

# 计算概论（C语言）习题课讲义08

## 内容概要

- 习题讲解
- 数组进阶
- 课堂练习: 排序初步

## 习题讲解

输出所有字符数不超过**10**的行

代码点评一:

1. 读入一行 =>如何判断读入一行?
2. 进行判断
3. 少于输出 =>输出时, 结果如何存储?
4. 否则跳过

```
#include <stdio.h>

int main()
{
    char Line[1000];
    char c;
    int count=0;

    while((c=getchar())!=EOF)
    {
        Line[count++]=c;
        if(c=='\n')    //读入一行
        {
            //进行判断
            if(count<=10) //输出
            {
                for(int i=0;i<count;i++)
                {
                    putchar(Line[i]);
                }
            }
            else //跳过
            {
            }
            count=0;
        }
    }
    return 0;
}
```

代码点评二:

```
#include <stdio.h>
#include <string.h>           // string.h?
int main(){
    int i=0,c,k;
    int line[10];
    while((c=getchar())!=EOF)
    {
        if(i==10)
        {
            i=0;
            while(getchar()!='\n') ;    // what for?
            continue;                  // continue?
        }
        else if(c=='\n')
        {
            for(k=0;k<i;++k)
                putchar(line[k]);
            printf("\n");
            i=0;
            continue;
        }
        else
        {
            line[i]=c;
            ++i;
        }
    }
    return 0;
}
```

## 牛顿迭代及二分法

代码点评一:

```
#include <stdio.h>
#include <math.h>
#define e0 1e-13
double f(double x);
double f_(double x);

void Newton(double x0){
    static int step=0;           // static 的作用?
    if(fabs(x0*exp(x0)-1) < e0)
        printf("Newton method, root: %f ,iterations: %d\n",x0,step);
    else{
        x0=x0-f(x0)/f_(x0);
        step++;
        Newton(x0);
    }
}
```

```

void Bisec(double a,double b){
    static int step=0;
    double c;
    if(fabs(b*exp(b)-1) < e0)
        printf("Bisection method, root: %f ,iterations: %d",b,step);
    else {
        c=(a+b)*0.5;
        step++;
        if(f(c)*f(a)<0)
            Bisec(a,c);
        else
            Bisec(c,b);
    }
}

double f(double x0){
    return x0*exp(x0)-1;
}
double f_(double x){
    return (x+1.)*exp(x);
}
int main(){
    Newton(0.5);
    Bisec(0.5,0.6);
    return 0;
}

```

## 丑数判断

```

int isUgly(int x)
{
    // Recursion exit
    if(x==1)
    {
        return 1;
    }
    // Recursion
    if(x%2==0)
    {
        return isUgly(x/2);
    }
    if(x%3==0)
    {
        return isUgly(x/3);
    }
    if(x%5==0)
    {
        return isUgly(x/5);
    }
    // Have other factors
    return 0;
}

```

典型错误:

1. 递归调用缺少return
2. `x==1` 误写为 `x=1`

## 反转数字

```
int reverse(int x)
{
    int res=0;
    while(x!=0)
    {
        int tmp=x%10;
        x=x/10;
        res=res*10+tmp;
    }
    return res;
}
```

## 计算天数

闰年的计算: 四年一闰, 百年不闰, 四百年再闰.

Julian calendar and Gregorian calendar

<https://www.timeanddate.com/calendar/julian-gregorian-switch.html>

<https://baike.baidu.com/item/%E5%84%92%E7%95%A5%E5%8E%86/3052736>


Create calendar: Month: September Year: 1752 Country: United States Show Show & Save

Quick Design Formatting More options

◀ August September October ▶ Full year 1752

### September 1752 (United States)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Phases of the moon: 15  23  30 

#### Printer-friendly calendar

Printing [Help page](#) for better print results.

