北京航空航天大學
BEIHANG UNIVERSITY

程序设计基础

Fundamentals of Programming

北京航空航天大学 程序设计课程组

软件学院 谭火彬

2022年

北京航空航天大學
BEIHANG UNIVERSITY

第七讲 指针初步（I）

Pointer

- 指针与地址
- 使用指针变量
- 指针运算
- 指针与数组

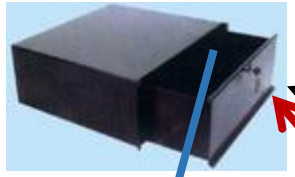
提纲：指针（1）

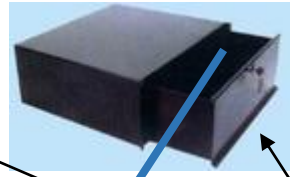
- 7.1 从地址到指针
- 7.2 使用指针变量
- 7.3 指针运算
- 7.4 指针与数组

7.1 指针

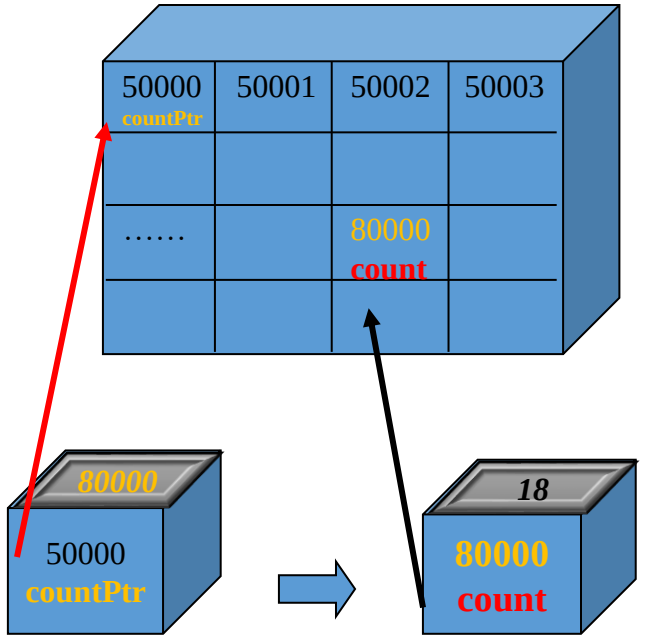
通常变量直接包含特定值，而指针则包含变量的地址

上天入地，无所不能






```
int count = 18;
int *countPtr = &count;
```



count: 一般变量

countPtr: 指针变量

听也不会，写了更废

指针

- 指针是C编程语言一个最强大的特性
- 指针是C的精髓之一
  - ✓使程序简洁、高效、紧凑
- 指针是C中较难掌握的问题之一
  - ✓即便是优秀的程序员，也可能对指针望而生畏！
- 指针使程序可以模拟按引用调用、生成和操作动态数据结构

回顾：变量与内存的关系

◆变量的属性

- ✓名称(name)：有效的标识符
- ✓类型(type)：int 等
- ✓长度(size)：由类型决定
- ✓值(value)：根据输入或赋值
- ✓地址(address)：系统分配（简单变量和数组）

```
int main()
{
    char c;
    short s = 0;
    int a = 55, b, sum;
    double d;
    int x[20];
    ……
    return 0;
}
```

内存（Memory）

	……	60FEF8 d	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	…02	…03	…04 b	…05	…06	…07	…08 a	…09
…0A	…0B	…0C s	…0D	…0E	…0F c	…10	……		

回顾：变量与内存的关系

◆数组变量：具有相同名称和相同类型的一组连续内存空间

```
int a[12] = {1, 3, 5, -2, -4, 6};
for (i = 0; i < 12; i++)
    printf("%d ", a[i]);
printf("\n");
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
1	3	5	-2	-4	6	0	0	0	0	0	0

内存（Memory）

	……	60FEF8a	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	…	…	…	…	……	……		

从地址到指针

◆指针：数据实体地址，其指向相应数据实体所在的内存空间

✓指向哪个数据实体：哪些可以访问

✓具有什么样的类型：什么操作规则

访问数据
- 数据实体的名称 → 直接访问（通过变量）
- 数据实体的地址 → 间接访问（通过指针）

```
char a;
char *aPtr;
aPtr = &a;
a = 'c';
*aPtr = 'x';
```

char类型变量a的地址是一个指向变量a的指针，其类型是一个指向char类型的指针

内存（Memory）

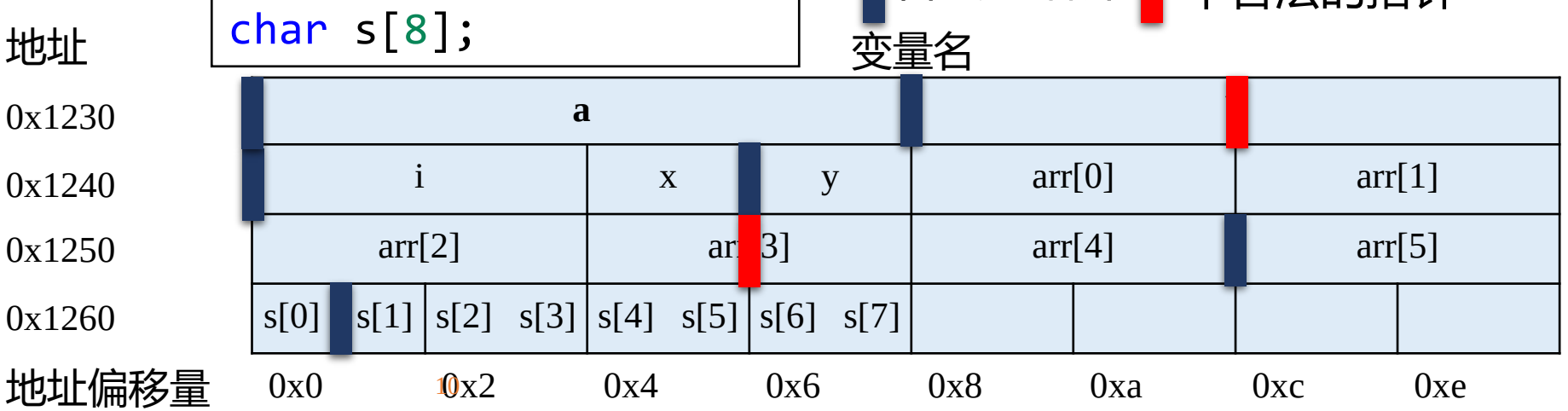
	……	60FEF8a	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	……							

从地址到指针

- 合法的具有指针类型的数据必须指向一个完整的数据实体
  - ✓并不是内存中的任意的地址都可以作为指针
  - ✓计算机的内存空间以字节为单位编址。对于单位长度为多字节的数据实体，其地址是其第一个字节的地址

```
double a, b;
int i;
short x, y;
unsigned int arr[6];
char s[8];
```

一个指针必须指向这些数据实体的第一个字节。在分割线上的地址均为合法的指针。



获取变量的地址

- 数据实体地址：其指向相应数据实体所在的内存空间
- 如何获得数据实体的地址：

普通变量	&a
数组元素	&s[6]
数组	s
函数	max()

```
double a;
char s[8];
```

```
int max(int x, int y, int z)
{  int max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

C07-01. 了解变量地址和存储空间

```c
#include <stdio.h>
int main()
{
    int a = 0;
    printf("address of a: %p\n", &a);
    printf(" sizeof of a: %d\n", sizeof(a));
    return 0;
}
```

计算机的内存空间以字节为单位编址。对于单位长度为多字节的数据实体，其地址是其第一个字节的地址（低地址端）

注意%X和%p输出的区别：两者都是16进制输出，但%p按照地址位数补前导0。

程序输出：

```
address of a: 000000000060FEFA
 sizeof of a: 4
```

◆输出的第 1 行是 a 的地址值，用 16 进制表示的 64 位二进制数（8 字节）

✓在计算机中，用于表示地址值的二进制位数（或字节数）是固定宽度的，又称为 CPU 的寻址空间，一般是 4 字节（32 位编译器）或 8 字节（64 位）

◆上述结果表明：程序在运行时，变量 a 存放在以000000000060FEFA 开始的连续 4 个字节的内存空间中（变量 a 占 4 个字节，见输出的第 2行）

7.2 使用指针变量

- 指针变量是用来存放所指对象地址的变量。
- 定义语法：<类型> * <变量名>;
  - ✓ *　　说明名为<变量名>的变量是一个指针
  - ✓<类型>　是指针所指向的数据类型

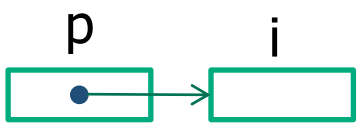
```
int   *px;          //定义一个指向int类型的指针px
char *pc;           //定义一个指向char类型的指针pc
char *aca[10];   //定义一个长度为10的数组aca，数组的每个成员为char*为指针
char (*acp)[10];//定义一个指针变量acp，指向一个长度为10的char数组
int f( );           //定义一个函数f，没有参数，返回值为int
int *fp(int); //定义一个函数fp，一个int类型的参数，返回int类型的指针
int (*fpi)(int);//定义一个函数指针fpi，指向一个int类型参数，返回int类型函数
```

