计算机导论期末理论复习

- **第一部分：计算、计算机发展史、计算模型**
  - **什么是计算？转换/变换 什么是计算思维？**
    - **计算**是规则下的变换，字符串或集合->字符串或集合
    - **计算思维**是运用计算机科学的基础概念进行问题求解、系统设计、以及人类行为理解等涵盖计算机科学之广度的一系列思维活动。
  - **什么是程序？**
    - **程序**：指令（操作）序列。
      - 按照工作步骤事先编排好的、具有特殊功能的指令序列。
    - **程序设计**：程序设计是给出解决特定问题程序的过程，是程序构造活动中的重要组成部分。程序设计往往以某种程序设计语言为工具，给出这种语言下的程序。
    - **程序设计语言**：用于书写计算机程序的语言，用于表达和描述要加工的数据以及求解问题的步骤和过程。是根据预先定义的规则（词法、句法）、由一个有限字母表上的字符构成的字符串的总体。
  - **什么是存储程序的概念？**
    - 是将根据特定问题编写的程序存放在计算机***存储器***中，然后按存储器中的存储程序的***首地址***执行程序的第一条指令，以后就按照该程序的规定顺序执行其他指令，直至程序结束执行。
- **第二部分：计算机组成与原理**
  - **冯诺依曼计算机的组成结构**
    - **指令和数据**采用***二进制***表示，从而简化机器的逻辑线路；
    - **指令和数据**一样存储在***主存储器***中；
    - 计算机由***运算器、控制器、（主）存储器（内存）、输入设备、输出设备***五大部分组成。（**CPU**（中央处理器）包含运算器和控制器，**主机**包括 CPU 和内存）
      - ***控制器***
        - **控制器结构**
          - 是计算机中的控制中心，用来翻译指令代码，安排操作次序，向其他部件发出控制信号,指挥计算机部件协同工作。
        - **指令及指令系统**
          - *指令*由“0”和“1”的二进制码组成 ,是指挥计算机工作的命令，是计算机唯一可以直接识别的语言；
          - *指令系统*是一种计算机能够直接识别和执行的全部指令的集合
        - **指令的执行**

        - CPU 运行一个程序包含三个主要步骤：取指令；分析指令；执行指令
    - **指令的组成**
        - **操作码**指出指令应该执行什么性质的操作和具有何种功能;
        - **地址码**指出指令中操作数所在的存储器地址、寄存器地址或 I/O 地址。
    - **指令计数器 PC**
        - 用来存放下一条待执行的指令在*主存储器中的地址*
- ***运算器***
    - 用来完成各种算术运算和逻辑运算。
    - 核心部件是算术逻辑单元(ALU)和若干寄存器。
    - 根据**指令**所规定的寻址方式，运算器从主存储器或寄存器中取得操作数，进行计算后，返回到指令所指定的寄存器或主存储器中。
- ***存储器与存储系统***
    - **存储器**
        - 存放要执行的程序和要处理的数据的部件。
        - 基本功能是按照要求向指定的位置写入或读出信息，由存储单元、存储地址寄存命（MAR ）、存储数据寄存器（MDR）组成。
        - *存储空间，存储地址、存储单元，位与字节*
            - 主存储器被划分为**存储单元**。
            - 对每一个存储单元进行有序编号，这种编号就是存储单元地址。
            - 一个存储器中包会的存储单元总数叫做**存储容量**，是衡量存储器空间大小的指标，以字节(byte)为基本单位。
            - **存储周期**是指存储器连续读出或写信息所需时间,单位为 ns ,用米衡量存储器读写操作的工作效率。
            - 主存储器大都采用半导体存储器,按功能分为随机存取存储器(RAM)、只读存储器(ROM), RAM 对任意存储单元的读写时间都是相同的, ROM 只能读不能写。
    - **存储系统**：高速缓存、内存、外存
        - 存储系统是指计算机中由存放程序和数据的各种存储设备、控制部件及管理信息调度的设备（硬件）和算法（软件）所组成的系统。
            - **高速缓冲存储器**（Cache）其原始意义是指存取速度比一般随机存取记忆体（RAM）来得快的一种 RAM。
            - ***内存***(Memory)是计算机的重要部件之一，也称**内存储器和主存储器**，它用于暂时存放 CPU 中的运算数据，与硬盘等外部存储器交换的数据。

