项目经理带你-零基础学习 C/C++
【从入门到精通】

项目九 地震监测系统

第 1 节 项目需求

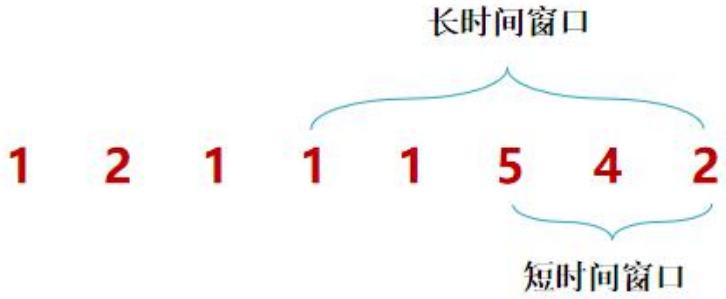
地震监测系统主要是利用地震检波器收集到的地壳运动信息，从而预测和确定地震的震中以及强度 。

预测方法

地震检波器每隔固定的时间间隔采样一次预测地震的能量数据，并保存到文件中，地震监测系统会从文件中读取相应的能量数据，测试在给定的时间点上，一个短时间窗口内的取样值与一个长时间窗口内取样值的商，如果这个比例高于给定的阈值，那么在这个事件点上极有可能发生地震。

取样方法：无论短/长时间内的取样值都是使用给定点能量数据的平方加上该点之前的一小部分点能量值的平方之和再求平均值

如： 某个时间点的及之前的 7 个能量数据如下,时间间隔是： 0.01 秒，短时间周期取 2 个点，长时间周期取 5 个点:



则： 短时间窗口内的取样值： (5x5 + 4x4 + 2x2) / 3 = 15

长时间窗口内的取样值： (5x5 + 4x4 + 2x2 +1x1 + 1x1) / 5 = 9

具体开发需求

1.问题描述：

使用数据文件中的一组地震检波器测量值确定可能的地震事件的位置。

2. 输入输出描述：

- 程序的输入是名为 seismic.dat 的数据文件和用于计算短时间能量和长时间能量的取样值的数目。输出是给出关于潜在的地震事件次数的报告。
- seismic.dat 的结构是这样的，第一行包含两个值： 地震检波器能量值的数目和时间间隔，从第二行开始就是能量值的数据，以空格分开
- 短时间窗口和长时间窗口的值可以由键盘读入
- 判定地震事件给定的阀值是 1.5

