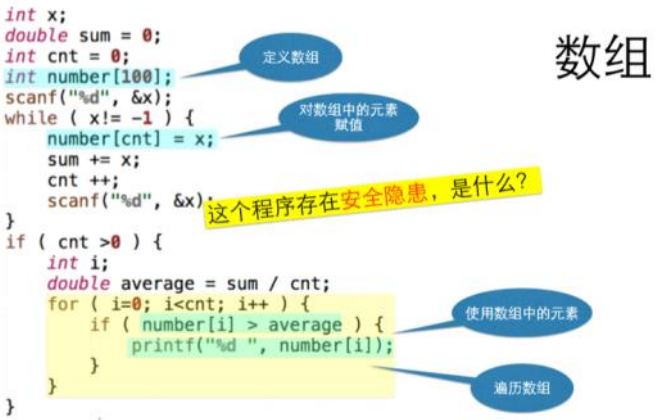
数组

2020年7月14日 19:53

数组

- int number[100];
- scanf("%d", &number[i]);

数组



定义数组

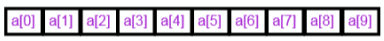
- <类型> 变量名称[元素数量];
  - int grades[100];
  - double weight[20];
- 元素数量必须是整数
- C99之前：元素数量必须是编译时刻确定的字面量

数组

- 是一种容器（放东西的东西），特点是：
  - 其中所有的元素具有相同的数据类型；
  - 一旦创建，不能改变大小
  - *（数组中的元素在内存中是连续依次排列的）

int a[10]

- 一个int的数组
- 10个单元：a[0],a[1],…,a[9]



- 每个单元就是一个int类型的变量
- 可以出现在赋值的左边或右边：
  - a[2] = a[1]+6;
- * 在赋值左边的叫做左值

数组的单元

- 数组的每个单元就是数组类型的一个变量
- 使用数组时放在[]中的数字叫做下标或索引，下标从0开始计数：
  - grades[0]
  - grades[99]
  - average[5]

有效的下标范围

- 编译器和运行环境都不会检查数组下标是否越界，无论是对数组单元做读还是写
- 一旦程序运行，越界的数组访问可能造成问题，导致程序崩溃
  - **segmentation fault**
- 但是也可能运气好，没造成严重的后果
- 所以这是程序员的责任来保证程序只使用有效的下标值：[0,数组的大小－1]

```
int x;
double sum = 0;
int cnt = 0;
int number[100];
scanf("%d", &x);
while ( x!= -1 ) {
    number[cnt] = x;
    sum += x;
    cnt ++;
    scanf("%d", &x);
}
if ( cnt >0 ) {
    int i;
    double average = sum / cnt;
    for ( i=0; i<cnt; i++ ) {
        if ( number[i] > average ) {
            printf("%d ", number[i]);
        }
    }
}
```

- 这个程序是危险的，因为输入的数据可能超过100个

```c
#include <stdio.h>

void f();

int main()
{
    f();
    printf("here\n");
    return 0;
}

void f()
{
    int a[10];
    a[10] = 0;
}
```

下标越界不会报错，只可能会warning，编译器会找一个地方放a[10]，printf不出here

```
/Users/wengkai/cc/test.c:14:2: note: 
        int a[10];
        ^
1 warning generated.
bash: line 1:  4344 Abort trap: 6
[Finished in 0.6s with exit code 134]
```

计算平均数

- 如果先让用户输入有多少数字要计算，可以用C99的新功能来实现

```c
int x;
double sum = 0;
int cnt;
printf("请输入数字的数量: ");
scanf("%d", &cnt);
if ( cnt>0 ) {
    int number[cnt];
    scanf("%d", &x);
    while ( x!= -1 ) {
        number[cnt] = x;
        sum += x;
        cnt ++;
        scanf("%d", &x);
    }
```

C99 ONLY!

长度为0的数组?

数组中不能放东西，数组长度为0。a[0]a即下标越界了。

- int a[0];
- 可以存在，但是无用

字符可以做下标吗?

数组的下标必须是整数，那么字符可以做下标吗?