指标变量的定义与赋值：指向简单变量

◆取地址运算符：&

✓一元运算符，取出某个变量的地址，返回变量对应的指针类型

```
int i, *p;
p = &i;
```

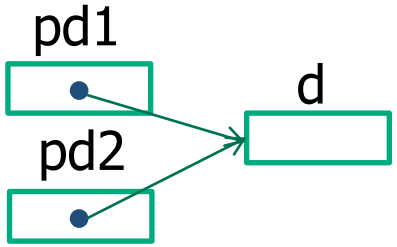


指针p指向了变量i

◆类型相同的指针变量之间可以相互赋值

✓不同类型的指针之间不能直接相互赋值

```
double d, *pd1, *pd2;
pd1 = &d;
pd2 = pd1;
```



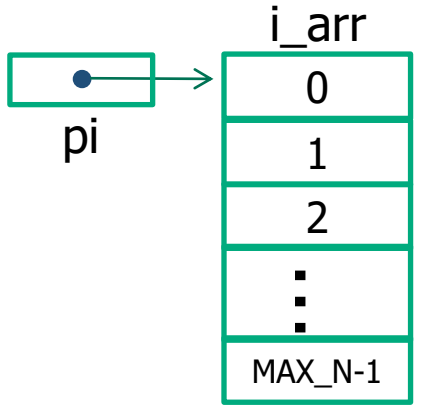
指针pd1和pd2都指向了变量d

指标变量的定义与赋值：指向数组变量

◆数组：可直接赋值给类型相同的指针变量

✓数组名即为指向该数组首地址的指针，即下标为0的元素的地址，

```
int i_arr[MAX_N], *pi;
pi = i_arr;
```



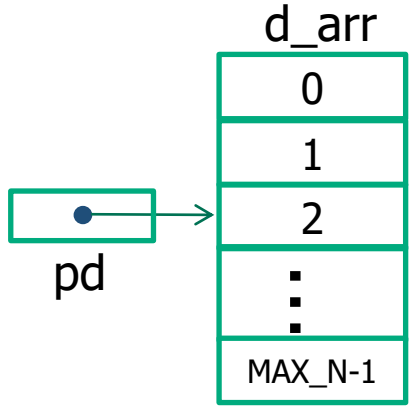
pi指向了数组i_arr

or

pi指向了数组元素i_arr[0]

✓单个数组元素等价于普通变量，其地址可以赋值给类型相同的指针变量

```
double d_arr[MAX_N];
double *pd;
pd = &d_arr[2];
```



pd指向了数组元素d_arr[2]

or

pd指向了数组d_arr[]中从下标为2的元素开始的数组的后部

通过指针访问数据

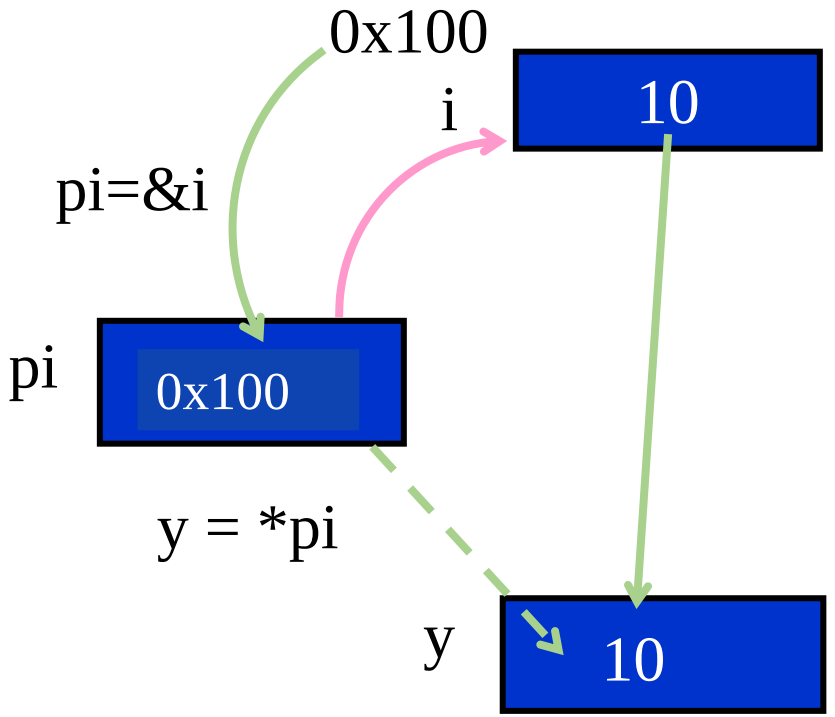
◆取内容运算符：*

✓一元运算符，取指针变量所指向的内容空间的值

✓指针表示的是数据的存储地址，而非数据本身

✓通过指针访问数据时，必须在其左侧加上取内容运算符'*'。

```
int i = 10, y = 20, *pi;
//取普通变量i的地址，赋值给指针
pi = &i;
//取指针变量pi所指向空间的内容，赋值给y
y = *pi;
```



小结：使用指针变量

- 通过指针变量，可以访问两类信息
  - ✓指针本身所代表的地址
  - ✓指针所指向地址的内容值
- 务必要清楚是操作指针本身还是所指向的内容！
  - ✓取变量的地址：&
  - ✓取指针所指向的内容：*

```
int d, e = 8;
d = *&e;
*&e = 10;
```

```
d = e;
e = 10;
```

C07-02：数据的最长行

◆从标准输入上读入多行文本数据，在标准输出上输出其中最长行的长度和内容

输入

```
abcd
abc123
abcdefg
xyz
```

输出

```
7: abcdefg
```

◆问题分析

✓数据结构设计

➢定义两个一维字符数组来存储新输入行及当前最长行

✓主算法设计

```
While (还有新输入行)
        if(新行比以前保存的最长行更长)
                保存新行及其长度;
输出所保存的最长行;
```

C07-02：数据的最长行

```c
#include <stdio.h>
#include <string.h>
#define MAXLINE 1024
int main() {
    int len, max;          //当前行的长度和目前最长行长度
    char line[MAXLINE]; //保存当前输入行 的内容
    char save[MAXLINE]; //保存最长行的内容
    max = 0;
    while (gets(line) != NULL){
        len = strlen(line);
        if (len > max) {
            max = len;
            strcpy(save, line);
        }
    }
    if (max > 0) printf("%d: %s", max, save);
    return 0;
}
```

C07-02a：数据的最长行（指针实现）

◆算法：用指针实现

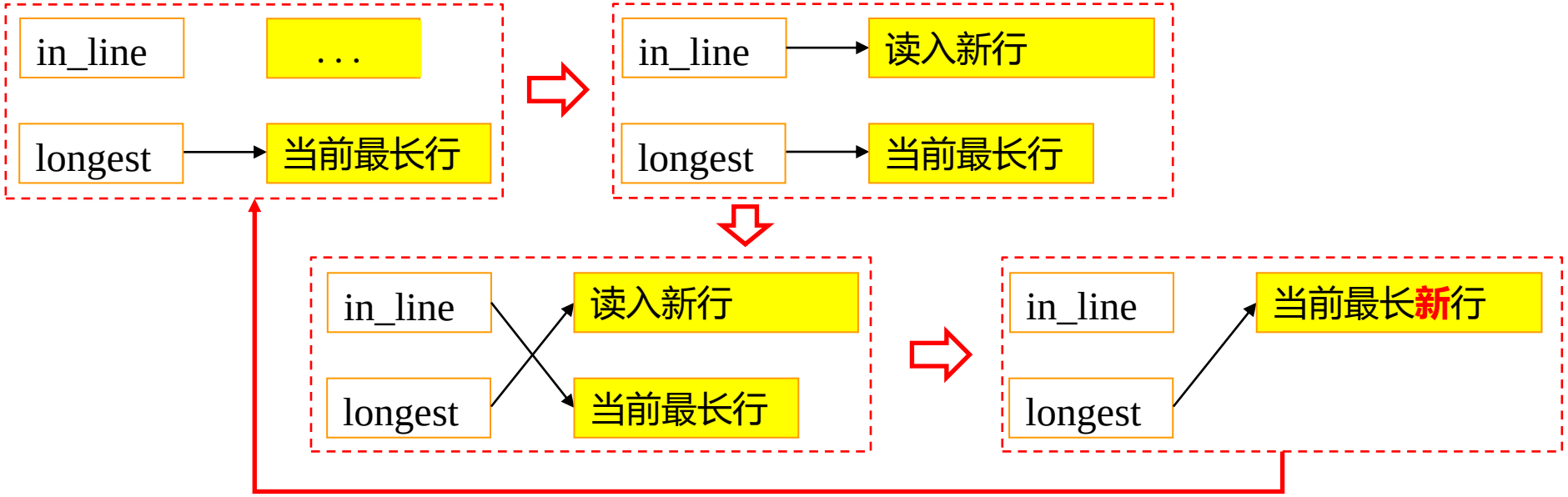
设指针变量in_line和longest分别指向当前行（新行）和当前最长行

while (还有新输入行)

if(in_line所指向的行比longest所指向的行长)

交换in_line和longest指针并保存新行长度；

输出longest所指内容



C07-02a：数据的最长行（指针实现）

```c
#include <stdio.h>
#include <string.h>
#define MAX 1024
int main()
{
  char arr_1[MAX], arr_2[MAX] = "";
  char *in_line = arr_1, *longest = arr_2, *tmp;
  int max_len = 0, len;
  while(gets(in_line) != NULL){
     len = strlen(in_line);
     if(len > max_len){//只交换指针，不交换内容！
        max_len = len;
        tmp = in_line;
        in_line = longest;
        longest = tmp;
     }
  }
    printf("%d:%s\n",max_len, longest);
}
```

