苏州大学实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学院 | 计算机科学与技术学院 | 专业 | 计算机科学与技术 |
| 学号 | 2427405039 | 姓名 | 刘志轩 |
| 课程名称 | C++程序设计 | 指导老师 | 陈伟 |
| **综合实验二** | | | |
| **一．实验目的**   1. 掌握字符串、函数、向量等使用方法。 2. 掌握通过多个步骤，完成较复杂的处理过程。   **二．实验内容**  在工业生产中，企业在收集产品的质量数据以后需要进行异常判断。但由于收集过程中有多种不可控因素，只能以字符串数组的形式返回收集结果。现假设该数组为arr:   1. 编写函数func1，去掉arr中的非数值类型的元素，结果以向量形式vector<string>返回； 2. 编写函数func2，以第1题返回的向量作为输入，进行如下异常判定：给定一组样本，如果某个样本不在[均值-sigma,均值+sigma]闭区间内，则该样本被认为是异常值，如下图所示的红色区域。其中，sigma代表样本标准差，其计算公式为，代表一个样本，是输入样本的均值，是样本的个数。最终，剔除所有异常值，并以向量形式vector<double>返回所有正常样本；      1. 编写函数func3，第2题返回的向量作为输入，并进行如下异常判定：给定一组样本和正整数num，其中num小于等于样本个数，如果样本中存在连续num个、或大于num个样本是递增或递减，则都被认为是异常值。如下图所示，给定num=3，由于、、是连续递减，所以都是异常值；、、、是连续递增，所以也都是异常值；最终，函数需返回异常样本的总个数，本例中该返回值是6。      1. 编写函数func4，同样以第2题返回的向量作为输入，直接进行如下异常判断：给定一组样本和正整数num，其中num小于等于样本个数，如果样本中存在连续num个、或大于num个样本上下交错，则都被认为是异常值。如下图所示，给定num=3，由于、、上下交错，所以都是异常值；同理，、、、也都是异常值；最终，函数需返回异常样本下标组成的字符串(下标用空格隔开)，本例的返回结果是“1 2 3 5 6 7 8”。     **答题提示：**   1. 在数组arr中，正常的数值形如“XXX.XX”或“XXX”，其中X代表0到9的某个数字； 2. 如果两个值相同，则无法构成递减、递增、交错； 3. 下图是5个测试用例与对应的运行结果：+++             **三、其他说明**   1. 在给定的sample.cpp的基础上完善程序; 2. 程序主体部分不能修改。   **四、实验程序（含必要的注释）及测试结果**  #include<iostream>  #include<vector>  #include<string>  #include<sstream>  #include<cmath>  #include <regex>  using namespace std;  vector<string> func1(string arr[], int n)  {      vector<string> vec;      // 正则表达式匹配数值（整数或浮点数）      std::regex pattern(R"([+-]?\d+(\.\d\*)?)");      for (int i = 0; i < n; ++i) {          if (std::regex\_match(arr[i], pattern)) {              vec.push\_back(arr[i]);          }      }      return vec;  }  vector<double> func2(const vector<string>& vec)  {      vector<double> res;      double sum=0,ave=0,cnt=0,sigma=0;      for (auto i=vec.begin();i<vec.end();i++){          sum+=stod(\*i);          cnt+=1;      }      ave=sum/cnt;      sum=0;//节省内存      for (auto i=vec.begin();i<vec.end();i++){          sum+=(ave-stod(\*i))\*(ave-stod(\*i));      }      sigma=pow((sum)/(cnt-1),0.5);//计算sigma      for (auto i=vec.begin();i<vec.end();i++){          if (stod(\*i)<=sigma+ave && stod(\*i)>=ave-sigma){              res.push\_back(stod(\*i));          }      }      return res;  }  int func3(const vector<double>& vec, unsigned int num)  {      int res = 0;      int cntrise=1,cntdown=1,cover=0,cnt=0;      int check[vec.size()]={0};//记录每一段末尾，防止重复计数      for (auto i=vec.begin()+1;i<vec.end();i++){          cnt++;          if ((\*i)>(\*(i-1))){              cntrise++;              if (cntdown>=num){                    res=res+cntdown-check[cnt-cntdown];                  check[cnt-1]=1;              }              cntdown=1;          }          else if ((\*i)<(\*(i-1))){              cntdown++;              if (cntrise>=num){                  res=res+cntrise-check[cnt-cntrise];                  check[cnt-1]=1;              }              cntrise=1;          }          else {              if (cntrise>=num){                  res=res+cntrise-check[cnt-cntrise];}              if (cntdown>=num){                  res=res+cntdown-check[cnt-cntrise];}              check[cnt-1]=1;              cntdown=1;cntrise=1;          }      }      if (cntrise>=num){                  res=res+cntrise-check[cnt-cntrise];}      if (cntdown>=num){                  res=res+cntdown-check[cnt-cntrise];}      return res;  }  string func4(const vector<double>& vec, unsigned int num)  {      string res = "";      int check[vec.size()]={0};//记录上下      for (int i=1;i<vec.size()-1;i++){          if ((vec[i]!