比如:

1. `int a[255];`
2. `a['A'] = 1;`

这样的代码可行吗？为什么?

可以，‘A’表示的是ASCII中的数值。 char类型在C语言中就当做int类型对待

- 写一个程序，输入数量不确定的[0,9]范围内的整数，统计每一种数字出现的次数，输入-1表示结束

```c
const int number = 10;
int x;
int count[number];
int i;

for ( i=0; i<number; i++ ) {
    count[i] = 0;
}
scanf("%d", &x);
while ( x!= -1 ) {
    if ( x>=0 && x<=9 ) {
        count[x]++;
    }
    scanf("%d", &x);
}
for ( i=0; i<number; i++ ) {
    printf("%d:%d\n", i, count[i]);
}
```

数组的大小 C99

定义数组

初始化数组

数组参与运算

遍历数组输出

```c
/**
找出key在数组a中的位置
@param key 要寻找的数字
@param a 要寻找的数组
@param length 数组a的长度
@return 如果找到，返回其在a中的位置；如果找不到则返回-1
*/
int search(int key, int a[], int length);
```

```c
int search(int key, int a[], int length)
{
    int ret = -1;
    int i;
    for ( i=0; i< length; i++ ) {
        if ( a[i] == key ) {
            ret = i;
            break;
        }
    }
    return ret;
}
```

```c
int main(void)
{
    int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};
    int x;
    int loc;
    printf("请输入一个数字: ");
    scanf("%d", &x);
    loc=search(x, a, sizeof(a)/sizeof(a[0]));
    if ( loc != -1 ) {
        printf("%d在第%d个位置上\n", x, loc);
    } else {
        printf("%d不存在\n", x);
    }

    return 0;
}
```

```c
int main(void)
{
    int a[13] = {2};//{2,4,6,7,1,3,5,9,11,13,23,14,32};
    {
        int i;
        for ( i=0; i<13; i++ ) {
            printf("%d\t", a[i]);
        }
        printf("\n");
    }
```