两种方式的对比

```c
#include <stdio.h>
#include <string.h>
#define MAX 1024
int main()
{
  char arr_1[MAX], arr_2[MAX] = "";
  char *in_line = arr_1, *longest = arr_2, *tmp;
  int max_len = 0, len;
  while(gets(in_line) != NULL){
     len = strlen(in_line);
     if(len > max_len){//只交换指针，不交换内容!
        max_len = len;
        tmp = in_line;
        in_line = longest;
        longest = tmp;
     }
  }
    printf("%d:%s\n",max_len, longest);
}
```

```c
#include <stdio.h>
#include <string.h>
#define MAXLINE 1024
int main() {
    int len, max; //当前行的长度和目前最长行长度
    char line[MAXLINE];//保存当前输入行 的内容
    char save[MAXLINE];//保存最长行的内容
    max = 0;
    while (gets(line) != NULL){
        len = strlen(line);
        if (len > max) {
            max = len;
            strcpy(save, line);
        }
    }
    if (max > 0) printf("%d: %s", max, save);
    return 0;
}
```

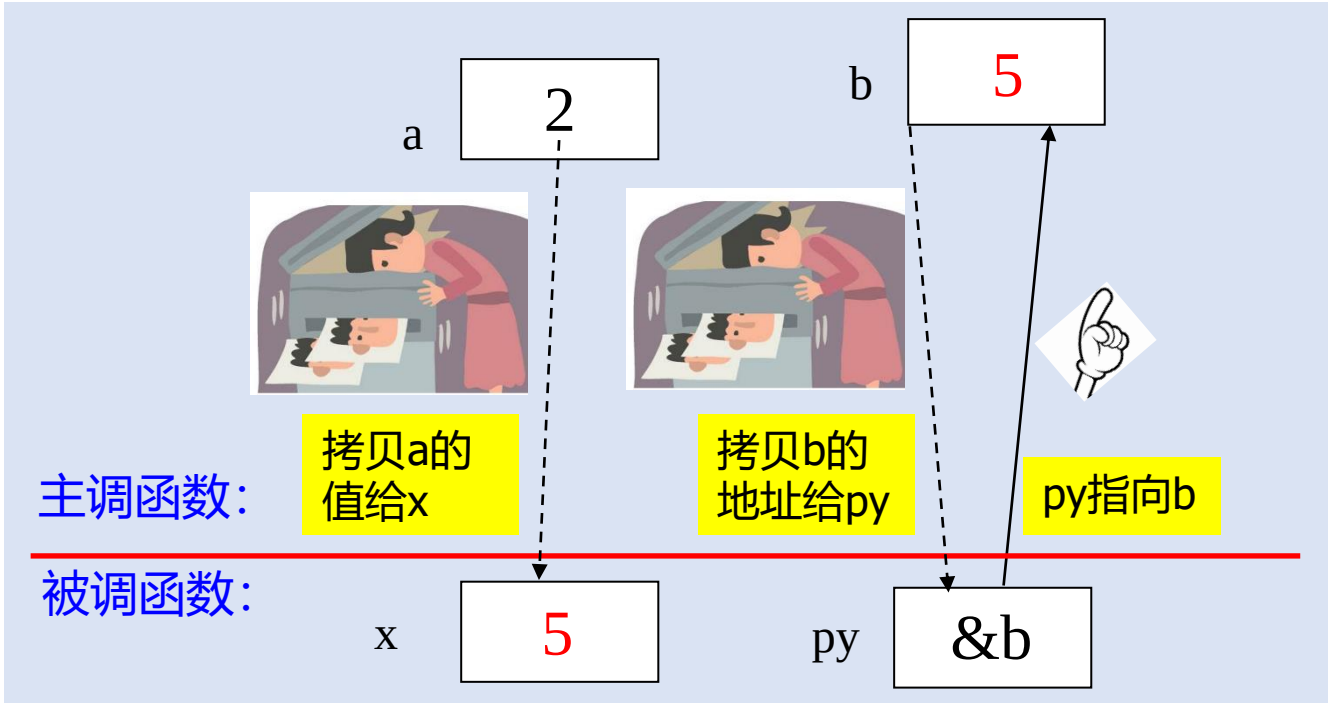
使用指针作为函数参数

- C语言中的函数参数是值传递的，函数调用不会改变实参变量的值，一般通过返回值或者修改全局变量的方式实现
- 定义指针作为函数参数，可以以指针的方式改变函数外部的变量值
  - ✓当一个函数需要同时向外部传递多个数值而又不希望借助全局变量时，可使用指针类型的函数参数

```
int a = 2, b = 3, c = 4;
c = sum(a, &b);
...
int sum(int x, int *py)
{
    x += *py;
    *py = x;
    return x;
}
```

实参传递的是变量的地址

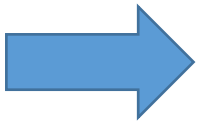
形参定义为指针



使用指针作为函数参数：两个数交换函数

◆自定义函数实现两个整数交换

```
#include <stdio.h>
void swap(int, int);
int main(){
    int a = 5, b =6;
    swap(a, b);
    printf("a=%d,b=%d", a, b);
    return 0;
}
void swap(int x, int y){
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```



```
#include <stdio.h>
void swap(int *, int *);//指针传递
int main()
{
    int a = 5, b = 6;
    swap(&a, &b); //指针传递
    printf("a=%d,b=%d", a, b);
    return 0;
}
void swap(int *x, int *y)
{
    int temp;
    temp = *x; //取内容交换
    *x = *y;
    *y = temp;
}
```

C07-03：多个输出参数：指针作为函数参数

- ◆函数最多只能有一个返回值，如果需要返回多个结果，只能使用指针类型的参数（输出参数，函数内容更新参数的值）
- ◆C07-03：从标准输入上读入多个学生成绩（至少读入 1 个，至多不超过 1000 个，成绩是 0~100的实数），写一个函数 gradeAnalyse，功能包括：接收读入的成绩，计算平均成绩和标准差
- ◆问题分析
  - ✓函数需要返回两个统计量，所以依靠程序的返回值是不行的，可以引入两个指针参数，函数计算的两个统计量写入到参数指针指向的变量中
  - ✓函数要接收读入的成绩，成绩个数可能比较多，把所有成绩都以值传递方式传入函数也不适合，用一个指针参数指向数组元素即可

C07-03：指针作为函数参数

```
void gradeAnalyse(const double *pData, int n, double *pAver, double *pStd)
{
    int i;
    double d, sum = 0.0f, std = 0.0f;
    for (i = 0; i < n; i++)
        sum += pData[i];
    *pAver = sum / n;
    for (i = 0; i < n; i++)
        std += pow(pData[i] - *pAver, 2);
    *pStd = sqrt(std / n);
}
```

```
#include <stdio.h>
#include <math.h>
#define ARRAY_SIZE 1000
void gradeAnalyse(const double *, int,
                  double *, double *);
int main()
{
    int n = 0;
    double data[ARRAY_SIZE], average, std;
    while (scanf("%f", &data[n]) > 0)
        n++;
    gradeAnalyse(data, n, &average, &std);
    printf("%.2f, %.2f\n", average, std);
    return 0;
}
```

数组作为参数本质上就是指针传递

- 若函数的参数列表里有数组，则其本质上指针类型
  - ✓当函数调用时，作为实际参数的数组向函数传递的是数组第一个元素的地址
- 数组名 (a)、数组元素地址 (&a[5]) 和普通变量地址 (&b) 都是指针类型的数据，只要数据类型匹配，都可以作为实际参数传递给函数中指针类型的形式参数
- 无论是形式参数还是实际参数，指针和数组在语法上都是等价的，可以互换
  - ✓一个指针类型的参数所要求的到底是一个数组还是一个普通变量的地址，需要由函数的定义来解释
  - ✓当函数被调用时，实际参数必须符合函数定义的要求，这一点是由程序员而非编译系统来保证的

```
char *strcpy(char dest[], char src[]);
char *strcpy(char *dest, char *src);
```

数组名本质上就是地址

◆数组作为形参，本质上传递的是这组数据的首地址

```c
#include <stdio.h>
void fill_array(int array[10], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        array[i] = i;
}
void print_array(int array[10], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf(i ? " %d" : "%d", array[i]);
    printf("\n");
}
```

```c
int main()
{
    int a[10];
    fill_array(a, 10);
    print_array(a, 10);
    return 0;
}
```

输出：

```
0 1 2 3 4 5 6 7 8 9
```

数组名本质上就是地址

◆C 语言编译器不检查形式参数中数组的长度（忽略它），因此以下三种函数声明均等价

```
void fill_array(int array[10], int n);
void fill_array(int array[], int n);
void fill_array(int *array, int n); // 用这种形式更常见
```

```
#include <stdio.h>
void dummy_array(int array[10]) {
    printf("%d\n", sizeof(array));
}
int main(){
    int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    printf("%d\n", sizeof(array));
    dummy_array(array);
    return 0;
}
```

想一想左边代码运行结果?
第二行输出10？ 40？ 4？ 8？

指针作为函数的返回值

◆指针不但可以用作函数的参数，也可以作为函数的返回值

```
char *strchr(char *s, int c);
char *strrchr(char *s, int c);
```

字符串s中存在字符c，strchr()和strrchr()分别返回字符c在s中第一次和最后一次出现的位置的指针。若无字符c则返回NULL