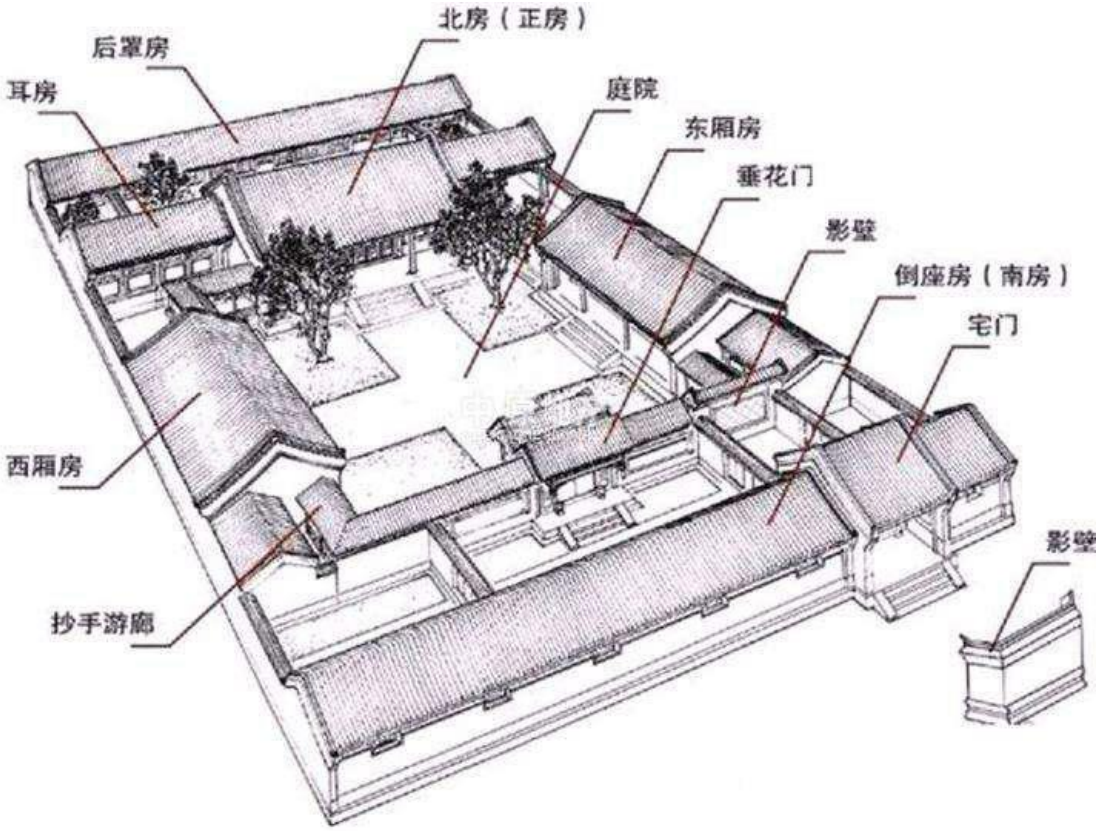
seismic.dat 中的数据如下：

11　0.01

1　2　1　1　1　5　4　2　1　1　1

第 2 节 项目精讲

1. C++程序的内存分区



北京四合院

1、栈区（stack）：由编译器自动分配释放，存放函数的参数值，局部变量值等。

2、堆区（heap）：一般由程序员分配释放，随叫随到，挥之即走。

3、全局/静态区（static）：全局变量和静态变量的存储是放在一起的，在程序编译时分配。

4、文字常量区：存放常量字符串。

5、程序代码区：存放函数体（包括类的成员函数、全局函数）的二进制代码

```
// demo 9-1.c

#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>


int laoyezi = 58;

//1.栈的内存
void say_hello(const char * msg){ //函数的参数和局部变量存放在栈区
    //int x,y,z;

    if(msg){
        printf("Hello, %s!", msg);
    }else {
        printf("Hello, who are u?\n");
    }

    printf("老爷子今年  %d  岁!\n", laoyezi);
```

```
	//printf("小姐今年  %d  岁!\n", girl);
}

void gui_fang(int jiaren){
	static int girl = 17;
	if(girl==17) {
		girl = 18;
	}else {
		girl = 16;
	}

	printf("小姐今年  %d  岁!\n", girl);
}

int main(void) {
	int num = 0;
	int *salary = NULL;

	//4.字符串常量
	char *p = "童养媳";
	char *p1 = "童养媳";

	printf("p: 0x%p    p1: 0x%p\n", p, p1);
	system("pause");
	exit(0);

	//3 .全局变量/静态变量
	printf("老爷子今年  %d  岁!\n", laoyezi);
	gui_fang(0);
	gui_fang(0);
	//salary = new int(666);

	//2. 堆，动态没存分配
	salary = new int;     //在堆区分配动态内存
	*salary = 100;
	printf("salary: %d\n", *salary);
	delete salary;
	system("pause");
	exit(0);
	//say_hello("骚年们！ ");
	printf("请输入需要雇佣的农民数量：\n");
	scanf_s("%d", &num);
	if(num<=0){
		//提示用户重新输入
	}

	salary = new int[num];
```

```
    for(int i=0; i<num; i++){
        *(salary+i)= i+1;
    }

    for(int i=0; i<num; i++){
        printf("第%d 个农民的薪资: %d\n", i+1, salary[i]);
    }
    printf("---over---\n");
    delete[] salary;

    //切记，delete 后的动态内存，禁止访问
    /*for(int i=0; i<num; i++){
        printf("第%d 个农民的薪资: %d\n", i+1, salary[i]);
    }*/

    system("pause");
    return 0;
}
```

