指针、结构体进阶学习

C语言面向过程，基本组成单位是函数。

1 指针

1.1 指针概念

可以通过地址找到内存单元，因此将地址形象的称为指针，指向内存单元。内存单元地址编号是由电脑的地址线（32位）决定，存放地址的变量就是指针变量。

1.2 指针和指针类型

指针指向变量的地址，指向的是一个变量或数组等的首地址（第一个字节的地址），而指针变量的类型由被指向的变量的类型决定（因为指针类型尤其意义），如被指向的变量为int型，则指针变量的类型为（int \*）。一个指针的大小即地址编号的大小，如32位电脑指针的大小始终为32位，即四个字节。

虽然指针大小始终不变，但指针类型也有其存在的意义：

意义1：指针类型决定了：指针解引用的权限有多大，如int权限4字节，而char指针权限只有1个字节。

意义2：指针类型决定了：指针走一步的偏移量（即步长），如int步长为4字节，而char指针步长只有1个字节。（字符串、整型数组等）

**例1：**

#include<stdio.h>

int main()

{

int a = 0x11223344;//四字节44 33 22 11从低位到高位存储

//int\* p = &a;//指针类型对应变量类型

//\*p = 0;//改变变量的所有字节值00 00 00 00

char\* p = &a;//指针类型与变量类型不一致 以指针类型决定改变的字节数

\*p = 0;//字符指针指向首地址，改变一个字节低位00 33 22 11

return 0;

}

**例2：**

#include<stdio.h>

int main()

{

int arr[10] = { 0 };

int\* p = arr;

char\* q = arr;

printf("%p\n", p);

printf("%p\n", p+1);//指针类型决定了指针移动的步长4字节

printf("%p\n", q);

printf("%p\n", q+1);//1字节

return 0;

}

1.3 野指针

野指针就是指针指向的位置是不可知的（随机的、不正确的、没有明确限制的）

野指针的成因：

（1）指针未初始化，所以指针可能为随机值。

（2）指针越界访问，如超过数组访问区域。

（3）指针指向的空间释放了。

避免野指针的出现：

（1）指针初始化（编程习惯），当不知道指针初值时，可以使用NULL。

（2）小心指针越界

（3）指针指向空间被释放即置NULL

（4）指针使用之前检查有效性

1.4 指针运算

（1）指针+ -去一个整数：p + 2;走步长。

（2）指针关系运算，指针比较大小，地址按低到高排列，则指针也有大小的关系。p<&a[5]

#include<stdio.h>

int main()

{

int arr[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int\* p = arr;

int\* p\_end = arr + 9;

while (p <= p\_end)

{

printf("%d ", \*p);

p++;

}

return 0;

}

（3）指针-指针

得到的是两个指针之间的元素数。但需要前提条件：两个指针指向同一块空间。

**例：**求字符串长度（利用指针减指针方式）

#include<stdio.h>

int main()

{

char str[20] = { 0 };

char\* p = str;

char\* q = str;

int len = 0;

scanf("%s", str);

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

{

p++;

}

len = p - q;

printf("%d\n", len);

return 0;

}

1.5 指针和数组

指针p+i就为数组下标为i的元素的地址。arr[i]==\*(p+i)==p[2]==i[arr]，[]是数组下标引用操作符，两个操作数可交换，i和arr可交换。

1.6 二级指针

#include<stdio.h>

int main()

{

int a = 10;

int\* pa = &a;//pa指针变量指向a的地址

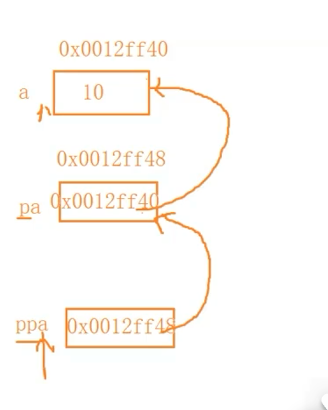
int\*\* ppa = &pa;//pa也为变量，也有属于自己的地址，且它的类型为int\*

//二级指针ppa 靠近ppa的\*说明其为指针，前面int\*是其指向的变量的类型

printf("%d\n", \*(\*ppa));//二级指针解引用访问到原始变量

return 0;

