C++第二次作业解答

一、 综述。

1、 作业内容。

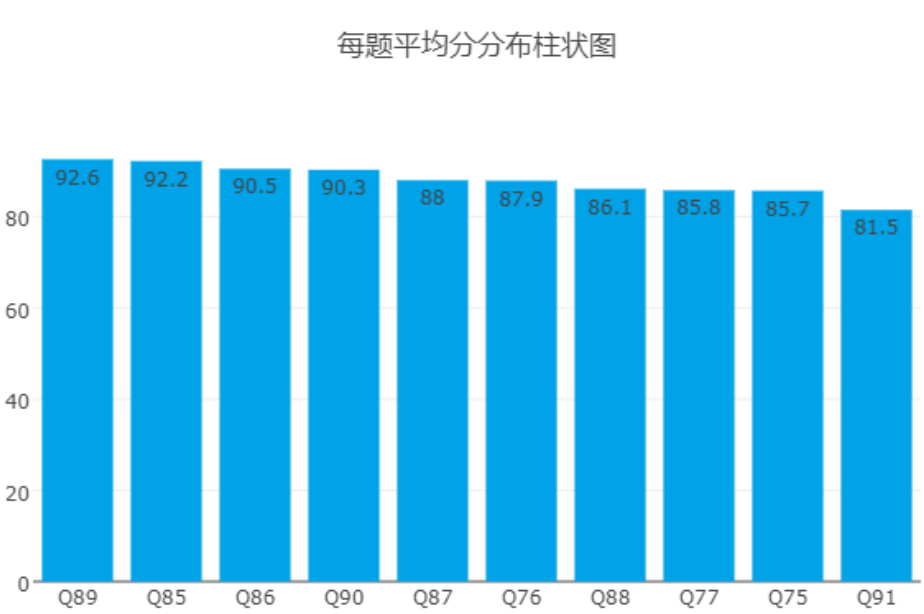
总共 10 道题，时间为 9 天。

2、 作业考察点。

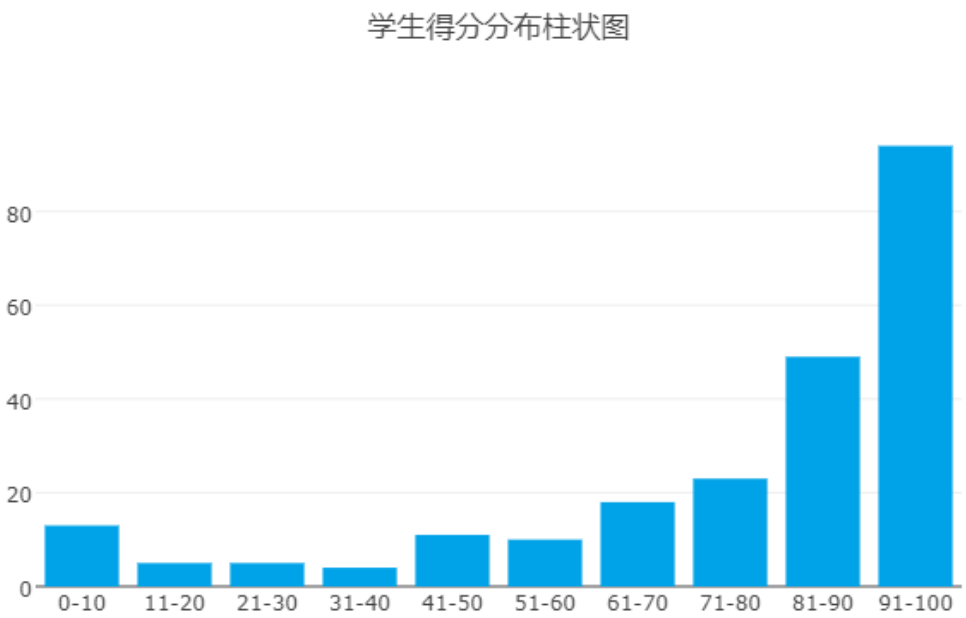
- 控制台 IO（cin/cout）
- 数组与数组操作
- 结构体
- 基本数据结构（链表、队列、栈、树）
- 指针操作