        - 是外存与 CPU 进行沟通的桥梁，计算机中所有程序的运行都在内存中进行。
        - 只要计算机开始运行，操作系统就会把需要运算的数据从内存调到 CPU 中进行运算，当运算完成，CPU 将结果传送出来。
        - 内存性能的强弱影响计算机整体发挥的水平
        - 内存的运行决定计算机整体运行快慢的程度。
        - **结构：**
            - **系统区**
            - **用户程序代码区**
            - **静态存储区**：存放程序运行期间不释放的数据**（静态局部变量、全局变量）；**
            - **栈区**：存放程序运行期间会被释放的数据**（函数参数、非静态局部变量）；先进后出**
            - **堆区（可以用 malloc()和 free()分配和释放）；**
    - **外储存器**是指除计算机内存及 CPU 缓存以外的储存器，此类储存器一般断电后仍然能保存数据。比如硬盘、软盘、光盘、U 盘。（属于输入/输出设备）
    - ***输入设备***
    - ***输出设备***
- **计算机系统**
    - **软件**
        - 包括程序、数据及其相关文档的完整集合。分为系统软件和应用软件
    - **硬件**
- **操作系统（系统软件）**
    - **操作系统是由一系列具有控制和管理功能的子程序组成的大型系统软件**
    - **功能**：作业管理、CPU(进程)管理、内存管理、设备管理、文件管理
    - **进程**：是一个具有独立功能的程序对某个数据集在处理机上的执行过程，是资源分配的基本单位。一个程序（作业）的执行总是会产生一个或多个进程。
    - **作业**：用户请求计算机计算的一个计算任务，由程序、数据和作业说明书组成，用于完成用户的一个计算目标。
    - **程序与进程的关系：**
        - 进程是动态的，强调执行过程，而程序是静态的；
        - 进程具有并发性(宏观上同时运行)，程序没有；
        - 不同的进程可以对应同一程序，只是该程序对应的数据集不同。

- # 第三部分 程序语言及程序设计基础
  - ## 标识符
    - **标识符**是由程序员定义的单词，用来给程序中的数据、函数和其他用户自定义对象命名。
    - 程序设计语言本身会定义一些**专用单词**，称之为保留字或关键字，它们具有**特定含义**，程序员**不能另做他用**。如：C 语言规定了 **32** 个关键字。
  - ## 数据类型及数据类型的三要素（表示、存储、操作）
    - **数据**是计算机处理的**对象。**
    - 数据类型三要素：
      - 1、**数据的抽象（定义）**——定义了一系列的逻辑表达-值(该类型数据能够取值的范围)
      - 2、**数据的存储空间（范围）**——所占存储空间大小，即占用多少字节，二进制
      - 3、**数据的操作**——能应用于这些值上的一系列操作
  - ## 变量及变量的三要素
    - 变量用来**代表*内存存储空间***，该存储空间用来存放被加工的数据或处理的结果。源程序中对变量的操作（读和赋值）实际上是对存储空间的读写操作。
    - 变量三要素：**名称、值和数据类型。**
      - 变量**名**（代表存储区地址）
      - 变量**值**（存储内容）
      - 变量**类型**（存储区存放的数据的类型）
  - ## 常量
    - 三类常量：**文字常量(1,2,3)、命名常量（const）、符号常量(define)**
    - **命名常量**和**符号常量**的区别
      - *内存分配*上（命名常量会在内存的程序运行数据区分配到内存，而符号常量不会）
      - *类型定义*上（命名常量精确定义了数据类型，排除了程序的不安全性；而符号常量只是简单的替换，并采用系统默认类型，存在不安全性）
  - ## 表达式
    - 由***运算符、操作数***和***括号***组成，是计算求值的基本单位
    - **操作数**
      - *常量*、*变量*、*函数调用* 和*表达式*;
    - **类型**
      - 算术、关系、逻辑、赋值；
      - **优先级：**
        - *算术运算符* > *关系运算符* > *逻辑运算符* > *赋值运算符*

