# 法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

# 第二课　一维数组与函数初步

林沐

# 内容概述

第一部分:一维数组

1.数组的引入
2.数组的概念
3.数组的赋值与使用
4.数组的初始化
5.例1-数组的查找
6.确定数组占用的空间与元素个数
7.数组的地址
8.数组越界
9.变长数组
10.例2-最长平台
11.例3-基于交换的排序

第二部分: 函数初步

1.程序的结构
2.函数的定义:名称、参数、返回值
3.例4-无重复数字
4.数组作为参数的函数
5.按值传递机制
6.例5-因式分解
7.函数的声明:函数原型
8.变量的作用域与生存期
9.递归函数

# 第一部分：一维数组

从键盘读入12名学生成绩，成绩为整型，计算这12名同学的平均成绩输出；观察如下两段代码，思考如果是120名或10000名学生会怎样？

```c
#include <stdio.h>
int main(){
    int grade0, grade1, grade2;
    int grade3, grade4, grade5;
    int grade6, grade7, grade8;
    int grade9, grade10, grade11;
    printf("Please input 12 integers:\n");
    scanf("%d %d %d", &grade0, &grade1, &grade2);
    scanf("%d %d %d", &grade3, &grade4, &grade5);
    scanf("%d %d %d", &grade6, &grade7, &grade8);
    scanf("%d %d %d", &grade9, &grade10, &grade11);
    double sum = 0;
    sum += grade0 + grade1 + grade2;
    sum += grade3 + grade4 + grade5;
    sum += grade6 + grade7 + grade8;
    sum += grade9 + grade10 + grade11;
    double average = sum / 12;
    printf("average = %lf\n", average);
    return 0;
}
```

//定义了12个整数

//一个一个读入

//计算12个整数的和与平均数

```c
#include <stdio.h>
int main(){
    int grade[12];
    printf("Please input 12 integers:\n");
    int i;
    for (i = 0; i < 12; i++){
        scanf("%d", &grade[i]);
    }
    double sum = 0;
    double average = 0;
    for (i = 0; i < 12; i++){
        sum += grade[i];
    }
    average = sum / 12;
    printf("average = %lf\n", average);
    return 0;
}
```

//定义了12个整数

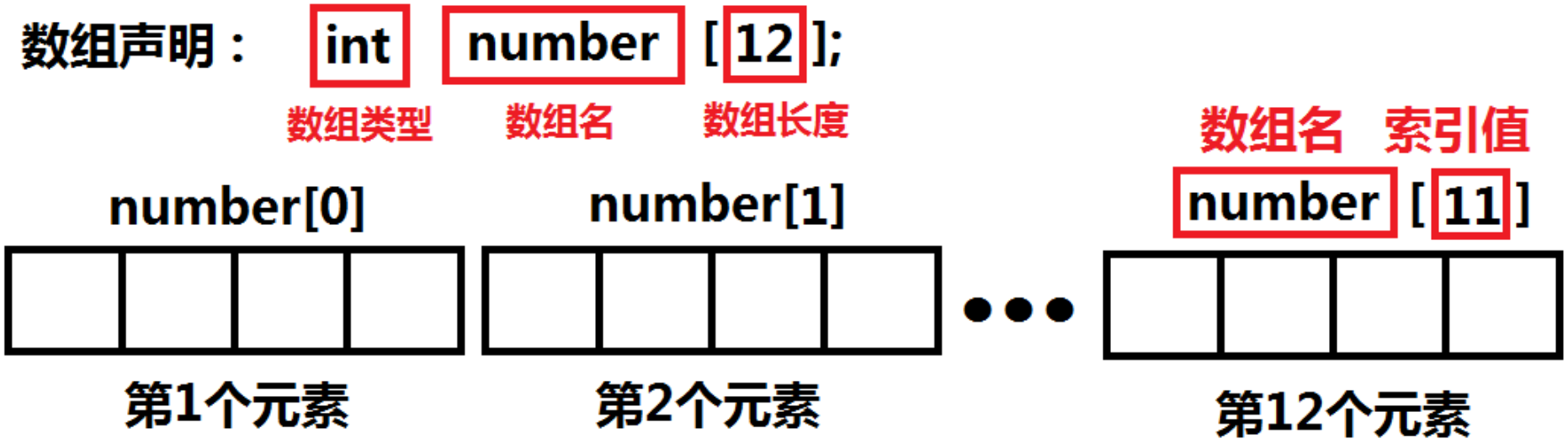
//使用一个循环读入

//计算12个整数的和与平均数

```
Please input 12 integers:
75 80 65 89 93 71 48 88 96 31 81 61
average = 73.166667
请按任意键继续. . .
```

# 数组的概念

我们经常需要在程序中存储某种类型的**大量**数据值，比如100个学生的成绩，这时我们就需要使用到**数组**。数组是一组**数目固定**、**类型相同**的数据项，数组中的元素可以是char、int、double等类似于单一数值的变量类型。




数组中的每个元素的访问均使用"**数组名**"进行访问，数组中的元素从0开始至数组长度-1，如对于**第i个元素**的访问，即为**numbers[i]**。

# 数组的赋值与使用

对于数组中的元素的**赋值、读入、输出、使用访问**与普通元素没有区别，一般使用一个循环变量(如i)对数组中的各个元素进行**访问处理**。

从键盘读入**12名**学生成绩，成绩为整型，计算这12名同学的**平均成绩**并输出？将存储学生成绩的数组全部元素再**赋值**为0，然后再将该数组中的元素进行**输出**。

```
Please input 12 integers:
75 80 65 89
93 71 48 88
96 31 81 61
The grade array:
75 80 65 89 93 71 48 88 96 31 81 61
average = 73.166667
The grade array:
0 0 0 0 0 0 0 0 0 0 0 0
```

```c
#include <stdio.h>
int main(){
    int grade[12];
    int i;
    printf("Please input 12 integers:\n");
    for (i = 0; i < 12; i++){
        scanf("%d", &grade[i]);
    }
    printf("The grade array:\n");
    for (i = 0; i < 12; i++){
        printf("%d ", grade[i]);
    }
    printf("\n");
    double sum = 0;
    double average = 0;
    for (i = 0; i < 12; i++){
        sum += grade[i];
    }
    average = sum / 12;
    printf("average = %lf\n", average);
    for (i = 0; i < 12; i++){
        grade[i] = 0;
    }
    printf("The grade array:\n");
    for (i = 0; i < 12; i++){
        printf("%d ", grade[i]);
    }
    printf("\n");
    return 0;
}
```