=vec[i+1] && vec[i]!=vec[i-1])&&(vec[i]-vec[i+1])/(vec[i]-vec[i-1])>0){              check[i]=1;              check[i-1]=1;              check[i+1]=1;          }      }      int t=0;string temp="";      for (int j=0;j<vec.size();j++){            if (check[j]){              temp+=to\_string(j);              temp+=' ';              t++;          }          else{              if (t>=num){                  res+=temp;                  temp="";t=0;              }          }      }      if (t>=num){//最后一段防止漏记          res+=temp;      }      return res;  }  int main()  {      //输入数据      string arr[] = { "", "12.51", "ab.2", "13.52", "13..52",  "13.5.1", "14.54", "14.52", "12.53", "13.56", "0013.54", "010", "   ", "13.   5" };      //string arr[]= {"35.26", "36.2.4", "35.42", "36.x15", "35.3 6", "35.98", "36.01", "39.99", "035.22", "35", " ", "35.01"};      //string arr[]={"110.12", "110.15", "110..16", "110", "110.18", "130.25", "110.4 5", " ", "110.09", "110.09", "110.09", "110.13", "110.10"};      //string arr[]={"237.2", "237.1", "236.9", "236.5", "236.1", "235..8", "235.5", "235", "234. 8", "2333", "234.1", "237.2", "236.3"};      //string arr[]={"555.3", "555.3", "555.3", "555.4", "555.5", "55.5.5", "555.5", " ", "555.5" "555.4", "555.3", "5555", "555.2", "555.1", "549.8", "550.6"};      //调用函数func1      vector<string> res1 = func1(arr, sizeof(arr) / sizeof(string));      for (unsigned int i = 0; i < res1.size(); i++)          cout << i << " " << res1[i] << endl;      cout << "---------------------" << endl;      //调用函数func2      vector<double> res2 = func2(res1);      for (unsigned int i = 0; i < res2.size(); i++)          cout << i << " " << res2[i] << endl;      cout << "---------------------" << endl;      //调用函数func3      unsigned int num3 = 3;      int res3 = func3(res2, num3);      cout << res3 << endl;      cout << "---------------------" << endl;      //调用函数func4      unsigned int num4 = 3;      string res4 = func4(res2, num4);      cout << res4 << endl;  }            **五、实验总结**  本次实验围绕工业质量数据的异常检测展开，通过设计四个核心函数实现了从数据清洗到多层异常判定的完整流程，有效掌握了字符串处理、向量容器操作及复杂逻辑封装。首先，func1利用正则表达式`R"([+-]?\d+(\.\d\*)?)`精准筛选出合法数值字符串，剔除包含非数字字符或格式错误的无效数据，为后续处理提供有效输入。接着，func2通过计算样本均值与标准差，依据统计区间`[均值-sigma, 均值+sigma]`剔除异常值，实现基于统计量的初步异常检测，其中标准差采用无偏估计（分母为`n-1`）确保计算准确性。针对序列模式异常，func3通过维护递增/递减计数器，标记连续长度≥`num`的异常区间，结合标记数组避免重复计数；func4则通过相邻差值符号变化判断上下交错模式，收集连续≥`num`的异常下标，实现模式识别与结果输出。实验中解决了正则表达式匹配范围、连续序列计数逻辑及交错模式判定等关键问题，通过提供的测试用例验证了各函数的正确性，例如func3在样例中准确识别出6个连续递增/递减异常值，func4按预期返回交错异常的下标字符串。通过本次实验，不仅强化了C++容器操作、数值计算和算法设计能力，还学会将复杂问题拆解为子任务分步解决，为后续开发积累了经验。未来可针对大规模数据优化性能，增加极端场景测试以提升代码鲁棒性，并通过改进变量命名和注释增强可读性。 | | | |