

算法与数据结构设计报告

（ 2019 / 2020学年 第 一 学期）

题 目： 代码分析器

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **学 生 姓 名** | **朱昌宏** |
| **班 级 学 号** | **B17040312** |
| **指 导 教 师** | **徐康** |
| **指 导 单 位** | **计算机科学与技术系** |
| **日 期** | **2019.10.21-2019.11.03** |

|  |  |  |
| --- | --- | --- |
| **支撑指标点** | **评价准则** | **计分（每项10分）** |
| 3.1掌握设计/开发复杂工程问题解决方案所需要的专业知识和开发工具（40分） | 1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，进行合理的分析与设计 |  |
| 2、能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现 |  |
| 3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述 |  |
| 4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便 |  |
| 4.1能基于专业理论，针对计算机及应用领域复杂工程问题设计可行的实验方案（40分） | 5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标 |  |
| 6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解 |  |
| 7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能 |  |
| 8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试 |  |
| 4.2能够根据实验方案，配置实验环境、开展实验，使用定性或定量分析方法进行数据分析与处理，综合实验结果以获得合理有效的结论（20分） | 9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握 |  |
| 10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范 |  |
| 算法与数据结构设计能力测评总分 | |  |
| **指导教师： 年 月 日** | | |
| **备注：** | | |

