
Welcome to C++
复习：数组、动态分配、vector

1. C++数组

- 定义和初始化

```
int   a[1000];
const   int N=1000;
int    b[N];
```

- 访问

```
for (int i=0; i<N; ++i )
      if( a[i] > 0 ) s += a[i];
```

- 插入

```
for ( int i=n; i>pos; --i )
      a[i] = a[i-1] ;
a[pos] = x;
```

- 删除

```
for (int  i=pos; i<n; ++i)
       a[i] = a[i+1];
```

怎样得到最大连续子串和?

1 2 -4 3 6 -8 7 -1 1 2 -8 7

1 2 -4 3 6 -8 7 -1 1 2 -8 7 ?

1 2 -4 3 6 -8 7 -1 1 2 -8 7

1 2 -4 3 6 -8 7 -1 1 2 -8 7

1 2 -4 3 6 -8 7 -1 1 2 -8 7 ?

1 2 -4 3 6 -8 7 -1 1 2 -8 7 ?

1 2 -4 3 6 -8 7 -1 1 2 -8 7

1 3 -1 3 9 1 8 7 8 10 2 9

应用举例：最大连续子序列和

```
#include <iostream>
using namespace std;
int main() {
    int n,nums[100],maxSum[100];
    cin >> n;
    for(int i=0; i<n; ++i) cin >> nums[i];

    int result = nums[0];
    maxSum[0] = nums[0];
    for(int i=1; i<n; ++i){
        if(maxSum[i-1] > 0) maxSum[i] = nums[i]+maxSum[i-1];
        else maxSum[i] = nums[i];
        if(maxSum[i] > result) result = maxSum[i];

    }

    cout << result << endl;

}
```

- 数组排序
- 选择排序、快速排序、希尔排序、堆排序（不稳定的排序算法）
- 冒泡排序、插入排序、归并排序和基数排序（稳定的排序算法）

a[0]	a[1]	a[2]	a[3]	a[4]
6	2	4	**6**	1

1	2	4	6	6
原a[4]	原a[1]	原a[2]	原a[0]	原a[3]
原a[4]	原a[1]	原a[2]	原a[3]	原a[0]

- 扩展：STL（标准模板库）排序函数

sort 对给定区间所有元素进行排序

stable_sort 对给定区间所有元素进行稳定排序

……

- sort函数简单用法

```
#include <algorithm>
```

Sort函数有三个参数：（1）排序的数组的起始地址。（2）结束的地址（最后一位要排序的地址的下一地址）（3）排序的方法，可以是从大到小也可是从小到大，还可以不写第三个 参数，此时默认的排序方法是从小到大排序。

```
……
int num[100];
……
sort(num,num+100);
```

- 自定义比较函数

```
#include <algorithm>
……
bool cmp(int d1, int d2){
     return d1>d2;
}

……
int num[100];
……
sort(num,num+100,cmp);
```

- 扩展：逆转函数 reverse

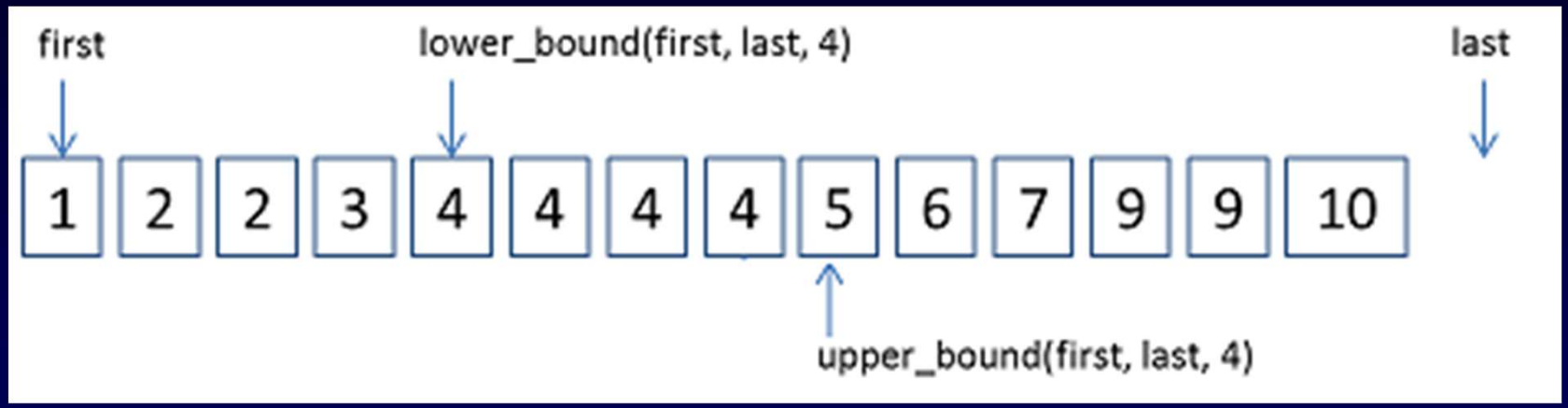
```
#include <algorithm>
……
int num[100];
……
for(int i=0; i<n; ++i)
        cin >> num[i];
reverse(num,num+n);
```

