# 数组

## 定义

元素类型相同，大小相等。

**数组元素访问时间为什么是常数时间？**

访问数组元素的时候，需要计算出该元素距离数组基址的偏移量，而为了计算偏移量，必须先做乘法，然后把乘法所得的结果，加到基址上面，已确定出该元素的内存地址。

整个过程涉及一次乘法与一次加法。由于两项操作都只需要花费常数级别的时间，因此访问数组元素，只需要常数级别时间。

## 分类

在数组定义中，可以将元素数量指定为一个常量表达式，或者在特定情况下，指定为涉及变量的表达式。采用这两种方式定义的数组分别被称为固定长度数组（fixed-length）和长度可变（variable-length）数组。

### 固定长度数组

固定长度数组可以具有任意存储类别：可以将它们定义在所有函数的外面或语句块的里面，并且可以使用或不使用存储类修饰符 static。唯一的限制是数组不能作为函数参数。一个传入函数的数组参数需要被转换为指向数组第一个元素的指针。

下面 4 种数组定义方式都是合法的：

int a[10]; // a有外部链接

static int b[10]; // b有静态存储周期和文件作用域

void func()

{

static int c[10]; // c有静态存储周期和块作用域

int d[10]; // d有动态存储周期

/\* ... \*/

}

### 长度可变数组/动态数组

动态数组能够改变长度的数据结构。

有一种简单的实现方式，是先分配固定的数组空间，等到数组填满时，再多分配一倍的空间。同理，如果数组现存的元素还不到总量的1/2，那就把数组大小减半。

如果一个数组具有动态存储周期（也就是说，如果在语句块内定义数组，并且没有 static 修饰符），那么 C99 也允许把非常量表达式作为元素数量来定义该数组。这样的数组被称为长度可变数组（variable-length array）/动态数组。

而且，长度可变数组的名称必须是普通的标识符。长度可变数组不能作为结构或联合的成员。在下面的示例中，只有 vla 数组的定义是合法的：

void func( int n )

{

int vla[2\*n]; // 合法：存储周期为动态的

static int e[n]; // 非法：长度可变数组不可有静态存储周期

struct S { int f[n]; }; // 非法：f不是一个普通标识符

/\* ... \*/

}

与其他动态变量一样，每次程序流进入包含长度可变数组定义的语句块时，都会重新创建这个长度可变数组。因此，在每次实例化时，数组都可以有不同的长度。然而，一旦被创建，即便是长度可变数组，在它的当前存储周期内也不能改变数组长度。

动态对象被存储在栈中，当程序流离开对象所在的语句块时，动态对象的空间就会被释放。因此，只有对小的、临时的数组，定义长度可变数组才比较合理。如想动态地创建大型数组，通常应该使用标准函数 malloc和 calloc来显式地分配内存空间。

这种数组的存储周期会持续到程序结束，也可以调用函数 free（）来主动地释放被占用的内存空间。

例如：

#include <stdio.h>

#include <stdlib.h>

int main()

{

int n1,i;

int \*array;

printf("请输入所要创建的一维动态数组的长度：");

scanf("%d",&n1);

array=(int\*)calloc(n1,sizeof(int));

for(i=0;i<n1;i++)

{

printf("%d\t",array[i]);

}

printf("\n");

for(i=0;i<n1;i++)

{

array[i]=i+1;

printf("%d\t",array[i]);

}

free(array);//释放第一维指针

return 0;

## 初始化

## 特点

### 优点

1. 简单易用；
2. 可以迅速访问其元素（只需耗费常数级别的时间）

### 缺点

1. 大小固定；
2. 必须一次分配一整块内存；
3. 按照位置来插入元素是一种较为复杂的操作。

# 应用

## 数组中的查找

题目要求：一个整数数组中有一个元素出现了一次，其他元素都出现了两次，使用最小的时间复杂度找出出现一次的数。

代码：

#include <iostream>

#include <vector>

using namespace std;

/\*

一个数组中只有一个数字出现了一次，其他都是出现了两次

找到这个出现一次的数字

\*/

//将整个数组中的整数进行异或 最后得到的结果就是所要找的数

//两个相等的数异或操作得到的是0

//0和另外一个不等的数异或操作得到的是另外一个数

//将数组中所有整数异或操作，得到的就是想要找的数

int SingleNumber(vector<int>& vec)