}//二级指针的关系



1.7 指针数组

指针数组，本质上是一个数组，存放指针的数组。

#include<stdio.h>

int main()

{

int arr[10];//整型数组

char str[20];//字符数组

int\* parr[5];//整型指针数组

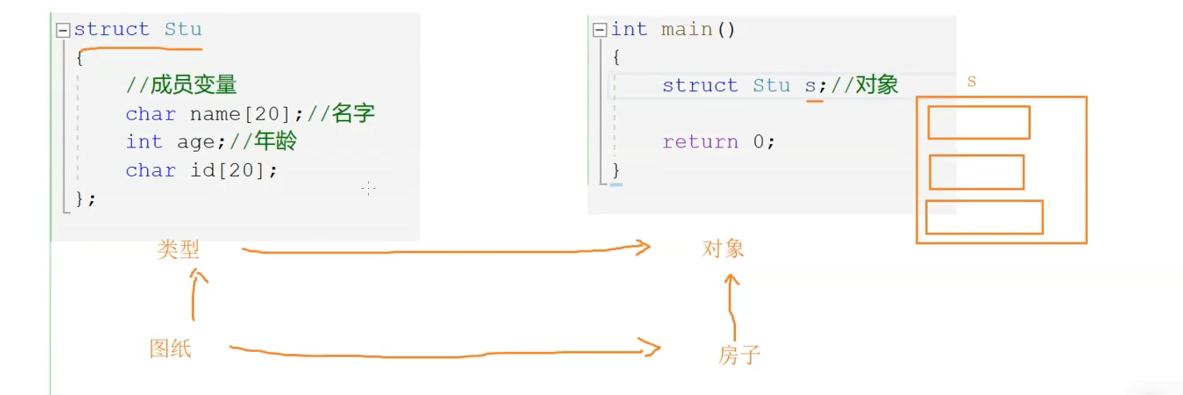
return 0;

}

2 结构体

2.1 结构体的声明

类型->对象->成员（有不同的定义和赋值方式）



成员变量的类型可以是所有类型。

2.2 结构体的初始化（见前）

2.3 结构体成员的访问（.操作符,->操作符）

2.4 结构体传参（函数传参相同，类型相同）

**例：**

#include<stdio.h>

struct score

{

float Chinese;

float math;

float English;

};

struct Stu

{

struct score obj;

char name[10];

int age;

char id[20];

};

void print(struct Stu\* ps)

{

printf("姓名：%s 学号：%s 年龄：%d\n", (\*ps).name, (\*ps).id, (\*ps).age);

printf("语文：%f 数学：%f 英语：%f\n", (ps->obj).Chinese, (ps->obj).math, (ps->obj).English);

printf("平均成绩为：%f\n", ((ps->obj).Chinese + (ps->obj).math + (ps->obj).English) / 3);

}

int main()

{

struct Stu s1 = { {92.5, 88.5, 70}, "王三", 16, "S21231321" };

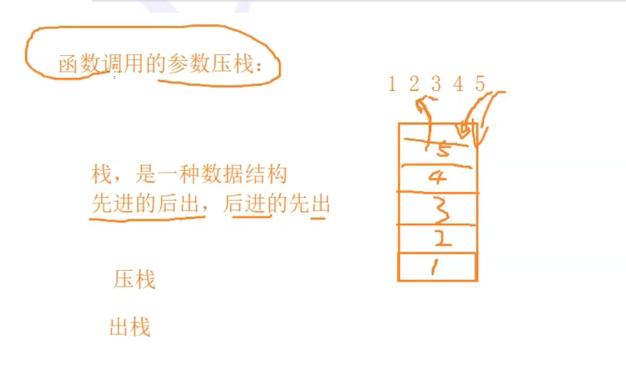
//写一个函数打印s1的值

print(&s1);//也可以不用指针，不需要改变实参。指针少占有空间和时间

return 0;

