。抛弃式原型的基本思路就是开始就做一个简单的界面设计，用来让用户由直观感受，从而可以提得出需求，等需求获取之后，可以把这个界面原型抛弃不用。

演化式，会把原型保留，通过不断的演化，逐步形成最终产品。

# 静态绑定和动态绑定

所谓静态绑定是指在程序编译过程中，把函数（方法或者过程）调用与响应调用所需的代码结合的过程。

动态绑定是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

# 面向对象

面向对象程序设计（OOP）：是一种实现方法，在这种方法中，程序被组织成许多相互协作的对象，每个对象代表某个类的一个实例，而类则属于一个通过继承关系形成的层次结构。

面向对象分析（OOA）：是一种分析方法，利用从问题域的词汇表中找到的类和对象来分析需求，重点是找到和描述问题领域的对象或者概念，然后构建真实世界的模型，利用面向对象的观点来看世界。

面向对象设计（OOD）：是一种设计方法，包括面向对象分解的过程和一种表示法，这种表示法用于展现被设计系统的逻辑模型（类和对象结构）和物理模型（模块和处理架构）、静态模型和动态模型。

# 设计原则

1、单一职责原则：设计目的单一的类

2、开放-封闭原则：对扩展开放，对修改封闭

3、李氏（Liskov）替换原则：子类可以替代父类

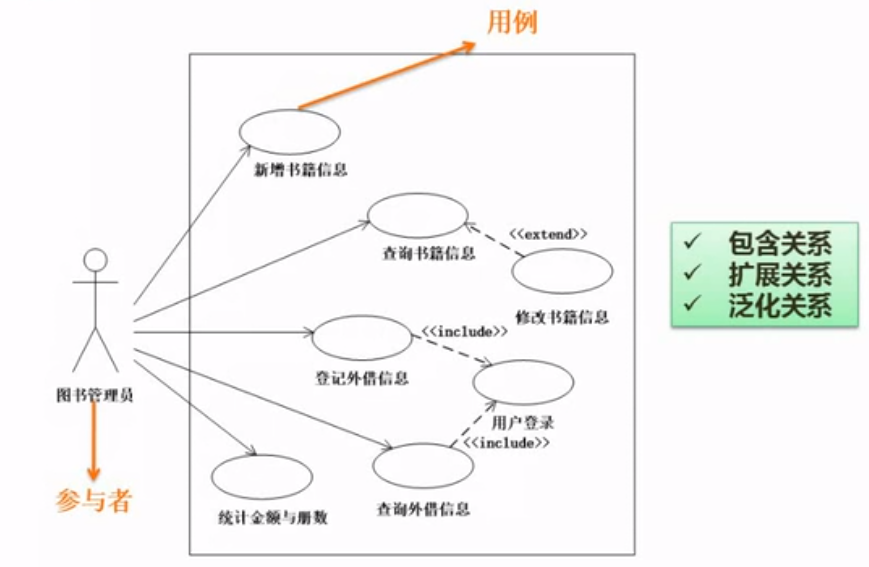
4、依赖倒置原则：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程

5、接口隔离原则：使用多个专门的接口比使用单一的总接口要好

6、组合重用原则：要尽量使用组合，而不是继承关系达到重用目的

7、迪米特（Demeter）原则（最少知识法则）：一个对象应当对其他对象有尽可能少的了解

# 用例图



# 《计算机软件保护条例》

由国务院颁布。

第十一条：接受他人委托开发的软件，其著作权的归属由委托人与受托人签订书面合同约定；无书面合同或者合同未明确约定的，其著作权由受托人享有。

# 《中华人民共和国著作权法》

第十八条规定：“美术等作品原件所有权的转移，不视为作品著作权的转移，但美术作品原件的展览权由原件所有人享有”。

著作权权力保护期限不受限制的由：署名权、修改权、保护作品完整权。保护期限为作者终身及死后50年的包括：发表权、使用权和获得报酬权。

著作权是提高对作品，包括文学、艺术、自然科学、社会科学和工程技术领域内具有独创性并能以某种有形形式复制的智力成果。专利权保护的是具有创造性的发明及设计等成果。商业秘密权用来保护商家的秘密，而软件的技术信息及经营信息正属于商业秘密的范畴，因此需要用商业秘密权来保护。