2. 为什么要使用动态内存

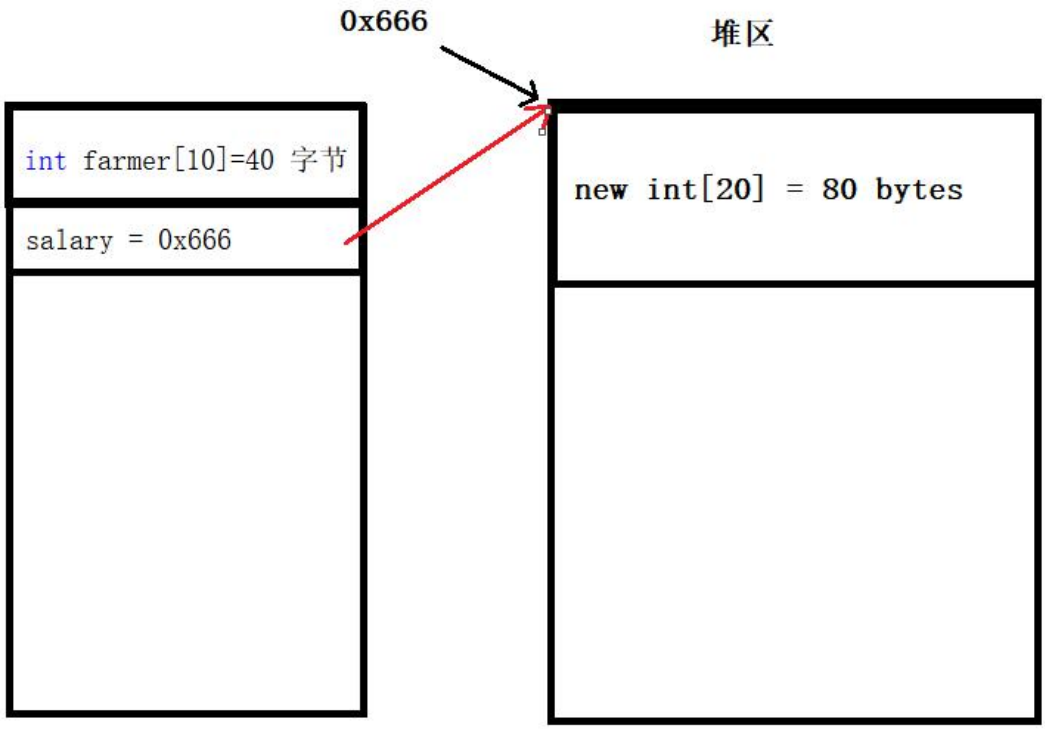
1.按需分配，根据需要分配内存，不浪费

```
// demo9-2.c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>

int main(void) {
    int farmer[10]={20, 22, 25, 19, 18, 23 ,17, 28, 30, 35};
    int num = 0;
    int *salary = NULL;

    printf("请输入需要雇佣的农民数量：\n");
    scanf_s("%d", &num);

    if(num<=10){
        //提示用户重新输入
    }

    //后面新增的都是 18
    salary = new int[num];
    //第一种，逐个赋值
    /*for(int i=0; i<sizeof(farmer)/sizeof(int); i++){
        *(salary+i)= farmer[i];
    }*/
    //第二种，内存拷贝
    memcpy(salary, farmer, sizeof(farmer));

    for(int i=sizeof(farmer)/sizeof(int); i<num; i++){
        //salary[i] = 18;
        *(salary+i) = 18;
    }

    for(int i=0; i<num; i++){
        printf("第%d 个农民的薪资: %d\n", i+1, salary[i]);
    }

    delete[] salary;

    system("pause");
    return 0;
}
```

