



中國人民大學  
RENMIN UNIVERSITY OF CHINA

# 第9讲 指针

余力

buaayuli@ruc.edu.cn

# 回顾一个问题 (1)

---

- 将两个整数x和y的值进行调换
  - 例子:  $x=10; y=20 \rightarrow x=20; y=10$
- 设计一个函数swap实现这一功能

```
void swap (int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}
```

## 回顾一个问题 (2)

---

```
void swap(int x, int y);  
int main() {  
    int x=10, y=20;  
    printf("Before swapping: x=%d, y=%d\n", x, y);  
    swap(x,y);  
    printf("After swapping: x=%d, y=%d\n", x, y);  
}
```

Before swapping: x=10, y=20

After swapping: x=10, y=20

# 指针部分内容提要

---

1 指针的基本概念

2 指针与数组

3 指针与字符串

4 指针与函数

5 指针与结构体



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

# 1. 指针的基本概念

---

# 什么是指针 (1)

---

## ■ 指针是一类变量

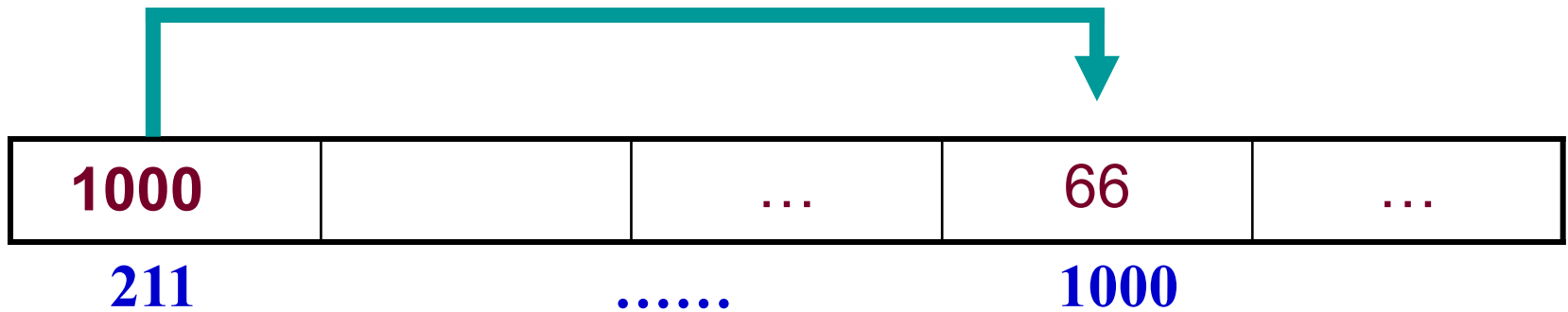
- 学过的变量：整型int、字符型char，等等
- 指针也是一类变量，本质并无不同

## ■ 指针的取值是内存的地址

- 学过的变量取值
  - 整型变量的取值是一个整数，如1024
  - 字符型变量的取值是字符，如' A'
- 指针的取值是一个内存地址！

# 什么是指针 (2)

- 指针的取值是内存的地址



- 生活中的例子
  - 将信箱想象成内存(Memory)
  - 信箱221号有信息：去1000号找重要数据
  - 信箱1000号有信息：重要数据66

# 指针的声明 (1)

---

- 强调：指针只是一类特殊的变量
- 与普通变量声明的“同”
  - 变量名：这与一般变量取名相同，由英文字符开始
- 与普通变量声明的“异”
  - 指针变量的类型：是指针所指向的变量的类型，而不是自身的类型。
  - 指针的值是某个变量的内存地址。



# 指针的定义 (2)

- 声明的格式:

类型标识符 \*变量名;

- 示例:

- `int *p, *q;`      // 指向整数类型变量的指针
- `float *point;`      // 指向float型变量的指针
- `double *pd;`      // 指向double型变量的指针
- `char *pc;`      // 指向char型变量的指针

# 指针的初始化

---

```
int *p = NULL;
```

## ■说明:

- NULL在头文件中定义，是符号化的常量0，是唯一的一个允许赋值给指针的整数值
- 表示指针不指向任何内存地址
- 防止其指向任何未知的内存区域
- 避免产生难以预料的错误发生

## ■把指针初始化为NULL是**好习惯**

# 指针的赋值

- 将一个内存地址装入指针变量

## 取址运算符&

- 例如：

```
int a=66;           // 定义整型变量 a, 赋初值66
```

```
// 定义p,q指针变量, 赋初值为0
```

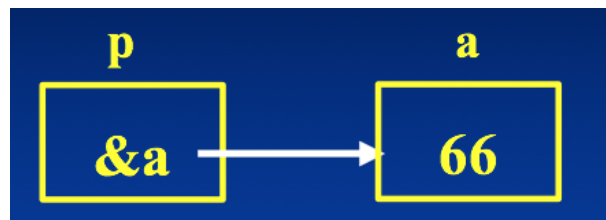
```
int *p=NULL, * q=NULL;
```

```
p = &a;             //将变量 a 的地址赋给 p
```

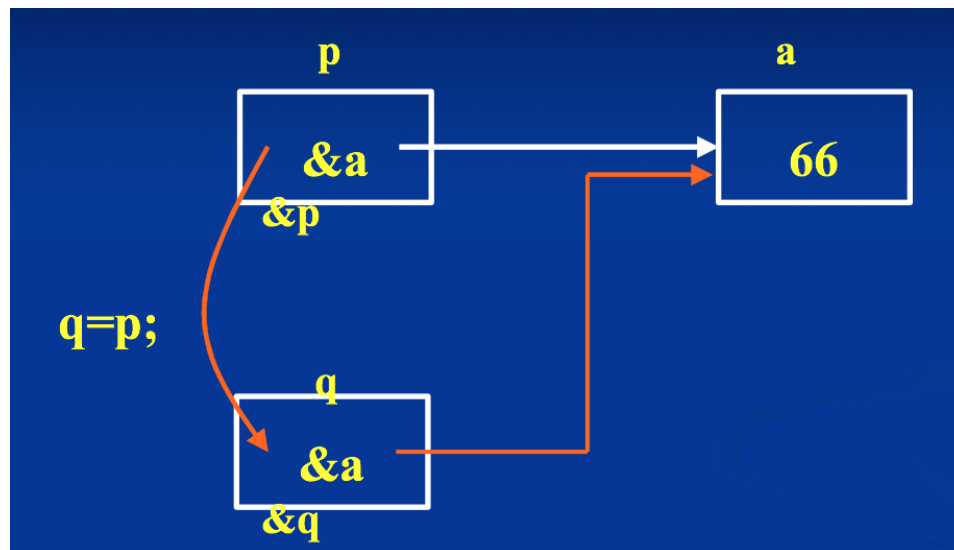
```
q = p;              // 将 p 的值赋给 q
```

# 画图理解指针

- $p = \&a;$       //将变量  $a$  的地址赋给  $p$



- $q = p;$



# 指针的赋值

---

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

```
int main() {
```

```
    int a[5]={0,1,2,3,4}; //定义数组，赋初值
```

```
    int *p1=NULL,*p2=NULL; //定义指针变量
```

```
    p1=&a[1]; //赋值给指针变量，让p1指向a[1]
```

```
    p2=&a[2]; //赋值给指针变量，让p2指向a[2]
```

```
    printf("%d,%d\n", *p1, *p2);
```

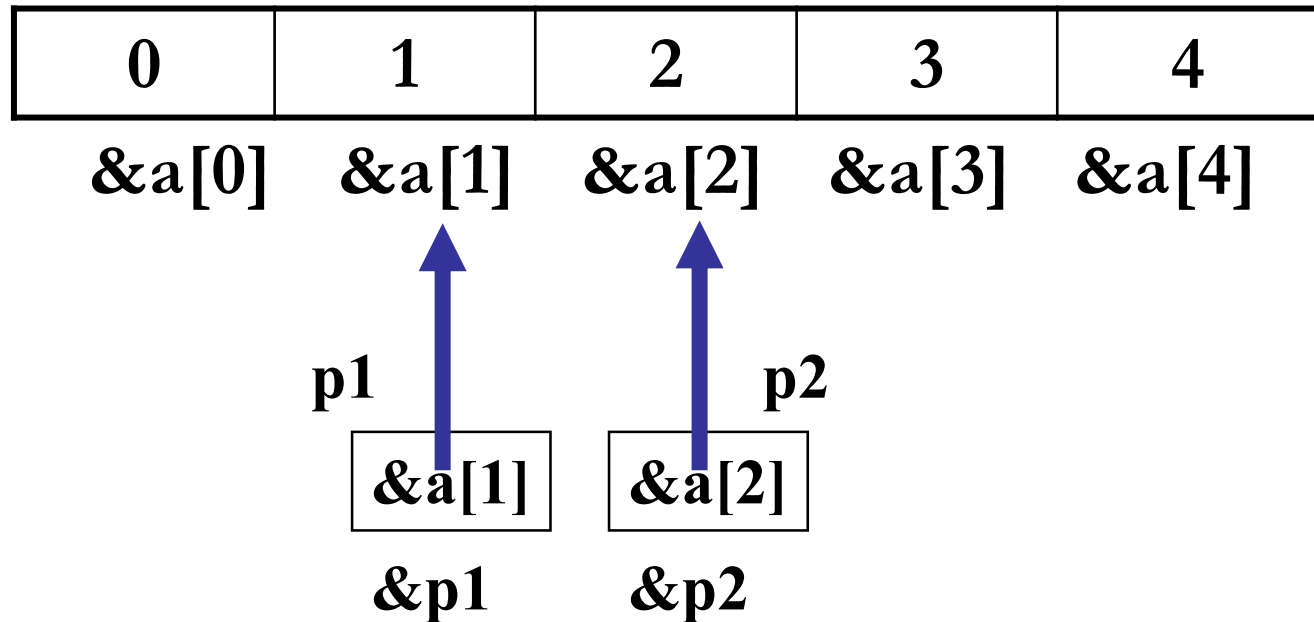
```
    printf("%d,%d\n", p1, p2);
```

```
    //输出a[1]和a[2]
```

```
    return 0;
```

```
}
```

# 指针的赋值



- $p1$  和  $p2$  分别指向  $a[1]$ ,  $a[2]$ , 这里
- $\&$  —— 取地址运算符
- $*$  —— 指针运算符 (间接访问运算符)
- $*p1$  —— 间接访问  $p1$  所指向的内存单元, 输出  $a[1]$  的值
- $*p2$  —— 间接访问  $p2$  所指向的内存单元, 输出  $a[2]$  的值

# 用指针实现swap函数

```
#include <stdio.h>
void interchange(int * u, int * v);
int main(void) {
    int x = 5, y = 10;
    printf("Originally x = %d and y = %d.\n", x, y);
    swap(&x,&y); /* send addresses to function */
    printf("Now x = %d and y = %d.\n", x, y);
    return 0;
}

void swap (int * u, int * v) {
    int temp;
    temp = *u;
    *u = *v;
    *v = temp;
}
```

```
void swap (int u, int v) {
    int temp;
    temp = u;
    u = v;
    v = temp;
}
```

```
int* temp;
temp = u;
u = v;
v = temp;
```

行吗?