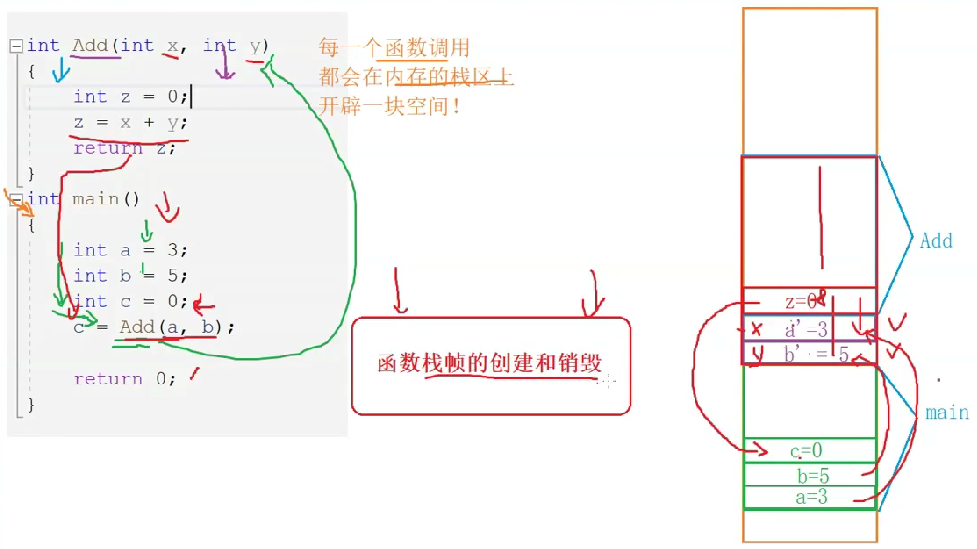
}

结构体传参尽量使用地址传参，节省空间和时间，因为结构体通常占有的空间较大，直接传值调用会导致时间和空间的浪费。



栈是一种数据结构，即数据的存取方式。栈的特点：先进的后出，后进的先出，有两种操作，压栈（进栈）和出栈。

函数的的运行及参数的调用在内存区域的规划：



3 实用调试技巧

3.1 什么是bug

计算机硬件或软件存在的一些缺陷，bug，小故事。

3.2 调试是什么？有多重要？

调试就是寻找bug的过程，有迹可循，侦探。尝试的方式不是可取的调试方式，而应该进行科学的调试。

发现问题（结果不理想）-定位问题大致位置-确定错误产生的原因-应对办法-改正并进一步测试。

发现问题的对象：程序员自己、软件测试人员、用户。

3.3 debug和release的介绍

Debug调试版本-程序员调试exe，版本大一点

Release发布版本-完成后包装成用户软件exe（不能调试了，进行最大限度的优化了）-测试人员 已经做过部分优化了。

Linux调试环境为gdb，后面学习。

Windows调试：预期的结果-调试的结果

（1）F5启动调试（fn+F5）

启动调试，通常用来直接跳到断点处。

（2）F9创建和取消断点与F5搭配使用

可以在程序任意位置设置断点，使得程序可以在任意位置启动和停止执行，进而完成整个代码得调试。快速跳动到断点处。

（3）F10逐语句运行

根据语句一步一步运行，可以进入到函数内部。

（4）F11逐过程运行

根据过程一个一个过程运行，函数也属于一个过程，直接运行跳过函数。

（5）调试-窗口-内存查看地址，也可看值

（6）调试-窗口-监视查看变量的值或者空间的值

（7）ctrl+F5直接执行不调试

调试-窗口-断点-右击条件-设置条件断点。

调试-窗口-监视-自动窗口

调试-窗口-局部变量

调试-窗口-反汇编

调试-窗口-寄存器

调试-窗口-调堆栈（反映的是函数调用的逻辑，压栈和退栈）

3.4 一些调试的实例（Windows环境调试）

**例1：**求1!+2!+3!+4!+…+n!的结果

#include<stdio.h>

int main()

{

int n = 0;

int i = 0;

int j = 0;

int ret = 1;

int sum = 0;

scanf("%d", &n);

for (i = 1; i <= n; i++)

{

//ret = 1;

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

{

ret \*= j;//ret没有清零

}

sum += ret;

}

printf("%d\n", sum);

return 0;

}

**例2：**一个错误的程序应该运行的结果是什么，为什么？（调试）

#include<stdio.h>

int main()

{

int i = 0;

int arr[10] = { 1,2,3,4,5,6,7,8,9,0 };

for (i = 0; i <= 12; i++)