```
char a[] = "this is test";
printf("%x\n", strchr(a, 's'));
printf("%c\n", *strchr(a, 's'));
printf("%x\n", strrchr(a, 's'));
printf("%c\n", *strrchr(a, 's'));
```

输出

```
62fe43
s
62fe4a
s
```

```
//自己实现strchr()函数
char *my_strchr(const char *s, int c)
{
    if (s == NULL)
        return NULL;
    while (*s != '\0')
    {
        if (*s == (char)c)
            return (char *)s;
        s++;
    }
    return NULL;
}
```

思考：如何实现strrchr?

使用子串查找函数：C07-04

◆子串查找函数：strstr

```
//在字符串中查询是否存在子字符串sub_str
//返回中第一次出现的位置指针
//如果不存在，则返回NULL。
char *strstr(char *s, char *sub_str);
```

◆C07_04：字符串替换

✓从标准输入读入数据中，每行最多包含一个字符串 "_xy_"。将输入行中的"_xy_"替换为"_abc_"，在标准输出上输出替换后的结果

◆问题分析

✓如何找到”_xy_”：strstr?

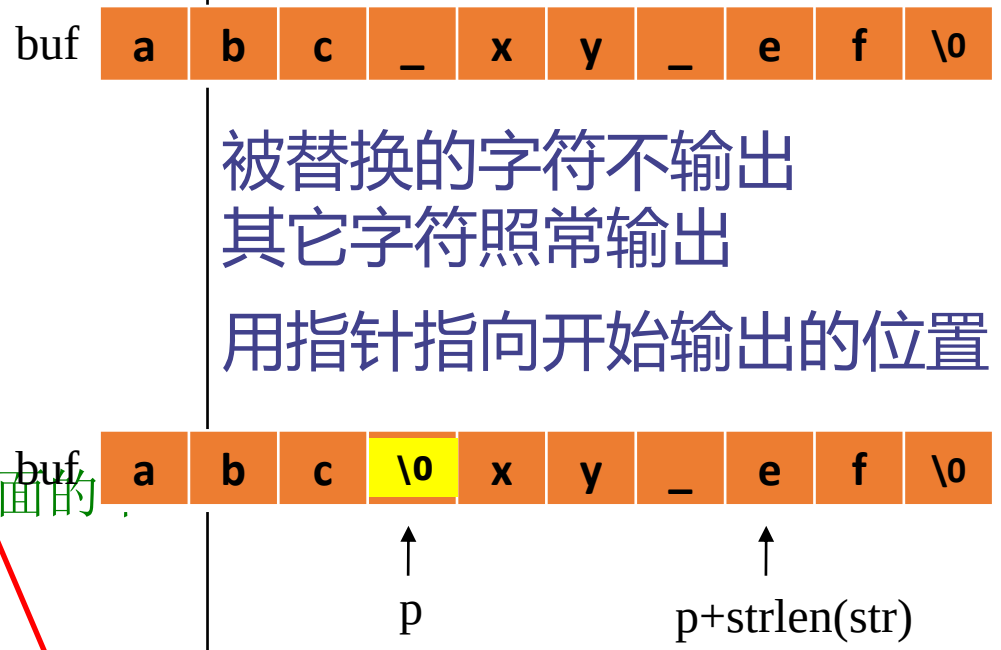
✓找到后，如何替换为“_abc_”：数组操作、指针运算?

C07-04：字符串替换

提示：BUFSIZ是一个常用的说明缓冲区大小的符号常量，实际数量取决于编译系统的版本和编译选项，常见的数值为512或4096

```
#include <stdio.h>
#include <string.h>
int main(){
    char buf[BUFSIZ], *p, *str = "_xy_";
    while (gets(buf) != NULL){//读入字符串
        p = strstr(buf, str);//字符串中查询子串
        if (p == NULL){//找不到子串，直接输出
            printf("%s", buf);
            continue;
        }
        *p = '\0';//将找到的位置设置为\0，从而输出前面的串
        //用%s输出时，通过buf只能输出前面的串
        //代码中直接输出_abc_子串
        //p+strlen(str)通过指针运算，输出_xy_后面的串
        printf("%s_abc_%s", buf, p + strlen(str));
    }
    return 0;
}
```

输出示例 **abc_xy_ef**



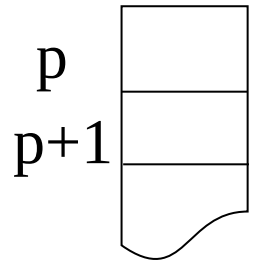
从子串 "_xy_" 的第一个字符将输入分为两部分

7.3 指针运算

- 针对指针类型，C语言提供了各类运算操作
- 加减运算
  - ✓指针与整数：加、减运算
  - ✓两个同类型的指针：减法运算
- 比较运算
  - ✓两个指向同一类型的指针，可进行 ==，>，<，!=等关系运算，即地址的比较
- 强制类型转换

指针与整数的加减运算

- 指针和整型量可以进行加减
  - ✓计算结果同指针所指对象类型有关
- 若p为指针，n为整数
  - ✓p+n: 将指针的值加上n个地址空间，返回后n个元素的地址
  - ✓p-n：将指针的值加上n个地址，返回前n个元素的地址
- 自增和自减运算符
  - ✓p++、++p
  - ✓p--、--p



指针与整数的加减运算

◆指针相加/相减的整数表示元素个数，而不是指针数值增量

✓相加/相减的整数**不是地址的绝对值**，而是以当前指针所指向**内容的长度为单位**进行运算

✓指针数值的改变是以其所指向的数据类型的长度为单位的

```
int a[10], *pi;
pi=a;
//int类型指针，以int长度为单位增加，+1即增加一个int类型的长度
pi=pi+1;
short s[10], *ps;
ps=s;
//short类型指针，以short长度为单位增加，+3即增加一个short类型的长度
ps=ps+3;
```

指针与整数的加减运算

- 后置自增/自减运算符的优先级高于取内容的优先级
- 前置自增/自减运算符的优先级与取内容的优先级相同

```
//等价于 *(p++)，即：
//先执行指针后置自增（指针+1）
//之后取内容（取的是自增之前指针所指向的内容）
*p++;

//先取指针的内容，再对内容+1
//指针本身没有改变
(*p)++;
```

示例：指针运算

```
int a[5] = {10, 11, 12, 13, 14};
int *pi;
pi = &a[1];                    // pi指向数组第2个元素11
pi++;                          // pi自增1，指向数组第第3个元素12
pi += 2;                       // pi加2，指向数组第第5个元素14
*(pi - 2) = *(pi - 4); // pi-2指向第3个元素12，pi-4指向第1个元素10
```

a[5] = {10, 11, 10, 13, 14};

```
int i, a[5] = {10, 11, 12, 13, 14};
int *pi = a;
//后置++优先级高，先执行指针后置++，再执行*pi
printf("%d\n", *pi++);
printf("%d\n", *(pi++));//运算规则同上
printf("%d\n", (*pi)++);//先执行*，后执行整数++
printf("%d\n", *++pi);//先执行指针前置++，后执行*
```

```
10
11
10
13
```

C07-05：利用指针运算实现字符串拷贝

```
//字符串拷贝，数组版本
char *strcpy(char dest[], char src[]){
  int i;
  for(i=0; (dest[i] = src[i]) != '\0'; i++)
     ;
  return dest;
}
```

```
//字符串拷贝，指针版本
char *strcpy(char *dest, char *src){
    char *d = dest;
    while((*d++ = *src++) != '\0')
        ;
    return dest;
}
```

两个指针不能相加，但可以相减

- 两个指针不能相加！没有物理含义
- 但两个同类型的指针可以相减，其结果为两指针间相差元素的个数

```
int *i_low, *i_hi, a[MAX_N];
double *d_low, *d_hi, d[MAX_N];
i_low = a;     i_hi = &a[2];
d_low = d;     d_hi = &d[2];
printf("i_hi = %d\ni_low = %d\n", i_hi, i_low);
printf("i_hi - i_low = %d\n\n", i_hi - i_low);
printf("d_hi = %d\nd_low = %d\n", d_hi, d_low);
printf("d_hi - d_low = %d\n\n", d_hi - d_low);
```

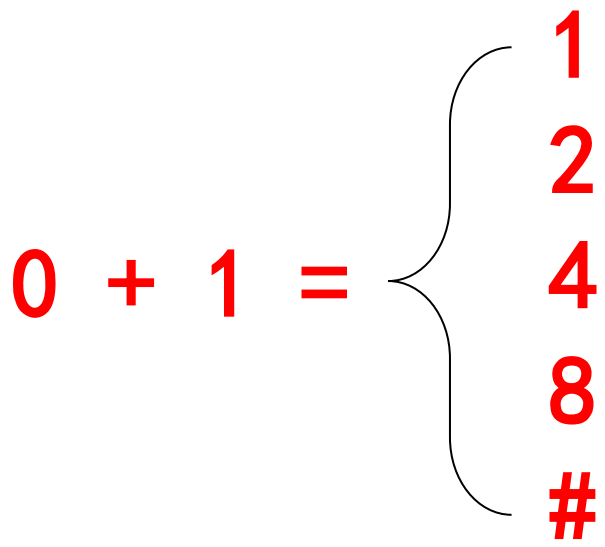
输出