根据我国法律法规的规定必须使用注册商标的是烟草类商品。

# PERT图

PERT图即计划评审技术，是目前项目管理的一项重要手段和方法，利用它能协调整个计划的各道工序，合理安排人力、物理、时间、资金，加速计划的完成。其中的关键路径就是要找到最长路径。PERT图不能清楚的描述各任务之间的并行情况。

# 项目估算

项目估算的常用方法主要有专家判断法、启发式法和机器学习法等。

专家判断法是指向学有专长、见识广博并有相关经验的专家进行咨询、根据他们多年来的实践经验和判断能力对计划项目作出预测的方法。很显然，采用这种方法容易受到专家经验和主观性的影响。

启发式方法使用一套相对简单、通用、有启发性的规则进行估算的方法，它具有参数难以确定、精确度不高等特点。

机器学习法是一种基于人工智能与神经网络技术的估算方法，它难以描述训练数据的特诊和确定其相似性。

无论采用哪种估算方法，估算得到的结果都是大概的，而不是精确的。

# 项目管理工具

项目管理工具用来辅助软件的项目管理活动。通常的项目管理活动包括项目的计划、调度、通信、成本估算、资源分配即质量控制等。一个项目管理工具通常把重点放在某一个或某几个特定的管理环节上，而不提供对管理活动包罗万象的支持。

项目管理工具具有以下特征：

1. 覆盖整个软件的生存周期
2. 为项目调度提供多种有效手段
3. 利用估算模型对软件的费用和工作质量进行估算
4. 支持多个项目和子项目的管理
5. 确定关键路径，松弛时间，超前时间和滞后时间
6. 对项目组成员和项目任务之间的通信给予辅助
7. 自动进行资源平衡
8. 跟踪资源的使用
9. 生成固定格式的报表和裁剪项目报告

成本估算工具就是一种典型的项目管理工具。

# 业务用例

业务用例其实是对用例思想的一种延续，只是改变了使用场合。用例是从使用者的角度定义“软件系统”的需求。而业务用例不研究”软件系统“需求，它关心的是一个”业务组织“对外提供哪些服务，支持哪些业务过程。业务用例描述的是业务参与者如何使用业务组织提供的服务的过程。因此业务用例其实是一种业务流程。

业务对象模型是描述业务用例实现的对象模型，即业务结构以及结构元素如何完成业务用例。

# 项目配置管理

配置项是构成产品配置的主要元素，配置项主要有以下两大类：

1. 属于产品组成部分的工作成果：如需求文档、设计文档、源代码和测试用例等
2. 属于项目管理和机构支撑过程产生的文档：如工作计划、项目质量报告和项目跟踪报告等。这些文档虽然不是产品的组成部分，但是值得保存。

# 需求变更

在需求管理过程中需求的变更是受严格管控的，其流程为：

1. 问题分析和变更描述。这是识别和分析需求问题或者一份明确的变更提议，以检查他的有效性，从而产生一个更明确的需求变更提议
2. 变更分析和成本计算。使用可追溯性信息和系统需求的一般知识，对需求变更提议进行影响分析和评估。变更成本计算应包含对需求文档的修改、系统修改的设计和实现的成本。一旦分析完成且确认，应该进行是否执行这一变更的决策
3. 变更实现。这要求需求文档和系统设计以及实现都要同时修改。如果先对系统的程序做变更，然后再修改需求文档，这几乎不可避免地会出现需求文档和程序的不一致。

