C语言技巧集锦

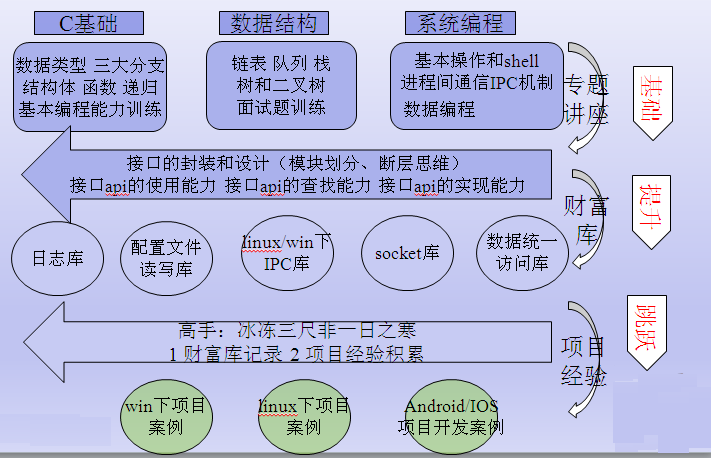
**参考资料：**

视频（C语言提高——王保明）

在开发过程中的一些技巧

结构体数据组包 （PDF中讲过）

# C语言技术模型分层



其中，接口的封装和设计尤为重要！

挑战\*p、\*\*p、\*\*\*p；

# 《内存专题》

当数组作为函数参数的时候会退化为指针，那样数组的大小就获取不了。

即此时的sizeof(pArray)=4为指针所占空间，而不是数组大小。

数组

eg：冒泡排序法

void arraySrot(int a[], int len)

{

int i=0;

int j=0;

int tmp;

for (i=0; i<len; i++)

{

for (j=i+1; j<len; j++)

{

if (a[i] > a[j])

{

tmp = a[i];

a[i] = a[j];

a[j] = tmp;

}

}

}

}

总结：函数调用的时候，把数组首地址和有效数据长度传给被调用函数才是正确的做法。

写在函数上的形参变量，与写在函数内的变量是没有任何区别的。只不过写在函数上的形参变量具有对外的属性而已。

**站在编译器的角度去写代码，思考问题，能很快提高水平。**

指针指向谁，就把谁的地址赋给指针。

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

//返回字符串的首地址

char \*getStr1(void)

{

char \*p = "abcd1";

return p;

}

char \*getStr2(void)

{

char \*p = "abcd2";

return p;

}

char \*getStr3(void)

{

char buf[100];

memset(buf, 0, sizeof(buf));

strcpy(buf, "abcd");

return buf;

}

int main(void)

{

int a = 0;

char \*p1 = NULL;

char \*p2 = NULL;

char \*p3 = NULL;

p3 = (char \*)malloc(100);

p1 = getStr1();

p2 = getStr2();

printf("getStr2=%s\n", getStr2()); //（1）

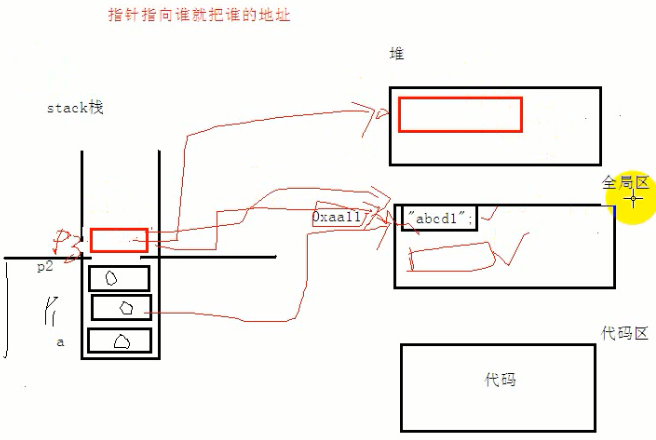
printf("p1=%d\n", p1); //（2）

printf("p2=%d\n", p2); //（3）

printf("getStr3=%s\n", getStr3()); //（4）

system("pause");

}



变量是一个标签，它的内容才是在内存中。

p1、p2、p3指针变量都是标签，其内容为地址，存放在栈区。其中，p1、p2的地址所指向的内容为字符串常量，存放在全局区，p3地址所指向的内容存放在堆区。

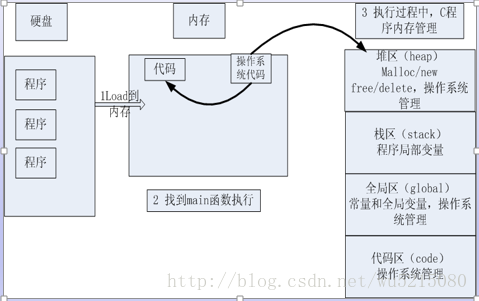
如果getStr1中p和getStr2中p指针指向的都是字符串”abcd1”，那么main函数中打印的p1、p2的地址值都是一样的，因为在编译器中，会优化认为全局区只有一个字符串”abcd1”，两个指针变量都指向这个字符串的首地址，当然其地址值也是一样的。

