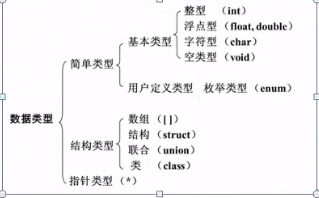
**内存四区专题**

**1数据类型本质分析**

**1.1数据类型概念**

* “类型”是数据的抽象
* 类型相同的数据具有相同的表示形式，存储格式以及相关的操作
* 程序中使用的所有的数据必定属于某一种数据类型



**1.2数据类型的本质思考**

* 思考数据类型和内存有关系吗
* C/C++为什么会引入数据类型

**1.3数据类型的本质**

* 数据类型可理解为创建变量的模具（模子）：是固定大小内存块的别名
* 数据类型的作用：编译器预算对象（变量）分配的内存空间大小

**1.4数据类型大小**

* 程序举例，如何求数据类型的大小

**1.5数据类型别名**

* 请问：数据类型可以有别名吗？数据类型可以自定义吗？

**1.6数据类型的封装**

* Void的字面意思是“空类型”，void \*则为“空类型指针”，void \*可以指向任何类型的数据
* 用法1：数据类型的封装

Int initHardEnv(void \*\*handle)

典型的如内存操作函数memcpy和memset的函数原型如下

Void \*memcpy(void \*dest, const void \*src, size\_t len);

Void \*memset(void \*buffer, int c, size\_t num);

* 用法2：void修饰函数返回值和参数，仅表示无

如果函数没有返回值，应该将其声明为void型

如果函数没有参数，应该声明其参数为void

Int function(void)

{return 1;}

* void指针的意义

C语言规定只有相同类型的指针才可以相互赋值

Void \*指针作为左值用于“接收”任何类型的指针

Void \*指针作为右值赋值给其他指针时需要强制类型转换

Int \*p1 = NULL;

Char \*p2 = (char \*)malloc(sizeof(char)\*20);

* 不存在void类型的变量

C语言没有定义void究竟是多大内存的别名

**1.7数据类型总结与扩展**

**2变量本质分析**

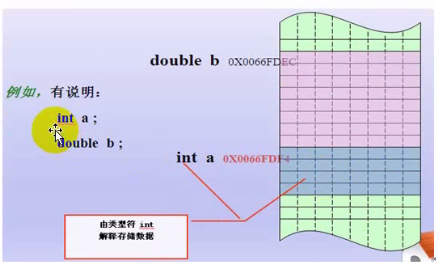
**2.1变量概念**

* 概念：既能读又能写的内存对象，称为变量；一旦初始化之后不能修改的对象则为常量
* 变量定义形式：类型 标识符,标识符,标识符,...,标识符;
* 例如：

Int x;

Int wordCut, Radius, Height;

Double FlightTime, Mileage, Speed;



**2.2变量本质**

1. 程序通过变量来申请和命名内存空间
2. 通过变量名访问内存空间

（一段连续）内存空间的别名（是一个门牌号）

1. 修改变量有哪些方法？
2. 直接
3. 间接。内存有地址编号，拿到地址编号也可以修改内存
4. 可以再给内存空间取别名吗？
5. 数据类型和变量的关系

通过数据类型定义变量

1. 总结及思考题
   1. 对内存，可读可写；
   2. 通过变量往内存读写数据
   3. 不是向变量读写数据，而是向变量所代表的内存空间中读写数据。问：变量跑哪里去了？