数组的集成初始化

```
int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};
```

- 直接用大括号给出数组的所有元素的初始值
- 不需要给出数组的大小，编译器替你数数

```
int b[20] = {2};
```

- 如果给出了数组的大小，但是后面的初始值数量不足，则其后的元素被初始化为0

集成初始化时的定位

C99 ONLY!

```
int a[10] = {
    [0] = 2, [2] = 3, 6 ,
};
```

- 用[n]在初始化数据中给出定位
- 没有定位的数据接在前面的位置后面
- 其他位置的值补零
- 也可以不给出数组大小，让编译器算
- 特别适合初始数据稀疏的数组

```
23      // int x;
24      // int loc;
25      // printf("请输入一个数字: ");
26      // scanf("%d", &x);
27      // loc=search(x, a, sizeof(a)/sizeof(a[0]));
28      // if ( loc != -1 ) {
29      //  printf("%d在第%d个位置上\n", x, loc);
30      // } else {
31      //  printf("%d不存在\n", x);
32      // }
33
34      return 0;
2   0   0   0   0   0   0   0   0   0   0   0
[Finished in 0.1s]
```

```
12 int main(void)
13 {
14     int a[13] = {[1]=2,4,[5]=6};//{2,4,6,7,1,3,5,9,11
15     {
16         int i;
17         for ( i=0; i<13; i++ ) {
18             printf("%d\t", a[i]);
19         }
20         printf("\n");
21     }
22
23     // int x;
24     // int loc;
25     // printf("请输入一个数字: ");
26     // scanf("%d", &x);
27     // loc=search(x, a, sizeof(a)/sizeof(a[0]));
28     // if ( loc != -1 ) {
29     //  printf("%d在第%d个位置上\n", x, loc);
30     // } else {
31     //  printf("%d不存在\n", x);
32     // }
33
34     return 0;
35 }
0   2   4   0   0   6   0   0   0   0   0   0   0
[Finished in 0.2s]
```

```
12 int main(void)
13 {
14     int a[] = {[1]=2,4,[5]=6};//{2,4,6,7,1,3,5,9,11,13,23,14,32};
15     {
16         int i;
17         printf("%lu\n", sizeof(a));
18         printf("%lu\n", sizeof(a[0]));
19         for ( i=0; i<13; i++ ) {
20             printf("%d\t", a[i]);
21         }
22         printf("\n");
23     }
24
25     // int x;
26     // int loc;
27     // printf("请输入一个数字: ");
28     // scanf("%d", &x);
29     // loc=search(x, a, sizeof(a)/sizeof(a[0]));
30     // if ( loc != -1 ) {
31     //  printf("%d在第%d个位置上\n", x, loc);
32     // } else {
33     //  printf("%d不存在\n", x);
34     // }
24
4
0   2   4   0   0   6   0   0   1559063872  0   -557699994   1392483769 1559063872
[Finished in 0.2s]
```

```
12 int main(void)
13 {
14     int a[] = {[1]=2,4,[5]=6};//{2,4,6,7,1,3,5,9,11,13,23,14,32};
15     {
16         int i;
17         for ( i=0; i<6; i++ ) {
18             printf("%d\t", a[i]);
19         }
20         printf("\n");
21     }
22
23     // int x;
24     // int loc;
25     // printf("请输入一个数字: ");
26     // scanf("%d", &x);
27     // loc=search(x, a, sizeof(a)/sizeof(a[0]));
28     // if ( loc != -1 ) {
29     //  printf("%d在第%d个位置上\n", x, loc);
30     // } else {
31     //  printf("%d不存在\n", x);
32     // }
33
34     return 0;
35 }
0   2   4   0   0
[Finished in 0.2s]
```

```
const int number = 10;
int x;
int count[number] = {0};
int i;

// for ( i=0; i<number; i++ ) {
//  count[i] = 0;
// }
scanf("%d", &x);
while ( x!= -1 ) {
    if ( x>=0 && x<=9 ) {
        count[x]++;
    }
    scanf("%d", &x);
}
for ( i=0; i<number; i++ ) {
    printf("%d:%d\n", i, count[i]);
}

return 0;
```

简化初始化!

数组的大小

- sizeof给出整个数组所占据的内容的大小，单位是字节

```
sizeof(a)/sizeof(a[0])
```

可扩展性!

- sizeof(a[0])给出数组一个单个元素的大小，于是相除就得到了数组的单元个数
- 这样的代码，一旦修改数组中初始的数据，不需要修改遍历的代码

数组的赋值

```
int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};
int b[] = a;
```

- 数组变量本身不能被赋值
- 要把一个数组的所有元素交给另一个数组，必须采用遍历

```
for ( i=0; i<length; i++ ) {
    b[i] = a[i];
}
```

遍历数组

```
for ( i=0; i<length; i++ ) {
    b[i] = a[i];
}
```

```
for ( i=0; i<number; i++ ) {
    count[i] = 0;
}
```

```
for ( i=0; i< length; i++ ) {
    if ( a[i] == key ) {
        ret = i;
        break;
    }
}
```

```
for ( i=0; i<cnt; i++ ) {
    if ( number[i] > average ) {
        printf("%d ", number[i]);
    }
}
```

```
for ( i=0; i<number; i++ ) {
    printf("%d:%d\n", i, count[i]);
}
```

- 通常都是使用for循环，让循环变量i从0到<数组的长度，这样循环体内最大的i正好是数组最大的有效下标
- 常见错误是：
  - 循环结束条件是<=数组长度，或；
  - 离开循环后，继续用i的值来做数组元素的下标！

离开时访问的话正好访问数组的第一个无效下标!

```
/**
找出key在数组a中的位置
@param key 要寻找的数字
@param a 要寻找的数组
@param length 数组a的长度
@return 如果找到，返回其在a中的位置；如果找不到则返回-1
*/
int search(int key, int a[], int length);
```

```
int search(int key, int a[], int length)
{
    int ret = -1;
    int i;
    for ( i=0; i< length; i++ ) {
        if ( a[i] == key ) {
            ret = i;
            break;
        }
    }
    return ret;
}
```

```
int main(void)
{
    int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};
    int x;
    int loc;
    printf("请输入一个数字：");
    scanf("%d", &x);
    loc=search(x, a, sizeof(a[0]));
    if ( loc != -1 ) {
        printf("%d在第%d个位置上\n", x, loc);
    } else {
        printf("%d不存在\n", x);
    }

    return 0;
}
```