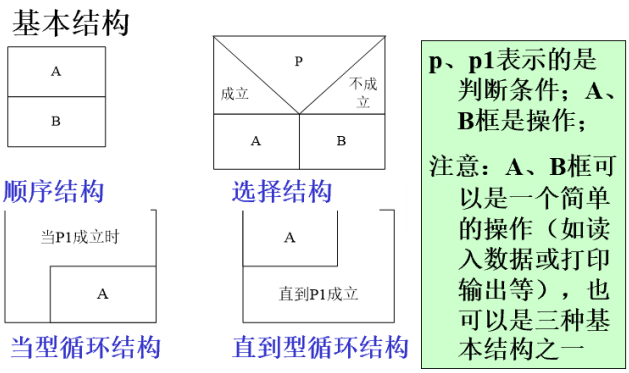
- **语句**
  - **表达式语句**
  - **复合语句**
    - 复合语句是包含零个或多个语句的代码单元，使得一组语句成为一个整体，也被称为块。
  - **选择语句（分支语句）**
    - 在写程序时，有时往往需要测试某一个条件 是否成立，然后根据测试结果来控制程序后续执行路径。此时要用到选择语句。
      - 1）**if-else** 语句
      - 2）**switch** 语句
  - **循环语句**
    - 有时程序中需要多次运行同一段代码（重复做相同的事情）。这种**控制结构**称为**循环结构。**
    - 在循环结构中，需要刻画出**重复执行**的是哪些**动作**（**循环体**），以及什么**条件**下需要重复（**循环条件**）。
    - C 语言提供了三类用于实现循环结构的语句：
      - **1）while**
      - **2）do-while**
      - **3）for**
  - 标记语句
  - 跳转语句

第四部分 算法设计方法

- **什么是算法？算法与程序的区别**
  - **算法**
    - 是解决问题的步骤序列（操作序列）
  - **算法与程序的区别**
    - 算法：面向人对问题求解的方法的表达；
    - 程序：面向机器用计算机语言实现并运行；
- **算法的五大特征**
  - **可执行性**：算法中的每一个步骤都是计算机可执行的（在计算机能力集范围内）；
  - **确定性**：算法中的每一个步骤，必须是明确定义的，不得有任何歧义性；
  - **有穷性**：算法必须在执行有穷步之后结束；
  - **有输入信息的说明**：加工对象的要求；

- **有输出信息的步骤**： 输出问题的答案。

- **基本组成要素**
  - 数据对象的定义
  - 表达式（**算数、关系、逻辑**）
  - 三种基本**语句**:
    - *赋值、输入、输出*
  - 三种基本**程序结构**:
    - *顺序、分支、循环*
  - 函数调用（子程序）
- **N-S 流程图**
  -



