项目经理带你-零基础学习 C/C++
【从入门到精通】

项目八 人工智能之双色球预测系统

第 1 节 项目需求

中国福利彩票“双色球”是一种联合发行的“乐透型”福利彩票。采用计算机网络系统发行销售，定期电视开奖。

游戏规则

1.“双色球”彩票投注区分为红色球号码区和蓝色球号码区。

2.“双色球”每注投注号码由 6 个红色球号码和 1 个蓝色球号码组成。红色球号码从 1--33 中选择；蓝色球号码从 1--16 中选择。

3.“双色球”每注 2 元。

项目需求

双色球出奖结果是随机的，但现实生活中有个很神奇的概念叫“**概率**”，并且偶尔会出现规律,让你和百万大奖擦肩！
比如：如果某个号子历史出现的机率高，那它出现的机率就相应会高！

期号 排序	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
19037	1	1	6	5	2	10	7	5	2	1	3	12	1	14	3	1	8	18	4	2	4	3	1	3	25	3	11	5	4	9	14	1
19038	1	2	7	6	3	11	1	6	9	2	4	12	2	1	4	2	9	1	5	3	21	4	2	4	1	4	27	6	29	30	15	2
19039	2	3	8	7	4	6	7	7	1	3	11	1	3	14	5	3	10	2	6	4	1	5	3	5	2	5	27	7	1	1	16	32
19040	3	4	9	8	5	6	1	8	9	4	1	2	4	1	6	4	11	18	7	5	2	6	23	6	3	6	1	8	2	2	31	1
19041	4	2	10	9	1	1	2	9	9	5	2	3	13	2	7	5	12	1	8	6	3	7	23	24	4	26	2	9	3	3	1	2
19042	5	1	11	10	2	2	3	10	1	6	3	4	1	3	15	6	17	2	19	7	4	22	1	1	25	26	3	10	4	4	2	3
19043	1	2	12	11	3	6	4	11	2	7	4	12	13	4	1	7	1	3	1	8	5	1	2	24	1	1	4	11	5	5	3	32
19044	1	3	13	12	4	6	5	12	3	8	5	1	1	14	2	16	17	4	2	9	6	2	23	1	2	2	5	12	29	6	4	1
19045	1	4	14	13	5	6	6	13	4	9	6	2	2	1	3	1	17	5	19	10	7	3	1	2	3	3	27	13	1	7	31	2
19046	1	2	15	14	6	1	7	14	5	10	7	12	3	2	4	16	1	6	1	11	8	22	2	3	25	4	1	14	2	8	1	32
19047	2	1	3	15	7	2	8	15	6	11	11	1	4	3	5	1	2	18	2	12	9	1	3	4	25	5	2	15	3	30	2	1
19048	3	2	3	16	8	3	7	16	7	10	1	12	5	4	6	2	3	18	3	13	10	2	4	5	1	6	3	16	29	1	3	2
19049	4	3	3	17	9	4	1	17	8	10	2	1	13	5	7	3	4	1	4	14	11	22	23	6	2	7	4	28	1	2	4	3
19050	5	4	1	4	10	6	2	18	9	10	11	2	1	6	8	4	5	2	5	15	21	1	23	7	3	8	5	1	2	3	5	4
19051	6	5	2	1	11	1	3	8	9	10	1	3	13	7	15	5	6	3	6	16	1	2	1	8	4	9	6	28	3	4	6	5
19052	7	6	3	2	12	6	4	1	9	1	2	4	13	8	1	16	7	4	19	17	2	3	2	9	5	10	7	1	4	5	7	6
19053	8	7	1	4	13	1	5	2	1	2	3	5	1	9	2	16	8	5	1	18	3	22	3	10	25	11	8	2	29	6	31	7
19054	9	8	2	1	14	2	7	3	2	10	11	6	2	10	15	1	9	6	2	19	4	1	4	24	1	26	9	3	1	7	1	8
19055	1	9	3	2	15	6	1	4	3	1	11	7	3	11	15	2	10	7	19	20	5	2	5	1	2	1	10	4	2	8	31	9
19056	1	10	4	3	16	1	2	5	4	2	1	8	13	14	1	3	17	8	19	21	21	3	6	2	3	2	11	5	29	9	1	10
19057	2	11	5	4	5	6	3	8	5	3	2	9	13	1	2	4	1	18	1	22	1	4	7	3	4	3	12	6	1	10	2	11
19058	3	12	6	1	1	1	7	8	6	4	3	12	1	2	3	5	2	1	2	23	21	5	23	4	5	4	27	7	2	11	3	12
出现总次数	4	2	4	3	2	9	5	3	5	5	5	6	7	4	4	4	4	5	5	0	4	4	6	3	5	3	4	2	5	2	4	3
平均遗漏值	5	11	5	7	11	2	4	7	4	4	4	3	3	5	5	5	5	4	4	23	5	5	3	7	4	7	5	11	4	11	5	7
最大遗漏值	9	12	15	17	16	11	8	18	9	11	7	9	5	11	8	7	12	8	8	23	11	7	7	10	5	11	12	16	5	11	16	12
最大连出值	1	1	3	1	1	3	1	1	2	4	2	2	2	1	2	2	2	2	2	1	1	1	2	1	2	2	2	1	1	1	1	1
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

现在要求我们编写程序找出最近一段时间每个号码出现的次数并把结果保存到一个数组，供

其它分析模块调用

项目精讲

1. 为什么要使用指针

- ✓ 函数的值传递，无法通过调用函数，来修改函数的实参
- ✓ 被调用函数需要提供更多的“返回值”给调用函数
- ✓ 减少值传递时带来的额外开销，提高代码执行效率

```
// demo 8-1.c
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

//1.函数的值传递，无法通过调用函数，来修改函数的实参
void add_blood(int blood){
        blood +=1000;
}



//2.  被调用函数需要提供更多的“返回值”给调用函数

//加血成功返回 ture ,否则返回 false ;
bool add_blood2(int blood){
    if(blood >= 1000) {
        return false;
    }else {
        blood +=1000;
    }
    return true;
    //另外还需要返回最新的血量，不能实现

}


//3.  减少值传递时带来的额外开销，提高代码执行效率

struct _hero_stat{
    int blood;   //英雄的血量
    int power;          //英雄攻击力
    int level;          //英雄级别
    char name[64];      //英雄名字
    char details[1024];//状态描述
};
```