**代码分析器**

**一、课题内容和要求**

（一）课题内容

读入C源文件，统计文件中的代码、注释和空行的行数以及函数的个数和平均行数，并利用统计信息分析评价该程序的风格。

（二）课题要求

(1) 将程序文件按字符顺序读入源程序；

(2) 边读入程序，边识别统计代码行、注释行和空行，同时还要识别函数的开始和结束，以便统计其个数和平均行数。

(3) 程序的风格评价分为代码（函数平均长度）、注释（占总行数比例）和空行（占总函数比例）三个方面。每个方面分为A，B，C:和D 四个等级。等级的划分标准如下：



图1 程序评价标准图

(4) 测试时先对较小程序进行分析。当你的程序能正确运行时，对你的程序本身进行分析。

**二、课题需求分析**

本课题目标系统“垃圾转运路径规划系统”的功能框架图如图1所示。

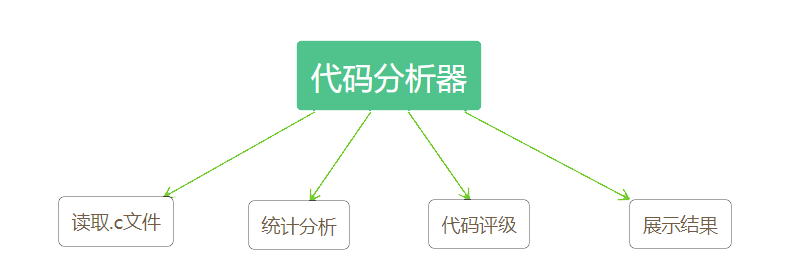


图2 代码分析器功能框架图

（1）支持用图形界面打开C源文件文件，并能够检测识别打开的文件是否为C源文件文件；

（2）利用有限状态自动机实现对C源文件中的函数、注释以及空行进行统计分析；

（3）按照要求对C源文件进行评级；

（4）支持用图形界面显示C源文件的评级结果。

**三、课题相关数据结构及算法设计**

1 主要数据结构

采用Java语言编写程序实现课题，UML设计图如下：

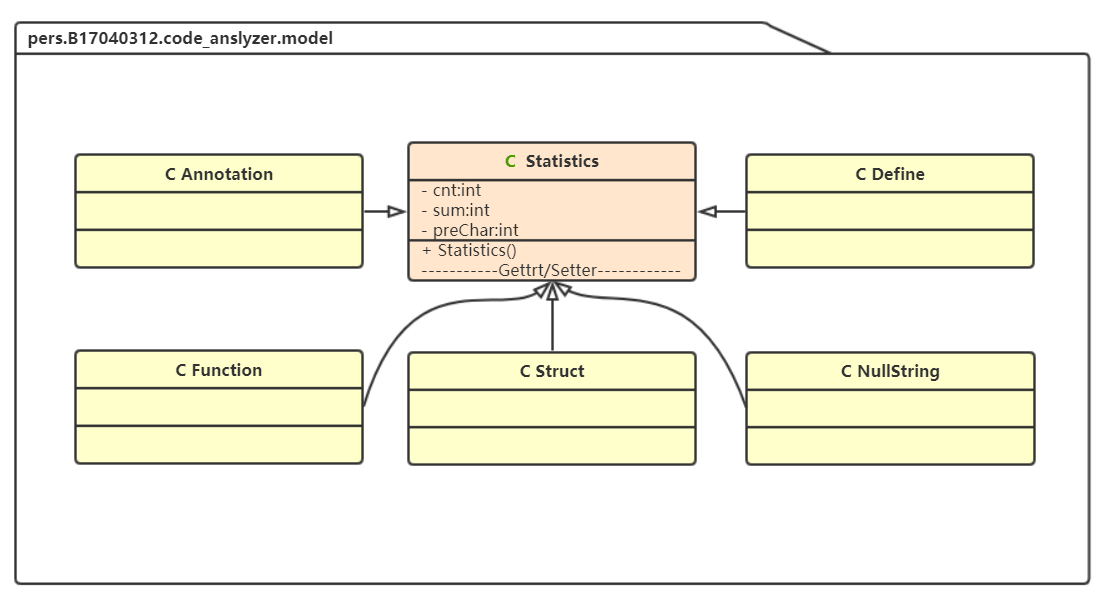


图3 领域模型UML设计图

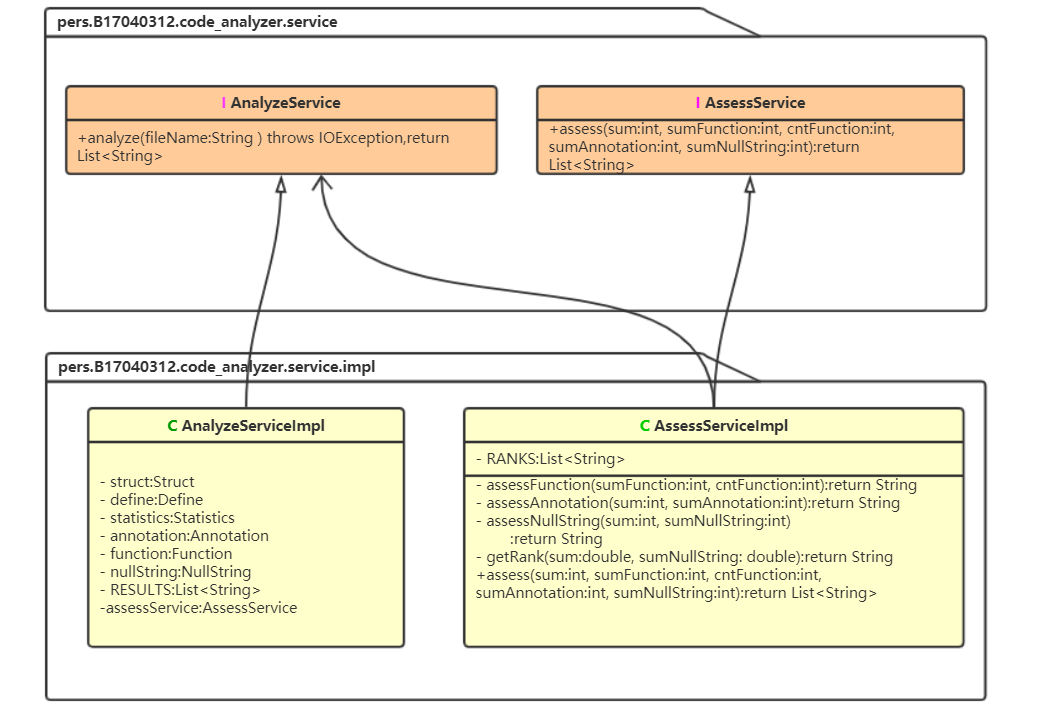


图4 接口模型UML设计图

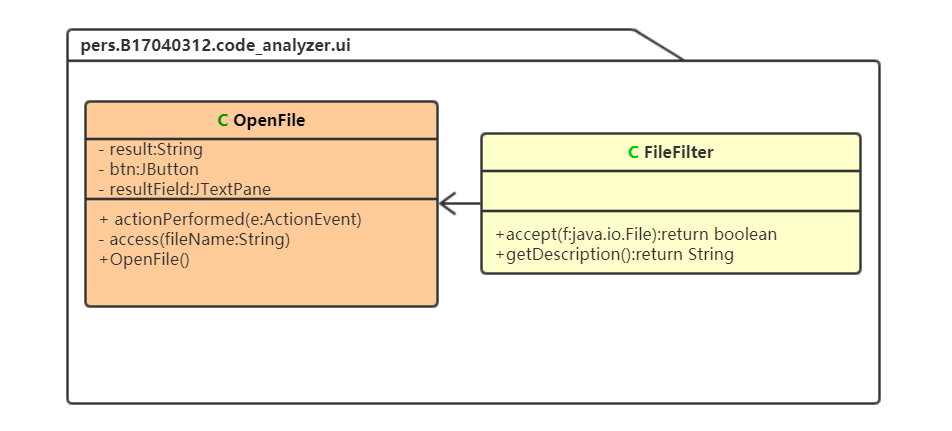


图5 UI模型UML设计图

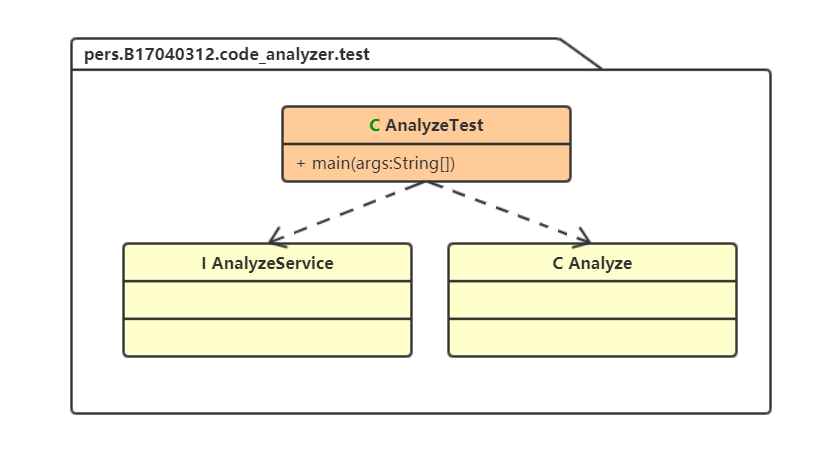


图6 测试模型UML设计图

/\*\*

\* @title: Statistics

\* @description: 用于统计代码长度、个数以及在统计时用到的前字符标记

\* @author: laochonger

\*/

public class Statistics {

int cnt;

int sum;

int preChar;

public int getCnt() {

return cnt;