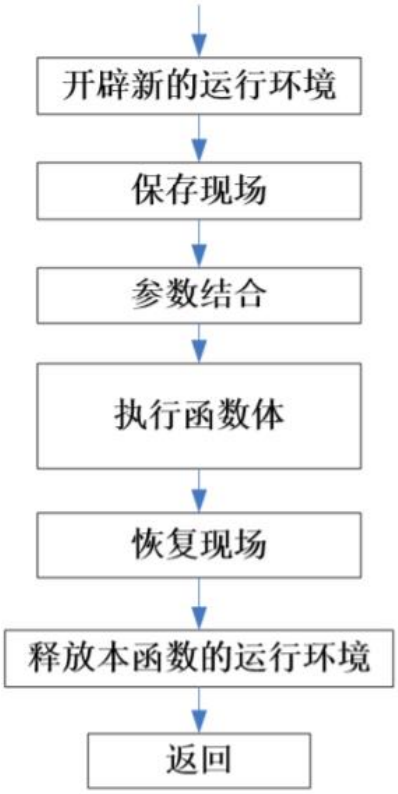
- **基本设计思想：**
  - **自顶向下、逐步求精**

第五部分 子程序（函数）

- **函数的定义**
  - **函数**是封装并给以命名的一段程序代码，完成明确的功能，可供调用。**是 C 语言程序设计的基本单位。**一个 C 语言程序是由一个主函数（main）和其他若干个子函数组成的。
- **函数原型**
  - **作用：**函数原型是对被调用函数的接口声明；
  - 告诉编译器**函数返回的***数据类型*、**函数所要接收的***参数个数、参数类型*和*参数顺序*;
  - 编译器用函数原型**校验**函数调用是否正确。
- **函数的调用**
  - **参数原理**
    - ***形参与实参***

- **实参**可以是**常量、变量、表达式、函数**等，在调用前必须具有确定的值。调用时，会为形参分配内存，然后将实参的值复制一份到形参中。
- **实参**的**个数、类型和顺序**，应该与形参一致，才能正确地进行数据传递。
- **实参**和**形参**占用不同的内存单元，即使同名也互不影响。（此处实参是变量的情况）；不同函数中可以使用相同名称和类型的变量，它们占用不同的内存单元，互不影响。
- **实参**对**形参**的数据传送是单向的，即只能把实参的值传送给形参，而不能把形参的值反向传送给实参。
- **形参变量**只有在被调用时，才分配内存单元；调用结束时，即刻释放所分配的内存单元。

- **调用过程**
  - **堆栈**
    - 堆栈是一种**数据结构**，而且是一种数据项按序排列的数据结构，只能在一端(称为栈顶(top))对数据项进行插入和删除
  - **函数活动记录**
    -



    - **如何理解函数调用和返回?**
      - **函数的活动记录（栈区）：**
        - 函数执行时需要的**数据空间**:
          - 1.本次函数执行过程中的**数据对象**，如形参、局部变量等；
          - 2.用以管理函数调用过程的信息（**当前机器的状态信息等**）。

        - 开辟新的运行环境
            - 即指在栈区的栈顶创建一个函数活动记录。
        - 释放函数的运行环境
            - 即指从栈顶将活动记录释放。
    - **如何确保能够逐层返回到上一级调用?**
        - **函数 A 调用函数 B，则在函数 B 的活动记录中记录了 A 的返回地址。**返回前取出该地址，即能正确返回。
    - **为何不同的函数可以使用同名的参数和变量?**
        - **因为不同函数的活动记录占用不同的内存单元，程序运行时始终是从位于栈顶的活动记录中取形参和变量的值。**
- **子程序设计（函数设计）**
    - **设计原则**
        - **高内聚**：功能相对独立和完整；
        - **低耦合**：与外界（调用者）的关系尽量松散，不要太紧密，使其能方便地被重用；
        - 需要合理地设计子程序参数和子程序执行的局部环境 来达到以上目标。
    - **函数设计**
        - 明确该函数的**功能**（函数应该只完成单一的任务）；
        - 定义该函数的**接口**（即**函数头**，包括*函数名、参数和返回值*）
        - 定义该函数的**功能实现部分**
- **变量的作用域**
    - 全局变量、局部变量
- **目的**
    - 避免程序“复用”，避免在程序中使用重复代码；
    - 结构化程序设计的需要：
        - 自顶向下、逐步细化，将复杂问题分解为相对简单的子问题，这些子问题用子程序实现，从而提高主程序结构的清晰性和易读性。
    - 使程序的调试和维护变得更加容易