# 数组的初始化

数组**定义**后，可以给数组指定**初始值**，即数组的**初始化**，数组的初始化方式如**左图**。

```
1.10 2.35 6.00 7.80 9.20
1.10 2.35 6.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
1.10 2.35 6.00 7.80 9.20
2760103074568314200000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
.00 0.00
请按任意键继续. . .
```

```c
#include <stdio.h>

int main(){
    int i;

    double value1[5] = {1.1, 2.35, 6.0, 7.8, 9.2};
    for (i = 0; i < 5; i++){
        printf("%.2lf ", value1[i]);
    }
    printf("\n");

    double value2[5] = {1.1, 2.35, 6.0};
    for (i = 0; i < 5; i++){
        printf("%.2lf ", value2[i]);
    }
    printf("\n");

    double value3[5] = {0};
    for (i = 0; i < 5; i++){
        printf("%.2lf ", value3[i]);
    }
    printf("\n");

    double value4[] = {1.1, 2.35, 6.0, 7.8, 9.2};
    for (i = 0; i < 5; i++){
        printf("%.2lf ", value4[i]);
    }
    printf("\n");

    double value5[5];
    for (i = 0; i < 5; i++){
        printf("%.2lf ", value5[i]);
    }
    printf("\n");

    return 0;
}
```

方法1:声明了包含5个元素的数组，这些元素的初始值放到{}内

没有初值的元素默认设置为0

方法2:如果初值得个数(3)少于数组元素个数(5)，

方法3:直接将数组各元素的值赋值为0

方法4:在声明数组时，不提供数组打下，则数组大小为初始值的个数

若声明数组后，不赋初值，则数组中的值是随机数据

# 例1-数组的查找

**问题描述：**已知n个整数，n个数中**不存在相同的**元素。在这n个整数中查找某个整数，如果找到则输出与它相邻的元素，**没有找到或没有相邻元素**，输出NULL。

**输入与输出要求：**

首先输入一个整数n，代数组元素的个数，$1 \le n \le 1000$。然后是n个整数，每个整数的取值范围是int型范围。最后输入需要查找的数num。如果找到num输出与它相邻的元素(从左到右)，没有与它相邻的元素输出NULL。

**Input Sample:**
5
89 7890 -22 56 87
56
**Output Sample:**
-22 87

**Input Sample:**
5
89 7890 -22 56 87
87
**Output Sample:**
56

**Input Sample:**
5
89 7890 -22 56 87
17
**Output Sample:**
NULL

**Input Sample:**
1
20
20
**Output Sample:**
NULL

# 例1-思考与分析

**Input Sample:**
5
89 7890 -22 56 87
56
**Output Sample:**
-22 87

**Input Sample:**
5
89 7890 -22 56 87
87
**Output Sample:**
56

**Input Sample:**
5
89 7890 -22 56 87
17
**Output Sample:**
NULL

**Input Sample:**
1
20
20
**Output Sample:**
NULL

**思考：**

1.如何**查找**数组中某个元素?

2.数组在声明时，应该声明**多大的**数组?

3.若找到该元素后，如何保存与它**相邻**的元素?

4.当遇到边界，即若该元素是数组的**第1个元素**或**最后一个元素**，如何处理?

思考**1分钟**。

# 例1-算法设计

设置两个变量left_pos、right_pos，分别存储目标元素**左侧元素下标**与**右侧元素下标**，初始化为-1。

使用变量i**从头到尾遍历数组**，当找到目标元素时(array[i] == num)，

若 **i != 0**，记录目标元素左侧元素下标 left_pos = i - 1；

若 **i != n-1**，记录目标元素右侧元素下标 right_pos = i + 1；

若left_pos与right_pos均不为-1时，输出array[left_pos]、array[right_pos]
若left_pos不为-1、right_pos为-1时，输出array[left_pos]
若left_pos为-1、right_pos不为-1时，输出array[right_pos]
若left_pos与right_pos同时为-1时，输出NULL

**Input Sample:**
**5**
**89 7890 -22 56 87**
**56**
**Output Sample:**
**-22 87**

**Input Sample:**
**5**
**89 7890 -22 56 87**
**87**
**Output Sample:**
**56**

**Input Sample:**
**5**
**89 7890 -22 56 87**
**17**
**Output Sample:**
**NULL**

**Input Sample:**
**1**
**20**
**20**
**Output Sample:**
**NULL**

# 例1-课堂练习

```c
#include <stdio.h>
#define MAX_ARRAY_LEN 1000

int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int num;
    int left_pos = -1;
    int right_pos = -1;
    int i;
    scanf("%d", &n);                    //输入整数n，并读取n个整数至数组array
    for (i = 0; i < n; i++){
        scanf("%d", &array[i]);
    }
    scanf("%d", &num);
    for (i = 0; i < n; i++){
        if (  1  ){                     //从头到尾遍历数组，当找到目标元素
            if (i != 0){                //时，记录array[i]左边与右边的元素下标
                  2
            }
            if (  3  ){
                right_pos = i + 1;
            }
        }
    }
    if (  4  ){
        printf("%d %d\n", array[left_pos], array[right_pos]);
    }
    else if (left_pos != -1 && right_pos == -1){
        printf("%d\n", array[left_pos]);
    }
    else if (  5  ){
        printf("%d\n", array[right_pos]);
    }
    else if (left_pos == -1 && right_pos == -1){
        printf("NULL\n");
    }
    return 0;
}
```

**3分钟**，填写代码，有问题提出！

# 例1-实现与测试

```c
#include <stdio.h>
#define MAX_ARRAY_LEN 1000
int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int num;
    int left_pos = -1;
    int right_pos = -1;
    int i;
    scanf("%d", &n);                    //输入整数n，并读取n个整数至数组array
    for (i = 0; i < n; i++){
        scanf("%d", &array[i]);
    }
    scanf("%d", &num);
    for (i = 0; i < n; i++){
        if ( array[i] == num ){         //从头到尾遍历数组，当找到目标元素时，记录array[i]左边与右边的元素下标
            if (i != 0){
                left_pos = i - 1;
            }
            if ( i != n - 1 ){
                right_pos = i + 1;
            }
        }
    }
    if ( left_pos != -1 && right_pos!= -1 ){
        printf("%d %d\n", array[left_pos], array[right_pos]);
    }
    else if (left_pos != -1 && right_pos == -1){
        printf("%d\n", array[left_pos]);
    }
    else if ( left_pos == -1 && right_pos != -1 ){
        printf("%d\n", array[right_pos]);
    }
    else if (left_pos == -1 && right_pos == -1){
        printf("NULL\n");
    }
    return 0;
}
```

```
5
89 7890 -22 56 87
56
-22 87
请按任意键继续. . .
```

```
1
20
20
NULL
请按任意键继续. . .
```

# 确定数组占用的空间与元素个数

sizeof运算符可以计算出**指定类型**的变量所**占用的字节数**，同样可以计算**数组**所**占用的字节数**。利用sizeof运算计算**数组占用的字节数**后，再除以数组类型占用的字节数，可以得到数组**元素个数**。

```
double array value takes 40 bytes.
int array number takes 32 bytes.
int array number has 8 elements.
请按任意键继续. . .
```

```c
#include <stdio.h>

int main(){
    double value[5];

    printf("double array value takes %d bytes.\n", sizeof(value));

    int number[] = {1, 2, 3, 4, 5, 6, 7, 8};
    printf("int array number takes %d bytes.\n", sizeof(number));

    int element_count = sizeof(number) / sizeof(int);
    printf("int array number has %d elements.\n", element_count);

    return 0;
}
```