内存拷贝函数
void *memcpy(void *dest, const void *src, size_t n);
#include<string.h>

功能：从源 src 所指的内存地址的起始位置开始拷贝 n 个字节到目标 dest 所指的内存地址的起始位置中

new 分配内存布局图



2.被调用函数之外需要使用被调用函数内部的指针对应的地址空间

```
// demo9-3.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//通过返回动态内存的指针
int * demo(int count){
    int *ap = NULL;

    //new delete C++ 的动态内存分配操作符 c 语言是 malloc
    ap = (int *)malloc(sizeof(int) * count);//参数：所需内存的字节数
    //ap = new int[count];

    for(int i=0; i<count; i++){
        ap[i] = 100+i;
    }

    for(int i=0; i<count; i++){
        printf("*(ap+%d) = %d\n", i, *(ap+i));
    }

    return ap;

}

//通过二级指针来保存
void demo1(int count, int ** pointer_p){
    int * ap = NULL;

    *pointer_p=(int *)malloc(sizeof(int) * count);

    ap = *pointer_p;

    for(int i=0; i<count; i++){
        ap[i] = 100+i;
    }

    for(int i=0; i<count; i++){
        printf("*(ap+%d) = %d\n", i, *(ap+i));
    }
}

int main(void){
    //两种方式获取被调用函数内部的内存
    int * pointer = NULL;
    int count = 10;
```

```
    //第一种，通过返回动态内存的指针
    //pointer = demo(count);

    //第二种，通过二级指针来保存
    demo1(count, &pointer);

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

    //用完了，要记得释放
    free(pointer); //c 语言中的释放内存函数，相当于 delete

    system("pause");
    return 0;
}
```

C 内存分配：

void *malloc(size_t size);

void free(void *);

malloc　在内存的动态存储区中分配一块长度为 size 字节的连续区域返回该区域的首地址.

3.突破栈区的限制，可以给程序分配更多的内存

```
// demo 9-4.c
#include <stdlib.h>
#include<stdio.h>
#include<string.h>


//栈区的空间大小是有限制的，windows 上一般是 1M - 2M
void demo(){
    //int a1[102400*2]; //100k*2*4 = 800K
    //int a1[102400*3]; //100k*3*4 = 1200K = 1.2M
    int * a1;
    //如果使用堆的话，64 位 windows 10 系统的限制是 2G
    a1 = (int *)malloc((int)(1024*1000*1000));//分配 2G
    a1[0]=0;
    printf("This is a demo.\n");
}

int main(void){
    printf("--start--\n");
    demo();
    printf("--end--\n");
    system("pause");
    return 0;
}
```

3. 动态内存的分配、使用、释放

new 和 delete 基本语法

1）在软件项目开发过程中，我们经常需要动态地分配和撤销内存空间，特别是数据结构中结点的插入与删除。在 C 语言中是利用库函数 malloc 和 free 来分配和撤销内存空间的。C++提供了较简便而功能较强的运算符 new 和 delete 来取代 malloc 和 free 函数。

(注意： new 和 delete 是运算符，不是函数，因此执行效率高。)

2）虽然为了与 C 语言兼容，C++仍保留 malloc 和 free 函数，但建议用户不用 malloc 和 free 函数，而用 new 和 delete 运算符。

new 运算符的例子：

new int; //开辟一个存放整数的存储空间，返回一个指向该存储空间的地址(即指针)
new int(10); //开辟一个存放整数的空间，并指定该整数的初值为 10，返回一个指向该存储空间的地址
new char[100]; //开辟一个存放字符数组(包括 100 个元素)的空间，返回首元素的地址
new int[5][4]; //开辟一个存放二维整型数组(大小为 5*4)的空间，返回首元素的地址
float *p=new float (3.14159); //开辟一个存放单精度数的空间，并指定该实数的初值为//3.14159，将返回的该空间的地址赋给指针变量 p