第六部分 递归（函数递归）

- **概念&定义**
    - 从程序书写来看，在定义一个函数时，若在函数的功能实现部分又出现对它本身的调用，则称该函数是递归的或递归定义的。

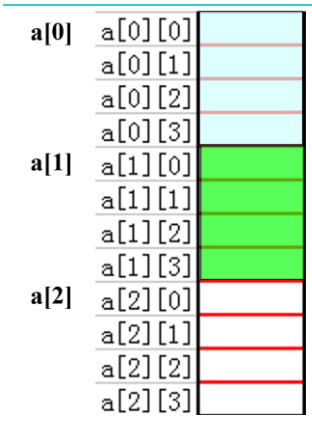
    - 从函数动态运行来看，当调用一个函数 A 时，在进入函数 A 且还没有退出（返回）之前，又再一次调用 A，则称为函数 A 的递归调用。
- **种类：**
    - 直接递归：函数体里面发生对自己的调用；
    - 间接递归：函数 A 调用函数 B，而函数 B 又直接或间接地调用函数 A。
- **参数设计：**
    - 不建议使用全局变量。
- **阶段：**
    - **递归**：不断简化问题，把对较复杂问题（规模为 n）的求解转化为比原问题简单的问题（规模小于 n），当递推到最简单的不用再简化的问题时，递推终止。
    - **回归**：当获得最简单情况的解后，逐级返回，依次得到稍复杂问题的解 。
- **核心思想：**
    - （化简为同类问题，分解直至能求解。）先将一个问题转化为与原问题性质相同、但规模小一级的子问题，然后再重复这样的转化，直到问题的规模减小到我们很容易解决为止。
    - **注意：递归算法最外层肯定采用的是选择结构！**

第七部分 数组

- **概念**
    - 数组是一组连续的存储单元（存储结构）
- **数组的定义、下标运算符**
    - 数组名[元素序号]。其中元素序号又称下标。
    - 下标运算符“[ ]”（数组操作）
- **数组作为函数参数 （数组名是引用，传引用）**
    - **一、数组元素作函数实参**
        - **数组元素就是下标变量，它与普通变量并无区别。 因此它作为函数实参使用时与普通变量完全相同，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，实现单向的按值传送。**
    - **二、将数组名作为函数的实参**
        - ***数组名具有特殊含义，它代表数组的首地址。***
        - **需求**：需要在函数（子程序）中操作数组
        - **问题**：函数参数是单向传值。简称“值参”
        - 另外一种参数传递机制– **传引用（变量地址）**
        - **地址-引用-指针 传地址=传引用**

- 如果函数要接收一个数组进行处理，则形参中必须有一个是同类型的数组，如 int b[]，用于接收调用函数中数组的首地址。通常还要设计一个形参用于接收数组的大小，如 int size。
- **形参的“[ ]”中不必包含数组大小**；如果包括了，编译器会将其忽略掉。同理：函数原型

- **二维数组**
  - **在内存中的存放**
    - 在C语言中，二维数组是按行排列的。按行顺次存放，先存放 a[0]行，再存放 a[1]行，最后存放 a[2]行。
    -



int a[3][4] ;

第1行第0列元素的地址表示：
printf("%p\n", &a[1][0]);
printf("%p\n", a[1]);
以上两种方式是等价的！

二维数组可以看成是一维数组组成的数组，a[1]代表的是一维数组的数组名。

C语言中二维数组是按行排列的

A[i]的内存地址：A+i*M（M是一个数组元素的字节数）

A[0]	A[0]		A[i]		A[ArraySize-1]
↑	A[1]		↑		↑
A	A+1*M		A+i*M		A+(ArraySize-1)*M

因此，要访问一个一维数组元素，要知道数组首地址、下标。

A[i][j]的内存地址：A+(i*数组列数+j)*M

因此，要访问一个二维数组元素,除了要知道数组首地址、行下标、列下标，还需要知道数组的**列数**。

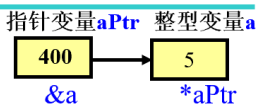
第八部分 指针与数组

- **指针的概念，指针的定义（语法）**
  - **概念**：指针是变量，存放内存地址
  - **定义**：
    - **[存储类型]　数据类型　* 指针变量名;**
    - 从变量讲，指针也具有变量的三个要素：
      - **变量名**：这与一般变量取名相同。
      - **指针变量的类型**：是指针所指向的变量的类型，而不是自身的类型。
      - **指针的值**：是某个变量的内存地址。
    - 指针所存放值的类型是整型，因为任何内存地址都是整型的。