# 数组的地址

数组名number指定了**数组的地址**，即为数组**第一个元素**的地址&number[0]。由于编译器为数组分配的空间是**连续的**，数组后续元素的**地址**，即为数组**首元素的地址**(数值) + 该元素的**数组索引值** * **类型所占空间**。

```
array_address = 0028FF10, 0028FF10
element 0 address 0028FF10 == 0028FF10 values 0.
element 1 address 0028FF14 == 0028FF14 values 10.
element 2 address 0028FF18 == 0028FF18 values 20.
element 3 address 0028FF1C == 0028FF1C values 30.
element 4 address 0028FF20 == 0028FF20 values 40.
element 5 address 0028FF24 == 0028FF24 values 50.
element 6 address 0028FF28 == 0028FF28 values 60.
element 7 address 0028FF2C == 0028FF2C values 70.
element 8 address 0028FF30 == 0028FF30 values 80.
element 9 address 0028FF34 == 0028FF34 values 90.
element 10 address 0028FF38 == 0028FF38 values 100.
element 11 address 0028FF3C == 0028FF3C values 110.
请按任意键继续. . .
```

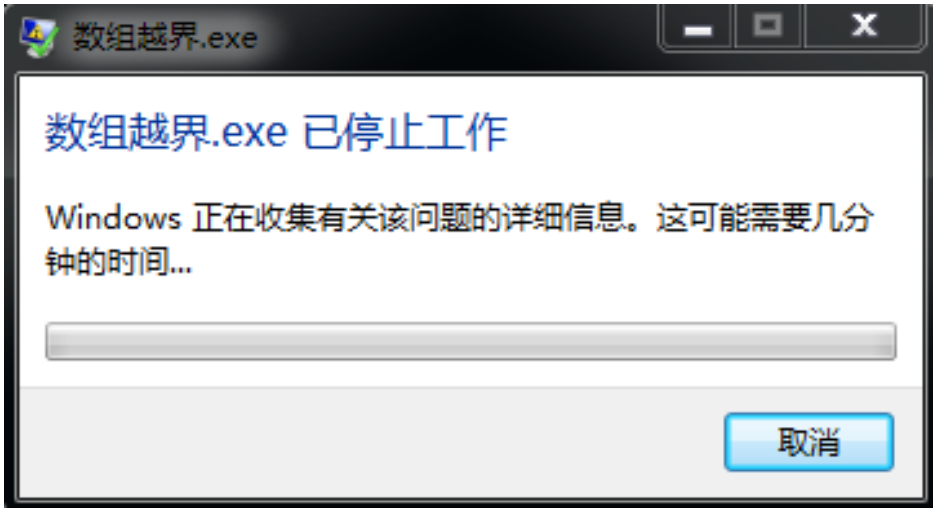
```c
#include <stdio.h>
int main(){
    int number[12] = {0};
    int i;
    for (i = 0; i < 12; i++){
        number[i] = i * 10;
    }
    //打印数组地址(首元素地址)
    printf("array_address = %p, %p\n", number, &number[0]);
    for (i = 0; i < 12; i++){
        //通过计算，得到数组各元素地址，注意 必须将number的值强转为(int)
        int address = (int)number + sizeof(int) * i;
        printf("element %d address %p == %p values %d.\n"
            , i, &number[i], address, number[i]);
    }
    return 0;
}
```

# 数组越界

由于可以通过变量**索引**(访问)数组中的各个元素，那么当索引值超过**数组的大小**时，就会发生**数组越界**。数组越界问题是程序中**最为常见的bug**，它无法在程序**编译时**检查出。当数组越界后，程序可能**发生崩溃**，也可能计算出**错误的结果**。
一般来讲这种bug有时候是**难以避免甚至无法避免**的(毕竟程序员都是**人类**)，我们一般使用**大规模集成化的测试**来检验程序是否有bug(例如排查数组越界问题)。

```c
#include <stdio.h>

int main(){
    int number[12] = {0};
    int i;
    for (i = 0; i < 20; i++){
        number[i] = i;
    }
    return 0;
}
```



# 变长数组

一般来讲，以**静态方式**定义数组(**编译时即分配内存**，非程序运行时分配内存)，**数组的长度**必须由**常量**指定。所以我们开发程序时需要**提前估计**需要使用的数组大小(对问题规模进行估计后，定义**大小合适**的数组)。在一些非严格执行**C11标准**(C11标准是C语言标准的第三版，2011年由ISO/IEC发布，前一个标准版本是C99标准。)的C语言编译器下，支持以静态方式定义**变长数组**，即数组的长度由**变量**指定。我们**并不建议**使用非标准的C语言开发方式(因为很有可能某编译器不支持该方式，却可以**编译通过**，**运行时出bug**。

```c
#include <stdio.h>
int main(){
    int n;
    printf("Please input scores number:\n");
    scanf("%d", &n);

#ifdef __STDC_NO_VLA__
    printf("Variable length arrays are not supported.\n");
    return 1;
#endif

    int grade[n];
    int i;

    for (i = 0; i < n; i++){
        scanf("%d", &grade[i]);
    }
    double sum = 0;
    double average = 0;
    for (i = 0; i < n; i++){
        sum += grade[i];
    }
    average = sum / n;
    printf("average = %lf\n", average);
    return 0;
}
```

//虽然不建议使用静态方式定义变长数组
但一定要用，务必加入该宏定义

//静态方式定义变长数组，对于一些"老C语言程序员"，
看到 这样的代码其实对于他们是很奇怪的，
潜意识下认为是错误的

//在后面的课程中，我们会介绍动态内存分配来解决问题规模不确定的情况下数组的申请问题。

```
Please input scores number:
12
75 80 65 89
93 71 48 88
96 31 81 61
average = 73.166667
请按任意键继续. . .
```

# 例2-最长平台

**问题描述：**给定一个长度为n的整数数组，数组中**连续的相等元素**构成的子序列称为**平台**。求出数组中**最长平台**的长度。

**输入与输出要求：**

首先输入一个整数n，代数组元素的个数，$1 \le n \le 1000$。然后是n个整数，每个整数的取值范围是**int型范围**。输出最长平台的长度，占一行。