```
i_hi = 6356704
i_low = 6356696
i_hi - i_low = 2

d_hi = 6356632
d_low = 6356616
d_hi - d_low = 2
```

704 – 696 = 2
32 – 16 = 2

指针与整数的加减



- 编程是一个数学问题?
- 艺术问题?
- 哲学问题?

指针的比较运算

- 两个指向同一类型的指针，可进行 ==，>，< 等关系运算，
  - ✓本质上就是地址值的比较
- 特殊的指针变量：NULL
  - ✓在指针未指向任何实际的存储单元时，或指针所指向的存储单元已经不存在时，需要将其标记为无效指针（空指针）
  - ✓为了表示指针的 0 在类型上不同于整数类型的 0，在C的标准头文件中定义了一个等于 0 的符号常量NULL
  - ✓数值为 0 或 NULL 的指针不指向任何内容
  - ✓数值 0 是唯一可以不将整数转换为指针类型而直接赋给指针变量的整数值

C07-06：多行数据读入并求平均值

◆C07-06：多行数据的平均值

✓从标准输入上读入N（0<N<1000000）行数据，每行含有若干个由空格符分隔的实数

✓在标准输出上输出每个输入行的行号、数据个数以及该行数据平均值，每行输入数据对应一个输出行，行号与数据个数间以冒号分隔，数据个数与数据平均值之间以空格符分隔

◆问题分析

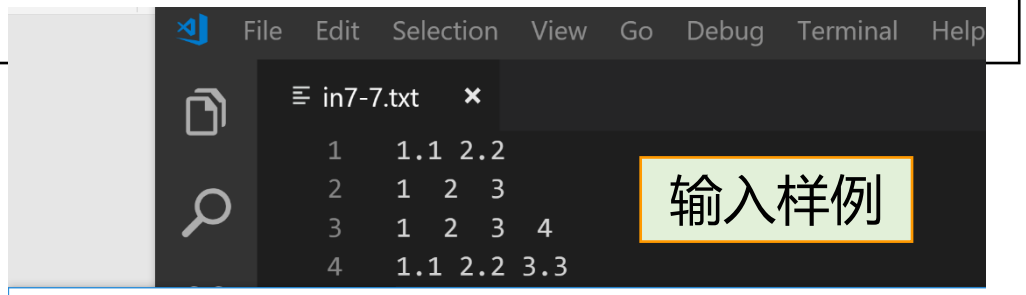
✓关键问题在于无法确定每行数据个数，无法使用scanf读入数据

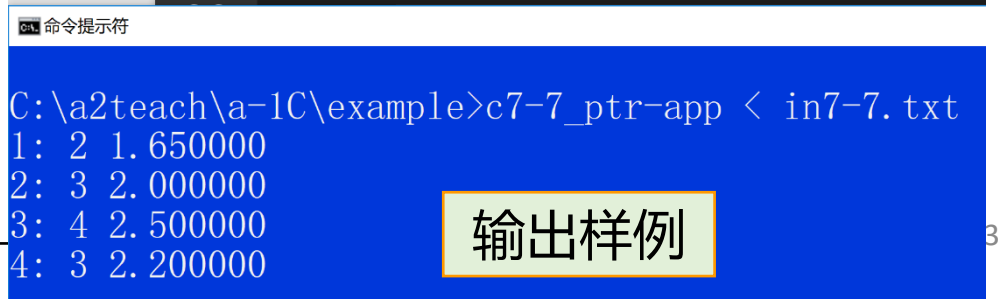
✓使用gets读入一行数据，之后进行字符串处理，根据空格，分隔每一个数字

C07-06：多行数据读入并求平均值

```c
#include <stdio.h>
int main()
{
    int i, n;
    double d, subsum;
    char buf[BUFSIZ], *p;
    //使用gets读入一行，返回NULL表示输入结束
    for (i = 1; gets(buf) != NULL; i++)
    {
        subsum = 0;
        //循环读入一个浮点数，直到返回NULL
        for (n=0, p=buf;
             (p=get_value(p, &d)) != NULL; n++)
            subsum += d;
        if (n > 0)
            printf("%d:%d %f\n", i, n, subsum / n);
    }
    return 0;
}
```

```c
//从字符串中读取一个浮点数，读入后指针后移
char *get_value(char *s, double *d){
    //过滤掉空白字符
    while (*s != '\0' && isspace(*s))
        s++;
    //从字符串中读入一个浮点数
    if (sscanf(s, "%lf", d) != 1)
        return NULL;
    //指针移到实数后的空白位
    while (*s != '\0' && !isspace(*s))
        s++;
    return s;
}
```





C07-07：子串逆置

◆C07-07：子串逆置

✓从标准输入上读入以空格分隔的字符串s和t，将s中首次与t匹配的子串逆置后再输出 s，当s中无与 t 匹配的子串时直接输出 s

输入样例：

