第8章 指针(2)

内容

- 指针类型：指向变量的数据类型， unsigned int
- 指针参数：指向实际内存单元；数组；指针运算；
- 指针与数组
  - 冒泡排序算法
  - 查找算法
- 字符串与指针

内容

- 指针类型
- 指针参数
- 指针与数组
  - 冒泡排序算法
  - 查找算法
- 字符串与指针

回顾“选择”排序算法(例7-5)

1. 找到最小元素，并记录其下标
2. 将最小元素交换到最前面
3. 对数组的其余部分(最小值之后的部分)重复上述1和2步骤，直至剩余部分只有一个元素

选择排序计算量分析(算法的时间复杂度)

$$(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$$

$$O(n^2)$$

另一种等价的“选择”排序算法

1. 找到最大元素，并记录其下标
2. 将最大元素交换到最后面
3. 对数组的其余部分(最大值之前的部分)重复上述1和2步骤，直至剩余部分只有一个元素

- 算法复杂度：$O(n^2)$

冒泡排序算法

- 算法思想(从小到大)
  1. 对数组中的元素，按下标顺序对相邻两个元素比较，将较大者交换到后面 (将最大元素交换到最后面)
  2. 对数组的其余部分(最大值之前的部分)重复上述步骤，剩余部分只有一个元素

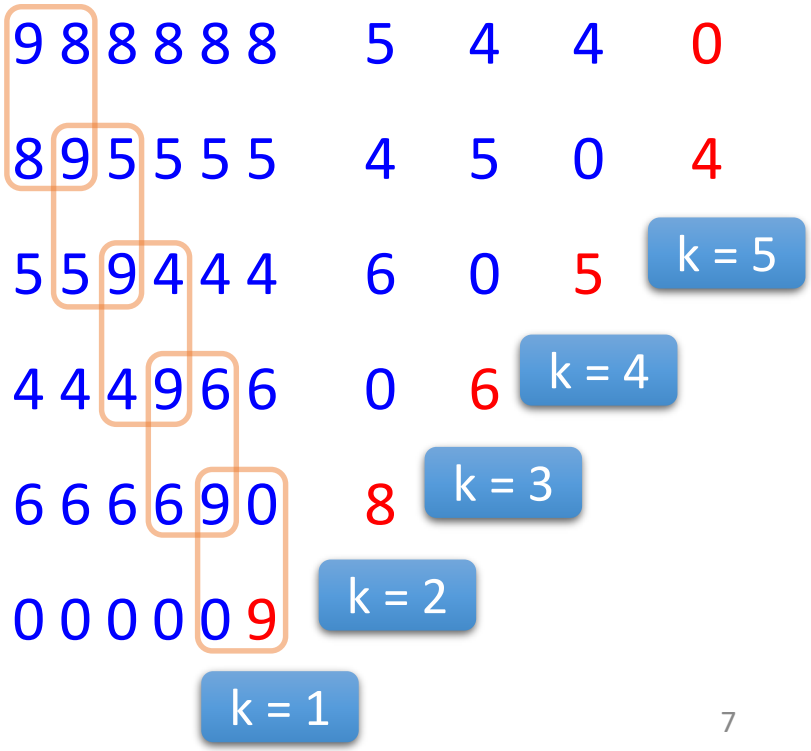
- 如何将最大元素交换到最后面?
  1. 从第一个元素开始，与紧随其后的元素比较。如果前面的大，那么交换这两个元素的值
  2. 重复上述步骤，直到最后一个未被排序的元素

冒泡排序算法

```
void bubble(int a[], int n)
{
    int k, j, temp;
    for( k = 1; k < n; k++ ) {
    /*将 a0 ~ an-k 最大者交换到an-k */
        for( j = 0; j < n-k; j++ )
            if( a[j] > a[j+1] ) {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
    }
}
```

输入数组：9 8 5 4 6 0

用冒泡法从小到大排序的过程



冒泡排序算法

- 冒泡排序计算量分析:

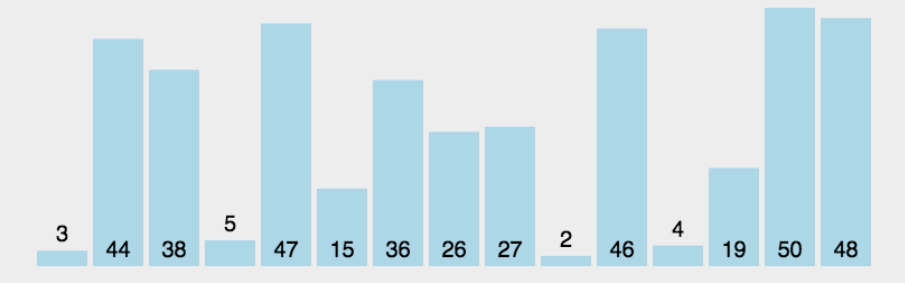
$$(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$$

$$O(n^2)$$

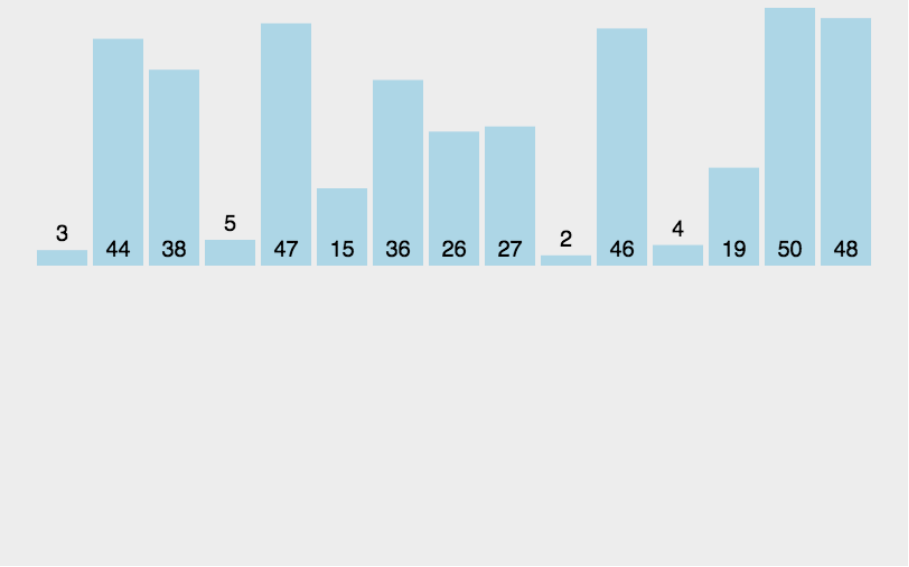
- 如何实现从大到小排序?

其它排序算法*：基于分而治之

- 快速排序 O(nlogn)：使用分而治之策略把一个序列分为较小和较大的2个子序列，然后递归地排序两个子序列。



- 归并排序O(nlogn)：递归地把当前序列平均分割成两半、排序，将两个已经排序的序列合并成一个序列。



内容

- 指针类型
- 指针参数
- 指针与数组
  - 冒泡排序算法
  - 查找算法
- 字符串与指针

查找算法：线性查找

问题：如何在数组中查找给定元素？

线性查找：逐个比对，直到找到或者数组结束

```
int find(int a[ ], int n, int x)
{
    int k;
    for( k = 0; k < n; k++ )
        if( a[k] == x )  return k;
    return -1;
}
```

关于线性查找

- 算法的平均复杂度为O(n)
  - 最好的情况，只要比对 1 次
  - 最坏的情况，需要比对 n 次
  - 如果需要查找的元素在每个位置出现的概率是一样的，那么平均需要比对 n/2 次

- 提高算法的查找效率：二分查找法（分而治之）

[例7.7 P165] 二分法：数据排序，分段查找

```
int binary_search(int a[], int n, int x)
{   /* a[ ] 具有从小到大顺序*/
    初始化搜索区间为[0, n-1];
    while ( 搜索区间不为空) {
        计算搜索区间中间元素序号;
        if( 中间元素 == x )   return 该元素index;
        if( 中间元素 < x )    搜索区间缩为右半边;
        else    搜索区间缩为左半边;
    }
    return ( 没找到 );
}
```

[例7.7 P165] 二分法：数据排序，分段查找

```
int binary_search(int a[], int n, int x)
{    /* a[] 具有从小到大顺序*/
     int low = 0, high = n-1, mid;
     while ( low <= high ) {
          mid = ( low + high ) / 2;
          if( a[mid] == x )    return mid;
          if( a[mid] < x )     low = mid + 1;   // 为什么 “mid+1” ？
           else  high = mid -1;  // 为什么 “mid-1” ？
     }
     return ( -1 );
}
```

二分查找算法复杂度

- 每执行一次比较，将搜索空间降为一半
- 算法的时间复杂度为$\log_2 n$
  - 最好的情况，只要比对 1 次
  - 最坏的情况，需要比对 $\log_2 n$ 次

- 二分查找算法说明：合理组织数据的重要性!