- 扩展：查找函数find

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
    int num[100];
    int x;
    for(int i=0; i<5; ++i)
          cin >> num[i];
     cin >> x;
     int *p = find(num, num+5, x);
     cout << p-num << "  " << *p << endl;
}
```

如果找不到?

- 扩展：查找函数upper_bound, lower_bound



- 算法基于二分查找
- 其中**first**指需查找线性表（已按升序排列）的起始位置，**last**指结束位置（前闭后开）

比如需在a数组(存放了n个有效数据)搜索x这个数：

- `upper_bound(a,a+n,x)`

//返回最后一个<=x的元素的下一个地址值, 或者a（所有元素都大于x）

upper_bound(begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

- `lower_bound(a,a+n,x)`

//返回第一个>=x的元素的地址值, 或者a+n（所有元素都小于x）

lower_bound(begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于或等于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

-

例：输入n(n<1000000),并输入n个数据，存放在数组中。接着输入若干个数，查找每个数出现的次数。

- 数组定义：　　外部or内部？
- 排序：
- 利用upper_bound和lower_bound查找

扩展练习：1339 分类搜索 POI

Sample Input:

```
5 3
1
3
4
1
1
3
-6
1
```

5个数据
3次查找

Sample Output:

```
1
0
3
```

- C数组的问题：

(1) 数组没有越界检查，这意味着，程序员可以访问数组最后一个元素以后的地址，或者第一个元素之前的地址（例如，a[-1]、a[-2]这种形式是合法的）。

(2) 数组大小确定，不支持自动增长。

2. 动态分配实现动态数组

- 操作符 new

```
int   *p1,*p2;
…….
p1 = new int[n];
p2 = new int;
```

- 操作符 delete

```
delete[]  p1;
delete  p2;
```

```
int   *p = new int[10];
```
// 分配10个元素的动态数组，p指向该数组第一个对象
//方括号中的数据必须是整形，但不必是常量

```
    ……
    p[0]=1;
    p[1]=5;
    ……
```

例：输入n(n<1000000),并输入n个整数，存放在数组中。接着输入若干个整数，查找每个数出现的次数。

```
int n;
cin >> n;
int *P = new int[n];
……
```

- 释放动态数组空间

```
delete[ ]  p;
```

销毁p指向的数组中的元素，并释放对应的内存。

- 例：给定一个整型数组，在数组中找出两个数使这两个数的和为给定数，从小到大输出这两个数在数组中的位置

注意：Heap里的内容如果不主动释放，则一直占用。

```
#include <iostream>
using namespace std;
int* twoSum(int* nums, int numsSize, int target) {
    int *p=(int *)malloc(2*sizeof(int)); //int *p = new int[2];
    for(int i=0; i<numsSize-1; ++i){
        int a = target-nums[i];
        for(int j=i+1; j<numsSize; ++j)
            if(nums[j] == a){
                p[0]=i; p[1]=j;
                return p;       }
    }
    p[0]=p[1]=-1;  return p;
}//-----------------------------------------
int main(){
    int a[100];
    int n;
    cin >> n;
    for(int i=0; i<n; ++i) cin >> a[i];
    int *p = twoSum(a,n,10);
    cout << p[0] << " " << p[1] <<endl;
    free(p); //delete[] p;
}
```

- 连续空间的初始化

  - **memset**

功 能：将s所指向的某一块内存中的每个字节的内容全部设置为ch指定的ASCII值。通常为新申请的内存做初始化工作

```
void *memset(void *s,int c,size_t n)
```

作用：将s为首地址的 n 个字节的内存空间设为值 c（给空间初始化）；

- 连续空间的拷贝

  - memcpy

```
void *memcpy(void *dest, const void
  *src, size_t n);
```

用法：用来将src为首地址的空间的内容拷贝n个字节的数据至目标地址dest指向的内存中去。函数返回指向dest的指针。

- C语言需要包含头文件string.h；C++需要包含cstring 或 string.h。

3. 向量（vector)

- STL的序列容器
- 向量与数组的共同特征是元素的排列在逻辑上是线性序列结构，可以用下标进行访问