```
abc123defg 123
abc123defg 234
```

输出样例：

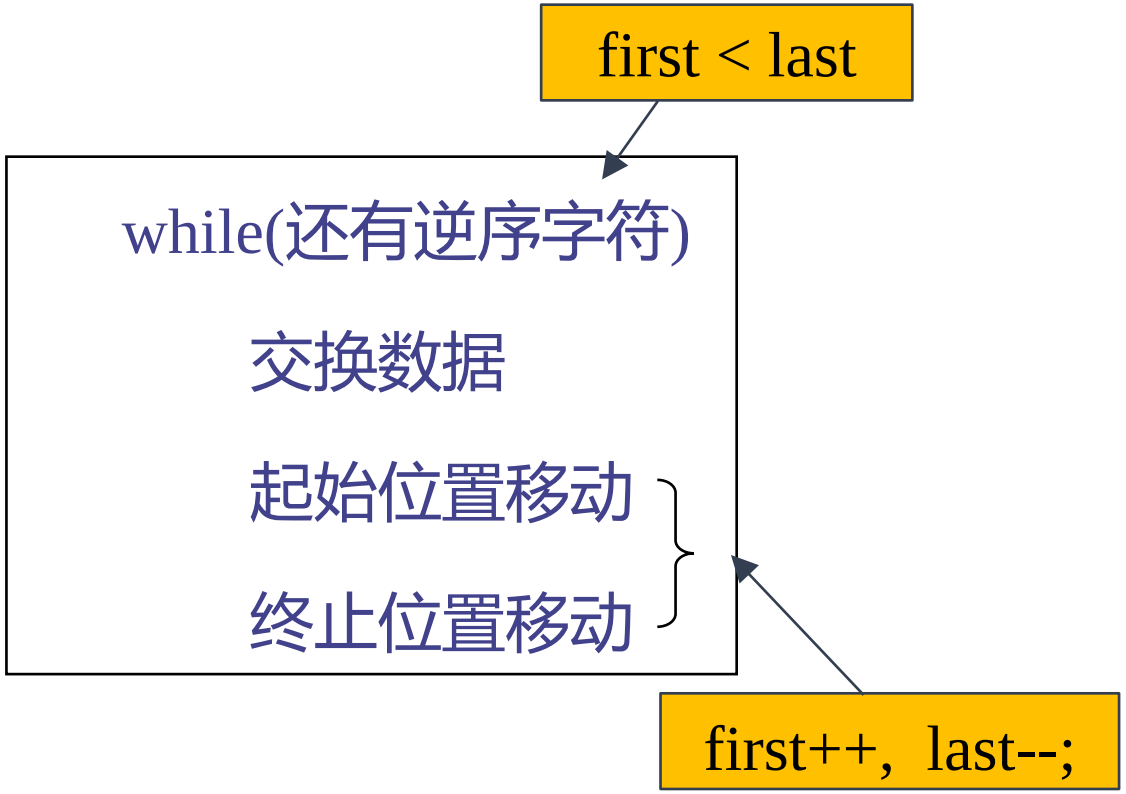
```
abc321defg
abc123defg
```

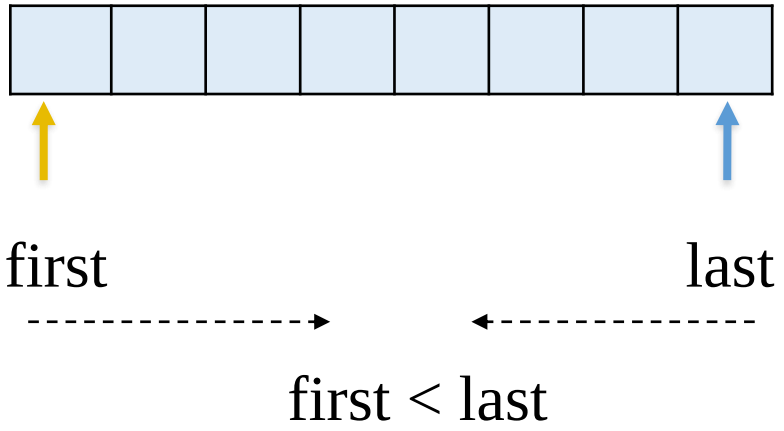
◆问题分析

✓使用strstr函数查询在主串中查询子串的位置

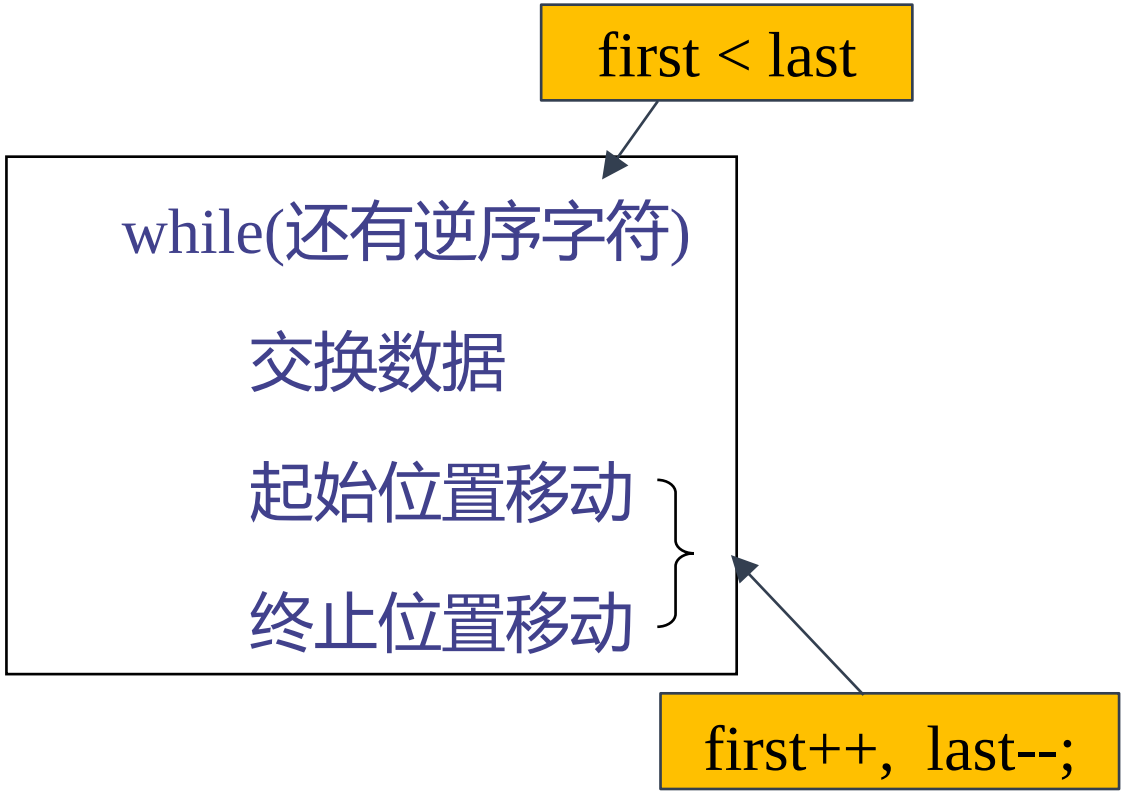
✓根据子串长度获得子串最后的位置（last），与最前面的位置（first）交换，first--、last++，依次交换

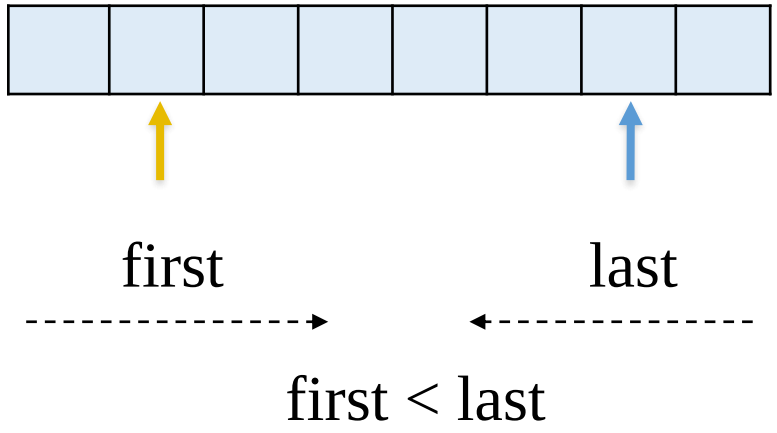
字符串逆序函数的实现思路



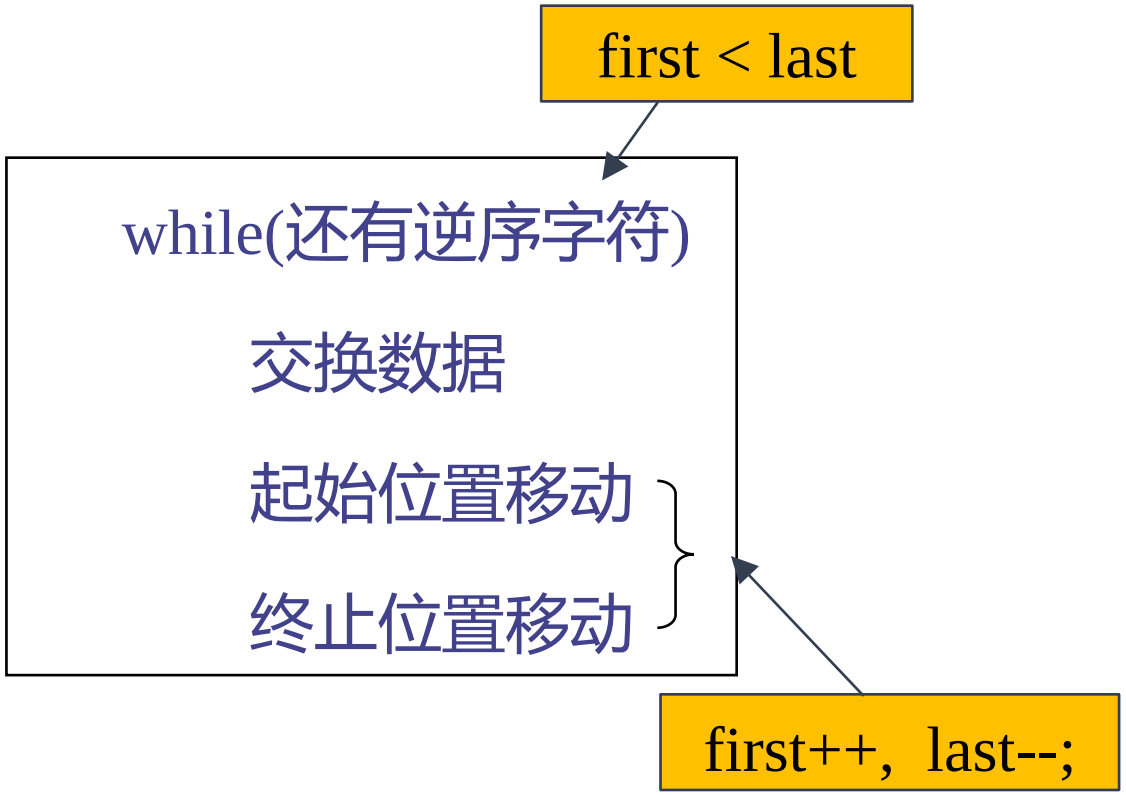


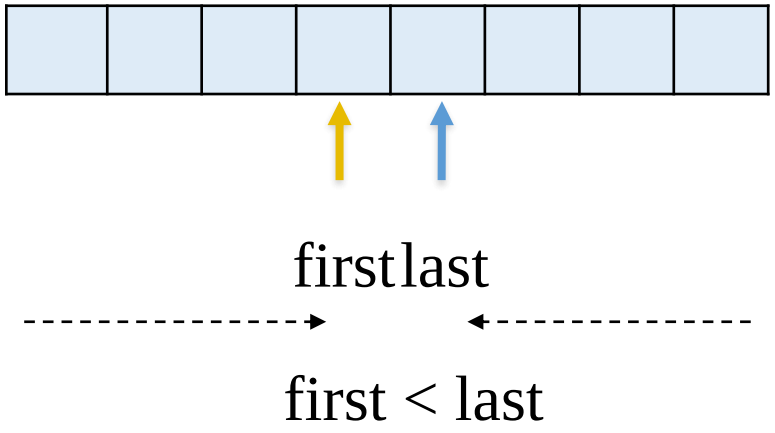
字符串逆序函数的实现思路



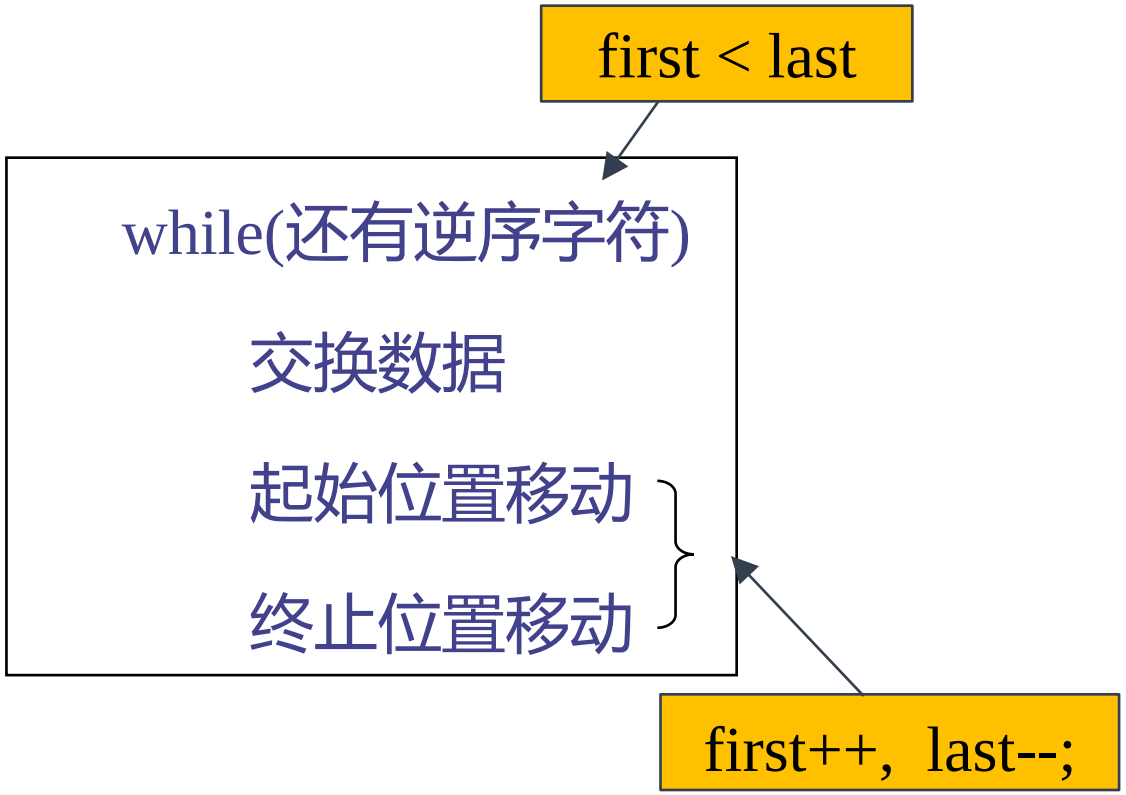


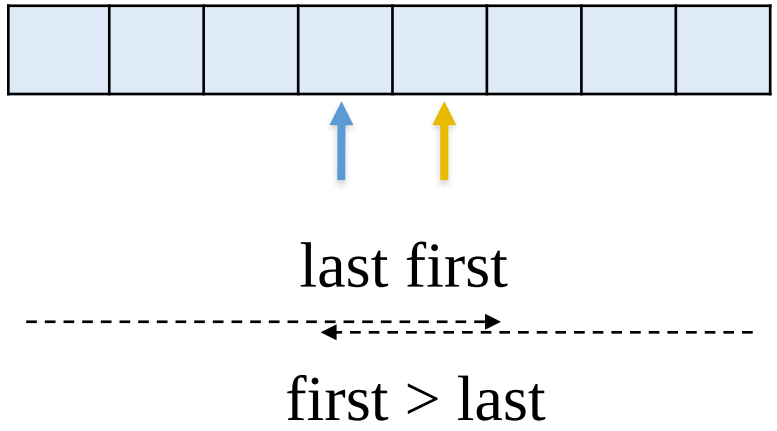
字符串逆序函数的实现思路





字符串逆序函数的实现思路





C07-07：字符串逆置