# 指针的指针

## ■ 可以定义指向指针的指针

- 二级指针，例子：int \*\*q
- 三级指针，例子：int \*\*\*r

```
int var = 1025;
int *p = &var;
int **q = &p;
int ***r = &q;
printf("*p = %d\n", *p);
printf("*q = %d\n", *q);
printf("**q = %d\n", **q);
printf("***r = %d\n", **r);
printf("***r = %d\n", ***r);
***r = 10;
printf ("var = %d\n", var);
**q = *p + 2;
printf ("var = %d\n", var);
```





中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

## 02. 指针与数组

---

# 指针 vs. 数组下标

---

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

```
int main() {
```

```
    int a[5]={1,3,5,7,9};
```

```
    int *p; //定义指针变量
```

```
    int i; //定义整型变量
```

```
    p=a; //赋值给指针变量，让p指向a数组
```

```
    for(i=0; i<5; i++) {
```

```
        printf("a[%d]=%d\n", i, *p); //输出a数组元素的值
```

```
        p++; //指针变量加1
```

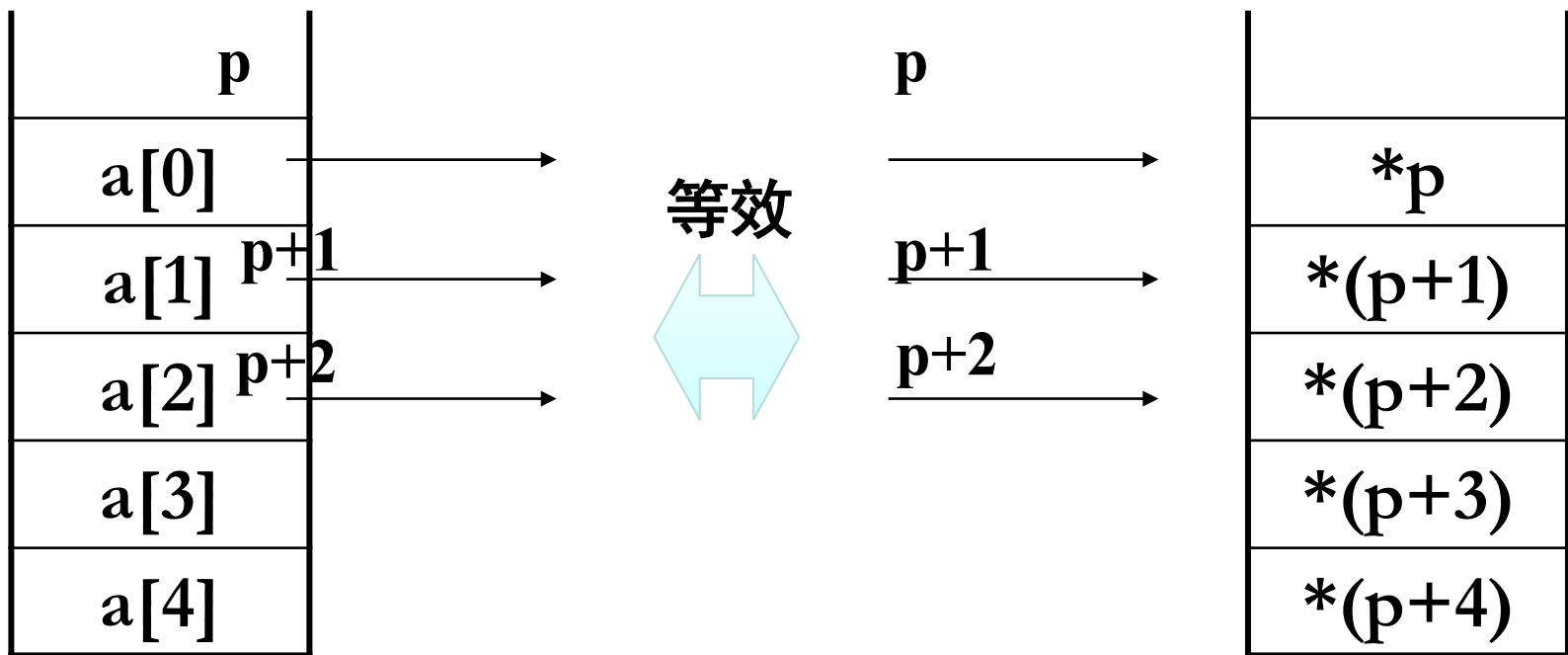
```
    }
```

```
    return 0;
```

```
}
```

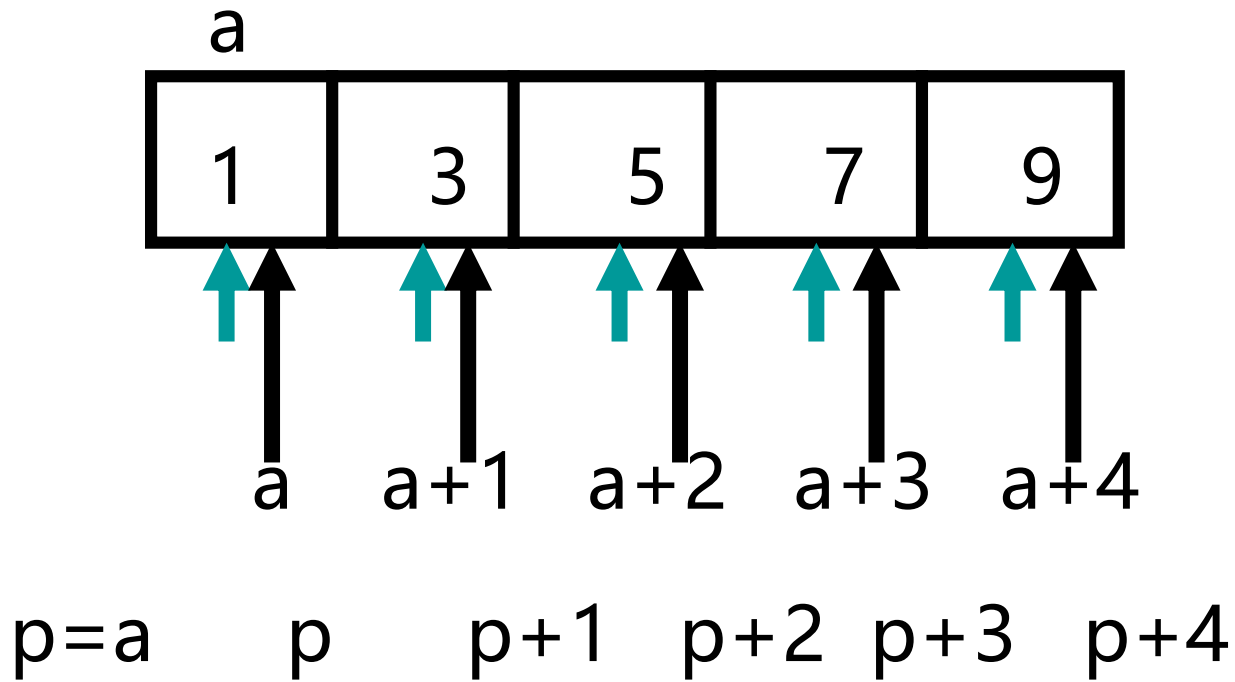
## 说明

- (1)  $p=a$ ; 这里数组名作为数组的起始地址, 即 $a[0]$ 的地址。  
因此  $p=a$  等效于  $p=\&a[0]$ ;
- (2)  $p=p+1$ ; 如 $p$ 指向 $a[0]$ , 则 $p=p+1$ 之后,  $p$ 指向 $a[1]$
- (3) 如果 $p=a$  等效于  $p=\&a[0]$ ;  
则  $p=a+4$  等效于  $p=\&a[4]$ ;



```
#include <stdio.h>           //预编译命令

int main()                   //主函数
{
    //函数体开始
    int a[5]={1,3,5,7,9};    //定义数组, 赋初值
    int *p, i=0;
    for(p=a; p<a+5;p++)      //赋值给指针变量, 让p指向a数组
    {
        //循环体开始
        printf("a[%d]=%d\n", i, *p); //输出a数组元素的值
        i++;                  //让i加1
    }
    //循环体结束
    return 0;
}
//函数体结束
```



数组名是一个常量指针，指向该数组的首地址，例

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

```
int main( )
```

```
{
```

```
    char *p=NULL;           // 定义指向字符类型的指针变量p
```

```
    char s[] = "abcdefgh" ; // 定义字符数组，并赋值
```

```
    p=s;                     // 数组名是一个常量指针，
```

```
                             // 它指向该数组首地址
```

```
    while(*p != '\0')       // 当p所指向的数组元素不为'\0' 时
```

```
    {
```

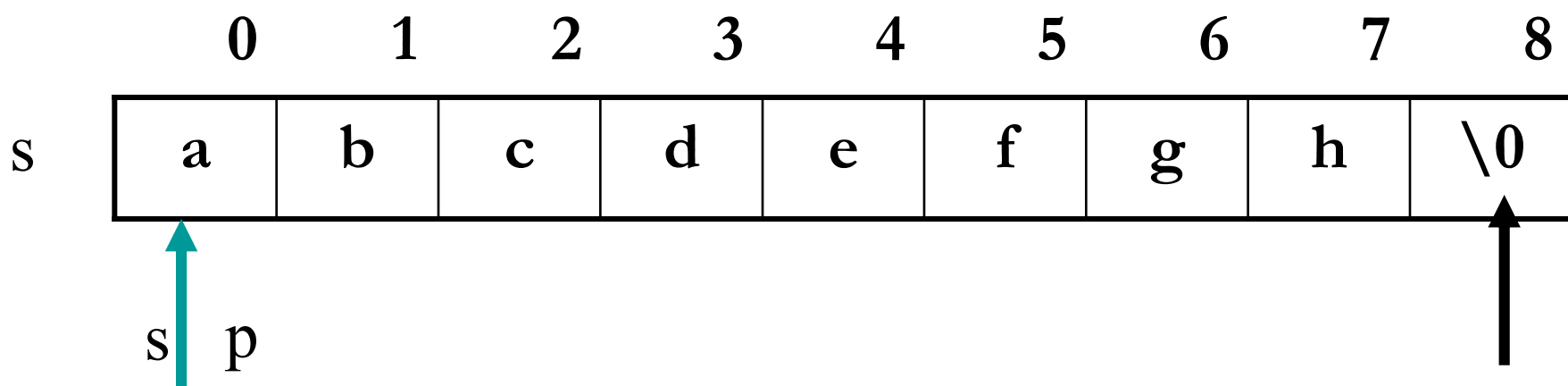
```
        p++;                // 让指针加1
```

```
    }
```