内容

- 指针类型
- 指针参数
- 指针与数组
  - 冒泡排序算法
  - 查找算法
- 字符串与指针

字符串与指针

- 字符串的存储
  - 连续空间、存储字符序列

  - 末尾添加一个结束标志字符 '\0'
- 字符串的表示
  - 表示为一个指向首字符的字符指针
  - char * 类型
- 字符串常量
  - 实际是一个指向首字符的字符指针常量

字符串举例

```
/* 定义字符指针，指向字符串常量*/
char *sp = "Hello";

/* 定义字符数组，初始化为字符串常量*/
char sa[] = "World";
char sb [100] = "Today";

/* 输出字符串*/
printf("%s\n", sp);
printf("%s\n", sa);
printf("%s\n", sb);
printf("%s\n", "Morning");
```

➔ Hello

➔ World

➔ Today

➔ Morning

字符串举例

```
/* 定义字符指针，指向字符串常量*/
char *sp = "Hello";

/* 定义字符数组，初始化为字符串常量*/
char sa[] = "World";
char sb [100] = "Today";

/* 输出字符串*/
printf("%s\n", sp+2);          ➔ llo
printf("%s\n", sa+2);          ➔ rld
printf("%s\n", sb+2);          ➔ day
printf("%s\n", "Morning"+2);   ➔ rning
```

printf 函数与字符指针

char *p = “Hello”;

printf(“%s”, p); 输出指针所指向的字符串

printf(“%d”, p); 以整数输出指针地址值

printf(“%u”, p); 以无符号整数输出指针地址值

printf(“%x”, p); 以16 进制整数输出指针地址值

scanf 函数与字符指针

char sa[100], *p, *p1 = sa, *p2 = “Hello”;

scanf(“%s”, p1); 读入一个字符串

scanf(“%s”, p2); 错! 因为p2 指向的是字符串常量

scanf(“%s”, p); 错! 因为p没有指向实际内存空间

字符串处理函数

- scanf/printf　输入输出
- gets/puts　输入输出
- strcpy　字符串拷贝复制
- strcat　字符串连接
- strcmp　字符串比较
- strlen　求字符串长

需包含库函数头文件

#include <string.h>

scanf/printf 输入输出

- scanf(“%s”, p);
  - 输入之前，要求 p 指向了实际内存单元
  - 碰到空格、制表符、回车停止读入
  - 输入末尾添加‘\0’，作为字符串结束标志

- printf(“%s”, p);
  - 要求p 指向了实际字符串(以‘\0’ 结束)

字符串处理函数—— 练习

假设定义变量如下:

```
char p[100];
int n;
```

下面哪一句会出错?

```
scanf(“%d”, n);
scanf(“%d”, &n);
scanf(“%s”, p);
scanf(“%s”, &p);
```

gets/puts 字符串输入/输出函数

- char * gets(char * s);　　/* 读入字符串*/
  - s 指向一段可写入字符串的内存的起始地址
  - 返回值为：指针 s
- int puts(char * s);　　/* 输出字符串*/
  - s 为指向字符串的指针
  - 返回值为：成功，输出非负整数；失败，EOF