```c
#include <stdio.h>
#include <string.h>
void rev(char *, char *);
int main()
{
    char str[BUFSIZ], substr[BUFSIZ], *p;
    scanf("%s%s", str, substr);
    //使用strstr查询子串，返回子串在主串中的位置
    if ((p = strstr(str, substr)) != NULL)
        rev(p, p + strlen(substr) - 1);
    puts(str);
    return 0;
}
```

```c
//从first开始到last结束，逆置字符串
void rev(char *first, char *last)
{
    int tmp;
    //比较指针的值
    while (first < last)
    {//交换字符
        tmp = *last;
        *last = *first;
        *first = tmp;
        first++;//从前往后
        last--;//从后往前
    }
}
```

指针的强制类型转换

- 指针类型的强制转换
  - ✓指向不同类型内容的指针类型也不相同
  - ✓当需要不同类型指针互相赋值时，可通过强制类型转换改变对指针类型的解释，以保证所需操作在语法上的正确性
    - ➢在指针前加上小括号括起来的目标类型
- 常见的非法指针运算包括：
  - ✓指针间的加、乘、除，指针加减浮点数
  - ✓指针移位操作
  - ✓指针的位运算
  - ✓不同类型指针间的直接赋值

```
int *ia,*ip, n,
arr[3][6];
short s, sa[16], *id;
ip = (int *) sa;
ia = (int *) arr;
id = (short *) &n;
ip = (int *) id;
```

动态内存分配

- 定义指针后，仅仅声明了一个地址变量
- 如果要使用该指针，需要把该指针指向一个已有的合法地址空间（即要初始化指针）
  - ✓之前的指针都是指向已有的数据变量、数组等
  - ✓声明指针后，也可以根据需要，主动申请新的数据空间
- <stdlib.h>头文件中提供了内存分配和释放函数

```
//分配size字节的内存空间，返回首地址
void* malloc(size_t size);
//释放由malloc分配的内存空间
void free(void* mem);
```

- 通用指针类型：void*
  - ✓具有void*类型的指针可以赋给任何类型的指针变量，具有void*类型的指针变量可以接受和保存任意类型的指针

使用malloc动态申请内存空间

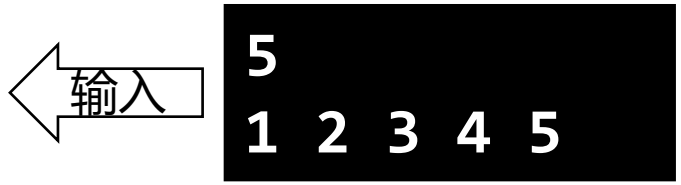
```
char *s1, *s2;
int *intptr;
//s1指向大小为32个字节（字符）的空间
s1 = (char *) malloc(32);
//根据字符串p的大小分配内存空间
s2 = (char *) malloc(strlen(p)+1);
//分配10个整数大小的空间，使用sizeof计算int所占的空间大小
intptr = (int *) malloc( sizeof(int) *10);
…
//必须使用free释放分配的空间
free(s1);
free(s2);
free(intptr);
```

使用malloc动态申请空间

- 普通变量由编译器自动管理其内存空间
  - ✓普通局部变量，函数结束后即销毁，空间释放
  - ✓全局变量、静态变量在程序结束后，也自行销毁
- 由malloc申请的动态空间，原则上不随函数、程序结束而自行释放，必须要用free函数来释放
  - ✓由malloc申请的动态空间不及时释放，是造成许多软件出现内存泄漏的主要原因！
  - ✓内存泄漏(memory leak)：指软件在长时间运行过程中造成内存越来越少，最终可能导致系统内存耗尽而导致软件性能下降或不能使用的现象

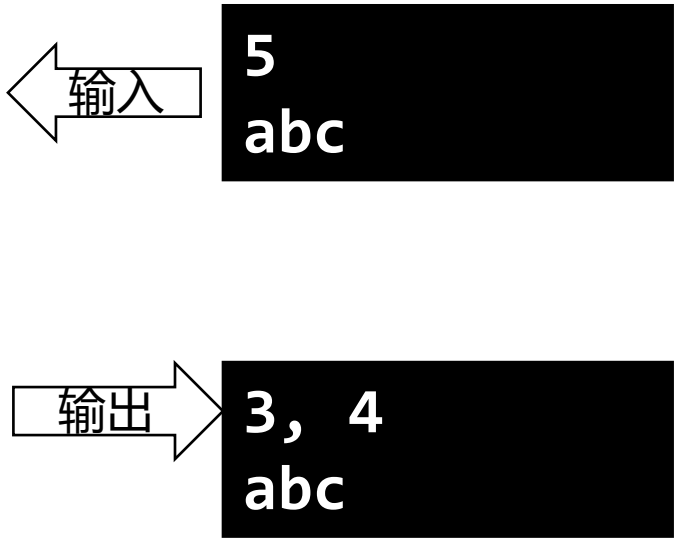
示例：使用malloc申请空间

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *intptr, n, i;
    scanf("%d", &n);
    //根据输入变量n，申请n个整数空间，可当数组使用
    intptr = (int *) malloc(sizeof(int)*n);
    for(i=0; i<n; i++)//使用指针操作
        scanf("%d", intptr+i);
    for(i=0; i<n; i++)//按照数组操作
        printf("%d ", intptr[i]);
    printf("\n");
    free(intptr);//释放内存空间
    return 0;
}
```



示例：使用malloc申请空间

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char *s;
    int n, i;
    scanf("%d", & n);
    //不是用数组，利用malloc动态内存分配
    s = (char *) malloc(n);
    scanf("%s", s);
    printf("%d, %d\n", strlen(s), sizeof(s));
    for (i = 0;  * (s + i) != '\0';  i++)
        printf("%c", * (s + i));
    printf("\n");
    free(s);
    return 0;
}
```



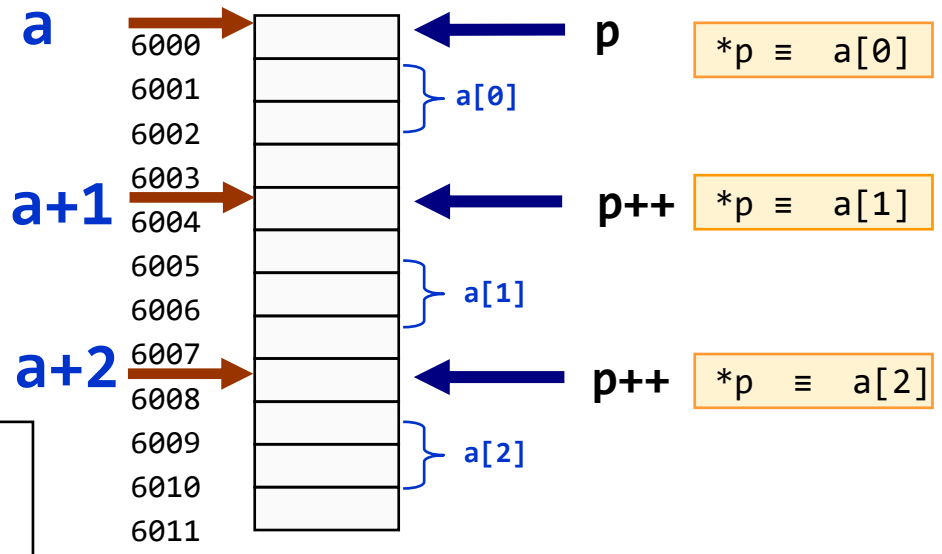
7.4 指针与数组

- 数组是提前分配好的一组连续空间，数组名为指向该连续空间的首地址，本质上就是指针
  - ✓数组名这个指针地址不能被修改（指针常量，不能被赋值），只能指向当前数组空间；而普通指针可以修改
  - ✓数组名可以直接当指针，赋值给对应的指针类型变量
- 如果指针指向的是一组连续内存空间，也可直接当数组使用

指针与一维数组

◆可以将数组名直接赋值给指针