数组作为函数参数时，往往必须再用另一个参数来传入数组的大小

- 数组作为函数的参数时：
  - 不能在[]中给出数组的大小
  - 不能再利用sizeof来计算数组的元素个数！

从2到x-1测试是否可以整除

```
int isPrime(int x)
{
    int ret = 1;
    int i;
    if ( x == 1 ) ret = 0;
    for ( i=2; i<x; i++ ) {
        if ( x % i == 0 ) {
            ret = 0;
            break;
        }
    }
    return ret;
}
```

- 对于n要循环n-1遍
- 当n很大时就是n遍

去掉偶数后，从3到x-1，每次加2

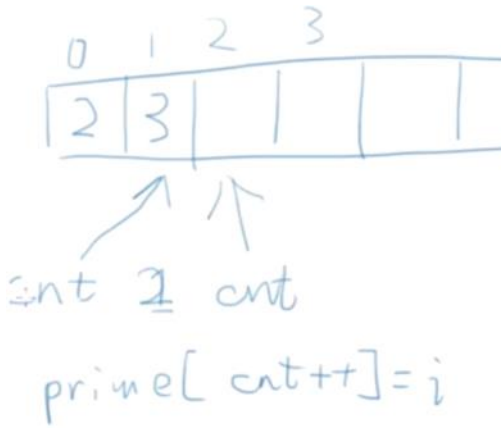
```
int isPrime(int x)
{
    int ret = 1;
    int i;
    if ( x == 1 ||
         (x%2 == 0 && x!=2) )
        ret = 0;
    for ( i=3; i<x; i+=2 ) {
        if ( x % i == 0 ) {
            ret = 0;
            break;
        }
    }
    return ret;
}
```

- 如果x是偶数，立刻
- 否则要循环(n-3)/2+1遍
  - 当n很大时就是n/2遍

无须到x-1，到sqrt(x)就够了

```
int isPrime(int x)
{
    int ret = 1;
    int i;
    if ( x == 1 ||
         (x%2 == 0 && x!=2) )
        ret = 0;
    for ( i=3; i<sqrt(x); i+=2 ) {
        if ( x % i == 0 ) {
            ret = 0;
            break;
        }
    }
    return ret;
}
```

- 只需要循环sqrt(x)遍



判断是否能被已知的且<x的素数整除

写进去，cnt再加一。套路！！

```
int main(void)
{
    const int number = 100;
    int prime[number] = {2};
    int count = 1;
    int i = 3;
    while ( count < number ) {
        if ( isPrime(i, prime, count) ) {
            prime[count++] = i;
        }
        i++;
    }
    for ( i=0; i<number; i++ ) {
        printf("%d", prime[i]);
        if ( (i+1)%5 ) printf("\t");
        else printf("\n");
    }
    return 0;
}
```

```
int isPrime(int x, int knownPrimes[], int numberOfKnownPrimes)
{
    int ret = 1;
    int i;
    for ( i=0; i<numberOfKnownPrimes; i++ ) {
        if ( x % knownPrimes[i] ==0 ) {
            ret = 0;
            break;
        }
    }
    return ret;
}
```

```
const int number = 10;
int prime[number] = {2};
int count = 1;
int i = 3;

while ( count < number ) {
    if ( isPrime(i, prime, count) ) {
        prime[count++] = i;
    }
    {
        printf("i=%d \tcnt=%d\t",i, count);
        int i;
        for ( i=0; i<number; i++ ) {
            printf("%d\t", prime[i]);
        }
        printf("\n");
    }
    i++;
}
for ( i=0; i<number; i++ ) {
    printf("%d", prime[i]);
    if ( (i+1)%5 ) printf("\t");
    else printf("\n");
}
```