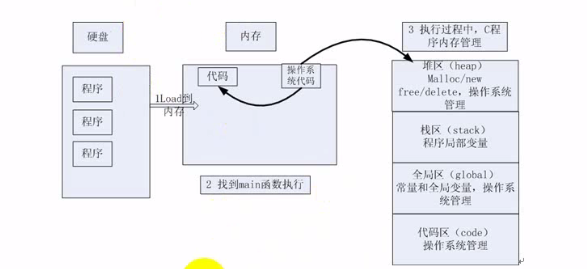
思考1：变量三要素（名称、大小，作用域），变量的生命周期？

思考2：C++编译器是如何管理函数1，函数2变量之家的关系的？

引出两个重要话题：内存四区模型、函数调用模型

**3程序的内存四区模型**

**3.1内存四区的建立流程**



流程说明：

1. 操作系统把物理硬盘代码load到内存
2. 操作系统把C代码分成四个区
3. 操作系统找到main函数入口，执行

**3.2各区元素分析**

|  |
| --- |
| 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 |
| 堆区（heap）：一般由程序员分配释放（动态内存申请与释放），若程序员不释放，程序结束时可能由操作系统回收 |
| 全局区（静态区）（static）：全局变量和静态变量的存储是放在一起的，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静态变量在相邻的另一块区域，该区域在程序结束后由操作系统释放 |
| 常量区：字符串常量和其他常量的存储位置，程序结束后由操作系统释放 |
| 代码区：程序代码区，存放函数体的二进制代码 |

结论：1.指针指向谁，就把谁的地址赋值给指针

1. 指针变量 和 它所指向的内存空间变量是两个不同的概念
2. 理解指针的关键是内存（没有内存就没有首地址编号，没有编号就没有指针）

**4函数调用模型**

**5内存四区强化训练**

**5.1全局区训练**

* Char \*p1 = “abcdefg”;

**5.2 堆栈区生命周期训练**

* Char p1[] = “abcdefg”;
* 返回基本类型
* 返回非基本类型

**5.3 堆栈属性训练**

* 测试heap生长方向
* 测试stack生长方向
  + Heap、stack生长方向和内存存放方向是两个不同的概念
  + 野指针
  + Malloc空间释放问题的训练
    - Free(p)
    - Free(p+1)

**6作业强化**

**指针专题**

**1铁律1：指针是一种数据类型**

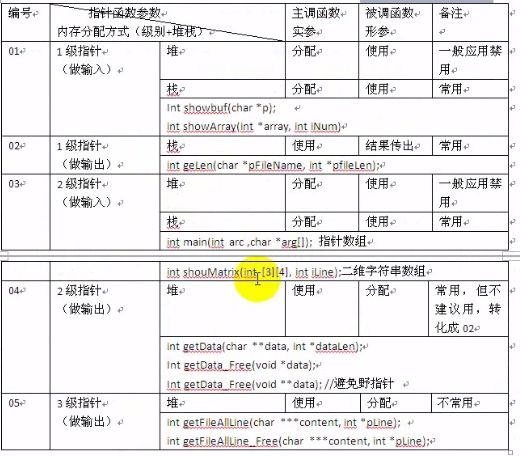
**2铁律2：间接赋值（\*p）是指针存在的最大意义**

1. 两码事：指针变量和它指向的内存空间变量
2. 条件反射：指针指向某个变量，就是把某个变量地址赋值给指针
3. \*p间接赋值的成立条件：3个
   * 1. 2个变量（通常一个实参，一个形参）
     2. 建立关系，实参取地址赋给形参指针
     3. \*p形参去间接修改实参的值
4. 引申：函数调用时，用n级指针（形参）改变n-1级指针（实参）的值
   * 1. 改变0级指针（int iNum=1）的值，有2种方式
     2. 改变1级指针（char \*p=0x111）的值，有2种方式
     3. 改变2级指针（char \*\*p1=0x111）的值，有2种方式
     4. 函数调用时，实参传给形参，用实参取地址，传给形参，在被调用函数里面用\*p，来改变实参，把运算结果传出来
     5. 指针作为函数参数的精髓

**3铁律3：理解指针必须和内存四区相结合**

1. 主调函数 被调函数
   * 1. 主调函数可把堆区、栈区、全局数据内存地址传给被调用函数
     2. 被调用函数只能返回堆区、全局数据
2. 内存分配方式
   * 1. 指针做函数参数，是有输入和输出特性的

**4铁律4：应用指针必须和函数调用相结合（指针做函数参数）**



指针做函数参数，问题的实质不是指针，而是看内存块，内存块是1维还是2维。

1. 若是基础类int变量，不需要用指针

2）若内存块是1维、2维

**5铁律5：一级指针典型用法（指针做函数参数）**

一级指针做输入

Int showbuf(char \*p)

Int showArray(int \*array, int iNum)

一级指针做输出

Int getLen(char \*pFileName, int \*pFileLen)