3、 作业情况。

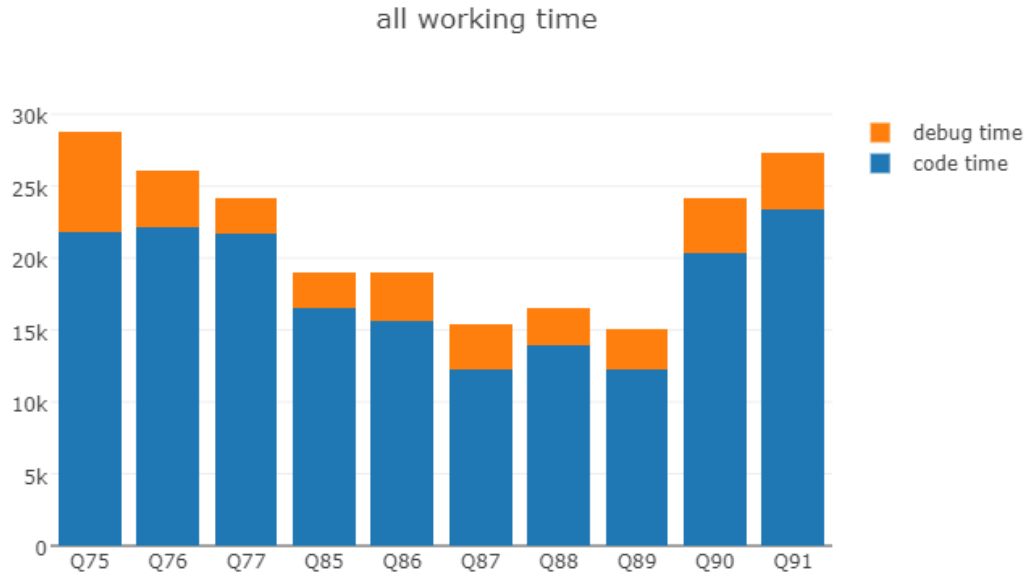
- 题目平均分



- 得分情况分布



- 编码时间分布



4、 特殊情况。

- Visual Studio 崩溃

初步解决方案： 断网 -> 打开 VS -> 退出 outlook 账户 -> 重新联网，打开 VS

二、 Q75。

1、 题目要求。

输入二叉树，输出该树的中序序列。

二叉树的输入使用二叉树的数组形式表示，在输入中如果节点为空，则使用字符串“null”表示。如二叉树：

aa

bb cc

dd ee ff

使用如下输入表示：

aa bb cc dd null ee ff

注意：输入在同一行，不会有第二行输入

例 1：

输入：

1 2 3

输出:

2 1 3(没有换行符)

2、 考察点。

- 二叉树的理解与构造
- 结构体的使用
- 指针操作
- 对二叉树进行中序遍历（递归 / 非递归）

3、 程序思路。

- 二叉树结构体的构造：

```
struct Tree{
    Tree* left;    //左节点指针
    Tree* right;   //右节点指针
    string val;    //值
    Tree(string& val1){
        left = NULL;
        right = NULL;
        val = val1;
    }
}*PTree;
```

- 二叉树的中序遍历（递归算法）：

```
void print_middle(Tree* root){
    if (root->left != NULL){
        print_middle(root->left);
    }
    if (root->val!="null"){
        if (!isFirst){
            cout << " ";
        }
        isFirst = 0;
         cout << root->val;
     }
     if (root->right != NULL){
         print_middle(root->right);
     }

}
```

三、 Q76。

1、 题目要求。

用单链表模拟大整数加法运算。输入两个非空的链表表示两个非负整数。链表的表头表示正整数的高位，每一个节点的值是 0-9 的整数，模拟加法的结果如下：

9>9>9>null + 1>null = 1>0>0>0>null

链表的输入用如下形式，

9>9>9>null

需要考虑形式的输入：

0>0>1>null

例 1：

输入：

9>null

1>null

输出：

1>0>null

例 2：

输入：

1>2>3>null

2>4>null

输出：

1>4>7>null

2、 考察点。

- 指针操作
- 链表的构造
- 结构体
- 对链表进行运算
- 进位的处理方法

3、 程序思路。

- 构造链表的结构体，本题可以构造双向链表

```
struct LinkList{
    LinkList* pre;   //链表前一结点
    LinkList* next;  //链表后一结点
    int val;  //结点的值
    LinkList(int val1){
        val = val1;
        pre = NULL;
        next = NULL;
    }
};
```

