指针和数组

指针本质

> 本质是一个变量
>
> 需要占用一定的内存空间
>
> 用于保存内存地址的值
>
> 不同类型的指针占用内存空间大小相同
>
> 指针声明时，*表示所声明的变量为指针
>
> 指针使用时，*表示所取指针所指向的内存空间的值

传值 /传址调用

> 需要改变实参的值，使用指针
>
> 函数调用时实参值 将复制到形参
>
> 指针适用于复杂数据类型作为参数的函数，效率更高

常量与指针

> 左数右指：const在*左边，指针指向的数据为常量,const在 *右边指针本身为常量。
>
> 注意：int const * 和 const int *等价，只看 * 位置

数组

> 数组是相同类型的变量的有序集合
>
> a代表数组第一个元素的起始地址
>
> 数组在一片连续的内存空间中存储元素
>
> 数组元素可以显式或者隐式指定
>
> ```
> int a[5]={1,2};  //1，2，0，0，0
> int b[]={1,2}; //1，2
> ```
>
> memset赋值效率很低

数组地址与数组名

> 数组地址需要&符号 ,&a
>
> 数组名代表数组首元素的地址,a ， 二者概念不同
>
> 但是数组地址值和数组首元素的地址值相同
>
> 数组名只能作为右值使用

> 数组名可以看成常量指针，特殊情形1.数组名作为sizeof的参数，2数组名作为&运算符的参数不能看作常量指针

定义为指针，声明为数组

> 编译器处理数组
>
> 将指针的值打印出来，即地址。无寻址操作，直接取值
>
> 编译器处理指针
>
> 将指针中的值作为地址，取该地址中的值。有一次寻址操作，从该地址中取值

字符串

> 分配在栈空间/堆空间/只读存储区
>
> c语言中的字符串是以‘/0’结束的字符数组
>
> 堆空间必须用malloc方式分配

```
char* s ={'1','2','\0'} //栈空间
char* s3 = "hello"; //只读存储区,里面的数据不能改变，比较特殊
char *s4 = (char*)malloc(6*sizeof(char));//堆空间
```

```
//求字符串长度,即第一个'\0'前出现的字符个数
size_t strlen(const char* s)
{
    size_t length=0;
    assert(s); //判空
    while(*s++)
    {
        length++;
    }
    return length;
}
//左++ > * > 右++优先级
//strlen的返回值size_t类型是无符号数，不能相减，因为始终为正；
if(strlen(a)>strlen(b)){};
if(strlen(a)-strlen(b)>=0){};  //不可以
```

> 使用一条语句实现strlen

```
size_t strlen(const char* s)
{
    return (assert(s),(*s ? (strlen(s+1)+1) :0));
}
```

> strcpy,strcat必须保证目标字符数组的剩余空间足够
>
> strcmp不会修改参数值，但依然以'/0'结尾，相等返回0
>
> 实现strcpy

```
char* strcpy(char* dst,char* src)
{
    char *ret = dst;
    assert(dst && src);
    while((*dst++=*str++)!='\0');
    return ret;
}
```

strncpy只复制len个字符到目标字符串，若长度小于len,'\0'填充，大于len,只复制len个，且不会以'\0'结束

strncat总是最结果字符串后面加'\0',不会用'\0'填充剩余空间

strcmp只比较len个字符是否相等

指针 数组对比

数组

连续内存空间，空间大小sizeof(array_type)*array_size

数组名是指向数组第一个元素的常量指针

指针

运算规则

```
int *p=NULL;
p+n;
// 等价于(int)p+n*sizeof(*p)
//结论：当p指向一个同类型的数组时，P+1将指向当前元素的下一个元素，p-1将指向当前元素的上一个元素
```

只有当两个指针指向同一个数组的元素时，指针相减才有意义，代表指针所指元素的下标差

若不在同一个数组，则结果未定义

指针可以进行== ，！ =比较运算

```
#define DIM(a) (sizeof(a)/sizeof(*a))

int main()
{
    char a[] = { 'H','e','l','l','o' };
    char* pBegin = a;
    char* pEnd = a + DIM(a);
    char* p = NULL;
    for (p = pBegin; p != pEnd; ++p)
    {
        printf("%c", *p);
    }
}
```

访问数组的形式

a[1] 和*(a+1) 等价

> 但是下标形式更慢，因为取下一个地址需要用乘法

a vs &a

```
a+1;//加一个元素
(unsigned int)a+sizeof(*a);

&a+1;  //加整个数组
(unsigned int)a+sizeof(*&a);
```

```
#include <stdio.h>
static int i = 1;
#define print(p) printf("result of %d : %d\n",i++,(*p))

int main()
{
    int a[5] = { 1,2,3,4,5 };
    int* p1 = (int*)(&a + 1);
    int* p2 = (int*)((int)a + 1);//a地址+1字节
    int* p3 = (int*)(a + 1);

    printf("%d,%d,%d\n", p1[-1], p2[0], p3[1]);
    //5,随机数，3
}
```

数组参数

> 数组作为函数参数时，编译器将其编译成对应指针，丢失了长度信息。
>
> 因此一般情况下，函数中有数组参数时，需要定义另一个参数标示数组的大小

```
void f(int a[]); //等价下面
void f(int* a);

void f(int a[5]);//等价下面
void f(int* a);
```

```
void f(int a[1000])
{
    printf("%d\n", sizeof(a));
}
//这里就是丢失了长度信息，替换为f(int *a)
int main()
{
    int a[5] = { 0 };
    f(a); //4
}
```