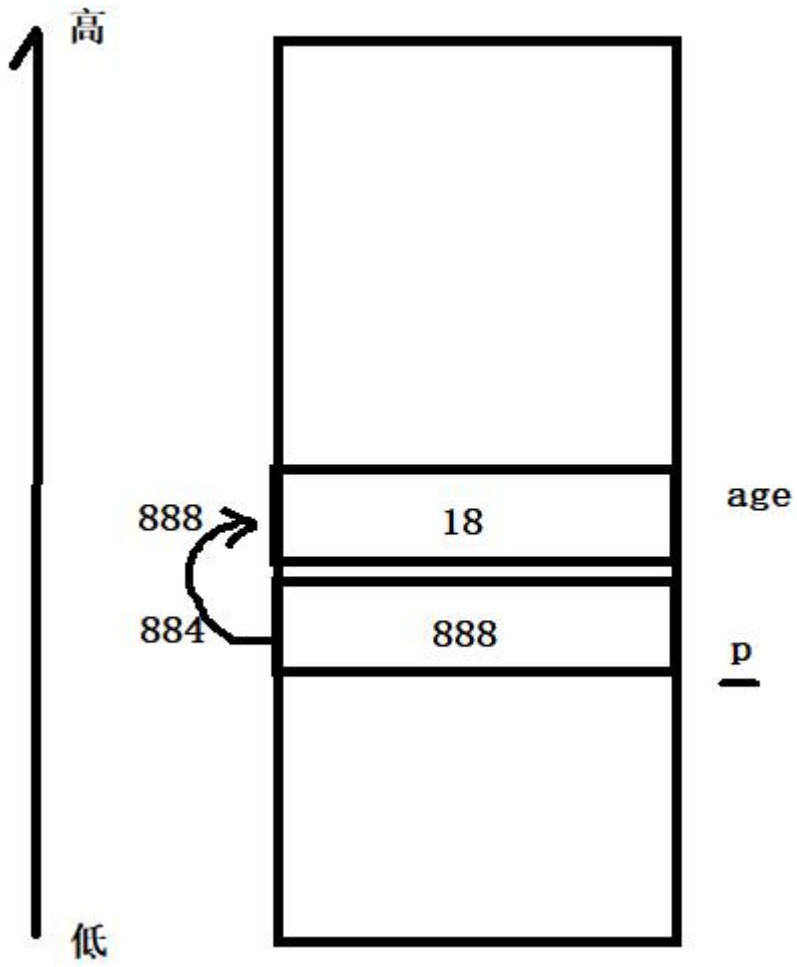
```c
//3.1 Martin 's power: -1474835680     using 16
struct _hero_stat upgrade1(struct _hero_stat hero, int type){

    switch(type){
        case 1://攻击型的英雄
            hero.blood += 1000;
            hero.power += 200;
            hero.level++;
            break;

        case 2://防御型的英雄
            hero.blood += 2000;
            hero.power += 50;
            hero.level++;
            break;

        default:
            break;
    }

    return (hero);
}

//3.2.指针实现升级  Martin 's power: -1474835680    using 3
void  upgrade2(struct _hero_stat *hero, int type){

    switch(type){
        case 1://攻击型的英雄
            hero->blood += 1000;
            hero->power += 200;
            hero->level++;
            break;

        case 2://防御型的英雄
            hero->blood += 2000;
            hero->power += 50;
            hero->level++;
            break;

        default:
            break;
    }

}

int main(void){
```

```
    //int martin = 1000;
    time_t start, end;
    struct _hero_stat martin;
    strcpy(martin.name, "martin");

    martin.blood = 1000;
    martin.level = 100;
    martin.power = 1000;

    time(&start);//1970 年 1 月 1 日 0 时 0 分 0 秒 至今的秒数
    for(int i=0; i<99999999; i++){
        //martin = upgrade1(martin, 1);
        upgrade2(&martin, 1);
    }
    time(&end);

    cout<<"Martin 's power: "<<martin.power<<endl;
    cout<<"using "<<end-start<<endl;

    //add_blood(martin);
    //martin = add_blood(martin);
    //cout<<"martin 's blood: "<<martin<<endl;
    system("pause");
    return 0;
}
```

2. 指针定义

指针是什么

```
// demo 8-2.c

#include <stdio.h>
#include <stdlib.h>


int main(void){
    int age;
    char ch;

    //定义了一个指针
    //指针本身也是一个变量
    //名称是 p, 它是一个指针，可以指向一个整数
    //也就是说： p 的值就是一个整数的地址！！！
    int  *p ;
    char * c;

    //指针 p 指向了 age
    //p 的值，就是 age 的地址
    p = &age;
    c = &ch;

    //scanf_s("%d", &age);
    scanf_s("%d", p);

    printf("age: %d\n", age);
    system("pause");
    return 0;
}
```

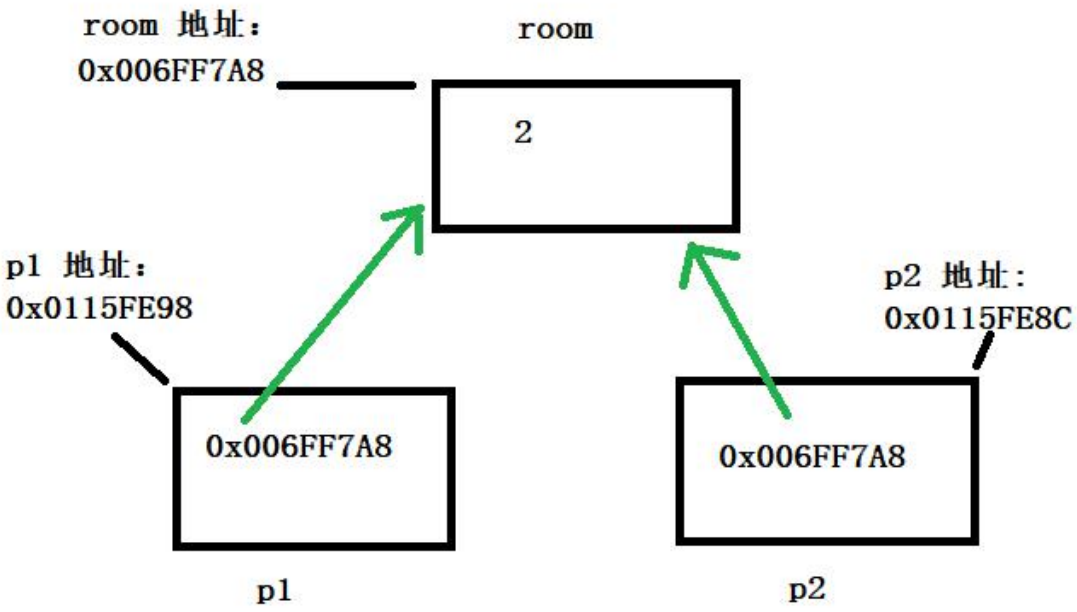


指针的定义

int *p; // int *p1, *p2;
或者
int* p; // int* p1,p2; //p1 是指针， p2 只是整形变量
或者
int * p;
或者
int*p;//不建议

3. 指针的初始化、访问

指针的初始化



demo：

```c
// demo 8-3.c

#include <stdio.h>
#include <stdlib.h>


int main(void){
    int room = 2;

    //定义两个指针变量指向 room
    int *p1 = &room;
    int *p2 = &room;

    printf("room 地址：0x%p\n", &room);

    printf("p1 地址：0x%p\n", &p1);
    printf("p2 地址：0x%p\n", &p2);

    printf("room 所占字节：%d\n", sizeof(room));
    printf("p1 所占字节：%d\n", sizeof(p1));
    printf("p2 所占字节：%d\n", sizeof(p2));

    system("pause");
```

```
    return 0;
}
```

注意：

32 位系统中，int 整数占 4 个字节，指针同样占 4 个字节

64 位系统中，int 整数占 4 个字节，指针同样占 8 个字节

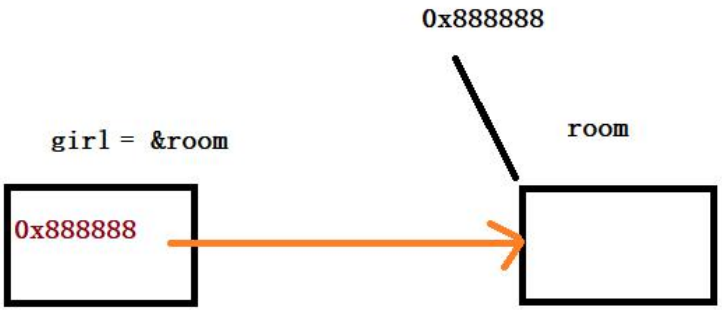
指针的访问

访问指针

demo

```
// demo 8-4.c

#include <stdio.h>
#include <stdlib.h>

int main(void){
    int room = 2 ;
    int room1 = 3 ;//3p

    int *p1 = &room;
    int *p2 = p1;//int *p2 = &room;

    //1.访问（读、写）指针变量本身的值，和其它普通变量的访问方式相同

    int *p3 = p1;
    printf("room 的地址: %d\n", &room);
    printf("p1 的值: %d p2 的值: %d\n", p1, p2);
    printf("p3 的值: %d\n", p3);

    p3 = &room1;
    printf("p3 的值: %d, room1 的地址：%d\n", p3, &room1);//不建议用这种方式

    //使用 16 进制打印，把地址值当成一个无符号数来处理
    printf("p1=0x%p\n", p1);
    printf("p1=0x%x\n", p1);
    printf("p1=0x%X\n", p1);

    system("pause");
    return 0;
}
```