{

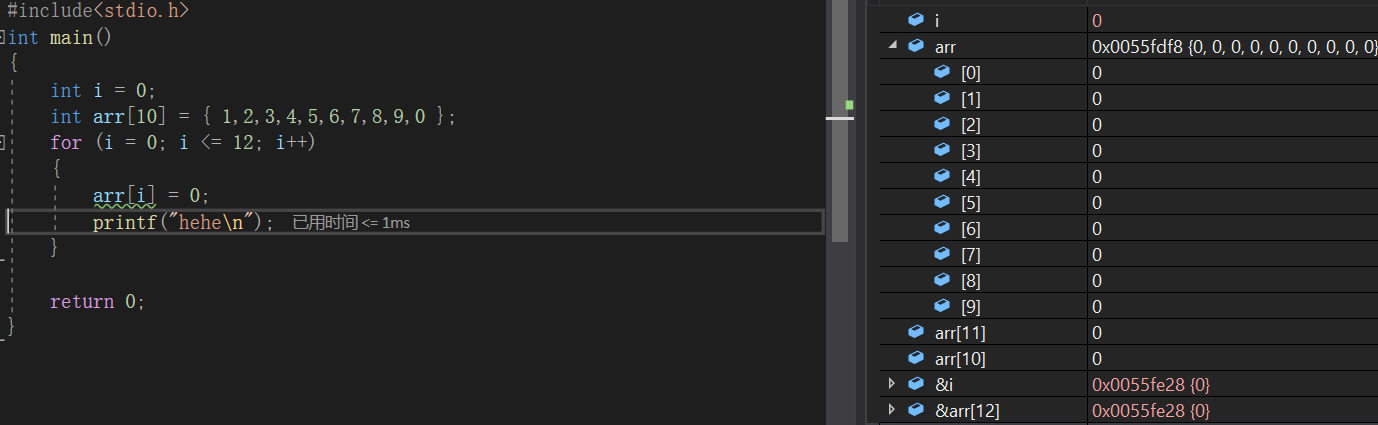
arr[i] = 0;//当i=12时，i被赋值为0.

printf("hehe\n");

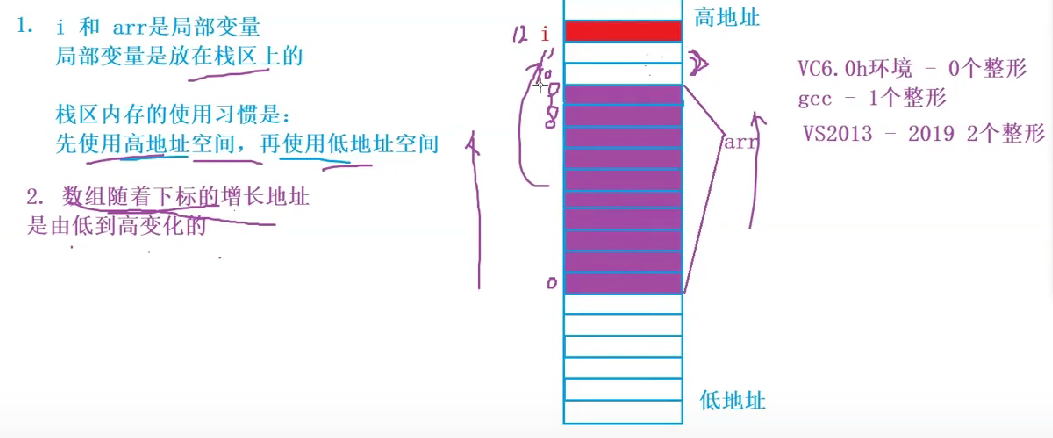
}

return 0;

}



可以看到，&i与&arr[12]相同，所以当i=12时，arr[12]和i都被赋值为0.出现这样的情况，即使巧合也存在一些必然因素（注：栈区与数据结构的堆栈并非同一概念）：《C陷阱和缺陷》《明解C语言》《C和指针》