大括号作用：里面可以有自己的变量i，不影响外面的变量

构造素数表

- 欲构造n以内的素数表

1. 令x为2
2. 将2x、3x、4x直至ax<n的数标记为非素数
3. 令x为下一个没有被标记为非素数的数，重复2；直到所有的数都已经尝试完毕

构造素数表

- 欲构造n以内（不含）的素数表

1. 开辟prime[n]，初始化其所有元素为1，prime[x]为1表示x是素数
2. 令x=2
3. 如果x是素数，则对于(i=2;x*i<n;i++)令prime[i*x]=0
4. 令x++，如果x<n，重复3，否则结束

构造素数表

```
const int maxNumber = 25;
int isPrime[maxNumber];
int i;
int x;
for ( i=0; i<maxNumber; i++ ) {
    isPrime[i] = 1;
}
for ( x=2; x<maxNumber; x++ ) {
    if ( isPrime[x] ) {
        for ( i=2; i*x<maxNumber; i++ ) {
            isPrime[i*x] = 0;
        }
    }
}
for ( i=2; i<maxNumber; i++ ) {
    if ( isPrime[i] ) {
        printf("%d\t", i);
    }
}
printf("\n");
```

- 算法不一定和人的思考方式相同

二维数组

- int a[3][5];
- 通常理解为a是一个3行5列的矩阵

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

二维数组的遍历

```
for ( i=0; i<3; i++ ) {
    for ( j=0; j<5; j++ ) {
        a[i][j] = i*j;
    }
}
```

- a[i][j]是一个int
- 表示第i行第j列上的单元
  - a[i,j]是什么？ i,j是逗号表达式,a[i,j]即a[j]

也可以里面不用括号，像一维数组那样写，因为二维数组在内存中和一维数组一样排列。但是写了两重括号增加了可读性。

二维数组的初始化

```
int a[][5] = {
    {0,1,2,3,4},
    {2,3,4,5,6},
};
```

- 列数是必须给出的，行数可以由编译器来数
- 每行一个{}，逗号分隔
- 最后的逗号可以存在，有古老的传统
- 如果省略，表示补零
- 也可以用定位（* C99 ONLY）

tic-tac-toe游戏

- 读入一个3X3的矩阵，矩阵中的数字为1表示该位置上有一个X，为0表示为O
- 程序判断这个矩阵中是否有获胜的一方，输出表示获胜一方的字符X或O，或输出无人获胜

本页图片来源：wikipedia.org

读入矩阵

```
const int size = 3;
int board[size][size];
int i,j;
int numOfX;
int numOfO;
int result = -1;    //  -1:没人赢，1:X赢，0:O赢

//  读入矩阵
for ( i=0; i<size; i++ ) {
    for ( j=0; j<size; j++ ) {
        scanf("%d", &board[i][j]);
    }
}
```

行和列?

```
//  检查行
for ( i=0; i<size && result == -1; i++ ) {
    numOfO = numOfX = 0;
    for ( j=0; j<size; j++ ) {
        if ( board[i][j] == 1 ) {
            numOfX ++;
        } else {
            numOfO ++;
        }
    }
    if ( numOfO == size ) {
        result = 0;
    } else if (numOfX == size ) {
        result = 1;
    }
}
```

```
if ( result == -1 ) {
    for ( j=0; j<size && result == -1; j++ ) {
        numOfO = numOfX = 0;
        for ( i=0; i<size; i++ ) {
            if ( board[i][j] == 1 ) {
                numOfX ++;
            } else {
                numOfO ++;
            }
        }
        if ( numOfO == size ) {
            result = 0;
        } else if (numOfX == size ) {
            result = 1;
        }
    }
}
```

能否用一个两重循环来检查行和列?

检查对角线

```
numOf0 = numOfX = 0;
for ( i=0; i<size; i++ ) {
    if ( board[i][i] == 1 ) {
        numOfX ++;
    } else {
        numOf0 ++;
    }
}
if ( numOf0 == size ) {
    result = 0;
} else if (numOfX == size ) {
    result = 1;
```

```
numOf0 = numOfX = 0;
for ( i=0; i<size; i++ ) {
    if ( board[i][size-i-1] == 1 ) {
        numOfX ++;
    } else {
        numOf0 ++;
    }
}
```