访问指针所指向的内容



*girl == room

```c
// demo 8-5.c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int room = 2 ;

    int * girl = &room;

    int x = 0;
    x = *girl;// *是一个特殊的运算符，*girl 表示读取指针 girl 所指向的变量的值，*girl 相当于 room
    printf("x: %d\n", x);

    *girl = 4; //相当于 room = 4
    printf("room: %d, *girl: %d\n", room, *girl);


    system("pause");
    return 0;
}
```

4. 空指针和坏指针

```c
//8-6.c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int room1 = 666;
    int room2 = 888;

    int girl ;

    int *select ;

    scanf_s("%d", &girl);

    if(girl == 666){
        select = &room1;
    }else if(girl == 888){
        select = &room2;
    }

    printf("选择的房间是:  %d\n", *select);
    system("pause");
    return 0;
}
```

1. 什么是空指针？

空指针，就是值为 0 的指针。（任何程序数据都不会存储在地址为 0 的内存块中，它是被操作系统预留的内存块。）

int *p = 0;

或者

int *p = NULL; //强烈推荐

2. 空指针的使用

1）指针初始化为空指针

int *select = NULL;

目的就是，避免访问非法数据。

2）指针不再使用时，可以设置为空指针

int *select = &xiao_long_lv;

//和小龙女约会

select = NULL;

3)表示这个指针还没有具体的指向,使用前进行合法性判断

```
int    *p = NULL;
  // 。。。。。
 if (p) {    //p  等同于  p!=NULL
  //指针不为空，对指针进行操作
 }
```

3. 坏指针

```
int *select;    //没有初始化
```

情形一

```
printf("选择的房间是：  %d\n", *select);
```

情形二

```
select = 100;
printf("选择的房间是：  %d\n", *select);
```

5. 项目精讲- 渣男、直男、暖男的区别：const

```c
//8-7.c
#include <stdio.h>
#include <stdlib.h>

//const 和指针
int main(void){
    int wife = 24;
    int girl = 18;

    //第一种  渣男型
    int * zha_nan = &wife;
    *zha_nan = 25;
    zha_nan = &girl;
    *zha_nan = 19;

    printf("girl : %d  wife: %d\n", girl, wife);

    //第二种 直男型
    //const int * zhi_nan = &wife; //第一种写法
    int const * zhi_nan = &wife;   // 第二种写法
    //*zhi_nan = 26;
    printf("直男老婆的年龄：%d\n", *zhi_nan);
    zhi_nan = &girl;
    printf("直男女朋友的年龄：%d\n", *zhi_nan);
    //*zhi_nan = 20;

    //第三种 暖男型
    int * const nuan_nan = &wife;
    *nuan_nan = 26;
    printf("暖男老婆的年龄：%d\n", wife);
    //nuan_nan = &girl;   //不允许指向别的地址

    //第四种超级暖男型的
    const int * const super_nuan_nan = &wife;  //不允许指向别的地址，
不能修改指向变量的值
    //*super_nuan_nan = 28;  //
    //super_nuan_nan = &girl;

    system("pause");
    return 0;
}
```

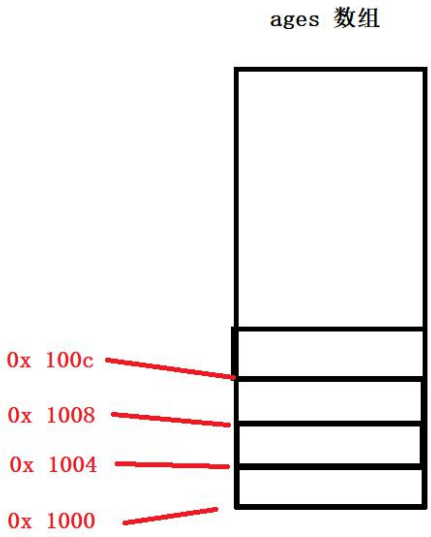
总结： 看 const 离类型(int)近，还是离指针变量名近，离谁近，就修饰谁，谁就不能变

6. 指针的算术运算

指针的自增运算

```
//8.8.c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int ages[]={21,15,18,14,23,28,10};
    int len = sizeof(ages)/sizeof(ages[0]);

    //使用数组的方式来访问数组
    for( int i=0; i<len; i++){
        printf("第%d 个学员的年龄是:%d\n", i+1, ages[i]);
    }

    //打印数组的地址和第一个成员的地址
    printf("ages 的地址: 0x%p , 第一个元素的地址: 0x%p\n", ages,
&ages[0]);

    int *p = ages;
    //访问第一个元素
    printf("数组的第一个元素：%d\n", *p);

    //访问第二个元素
    //p++; // p = p+ 1*(sizeof(int))
    //printf("数组的第二个元素：%d,  第二个元素的地址: 0x%p\n", *p, p);

    for(int i=0; i<len; i++){
        printf("数组的第%d 个元素：%d  地址:0x%p\n", i+1, *p, p);
        p++;
    }
    printf("-----------------------\n");
    char ch[4]={'a','b','c','d'};
    char * cp = ch;

    for(int i=0; i<4; i++){
        printf("数组的第%d 个元素：%c  地址:0x%p\n", i+1, *cp, cp);
        cp++;
    }
    //总结： p++ 的概念是在 p 当前地址的基础上 ,自增 p 对应类型的大小 p
= p+ 1*(sizeof(类型))

    system("pause");
```

```
    return 0;
}
```



总结： p++ 的概念是在 p 当前地址的基础上 ,自增 p 对应类型的大小, 也就是说 p = p+ 1*(sizeof(类型))

指针的自减运算

```
//8-9.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
 * 让用户输入一个字符串，然后反向输出，注意：不能改变原来的字符串！
 * 如： "12345" 逆转成 "54321" 输出
 */

int main(void){
    char input[128];
    int  len;
    char tmp;

    scanf_s("%s", input, 128);
    len = strlen(input);

    //方法 1        交换
    /*for( int i=0; i<len/2; i++){
        tmp = input[i];
        input[i] = input[len-i-1];
        input[len-i-1] = tmp;
    }*/
```

```
	/*for(int i=0; i<len; i++){
		printf("%c", input[i]);
	}*/
	//printf("逆转后：%s\n", input);

	//第二种方法
	/*for(int i=0; i<len; i++){
		printf("%c", input[len-i-1]);
	}
	printf("\n");
	*/

	//第三种方法
	char *p = &input[len-1];

	for(int i=0; i<len; i++){
		printf("%c", *p--);
		//p--;
	}
	printf("\n");

	system("pause");
	return 0;
}
```