}

public void setCnt(int cnt) {

this.cnt = cnt;

}

public int getSum() {

return sum;

}

public void setSum(int sum) {

this.sum = sum;

}

public int getPreChar() {

return preChar;

}

public void setPreChar(int preChar) {

this.preChar = preChar;

}

/\*\*

\*

\* @description: 构造函数

\* @param: []

\*

\*/

public Statistics(){

this.cnt=0;

this.sum=0;

this.preChar= 0;

}

}

/\*\*

\* @title: Annotation

\* @description: 继承于Statistics类，统计注释的行数以及保存统计注释时用到的前字符标记

\*/

public class Annotation extends Statistics{

public Annotation(){

super();

}

}

/\*\*

\* @title: Define

\* @description: 继承于Statistics类，保存用于去除"#define"对统计函数的影响时用到的前字符标记

\*/

public class Define extends Statistics{

public Define(){

super();

}

}

/\*\*

\* @title: Function

\* @description: 继承于Statistics类，统计函数的行数以及个数

public class Function extends Statistics{

public Function(){

super();

}

}

/\*\*

\* @title: Struct

\* @description: 继承于Statistics类，保存用于去除"struct"对统计函数影响时用到的前字符标记

\*/

public class Struct extends Statistics{

public Struct(){

super();

}

}

2 主要算法流程

1. 编写图形界面打开C源文件， 设置一个文件选择器，设置按钮标签。

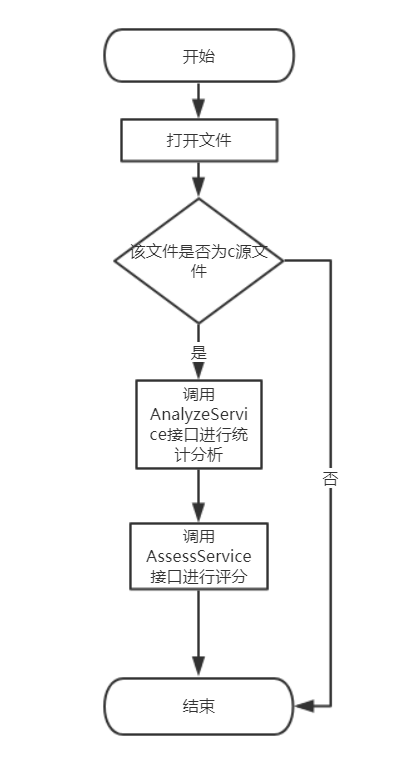


图8 算法总流程图

1. C代码统计分析算法：基于有限状态自动机的思想，设计了8个状态，分别为：
   * 1. 状态0（初始状态）

\* define、函数标记状态

\* 若nowChar=='#'，进入define状态，状态标记开启

\* 若define状态标记开启，且前一字符为'\n'或'\\'，则关闭define状态标记；记录nowChar为define的前一字符,break

\* 否则若nowChar=='/'，可能为单行注释或多行注释，转状态1；

\* 否则若nowChar=='\'' or '"'，则为字符或字符串，用记录下当前字符，转状态6；

\* 否则若nowChar=='{'&&struct.getPreChar()==')'，则模拟函数括号栈变量++

\* 否则若nowChar=='}'&&simulationStack >0，则模拟函数括号里变量++，若变量==0，则函数计数++

\* 最后记录struct.preChar为当前字符

\* 其它情况，状态不变。

* + 1. 状态1

\* 单行或多行注释半状态

\* 若nowChar=='/'，则确定为单行注释{//}，转状态2；

\* 否则若nowChar=='\*'，则确定为多行注释{/\*}，转状态4；

\* 否则，转状态0。

* + 1. 状态2

\* 单行注释开始

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='\n'，则单行注释结束，转状态0；

\* 否则若nowChar=='\\'，是单行注释行继续符，属于情况{//xxx\}，转 状态3；

\* 否则，状态不变。

* + 1. 状态3

\* 单行注释行继续符开始

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='\\'，仍为单行注释行继续符，属于状态{//xxx\\}，状态不变；

\* 否则，属于状态{//xxx\a} 或 {//xxx\n}，转状态2。

* + 1. 状态4

\* 多行注释开始

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='\*'，属于情况{/\*xxx\*}或{/\*\*}，转状态5；

\* 否则，属于情况{/\*x}或{/\*xxx\*aa}，状态不变。

* + 1. 状态5

\* 多行注释可能结束

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='/'，则多行注释结束，转状态0；

\* 否则若nowChar=='\*'，属于情况{/\*xxx\*\*}，状态不变；

\* 否则，属于情况{/\*xxx\*a}，转状态4。

* + 1. 状态6

\* 字符或字符串状态

\* 若nowChar=='\\',为转义字符，属于状态{'\}或{"\}，转状态7

\* 否则，若nowChar==annotation.getPreChar(),说明字符或字符串状态结束，转状态0；

\* 其它情况，状态不变。

* + 1. 状态7

\* 消除转义字符后面的字符的影响

\* 转状态6。

由于该算法是单字符输入，不需要额外的存储空间，所以空间复杂度为O(1)