大工程设置断点慢慢调试。

3.5 如何写出好的（易于调试）代码

好的代码：运行正常、bug少、效率高、可读性高、可维护性高，注释清晰、文档齐全。

常见写代码技巧：

（1）使用assert断言

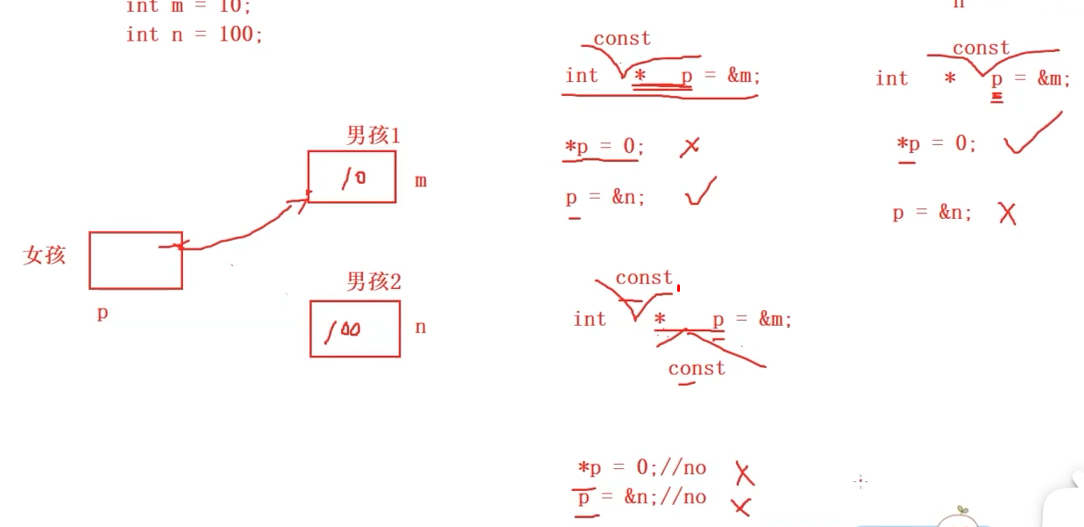
Assert在表达式计算结果为false(0)时打印诊断消息，并调用abort来终止程序执行。如果表达式为真(非零)，则不采取任何操作。诊断消息包括失败的表达式、断言失败的源文件的名称和行号。（常用在有效性检验位置）

（2）尽量使用const

Const值不能被修改的常变量，修饰变量时，使得变量被赋值时报错，防止代码书写时赋值方向错误。

Const修饰指针时：当const放在\*号左边时，修饰\*p（表示指针p指向的内容不能被改变）；当const放在\*号右边时，修饰p（表示指针p不能被改变）；当\*号的左右都放const时，则指针和指针所指向的值都不能被改变。

**例：**模拟实现库函数strcpy的功能。



#include<stdio.h>

#include<assert.h>

//模拟实现strcpy库函数的功能，strcpy(str2, str1);将字符串1的内容复制到字符串2，且结果为字符串2

//char\* my\_strcpy(char str2[], char str1[])

//{

// int i = 0;

// while (str1[i] != '\0')

// {

// str2[i] = str1[i];

// i++;

// }

// str2[i] = str1[i];//'\0'库函数的实现逻辑，可以看看调试过程，可以发现结束标志也被拷贝进去了

// return str2;

//}//有重复语句，值得优化

char\* my\_strcpy(char\* dest, const char\* src)//const作用防止赋值出错

{

//有效性检验

assert(src != NULL);//如果为真，则什么也不发生；如果为假，则打印一个警告信息。

assert(dest != NULL);//如果为真，则什么也不发生；如果为假，则打印一个警告信息。

char\* ret = dest;//保存dest的首地址方便后面返回

while (\*dest++ = \*src++)//'\0'的ASCII码为0跳出，既实现了完全拷贝，又实现了循环跳出

{

;

}

return ret;

}

int main()

{

char str1[10] = { 0 };

char str2[] = "\*\*\*\*\*\*\*\*\*\*\*";

scanf("%s", str1);

printf("str1=%s\n", str1);

printf("str2=%s\n", my\_strcpy(str2, str1));

return 0;

}

（3）良好的编程风格

（4）添加必要的注释

（5）避免代码陷阱

**例：**实现库函数strlen。

#include<stdio.h>

#include<assert.h>

unsigned int my\_strlen(const char\* p)

{

const char\* q = p;

assert(p != NULL);

while (\*p++ != '\0')