指针与整数之间的加减运算

```
//demo8-10.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void){
	int ages[] = {20,18,19,24,23,28,30,38, 35, 32};
	int len = sizeof(ages) / sizeof(ages[0]);

	int *p = ages;
	printf("第 7 个美女的年龄: %d\n", *(p+6));
	printf("*p+6 = : %d\n", *p+6);
	printf("第 3 个美女的年龄: %d\n", *(p+2));

	int *p1 = &ages[4];

	printf("相对于第 5 个美女，她的前一位的年龄: %d\n", *(p1 -1));
	printf("相对于第 5 个美女，她的前三位的年龄: %d\n", *(p1 -3));
```

```
    system("pause");
    return 0;
}
```

知识点：

（1）指针与整数的运算，指针加减数字表示的意义是指针在数组中位置的移动；

对于整数部分而言，它代表的是一个元素，对于不同的数据类型，其数组的元素占用的字节是不一样的，

比如指针 +1，并不是在指针地址的基础之上加 1 个地址，而是在这个指针地址的基础上加 1 个元素占用的字节数：

- 如果指针的类型是 char*，那么这个时候 1 代表 1 个字节地址；
- 如果指针的类型是 int*，那么这个时候 1 代表 4 个字节地址；
- 如果指针的类型是 float*，那么这个时候 1 代表 4 个字节地址；
- 如果指针的类型是 double*，那么这个时候 1 代表 8 个字节地址。

（3）通用公式：

数据类型 *p;

p + n 实际指向的地址：p 基地址 + n * sizeof(数据类型)

p - n 实际指向的地址：p 基地址 - n * sizeof(数据类型)

比如

（1）对于 int 类型，比如 p 指向 0x0061FF14，则：

p+1 实际指向的是 0x0061FF18，与 p 指向的内存地址相差 4 个字节；

p+2 实际指向的是 0x0061FF1C，与 p 指向的内存地址相差 8 个字节

（2）对于 char 类型，比如 p 指向 0x0061FF28，则：

p+1 实际指向的是 0x0061FF29，与 p 指向的内存地址相差 1 个字节；

p+1 实际指向的是 0x0061FF2A，与 p 指向的内存地址相差 2 个字节；

指针与指针之间的加减运算

```
// demo 8-11.c

#include <stdio.h>
#include <stdlib.h>

/**
 *  （1）使用“指针-指针”的方式计算整数数组元素的偏移值；
 * */

int main(void){
    int ages[] = {20,18,19,24,23,28,30,38, 35, 32};
    int ages1[] = {18, 19, 20, 22};
    int len = sizeof(ages) / sizeof(ages[0]);

    int *martin = ages+6;
    int *rock = ages+9;
```

```
    printf("rock - martin = %d\n", rock - martin);
    printf("martin - rock = %d\n", martin - rock);

    martin = ages+6;
    rock = ages1+3;

    printf("martin: %p rock: %p rock-martin: %d\n", martin, rock ,rock -martin);


    system("pause");
    return 0;
}
```

知识点：

（1）指针和指针可以做减法操作，但不适合做加法运算；

（2）指针和指针做减法适用的场合：两个指针都指向同一个数组，**相减结果为两个指针之间的元素数目**，而不是两个指针之间相差的字节数。

比如：int int_array[4] = {12, 34, 56, 78};

int *p_int1 = &int_array[0];

int *p_int2 = &int_array[3];

p_int2 - p_int1 的结果为 3，即是两个之间之间的元素数目为 3 个。

如果两个指针不是指向同一个数组，它们相减就没有意义。

（3）不同类型的指针不允许相减，比如

char *p1;

int *p2;

p2-p1 是没有意义的。

7. 项目精讲-李连杰的二级指针

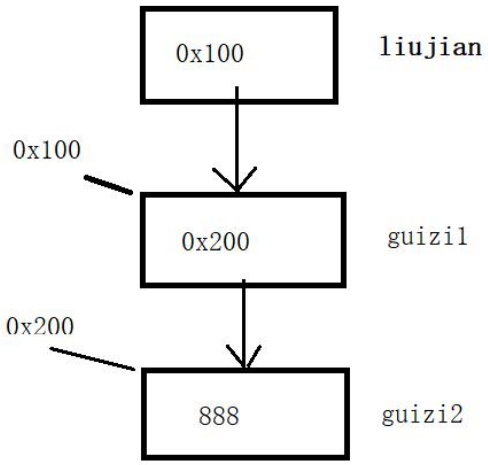
二级指针也是一个普通的指针变量，只是它里面保存的值是另外一个一级指针的地址

定义：

int guizi1 = 888;

int *guizi2 = &guizi1; //1 级指针，保存 guizi1 的地址

int **liujian = &guizi2; //2 级指针，保存 guizi2 的地址，guizi2 本身是一个一级指针变量

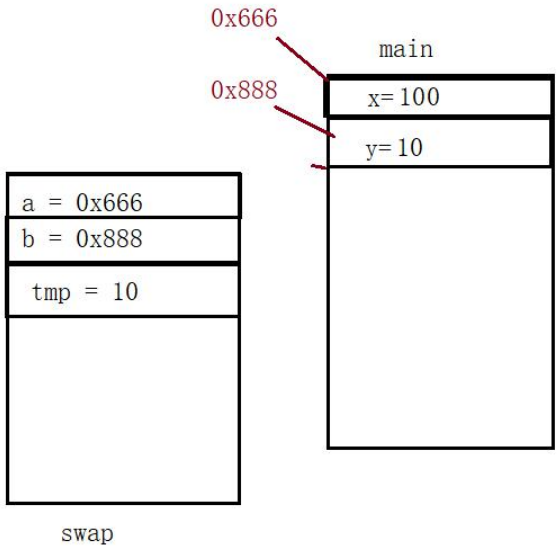



```
// demo 8-12.c

#include <stdio.h>
#include <stdlib.h>

int main(void){
	int guizi2 = 888;		//存枪的第 2 个柜子
	int *guizi1 = &guizi2;	//存第 2 个柜子地址的第一个柜子
	int **liujian = &guizi1; //手握第一个柜子地址的刘建

	printf("刘建打开第一个柜子，获得第二个柜子的地址：0x%p\n", *liujian);
	printf("guizi2 的地址:0x%p\n", &guizi2);

	int *tmp;
	tmp = *liujian;

	printf("访问第二个柜子的地址，拿到枪：%d\n", *tmp);
	printf("刘建一步到位拿到枪：%d\n", **liujian);	//缩写成 **liujian

	system("pause");
	return 0;
```

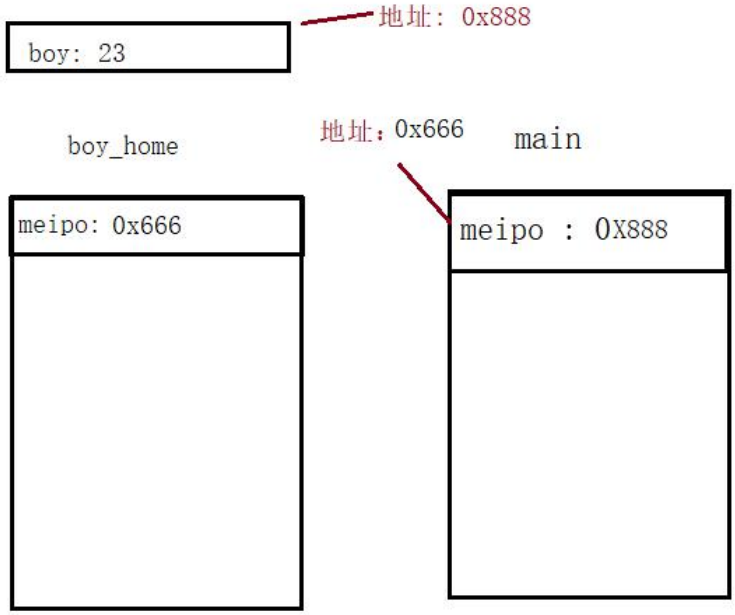
```
}
```

二级指针的用途:

1. 普通指针可以将变量通过参数“带入”函数内部，但没办法将内部变量“带出”函数



2. 二级指针可以不但可以将变量通过参数函数内部，也可以将函数内部变量 “带出”到函数外部。




```
// demo 8-13.c

#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int *b){
```

```
	int tmp =*a;
	*a= *b;
	*b= tmp;
}

void boy_home(int **meipo){
	static int boy = 23;
	*meipo = &boy;
}

int main(void){
	//int x=10, y=100;
	//swap(&x, &y);
	//printf("x=%d, y=%d\n", x, y);
	int *meipo = NULL;
	boy_home(&meipo);
	printf("boy: %d\n", *meipo);
	system("pause");
	return 0;
}
```

8. 项目精讲-多级指针的定义、使用

1. 可以定义多级指针指向次一级指针

比如：

```
int guizi1 = 888;
int *guizi2 = &guizi1;        //普通指针
int **guizi3 = &guizi2;       //二级指向一级
int ***guizi4 = &guizi3;      //三级指向二级
int ****guizi5 = &guizi4;     //四级指向三级
//      有完没完。。。
```

```
// demo 8-14.c

#include <stdio.h>
#include <stdlib.h>

int main(void){
        int guizi1 = 888;
        int *guizi2 = &guizi1;        //普通指针
        int **guizi3 = &guizi2;       //二级指向一级
        int ***guizi4 = &guizi3;      //三级指向二级
        int ****guizi5 = &guizi4;     //四级指向三级

        printf("柜子 2 拿枪: %d\n", *guizi2);
        printf("柜子 3 拿枪: %d\n", **guizi3);
        printf("柜子 4 拿枪: %d\n", ***guizi4);
        printf("柜子 5 拿枪: %d\n", ****guizi5);


    system("pause");
    return 0;
}
```

9. 项目精讲-指针和数组的纠缠

1. 指针表示法和数组表示法

数组完全可以使用指针来访问， days[3] 和 *(days+3) 等同

```
// demo 8-15.c

#include <stdio.h>
#include <stdlib.h>
```

```
void print_months1(int days[], int    months){
	int index = 0;
	for (index = 0; index < months; index++){
		//数组表示法
		printf("Month    %2d    has    %d    days.\n", index+1, days[index]);
		//指针表示法
		//printf("Month    %2d    has    %d    days.\n", index+1, *(days+index));
	}
}

void print_months2(int *days, int months){
	int index = 0;
	for (index = 0; index < months; index++){
		//指针表示表示法
		printf("Month    %2d    has    %d    days.\n", index+1, *(days+index));
		//数组表示法
		printf("Month    %2d    has    %d    days.\n", index+1, days[index]);
	}
}

int main(void)
{
	int days[12]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
	/*int index = 0;
	for (index = 0; index < 12; index++){
		//数组表示法
		//printf("Month    %2d    has    %d    days.\n", index+1, days[index]);

		//指针表示法
		printf("Month    %2d    has    %d    days.\n", index+1, *days+index);
	}*/

	print_months1(days, 6);
	//print_months2(days, 12);

	system("pause");
	return 0;
}
```

2. 存储指针的数组

定义： 类型 *指针数组名[元素个数] ；

```
// demo 8-16.c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
	int girls[4][3]={{173, 158, 166},
	                 {168, 155, 171},
	                 {163, 164, 165},
	                 {163, 164, 172}};
	//int x1,y1,  x2,y2;
	int *qishou[2];//定义一个有两个元素的指针数组，每个元素都是一个指针变量

	if(girls[0][0] > girls[0][1]){
		qishou[0] = &girls[0][0];
		qishou[1] = &girls[0][1];
	}else {
		qishou[0] = &girls[0][1];
		qishou[1] = &girls[0][0];
	}

	for(int i=2; i<12; i++){
		//girls[i/3][i%3]
		if(*qishou[1] >=  girls[i/3][i%3]){
			continue;
		}

		//候选者高于第二位棋手候选女兵
		//1.候选者比"冠军"矮
		if(girls[i/3][i%3] <= *qishou[0]){
			qishou[1] = &girls[i/3][i%3];
		}else { //2.候选者比"冠军"高
			qishou[1] = qishou[0];
			qishou[0] = &girls[i/3][i%3];

		}
	}

	printf("最高女兵的身高: %d , 次高女兵的身高: %d\n", *qishou[0], *qishou[1]);

	system("pause");
```

```
    return 0;
}
```

10. 项目精讲-指针和二维数组

1. 指向数组的指针

int (*p)[3]; //定义一个指向三个成员的数组的指针

访问元素的两种方式：

数组法： (*p)[j]

指针法： *((*p)+j)

```
// demo 8-17.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    /*据同学们报告，A 栋学生楼有学生用高倍望眼镜偷看别人洗澡，
      宿管办领导决定逐个宿舍排查，得到的线报是 A0 到 A3 宿舍的
      某个子最矮的男生。
      */
    int A[4][3]={{173, 158, 166},
                 {168, 155, 171},
                 {163, 164, 165},
                 {163, 164, 172}};

    int (*p)[3]; //定义一个指向三个成员的数组的指针
    int * boy = NULL;

    p = &A[0];

    //第一种    数组下标法
    /*for(int i=0; i<4; i++){
        for(int j=0; j<3; j++){
            printf(" %d", (*p)[j]);  //(*p) 等同于  a[0]  ,a[0][0]等同于 (*p)[0]
        }
        printf("\n");
        p++;
    }*/

    boy = &(*p)[0];
    //boy = (*p);

    //第二种  指针访问法  //int a[3];  int * p ; p = a; 数组成员：  *p  *(p+1) *(p+2)
    for(int i=0; i<4; i++){
        for(int j=0; j<3; j++){
```

```
                printf(" %d", *((*p)+j));
                if( *boy > *((*p)+j)){
                    boy = (*p)+j;
                }

            }
            printf("\n");
            p++;
        }

        printf("偷窥的学生是：  %d\n", *boy);

        system("pause");
        return 0;
}
```

2. 使用普通指针访问二维数组

int A[4][3];
int *p; //定义一个指针
 p = A[0]; 或者 p=&A[0][0];
访问元素的两种方式：

```
// demo 8-18.c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    /*据同学们报告，A 栋学生楼有学生用高倍望眼镜偷看别人洗澡，
      宿管办领导决定逐个宿舍排查，得到的线报是 A0 到 A3 宿舍的
      某个子最矮的男生。
      */
    int A[4][3]={{173, 158, 166},
                 {168, 155, 171},
                 {163, 164, 165},
                 {163, 164, 172}};

    int *boy = NULL;//坏男孩
    int *p =NULL; //定义指针,用以遍历二维数组

    //p = A[0];
    p = &A[0][0];
    boy = p;

    for(int i=1; i<4*3; i++,p++){
```

```
            if(*boy > *p){    //*p  可以替换成 A[i/3][i%3]

                boy = p;
            }
        }

        printf("偷窥的学生是：    %d\n", *boy);
        //2.根据指针计算下标
        int pos = boy - A[0];

        printf("index: %d\n", pos);
        printf("位于  A[%d]  宿舍\n", pos/3);
        printf("是第  %d  个成员\n", pos%3);

        system("pause");
        return 0;
}
```

11. 项目精讲-“我们不一样“之数组与指针的区别

数组：数组是用于储存多个相同类型数据的集合。
指针：指针是一个变量，但是它和普通变量不一样，它存放的是其它变量在内存中的地址。

1. 赋值

数组：只能一个一个元素的赋值或拷贝
指针：指针变量可以相互赋值

2. 表示范围

数组有效范围就是其空间的范围，数组名使用下表引用元素，不能指向别的数组
指针可以指向任何地址，但是不能随意访问，必须依附在变量有效范围之内

3. sizeof

数组：

数组所占存储空间的内存：sizeof（数组名）
数组的大小：sizeof（数组名）/sizeof（数据类型）

指针：

在 32 位平台下，无论指针的类型是什么，sizeof（指针名）都是 4.
在 64 位平台下，无论指针的类型是什么，sizeof（指针名）都是 8.