**Input Sample:**

**15**

**1 1 2 2 4 5 7 7 7 7 7 3 3 9**

**Output Sample:**

**6**

最长平台

**1 1 2 2 4 5 7 7 7 7 7 7 3 3 9**

**Input Sample:**

**7**

**102 102 102 102 102 102 102**

**Output Sample:**

**7**

最长平台

**102 102 102 102 102 102 102**

# 例2-思考与分析


**思考:**

1.若要找到**最长平台**，是否需要遍历数组的**全部元素**。

2.在遍历数组时，如何记录**连续的相同元素**组成的**平台**。

3.在遍历数组时，如何记录连续的相同元素组成的**最长**的平台。

4.**整体算法**如何设计。

5.**边界**问题如何考虑。

思考**1分钟**。

# 例2-算法设计


1.初步思路:

**从头到尾**遍历数组，找出**最长的**连续相同元素组成的"平台"。

2.遍历过程中需要记录的数据:

1)current_length：遍历数组时，记录以**当前元素**为**结尾**的**连续的相同元素**组成的平台的长度。

2)max_length：遍历数组时，记录遍历**到当前元素时最长的**平台的长度。

3.核心算法:

使用变量i**从头至尾**遍历数组：

如果当前数组元素array[i] == array[i-1]：

current_length++；(当前元素为结尾的平台长度+1)

否则：

current_length = 1；(以当前元素为结尾成为一个新的平台)

遍历过程中，如果max_length < current_length，将max_length**更新**为current_length。

4.边界条件:

从数组的**第二个元素**开始遍历，初始时current_length = 1；max_length = 1。

# 例2-算法设计

初始时：current_length = 1 max_length = 1

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

图1：current_length = 2 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 1

图2：current_length = 1 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 2

图3：current_length = 2 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 3

图4：current_length = 1 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 4

图5：current_length = 1 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 5

图6：current_length = 1 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 6

图7：current_length = 2 max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 7

图8：current_length = 3 max_length = 3

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 8

图9：current_length = 4 max_length = 4

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 9

# 例2-课堂练习

```
#include <stdio.h>
#define MAX_ARRAY_LEN 1000

int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int current_length = 1;
    int max_length = 1;
    int i;
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        scanf("%d", 1 );
    }
    for ( 2 ; i < n; i++){
        if ( 3 ){
            current_length++;
        }
        else{
            4
        }
        if ( 5 ){
            max_length = current_length;
        }
    }
    printf("%d\n", max_length);
    return 0;
}
```

current_length = 1  max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 5

current_length = 1  max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 6

current_length = 2  max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

⇧ i = 7

3分钟，填写代码，有问题提出！

# 例2-实现与测试

```c
#include <stdio.h>
#define MAX_ARRAY_LEN 1000

int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int current_length = 1;
    int max_length = 1;
    int i;
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        scanf("%d", &array[i]);
    }
    for (i = 1; i < n; i++){
        if (array[i] == array[i-1]){
            current_length++;
        }
        else{
            current_length = 1;
        }
        if (max_length < current_length){
            max_length = current_length;
        }
    }
    printf("%d\n", max_length);
    return 0;
}
```

current_length = 1  max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

i = 5

current_length = 1  max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

i = 6

current_length = 2  max_length = 2

1 1 2 2 4 5 7 7 7 7 7 7 3 3 9

i = 7

```
15
1 1 2 2 4 5 7 7 7 7 7 7 3 3 9
6
请按任意键继续. . .
```

# 例3-基于交换的排序

**问题描述：**给定一个长度为n的整数数组，对该数组进行**从小到大的排序**。
输入与输出要求：
首先输入一个整数n，代数组元素的个数，$1 \le n \le 1000$。然后是n个整数，每个整数的取值范围是int型范围。输出这n个数**从小到大的排序结果**占一行。

**Input Sample:**

**8**

**49 38 65 97 76 13 27 49**

**Output Sample:**

**13 27 38 49 49 65 76 97**

**Input Sample:**

**15**

**273 -293 2132 0 10000 -213 -23 1 1 999 11 23 87 67 56**

**Output Sample:**

**-293 -213 -23 0 1 1 11 23 56 67 87 273 999 2132 10000**

**思考：**

1.若要对数组进行排序，至少需要**几层循环**？
2.若使用2层循环，**外层循环**做什么，**内层循环**做什么？

# 例3-算法设计

**核心算法:**

使用变量i遍历数组，对于数组中的每一个元素a[i]，利用变量j，遍历a[i]后面(i+1, i+2,...,n-1)的元素a[j]，若a[i] > a[j]，则交换a[i]与a[j]两元素的位置。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

i = 0 ( 49 38 65 97 76 13 27 49 )

j = 1(交换a[0]与a[1])

38 49 65 97 76 13 27 49

j = 2,3,4 (数组无变化)

j = 5(交换a[0]与a[6])

13 49 65 97 76 38 27 49

j = 6,7(数组无变化)

13 49 65 97 76 38 27 49

已完成

i = 1 (13 49 65 97 76 38 27 49)

j = 2,3,4(数组无变化)

j = 5(交换a[1]与a[5])

13 38 65 97 76 49 27 49

j = 6(交换a[1]与a[6])

13 27 65 97 76 49 38 49

j = 7(数组无变化)

13 27 65 97 76 49 38 49

已完成

i = 2 (13 27 65 97 76 49 38 49)

j = 3,4(数组无变化)

j = 5(交换a[2]与a[5])

13 27 49 97 76 65 38 49

j = 6(交换a[2]与a[6])

13 27 38 97 76 65 49 49

j = 7(数组无变化)

13 27 38 97 76 65 49 49

已完成

# 例3-课堂练习

```
#include <stdio.h>
#define MAX_ARRAY_LEN 1000

int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int i;
    int j;
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        scanf("%d", &array[i]);
    }
    for (i = 0; i < n; i++){
        for (j = 1; j < n; j++){
            if ( 2 ){
                int temp = 3
                array[i] = 4
                5
            }
        }
    }
    for (i = 0; i < n; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
    return 0;
}
```

**核心算法:**

使用变量i遍历数组，对于数组中的**每一个元素**a[i]，遍历该元素a[i]后面(i+1, i+2,...,n-1)的元素a[j]，若a[i] > a[j]，则**交换**a[i]与a[j]两元素的**位置**。

**3分钟**，填写代码，有问题提出！

# 例3-实现与测试

```c
#include <stdio.h>
#define MAX_ARRAY_LEN 1000

int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int i;
    int j;
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        scanf("%d", &array[i]);
    }
    for (i = 0; i < n; i++){
        for (j = i + 1; j < n; j++){
            if (array[i] > array[j]){
                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
    }
    for (i = 0; i < n; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
    return 0;
}
```

**核心算法:**

使用变量i遍历数组，对于数组中的**每一个元素**a[i]，遍历该元素a[i]后面(i+1, i+2,...,n-1)的元素a[j]，若a[i] > a[j]，则**交换**a[i]与a[j]两元素的**位置**。

```
8
49 38 65 97 76 13 27 49
13 27 38 49 49 65 76 97
请按任意键继续. . .
```