且该算法的每个字符只判断一次，渐进时间复杂度为O(n)。

该有限状态自动机的状态转移图如下图所示：

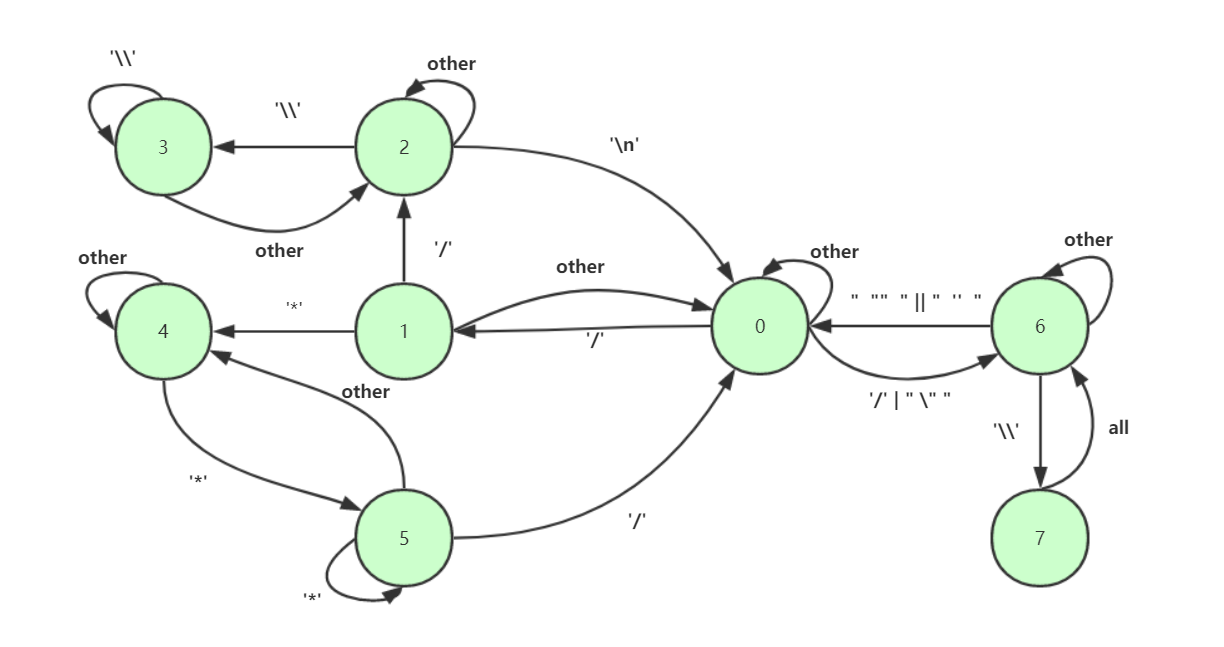


图8 有限自动机的状态转移图

1. C源程序评级算法：根据C源程序统计分析算法中求得的代码总行数、空行行数、注释行数、函数行数以及函数个数，按照给定的评级方案，为C源程序评级。该算法并未引入额外的空间，且每次判断只需要常数级别的时间复杂度，所以该算法的空间复杂度为O(1),时间复杂度也为O(1)。
2. 展示结果：通过得到统计分析算法以及评级算法的数据，将其输出在图形界面上。

**四、源程序代码**

1 AnalyzeServiceImpl.java，内含有限状态自动机的关键代码以及打开文件操作

package pers.B17040312.code\_analyzer.service.impl;

import pers.B17040312.code\_analyzer.model.\*;

import pers.B17040312.code\_analyzer.service.AnalyzeService;

import pers.B17040312.code\_analyzer.service.AssessService;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* @title: AnalyzeServiceImpl

\* @package pers.B17040312.code\_analyzer.service.impl

\* @description: AnalyzeService接口的实现类

\* @author: laochonger

\* @date: 2019-10-29 16:19

\* @version: V1.0

\*/

public class AnalyzeServiceImpl implements AnalyzeService {

/\*\*

\* 创建statistics、annotation、function、nullString类分别为总行数、注释、函数、空行计数

\*/

private Statistics statistics = new Statistics();

private Annotation annotation = new Annotation();

private Function function = new Function();

private NullString nullString = new NullString();

/\*\*

\* 创建struct、define类作为计数的辅助类

\*/

private Struct struct = new Struct();

private Define define = new Define();

/\*\*

\* 储存分析统计以及评级结果的容器

\*/

private List<String>RESULTS = new ArrayList<>();

/\*\*

\* 创建assessService接口

\*/

private AssessService assessService = new AssessServiceImpl();

/\*\*

\*

\*@title: analyze

\*@description: 实现设计的有限状态自动机来分析统计c语言源代码

\*@author: laochonger

\*@date: 2019-10-29 20:13

\*@param: [fileName]

\*@return: java.util.List<java.lang.String>

\*@throws:

\*/

@Override

public List<String> analyze(String fileName) throws IOException {

FileReader fileReader = null;

/\*\*

\* 检测是否打开的是文件夹

\*/

try {

fileReader = new FileReader(fileName);

} catch (FileNotFoundException e) {

RESULTS.add("\r\n\r\n\r\n\r\n 大家就当无事发生");

return RESULTS;

//e.printStackTrace();