{

if(vec.size()<=0)

return 0;

int i;

int value = vec[0];

for(i=1;i<vec.size();i++)

value ^= vec[i];

return value;

}

/\*

如果一个数组中有两个数出现了一次，其他都是出现了两次，这两个数分别是什么

\*/

void SingleNumberII(vector<int>& a,int& pN1,int& pN2)

//入参：待输入参数a，2个输出参数pN1,pN2

{

int i,j,temp;

//计算这两个数的异或结果（计算两个出现1次的数异或操作）

temp =0;

for(i=0;i<a.size();i++)

temp ^= a[i];

//找到异或结果中第一个为1的位

//（依据这个位将大数组分割成2个，这样就与前面的情况一致了）

for(j=0;j<sizeof(int)\*8;j++)

if(((temp >> j)&1) ==1)

break;

//第j位为1，说明这两个数字在第j位上不同，根据这个来进行分组

pN1 =0;

pN2 =0;

for(i=0;i<a.size();i++)

if(((a[i] >> j )&1) ==0)

pN1 ^= a[i];

else

pN2 ^= a[i];

}

/\*

一个数组中只有一个数出现了一次，其他都是出现了三次，找出这个数

\*/

//同样根据位来判断

int SingleNumberIII(vector<int>& vec)

{

int i,j;

int low=0,high=vec.size()-1;

int bit =1;

int flag =0;

//通过某一位的值将数组分为两部分，其中一部分包含了待找的数，另一

部分不包含

while(low <= high)

{

if(low == high)

return vec[low];

i = low-1;

for(j = low;j<= high;j++)

{

if((vec[j]&bit) == 0)

{

i++;

swap(vec[i],vec[j]);

}

}

if(i >= low)

{

if((i-low+1)%3 == 0)

low = i+1;

else

high =i;

}

bit = bit<<1;

}

return 0;

}

int main()

{

int array[]={12,23,12,45,56,45,23,78,78,78,12,23,45};

vector<int> vec(array,array+sizeof(array)/sizeof(int));

//上述是用C语言定义数组，然后拷贝到C++的vector中

int fir,sec;

cout<<SingleNumberIII(vec);

// cout<<fir<<endl<<sec<<endl;

return 0;

}

拓展：

如果有两个数均出现了一次，其他都出现了两次，如何查找这两个数？

如果一个数组中有一个数出现了一次，其他数都出现了三次，如何找到出现一次的数？

## 设置二维数组中的元素值

题目要求：在一个二维数组中，如果某一个元素的值为0，那么就将该元素所在的行与列中的所有元素都赋值为0，你能达到的最小空间复杂度是多少？

代码：

#include <iostream>

#include <vector>

using namespace std;

/\*

在一个二维数组中，如果某一个元素的值为0 将其所在的行和列所有元素都置为0，尽可能小的空间复杂度

使用某一行列存储即可，不需要存储全部行列，其他行列根据第1行1列设置

\*/

void SetZero(vector<vector<int> >& matrix)

{

bool firstLine = false;

bool firstRow = false;

//判断第一行是否有值为0 的元素

for(int i=0;i<matrix[0].size();i++)

{

if(matrix[0][i] == 0)

{

firstLine = true;

break;

}

}

//判断第一列是否有0的元素

for(int i=0;i<matrix.size();i++)

{

if(matrix[i][0] == 0)

{

firstRow = true;

break;

}

}

//先处理第一行和第一列

for(int i=1;i<matrix.size();i++)

{

for(int j=1;j<matrix[0].size();j++)

{

if(matrix[i][j] == 0)

{

matrix[i][0] =0;

matrix[0][j] =0;

}

}

}

//根据第一行和第一列的元素标记，更新其他元素

for(int i=1;i<matrix.size();i++)

{

for(int j=1;j<matrix[0].size();j++)

{

if(matrix[i][0]==0 || matrix[0][j] ==0)

{

matrix[i][j] =0;

}

}

}

//更新第一行

if(firstLine)

{

for(int i=0;i<matrix[0].size();i++)

matrix[0][i] =0;

}

//更新第一列

if(firstRow)