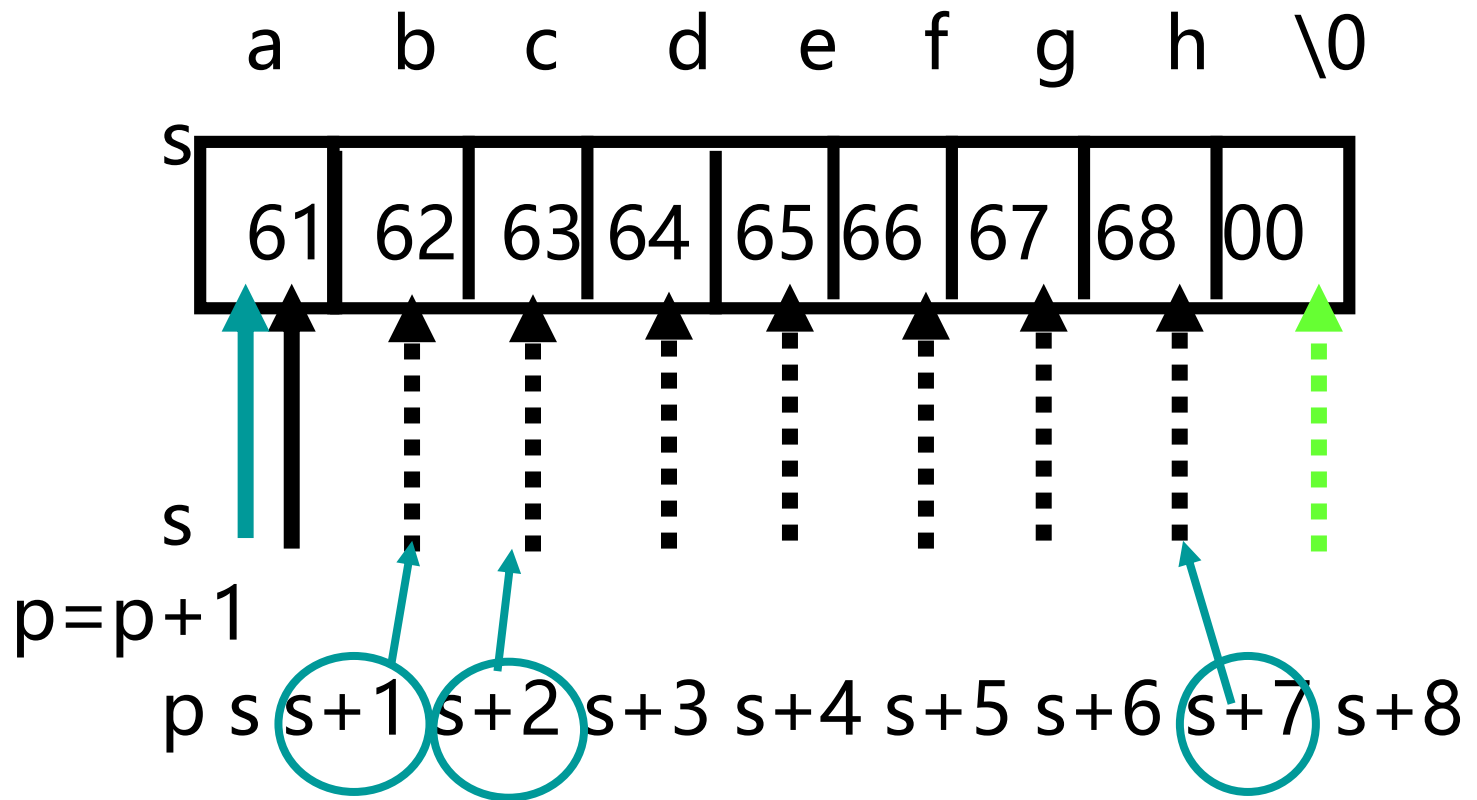
```
    printf("字符串长度为%d\n", p-s); // 输出字符串长
```

```
    return 0;
```

```
}
```



图中数组的首地址是 `s[0]` 的地址，即 `&s[0]`。s 可看作是指向 `s[0]` 的指针。s 是不会动的，是常量指针。



$$p = s + 8$$

$$p - s = 8$$

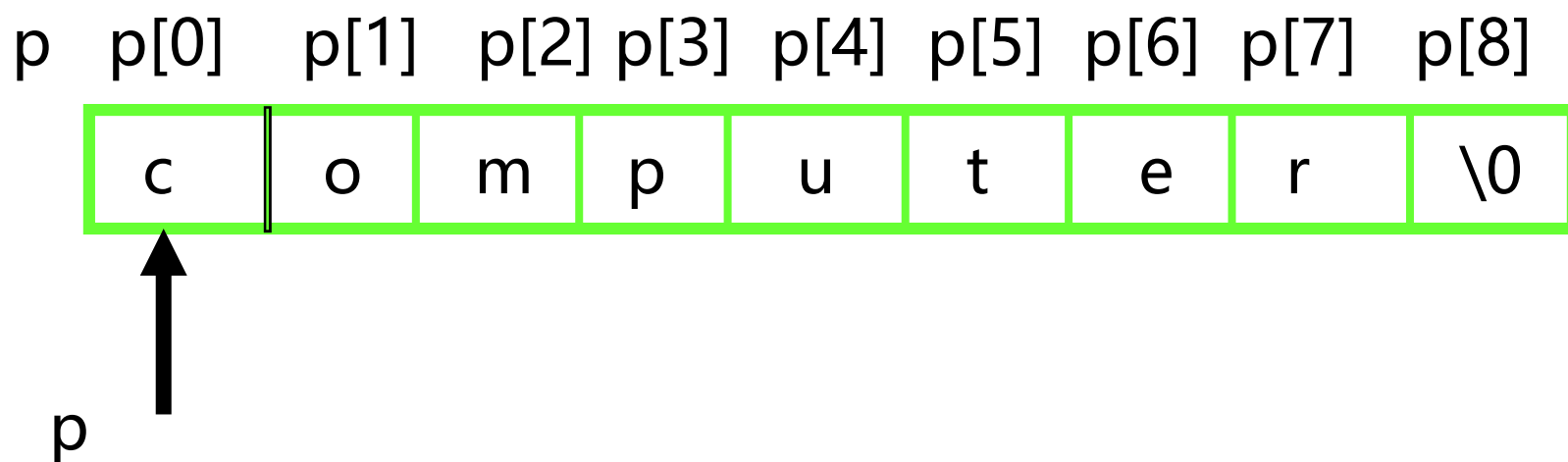


数组名是一个常量指针，指向该数组的首地址，例

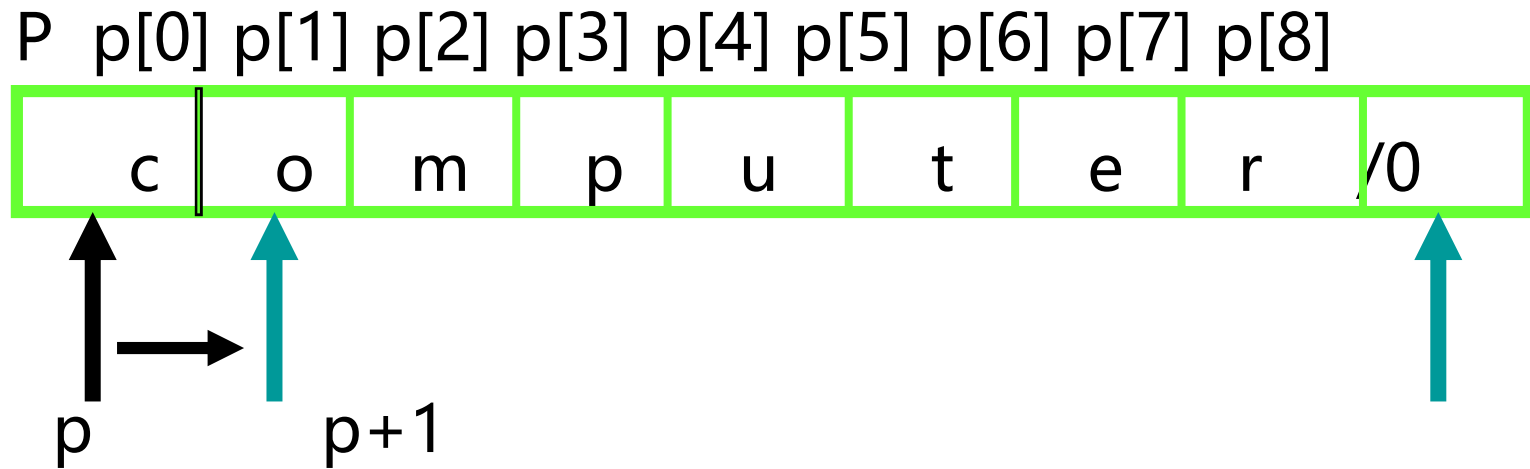
```
#include <stdio.h>           //预编译命令
int main()                   //主函数
{                             //函数体开始
    char shuzi[]="987654321"; //定义数组，
                                // 赋初值为数字字符串
    char *p=&shuzi[8];       //让指针p指向shuzi[8]元素，
                                // 该处是字符 '1'
    do                        // 直到型循环
    {                         // 循环体开始
        printf("%c", *p);    // 输出一个字符，该字符由p指向
        p--;                 // 让p减1
    }                        // 循环体结束
    while (p>=shuzi);        // 当p>=shuzi时，继续循环
    printf("\n");             // 换行
    return 0;
}
```

**针对实现数组逆向输出**

```
#include <stdio.h>
int main()
{
    int i;
    char *p;
    p="computer";
    printf("%s\n", p);
    for (i=0; i<8; i++)
    {
        printf("%c", p[i]);
    }
    cout<<endl;
    while(*p)
    {
        printf("%c", *p);    //输出p所指向的字符
        p++;                //指针变量值加1
    }
    cout<<endl;
    return 0;
}
```