# 项目范围管理

在初步项目范围说明书中已文档化的主要的可交付物、假设和约束条件的基础上准备详细的项目范围说明书，是项目成功的关键。范围定义的输入包括以下内容：

1. 项目章程。如果项目章程或初始的范围说明书没有在项目执行组织中使用，同样的信息需要进一步收集和开发，以产生详细的项目范围说明书
2. 项目范围管理计划
3. 组织过程资产
4. 批准的变更申请

# 逆向工程

逆向工程是在软件维护时，由于缺少文档资料，而对软件的一种分析。

# 判定表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 |
| 条件 | 订单金额 | >5000 | >5000 | <=5000 | <=5000 |
| 欠款时间 | >60 | <=60 | >60 | <=60 |
| 操作 | 不予批准 | √ |  |  |  |
| 发出批准书 |  | √ | √ | √ |
| 发出发货单 |  | √ | √ | √ |
| 发借款通知书 |  |  | √ |  |

第2列和第4列操作相同，可以合并

# COCOMOII

是一种成本估算模型，估算选择有：对象点、功能点和代码行

# CMMI（能力成熟度模型集成）

是若干过程模型的综合和改进。连续式模型和阶段式模型是CMMI提供的两种表示方法，而连续式模型包括6个过程域能力等级：

CL0：未完成的，未执行或未达到CL1定义的所有目标

CL1：已执行的，共性目标是过程将可标识的输入工作产品转换成可标识的输出工作产品，以实现支持过程域的特定目标

CL2：已管理的，共性目标集中于已管理的过程的制度

CL3：已定义级的，共性目标集中于以定义过程的制度

CL4：定量管理的，共性目标集中于可定量管理的过程的制度化

CL5：优化的使用量化（统计学）手段改变和优化过程域，以对付客户要求的可持续改进计划中的过程域的功效

# UML

UML用关系把事物结合在一起，主要有下列四种关系：

1. 依赖（Dependency），依赖是两个事物之间的语义关系，其中一个事物发生变化会影响另一个事物的语义。
2. 关联（Association），关联描述一组对象之间连接的结构关系。
3. 泛化（Generalization）。泛化是一般化和特殊化的关系，描述特殊元素的对象可替换一般元素的对象。
4. 实现（Realization），实现是类之间的语义关系，其中的一个类指定了由另一个类保证执行的契约。

# 模块化设计

模块化设计的原则应该是尽量考虑高内聚、低耦合。耦合取决于各个模块之间接口的复杂程度、调用模块的方式及通过接口的信息类型。

1. 模块独立，模块的独立性原则表现在模块完成独立的功能，与其他模块的联系应该尽可能的简单，各个模块具有相对独立性。
2. 模块的规模要适当，模块的规模不能太大，也不能太小。如果模块的功能太强，可读性就会较差，若模块的功能太弱，就会有很多的接口。
3. 分解模块时要注意层次，在进行多层次任务分解时，要注意对问题进行抽象化。在分解初期，可以只考虑大的模块，在中期，再逐步进行细化，分解成较小的模块进行设计。

# 系统维护

更正性：由于系统测试不可能揭露系统存在的所有错误，因此在系统投入运行后频繁的实际应用过程中，就有可能暴露出系统内隐藏的错误。

适应性维护：是为了使系统适应环境变化而进行的维护工作。

完善性维护：在系统的使用过程中，用户往往要求扩充原有系统的功能，增加一些在软件需求规范中没有规定功能与性能特征，以及对处理效率和编写程序的改进。

预防性维护：系统维护工作不应总是被动地等待用户提出要求后才进行，应进行主动的预防性维护，即选择那些还有较长使用寿命，目前尚能正常运行，但可能将要发生变化或调整的系统进行维护，目的是通过预防性维护为未来的修改与调整奠定更好的基础。

# 数据流图

在结构化分析中，数据流图用来记录系统中的数据和数据在特定过程中的流动，即数据如何被采集、处理、保存和使用的（围绕信息系统的功能）。外部实体指系统之外又与系统有联系的人或事物。他表达了该系统数据的外部来源和去处。