注：不同编译器可能有不同的返回值，可能是最后一个字符、输出字符个数(/INT+MAX)、非负的常数
https://en.cppreference.com/w/c/io/puts

gets/puts 字符串输入/输出函数

- char * gets(char * s); /* 读入字符串*/
- int puts(char * s); /* 输出字符串*/

```
char s[100] , *p;
gets(s);   /* 读入一个字符串*/
gets(p);   /* 错!  指针p 没有指向内存*/
puts(s);   /* 输出字符串s */
```

gets 函数定义

```
char * gets( char * s )
{
    char *s0 = s, ch;
    while( (ch=getchar ()) != '\n' &&  ch != EOF  )
        *s++ = ch;
    *s = '\0';
    return s0;
}
```

puts 函数定义

```
int puts( char * s )
{
    int n = 0;
    while( s[n] ) putchar( s[n++] );
    return n;
}
```

scanf (“%s”) 与 gets 的异同

- 相同点：输入一个字符串
- 不同点：
  - scanf：遇到回车或空格、制表符，结束
  - gets：遇到回车结束，允许输入空格、制表符

printf与puts的异同

- 相同点：输出一个字符串
- 不同点：
  - printf：有大量格式控制符，可以输出转义字符
  - puts：将‘\0’ 转换为‘\n’输出，无格式控制

strcpy 字符串复制函数 (string.h)

char * strcpy(char * s1 , char * s2);

等价于赋值操作：字符串s1 = 字符串s2

参数要求：

- s2 指向一个以‘\0’ 为结尾的字符串
- s1 指向一段可写入字符串的内存的起始地址
- s1 所指向的内存足以容纳字符串s2

返回值为：字符串指针s1

strcpy 字符串复制函数 (string.h)

```
char s[100];
strcpy(s, “Hello”);
printf(“%s”, s);
```

或者

```
char s[100];
printf(“%s”, strcpy(s, “Hello”));
```

strcpy 函数定义

```
char * strcpy( char * s1 , char * s2 )
{
    char * s0 = s1;
    while( *s1++ = *s2++ )
        ;
    return s0;
}
```

strcat 字符串连接函数(string.h)

char * strcat(char * s1 , char * s2);

等价于复合赋值操作：字符串s1 += 字符串s2

参数要求：

- s1 和s2 都指向以‘\0’为结尾的字符串
- 字符串s1之后还有足够空间，可容纳字符串s2

返回值为：字符串指针s1

strcat 字符串连接函数(string.h)

```
char s[100] = “water”;
strcat(s, “mellon”);    /* s += “mellon” */
puts(s);                /* 输出是什么？ */
```

或者

```
char s[100] = “water”;
puts(strcat(s, “mellon”));
```

下面的调用可以吗？为什么？

```
strcat(“hot”, s));
```

strcat 函数定义

```
char * strcat( char * s1 , char * s2 )
{
    char * s0 = s1;
    while( *s1 ) s1++;          /* 移动到字符串末尾*/
    while( *s1++ = *s2++ ) ; /* 复制s2 */
    return s0;
}
```

字符串比较规则

给定两个字符串s1 和s2，比较规则如下：

- 从第 0 个字符，逐个字符对进行比较，直到
  1. 出现不一样的字符。假设第k 个字符不一样，那么字符串s1 和s2 大小关系定义为：s1[k] 和s2[k] 大小关系
  2. 两个字符串都结束了。那么字符串s1 和s2 完全相等

例如：“abc” < “b”

例如：“Abc” < “abc”

strcmp字符串比较函数(string.h)

int strcmp(char * s1 , char * s2);

可以理解为：字符串s1 - 字符串s2

要求：s1 和s2 都指向以‘\0’为结尾的字符串

返回值为：

- 负数：当字符串s1 < 字符串s2
- 为零：当字符串s1 == 字符串s2
- 正数：当字符串s1 > 字符串s2

strcmp 函数实现

```
int strcmp( char * s1 , char * s2 )
{
    while( *s1 == *s2 && *s1 )
        s1++, s2++;
    return *s1 - *s2;
}
```

strlen字符串长度(string.h)

```
int strlen( char * s );
```

参数s 指向以‘\0’为结尾的字符串

返回值：字符串s的长度，即s中的字符个数
不包括末尾的‘\0’

strlen字符串长度(string.h)

char s[] = “Hello”;

strlen(s) 等于多少？ 5
sizeof(s) 等于多少？ 6
strlen(s+2) 等于多少？ 3

strlen函数实现

```
int strlen( char * s )
{
    char * p = s;
    while( *p ) p++;
    return p - s;
}
```

内容

- 指针类型
- 指针参数
- 指针与数组
  - 冒泡排序算法
  - 查找算法
- 字符串与指针