```
int *p, a[10];
p = a; //数组名直接赋值给指针变量
```



◆指向连续空间指针，可以当数组使用

```
int *ptr, i;
//分配10个整数空间，可当数组使用
ptr = (int *)malloc(sizeof(int) * 10);
for(i = 0; i < 10; i++)//以数组方式使用
    ptr[i] = i;
```

```
p = a;
```

✓

```
a = p ;
a++;
```

✗

通过指针和数组的方式访问

- 使用数组，以“[下标]”的方式访问内容
- 使用指针，以“*(指针运算)”的方式访问内容

```
int a[10], *p=a, i;
//以下访问方式是等价的
 a[i]    //数组+下标
 *(a+i) //数组+偏移量
 p[i]    //指针+下标
*(p+i)  //指针+偏移量
```

指针和一维数组

```c
//数组版
#include <stdio.h>
int main()
{
    int  a[10], i;
    for (i=0; i<10; i++)
        scanf("%d", &a[i]);
    for (i=0; i<10; i++)
        printf("%d ", a[i]);
    return 0;
}
```

```c
//指针版
#include <stdio.h>
int main()
{
    int  a[10], *p, i;
    for (p=a; p<(a+10); p++)
        scanf("%d", p);
    for (p=a; p<(a+10); p++)
        printf("%d ", *p);
    return 0;
}
```

◆注意：由于数组名是常量指针，因此不能执行修改操作

✓对于数组a，for循环头不能写成：for(p=a; a<(p+N); a++) ?

通过指针和数组的方式访问数据

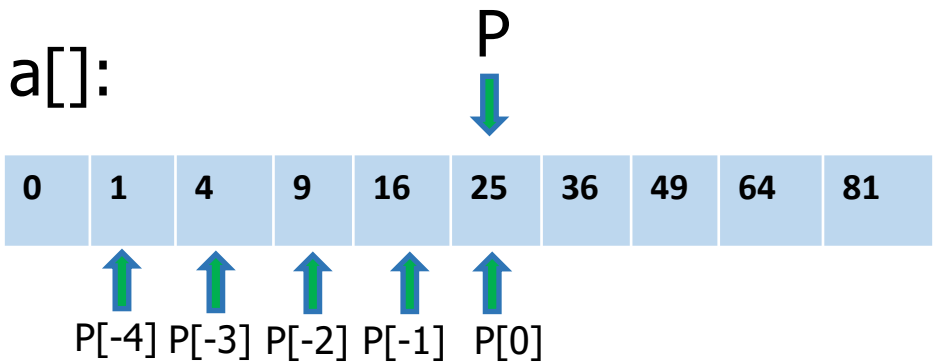
```
#include <stdio.h>
int main()
{
    int a[] = {1, 2, 3, 4, 5, 6};
    int *p;
    p = a;//指针指向数组
    p[0] = p[2] + p[3];//指针也可以直接按数组使用
    *(p + 3) += 2;  //也可以通过指针运算访问
    //数组a也可以按指针使用，但不能修改
    printf("%d, %d\n", *p, *(a + 3));
    …
```

输出 7, 6

利用指针实现负下标的访问

- 可以利用指针与数组的可互换性来支持数组的负数下标
  - ✓语法合法
  - ✓语义是否合法取决于是否越界

```
int a[10], *p = &a[5];
int i;
for (i = 0; i < 10; i++)
    a[i] = i * i;
for (i = 0; i < 5; i++)//负下标
    printf("%d  ", p[-i]);
```




```
25  16  9  4  1
```

指针与字符串

- 字符串可用带\0的一维字符数组实现
- 可以使用char*类型的指针指向字符串
- 同理，char*类型是指针，能否存储字符串取决于：
  - ✓是否有足够的内容空间
  - ✓是否以\0结束

```
char *char_ptr, word[20];
//正确，  把字符常量的首地址赋给指针变量
char_ptr = "point to me";
//错误，  word是常量，正确做法为: strcpy(word, "…");
word = "you can't do this"; //错误实现！！
```

理解采用数组和指针方式实现的字符串

◆注意字符数组和字符指针的区别

✓数组已有变量空间

✓指针不一定有空间，若指向某一个空间，主要该空间的性质

```
//a是数组，初始化为hello，长度为5+1
//p是指针，指向一个字符串字面量
char a[]="hello", *p="hello";
//修改数组的某个元素的值
a[0] ='c';
//修改指针的内容，但此处由于p指向的字符串字面量，内容无法修改
*p = 'c'; //代码错误，不能修改!
```

理解字符指针初始化的含义

◆执行字符指针变量赋值语句时发生了什么?

```
char *p = "OK";
```

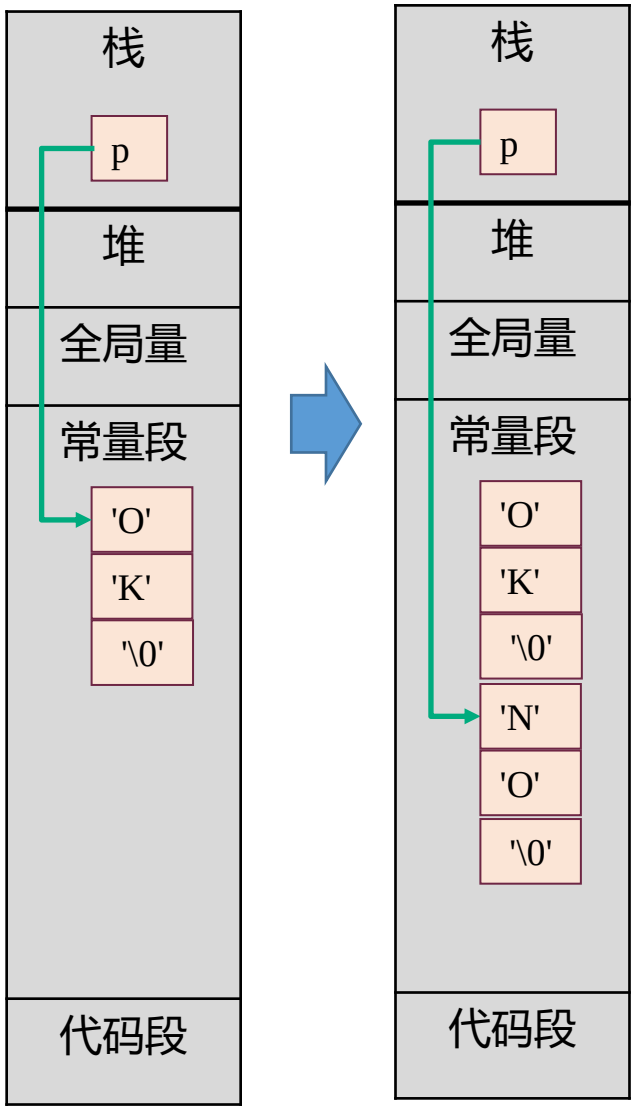
✓在常量区申请合适的内存空间，将字符串"OK"依次存入内存单元

✓在字符串尾部添加 '\0'

✓返回常量字符串中首字符地址，将其赋值给字符指针p

✓如果对p赋新值，如：`p = "No";`

➢不是修改原字符串值，而是让指针变量p指向新字符串的首字符



理解指针类型与数组类型的差异

- 数组是一片连续的存储空间，而指针只是一个保存地址的存储单元
- 未经正确赋值之前不指向任何合法的存储空间，不能通过它进行任何数据访问

```
double d,*dp;
*dp = 5.678;
*dp = 100;
```

运行时错误

```
char *string;
scanf("%s", string);
strcpy(string, "Hello");
```

- dp：未指向合法空间，是野指针
  - ✓当要向一个指针指向的空间赋值时，一定要检查有没有给这个指针分配空间
  - ✓建议：定义指针时如果没有初始化，就让它指向NULL，即空指针，表示不能操作空指针所指向的空间。

理解指针类型与数组类型的差异

- 数组名是一个指针常量而不是变量，是与一片固定的存储空间相关联的
  - ✓可以对数组元素赋值而不可以对数组变量本身赋值
- 指针变量本身是一个变量，可以根据需要进行赋值，从而指向任何合法的存储空间
- 通过数组所能访问的数据的数量在数组定义时就已确定
- 通过指针所能访问的数据的数量取决于指针所指向的存储空间的性质和规模

小结

- 理解指针的含义
  - ✓指针本身所代表的地址
  - ✓通过该地址能够访问到的数据内容
- 使用指针变量
  - ✓取地址&、取内容*
  - ✓指针作为函数参数、返回指针类型的函数
- 指针运算
  - ✓指针与整数的加减、指针比较
  - ✓强制类型转换和动态存储分配
- 指针与数组的区别和联系
  - ✓数组是已经分配好空间的指针常量
  - ✓普通指针是一个变量，可以指向任何同类型的内容空间