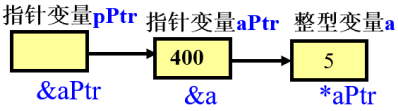
    - 但是指针变量的类型却定义成它所指向的变量的类型，因为地址对于编译器来说不足以解析地址的数据类型。

- **初始化：**
    - 没有初始化的指针变量可能指向任意地址。
    - 指针在使用前**必须初始化**，否则将会导致意想不到的问题！
    - 指针变量可以赋值为 **0**（**唯一可直接赋给指针变量的整数值**）、**NULL**（在 stdlib.h 中定义的常量, 代表 0）或**某个内存地址**；
    - 当赋值为 0 或者 NULL 时，表示该指针为**空指针**，不指向任何地址。推荐当指针不指向任何地址时，赋值为 **NULL**。
- **指针运算符：**
    - **& 取变量的地址**
    - *** 取指针所指变量的值 （间接访问）**
        - 注意：当一个指针 p 是空指针时(即 p==NULL 或 p==0)，禁止询问该指针指向的内存值！
    - **= 指针的赋值运算**
    -

解析 &* aPtr=&(*aPtr)=&a=aPtr



解析* & aPtr =
*(&aPtr)=*pPtr=aPtr



aPtr = &a = &(*aPtr) = * (& aPtr)
a = *aPtr = *(&a) 但不能写成 & (* a) ！

***与&为互逆运算**

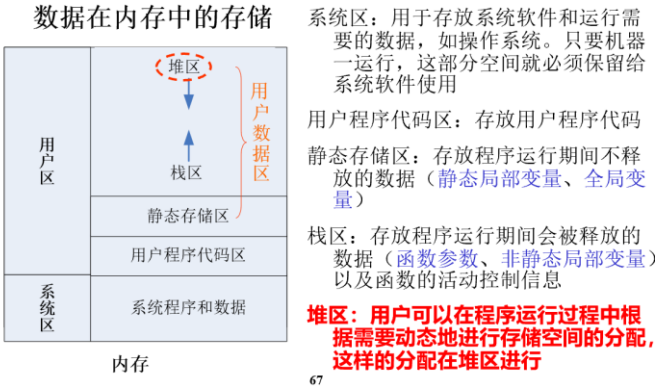
- **指针作为函数参数：传引用**
    - **传引用调用：**
        - 实参是个变量，对形参的修改其实就是对实参的修改，这样就可以得到（返回）多个函数处理结果。
    - C 语言中所有的函数调用都是**传值调用(将实参的值传递给形参)**，但 C 语言可以**模拟实现传引用调用**，使得被调用函数可以**修改主调函数中的变量**，从而得到**多个函数处理结果；**
    - 在主调函数中，将要让**被调函数修改的变量的地址作为实参传递给被调函数**，然后在被调函数内部**通过指针和间接访问运算符***即可实现对这些变量的访问！
    - **特点：**
        - 地址传递；