{

;

}

return p – q - 1;//p最终指向’\0’ 要减一

}

int main()

{

char str[20] = { 0 };

unsigned int lenth = 0;

scanf("%s", str);

lenth = my\_strlen(str);

printf("%d\n", lenth);

return 0;

}

3.6 编程常见的错误

（1）编译型错误：程序错误，会有提示。

（2）链接型错误：可能存在未定义或写错的符号，或头文件没引。

（3）运行结果错误：逐步调试定位改正。

**作业：**编写实现：两个int型数的二进制中，不同位的个数。例1999和2299不同的位个数为7.（如果负数以补码形式表示也是可以的）

#include<stdio.h>

int diff\_bit(int a, int b)

{

//int i = 0;

int n = 0;

int count = 0;

//方法二

n = a ^ b;

while (n)

{

n = n & (n - 1);

count++;

}

////方法一

//for (i = 0; i < 32; i++)

//{

// if (((a & 1) ^ (b & 1)) == 1)

// {

// count++;

// }

// a >>= 1;

// b >>= 1;

//}

return count;

}

int main()

{

int a = 0;

int b = 0;

scanf("%d%d", &a, &b);

printf("count = %d\n", diff\_bit(a, b));

return 0;

}

**例：**判断一个整数二进制位中1的个数

方法一：

#include<stdio.h>

int bit(int a)

{

int i = 0;

int count = 0;

for (i = 0; i < 32; i++)

{

if (((a >> i) & 1) == 1)

{

count++;

}

}

return count;

}

int main()

{

int a = 0;

scanf("%d", &a );

printf("count = %d\n", bit(a));

return 0;

}

方法二：

int bit(int a)

{

int count = 0;

while (a)

{

a = a & (a - 1);//与自身减一与，每次少一个“1”位，直到a变为0

count++;

}

return count;

}

**例：**判断一个整数是2的n次方。分析：2的n次方表明整数的二进制位中只有一个1，即可以运用一次n&(n-1)==0即表示n的二进制位中只有一个1。

#include<stdio.h>

int bit(int a)

{

if ((a & (a - 1)) == 0)//表明a中只有一个位为1，则说明a为2的次方

return 1;

else

return 0;

}

int main()

{

int a = 0;

scanf("%d", &a );

printf("a是否为2的n次方（0/1）: %d\n", bit(a));

return 0;

}

**初阶考试：**

1.读程序

#include<stdio.h>

int cnt = 0;

int fib(int n)

{

cnt++;

if (n == 0)

return 1;

else if (n == 1)

return 2;

else

return fib(n - 1) + fib(n - 2);

}

int main()

{

fib(8);

printf("%d\n", cnt);//67

return 0;

}

2.写程序：两个正整数的最小公倍数

#include<stdio.h>

int main()

{

int n = 0;

int m = 0;

int max = 0;

int i = 0;

scanf("%d %d", &n, &m);

if (n > m)

max = n;

else

max = m;

i = max;

while (1)

{

if ((i % n == 0) && (i % m == 0))

break;

i++;

}

printf("%d\n", i);

return 0;

}

3.写程序：逆序语句，I like you. -> you. like I

三步翻转法：1先整体翻转 2单词翻转找空格 3翻转使用同一函数。

#include<stdio.h>

#include<string.h>

void swap\_str(char\* left, char\* right)

{

char temp = 0;

while (left < right)

{

temp = \*left;

\*left = \*right;

\*right = temp;

left++;

right--;

}

}

int main()

{

char str[100] = { 0 };

gets(str);

int len = strlen(str);

char\* edge = str + len - 1;

//3步翻转法

//1.先整体翻转

swap\_str(str, edge);

//2.每个单词翻转 3.空格为标志，调用同一函数

char\* start = str;

char\* start1 = str;

char\* end = start;

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

{

if (\*start == ' ' || \*start == '\0')

{

end = start - 1;//每个单词的结束位置

swap\_str(start1, end);//翻转每个单词

start1 = end + 2;//记录每个单词的开始位置

}

start++;

}

//结束之后再翻转一次'\0'

end = start - 1;//每个单词的结束位置

swap\_str(start1, end);//翻转每个单词

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

return 0;

}