理解：

主调函数还是被调函数分配内存

被调函数是在heap/stack上分配内存

**6铁律6：二级指针典型用法（指针做函数参数）**

二级指针做输入

Int main(int src, char \*arg[]);字符串数组

Int shouMatrix(int [3][4], int iLine);

二级指针做输出

Int Demo64\_GetTeacher(Teacher \*\*ppTeacher);

Int Demo65\_GetTeacher\_Free(Teacher \*\*ppTeacher);

Int getData(char \*\*data, int \*dataLen);

Int getData\_Free(void \*data);

Int getData\_Free2(void \*\*data);//避免野指针

理解：

主调函数还是被调函数分配内存

被调函数是在heap/stack上分配内存

**7铁律7：三级指针典型用法**

三级指针做输出

Int getFileAllLine(char \*\*\*content, int \*pLine);

Int getFileAllLine\_Free(char \*\*\*content, int \*pLine);

理解：

主调函数还是被调函数分配内存

被调函数是在heap/stack上分配内存

**8铁律8：杂项，指针用法几点扩充**

1. 野指针 2种free形式

Int getData(char \*\*data, int \*dataLen);

Int getData\_Free(void \*data);

Int getData\_Free2(void \*\*data);

1. 2次调用

主调函数第一次调用被调函数求长度，根据长度，分配内存，调用被调用函数

1. 返回值char \*/int/char \*\*
2. C程序书写结构
   1. 商业软件，每一个出错的地方都要有日志，日志级别

**9铁律9：一般应用禁用malloc/new**

**10铁律1强化**

**10.1向NULL地址处copy数据**

* Strcpy(NULL, ”aaaa”);
* 向指定的内存空间copy数据，提示指定的值

**10.2不断地改变指针的指向**

**10.3野指针和内存首地址**

**10.4字面量**

**11铁律2强化**

**11.1技术推演**

**11.2间接赋值的工程意义**

* 函数调用时，实参传给形参，用实参取地址，传给形参，在被调用函数里面用\*p，来改 变实参，把运算结果传出来。
* 指针作为函数参数的精髓
* C语言特有的现象，是C语言的精华
* 主函数和被调用函数之间通过内存交换运算结果
* 函数参数=》接口的封装和设计=》模块的划分=》软件的分层