}

/\*\*

\* 按传入的文件名用BufferedReader逐字读入.c文件

\*/

BufferedReader bufferedReader = new BufferedReader(fileReader);

int nowChar; //当前读入字符

int state = 0; //初始状态

int simulationStack = 0; //用int变量模拟"{"、"}"的配对

int flagForDefine = 0; //用于状态0中判断是否还在#define状态中

int flagFirst = 1; //用于判断是否是首次读入

/\*\*

\* 循环逐字读入

\*/

while ((nowChar = bufferedReader.read()) != -1){

if(flagFirst==1&&nowChar=='\r')nullString.setSum(1);

//首次读入且是回车符，代表是空行

if(flagFirst==1)flagFirst=0;

//关闭首次读入开关

if(nowChar=='\n')statistics.setSum(statistics.getSum()+1);

//如果读入的为换行符，则总行数++

if(nowChar=='\n'&&simulationStack >0){

function.setSum(function.getSum()+1);

}

if(nullString.getPreChar()=='\n'&&nowChar=='\n'){

//如果连续读入'\n'，则空行++

nullString.setSum(nullString.getSum()+1);

}

if(nowChar!=' '&&nowChar!='\t'&&nowChar!='\r'){

//为空行前一个字符除去'\t','\r',' '的影响

nullString.setPreChar(nowChar);

}

/\*\*

\* 开始处理状态

\*/

switch (state) {

/\*\*

\* 状态0（初始状态）

\* define、函数标记状态

\* 若nowChar=='#'，进入define状态，状态标记开启

\* 若define状态标记开启，且前一字符为'\n'或'\\'，则关闭define状态标记；

记录nowChar为define的前一字符,break

\* 否则若nowChar=='/'，可能为单行注释或多行注释，转状态1；

\* 否则若nowChar=='\'' or '"'，则为字符或字符串，

\*用记录下当前字符，转状态6；

\* 否则若nowChar=='{'&&struct.getPreChar()==')'，则模拟函数括号栈变量++

\* 否则若nowChar=='}'&&simulationStack >0，则模拟函数括号里变量++，

\*若变量==0，则函数计数++

\* 最后记录struct.preChar为当前字符

\* 其它情况，状态不变。

\*/

case 0:

if(nowChar=='#'){

flagForDefine=1;

}

if(flagForDefine>0){

if(nowChar=='\n'&&define.getPreChar()!='\\')

flagForDefine=0;

define.setPreChar(nowChar);

break;

}

else if (nowChar == '/') {

state = 1;

} else if (nowChar == '\'' || nowChar == '"') {

annotation.setPreChar(nowChar);

state = 6;

} else if(nowChar=='{'&&struct.getPreChar()==')') {

simulationStack ++;

} else if(nowChar=='}'&&simulationStack >0) {

simulationStack --;

if(simulationStack ==0){

function.setSum(function.getSum()+1);

function.setCnt(function.getCnt()+1);

}

}

if(nowChar!=' '&&nowChar!='\n'&&nowChar!='\t') {

struct.setPreChar((char)nowChar);

}

break;

/\*\*

\*状态1

\* 单行或多行注释半状态

\* 若nowChar=='/'，则确定为单行注释{//}，转状态2；

\* 否则若nowChar=='\*'，则确定为多行注释{/\*}，转状态4；

\* 否则，转状态0。

\*/

case 1:

if(nowChar=='\n')annotation.setSum(annotation.getSum()+1);

if (nowChar == '/') {

state = 2;

} else if (nowChar == '\*') {

state = 4;

} else {

state = 0;

}

break;

/\*\*

\* 状态2

\* 单行注释开始

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='\n'，则单行注释结束，转状态0；

\* 否则若nowChar=='\\'，是单行注释行继续符，属于情况{//xxx\}，转状态3；

\* 否则，状态不变。

\*/

case 2:

if(nowChar=='\n')annotation.setSum(annotation.getSum()+1);

if (nowChar == '\\') {

state = 3;

} else if (nowChar == '\n') {

state = 0;

}

break;

/\*\*

\* 状态3

\* 单行注释行继续符开始

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='\\'，仍为单行注释行继续符，属于状态{//xxx\\}，状态不变；

\* 否则，属于状态{//xxx\a} 或 {//xxx\n}，转状态2。

\*/

case 3:

if(nowChar=='\n')annotation.setSum(annotation.getSum()+1);

if (nowChar != '\\') {

state = 2;

}

break;

/\*\*

\* 状态4

\* 多行注释开始

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='\*'，属于情况{/\*xxx\*}或{/\*\*}，转状态5；

\* 否则，属于情况{/\*x}或{/\*xxx\*aa}，状态不变。

\*/

case 4:

if(nowChar=='\n')annotation.setSum(annotation.getSum()+1);

if (nowChar == '\*') {

state = 5;

}

break;

/\*\*

\* 状态5

\* 多行注释可能结束

\* 若nowChar=='\n'，则注释计数++

\* 若nowChar=='/'，则多行注释结束，转状态0；

\* 否则若nowChar=='\*'，属于情况{/\*xxx\*\*}，状态不变；

\* 否则，属于情况{/\*xxx\*a}，转状态4。

\*/

case 5:

if(nowChar=='\n')annotation.setSum(annotation.getSum()+1);

if (nowChar == '/') {

state = 0;

annotation.setSum(annotation.getSum()+1);

} else if (nowChar != '\*') {

state = 4;

}

break;