[Printable Calendar \(PDF\)](#) for easy printing

[Add own events](#) to PDF Calendar



Phases of the moon are calculated using local time in New York.  
Moon symbols:  
● New Moon. ◐ 1st Quarter. [Disable moonphases.](#)  
○ Full Moon. ◑ 3rd Quarter.

```
// Julian calendar vs Gregorian calendar
int isLeapYear(int year)
{
    if((year%4==0)&&(year%100!=0))
    {
        return 1;
    }
    if(year%400==0)
    {
        return 1;
    }
    return 0;
}
```

计算天数:

```
int days(int year, int month, int day)
{
    int res=day;
    month--;
    switch(month)
    {
        case 11: res+=30;
        case 10: res+=31;
        case 9: res+=30;
        case 8: res+=31;
        case 7: res+=31;
        case 6: res+=30;
        case 5: res+=31;
        case 4: res+=30;
        case 3: res+=31;
        case 2: res+=28;
        case 1: res+=31;
    }
    if(isLeapYear(year)&&month>=2)
    {
        res++;
    }
    return res;
}
```

检查输入:

```
int valid(int year, int month, int day)
{
    // test year
    if(year<=0) return 0; //Not exactly accurate.
    // test month
    if(month<=0||month>=13) return 0;
    // test day
    if(day<=0) return 0;
    if(month==1&&day>31) return 0;
```

```

    if(isLeapYear(year)==0&&month== 2&&day>28) return 0;
    if(isLeapYear(year)==1&&month== 2&&day>29) return 0;
    if(month== 3&&day>31) return 0;
    if(month== 4&&day>30) return 0;
    if(month== 5&&day>31) return 0;
    if(month== 6&&day>30) return 0;
    if(month== 7&&day>31) return 0;
    if(month== 8&&day>31) return 0;
    if(month== 9&&day>30) return 0;
    if(month==10&&day>31) return 0;
    if(month==11&&day>30) return 0;
    if(month==12&&day>31) return 0;
    return 1;
}

```

输入:

```

while(1)
{
    if(scanf("%d %d %d",&year,&month,&day)==3)
    {
        if(valid(year,month,day)==1)
        {
            break; // Input OK.
        }
        else
        {
            printf("%d.%d.%d is invalid!\n",year,month,day);
        }
    }
    else
    {
        while(getchar()!='\n') //Eat invalid input
        {
        }
        printf("Please Input Three Integers!!!\n");
    }
}

```

## 数组进阶

### 多维数组

```

// 二维数组定义
int a[3][4];

// 二维数组的初始化
int a[3][4] = {
    {0, 1, 2, 3} , /* 初始化索引为 0 的行 */
    {4, 5, 6, 7} , /* 初始化索引为 1 的行 */
    {8, 9, 10, 11} /* 初始化索引为 2 的行 */
};

```

```
//内部嵌套的括号是可选的，下面的初始化与上面是等同的：
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
// 这实际上反映了二维数组, 在内存上的线性结构。

// 二维数组变量的使用
int val = a[2][3];
```

演示: 二维数组的线性结构

## 数组做为函数参数

```
//rev函数, 将数组元素颠倒
void rev(int n, int a[])
{
    ...
}
```

注意,以数组为参数的函数可以改变实参数组的元素值(这本质上是一种地址传递).

而多维数组做为函数的参数时, 除了最左处的维度的长度,其他都需要给定(具体原因和 multidimensional arrays 的线性结构相关).

```
double aaverage(int n, double a[][5])
{
    ...
}
```

## 数组做为函数的返回值

C语言不允许返回一个完整的数组作为函数的返回值。但是，可以通过数组名来返回一个指向数组的指针。

## 课堂练习

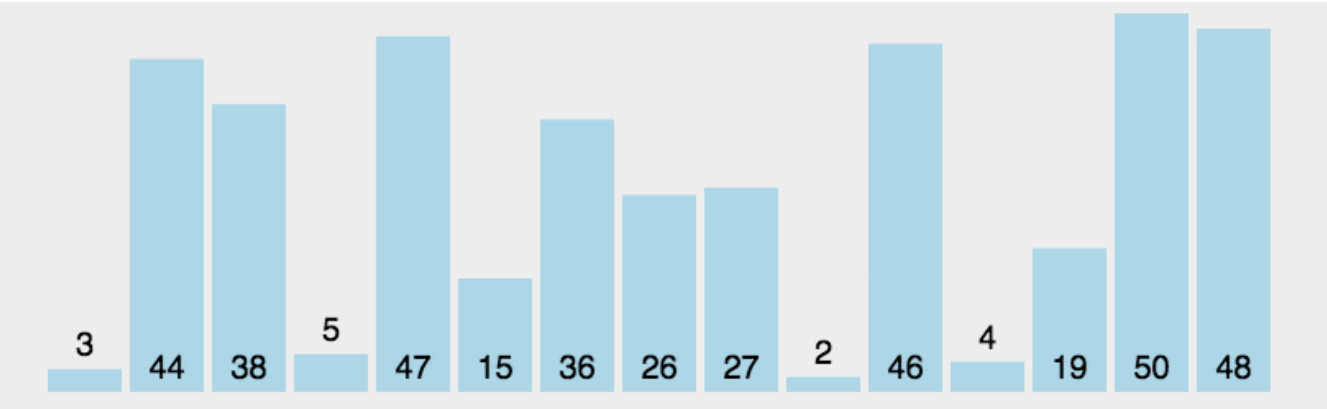
排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