    - 共享内存；
    - 被调函数可以修改主调函数中的变量

- **指针的运算**
    - **赋值运算**：将指针赋 0、NULL、变量地址，或相同类型指针之间的赋值运算；
    - **算术运算**：指向数组元素的指针加(减)一个整数的运算；指向相同数组中元素的指针之间的减运算；
        - 自增(p++）、自减(p--)、加上一个整数(p+3)、减去一个整数(p-3)、减去一个指针(p1-p2)。
        - **指针加上（减去）一个整数：并非简单地加上（减去）一个整数，而是加上（减去）该整数和指针所指对象的大小的乘积！**
        - **若 p1 与 p2 指向同一数组，p1-p2=两指针间元素的个数，而非　字节数**
        - **p1+p2 无意义**
    - **关系运算**：比较两个指针所指向的存储单元的内存地址大小
        - 指向相同数组中元素的指针之间的比较运算； 指针与 0、NULL 之间的比较运算。
        - **若 p1 和 p2 指向同一数组中的元素，则比较 p1 和 p2 有意义**
        - **若 p1 与 p2 不指向同一数组，则比较 p1 和 p2 无意义；**
    - *应用指针的算术运算可以实现用指针来访问数组元素！*
    - **注意：ptr+=i 与 ptr+i 的区别**
- **指针与数组：**
    - **指针与数组（存储结构），通过地址（数组名是地址、指针存放地址），建立关系—指针变量可以操作数组。**
    - **基于指针与数组的关系，使得指针运算在赋值运算的基础上，又增加了算术运算、关系运算、逻辑运算。**
        - *指向：修改指针变量的值*
        - *表示：不修改指针变量*
    - **数组元素的 4 种访问方式（下标、偏移量）**
        - **a[i]：数组/下标表示法**
        - ***(a+i)：数组/偏移量表示法**
        - **aPtr[i]：指针/下标表示法**
        - ***(aPtr+i) ：指针/偏移量表示法**
        - 实际上 a[i]在编译时要被转换成*(a+i) 。因此用*(a+i)访问数组元素可以节约编译时间。不过 a[i]方式可以使程序更清晰。
    - **函数**

- 函数定义：void bubbleSort(int array[], const int size)
- 函数调用：bubbleSort(a,6)
- 以上函数定义等价于：void bubbleSort(int * array, constint size)
- //将数组首地址传递给指针 array 后，就可以通过 array 来操作数组元素。

- **动态数组**
  - 可以定义一个指针，然后调用**内存申请库函数**在**堆区**分配一片连续的内存，并将内存的起始地址保存在这个指针中，这就建立了一个**动态数组**。然后通过指针可以访问动态数组的各个元素。



  - **库函数 malloc()**
    - **用法：void * malloc(unsigned size)**
    - **功能**：在内存的堆区分配 size 个字节的连续空间。
    - **返回值**：若申请成功，则返回新分配内存块的起始地址；否则，返回 NULL。
    - 函数原型所在**头文件**：stdlib.h。
  - 在程序运行时用 malloc()函数申请的内存，在不再需要时，**必须由程序员自己负责用 free()函数进行释放**。否则，即使程序运行结束，操作系统也不会去释放这些内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。
  - **库函数 free()**
    - **用法：void free(void * ptr)**
    - **功能：**释放由 ptr 指向的内存块（ptr 是用来调用 malloc() 函数的返回值）。
    - **返回值：**无。
    - 函数原型所在**头文件**： stdlib.h
  - **一个好的习惯是释放了内存后，将指针 array 赋值为 NULL，防止产生“野指针”**
    - **free(array);**
    - **array=NULL;**

- **指针数组**

- 数组中的元素为指针变量。常用于处理多个字符串

- **指针与数组的不同**
  - 在定义数组时为数组的各个元素分配了全部的存储区，而在定义指针时，仅仅分配四个字节的存储区存放地址。

第九部分 自定义数据类型 结构

- **用户自定义（构造）数据类型**
  - 结构是用其他类型的对象构造出来的派生数据类型（注意：结构是一种数据类型）。
  - 用户自己定义的结构是一种数据类型，与系统定义的标准类型（int、char 等）一样，可用来定义结构变量。结构变量是用一个名字引用的相关变量的集合。
  -

- **结构的定义**
  - **（1）间接定义法——先定义结构，再定义结构变量**

```
struct book{
   char title[MAXTITL];
   char author[MAXAUTL ];
   float value;
 } ;
int main(void)
{
   struct book library;
   ......
}
```

library是结构变量

  - **（2）*直接定义法*——在定义结构类型的同时， 定义结构变量**

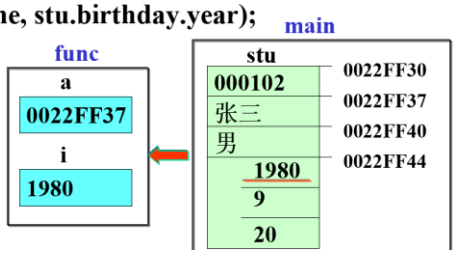
```
struct book{
   char title[MAXTITL];
   char author[MAXAUTL ];
  float value;
} library;  //在定义之后跟变量名
```