3）new 和 delete 运算符使用的一般格式为：

new 运算符 动态分配堆内存
使用方法：

```
指针变量 = new 类型(常量);
指针变量 = new 类型[表达式];     //数组
指针变量 = new 类型[表达式][表达式] //二维数组
```

作用：从堆上分配一块“类型”指定大小的存储空间，返回首偶地址
其中：“常量”是初始化值，可缺省
创建数组对象时，不能为对象指定初始值

delete 运算符 释放已分配的内存空间
使用方式：
普通类型（非数组）使用： delete **指针变量**;
数组 使用: delete[] **指针变量**;
其中“**指针变量**” 必须时一个 new 返回的指针！

```
// demo 9-5.c

#include <stdlib.h>
#include <iostream>
using namespace std;
```

```
//分配基础类型
int main007(void) {
    //第一种分配动态内存不执行初始化
    int *p1 = new int;
    *p1 = 100;

    //第二种分配动态内存同时执行初始化
    int *p2 = new int(100);
    // 第三种 malloc 返回值是 void *
    int *p3 = (int *)malloc(sizeof(int));


    free(p1);  //基础类型可以 new free  可以混搭
    delete p3; //基础类型可以 malloc delete  可以混搭
    delete p2; //free(p2);  同样效果

    system("pause");
    return 0;
}

//分配数组变量
int main(void) {
    int *p1 = (int *) malloc(sizeof(int)*10);
    //p[0] - p[9]   *(p+9)
    int *p2 = new int[10];

    delete p1;   // free(p1);    可以混搭
    //free(p2);  //可以混搭
    delete[] p2;

    system("pause");
    return 0;
}
```

4. C++程序员的噩梦-内存泄漏

内存泄漏（Memory Leak） - 是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

```
// demo 9-6.c

#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <Windows.h>

using namespace std;

void A_live(){
    int * p = new int[1024];
    //挥霍
    p[0]=0;
    //申请的内存必须要“还”
}

void B_live(){
    int * p = new int[1024];
    //正常的开支
    p[0]=0;
    delete[] p;
}

int main(void) {
   /*
   for(int i=0; i<100000; i++){
        A_live();
        Sleep(50);
    }*/


    for(int i=0; i<100000; i++){
        B_live();
        Sleep(50);
    }

    system("pause");
    return 0;
}
```

5. 变量的 4 种存储类型

所有的数据都有两种类型
数据类型： 如 int,float 等

存储类型： 总共有四种存储类型的变量，分别为自动变量（auto）、静态变量（static）、外部变量（extern)以及寄存器变量（register）。

auto - 函数中所有的非静态局部变量。

register - 一般经常被使用的的变量（如某一变量需要计算几千次）可以设置成寄存器变量，register 变量会被存储在寄存器中，计算速度远快于存在内存中的非 register 变量。

static - 在变量前加上 static 关键字的变量。

extern - 把全局变量在其他源文件中声明成 extern 变量，可以扩展该全局变量的作用域至声明的那个文件，其本质作用就是对全局变量作用域的扩展。

```
// demo 9-7.c

#include <stdlib.h>
#include <iostream>

using namespace std;

extern int extern_value;
static int yahuan_xiaoli = 24;//全局静态变量
//int yahuan_extern = 30;

//寄存器变量
void register_demo(){
    register int j= 0;
    printf("j: %d\n", j);
    //C++ 的 register 关键字已经优化，如果我们打印它的地址，它就变成了
    //普通的 auto 变量
```

```
    for(register int i=0; i<1000; i++){
        //....
    }

    printf("&j : 0x%p\n", &j);

    {
        int k=100;
        k+=j;

    }
    printf("register_demo -  register_demo: %d\n", yahuan_xiaoli);
}

//局部静态变量
void static_demo(){
    static int girl = 18;
    int yahuan = 17;

    ++girl;
    ++yahuan;
    printf("girl: %d   yahuan: %d\n", girl, yahuan);
    printf("static_demo -  register_demo: %d\n", yahuan_xiaoli);
}

//外部变量
void extern_demo(){
    extern_value++;
    printf("extern_value: %d\n", extern_value);
}

int main(void) {
    int i = 0;  //C 语言的 auto 不会报错，C++ auto 已经升级啦
    //register_demo();
    //static_demo();
    //static_demo();
    //static_demo();
    extern_demo();
    system("pause");

    return 0;
}
```