今天先介绍前四种.

选择排序

主要想法: 选择适当的元素交换到适当位置.

过程演示:

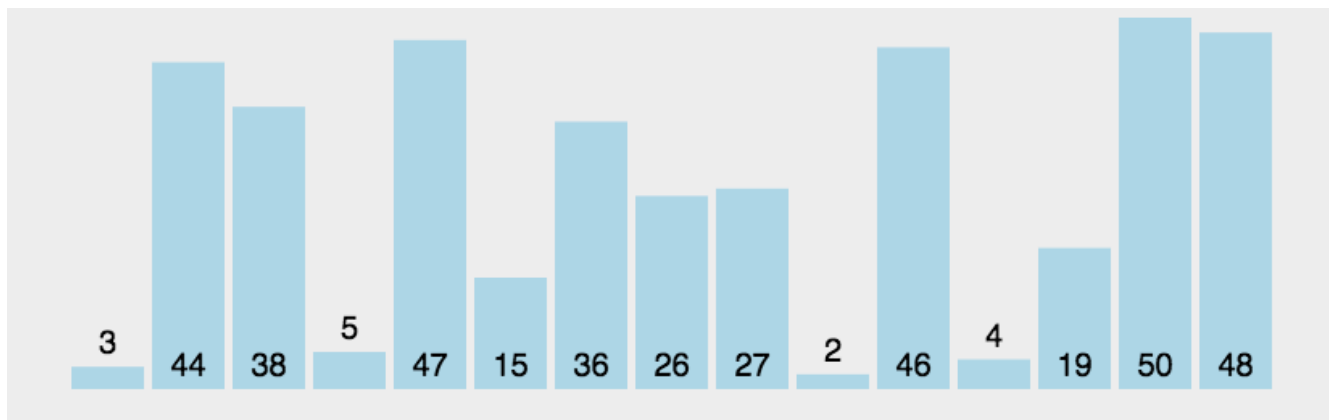


冒泡排序

主要想法: 让大的元素"冒"出来.

过程演示:



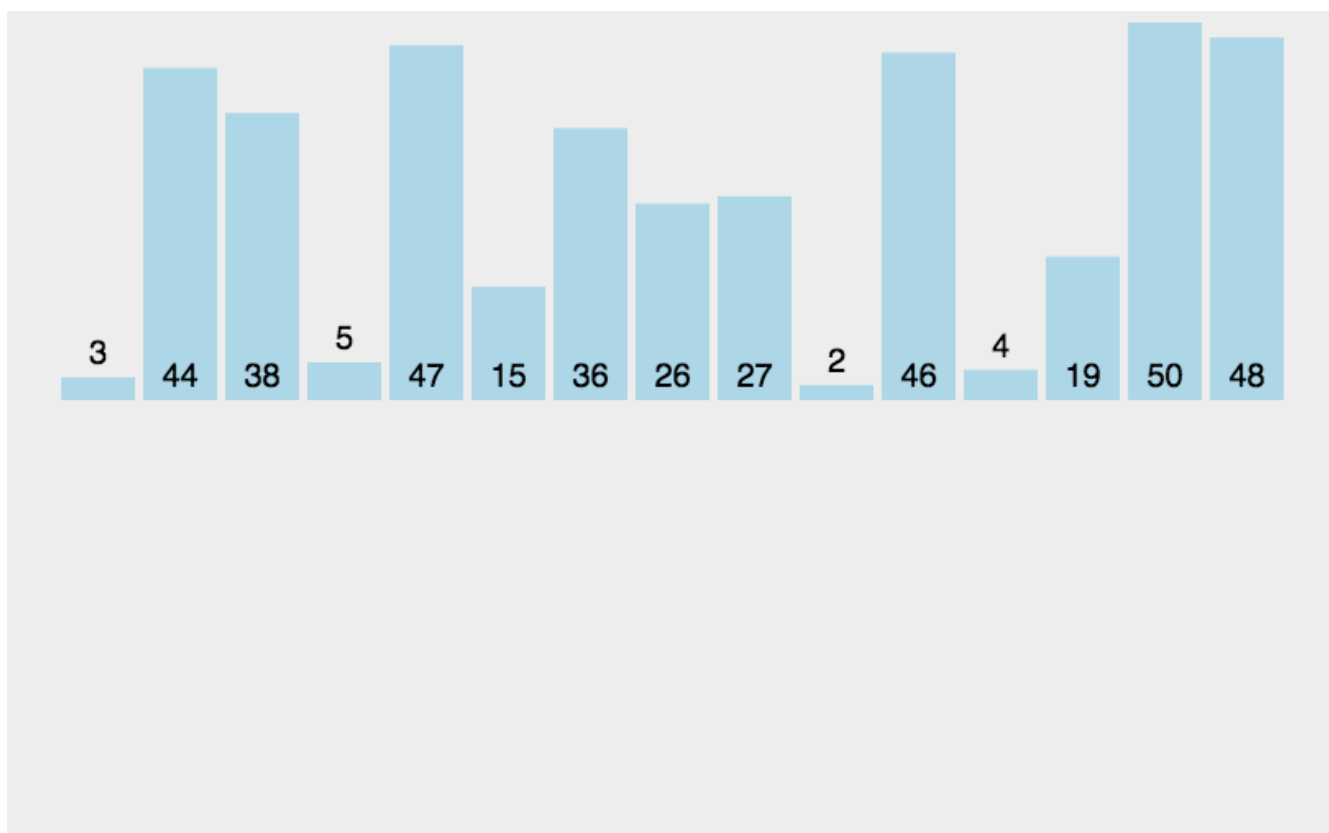


## 插入排序

主要想法:



过程演示:



## 希尔排序

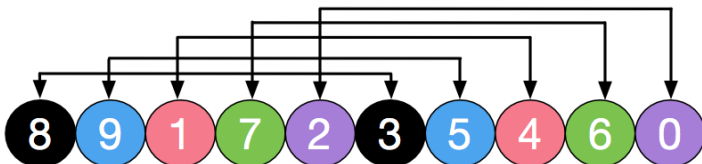
主要想法: 对子列使用插入排序.

过程演示:

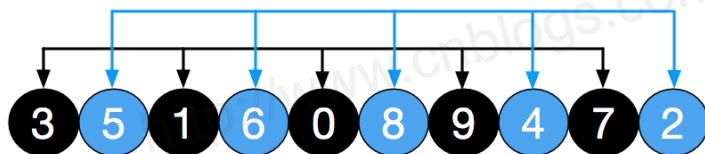
原始数组 以下数据元素颜色相同为一组



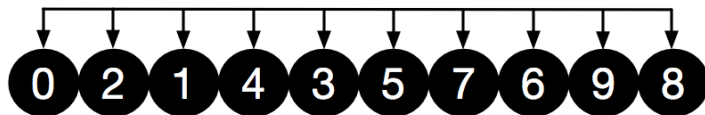
初始增量  $gap=length/2=5$ ，意味着整个数组被分为5组， $[8,3]$   $[9,5]$   $[1,4]$   $[7,6]$   $[2,0]$



对这5组分别进行直接插入排序，结果如下，可以看到，像3，5，6这些小元素都被调到前面了，然后缩小增量  $gap=5/2=2$ ，数组被分为2组  $[3,1,0,9,7]$   $[5,6,8,4,2]$



对以上2组再分别进行直接插入排序，结果如下，可以看到，此时整个数组的有序程度更进一步啦。再缩小增量  $gap=2/2=1$ ，此时，整个数组为1组  $[0,2,1,4,3,5,7,6,9,8]$ ，如下



经过上面的“宏观调控”，整个数组的有序化程度成果喜人。

此时，仅仅需要对以上数列简单微调，无需大量移动操作即可完成整个数组的排序。



参考链接:

[1] <https://www.toptal.com/developers/sorting-algorithms>

[2] <https://www.jianshu.com/p/a1e97094f61b>