4. 指针数组和数组指针

针指数组:

```
int *qishou[2];//定义一个有两个元素的指针数组，每个元素都是一个指针变量
int girl1= 167;
int girl2 = 171;
qishou[0] = &girl1;
qishou[1] = &girl2;
```

数组指针：

```
int (*p)[3]; //定义一个指向三个成员的数组的指针
访问元素的两种方式：
int A[4][3]={{173, 158, 166},
             {168, 155, 171},
             {163, 164, 165},
             {163, 164, 172}};
p = &A[0];
数组法：   (*p)[j]
指针法：   *((*p)+j)
```

5. 传参

数组传参时，会退化为指针!
（1）退化的意义：C 语言只会以值拷贝的方式传递参数，参数传递时，如果只拷贝整个数组，效率会大大降低，并且在参数位于栈上，太大的数组拷贝将会导致栈溢出。
（2）因此，C 语言将数组的传参进行了退化。将整个数组拷贝一份传入函数时，将数组名看做常量指针，传数组首元素的地址。

```
// demo 8-19.c

#include <stdio.h>
#include <stdlib.h>

/*------------------ <一维数组传参> ----------------------*/

/*方式一: 形参不指定数组大小
    用数组的形式传递参数，不需要指定参数的大小，
 因为在一维数组传参时，形参不会真实的创建数组，
 传的只是数组首元素的地址。
*/
void method_1(int arr[], int len)
{
    for(int i=0; i<len; i++){
        printf(" arr[%d] = %d\n", i, arr[i]);
    }
}


//方式二：指定数组大小
void method_2(int arr[10])
{
    for(int i=0; i<10; i++){
        printf(" arr[%d] = %d\n", i, arr[i]);
    }
}

//方式三: 一维数组传参退化，用指针进行接收，传的是数组首元素的地址
void method_3(int *arr, int len)
{
    for(int i=0; i<len; i++){
        printf(" arr[%d] = %d\n", i, arr[i]);
    }
}
```

```
int main102()
{
	int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
	method_1(arr, 10);
	printf("------华丽的分隔线------\n");
	method_2(arr);
	printf("------华丽的分隔线------\n");
	method_3(arr, 10);
	system("pause");
	return 0;
}

/*-------------------- <指针数组传参> ----------------------*/

//方式一: 指针数组传参，声明成指针数组，不指定数组大小
void method_4(int *arr[], int len)
{
	for(int i=0; i<len; i++){
		printf(" arr[%d] = %d\n", i, *arr[i]);
	}
}

//方式二: 指针数组传参，声明成指针数组，指定数组大小
void method_5(int *arr[10])
{
	for(int i=0; i<10; i++){
		printf(" arr[%d] = %d\n", i, *arr[i]);
	}
}

//方式三: 二维指针传参
//传过去是指针数组的数组名，代表首元素地址，而数组的首元素又是一个指针，
//就表示二级指针，用二级指针接收
void method_6(int **arr, int len)
{
	for(int i=0; i<len; i++){
		printf(" arr[%d] = %d\n", i, *(*(arr+i)));
	}
}

int main()
{
	int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
	int *arr_p[10] = {0};
	for(int i=0; i<10; i++){
		arr_p[i] = &arr[i];
	}
```

```
    method_4(arr_p, 10);
    printf("------华丽的分隔线------\n");
    method_5(arr_p);
    printf("------华丽的分隔线------\n");
    method_6(arr_p, 10);
    system("pause");
    return 0;
}
```

12. void 类型指针

void => 空类型

void* => 空类型指针，只存储地址的值，丢失类型，无法访问，要访问其值，我们必须对这个指针做出正确的类型转换，然后再间接引用指针。

所有其它类型的指针都可以隐式自动转换成 void 类型指针，反之需要强制转换

```
// demo 8-21.c

#include <stdio.h>
#include <stdlib.h>


int main(void){
    int arr[]={1, 2, 3, 4, 5};
    char ch = 'a';
    void *p = arr;//定义了一个void 类型的指针

    //p++;  //不可以,   void * 指针不允许进行算术运算


    p = &ch;  //其它类型可以自动转换成void * 指针

    //printf("数组第一个元素: %d\n", *p); //不可以进行访问
    printf("p: 0x%p ch: 0x%p\n", p, &ch);

    //强制类型转化
```

```
    char * p1 = (char *)p;

    printf("p1 指向的字符是： %c\n", *p1);

    system("pause");
    return 0;
}
```

13. 函数指针

```
// demo 8-22.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


int compare_int(const void *a, const void *b){
    //printf("调用compare_int 啦，你好骚气哦！ \n");
    int *a1 = (int *) a;
    int *b1 = (int *) b;
    //printf("a 的地址： 0x%p b的地址： 0x%p\n", &a, &b);

    return *b1 - *a1;
}


int compare_char(const void *a, const void *b){
    //printf("调用compare_char 啦，你好骚气哦！ \n");
    char c1 = *((char *) a);
    char c2 = *((char *) b);

    if(c1>='A' && c1<='Z') c1+=32;
    if(c2>='A' && c2<='Z') c1+=32;

    return c1 - c2;
}

int main(void){
    int x = 10;
    int y = 20;

    //函数有没有地址?
    //printf("compare_int 的地址： 0x%p \n", &compare_int);
    //compare_int(&x, &y);
```

```
    //函数指针的定义 把函数声明移过来，把函数名改成 (* 函数指针名)
    int (*fp)(const void *, const void *);

    /*贝尔实验室的C和UNIX的开发者采用第1种形式，而伯克利的UNIX推广者却采用第2种形式ANSI  C 兼容了两种方式*/
    fp = &compare_int;  //
    (*fp)(&x, &y);  //第1种,按普通指针解引的放式进行调用，(*fp) 等同于compare_int
    fp(&x, &y);       //第2种 直接调用

    //qsort 对整形数组排序
    int arr[]={2, 10, 30, 1, 11, 8, 7, 111, 520};
    qsort(arr, sizeof(arr)/sizeof(int), sizeof(int), &compare_int);

    for(int i=0; i<sizeof(arr)/sizeof(int); i++){
        printf(" %d", arr[i]);
    }

    //qsort 可以对任何类型的数组进行排序
    char arr1[]={"abcdefghiABCDEFGHI"};
    qsort(arr1, sizeof(arr1)/sizeof(char)-1, sizeof(char), &compare_char);
    for(int i=0; i<sizeof(arr1)/sizeof(char)-1; i++){
        printf(" %c", arr1[i]);
    }
    system("pause");
    return 0;
}
```

14. 项目精讲-特殊的“别名”：引用

引用专题

变量名回顾

变量名实质上是一段连续存储空间的别名，是一个标号(门牌号)