另外，getStr3中这种做法是错误的，在vs2015中，调试（debug）状态下(4)打印的是一串乱码，在release状态下，打印的是”abcd”，这对程序定位问题是一个极大的考验，函数内的栈区在函数退出后会销毁，一定注意避免该写法。

## 内存的四区模型

变量三要素：名称、大小、作用域；

函数三要素：名称、参数、返回值。



流程说明

1、应用程序肯定要保存到硬盘上，执行应用程序时（eg：双击.exe文件）操作系统把应用程序的代码从物理硬盘load到内存。其实，操作系统的源码也一样，在启动时将映像文件load到固定的内存区间内，提前入驻。

2、操作系统把c/c++代码分成四个区：堆区、栈区、全局区、代码区。

3、操作系统找到main函数入口执行

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）：一般由程序员手动分配释放（一定要释放），如果不释放，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，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。

程序代码区：存放函数体的二级制代码。

栈的特性：后进先出 LIFO

队列特性：先进先出 FIFO

一个主程序由n个函数组成，c++编译器会建立几个堆区，几个栈区？

如果这个程序是单进程的，c++编译器会建立一个堆区，一个栈区。多进程，另当别论。

**没有内存，哪里有指针啊！**

//指针变量和它所执行的内存空间变量是两个不同的概念

//在被调用函数里面malloc的内存，首地址传给别调用哪个函数有两种方法

//1 return

//2 指针做函数参数

//主调用函数分配的内存（堆 、 stack、全局区），可以在被调用函数里面使用（指针做函数参数）

//被调用函数分配的内存，malloc的内存、static内存可以让主调用函数使用，但是临时区stack属性的内存不可以

//内存泄露问题需要注意

## 程序的内存模型

建立正确的**程序运行内存布局图**是学好C语言的关键。

深入理解数据类型和变量“内存”属性；

一级指针内存布局图(int \*, char \*);

二级指针内存布局图(int \*, char \*）。

测试堆栈属性（地址高到低还低到高）？

void main()

{

int a;

int b;

printf(“&a:%d &b:%d\n”, &a, &b);

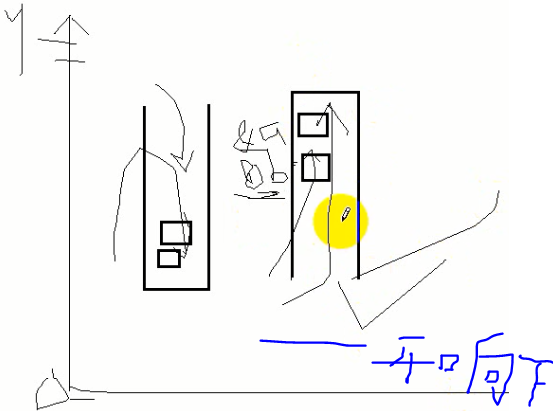
system(“pause”);

}

运行结果：

&a:14023496 &b: 14023492

a先定义，b后定义，则说明a先入栈，b后入栈，栈是先进后出。运行结果显示a地址比b地址大，地址是由低到高的，说明栈是开口向下的。

 y轴是低地址到高地址。

**堆和栈的生长方向：**

栈向下生长，即向小地址方向生长；

堆向上增长，即向大地址方向；

其间剩余部分是自由空间。使用过程中要防止增长过度而导致覆盖。

注：堆栈的生长方向和内存的存放方向是两个不同的概念，不管在堆中，还是栈中，内存的存放方向都是地址到高地址（当然，这是小端模式的内存，大端模式则不同）。

# 指针铁律专题

## 铁律1：指针是一种数据类型

1. 指针也是一种变量，占有内存空间，用来保存内存地址

测试指针变量占有内存空间大小

2）\*p操作内存

在指针声明时，\*号表示所声明的变量为指针

在指针使用时，\*号表示 操作 指针所指向的内存空间中的值

\*p相当于通过地址(p变量的值)找到一块内存；然后操作内存

\*p放在等号的左边赋值（给内存赋值） 写

\*p放在等号的右边取值（从内存获取值） 读

3）指针变量和它指向的内存块是两个不同的概念

//含义1 给p赋值p=0x1111; 只会改变指针变量值，不会改变所指的内容；p = p +1; //p++

//含义2 给\*p赋值\*p='a'; 不会改变指针变量的值，只会改变所指的内存块的值

//含义3 =左边\*p 表示 给内存赋值， =右边\*p 表示取值 含义不同切结！

//含义4 =左边char \*p

//含义5 保证所指的内存块能修改

4）指针也是一种数据类型，是指它指向的内存空间的数据类型

含义1：指针步长（p++），根据所致内存空间的数据类型来确定

p++=🡺(unsigned char )p+sizeof(a);