# 课间休息10分钟！

有问题提出！

# 第二部分：函数初步

C语言中的一个**重要观念**是，将每个程序按照各个**小功能切割**为许多小函数，当每个小功能对应的**函数**正确工作时，**程序整体**就能够正确工作。这么做的好处是使得整体代码的易于**升级、测试、维护**，提升程序代码的**可读性**，代码看起来更加的**清爽**。

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

int main(){
    int number;
    scanf("%d", &number);
    int i, j;
    for (i = 2; i <= number / 2; i++){
        int prime1 = i;
        int max_factor = sqrt(prime1) + 1;
        for (j = 2; j < max_factor; j++){
            if (prime1 % j == 0){
                prime1 = 0;
                break;
            }
        }
        int prime2 = number - i;
        max_factor = sqrt(prime2) + 1;
        for (j = 2; j < max_factor; j++){
            if (prime2 % j == 0){
                prime2 = 0;
                break;
            }
        }
        if (prime1 && prime2){
            printf("%d = %d + %d\n", number,
                    prime1, prime2);
        }
    }
    return 0;
}
```

判断i是否为素数

判断number-i是否为素数

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

int is_prime(int num){
    int max_factor = sqrt(num) + 1;
    int i;
    for (i = 2; i < max_factor; i++){
        if (num % i == 0){
            return 0;
        }
    }
    return 1;
}
int main(){
    int number;
    scanf("%d", &number);
    int i, j;
    for (i = 2; i <= number / 2; i++){
        if (is_prime(i) && is_prime(number - i)){
            printf("%d = %d + %d\n",
                    number, i, number - i);
        }
    }
    return 0;
}
```

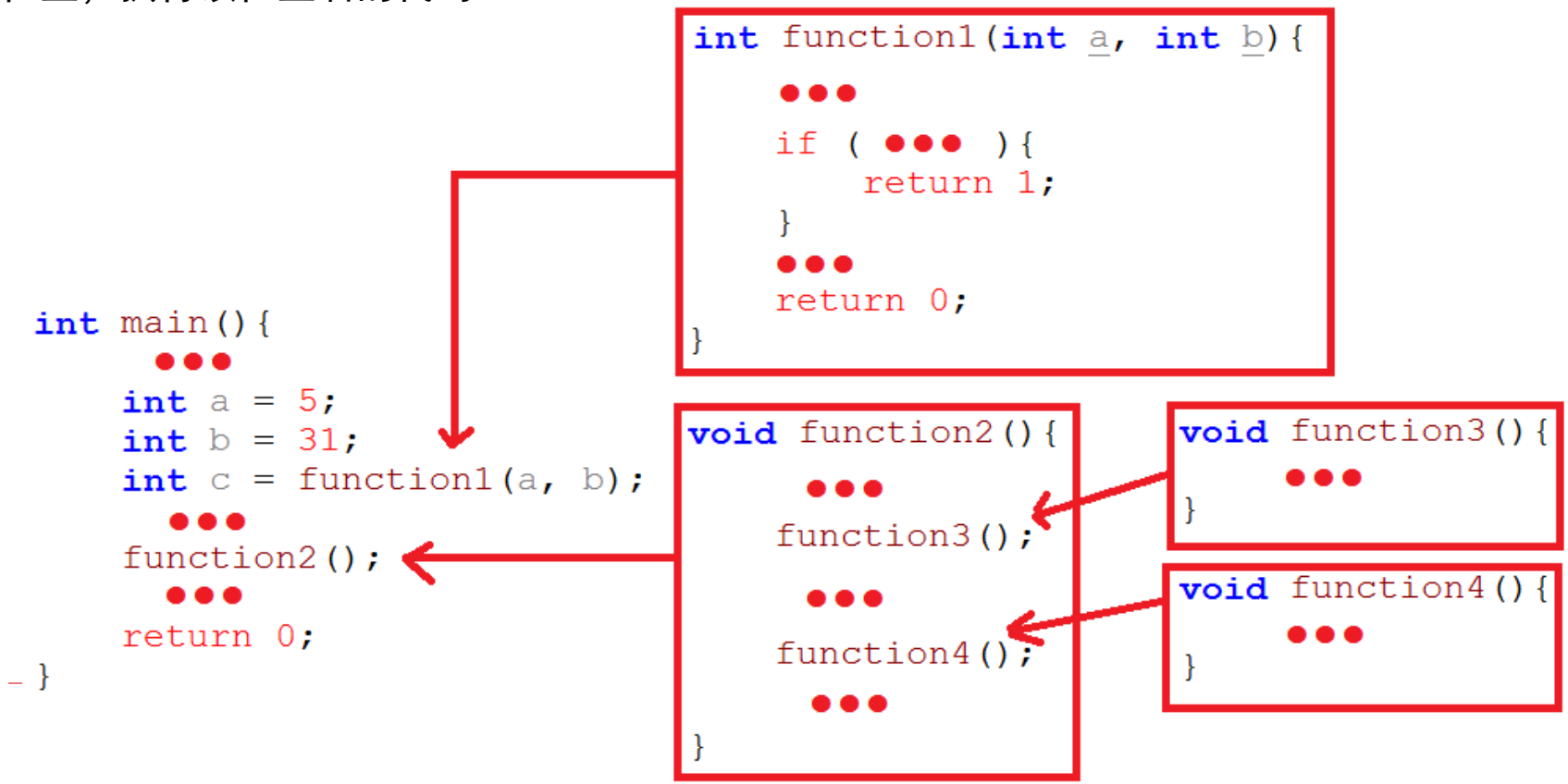
一个判断num是否为素数的函数

调用函数is_prime判断i与number-i是否为素数

# 程序的结构

一般C程序由**许多**函数组成，其中最重要的是**main函数**，它是程序执行的**起点**；函数中可以调用任何函数，包括它**本身**。**调用**一个函数时，就是执行该函数体内的代码，这个函数**执行结束后**，就**回到调用**该函数的地方。

程序依然是按照语句顺序**顺序执行**，当调用某函数时，在**调用位置**将**参数值**传递给函数，之后从该函数的第一句代码开始执行直到执行到**return语句**或到达这个函数体的**结束括号**，返回调用它的位置，执行该位置后的代码。



# 函数的定义:名称、参数、返回值

函数定义时，需要指定**函数的名称**、**函数参数**、**函数返回值的类型**。
**函数的名称**规则遵循**变量的命名规则**；
**函数的参数**是**一列变量名称**与**它们的类型**，函数参数之间使用**逗号分隔**，函数也可以**没有参数**，参数的作用就将调用代码中的**必要信息传递**给被调用函数；
**函数返回值类型**即函数调用完成后，返回给调用代码一个**该类型的值**，任何**合法的变量类型**都可以作为函数返回值的类型，函数也可以没有返回值，即void。函数返回时，使用**return语句**，return后紧跟一个与函数返回值类型值相同类型的值。