**11.3间接赋值成立的条件**

三个条件：

1. 定义一个变量（实参），一个变量（形参）
2. 建立关联，把实参取地址赋给形参
3. 形参去间接地修改实参的值

**11.4间接赋值的应用场景**

间接赋值条件应用深入分析

三个条件的组合，分别产生三种重要的语法现象

1）123都写在一个函数里面

2）12写在一个函数里，3写在另外一个函数里

3）1写在一个函数里，23写在另外一个函数里，

**11.5间接赋值推论**

在函数调用时：

用一级指针形参，去间接地修改0级指针（实参）的值

用二级指针形参，去间接地修改1级指针（实参）的值

用三级指针形参，去间接地修改2级指针（实参）的值

用n级指针形参，去间接地修改n-1级指针（实参）的值

**11.6寻路**

指针做函数参数使我们的研究重点

指针是子弹，函数是枪管，子弹枪管才能发挥它的威力

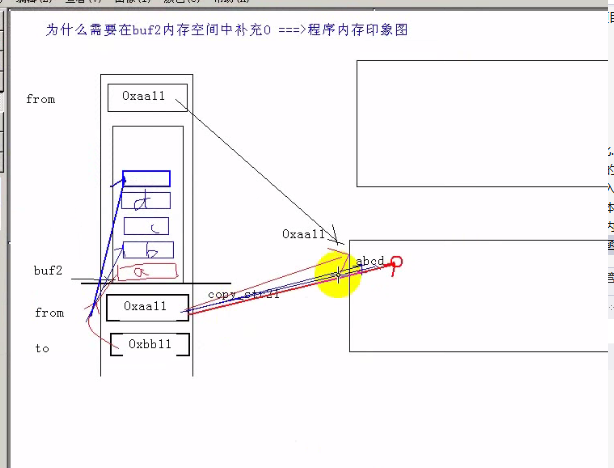
下一步方向：函数指针

**12铁律3强化**

**字符串专题**

**1字符串基本操作**

**2字符串做函数参数**



**3库函数api**

**4字符串相关一级指针内存模型**

**5项目开发字符串模型**

**5.1strstr-whiledowhile模型**

**5.2两头堵模型**

**5.3字符串反转模型**

**6一级指针(char \*)易错模型分析**

**6.1char \*（字符串）作函数参数出错模型分析**

**6.2越界**

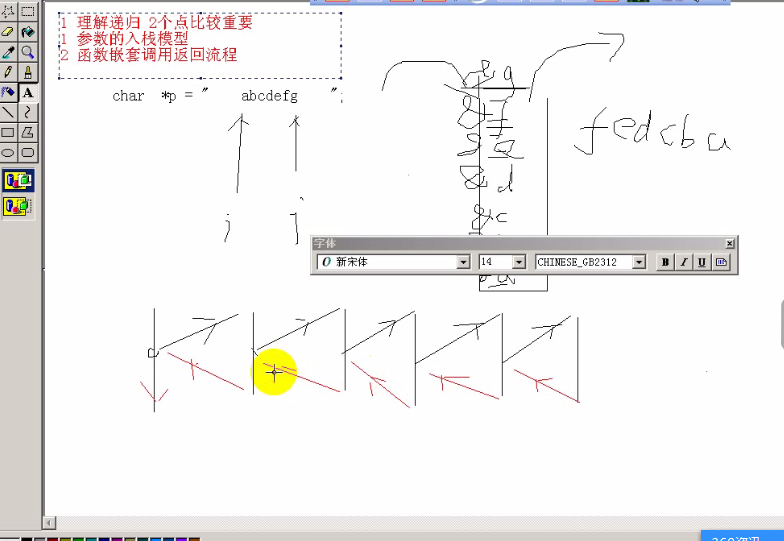
**6.3不断修改指针变量的值**

**6.4你向外界传递什么**

**6.5看图**

**6.6重复的错误何时休**

**6.7考试强化训练**



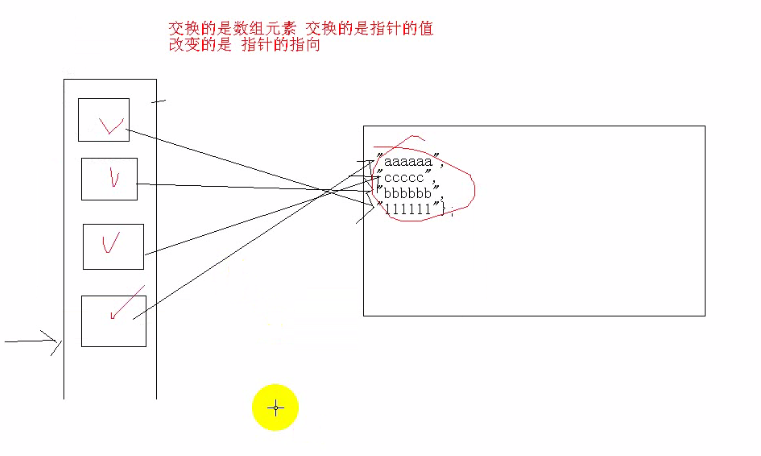
**7const专题**

**二级指针的三种内存模型**

**1.1二级指针输入和输出模型**

**1.1.1二级指针的三种内存模型**

**1.1.2二级指针的第一种内存模型**



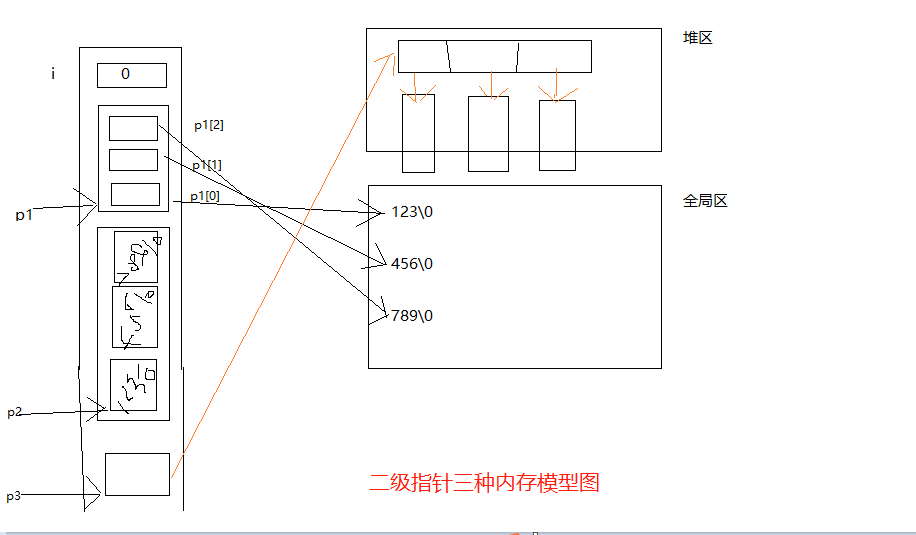
**1.1.3二级指针的第二种内存模型**

**1.1.4二级指针的第三种内存模型**

**1.1.5作业强化：两个辅助指针变量挖字符串**

**1.1.6思想分享：强化训练到极致**

**1.2二级指针内存模型建立**

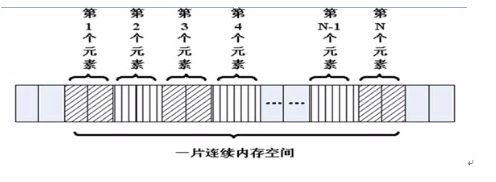


**数组类型和多维数组本质**

**1数组概念**

**1.1概念**

1. 元素类型角度：数组是相同类型的变量的有序集合 测试指针变量占有内存空间的大 小
2. 内存角度：连续的内存空间