结论：指针的步长，根据所指内存空间类型来定。

有的p+1移动1个字节，有的移动40个字节，那是因为该指针所指向的内存空间的数据类型不同导致的，即步长不同。

5）不管是1个\*，还是8个\*，对C++编译器来说（对32位系统来说），只会分配4个字节。

int arr[10];

C规定，数组名代表数组首元素的地址 arr + 1只会跳一个元素 ，步长为1个元素

C规定，&arr则代表整个数组 arr + 1只会跳整个数组，步长为整个数组长度

\*(NULL) = 0; //错误，因为地址0是操作系统保护的一块内存区域，不允许随便写入。NULL即地址0x00000000。

## 铁律2：通过\*p/\*p++ 来改变变量的值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块变量

2）条件反射：指针指向某个变量，就是把某个变量地址否给指针

3）\*p间接赋值成立条件：3个条件

a)2个变量（通常一个实参，一个形参）

b) 建立关系，实参取地址赋给形参指针

c) \*p形参去间接修改实参的值

|  |
| --- |
| Int iNum = 0; //实参  int \*p = NULL;  p = &iNum;  iNum = 1;  \*p =2 ; //通过\*形参 == 间接地改变实参的值  \*p成立的三个条件： |
|  |

**4）引申： 函数调用时,用n指针（形参）改变n-1指针（实参）的值。**

//改变0级指针（int iNum = 1）的值有2种方式

//改变1级指针（eg char \*p = 0x1111 ）的值，有2种方式

//改变2级指针的（eg char \*\*pp1 = 0x1111 ）的值，有2种方式

//函数调用时，形参传给实参，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。

|  |
| --- |
| 间接赋值的应用场景分析 |
| //C++多态实现的技术推演  //AOP切面编程 （java 2-3），3  //1定义一个变量（通常是实参） 定义一个变量（通常是形参）  //2建立联系 把实参取地址取地址传给形参  //3\*p形参，就间接地修改了是实参的值。  //\*p = 10;  //间接赋值应用场景概念建立  //1 在一个函数里面  //\*p1++ = \*p2++ //完成内存块的copy  //2 在两个函数之间，指针做函数参数把运算结果给甩出来  //123  //12 3  //1 23 |
| 间接赋值的推论=====🡺指针做函数参数 |
| //在函数调用的时候  //用1级指针（形参）去改变了0级指针（实参）的值 （通过\*p去间接修改了实参的在值）  //用2级指针（形参）去改变了1级指针（实参）的值 （通过\*p去间接修改了实参的在值）  //用3级指针（形参）去改变了2级指针（实参）的值 （通过\*p去间接修改了实参的在值）  //用n级指针（形参）去改变了n-1级指针（实参）的值 （通过\*p去间接修改了实参的在值） |
| 形象的比如: |
| 子弹沿着枪管发射，才有威力！  指针就像子弹,函数就像枪管，指针沿着枪管发射出去，才能显示它的威力。。。。。。  我们的研究方向。。。。。。（你明白了?） |

# 字符串专题讲座

## 字符串操作基础

#include "stdlib.h"

#include "stdio.h"

#include "string.h"

//int \* char \*

//c语言里面没有字符串这种类型。。。。。

//通过字符数组来模拟字符串

//C风格字符串是以零结尾的字符串

//

void main11()

{

//字符数组初始化

//指定长度 如果定义的长度剩余部分补充0

char buf1[100] = {'a', 'b', 'c'};

//不指定长度

char buf2[] = {'a', 'b', 'c'};

char buf3[] = {'a', 'b', 'c','\0'};

//通过字符串初始化字符数组 并且追加\0

char buf4[] = "abcdefg";

printf("%s\n", buf4 );

system("pause");

}

//sizeof

void main12()

{

//字符数组初始化

//指定长度 如果定义的长度剩余部分补充0

char buf1[100] = {'a', 'b', 'c'};

//不指定长度

char buf2[] = {'a', 'b', 'c'};

char buf3[] = {'a', 'b', 'c','\0'};

//通过字符串初始化字符数组 并且追加\0

char buf4[] = "abcd";

printf("%s\n", buf4 );

printf("sizeof(buf4): %d\n ", sizeof(buf4)); //注意sizeof是对数组类型进行大小测量 包括了\0

printf("strlen(buf4): %d \n", strlen(buf4));//strlen是求字符串的长度不包括\0

system("pause");

}

//操作数组的方法

//下标法和指针法

void main()

{

int i = 0;

char \*p = NULL;

//通过字符串初始化字符数组 并且追加\0

char buf4[] = "abcd";

for (i=0; i<strlen(buf4); i++)