6. 变量的作用域和生存周期

存储类别	存储期	作用域	声明方式
auto	自动	块	块内
register	自动	块	块内,使用关键字 register
static(局部)	静态	块	块内，使用关键字 static
static(全局)	静态	文件内部	所有函数外,使用关键字 static
extern	静态	文件外部	所有函数外

7. 函数返回值使用指针

可以返回函数内部：动态分配内存地址　局部静态变量地址 以及全局静态变量和外部变量地址

```
// demo 9-8.c

#include <iostream>
#include <stdlib.h>

using namespace std;

int * add(int x, int y)
{
    int sum = x + y;
    return &sum;
}

//返回动态内存分配地址
int * add1(int x, int y)
{
    int * sum = NULL;
    sum = new int;
    *sum = x + y;
    return sum;
}
```

```
//返回局部静态变量的地址
int * add2(int x, int y)
{
    static int sum = 0;
    printf("sum: %d\n", sum);
     sum = x + y;
     return &sum;
}

int main()
{
     int a = 3, b = 5;
    int *sum = NULL;
     //cout << add(a, b) << endl;
    //sum = add(a, b);//不能使用外部函数局部变量的地址 bad

    //接收外部函数动态内存分配的地址  ok
    //sum = add1(a, b);
    //cout<<*sum<<endl;
    //delete sum;

    //接收外部函数局部静态变量的地址
    sum = add2(a, b);
    cout<<*sum<<endl;
    *sum = 88888;
    add2(a, b);
    system("pause");
    return 0;
}
```

8. 常见错误总结

1. 申请的内存多次释放

2. 内存泄漏

3. 释放的内存不是申请时的地址

4. 释放空指针

5. 释放一个内存块，但继续引用其中的内容

6. 越界访问

```
// demo 9-9.c
#include <iostream>
#include <stdlib.h>

using namespace std;

int main()
{
    int * p = new int[18];
    p[0] = 0;
    char *p1 = NULL;
    //... ...
    //... ...
    //delete[] p;//只允许一次释放
    //... ...
    //... ...
    //delete[] p;//1.申请的内存多次释放, 程序出现异常

    //2. 忘记 delete,内存泄漏
    /*
    do{


    }while(1==1);
    */

    //3.释放的内存不是申请时的地址
    /*for(int i=0; i<18; i++){
        cout<<*(p++)<<endl;
    }

    delete [] p;*/
```

```
    //4.释放空指针
    // ... ...
    /*if(1==0){ //比如文件能打开的情况
        p1 = new char[2048];
    }
     ... ...
    delete p1;
    */

    //5.释放一个内存块，但继续引用其中的内容
    /*delete[] p;
    // ...继续码代码...
    p[0]='\0';//绝对禁止
    */

    //6.越界访问
    memset(p, 0, 18*sizeof(int));
    for(int i=0; i<18; i++){
        cout<<*(p++)<<endl;
    }

    //误判
    for(int i=0; i<18; i++){
        cout<<*(p++)<<endl;
    }

    cout<<"come here!"<<endl;

    system("pause");
    return 0;
}
```

9. 英语不是障碍：计算机英语加油站

storage	存储
class	类别
model	模型
option	选择、选项
be helpful to	有助于、对...有用
go over	复习
Concepts	概念
term	术语
hardware	计算机硬件
aspect	方面
occupy	占据，占领
physical	物理的
literature	文献、著作

Storage Classes

C provides several different models, or storage classes , for storing data in memory. To understand the options, it's helpful to go over a few concepts and terms first.