**函数的一般形式如下:**

```
返回值类型 函数名(参数1, 参数2, ..., 参数n){
    // 函数体内的程序
}
```

```
3 ^ 5 = 243.000000
请按任意键继续. . .
```

```c
#include <stdio.h>

double my_pow(double x, int y){
    double result = 1;
    int i;
    for (i = 1; i <= y; i++){
        result = result * x;
    }
    return result;
}

int main(){
    printf("3 ^ 5 = %lf\n", my_pow(3, 5));
    return 0;
}
```

函数返回值类型　函数的名称　函数参数列表

//计算x的y次方

# 例4-无重复数字

已知正整数a、b满足a<=b。你的任务是求出区间[a,b]内，所有满足以下条件整数:
1.该整数由1到n这n个数字组成。
2.整数中各个位上的数字不相同。

**输入与输出要求：**

输入三个正整数a、b、n，代表所求区间，满足1<=a<=b<=2000，且1<=n<=9。
输出满足条件的整数，每五个数为一行，整数之间用tab分隔，最后一个数后为换行符。
每组数据再多输出一个空行。当该区间没有符合条件的整数时，输出“There is no proper number in the interval.”

**Input:**

**200 500 4**

**Output:**

**213 214 231 234 241**
**243 312 314 321 324**
**341 342 412 413 421**
**423 431 432**

**Input:**

**1000 2000 3**

**Output:**

**There is no proper number in the interval.**

**Input:**

**1 50 2**

**Output:**

**1 2 12 21**

# 例4-思考与分析

**Input:**

**200 500 4**

**Output:**

| | | | | |
|---|---|---|---|---|
| 213 | 214 | 231 | 234 | 241 |
| 243 | 312 | 314 | 321 | 324 |
| 341 | 342 | 412 | 413 | 421 |
| 423 | 431 | 432 | | |

**Input:**

**1000 2000 3**

**Output:**

**There is no proper number in the interval.**

**Input:**

**1 50 2**

**Output:**

**1 2 12 21**

**思考：**

1.判断整数是否**满足条件**，需要对整数按照各个位的数字进行**拆分**，如何对一个整数进行**拆分**，然后判断该整数中是否有**重复数字**？并且保证数字在1-n的范围内？

2.如何组织代码，**设计函数**，使得程序更加**简洁**？函数的**参数**与**返回值**是什么？

3.如何按照题目要求的**输出格式**，每行5个元素进行打印？

思考**1分钟**。

# 例4-算法设计1，整体

**Input:**
**200 500 4**
**Output:**
**213 214 231 234 241**
**243 312 314 321 324**
**341 342 412 413 421**
**423 431 432**

**Input:**
**1000 2000 3**
**Output:**
**There is no proper number in the interval.**

**Input:**
**1 50 2**
**Output:**
**1 2 12 21**

1.整体算法，遍历**区间范围**内的数字，对每个数字是否**满足条件**进行判断，设置**count计数器**记录满足条件的数字个数，对每个满足条件的数字进行输出，每输出5个元素即打印一个换行符。
2.设计函数，**int is_the_digit_ok(int num, int n)**:
(输入)num:**待判断**的整数。
(输入)n:判断整数num中的**各个数字**是否在**1-n之间**。
返回值:若num**满足条件**:1.该整数由**1到n**这n个数字组成；2.整数中各个位上的**数字不相同**两个条件，则返回1，否则返回0。

# 例4-算法设计2，判断符合条件的整数

**Input:**

**200 500 4**

**Output:**

| | | | | |
|---|---|---|---|---|
| 213 | 214 | 231 | 234 | 241 |
| 243 | 312 | 314 | 321 | 324 |
| 341 | 342 | 412 | 413 | 421 |
| 423 | 431 | 432 | | |

**2234：拆分出4,3,2,2**

**2234->223，拆分出4**

**223->22，拆分出3**

**22->2，拆分出2**

**2->0，拆分出2**

**利用mark数组记录:**

**mark[1]记录数字1的个数；**

**…**

**mark[digit]记录数字digit的个数；**

**…**

**mark[9]记录数字9的个数；**

1.**数字拆分**:拆分时通过**对10取余**得到个位数字digit，再利用循环将整数num不断的**对10整除**，直到num为0。

2.**数字统计**:利用数组mark，对digit(1-9)的**个数统计**。统计的过程中，同时对digit进行判断，当遇到digit不**满足条件**时，函数返回0。

3.若num上的各个数字都**满足条件**，则函数最终返回1。

# 例4-重复数字判断，课堂练习

```
//判断数字num，是否满足条件，满足返回1，不满足
int is_the_digit_ok(int num, int n){
    int mark[10] = {0}; //使用mark数组记录0-9十个数字，是否出现过
    while(num){ //循环拆分num，一个数字一个数字的拆分出
        int digit = 1
        if (2){
            return 0;
        }
        3
        num = num / 10; //拆分下一个数字，从最低位到最高位进行拆分
    }
    return 1; //当返回1时，代表满足条件
}
```

# 例4-重复数字判断，实现

```
//判断数字num，是否满足条件，满足返回1，不满足
int is_the_digit_ok(int num, int n){
    int mark[10] = {0}; //使用mark数组记录0-9十个数字，是否出现过
    while(num){ //循环拆分num，一个数字一个数字的拆分出
        int digit = num % 10;
        if (digit < 1 || digit > n || mark[digit]){
            return 0;
        }
        mark[digit] = 1;
        num = num / 10; //拆分下一个数字，从最低位到最高位进行拆分
    }
    return 1; //当返回1时，代表满足条件
}
```

# 例4-整体代码实现与测试

```
int main(){
    int a;
    int b;
    int n;
    scanf("%d %d %d", &a, &b, &n);
    int i;
    int count = 0;  //记录已打印了多少个数字
    for (i = a; i <= b; i++){
        if (is_the_digit_ok(i, n)){ //如果i满足要求，打印i
            printf("%d", i);
            count++;
            if (count % 5 == 0){  //每打印5个数字，则打印一个空行
                printf("\n");
            }
            else{    //每个数字后打印一个tab字符
                printf("\t");
            }
        }
    }
    if (count == 0){  //如果一个数字都未打印，打印提示字符串
        printf("There is no proper number in the interval.\n");
    }
    printf("\n");
    return 0;
}
```

```
200 500 4
213     214     231     234     241
243     312     314     321     324
341     342     412     413     421
423     431     432
请按任意键继续. . .
```

# 数组作为参数的函数

**函数的参数**不仅可以是普通的变量，也可以将**数组**作为函数的参数，当需要把**数组作为参数**传递给函数时，还必须传递一个**额外的**变量，即**数组的长度**。没有该参数，函数就无法知道数组中有**多少个元素**可以使用了。

```
#include <stdio.h>
#define MAX_ARRAY_LEN 1000

                    //待排序数组
void sort_array(int array[], int n){
    int i;
    int j;                          //数组的有效元素个数n
    for (i = 0; i < n; i++){
        for (j = i + 1; j < n; j++){
            if (array[i] > array[j]){
                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
    }
}
```

```
int main(){
    int array[MAX_ARRAY_LEN] = {0};
    int n;
    int i;
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        scanf("%d", &array[i]);
    }
    sort_array(array, n);
    for (i = 0; i < n; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
    return 0;
}
```

```
8
49 38 65 97 76 13 27 49
13 27 38 49 49 65 76 97
请按任意键继续. . .
```