{

printf("%c", buf4[i]); //p[]

}

//[] \*的本质到底是什么？

//\*p 是我们程序员手工的（显示）去利用间接赋值

//【】 只不过是，c/c++ 编译器帮我们做了一个\*p的操作。。。。。。

// buf4[i]======> buf4[0+i] ====> \*(buf4+i)

//===\*(buf4+i) --> bu4[i];

printf("\n");

p = buf4;

for (i=0; i<strlen(buf4); i++)

{

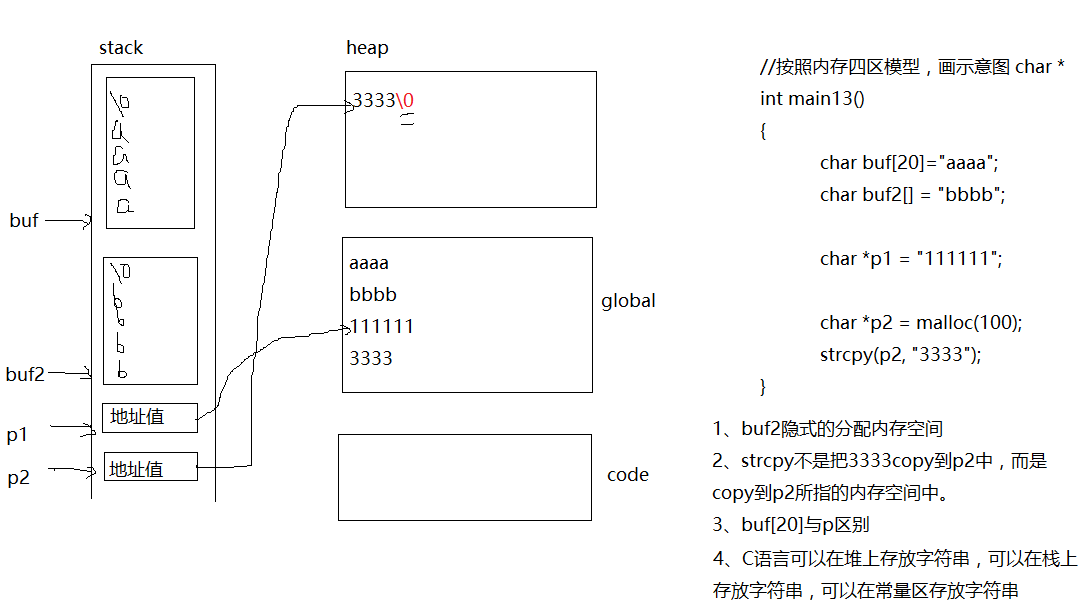
printf("%c", \*(p+i)); //\*p

}

system("pause");

}

## 字符串内存模型



上图中buf的首地址和字符串”aaaa”的首地址没有任何关系。

#include "stdlib.h"

#include "stdio.h"

#include "string.h"

//自定义：字符串copy函数，完成字符串from ，到to的copy

void copy\_str1(char \*from, char \*to)

{

for (; \*from!='\0'; from++, to++)

{

\*to = \*from;

}

\*to = '\0';

}

void copy\_str2(char \*from, char \*to)

{

while(\*from!='\0')

{

\*to = \*from;

from++;

to++;

}

\*to = '\0';

}

//++优先级高,但是这个++是后缀++

//所以先执行\*to = \*from; 再 from++; to ++from++;

void copy\_str3(char \*from, char \*to)

{

while(\*from!='\0')

{

\*to ++ = \*from ++;

}

\*to = '\0';

}

void copy\_str4\_1(char \*from, char \*to)

{

while((\*to = \*from) !='\0') //最后一个\0也先赋值进去了，再判断 == ‘\0’，退出

{

to++;

from++;

}

}

void copy\_str4\_2(char \*from, char \*to)

{

while((\*to++ = \*from++) !='\0')

{

;

}

}

void copy\_str5(char \*from, char \*to) //最好

{

while((\*to++ = \*from++))

{

;

}

}

void main()

{

char \*p = "abcdefg";

//char \*p2 = NULL;

char p2[100] ;

copy\_str5(p, p2);

//strcpy(p2, "abcdeeg");

printf("p2:%s\n", p2);

system("pause");

}

# 经典例程

## 1、字符串拷贝

void StrCopy(char \*from, char \*to)

{

while((\*to++ = \*from++))

{

;

}

}

2、字符串查找

查找字符串中含有特定的字符串的个数

int StrFind(char \*pSrcStr, char \*pDstStr)

{

int iCount = 0;

char \*pTmp = NULL;

do

{

pTmp = strstr(pSrcStr, pDstStr);

if (pTmp == NULL)

{

break;

}

else

{

count++;

pTmp += strlen(pDstStr);

}

} while (\*pTmp != ‘\0’)

return iCount;

}