{

for(int i=0;i<matrix.size();i++)

matrix[i][0] = 0;

}

}

## 第K大问题

题目要求：在一个整数数组中，查找该数组中第K大的元素，时间复杂度能达到O(n)吗？

代码：

#include <iostream>

#include <stdlib.h>

#include <algorithm>

#include <vector>

using namespace std;

//查找一个数组中的第K大的元素

int Find\_K(int a[],int low,int high,int k)

{

if(k<=0)

return -1;

if(k>high-low+1)

return -1;

//随机选择一个支点

int pivot = low+rand()%(high-low+1);

swap(a[low],a[pivot]);

int m = low;

int count = 1;

//每遍历一次，把较大的放到数组的左边

for(int i=low+1;i<=high;i++)

{

if(a[i] > a[low])

{

swap(a[++m],a[i]);

count++; //比关键点大的数的个数为count+1

}

}

swap(a[m],a[low]);// 将关键点放在左右两部分的分界点

if(count > k)

return Find\_K(a,low,m-1,k);

else if(count < k)

return Find\_K(a,m+1,high,k-count);

else

return m;

}

//查找最小的K个数

void FindMinTopK(vector<int>& vec,int k)

{

vector<int> heap(vec.begin(),vec.begin()+k);

make\_heap(heap.begin(),heap.end());

int i;

//开始处理剩余的数据

for(i =k;i<vec.size();i++)

{

if(vec[i]<heap[0])

{

pop\_heap(heap.begin(),heap.end());

heap.pop\_back();

heap.push\_back(vec[i]);

push\_heap(heap.begin(),heap.end());

}

}

for(i=0;i<heap.size();i++)

cout<<heap[i]<<endl;

}

int main()

{

int a[]={5,15,4,8,2,3,9,10};

int r = Find\_K(a,0,sizeof(a)/sizeof(int)-1,3);

// cout<<(r==-1? r : a[r])<<endl;

int array[]={5,3,6,7,4,2,1,9,8,10};

vector<int> vec(array,array+sizeof(array)/sizeof(int));

FindMinTopK(vec,4);

return 0;

}

拓展：

top K问题！

## 数组中的逆序对

题目要求：一个数组中，如果一个元素的值比它右边的值大，那么这一对元素构成逆序对，求一个乱序数组中逆序对的数目。

代码：

#include <iostream>

#include <vector>

#include <string>

using namespace std;

unsigned int CountInversions(vector<unsigned int>& a);

unsigned int MergeSortCount(vector<unsigned int>& a,int left,

int right,vector<unsigned int>& temp);

unsigned int MergeCount(vector<unsigned int>& a,int left,int mid,

int right,vector<unsigned int>& temp);

int main()

{

int array[]={4,2,10,3,5,7};

vector<unsigned int> a(array,array+sizeof(array)/sizeof(int));

unsigned int result = CountInversions(a);

cout <<result<<endl;

return 0;

}

unsigned int MergeCount(vector<unsigned int>& a,int left,int mid,

int right,vector<unsigned int>& temp)

{

int i = left;

int j= mid+1;

int k = left;

int count = 0;

//开始查找逆序对的个数

while( i<= mid && j<=right)

{

if(a[i]<= a[j])

temp[k++] = a[i++];

else

{

temp[k++]=a[j++];

count+= mid-i+1;

}

}

while(i<=mid) temp[k++]= a[i++];

while(j<= right) temp[k++] = a[j++];

for(i=left;i<=right;i++)

a[i]= temp[i];

return count;

}

unsigned int MergeSortCount(vector<unsigned int>& a,int left,

int right,vector<unsigned int>& temp)

{

if(left >= right) return 0;

int mid = (left+right)/2;

unsigned int Inverleft = MergeSortCount(a,left,mid,temp);

unsigned int Inverright = MergeSortCount(a,mid+1,right,temp);

unsigned int InverSum = MergeCount(a,left,mid,right,temp);

return Inverleft+Inverright+InverSum;

}

unsigned int CountInversions(vector<unsigned int>& a)

{

int n= a.size();

vector<unsigned int> temp(a.begin(),a.end());

unsigned int ans = MergeSortCount(a,0,n-1,temp);

return ans;

}