# 按值传递机制

调用程序在给函数**传递参数**时，调用程序中的变量**不会直接传递**给函数，会先制作变量的**副本(拷贝)**，存储在操作系统的**栈空间**中，函数内部使用这个**变量的拷贝**，所以无论函数内部怎样修改该参数，都**不会改变它在调用时存储的值**，这种传递方式被称为**按值传递**。

当数组作为参数时，是传递的**数组的地址**，所以函数内部可以**修改数组中的值**。我们不能利用普通变量作为参数**向外传递**函数的计算结果(后续的知识指针变量可以)，但可以**使用数组向外传递结果**。

```c
#include <stdio.h>

void modify_variable(int num){
    num = 999;
}

void modify_array(int array[], int n){
    int i;
    for (i = 0; i < n; i++){
        array[i] = 999;
    }
}
```

```c
int main(){
    int a = 0;
    int b[10] = {0};
    int i;
    printf("a = %d\n", a);
    printf("b = ");
    for (i = 0; i < 10; i++){
        printf("%d ", b[i]);
    }
    printf("\n");
    printf("after modify:\n");
    modify_variable(a);
    modify_array(b, 5);
    printf("a = %d\n", a);
    printf("b = ");
    for (i = 0; i < 10; i++){
        printf("%d ", b[i]);
    }
    printf("\n");
    return 0;
}
```

```
a = 0
b = 0 0 0 0 0 0 0 0 0 0
after modify:
a = 0
b = 999 999 999 999 999 0 0 0 0 0
请按任意键继续. . .
```

# 例5-因式分解

**问题描述:**

每个大于1的正整数都能表示为**素数的乘积**，将一个正整数表示为素数的乘积被称为**因式分解**。例如，数字60可以分解为2*2*3*5，其中每一个**约数**都是素数。注意，**同一个素数**在因数分解中**可以出现多次**。设计程序，求一段数据范围[a, b]内的各个数字的因式分解。

**输入与输出要求:**

输入两个正整数a、b，代表所分解的区间，满足1<=a<=b<=100000，输出a到b之间的正整数的因式分解。因式分解中的因子相乘时，**从小至大**相乘。

**Input Sample:**
**57 63**
**Output Sample:**
**57 = 3*19**
**58 = 2*29**
**59 = 59**
**60 = 2*2*3*5**
**61 = 61**
**62 = 2*31**
**63 = 3*3*7**

**Input Sample:**
**99990 99996**
**Output Sample:**
**99990 = 2*3*3*5*11*101**
**99991 = 99991**
**99992 = 2*2*2*29*431**
**99993 = 3*33331**
**99994 = 2*17*17*173**
**99995 = 5*7*2857**
**99996 = 2*2*3*13*641**

**思考:**

因式分解整体程序，需要由**几部分**功能组成？如何将它们开发为**函数**，对每个函数进行**算法设计**。

# 例5-算法设计(整体)

程序**核心算法**:对整数num进行因式分解并打印出。包括2部分:
1.将num的所有**素数因子分解**出来，存储至**factor数组**；在分解num的所有素数因子时，需要有一个**素数列表**。
按照素数**从小到大**的顺序，**循环的遍历**素数列表，若num可被某素数prime**整除**，则该素数是num的因子，num = num / prime，**重新进行**素数因子的从小到大试探，直到num为1停止循环。

**举例:**
素数列表2,3,5,7；分解num = 20:
20%2 == 0，num = 20/2 = 10；
10%2 == 0，num = 10/2 = 5；
5%5 == 0，num = 5/5 = 1；
故20有2、2、5三个素数因子。

2.给出num与它的因子**factor数组**，按照因式分解的格式进行打印。
如打印20的因式分解: 20 = 2*2*5

# 例5-函数设计(整体)

**20 = 2*2*5**

1.将num的**所有素数因子分解**出来，存储至factor数组；在分解num的所有素数因子时，需要有一个**素数列表**。

设计函数:

**int get_factor(int num, int factor[], int prime[], int prime_cnt)**

(输入)num:待因式分解的整数。

(输出)factor:分解出的结果保存在factor中。

(输入)prime:足够大的素数列表，保存了1-100000的所有素数(因为num的取值是该范围)。

(输入)prime_cnt:素数数组中的元素个数。

返回值:因式分解后，factor数组存储的因子个数。

2.给出num与它的因子factor数组，**按照因式分解的格式**进行打印。

**void print_factor(int num, int factor[], int factor_cnt)**

(输入)num:待因式分解的整数。

(输入)factor:整数num的因子数组。

(输入)factor_cnt:数组factor中的元素个数(因子个数)。

```c
#include <math.h>
```

//判断num是否是素数是返回1，否则返回0

```c
int is_prime(int num){
    int max_factor = sqrt(num) + 1;
    int i;
    for (i = 2; i < max_factor; i++){
        if (num % i == 0){
            return 0;
        }
    }
    return 1;
}
```

//返回获取的素数个数

//输出的素数列表

//本题n == 100000

```c
int get_prime_list( int prime[] , int n ){
    // 获取从2到n的素数，存储至prime数组中
    int prime_cnt = 0;
    int i;
    for (i = 2; i <= n; i++){
        if ( 1 ){
            2
            prime_cnt++;
        }
    }
    3
}
```

//如n = 10

则prime[0] = 2

prime[1] = 3

prime[2] = 5

prime[3] =7

函数返回值为4

# 例5-获取素数列表，课堂练习

3分钟，填写代码，有问题提出！

# 例5-获取素数列表，实现

```c
#include <math.h>
//判断num是否是素数是返回1，否则返回0
int is_prime(int num){
    int max_factor = sqrt(num) + 1;
    int i;
    for (i = 2; i < max_factor; i++){
        if (num % i == 0){
            return 0;
        }
    }
    return 1;
}
//输出的素数列表
//本题n == 100000
//返回获取的素数个数
int get_prime_list( int prime[] , int n ){
    // 获取从2到n的素数，存储至prime数组中
    int prime_cnt = 0;
    int i;
    for (i = 2; i <= n; i++){
        if ( is_prime(i) ){
            prime[prime_cnt] = i;
            prime_cnt++;
        }
    }
    return prime_cnt;
}
```