/\*\*

\* 状态6

\* 字符或字符串状态

\* 若nowChar=='\\',为转义字符，属于状态{'\}或{"\}，转状态7

\* 否则，若nowChar==annotation.getPreChar(),说明字符或字符串状态结束，

转状态0；

\* 其它情况，状态不变。

\*/

case 6:

if (nowChar == '\\') {

state = 6;

} else if (nowChar == annotation.getPreChar()) {

state = 0;

}

break;

/\*\*

\* 状态7

\* 消除转义字符后面的字符的影响

\* 转状态6。

\*/

case 7:

state = 6;

break;

//case :8

//不将define单独列为状态是由于define后可以跟注释

/\*\*

\* 其他状态

\* 用于debug

\*/

default:

System.out.println(nowChar);

if(nowChar=='\n'&&simulationStack >0){

function.setSum(function.getSum()+1);

}

break;

}

}

/\*\*

\* 将统计结果放入容器

\*/

RESULTS.add(" 代码总行数: "+statistics.getSum());

RESULTS.add(" 函数总个数: "+function.getCnt());

RESULTS.add(" 函数总行数: "+function.getSum());

RESULTS.add(" 空行总行数: "+nullString.getSum());

RESULTS.add(" 注释总行数： "+annotation.getSum());

/\*\*

\* 调用assessService评分接口，将返回的评价结果放入容器中

\*/

RESULTS.addAll(assessService.assess(statistics.getSum(),function.getSum(),function.getCnt(),annotation.getSum(),nullString.getSum()));

/\*\*

\* 关闭流与文件

\*/

bufferedReader.close();

fileReader.close();

return RESULTS;

}

}

2 AssessServiceImpl.java，实现评级

package pers.B17040312.code\_analyzer.service.impl;

import pers.B17040312.code\_analyzer.service.AssessService;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* @title: AssessServiceImpl

\* @package pers.B17040312.code\_analyzer.service.impl

\* @description: AssessService接口的实现类

\* @author: laochonger

\* @date: 2019-10-29 16:20

\* @version: V1.0

\*/

public class AssessServiceImpl implements AssessService {

/\*\*

\* 储存对代码的评级结果

\*/

private List<String>RANKS = new ArrayList<>();

/\*\*

\*

\*@title: assessFunction

\*@description: 用于代码评级

\*@author: laochonger

\*@date: 2019-10-29 20:05

\*@param: [sumFunction, cntFunction]

\*@return: java.lang.String

\*@throws:

\*/

private String assessFunction(int sumFunction,int cntFunction){

double averagrFunction = (double)sumFunction/(double)cntFunction;

String rank = "A";

if(averagrFunction>10.0&&averagrFunction<=16.0){

rank = "A";

}else if((averagrFunction>8.0&&averagrFunction<=10.0)

||(averagrFunction>15.0&&averagrFunction<=21.0)){

rank = "B";

}else if((averagrFunction>5.0&&averagrFunction<8.0)

||(averagrFunction>21.0&&averagrFunction<=24.0)){

rank = "C";

}else{

rank = "D";

}

return rank;

}

/\*\*

\*

\*@title: assessAnnotation

\*@description: 用于注释评级

\*@author: laochonger

\*@date: 2019-10-29 20:06

\*@param: [sum, sumAnnotation]

\*@return: java.lang.String

\*@throws:

\*/

private String assessAnnotation(int sum,int sumAnnotation){

return getRank(sum, sumAnnotation);

}

/\*\*

\*

\*@title: assessNullString

\*@description: 用于空行评级

\*@author: laochonger

\*@date: 2019-10-29 20:06

\*@param: [sum, sumNullString]

\*@return: java.lang.String

\*@throws:

\*/

private String assessNullString(int sum,int sumNullString){

return getRank(sum, sumNullString);

}

/\*\*

\*

\*@title: getRank

\*@description: 空行与注释的共同评级函数

\*@author: laochonger

\*@date: 2019-10-29 21:14

\*@param: [sum, sumNullString]

\*@return: java.lang.String

\*@throws:

\*/

private String getRank(double sum, double sumNullString) {

double averagrFunction = sumNullString / sum \*100.0;

String rank = "A";

if(averagrFunction>15.0&&averagrFunction<=26.0){

rank = "A";

}else if((averagrFunction>10.0&&averagrFunction<=15.0)

||(averagrFunction>26.0&&averagrFunction<=31.0)){

rank = "B";

}else if((averagrFunction>5.0&&averagrFunction<10.0)

||(averagrFunction>31.0&&averagrFunction<=35.0)){

rank = "C";

}else{

rank = "D";

}

return rank;

}

/\*\*

\*

\*@title: assess

\*@description: 实现接口声明的函数，得到代码、注释以及空行的评级

\*@author: laochonger

\*@date: 2019-10-29 20:06

\*@param: [sum, sumFunction, cntFunction, sumAnnotation, sumNullString]

\*@return: java.util.List<java.lang.String>

\*@throws:

\*/

@Override

public List<String> assess(int sum, int sumFunction, int cntFunction, int sumAnnotation, int s umNullString) {

RANKS.add("\r\n");

RANKS.add(" 代码评级： "+assessFunction(sumFunction,cntFunction));

RANKS.add(" 注释评级： "+assessAnnotation(sum,sumAnnotation));

RANKS.add(" 空行评级： "+assessNullString(sum,sumNullString));

return RANKS;

}

}

**五、测试数据及其结果分析**

对算法功能、性能以及健壮性评测所使用的输入数据进行介绍和说明，并给出与输入数据相匹配的算法执行结果，并进行分析。

1给出输入数据test1.c

#include <stdio.h>

#include <string.h>

const int N = 1000010, INF = 1e9;