C 提供了多种不同的模型或存储类别（storage class）在内存中储存数据。要理解这些存储类别，先要复习一些概念和术语。

Every programming example in this book stores data in memory. There is a hardware aspect to this—each stored value occupies physical memory. C literature uses the term "object" for such a chunk of memory. An object can hold one or more values. An object might not yet actually have a stored value, but it will be of the right size to hold an appropriate value.

本书目前所有编程示例中使用的数据都储存在内存中。从硬件方面来看，被储存的每个值都占用一定的物理内存，C 语言把这样的一块内存称为对象（object）。对象可以储存一个或多个值。一个对象可能并未储存实际的值，但是它在储存适当的值时一定具有相应的大小

10. 项目实现

1.问题描述：

使用数据文件中的一组地震检波器测量值确定可能的地震事件的位置。

3. 输入输出描述：

- 程序的输入是名为 seismic.dat 的数据文件和用于计算短时间能量和长时间能量的取样值的数目。输出是给出关于潜在的地震事件次数的报告。
- seismic.dat 的结构是这样的，第一行包含两个值： 地震检波器能量值的数目和时间间隔，从第二行开始就是能量值的数据，以空格分开
- 短时间窗口和长时间窗口的值可以由键盘读入
- 判定地震事件给定的阀值是 1.5

seismic.dat 中的数据如下：

11 0.01

1 2 1 1 1 5 4 2 1 1 1

算法设计：

1) 读取文件头并分配内存；
2) 从数据文件读取地震数据，从键盘读取计算能量的短时间和长时间窗口测量值的数目；
3) 计算各个时间点上的短时间窗口和长时间窗口的能量值，打印出可能的地震事件时间，在这里，因为会涉及到频繁调用短时间窗口和长时间窗口的能量值，我们可以将计算能量值设计为单独的一个函数

```
// demo 9-10.c
#include <fstream>
#include <string>
#include <iostream>
#include <cmath>

using namespace std;

const double THRESHOLD = 1.5;

//计算短/长时间窗口能量数据的采样值
double power_w(double arr[], int length, int n);


int main(){
    string filename;
    ifstream fin;
    int num = 0, short_window = 0, long_window = 0;
    double time_incr = 0, *sensor = NULL, short_power = 0, long_power =
0;
    double ratio;

    cout<<"Enter name of input file"<<endl;
    cin>>filename;

    fin.open(filename.c_str());
    if(fin.fail()){
        cerr<<"error opening input file"<<endl;
        exit(-1);
    }else {
        fin>>num>>time_incr;
        cout<<"num: "<<num <<" time_incr: "<<time_incr<<endl;

        if(num>=0){
            sensor = new double[num];

            for(int i=0; i<num; i++){
                fin>>sensor[i];
            }

            cout<<"Enter number of points for short-window:"<<endl;
            cin>>short_window;

            cout<<"Enter number of points for long-window:"<<endl;
            cin>>long_window;

            //分析能量数据找出地震事件
```

```
            for(int i=long_window-1; i<num; i++){
                short_power = power_w(sensor, i, short_window);
                long_power = power_w(sensor, i, long_window);

                ratio = short_power / long_power;

                if(ratio > THRESHOLD){
                    cout<<" Possible event at "<< time_incr * i<<"
seconds\n";
                }
            }

            delete[] sensor;
        }

        fin.close();
    }
    system("pause");
    return 0;
}

//统计短/长事件窗口对应能量的采样值
double power_w(double arr[], int length, int n){
    double xsquare = 0;

    for(int i=0; i<n; i++){
        xsquare+= pow(arr[length-i], 2);
    }
    return xsquare/n;
}
```