数组名是常量指针，`p`可理解为这个字符数组的名字。



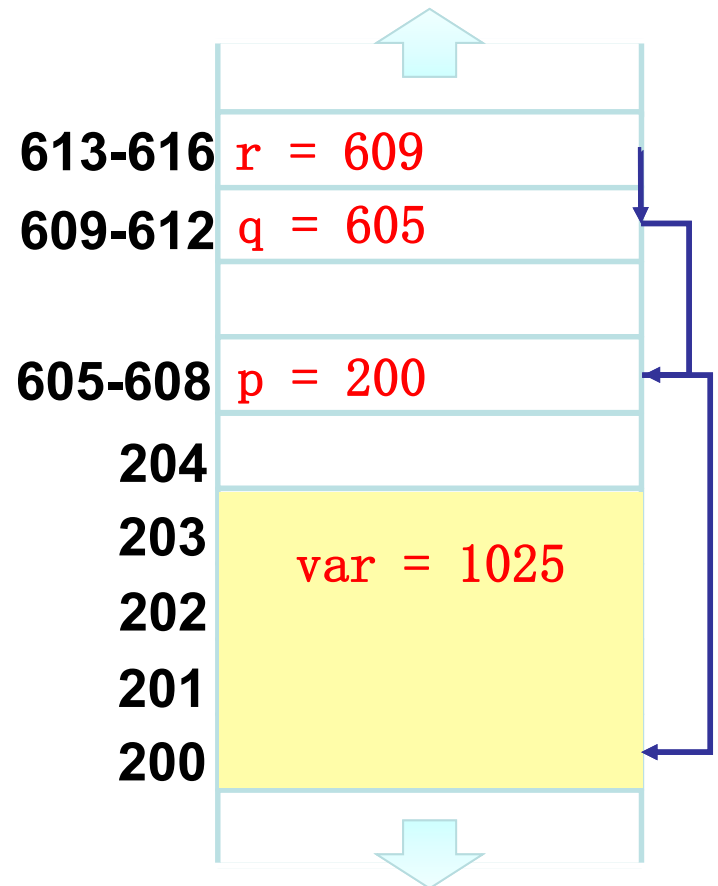
`*p == c, *(p+1) == o, *(p+2) == m`

# 指针的指针

## ■ 可以定义指向指针的指针

- 二级指针，例子：int \*\*q
- 三级指针，例子：int \*\*\*r

```
int var = 1025;
int *p = &var;
int **q = &p;
int ***r = &q;
printf("*p = %d\n", *p);
printf("**q = %d\n", **q);
printf("***q = %d\n", ***q);
printf("***r = %d\n", ***r);
printf("****r = %d\n", ****r);
***r = 10;
printf ("var = %d\n", var);
**q = *p + 2;
printf ("var = %d\n", var);
```



# 指针数组

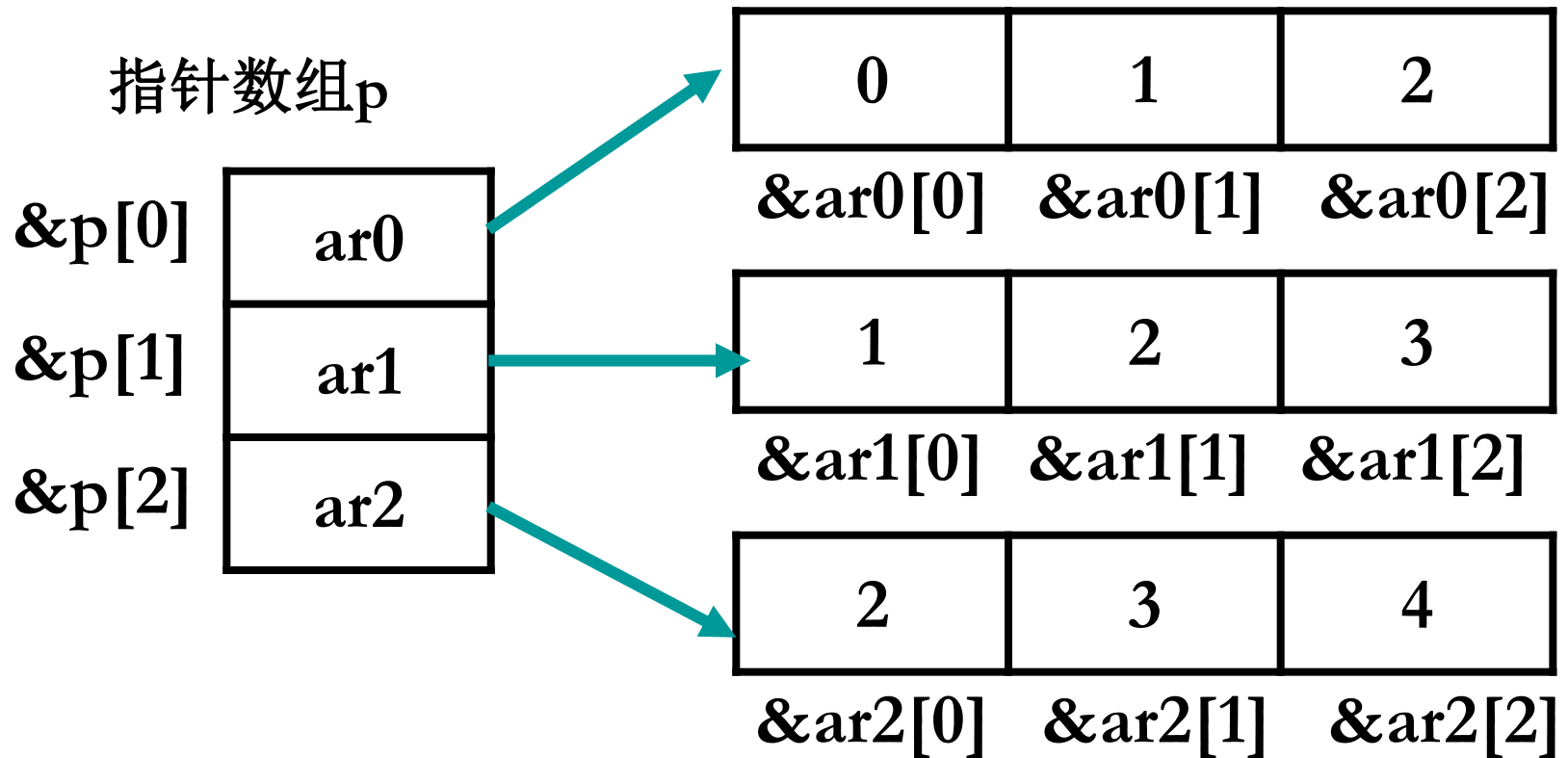
## ■ 概念

- 指针数组是指针的数组
- 数组单元中存放的是地址，这些地址指向同一种数据类型的变量。

## ■ 指针数组的定义和初始化

- `int ar0[ ]={0,1,2};`
- `int ar1[ ]={1,2,3};`
- `int ar2[ ]={2,3,4};`
- `int* p[3] = {ar0, ar1, ar2}`
- `P[0]= ar0    P[0]= &ar0[0]`

# 说明



数组名就是该数组的符号地址，是数组的元素首地址：

`ar0 = 0012FF74`

`&ar0 = 0012FF74`

`&ar0[0] = 0012FF74`

```
#include <stdio.h>           //预编译命令
int main()                   //主函数
{                             //函数体开始
    int ar0[ ]={0,1,2};      //定义整数数组ar0,并初始化
    int ar1[ ]={1,2,3};      //定义整数数组ar1,并初始化
    int ar2[ ]={2,3,4};      //定义整数数组ar2,并初始化
    int *p[ ]={ar0, ar1, ar2}; //定义指针数组,并初始化

    printf("数组名就是该数组符号地址，是数组元素首地址:\n");
    printf("ar0=%08X\n", ar0);
    printf("&ar0=%08X\n", &ar0); // 输出数组ar0的地址
    printf("&ar0[0]=%08X\n", &ar0[0]);
                                // 输出数组ar0的元素首地址}
```



```
int i;
```

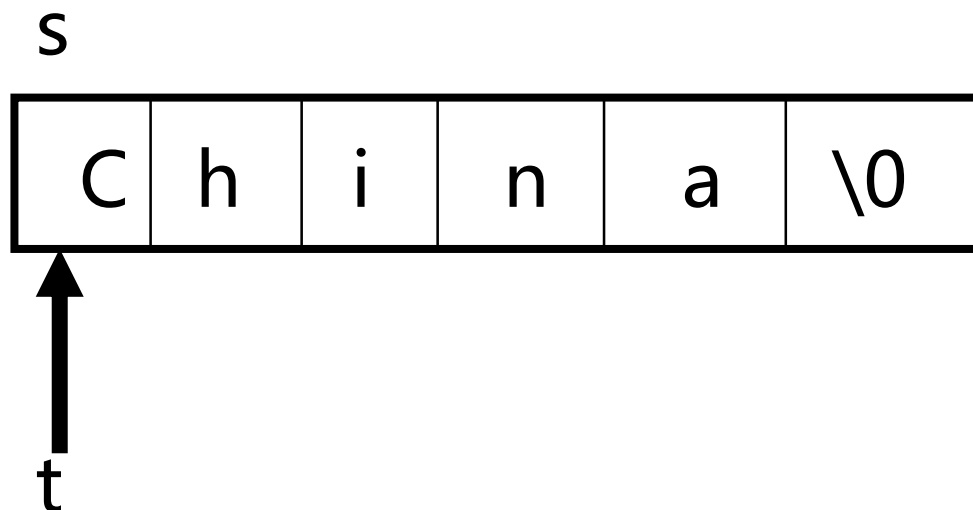
```
char s[ ] = "China"; //s[ ]为字符数组
```

```
char* t = s;
```