//struct SegmentTree {

// int lc, rc; // 左右子节点编号

// int sum;

//} tree[10000010];

//int n, m, t, tot,a[1000010], root[1000010];

int build(int l, int r) {

int p = ++tot; // 新建一个节点，编号为p，代表当前区间[l,r]

tree[p].sum = 0;

if (l == r) return p;

int mid = (l + r) >> 1;

tree[p].lc = build(l, mid);

tree[p].rc = build(mid + 1, r);

return p;

}

int insert(int now, int l, int r, int x, int delta) {

int p = ++tot;

tree[p] = tree[now]; // 新建一个副本

if (l == r) {

tree[p].sum += delta; // 在副本上修改

return p;

}

int mid = (l + r) >> 1;

if (x <= mid) tree[p].lc = insert(tree[now].lc, l, mid, x, delta);

else tree[p].rc = insert(tree[now].rc, mid + 1, r, x, delta);

tree[p].sum = tree[tree[p].lc].sum + tree[tree[p].rc].sum;

return p;

}

int ask(int p,int q,int l,int r,int L,int R) {

if(l>=L&&r<=R)return tree[p].sum - tree[q].sum;

int mid = (l+r)>>1;

int sum=0;

if(R<=mid)sum += ask(tree[p].lc,tree[q].lc,l,mid,L,R);

else if(L>mid)sum += ask(tree[p].rc,tree[q].rc,mid+1,r,L,R);

else sum+= ask(tree[p].rc,tree[q].rc,mid+1,r,L,R) + ask(tree[p].lc,tree[q].lc,l,mid,L,R);

//L R

return sum;

}

int lst[1000010];

int vis[1000010];

int b[1000010];

int main() {

int i=0;

scanf("%d", &n);

tot=0;

memset(root,0,sizeof(root));

memset(vis,0,sizeof(vis));

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

scanf("%d", &a[i]);

lst[i]=vis[a[i]];

b[i]=lst[i];

vis[a[i]]=i;

}

// sort(lst + 1, lst + n + 1); // 离散化

// t = unique(lst + 1, lst + n + 1) - (lst + 1);

root[0] = build(1, t); // 关于离散化后的值域建树

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

int x = 0;//lower\_bound(lst + 1, lst + t + 1, b[i]) - lst;

// 离散化后的值

root[i] = insert(root[i - 1], 1, t, x, 1); // 值为x的数增加1个

}

scanf("%d", &m);

for (i = 1; i <= m; i++) {

int l,r;

scanf("%d", &l);

scanf("%d", &r);

int x = 0;//lower\_bound(lst + 1, lst + t + 1, l-1) - lst;

if(lst[x]>l-1)--x;

if(x==t+1)--x;

printf("%lld", ask(root[r], root[l-1], 1, t, 1,x));

putchar('\n');

}

return 0;

}

预计代码总行数：80

预计函数总个数：4

预计函数总函数：65

预计空行总函数：4

预计注释总函数：16

代码评级：B

注释评级：A

空行评级：D

首先测试test1.c，得到结果图如下：



图9 test1.c测试结果图

验证成功

2将struct注释部分解除，测试struct部分的影响，存为test2.c，预期结果为：

预计代码总行数：80

预计函数总个数：4

预计函数总函数：65

预计空行总函数：4

预计注释总函数：12

代码评级：B

注释评级：B

空行评级：D

运行得到：



图10 test2.c测试结果图

验证成功

3加入#define f(x) fun(x){printf(“%d”, x);} (不计算在函数内)，测试#define对函数统计的影响，存为test3.c，预期结果为

预计代码总行数：81

预计函数总个数：4

预计函数总函数：65

预计空行总函数：4

预计注释总函数：12

代码评级：B

注释评级：B

空行评级：D

运行得到



图11 test3.c测试结果图

验证正确

4再加入

#define g(x) fun(x){printf("%d", x);\

\\\

\\\\\\

}

测试转义字符续行对程序的影响，存为test4.c，预期结果为

预计代码总行数：85

预计函数总个数：4

预计函数总函数：65

预计空行总函数：4

预计注释总函数：12

代码评级：B

注释评级：B

空行评级：D

运行得到



图12 test4.c测试答案图

验证成功

5将ask()函数用/\*注释，printf("%lld", ask(root[r], root[l-1], 1, t, 1,x));用//注释，测试/\*\*/注释，行数不变，存为test5.c，预期结果为

预计代码总行数：85

预计函数总个数：3

预计函数总函数：55

预计空行总函数：4

预计注释总函数：22

代码评级：B

注释评级：B

空行评级：D

运行得到



图13 test5.c测试答案图

验证成功

6在main函数中加入

printf("///\*\

\

fffff\*/");

测试字符串中的转义字符、注释字符对结果的影响，

得到test6.c，预期结果为

预计代码总行数：88

预计函数总个数：3

预计函数总函数：58

预计空行总函数：4

预计注释总函数：22

代码评级：B

注释评级：B

空行评级：D

运行得到：



图14 test6.c测试答案图

验证成功

7测试图形界面不合法操作

（1）选择文件夹时，预期输出提示“它只是一个文件夹啊”