程序中通过变量来申请并命名内存空间

通过变量的名字可以使用存储空间

问题 1：对一段连续的内存空间只能取一个别名吗？

1 引用概念

a) 在C++中新增加了引用的概念

b) 引用可以看作一个已定义变量的别名

c) 引用的语法：Type& name = var;

d） 引用做函数参数那？（引用作为函数参数声明时不进行初始化）

```
// demo 8-23.c

void main()
{
	int a = 10; //c编译器分配4个字节内存。。。a内存空间的别名
	int &b = a;   //b就是a的别名。。。
	a =11; //直接赋值
	{
		int *p = &a;
		*p = 12;
		printf("a %d \n",a);
	}
	b   = 14;
	printf("a:%d b:%d", a, b);
	system("pause");
}
```

2 引用是 C++的概念

属于C++编译器对C的扩展

```
// demo 8-24.c

问题：C中可以编译通过吗？
int main()
{
	int a = 0;
	int &b = a;
```

```
    b = 11;  //*b = 11;

    return 0;
}
```

结论：请不要用C的语法考虑 b=11

3 引用做函数参数

普通引用在声明时必须用其它的变量进行初始化，
引用作为函数参数声明时不进行初始化

```
// demo 8-25.c

//05复杂数据类型  的引用
struct Teacher
{
    char name[64];
    int age ;
};

void printfT(Teacher *pT)
{
    cout<<pT->age<<endl;
}

//pT是t1的别名 ,相当于修改了t1
void printfT2(Teacher &pT)
{
    //cout<<pT.age<<endl;
    pT.age = 33;
}

//pT和t1的是两个不同的变量
void printfT3(Teacher pT)
{
    cout<<pT.age<<endl;
    pT.age = 45; //只会修改pT变量 ,不会修改t1变量
}
void main()
{
    Teacher t1;
    t1.age = 35;

    printfT(&t1);

    printfT2(t1); //pT是t1的别名
    printf("t1.age:%d \n", t1.age); //33
```

```
    printfT3(t1) ;// pT是形参 ,t1 copy一份数据 给pT        //---> pT = t1
    printf("t1.age:%d \n", t1.age); //35

    cout<<"hello..."<<endl;
    system("pause");
    return ;
}
```

4 引用的意义

```
1）引用作为其它变量的别名而存在，因此在一些场合可以代替指针
2）引用相对于指针来说具有更好的可读性和实用性
int swap(int &a, int &b){
    int t = a;
    a = b;
    b = t;
    return 0;
}

int swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
    return 0;
}
```

5 引用本质思考

思考1：C++编译器背后做了什么工作？

```
// demo 8-26.c

int main()
{
    int a = 10;
    int &b = a;
    //b是a的别名，请问c++编译器后面做了什么工作？
    b = 11;
    cout<<"b--->"<<a<<endl;
    printf("a:%d\n", a);
    printf("b:%d\n", b);
    printf("&a:%d\n", &a);
```

```
    printf("&b:%d\n", &b);  //请思考：对同一内存空间可以取好几个名字吗？
    system("pause");
    return 0;
}
```

单独定义的引用时，必须初始化；说明很像一个常量

思考2：普通引用有自己的空间吗？

```
// demo 8-27.c

struct Teacer {
    int &a;
    int &b;
};
int main()
{
    printf("sizeof(Teacher) %d\n", sizeof(Teacer));
    system("pause");
    return 0;
}
```

引用是一个有地址，引用是常量。。。。。

char *const p

6 引用的本质

1）引用在C++中的内部实现是一个常指针

Type& name ←→Type* const name

2）C++编译器在编译过程中使用常指针作为引用的内部实现，因此引用所占用的空间大小与指针相同。

3）从使用的角度，引用会让人误会其只是一个别名，没有自己的存储空间。这是C++为了实用性而做出的细节隐藏

```
void func(int &a)
{
    a = 5;
}
```

```
void func(int *const a)
{
    *a = 5
}
```

```
void func(int &var){
    var = 15;
}
```

```
void func(int *const var){
    *var = 15;
}
```

```
int main()
{
    int   value= 10;
```

```
    func(value);
}
```

7 引用结论

1)当实参传给形参引用的时候，只不过是c++编译器帮我们程序员手工取了一个实参地址，传给了形参引用（常量指针）

2）当我们**使用引用**语法的时，我们不去关心编译器引用是怎么做的

当我们**分析奇怪的语法现象**的时，我们才去考虑c++编译器是怎么做的

8 函数返回值是引用(引用当左值和右值)

C++引用使用时的难点：

当函数返回值为引用时

- 若返回栈变量，不能成为其它引用的初始值，不能作为左值使用

若返回静态变量或全局变量

- 可以成为其他引用的初始值
- 即可作为右值使用，也可作为左值使用

(注：C++链式编程中，经常用到引用，运算符重载专题)

返回值是基础类型，当引用

```
// demo 8-28.c

int getA1()
{
    int a;
    a = 10;
    return a;
}

//基础类型a返回的时候，也会有一个副本
int& getA2()
{
    int a;
    a = 10;
    return a;
}

int* getA3()
{
    int a;
```

```
    a = 10;
    return &a;
}
```

返回值是static变量，当引用

```
//static修饰变量的时候，变量不是一个临时变量
int getA1()
{
    int a;
    a = 10;
    return a;
}


int& getA2()
{
    static int a ;
    a = 10;
    return a;
}

int& getAA2()
{
    int a;
    a = 10;
    return a;
}

int* getA3()
{
    static int a;
    a = 10;
    return &a;
}

int& getA4() {
    static int a = 10;
    return a;
}




int main(void) {
    int a1 = 1;
    int a2 = 2;
```

```
	a1 = getA1();
	a2 = getA2();
	//int &a3 = a1;
	//a2 = a3;

	int &a3 = getA2();
	int *a4 = getA3();

	//1.不管是指针还是引用，如果出现在右值里，我们要根据具体情况具体分析，如果是局部变量，则会出现问题，
	//但是，编译器不会报错，后果自负
	//2.如果是静态变量或全局变量，则不会出现问题
	printf("a1: %d\n", a1);
	printf("a2: %d\n", a2);
	printf("a3: %d\n", a3);
	printf("a4: %d\n", *a4);

	getA4() = 100;
	printf("getA4(): %d\n", getA4());
	system("pause");
}
```

返回值是形参，当引用

```
int& g1(int &a) {

	a = 99;
	return a;
}

int&	g2(int *p) //
{
	*p = 100;
	return *p;
}



//当我们使用引用语法的时候 ，我们不去关心编译器引用是怎么做的
//当我们分析乱码这种现象的时候，我们才去考虑c++编译器是怎么做的。。。。
int main()
{
	int a1 = 10;
	int &a5 = g1(a1);
	printf("a1: %d	a5: %d	&a1:%p	&a5:%p\n", a1, a5, &a1, &a5);

	system("pause");
```

```
    return 0;
}
```

返回值非基础类型

1. 结构体相对简单，和普通类型一样
2. 但如果返回值是类的对象，情况就变得复杂了，后面我们再进行讲解

9 指针引用

```
// demo 8-29.c

struct Teacher
{
    char name[64];
    int age;
};

struct Teacher * getT(void) {

    struct Teacher * tmp = (struct Teacher *)malloc(sizeof(struct Teacher));
    tmp->age = 37;
    return tmp;
}

int getT1(struct Teacher ** p1) {
    struct Teacher * tmp = (struct Teacher *)malloc(sizeof(struct Teacher));
    if (!tmp) {    //!tmp  不等同于  tmp == NULL
        return -1;
    }

    *p1 = tmp;
    tmp->age = 37;
    return 0;
}

int getT2(struct Teacher* &p2) {
    struct Teacher * tmp = (struct Teacher *)malloc(sizeof(struct Teacher));
    if (!tmp) {    //!tmp  不等同于  tmp == NULL
        return -1;
    }

    p2 = tmp;
```

```
	p2->age = 38;
	return 0;
}


int main(void)
{
	struct Teacher *p = NULL;
	//p = getT();
	//int ret = getT1(&p);
	int ret = getT2(p);
	printf("p->age: %d\n", p->age);
	system("pause");

	return 0;
}
```