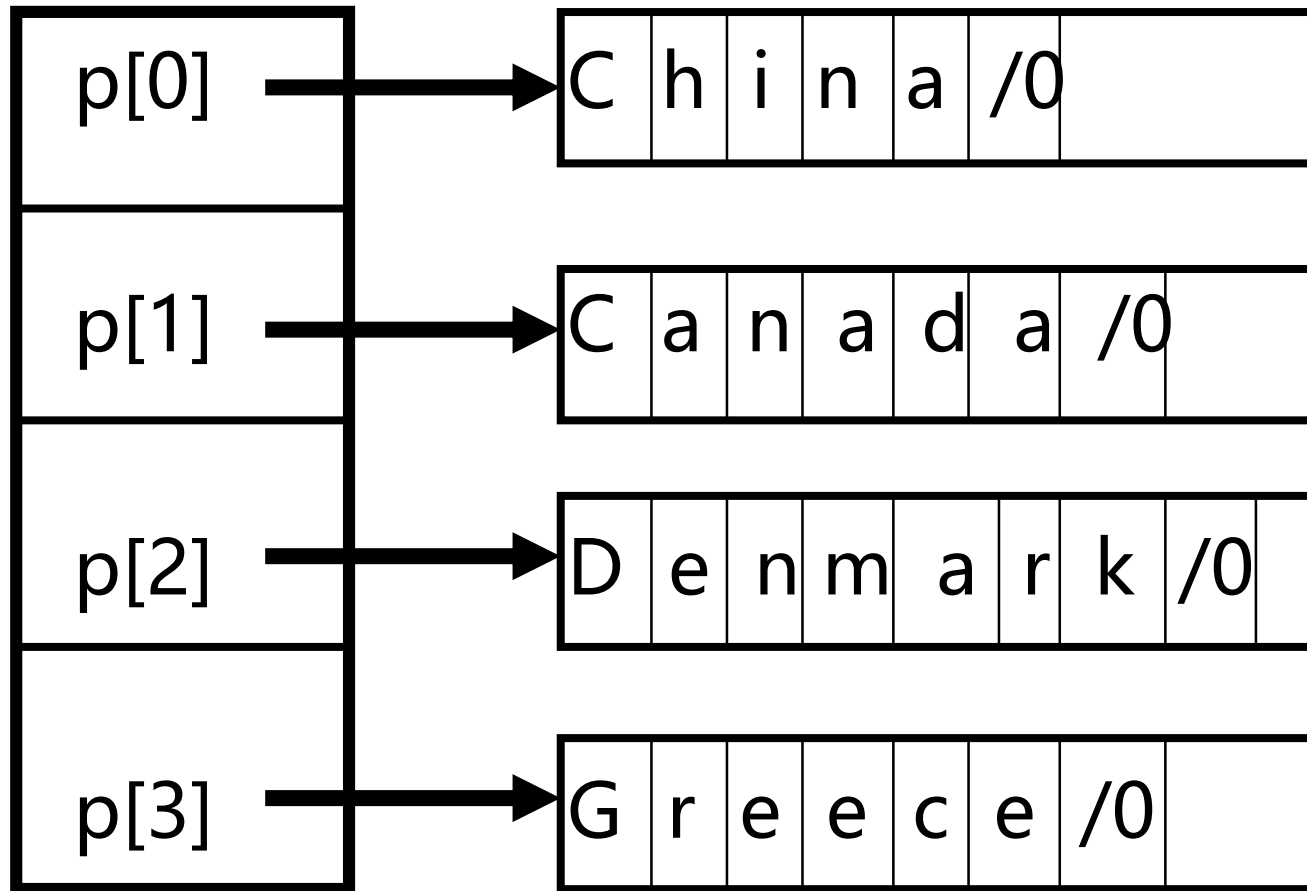
//t为指向字符数组的指针变量

P[0],p[1],p[2]

```
char*p[ ]={ "China","Canada",  
            "Denmark","Greece" };
```



```
char*p[ ]={ "China","Canada",  
            "Denmark","Greece" };
```



# 动态数组

```
#include<iostream>
using namespace std;
#include <stdlib.h>
int main() {
    int *ptr;
    int len;
    cout << "Input array length: ";
    cin >> len;
    ptr = (int *) malloc( len * sizeof(int) );
    for (int i = 0; i < len; i++) {
        ptr[i] = i + 1;
        cout << ptr[i] << endl;
    }
    free(ptr);
    return 0;
}
```

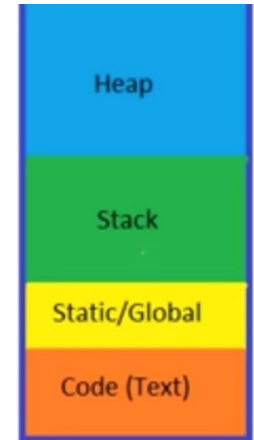
引入stdlib.h头文件!

```

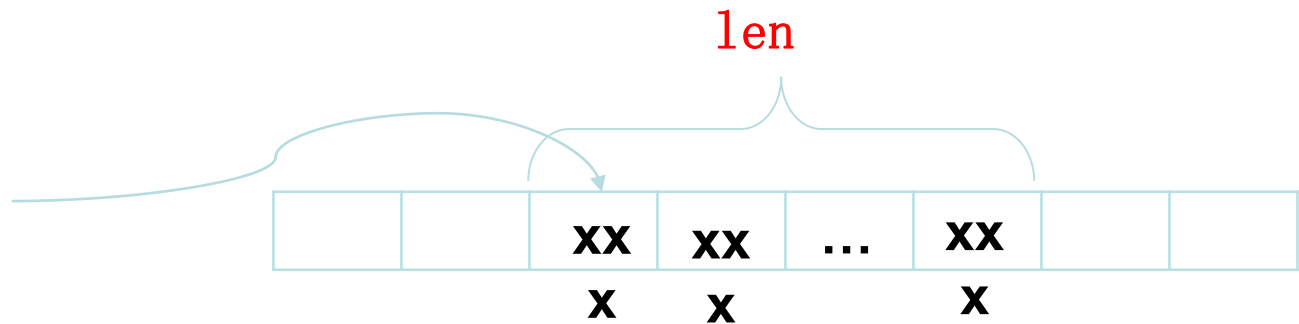
#include<iostream>
using namespace std;
#include <stdlib.h>
int main() {
    int *ptr;
    int len;
    cout << "Input array length: ";
    cin >> len;
    ptr = (int *) malloc(len * sizeof(int));
    for (int i = 0; i < len; i++) {
        ptr[i] = i + 1;
        cout << ptr[i] << endl;
    }
    free(ptr);
    return 0;
}

```

## 为程序分配的 内存空间



main  
ptr



# What's new in the code?

## ■ 动态内存分配函数

```
ptr = (int *) malloc( len * sizeof(int) );
```

➤ 函数原型      `void *malloc(unsigned int num_bytes);`

➤ 函数功能

- 向系统申请分配指定num\_bytes个字节的内存空间
- 返回指向内存空间首地址的指针。

## ■ 内存释放函数

```
free(ptr);
```

➤ 动态分配内存必须手动释放！

➤ 这是与静态数组的不同。不释放会产生内存泄露



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

## 3. 指针与字符串

---

# 复习：什么是字符串

---

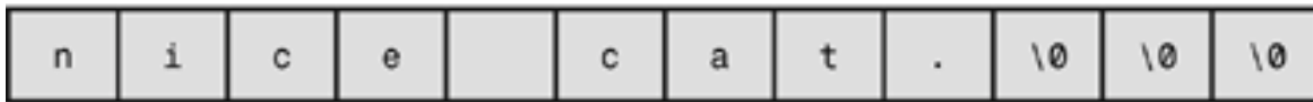
- C语言如何表示字符串
  - 字符型的数组，如char ary[10];
  - 以字符' \0'（ASCII码为0）结束
- 可以使用对数组的任何操作
- 字符串频繁被使用，因此C语言设计了特殊的函数处理字符串
  - 如何声明与初始化
  - 如何输入与输出
  - 如何进行操作

# 定义字符串 (1)

## ■ 方法1：使用字符数组

### ➤ 声明字符数组的方式

```
char pets[12] = "nice cat.";
```



多余位补\0

### ➤ 如何使用数组初始化的方式

```
char pets[12] = {  
    'n', 'i', 'c', 'e',  
    ' ', 'c', 'a', 't', '\0'  
};
```



# 定义字符串 (2)

---

## ■ 方法1：使用字符数组

- 更方便的方式：让C编译器决定数组长度

```
char pets[] = "nice cat.";
```

- 可以使用变量定义数组长度吗？

```
int n = 8;  
char cakes[2 + 5];  
char crumbs[n];
```

- C99标准之前不支持char crumbs[n]
- C99标准之后引入了变长数组(VLA)机制

# 定义字符串 (3)

## ■ 方法2: 使用字符指针

```
char* pets = "nice cat.";
```

### ➤ 使用字符数组与字符指针的区别

```
int main() {  
    char p[] = "china";  
    p[4] = 'e' ;  
    printf("=%s\n", p) ;
```

✓ 字符数组, 允许改变!

```
char* p = "china";  
p[4] = 'e';  
printf("p1=%s\n", p1);  
}
```

✗ 常量字符串, 不允许改变!

# 定义字符串 (4)

## ■ 方法3：使用字符指针数组

```
char *mytal[] = {  
    "Adding numbers swiftly",  
    "Multiplying accurately",  
    "Stashing data",  
    "Following instructions to the letter",  
    "Understanding the C language"  
};
```

➤ 以下取值分别是什么？

- \* mytal[2], mytal[3][0], mytal[1]

➤ 是否可使用二维数组定义？

```
char mytal_2[LIM][LINLIM];
```

# 对字符指针和字符数组的赋值

## 1、对字符指针变量赋值的写法

(1) `char *p;`  
    `p = "computer" ;`

(2) `char *p= "computer" ;`

以上两种都行。可以整体赋值。

## 2、对字符数组赋初值的写法

(1) `char as[12]= "department" ;`// 可以。在定义时可以整体赋值  
    `char as[] = "department" ;`// 可以。在定义时可以整体赋值

(2) `char as[12];`  
    `as = "department" ;`      // 不可以! 不可以整体赋值  
    `as[12]= "department" ;`    //不可以! 不可以整体赋值

# 字符串的输入 (1)

## ■ 字符串输入的方法

### ➤ 方法1: scanf

```
char name1[11], name2[11];  
int count;  
printf("Please enter 2 names.\n");  
count = scanf("%s %s", name1, name2);  
printf("I read the %d names %s and %s.\n",  
count, name1, name2);  
return 0;
```

- 遇到换行符和空白符(如空格字符)时停止输入
- 概念上更像是输入一个单词(word)

# 字符串的输入 (2)

## ■ 字符串输入的方法

### ➤ 方法1: gets

```
int main(void) {  
    char name[81];  
    char * ptr;  
    ptr = gets(name);  
    printf("%s? Ah! %s!\n", name, ptr);  
    return 0;  
}
```