**1.2数组初始化**

* 数组元素的个数可以显式或者隐式指定
* 分析数组初始化{0}和memset比较

**1.3数组名的技术盲点**

1. 数组首元素的地址和数组地址是两个不同的概念
2. 数组名代表数组首元素的地址，它是个常量
   * 1. 解析如下：变量本质是内存空间的别名，一旦定义数组，就分配内存，内存就固定了。所以数组名起名以后就不能被修改了
3. 数组首元素的地址和数组的地址值相等
4. 怎样得到整个一维数组的地址

C语言规定：

Int a[10];

Printf(“得到整个数组的地址：%d\n”, &a);

Printf(“数组首元素的地址：%d\n”, a);

怎样表达int a[10]这种数据类型

**2数组类型、数组指针类型、数组指针类型变量**

**2.1数组类型**

1. 数组类型分为基础、非基础，思考角度应该发生变化
2. C语言中的数组有自己特定的类型
   * 1. 数组的类型由元素类型和数组大小共同决定
     2. Eg: Int array[5]的类型为int[5]

/\*

Typedef int(MYINT5)[5];

Typedef float(MYFLOAT10)[10];

数组定义：

MYINT5 array; int array[5];

MYFLOAT10 array;

\*/

3）定义数组类型，并用数组类型定义变量

**2.2数组指针类型**

数组指针用于指向一个数组

Int a[10]

数组名是数组首元素的地址，但并不是数组的起始地址

通过将取地址符&作用于数组名可以得到整个数组的起始地址

定义数组指针，有两种方法：

1. 通过数组类型定义数组指针

Typedef int (ArrayType)[5];

Array Type\* pointer;

1. 声明一个数组指针类型

Typedef int (\*MyPointer)[5];

MyPointer myPoint;

1. 直接定义：Int (\*pointer)[n];
   1. Pointer 数组指针变量名
   2. Type 指向的数组的类型
   3. N 指向的数组的大小

**3多维数组本质技术推演**



**4多维数组作函数参数退化原因大剖析**

**5多维数组作函数参数技术推演**

1. C语言中只会以机械式的值拷贝的方式传递参数（实参把值传给形参）

Int fun(char a[20], size\_t b)

{

Printf(“%d\t%d”, b, sizeof(a));

}

原因1：高效

原因2：C语言处理a[n]时，它没有办法知道n是几，它只知道&n[0]是多少，它的值作 为参数传递进去了。虽然C语言可以做到直接int fun(char a[20])，然后函数能得到 20这个数字，但是，C没有这么做