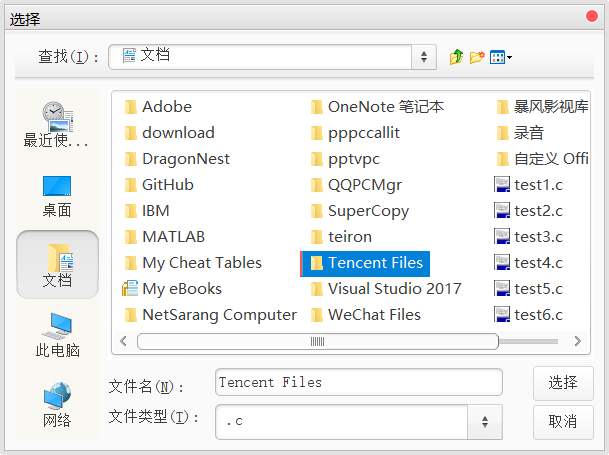


图15 打开文件图形界面图



图16 测试打开文件夹效果图

验证成功

（2）选择不存在的文件时，预期输出“大家就当无事发生”，

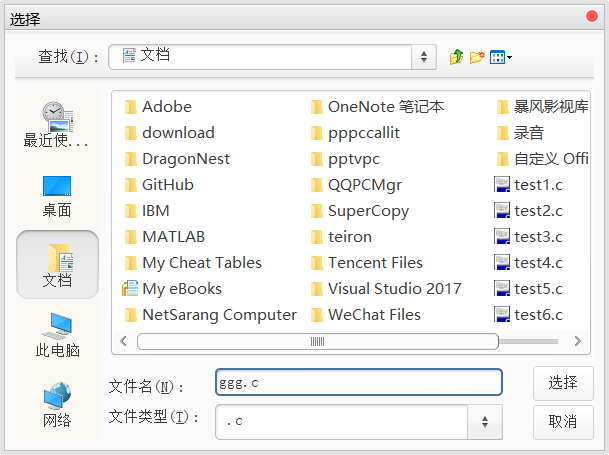


图17 打开不存在文件图



图18 测试打开不存在文件测试图

经过上述的测试，基本上可以确定了本程序的高鲁棒性，不会在输入错误以及特殊样例的情况下使得程序出错，但也不排除还有未被发现的问题。

**六、课题完成过程中遇到的问题及解决方法**

问题1：用单纯的if/else逻辑无法除去程序的所有bug

解决方法：换用有限状态自动机，该方法能够较完美地完成该程序的要求

问题2：统计分析C源代码时，出现了C源代码中#define后接’\’导入程序结果出错

解决方法：对#define单独添加一个状态标记

问题3：统计分析C源代码时，出现了由于Struct使得函数统计出错的情况

解决方法：为Struct单独添加判断逻辑

问题4：统计分析C源代码时，出现了空行统计出错的问题

解决方法：在Java中读入C源代码时，换行以’\r\n’表示，更换了判断条件后问题得以解决

问题5：无法理清注释与字符/字符串之间的逻辑关系

解决方法：画出状态转移图，设计有限状态自动机

问题6：输出时评级结果部分不正确

解决方法：打log标记，发现是向接口中的方法传参时写反，改正后问题解决

问题7：分析多个C源文件后，输出时会输出之前的记录

解决方法：发现是将接口实现类中的容器定义为了静态变量，去掉”static”后，问题解决

问题8：打开不存在的文件时程序出错

解决方法：在打开文件前进行判断，并捕获异常

问题9：打开文件选择界面时发现不仅仅显示了C源代码文件

解决方法：为文件选择器设置过滤器

**七、总结**

如何做一个合格的程序员？这是我大学以来一直在探究的问题。

经历了为期两周的程序设计周，虽然累，但也乐在其中。事实上也是如此，用了一个下午就写完了demo。但是测试demo的时候，发现了无数的BUG，可谓是代码5分钟，BUG两小时。在经过了慎重的思考后，我觉得放弃这版代码，重新审视题目，寻找合适的解法。为什么bug多？无非是因为我只用了简单的逻辑，而简单的逻辑恰恰无法完全满足题目的要求。想了一个下午，没有想出合适的数据结果或是算法来解决这个看似简单的题目。于是转念一想，编译器又是如何识别到函数、注释的呢？于是上网搜索，发现了“有限状态自动机”可以较完美地解决这个题目。于是我花了两天的时间，进一步了解了“有限状态自动机”的思想，并画出状态转移图，设计出了本题的“有限状态自动机”，画了半天的时间调试，最终通过测试。

大一的时候其实也有一次程序设计周，但是那次我并没有写出图形界面，也算是遗憾之一了。为了不让自己继续遗憾，我用Java重构了代码，写了图形界面，说实话，虽然只是简单的图形界面，但是写完的那一刻，成就感还是满满的。

这次编程，我特意按照规范命名、分包。虽然我还是学生，但是我认为规范地编码是十分必要的。当然，在编程的过程中出现了很多问题，比如某个接口的参数，所以需要不断地查文档等待。后来发现，这样思路容易被打断，不如用伪代码标记一下，这部分的逻辑写完再补充细节。

实际上，本次编程中我有其他的许多想法，比如让这个代码分析器能够分析Java以及C++等语言源程序、先测试这个源程序是否合法等等，但是由于时间紧迫，没有来得及实现。

如何才能做一个合格的程序员？这两周的程序设计任务，似乎让我明白了，要做一个合格的程序员，无非是需要工具、经验、天马行空的想法以及认真钻研的精神。