- 遇到换行符(\n)时候输入停止

```

int i, n;
char a[1000][20];
scanf("%d", &n);
getchar();
//如果没有上面getchar(), 下面gets就出错
//第一个gets获得是空串, 导致最后一个字符串没有获得
for (i = 0; i < n; i++)
    gets(a[i]); //用gets出错

```

```

/*
输入样例 //第一行以
5

```

```

aaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccc
ddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeee
*/

```

```

int i, n;
char a[1000][20];
scanf("%d", &n);
for (i = 0; i < n; i++)

```

## ■ 遇 “空格” 都结束

```

    scanf("%s", a[i]); //正常,但如果是gets就会出错。
//但用scanf的缺点是输入字符串不能有空格,
//用gets的好处是可以输入包含有空格的字符串

```

```

printf("%d\n", n);
for (i = 0; i < n; i++)
    puts(a[i]);

```

# 字符串的输出 (1)

---

## ■ 使用puts函数

```
#include <stdio.h>
#define DEF "I am a #defined string."
int main(void) {
    char str1[80] = "An array was initialized.";
    const char * str2 = "A pointer was initialized.";
    puts("I'm an argument to puts().");
    puts(DEF);
    puts(str1);
    puts(str2);
    puts(&str1[5]);
    puts(str2+4);
    return 0;
}
```

注意：puts函数会自动在输出的字符串后加入换行符！



# 字符串的输出 (2)

- 使用puts函数

- 程序挑错并修正

```
#include <stdio.h>
int main(void) {
    char side_a[] = "Side A";
    char dont[] = {'W', 'O', 'W', '!' };
    char side_b[] = "Side B";
    puts(dont);
    return 0;
}
```

- 注意：有无**\0结尾**标识着是否为字符串

# 字符串的输出 (3)

---

- 使用printf函数
- 使用输出流cout
- 与puts函数对比
  - 函数puts比printf使用起来更方便

```
printf("%s\n", string);  
puts(string);
```

- 函数printf比puts更灵活

```
printf("Well, %s, %s\n", name, MSG);
```

# 字符串处理函数

---

- C的库函数提供一组字符串处理函数
- 头文件: `string.h`
- 常用的字符串操作函数
  - 字符串长度: `strlen()`
  - 字符串连接: `strcat()`, `strncat()`
  - 字符串比较: `strcmp()`, `strncmp()`
  - 字符串拷贝: `strcpy()`, `strncpy()`

# 字符串拷贝strcpy函数 (1)

- 考虑以下字符串赋值问题

- 将指针pts1指向的字符串拷贝到pts2上

- 实现方法1:

**pts2 = pts1;**

- 只实现了地址的拷贝
- 没有实现字符串内容的拷贝

- 函数strcpy

- 输入：两个字符串str1和str2
- 输出：将str2拷贝到str1起始的位置上

## 字符串拷贝strcpy函数 (2)

---

```
#include <stdio.h>
#include <string.h>
int main(void){
    char qwords[5][40];
    char temp[40];
    int i = 0;
    printf("Enter %d words beginning with q:\n", 5);
    while (i < 5 && gets(temp)) {
        if (temp[0] != 'q')
            printf("%s doesn't begin with q!\n", temp);
        else {
            strcpy(qwords[i], temp);i++;
        }
    }
    return 0;
}
```

# 字符串连接strcat函数 (2)

- 对以下程序进行挑错

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char *p1 = "hello ";
    char *p2 = "world";
    strcat(p1, p1+2);
    puts(p1);
    return 0;
}
```

**strncat**(flower, addon, **10**);  
//把addon最多10个字符进行连接

- 第一个字符串必须足够大容纳连接后的新串

# 字符串比较strcmp函数 (1)

## ■ 考虑以下问题

- 针对某一问题，让用户输入回答，直到正确

```
#include <stdio.h>
#define ANSWER " Beijing"
int main(void) {
    char try[40];
    puts(" What is the capital of China?");
    gets(try);
    while (try != ANSWER) {
        puts("No, that's wrong. Try again.");
        gets(try);
    }
    puts("That's right!");
    return 0;
}
```

语句try!=ANSWER  
不能达到比较目的!

# 字符串比较strcmp函数 (2)

---

## ■ 字符串比较

- 输入：两个字符串str1和str2
- 输出：字符串的比较结果

`strcmp("A", "A")` is 0

`strcmp("A", "B")` is -1

`strcmp("B", "A")` is 1

`strcmp("C", "A")` is 1

`strcmp("Z", "a")` is -1

`strcmp("apples", "apple")` is 1



# 字符串比较strcmp函数 (3)

---

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

#define ANSWER "Beijing"

int main(void) {
    char try[40];
    puts("What is the capital of China?");
    gets(try);
    while (strcmp(try, Answer)!=0) {
        puts("No, that's wrong. Try again.");
        gets(try);
    }
    ...
}
```

# 字符串匹配

- 读入两个字符串a和b，判断a是否是b的子串。如果是，计算a在b中出现了几次。
  - 例如：a="aba"， b="ababab"， 则a在b中出现了2次。
- 分析：
  - a是b的子串，就是说存在一个整数i，使得 $a_0 = b_i$ ,  $a_1 = b_{i+1}$ ,  $a_2 = b_{i+2}$ , ...,  $a_{|a|-1} = b_{i+|a|-1}$  ( $0 \leq i \leq |b|-|a|$ ), 其中|a|表示字符串a的长度，|b|表示字符串b的长度

# 解题思路

- 用2个字符串数组a和b表示两个字符串
- 用cin读入两个字符串的内容
- 用循环变量i表示a在b中出现的位置，让它从0一直循环到  $(lb-la)$
- 判断是否满足  $a_0 = b_i, a_1 = b_{i+1}, a_2 = b_{i+2}, \dots, a_{la-1} = b_{i+la-1}$ 
  - 用一个循环变量j表示待匹配的位置，从0一直循环到  $(la-1)$ ，如果所有的  $a_j = b_{i+j}$ ，则说明a出现在从  $b_{i+1}$  到  $b_{i+la-1}$  的位置上
- 设置一个计数器Count，一发现a在b中出现一次，就让  $Count+1$
- 最终  $Count = 0$ ，则a不是b的子串，否则输出Count

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

```
char a[100], b[100];
int la, lb, Count;
void input_data() {
    scanf("%s", a);
    scanf("%s", b);
}
```

```
void solve() {
    int i, j;
    bool match;
    la = strlen(a); lb = strlen(b);
    Count = 0;
```

```
    for (i = 0; i <= lb-la; i++) {
        match = true;
        for (j = 0; j < la; j++) {
            if (a[j] != b[i + j]) {
                match = false;
                break;
            }
        }
        if (match) Count++;
    }
```

```
}
```

```
//读入字符串a
//读入字符串b
```

```
//定义2个循环变量;
//表示从某一个位置开始是否匹配
//计算a, b的长度
//计数器清0
//循环变量i表示起始位置
//先假设能够匹配
//循环变量表示待匹配位置
//如果某一位置上无法匹配
//跳出循环
```

```
//如果可以匹配，则计数器加1
```

# 字符串的3种操作

---

## ■ 字符串处理中的常用操作

- **取子串**：输入字符串、子串的起始位置、长度，输出子串内容
- **插入子串**：输入待插入字符串、目标字符串、制定位置，将待插入字符串插到目标串的制定位置前，并输出插入后的目标串
- **删除子串**：输入字符串、待删除子串的起始位置和长度，从字符串中删除子串，并输出删除后的字符串

# 取子串

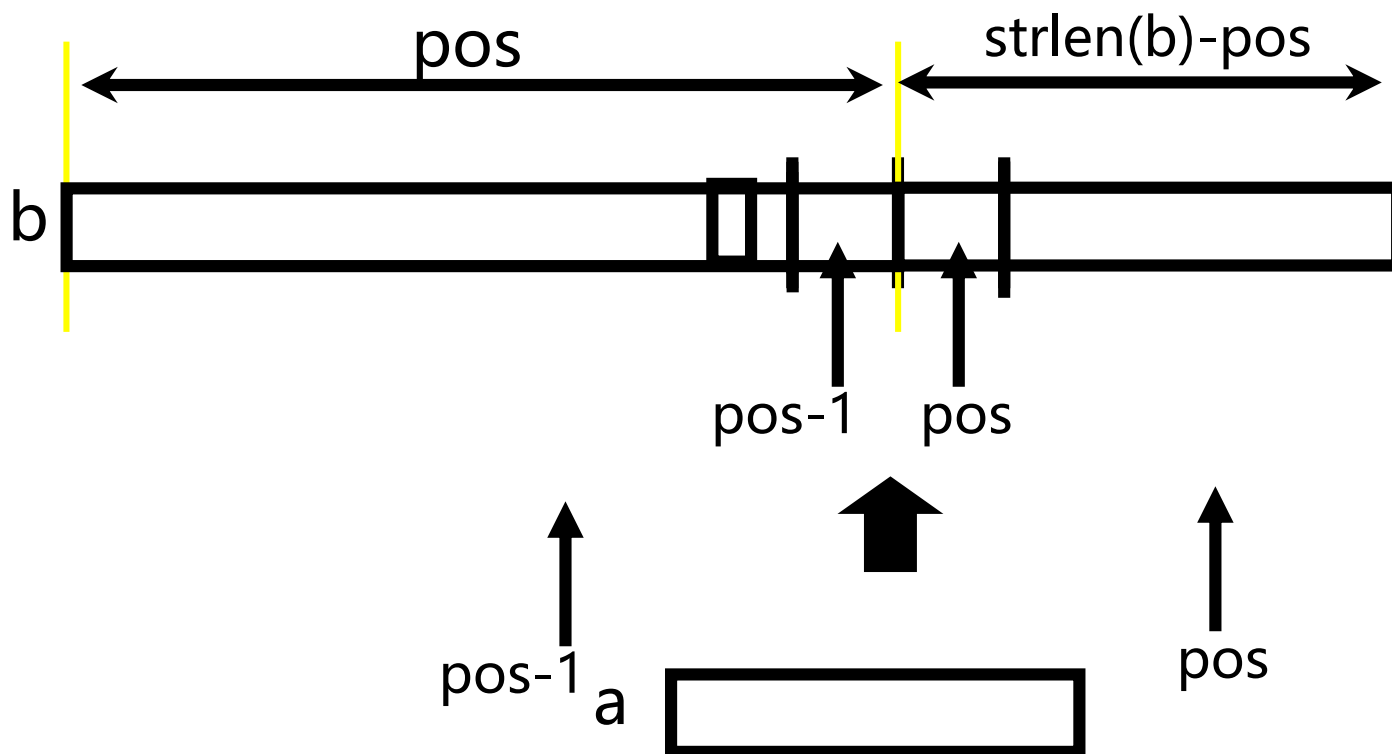
---

## ■ 取子串

- 原字符串a, 子串的起始位置start, 子串的长度len
- 字符串ans表示所求的子串
- 循环变量i表示当前复制的字符, 让i从start循环到start+len-1, 把a[i]复制到ans[i-start]的位置上
- `ans[len]='\0';`