1. 二维数组参数同样存在退化问题

二维数组可以看做是一维数组

二维数组中的每一个元素是一维数组

二维数组参数中第一维的参数可以省略

Void f(int a[5]) ===> void f(int a[]) ===> void f(int \*a);

Void g(int a[3][3]) ===> void g(int a[][3]) ===> void g(int (\*a)[3]);

1. 等价关系

数组参数 等效的指针参数

一维数组 char a[30] 指针 char \*

指针数组 char \*a[30] 指针的指针 char \*\*a

二维数组char a[10][30] 数组的指针char(\*a)[30]

**结构体专题**

1. **结构体类型定义及变量定义**
2. **结构体作函数参数及结构体数组**
3. **结构体在工程开发中的应用**

**文件操作专题**

**C语言文件读写概念**

**1文件分类**

* 按文件的逻辑结构
  + 记录文件：由具有一定结构的记录组成（定长和不定长）
  + 流式文件：由一个一个的字符（字节）数据顺序组成
* 按存储介质
  + 普通文件：存储介质文件（磁盘、磁带等）
  + 设备文件：非存储介质（键盘、显示器、打印机等）
* 按数据的组织形式
  + 文本文件：ASCII文件，每个字节存放一个字符的ASCII码
  + 二进制文件：数据按其在内存中的存储形式原样存放

|  |
| --- |
| ASCII形式  内存中存储形式 1 0 0 0 0    二进制形式 |

**文件操作API**

**1文件API的分类**

1. 文件读写API

Fgetc fputc 按照字符读写文件

Fgets fputs 按照行读写文件（读写配置文件）

Fread fwrite 按照块读写文件（大数据块迁移）

按照格式化进行读写文件

1. 文件控制API

文件是否结束

文件指针的定位、跳转

1. API做项目

**2标准文件的读写**

2.1文件的打开fopen

Fopen(char \*filename, char \*type);

方式 含义

R 只读方式打开

R+ 读写方式打开，指针指向文件头

Rb 只读方式打开二进制文件

Rb+ 读写方式打开二进制文件，指针指向文件头

W 只写方式打开，指针指向文件头，不存在则创建之

W+ 读写方式打开，指针指向文件头，不存在则创建之

Wb 只写方式打开二进制文件，指针指向文件头，不存在则创建之

Wb+ 读写方式打开二进制文件，指针指向文件头，不存在则创建之

A 只写方式打开，指针指向文件尾，不存在则创建之

A+ 读写方式打开，指针指向文件尾，不存在则创建之

Ab 只写方式打开二进制文件，指针指向文件尾，不存在则创建之

Ab+ 读写方式打开二进制文件，指针指向文件尾，不存在则创建之

**3非标准文件的读写**

**4注意点**

|  |
| --- |
| 文本文件：ASCII文件，每个字节存放一个字符的ASCII码  二进制文件：数据按其在内存中的存储形式原样存放 |
| 图片 |
|  |

**文件操作案例-配置文件读写**

实现分析：

1. 功能划分
   * 1. 界面测试（功能集成）

自己动手规划接口模型

* + 1. 配置文件读写
       1. 配置文件读（根据key，读取value）
       2. 配置文件写（输入key,value）
       3. 配置文件修改（输入key,value）
       4. 优化===》接口要求紧，模块要求松
    2. 实现及代码讲解
    3. 测试

**文件操作案例-大文件加解密**

**链表专题**

**概念**

链表是一种物理存储单元（内存）上非连续的存储结构，由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成，结点与结点之间通过指针链接，每个结点包括两个部分：一部分是存储数据元素的数据域，一个是存储下一个结点的指针域。

* 结构体
* 两个域：数据域和指针域
* 引用自身的结构体
* 非顺序存储

**分类**

* 带头链表&不带头链表
* 结点的组织形式
  + 单向链表、双向链表、循环链表等
* 静态链表&动态链表

**基本操作**

**链表编程元素分析**

* 头结点、当前结点、前驱结点、后继结点

pHead pCurrent pPre pNext

* 新建结点

pMalloc

**链表编程关键**

* 指针指向谁，就把谁的地址赋给指针
* 辅助指针变量&操作逻辑的关系