```
#include <vector>
……
vector<int> v(n);
……
v[1] = 0;
```

- 向量的特点

(1) 向量可以按需创建，拷贝创建，局部拷贝创建，异类拷贝和创建

(2) 灵活的初始化

(3) 随意扩容和元素增减

(4) 可通过异常来进行下标溢出追踪和处理

(5) 可比较

• 向量的定义

- **int** n=10;
  int t[5]={1,2,3,4,5};
  vector<**int**> a(n);　　//按需创建
- vector<**int**> b(10, 1);　　//元素赋全 1
- vector<**int**> c(b);　　// 整体拷贝创建
- vector<int> f(t, t+5);　　//异类拷贝创建
- vector<**int**> d(b.begin(), b.begin()+3);
  //局部拷贝创建d为b的前 3 个元素
- a.assign(100,0);
  //动态扩容至100个元素, 每个元素为0;

小练习：下列每个vector对象中元素个数是多少？各元素的值是什么？

```
(a) vector<int> ivec1;
(b) vector<int> ivec2(10);
(c) vector<int> ivec3(10,42);
(d) vector<string> svec1;
(e) vector<string> svec2(10);
(f) vector<string> svec3(10,"hello");
```

【解答】

(a) 空向量，元素个数为0.
(b) 元素个数为10，各元素的值均为0。
(c) 元素个数为10，各元素的值均为42。
(d) 元素个数为0。
(e) 元素个数为10，各元素的值均为空字符串。
(f) 元素个数为10，各元素的值均为"hello"。

- 向量常用操作

- a.assign(b.begin(), b.begin()+3);
  // b的前3个元素赋给a
- a.assign(4,2);
  // a向量含4个元素，全初始化为2
- **int** x = a.back();
  // a的最后一个元素赋给变量x
- a.clear();
  // a向量清空（不再有元素）
- **if**(a.empty()) cout<<"empty";
  // a判空操作
- **int** y = a.front();
  // a的第一个元素赋给变量y

- a.pop_back();

// 删除a的最后一个元素

- a.push_back(5);

// a最后插入一个元素，其值为5

- a.resize(10);

// a元素个数调至10。多删少补，其值随机

- a.resize(10,2);

//a元素个数调至10。多删少补，新添元素初值为2

- **if**(a==b) cout << "equal";

// a与b的向量比较操作

- 用数组方式访问vector元素:

```
#include <vector>
#include <iostream>
using namespace std;
int main( ){
      vector<int> v;
      v.push_back(1);
      v.push_back(2);
      v.push_back(3);
      for(int i = 0; i < v.size(); i++)
          cout << v[i] << "\n";
      return 0;
}
```

注意：适当减少动态操作

比较以下两种写法：

……

```
vector<int> a(m);
for(int i=0; i<m; i++) cin>>a[i];
```

```
vector<int> a;
for(int i=0; i<m; i++) {
        cin>>x;
        a.push_back(x);
}
```

- 插入操作：

```
#include <vector>
#include <iostream>
using namespace std;
int main(){
    vector<int> v;
    v.push_back(22);
    v.push_back(33);
    v.push_back(55);
    v.insert(v.begin(), 11);
    v.insert(v.begin() + 3, 44);
    for(int i = 0; i < v.size(); i++)
       cout << v[i] << endl;
     return 0;
}
```

- 删除操作:

```
……
int main(){
    vector<int> v(5,7);
    v[0] = 111;
    v.push_back(11);
    v.push_back(22);
    v.push_back(33);
    v.erase(v.begin()+1);
    v.erase(v.end()-2);
    for (int i=0; i<v.size();++i)
          cout << v[i] << " ";
     cout <<endl;
    return 0;
}
```

- 查找操作

```
for(int i=0; i<v1.size(); ++i)
   if(v1[i]==x) v1.erase(v1.begin()+i);
……
```

- vector的排序

```
#include <algorithm>
……
vector <int>  v;
……
sort( v.begin(), v.end());
```

- 迭代器（iterator）的使用*

```
vector<int>::iterator it;
for(it = v .begin(); it != v .end(); it++)
     cout << *it << endl;
```

例：输入n(n<1000000),并输入n个整数，存放在数组中。接着输入若干个整数，查找每个数出现的次数。

```
int n;
cin >> n;

……
```

- 扩展练习：12! 配对

- **Description:**

找出输入数据中所有两两相乘的积为12!的个数。

- **Input:**输入数据中含有一些整数n（1≤n＜2^32）
- **Output:**输出所有两两相乘的积为12!的个数。
- **Sample Input:**

1 10000 159667200 9696 38373635 1000000 479001600 3

- **Sample Output:**

2

更多语法细节参考

http://www.cplusplus.com/