- **结构成员的访问**
  - **使用结构成员运算符“.”**
    - **结构变量.成员.子成员.....最低一级子成员**
  - **使用指针运算符“->”**
    - 通过**指向结构变量的指针**访问结构成员：**指针名->结构成员名**

- 或 ***（指针名).结构成员名*** *//运算符. 优先级高于运算符*，故必须加括号
- 运算符. 和->优先级相同，自左向右结合，ptr->birthday.year 相当于(ptr->birthday).year

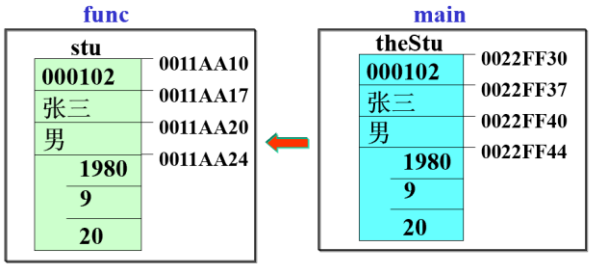
- **结构变量的操作（结构体的整体赋值）**
  - **1. 结构**与**结构变量是**两个**不同的概念，**其区别如同 int 类型与 int 型变量的区别一样。
  - **2.** 结构中的**成员名，**可以与**程序中的变量**同名，它们代表不同的对象，互不干扰。
  - **3. 对结构变量的操作有：**
    - **1) 获取结构变量的地址: &library;**
    - **2) 访问结构变量的成员（后面介绍）；**
    - **3) 把结构变量赋给同一类型的结构变量:**
      - **book1=book2;//将 book2 中各成员的值赋给 book1 中各成员**
    - **4) 用 sizeof 运算符确定结构变量的大小**
    - **sizeof(library)、sizeof(struct book)**
- **结构作为函数参数 结构传值、传引用（值参与变参）**
  - **把结构传递给函数的三种方式：**
    - **-传递单个成员**

```
void func(char a[],int i )/*函数定义*/
/*main函数中的函数调用*/
func(stu.name, stu.birthday.year);
```



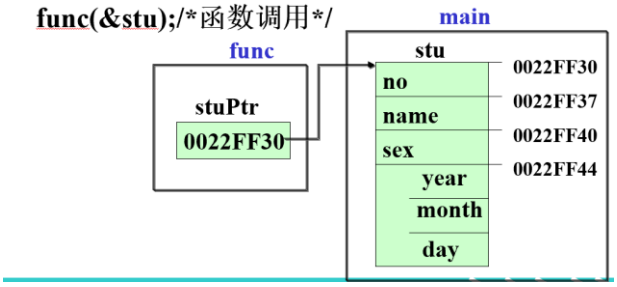
    - **- 传递整个结构（形参是实参的副本，两者互不影响）**

```
void func(struct student  stu)/*函数定义*/
func(theStu);/* main函数中的函数调用*/
```



    - **- 传递指向结构的指针（被调用函数可以操作调用函数中的结构变量）**

```
void func(struct student * stuPtr)/*函数定义*/
func(&stu);/*函数调用*/
```



- **传递结构、还是指向结构的指针?**
  - **传递指针**
    - **优点**：既可以工作在较早的 C 实现上，也可以工作在较新的 C 实现上；执行速度快：只需要传递一个地址。
    - **缺点**：缺少对数据的保护（可以使用 const 限定词解决这个问题）。
  - **传递结构**：
    - **优点**：安全性。函数处理的是原始数据的副本。
    - **缺点**：早期的 C 实现不处理这种代码，浪费时间和空间

第十部分：数据结构+算法=程序

- **基于数组、结构，信息与数据抽象，数据结构设计**
- **在数据结构设计的基础上进行算法设计**
- **计算机学科方法论：三个形态（抽象-理论-设计），**
- **算法设计的理论基础：递归函数（迭代设计）、图灵机**