❖ 编程思维修炼

【问题区】

在一次竞赛中，A、B、C、D、E 等五人经过激烈的角逐，最终取得了各自的名次，他们的一个好朋友很遗憾没有观看到比赛，在比赛结束后这个朋友询问他们之间的名次是得知：C 不是第一名，D 比 E 低二个名次，而 E 不是第二名，A 即不是第一名，也不是最后一名，B 比 C 低一个名次。编写程序，计算这五个人各自的名次并输出。

【提示区】

这是一道逻辑推理题。其中的关键点在 D 比 E 低两个名次，那么 D 和 E 的位置只可能在（1，3）、（2，4）或（3，5），而 E 不是第二名，后面还有其他条件，这下你懂了吧？

11. 职场修炼：如何应付老鸟的抱怨

任正非说道：“请转告他们，我实际上是在感谢他们推广了我们。”

面对老鸟抱怨，比较妥善的态度 ：

礼貌谦逊的态度

态度很重要，不仅仅体现的是你个人的素质问题，更重要的是影响你和他人的互动关系，社会就是人与人的关系综合体。

职场如战场，充斥着形形色色的人，也就会有各种各样复杂的关系。礼貌、谦虚的人总是更受他人喜欢，无论是领导还是同事，都讨厌那些耍个性的人。生僻怪异的性格只能让你无法融入到团队。

积极乐观的心态

任何一个行业都不是我们一眼看起来那么简单的，必须要有时间和实践的沉淀，方能获得真知。一年入门，三年入行，五年入道，七年入定，十年入化，请记住！

遇到困难时，要乐观面对，遇到工作不顺时，要及时调整好心态，心存希望，你才会有希望。少一点抱怨，因为抱怨解决不了任何问题，只会让事情更糟糕。多与传递正能量的人交流相处，不要被那些消极负面的人带进黑暗的深渊。

12. 逼格提升：内存泄漏检测工具

VisualC++ debugger 和 CRT 库

第一步： 包含以下头文件

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
```

第二步： 接管 new 操作符

```
#ifdef _DEBUG
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ ,__LINE__)
#define new DBG_NEW
#endif
#endif
```

第三步： 在代码结束出输出内存泄漏信息

```
_CrtDumpMemoryLeaks();
```

内存泄漏工具：

Windows : Purify,BoundsCheaker、Deleaker、VisualLeak Detector（VLD），

Linux 平台：Valgrind memcheck

13. 程序员的试金石：链表的使用

- C/C++ 高级 vip 学员请在马上还是的数据结构课程中学习
- C/C++ 初级学员会单独提供链表章节给大家学习，敬请期待!

14. 项目练习

1. 编写一个程序，链接两个字符串字面常量，将结果保存在一个动态分配的 char 数组中。重写这个程序，连接两个标准 string 对象。

2. 编写一个程序，使用 cin 从标准输入输入 3 段文字，保存到一段动态分配的内存中，每一段文字输入后，必须要即时保存到动态内存中。

3. 下列程序的功能是：首先,根据输入的二维数组的行数和列数，动态地为该数组分配存储空间；其次，向二维数组中 输入数据；最后输出该数组中的所有元素。请完善下面的程序。

```
#include<iostream>
#include <stdlib.h>

Using namespace std;

int main(void){
    int *p;
    int row, col;
    int i,j, k=1;
    cout<<”Input number of row:\n”;
    cin>>row;
    cout<<”Input number of column\n”;
    Cin>>col;
    p=new int[row*col];

    If( ___1___ ) {
        cout<<”Not allocate memory!\n”;
        exit(1);
    }

    for(i=0;i<row;i++){
        for(j=0; j<col; j++) p[____2_____] = k++;//根据行列下标，计算下表值
    }


    for(i=0;i<row;i++){
        for(j=0; j<col; j++)
            cout<<p[____3_____] <<’\t’;//根据行列下标，计算下表值
            cout<<endl;
    }

    delete []____4_____;

}
```