10 常引用

在 C++中可以声明 const 引用
语法：　　const Type& name = var;
const 引用让变量拥有只读属性

分两种情况：
1. 用变量初始化常引用
2. 用字面量初始化常量引用

```
int main(void) {
	int a = 10;
	int &b = a;

	printf("b: %d\n", b);
	//1.用变量初始化常引用
	int x = 20;
	const int &y = x;  //常引用是让变量引用变成只读，不能通过引用对变量进行修改

	//2>用字面量初始化常量引用
	//const int c1 = 10;
	const int &c2 = 10;// 这样是否可行？可行，这个是在 C++中，编译器会对这样的定义的引用分配内存，这算是一个特例



	system("pause");
	return 0;
}
```

Const 引用结论：

1）Const & int e　相当于 const int * const e

2）普通引用 相当于 int *const e1

3）当使用常量（字面量）对const引用进行初始化时，C++编译器会为常量值分配空间，并将引用名作为这段空间的别名

4）使用字面量对const引用初始化后，将生成一个只读变量

15. 项目精讲-常见错误总结

1. 使用未初始化的指针

```
int main(void) {

	int x, *p;
  //int y = 100;

  /*常见错误 1：  使用未初始化的指针  */
  //x = 666;

  //printf("p: 0x%p\n", p);
  *p = x; //错误，指针未初始化

	system("pause");
	return 0;
}
```

2. 将值当做地址赋给指针

```
int main(void) {
	int x, *p;

  /*常见错误 2：将值当做地址赋给指针  */
  p = x;

	system("pause");
	return 0;


}
```

3. 忘记解引直接访问内存

```
int main(void) {
 /*常见错误 3：忘记解引直接访问内存  */
  char input[64];
  //char *p1, *p2;
```

```
    //p1 = &input[0];
    //p2 = &input[1];

    //if(p1>p2){//比较数组成员的大小，这样不会报错
          //
    //}
    system("pause");
    return 0;
}
```

4. 再次使用忽略重新赋初值

```
int main(void) {

    /*常见错误 4：  再次使用忽略重新赋初值*/
    char input[64];
    char *p1 = input;
    do {
        gets(input);
        p1 = input;
        while(*p1) printf(" %c", *p1++);

    } while(strcmp(input, "done")!=0);

    system("pause");
    return 0;
}
```

16. 英语不是障碍：计算机英语加油站

pointer	指针
basically	基本地
variable	变量
generally	通常，普通地
data	数据
object	对象
memory	内存
address	地址
function	函数
parameter	变量
chapter	章节

Pointers? What are they? Basically, a pointer is a variable (or, more generally, a data object) whose value is a memory address.

指针？什么是指针？从根本上看，指针（pointer）是一个值为内存地址的变量（或数据对象）。

Just as a char variable has a character as a value and an int variable has an integer as a value, the pointer variable has an address as a value. Pointers have many uses in C; in this chapter, you’ ll see how and why they are used as function parameters.

正如 char 类型变量的值是字符，int 类型变量的值是整数，指针变量的值是地址。在 C 语言中，指针有许多用法。本章将介绍如 何把指针作为函数参数使用，以及为何要这样用。

17. 项目实现

需求： 编写程序找出最近一段时间每个号码出现的次数并把结果保存到一个数组，供其它分析模块调用，往期数据保存在一个名为 ball.txt 中：

```
 8 11 17 23 32 33    10
 4  5  7 10 12 22    16
 3 13 15 18 21 33    16
 4  8  9 13 28 33     4
 9 15 19 21 23 29    15
 9 11 15 22 24 26     3
 1  5  7  9 10 20    16
 2 10 13 16 23 32     8
 1 7  12 14 18 25    10
 9 12 21 27 29 30     5
```

算法设计

1) 将双色球往期数据从文件读入一维数组;
2) 逐行遍历一维数组的每个元素，统计前六个球在 1-33 范围内出现的总次数.

```
// demo 8-29.c

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

#define NUM 7

bool statistics(const char *path, int * ball_16, int ball_16_len ){
    int result[NUM];
    ifstream file;
    int i=0;

    if(!path) return false;

    file.open(path);
    if(file.fail()){
        cerr<<"打开输入文件出错."<<strerror(errno)<<endl;
        return false;
    }

    //从数据文件读数据到数组,一行必须能读取 7 个
    do{
```

```
            i=0;
            for(i=0; i<NUM; i++){
                file>>result[i];
                if(file.eof()){
                    break;
                }
                if(file.fail()){
                    cerr<<"读取文件失败，原因:"<<strerror(errno)<<endl;
                    break;
                }
            }



            if(i==0) break;//记录正常结束

            //如果最后未满 7 个
            if(i<(NUM-1)){
                cerr<<"仅读到"<<i<<"个记录，预期读取 7 个.";
                break;
            }

            for(i=0; i<NUM; i++){
                cout<<" "<<result[i];
            }
            cout<<endl;
            //对读入的数据进行统计
            for(i=0; i<NUM; i++){
                int index = *(result+i)-1;

                if(index >=0 && index<ball_16_len){
                    *(ball_16+index) += 1;
                }
            }

        }while(1);

        //关闭文件
        file.close();
        return true;
}

int main(){

        string filename;
        int ball_1_6[33]={0};
        int i=0;
```

```
    cout <<"请输入文件名.\n";
    cin   >> filename;
    if(!statistics(filename.c_str(), ball_1_6, 33)){
        cerr<<"统计出错!"<<endl;
    }

    for(i=0; i<33; i++){
        cout<<"第 "<<i+1<<" 出现次数："<<ball_1_6[i]<<endl;
    }

    system("pause");

    //结束程序
    return 0;
}
```

18. 项目练习

练习 1

1. 实现含有中文字符的字符串逆转，如: “我是小萌新”转换成“新萌小是我”

练习 2

有一个整形数组， a[3] = {7,2,5}， 要求使用指针实现数组成员由小到大的顺序排列，即结果为：a[3] = {2,5,7};

练习 3

实现一个函数，函数完成如下功能：

1.函数的输入为一个数组，数组的成员个数不定（即：可能为 0 个，也可能为多个）

2.函数找到成员的最大元素和最小元素，并能让函数的调用者知道最大元素和最小元素是哪一个

练习 4

实现一个函数，使用指针连接两个字符串。

函数输入： 两个源字符串的指针，目的字符串的指针

练习 5

编写一个程序，初始化一个 double 类型的数组，然后把该数组的内容 拷贝至 3 个其他数组中（在 main()中声明这 4 个数组）。使用带数组表示法的 函数进行第 1 份拷贝。使用带指针表示法和指针递增的函数进行第 2 份拷贝。 把目标数组名、源数组名和待拷贝的元素个数作为前两个函数的参数。第 3 个函数以目标数组名、源数组名和指向源数组最后一个元素后面的元素的指 针。也就是说，给定以下声明，则函数调用如下所示：

```
double source[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
double target1[5];
```

```
double    target2[5];
double    target3[5];
copy_arr(target1,    source,    5);
copy_ptr(target2,    source,    5);
copy_ptrs(target3, source, source + 5);
```