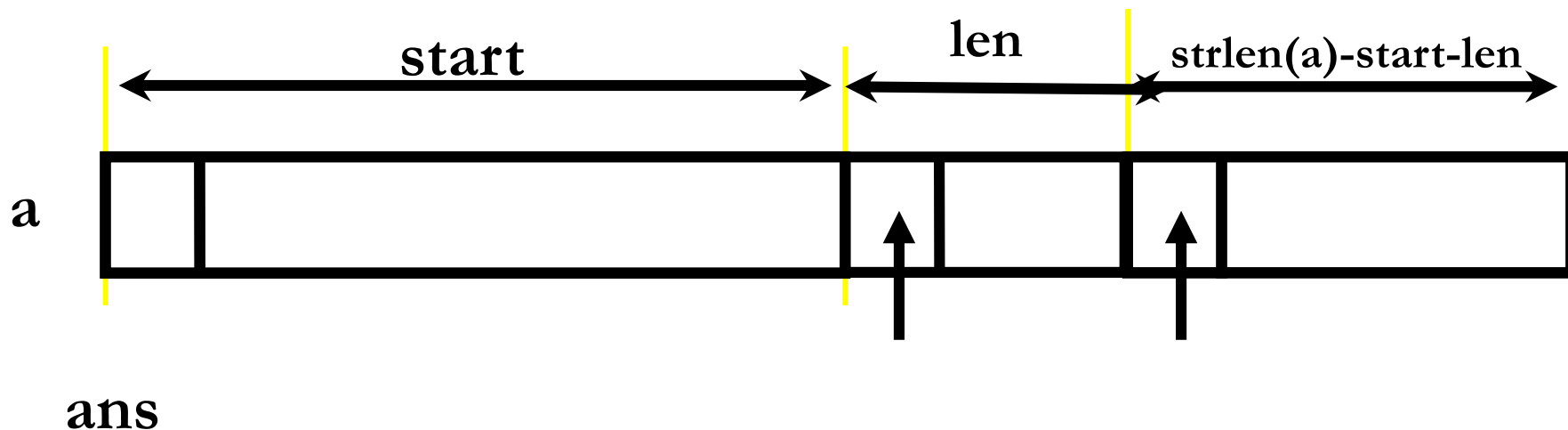
# 插入子串

- 将字符串a插入字符串b的pos位置前，相当于将b分成2部分  $b[0 \sim \text{pos}-1]$  和  $b[\text{pos} \sim \text{strlen}(b)-1]$ ，把a接在  $b[0 \sim \text{pos}-1]$  之后，再把  $b[\text{pos} \sim \text{strlen}(b)-1]$  接在a之后



# 删除子串

- 把字符串a当中从start开始，长为len的一段删除，相当于把a分成3部分， $a[0 \sim \text{start}-1]$ ,  $a[\text{start} \sim \text{start}+\text{len}-1]$ ,  $a[\text{start}+\text{len}-1 \sim \text{strlen}(a)-1]$ ，并且只保留第1段和第3段，即将这两段相连。







中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

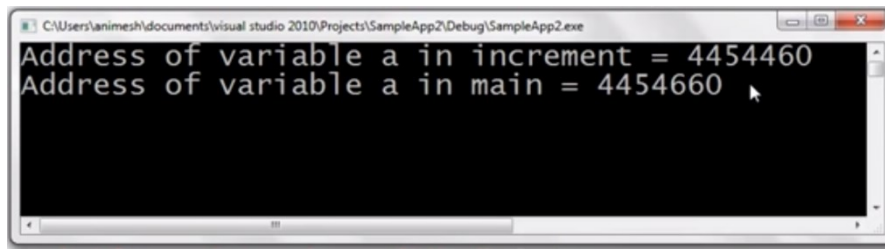
## 4. 指针与函数

---

# 指针作为函数的参数

## ■ 函数的传值调用 (Call by Value)

```
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```



```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
Address of variable a in increment = 4454460
Address of variable a in main = 4454660
```

# 指针作为函数的参数

---

- 函数的传引用调用 (Call by Reference)

```
void Increment (int *p) {  
    *p = (*p)+1;  
}
```

```
int main () {  
    int a = 10;  
    Increment (&a);  
    printf ("a = %d\n", a);  
    return 0;  
}
```

# 用Call By Reference实现swap函数

```
#include <stdio.h>
void interchange(int * u, int * v);
int main(void) {
    int x = 5, y = 10;
    printf("Originally x = %d and y = %d.\n", x, y);
    swap(&x, &y); /* send addresses to function */
    printf("Now x = %d and y = %d.\n", x, y);
    return 0;
}
```

```
void swap (int * u, int * v) {
    int temp;
    temp = *u;
    *u = *v;
    *v = temp;
}
```

```
int* temp;
temp = u;
u = v;
v = temp;
```

为什么不对?

```
#include <stdio.h>
#include <stdlib.h>
// 传地址 (指针) , 修改指针所间接访问的内容
void swap1(int * x, int * y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    printf( "子函数: *x=%d *y=%d\n" , *x, *y);
}
```

✓ 交换成功

// 传值，修改的子函数空间的变量值

**void swap2(int x, int y)**

```
{   int temp;
    temp = x; x = y; y = temp;
    printf( "子函数: x=%d y=%d\n" , x, y);
}
```

✗ 交换失败

// 传地址（指针），修改指针本身

**void swap3(int \*x, int \*y)**

```
{   int *p;
    p = x; x = y; y = p;
    printf( "子函数: *x=%d *y=%d\n" , *x, *y);
}
```

✗ 交换失败

// C++的引用调用方式

```
void swap4(int &x, int &y)
```

```
{
```

```
    int temp;
```

```
    temp = x;
```

```
    x = y;
```

```
    y = temp;
```

```
    printf( "子函数: x=%d y=%d\n" , x, y);
```

```
}
```

✓ 交换成功

# 函数的返回值是指针/指针作为函数的参数

```
// *****
// * 功 能：输出两数中的大者 *
// *****

#include <iostream>                // 预编译命令
using namespace std;
float * max(float *p, float *q)    // 函数，求两数中的大者，返回float*
{
    float *r;
    if (*p > *q) r=p;
    else r=q;
    return r;
}
int main()                        // 主函数
{
    float x, y, *s;
    scanf("%f", x);
    printf("x=%f\n", x);
    scanf("%f", y);
    printf("x=%f\n", y);
    s = max(&x, &y);
    printf("两数中的大数为%f\n", *s); // 输出x,y中的大者
    return 0;
}
```



```
int main()                                // 主函数
{
    float x, y, *s ;
    scanf("%f", x);
    printf("x=%f\n", x);
    scanf("%f", y);
    printf("x=%f\n", y);
    s = max(&x, &y);

    printf("两数中的大数为%f\n", *s);
    return 0;
}                                           // 主函数结束
```

```
float * max ( float *p, float *q )  
// 函数, 求两数中的大者, 返回指针  
{  
    float *r ;  
    if ( p > q ) r = p ;  
    else r = q;  
    return r ;  
}
```

# 例子

---

例 有a个学生，每个学生有b门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数实现。

解题思路：

- 定义二维数组score存放成绩
- 定义输出某学生全部成绩的函数search，它是返回指针的函数，形参是行指针和整型
- 主函数将score和要找的学号k传递给形参
- 函数的返回值是&score[k][0](k号学生的序号为0的课程地址)
- 在主函数中输出该生的全部成绩

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

```
int main()
```

```
{ float score[ ][4]={60,70,80,90}, {56,89,67,88},{34,78,90,66}};
```

```
float *search(float (*pointer)[4],int n);
```

```
float *p; int i, k;
```

```
scanf("%d",&k);
```

```
printf("The scores of No.%d are:\n",k);
```

```
p=search(score, k);
```

```
for(i=0;i<4;i++)
```

返回k号学生课程首地址

```
    printf("%5.2f\t",*(p+i));
```

```
printf("\n");
```

```
return 0;
```

```
}
```

```
float *search(float (*pointer)[4],int n)
```

```
{ float *pt;
```

```
    pt=*(pointer+n);
```

```
    return(pt);
```

```
}
```

# 找不出及格学生

---

.....

```
float *search(float (*pointer)[4]);  
float *p; int i,j;  
for(i=0;i<3;i++)  
{ p=search(score+i);  
  if(p==*(score+i))  
    { printf("No.%d score:",i);  
      for(j=0;j<4;j++)  
        printf( "%5.2f  ",*(p+j));  
      printf("\n");  
    }  
}
```

.....