//如n = 10
则prime[0] = 2
prime[1] = 3
prime[2] = 5
prime[3] =7
函数返回值为4

# 例5-因式分解函数，课堂练习

```
//将num的所有素数因子分解出来，存储至factor数组
int get_factor(int num, int factor[], int prime[], int prime_cnt){
//返回分解出了多少个素数因子                    //素数列表与素数列表中元素的个数
    int factor_cnt = 0;
    int i;
    while( 1 ){
        for (i = 0; i < prime_cnt; i++){
            if ( 2 ){
                factor[factor_cnt] = 3
                4
                factor_cnt++;
                5
            }
        }
    }
    return factor_cnt;
}
```

**3分钟**，填写代码，有问题提出！

程序**核心算法**:
按照素数**从小到大**的顺序，**循环的遍历**素数列表，若num可被某素数prime**整除**，则该整数是num的因子，num = num / prime，直到num为1停止循环。

**举例:**
素数列表2,3,5,7；分解num = 20:
20%2 == 0，num = 20/2 = 10；
10%2 == 0，num = 10/2 = 5；
5%5 == 0，num = 5/5 = 1；
故20有2、2、5三个素数因子。

# 例5-因式分解函数，实现

```
//将num的所有素数因子分解出来，存储至factor数组
int get_factor(int num, int factor[], int prime[], int prime_cnt){
//返回分解出了多少个素数因子                    //素数列表与素数列表中元素的个数
    int factor_cnt = 0;
    int i;
    while( num != 1 ){
        for (i = 0; i < prime_cnt; i++){
            if ( num % prime[i] == 0 ){
                factor[factor_cnt] = prime[i];
                num = num / prime[i];
                factor_cnt++;
                break;
            }
        }
    }
    return factor_cnt;
}
```

程序**核心算法**:

按照素数**从小到大**的顺序，**循环的遍历**素数列表，若num可被某素数prime**整除**，则该整数是num的因子，num = num / prime，直到num为1停止循环。

**举例:**

素数列表2,3,5,7；分解num = 20:

20%2 == 0，num = 20/2 = 10；

10%2 == 0，num = 10/2 = 5；

5%5 == 0，num = 5/5 = 1；

故20有2、2、5三个素数因子。

# 例5-打印分解结果与整体代码

```
void print_factor(int num, int factor[], int factor_cnt){
    int i;
    printf("%d = ", num);
    for (i = 0; i < factor_cnt - 1; i++){
        printf("%d*", factor[i]);
    }
    printf("%d\n", factor[factor_cnt - 1]);
}
int main(){
    int a;
    int b;
    int i;
    scanf("%d %d", &a, &b);
    int prime[10000];
    int prime_cnt = get_prime_list(prime, 100000);
    int factor[100];
    int factor_cnt;
    for (i = a; i <= b; i++){
        factor_cnt = get_factor(i, factor, prime, prime_cnt);
        print_factor(i, factor, factor_cnt);
    }
    return 0;
}
```

```
57 63
57 = 3*19
58 = 2*29
59 = 59
60 = 2*2*3*5
61 = 61
62 = 2*31
63 = 3*3*7
请按任意键继续. . .
```

**//1-100000的素数不到10000个，**
**可以预先跑出来后，看看有多少个再开**
**个合适大小的数组**

# 函数的声明:函数原型

我们之前开发的程序，**子函数**都放在main函数之前，若**先摆放main函数**代码，后摆放子函数的实现代码，需要**预先声明函数**，即**函数原型**；否则出现**未定义的函数**错误。**函数原型**是定义函数基本特性的语句，包括函数的名称、函数的参数列表、函数的返回值，即使只声明函数原型，不给出实现，C程序也可以编译通过，但会发生**链接错误**。

```c
#include <stdio.h>     //函数的原型
double my_pow(double x, int y);

int main(){
    printf("3 ^ 5 = %lf\n",
                    my_pow(3, 5));
    return 0;
}                      //函数的实现

double my_pow(double x, int y){
    double result = 1;
    int i;
    for (i = 1; i <= y; i++){
        result = result * x;
    }
    return result;
}
```

```c
#include <stdio.h>

double my_pow(double x, int y);

int main(){
    printf("3 ^ 5 = %lf\n", my_pow(3, 5));
    return 0;
}
```

构建

```
正在连接...
[Error] C:\Users\Desktop\C语言第二课_2017_12_30\第二部分\函数原型1.c:6: undefined reference to `my_pow'
[Error] collect2: ld returned 1 exit status

构建中止 函数原型1: 2 个错误，0 个警告
```

**链接错误！这个错误将是未来做C工程最常见的错误之一，undefined reference to "something"**

# 变量的作用域与生存期

我们一般将变量声明在main()函数的**起始处**，实际上，我们可以将变量定义在**任意代码块**的起始处或其他地方。这其中有**很大的不同**，因为变量只存在于定义他们的块中，在**声明时创建**，遇到**下一个闭括号**就会**销毁**。这种变量也称为**自动变量**，使用的是操作系统的栈空间。每个函数体也都是一个块，在一个函数内声明的自动变量是这个函数的**本地变量**，在**其他地方是不存在的**。

```
#include <stdio.h>

int main(){
    int i;
    for (i = 0; i < 10; i++){
        int i = 0;
        printf("i = %d\n", i);
    }
    printf("i = %d\n", i);
    return 0;
}
```

```
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 10
请按任意键继续. . .
```

# 递归函数

函数代码中**调用自己**时称为**递归**，该函数被称为**递归函数**。递归函数是一个很高效的开发技巧，可以极大的**简化代码**提高开发效率。递归函数与循环类似，循环可以完成的事情，递归函数都可以完成，并且对于一些复杂的问题，递归函数的实现**代码更简单**。

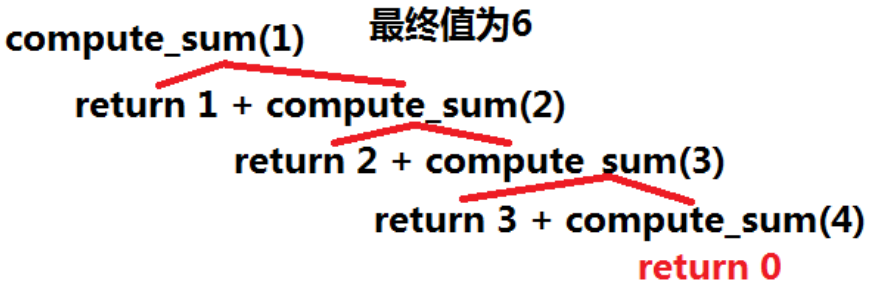
**递归函数的一般形式:**

```
void func(){
    if (...){   //需要有递归结束条件，
        ...     否则会陷入死循环
        return;
    }     //自己调用自己
    func();
}
```

```c
#include <stdio.h>

int compute_sum(int i){
    if (i > 3){
        return 0;
    }
    return i + compute_sum(i+1);
}

int main(){
    printf("%d\n", compute_sum(1));
    return 0;
}
```




```
6
请按任意键继续. . .
```

# 结束

**非常感谢大家！**

**林沐**

# 联系我们

## 小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**