11.1-一维数组

代码示例1：输入5个整数，范围在[1,1000]，输出这5个整数里面第二大的整数。（对于相同的数字，先输入的大）

int a,b,c,d,e,max,second; //max为当前所有数字中的最大值，second为第二大的值

cin>>a>>b>>c>>d>>e;

if(a>=b)

{

max=a;

second=b;

}

else

{

max=b;

second=a;

}

if(c>max)

{

//如果c比当前最大值大

second=max; //当前最大值变为第二大值

max=c; //c为当前最大值

}

else

{

if(c<=max && c>second)

{

//c比最大值小，但比第二大值大

second=c; //c为当前第二大值

}

//c如果比当前第二大值还小，则无需更新

}

if(d>max)

{

second=max;

max=d;

}

else

{

if(d<=max && d>second)

{

second=d;

}

}

if(e>max)

{

second=max;

max=e;

}

else

{

if(e<=max && e>second)

{

second=e;

}

}

cout<<second;

问题：如果改为“在100个整数中，求第二大的数”，怎么办？

int a,b,c,d,e,f,g.........

int num[100]; //包含100个整型变量的数组

数组：

数组是**同类型元素**的有序组合体。元素的类型可以是C++语言中允许使用的任何一种数据类型

int num[100]; //一个装有100个int型变量的储物柜

**一维数组：**

顾名思义，只有一个下标，是一个同类型数据的“线性”集合

**声明：**

数据类型 数组名[元素个数] （={元素初值}）

int a[5]; //编译器分配5个int型变量的存储空间

**访问：**

数组中的每个元素都有与其**对应的下标**以标明该元素在数组中的位置。对数组元素的访问通常借助于下标（0~n-1）来进行，每个数组元素（即下标变量）都可以当作一个与数组类型相同的变量来使用：

a[0]=5;

a[1]++;

a[3]=a[2]+a[1];

cout<<a[5]； X下标范围是0~4，访问超限（下标溢出），错误

**存储方式：**

相邻元素间的地址是连续的

int a[5];

for(int i=0;i<5;i++)

cout<<&a[i]<<" ";

（int型变量大小为4个字节）

初始化：

一维数组

（1）全部“明确”赋初识值：

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

for(int i=0;i<5;i++)

cout<<a[i]<<" ";

（2）部分赋初始值：

int a[5]={1,2,3};

for(int i=0;i<5;i++)

cout<<a[i]<<" "; //1，2，3，0，0

（3）“元素个数”缺失：

int a[]={1,2,3};

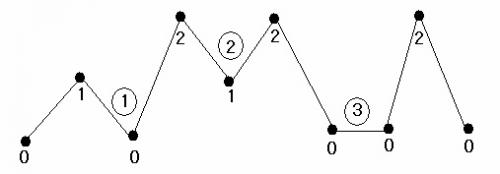
for(int i=0;i<5;i++)

cout<<a[i]<<" "; //1，2，3，两个垃圾数据，数组元素个数为3

代码示例1：

一组数，分别表示地平线的高度变化。高度值为整数，相邻高度用直线连接。找出并统计有多少个可能积水的低洼地？

如图：地高变化为 0 1 0 2 1 2 0 0 2 0



则应有3个可能积水的低洼地

输入：

两行，第一行n,表示有n个数。第2行连续n个数表示地平线高度变化的数据，保证首尾为0。(3<=n<=10000,0<=高度<=1000)

样例输入：

10

0 1 0 2 1 2 0 0 2 0

输出：

一个整数，可能积水的低洼地的数量

样例输出：

3

int height[10001]={-1};

int n;

cin>>n;

int i;

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

{

cin>>height[i];

}

int count=0; //记录水洼的数量

int left=0;

for(i=1;i<n;i++)

{

if(height[i]<height[i-1])

{

left=1; //形成左坡

}

if(height[i]>height[i-1] && left==1)

{

count++; //形成右坡，水洼数+1，左坡失效

left=0;

}

}

cout<<count<<endl;

代码示例2：

对N！进行质因子分解（1<N<=20）

20!=(2^18)\*(3^8)\*(5^4)\*(7^2)\*11\*13\*17\*19

10!=(2^8)\*(3^4)\*(5^2)\*7

int N;

long long result=1; //20!>2^31-1

cin>>N;

int pow=N; //N在后面还有用，保存原始数据

while(pow>=2)

{

result\*=pow;

pow--;

}

//cout<<result<<endl; //验证阶乘是否运算正确

//result=N!

int factor[9]={0}; //记录质因子（3,5,7,11,13,17,19）最多有8种,一定要多设置1个

int power[8]={0}; //记录质因子的幂的次数

int i,j,isprime;

int num=0;

for(i=2;i<=N;i++)

{

//找出N!以内的所有质因子也就是N以内所有的质数

isprime=1; //1表示i是质数，0表示不是

for(j=2;j<=sqrt(double(i));j++)

{

if(i%j==0)

{

isprime=0;

break;

}

}

if(isprime) //i是N!的质因子

{

factor[num]=i;

num++;

}

}

/\*

for(i=0;i<8;i++) //验证质因子即N以内质数寻找和存储是否正确

{

if(factor[i])

{

cout<<factor[i]<<" ";

}

}

\*/

long long res=result;

cout<<N<<"!="<<result<<"=";

for(i=0;i<8;i++)

{

if(factor[i])

{

while(res%factor[i]==0)

{

power[i]++; //记录每个质因子的幂的次数

res/=factor[i];

}

cout<<"("<<factor[i]<<"^"<<power[i]<<")";

if(factor[i+1]!=0) //当前质因子不是最后一个的质因子时，这里访问了i+1，如果声明的时候只留了8个会有问题

{

cout<<"\*";

}

}

}

cout<<endl;