```
float *search( float (*pointer)[4] )  
{ int i=0;  
  float *pt;  
  pt=NULL;  
  for( ; i<4; i++ )  
    if(*( *pointer+i )<60)  
      pt=*pointer;  
  return(pt);  
}
```

# 函数指针作为参数传递

- 举例: qsort函数
- `void qsort(void* base, size_t n, size_t size, int (*cmp)(const void*, const void*))`
  - base 为要排序的数组
  - n为要排序数组的长度
  - size为数组元素的大小 (以字节为单位)
  - cmp为判断大小函数的指针
    - 这个函数需要自己定义, 如果 $a > b$ , 函数返回1;  $a < b$ , 函数返回-1;  $a = b$ , 函数返回0

```
#include <stdio.h> //预编译命令
```

```
#include <time.h>
```

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

```
int compare(const void *a, const void *b);
```

```
int main(){
```

```
    int ary[20], i;
```

```
    srand((unsigned)time(NULL));    //随机产生20个数
```

```
    for (i=0; i<20; i++) ary[i]=rand();
```

```
    for (i=0; i<20; i++) printf("%6d ", ary[i]);
```

```
    printf("\n\n");
```

```
    qsort(ary, 20, sizeof(int), compare );    //排序
```

```
    for (i=0; i<10; i++) printf("%6d ", ary[i]);
```

```
    printf("\n");
```

```
    return 0;}
```



```
int compare(const void *a, const void *b)
{
    a = (int *)a;
    b = (int *)b;

    if ( *a > *b ) return 1;
    else
        if ( *a == *b ) return 0;
        else return -1;
}
```

**整数比较**

**如果a>b, 返回1, 则是从小到大排序**

```
int compare(const void *a, const void *b)
{
    char *p, *q;
    p = (char *)a;
    q = (char *)b;
    return strcmp(p, q);
}
```

**字符串比较**



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

# 5. 指针与结构体

---

# 静态结构数组

- 定义名为student的结构体，包含属性
  - name（字符串）、sex（字符）、birthday（长整型）、height和weight（浮点型）

```
struct student //名为student的结构类型
{
    char name[20]; //姓名
    char sex;      //性别
    unsigned long birthday; //生日
    float height;   //身高
    float weight;   //体重
};
```

# 静态结构数组

- 声明结构数组room [3]，并初始化

```
struct student room[3] = {  
    {"Li li", 'M', 19840318, 1.82, 65.0 },  
    {"Mi mi", 'M', 19830918, 1.75, 58.0 },  
    {"He lei", 'M', 19841209, 1.83, 67.1 }  
};
```

- 使用冒泡排序算法对room[3]按生日排序

```
for ( i = 0; i < 2; i ++ ) {  
    for ( j = 0; j < 2 - i; j ++ ) {  
        if (room[j].birthday > room[j+1].birthday) {  
            struct student q = room[j];  
            room[j] = room[j+1];  
            room[j+1] = q;  
        }  
    }  
}
```

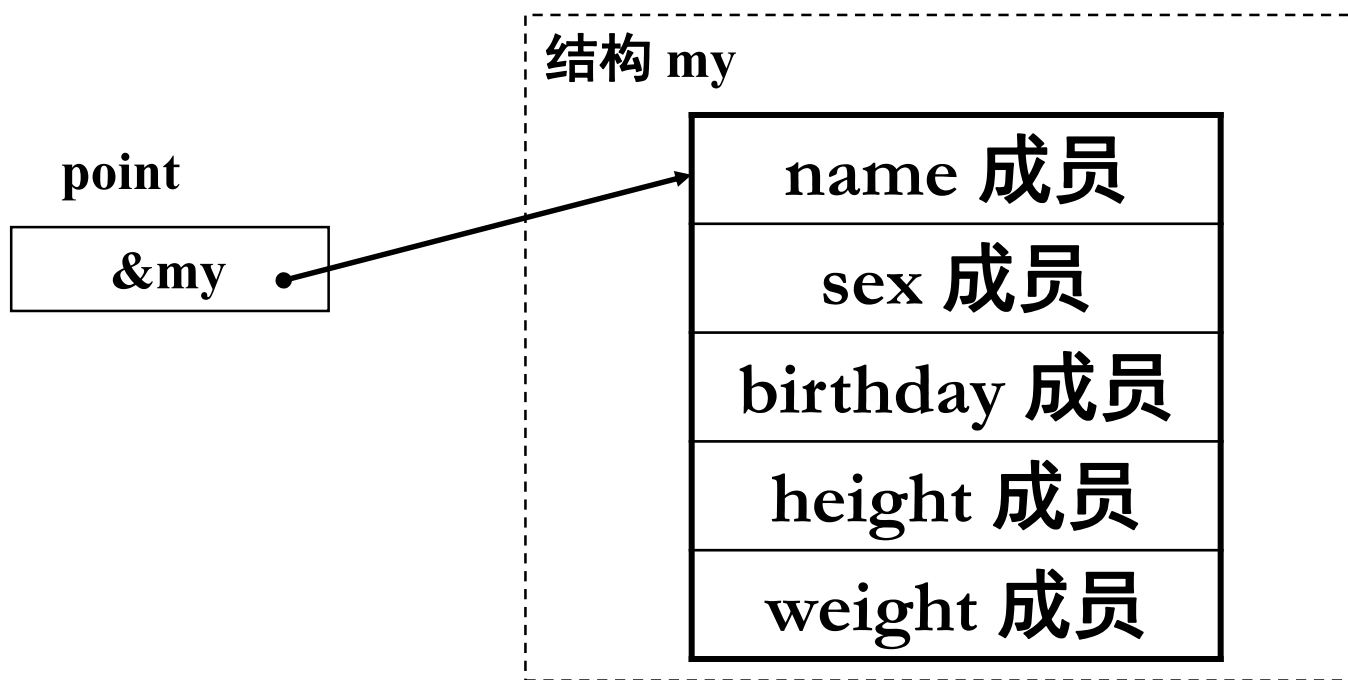
# 使用指针访问结构变量

---

```
int main() {  
    struct student my;  
    struct student *point; // 定义student指针  
    point = &my; // 将指针point 指向结构my  
  
    strcpy( point->name, "Wang Xiaorong");  
    point->sex = 'F';          // 输入学生数据  
    point->birthday = 19840923;  
    point->height = 1.62f;  
    point->weight = 51.5f;  
  
    printf( "%s\n" , my.name);  
    printf( "%c\n" , my.sex);  
    printf( "%d\n" , my.birthday);  
    printf( "%f\n" , my.height);  
    printf( "%f\n" , my.weight);  
    return 0;  
}
```

# What's new?

- `struct student *point` : 定义一个指向结构 `student` 的指针 `point`
- `point = &my` : 将结构变量 `my` 的地址赋给指针 `point`, 指向 `my` 结构第一个成员的地址。



# What's new?

---

- 有了指向结构的指针就可以通过箭头操作符 “->” 来访问结构的成员
  - `point->birthday = 19840923;`
  - `point->sex = 'F' ;`
  - `point->height = 1.62f;`
  - `point->weight = 51.5f;`
- 都是给结构成员赋值的语句。
- 思考 “->” 操作符与什么等价?
  - `point -> name = (*point).name`

# 使用指针数组处理结构 (1)

- 如何使用指针数组完成排序?

- 如何声明指向结构的指针数组

```
struct student *p[4]={&Room[0], &Room[1],  
                      &Room[2], &Room[3] };
```

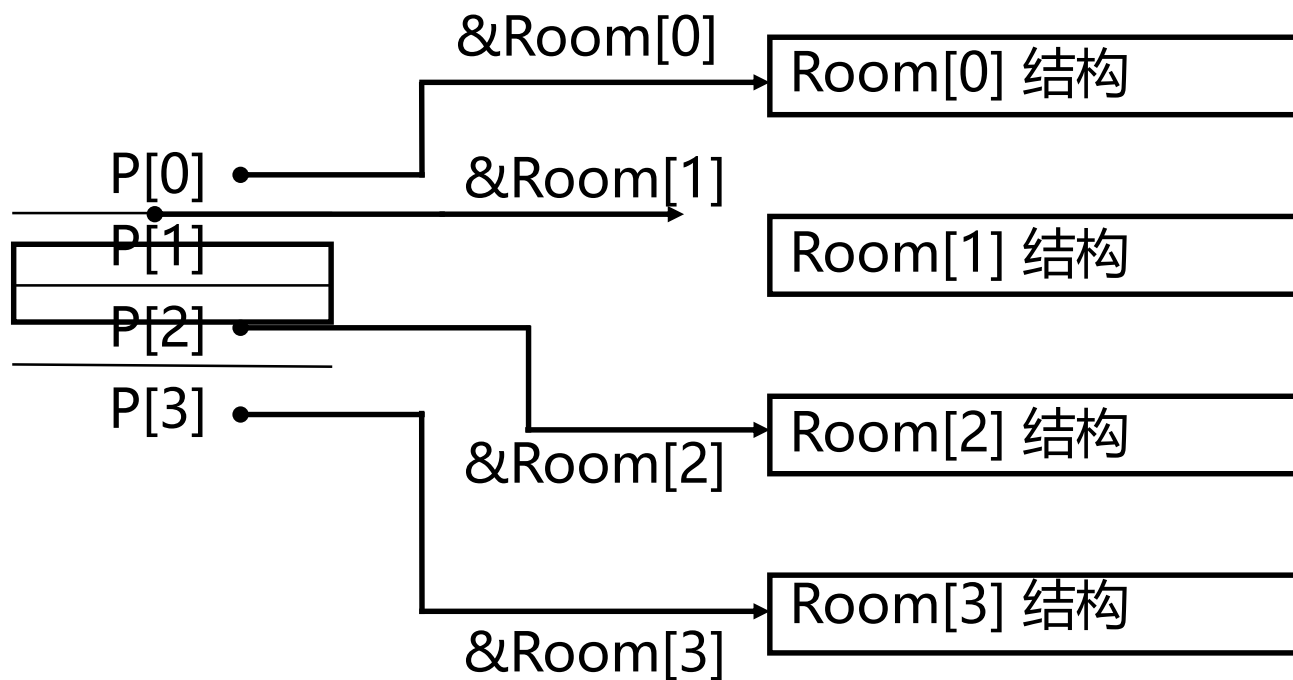
- 如何使用指针完成排序

```
for (int i=0; i<3; i++) {  
    for (int j=0; j<=3-i; j++) {  
        if (p[j]->height < p[j+1]->height) {  
            struct student *q = p[j];  
            p[j] = p[j+1];  
            p[j+1] = q;  
        }  
    }  
}
```



## 使用指针数组处理结构 (2)

- `p[ ]`是指向 `student` 结构的指针数组，经初始化赋入的是 `room` 数组中元素所在的地址



## 使用指针数组处理结构 (3)

- `p[j]->height`: `p[j]` 的指针通过箭头操作符来访问 `my` 结构成员 `height`。
- `if (p[j]->height < p[j+1]->height)` : 如果 `Room[j]` 中的 `height` 成员比 `Room[j+1]` 中的 `height` 成员小, 就去做下面的语句组:
  - 变换指针的指向, 让 `p[j]` 改去指向 `Room[j+1]`, 让 `p[j+1]` 指向 `Room[j]`。
  - 这样做就比前一个程序好。当着结构成员很多时, 直接让结构变量相互交换并不是好办法, 而现在的做法只需将指针的指向交换一下, 极为省力

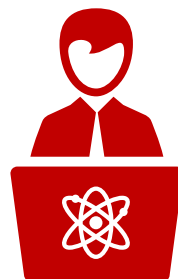
# 动态结构数组

---

```
int main() {  
    struct student *ptr;  
    int len;  
    cout << "Input array length: ";  
    cin >> len;  
  
    ptr = (struct student *) malloc( len * sizeof(struct student));  
  
    for (int i = 0; i < len; i++) {  
        // Initialize ptr[i]  
    }  
  
    free(ptr);  
    return 0;  
}
```



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

# 谢谢大家!

---