- 输入字符串的处理

```
while ((c = cin.get()) != 'n'){
}  //循环的设定
```

- 进位的处理

```
if (new_val > 9){
    plus = 1;
}  //plus 值为 1 则说明有进位
else{
    plus = 0;
}  //plus 值为 0 则说明无进位
```

四、 Q77。

1、 题目要求。

使用栈判断给定的代码片段中的 TAG 是否符合规。

规范如下:

1.代码必须由有效的 TAG 对包裹

2.有效 TAG 对形式：

<TAG_NAME>TAG_CONTENT</TAG_NAME>.其中，

<TAG_NAME>与</TAG_NAME>是起始 TAG 与结束 TAG，两个 TAG 的 TAG_NAME 必须一样。

3.TAG_NAME 必须是大写字母并且长度为 1-9。

4.TAG_CONTENT 可能含有其他的 TAG 对和<!-- -->、<>、</>、()符号对，且其中所有的<!-- -->、<>、</>、()、TAG 都必须成对，否则判定不符合规范

5.TAG_CONTENT 中可能含有<!--COMMENT-->形式的字符串，其中 COMMENT 内容可以为任意字符，包括不成对的 TAG、<>、</>、()等。（即 COMMENT 内容可以忽略，不影响判断）

例 1：

输入：

<DIV>Hello World!</DIV>

输出:

True(没有换行符)

例 2：

输入:

<DIV>Hello World!

输出:

False(没有换行符)

例 3：

输入：

<DIV>Hello World!<!--example<div>--></DIV>

输出：

True(没有换行符)

例 4:

输入：

<DIV>Hello World! < (</ </DIV>

输出：

False(没有换行符)

2、 考察点。

- 数组的处理
- 栈数据结构的建立与理解

3、 程序思路。

- 栈数据结构的建立

```
#include <stack>
stack<string> ss;
```

- 栈数据结构的操作

```
if (line[i] == '('){
    ss.push("(");
}

if (line[i] == ')'){
    top = ss.top();
    if (top == "("){
        ss.pop();
        continue;
    }
    else{
        isVaild = false;
        break;
    }
}
```

五、 Q85。

1、 题目要求。

输入一个数字数组 nums，长度为 n（n<100）。定义 4 种命令：replace、move、translate，print。

replace 命令将数组中所有的的数字 a 替换为数字 b，格式为 replace a b,a 为被替换的数字，b 为替换的数字。

move 命令将指定数字 a 全部移动到数组开头或末尾，命令格式为 move head(tail) a，a 为被移动的数字。

translate 命令将数组整体向前（或后）平移，尾部越界的数字补全到数组开头，开头越界的数字补全到数组末尾，命令格式为 translate head(tail) a，a 为移动的格数。

print 命令要求输出当前的数组信息。

首先输入 n，表示数组大小，接着 n 个输入为具体数组。然后输入 m ,表示命令数量 ,接着输入 m 条命令。将命令的结果进行输出。

Input：

5

0 1 0 3 12

7

print

replace 1 3

print

move tail 3

print

translate head 2

print

Output：

0 1 0 3 12

0 3 0 3 12

0 0 12 3 3

12 3 3 0 0

2、 考察点。

- 数组的建立与操作
- 数组的基本算法
- 命令的输入

3、 程序思路。

- replace 函数功能的实现

```
1.       int* replace(int*num,int len,int oldValue,int newValue){
2.      //对数组进行循环
3.      for (int i = 0; i < len;i++){
4.          if (num[i]==oldValue){
5.              num[i] = newValue;
6.          }//用新的值替换旧的值
7.      }
8.      return num;//返回数组
9.  }
```

- move 函数功能的实现

```cpp
//move 到数组头部功能函数
int* moveFront(int*num,int location){
    int value = num[location];
    for (int i = location; i >=1;i--){
        num[i] = num[i - 1];
    }
    num[0] = value;
    return num;
}
```

- translate 函数功能的实现

```cpp
//translate 到数组尾部功能函数
int* translateTail(int*num, int shift, int len){
    int *newNum = new int[len];//创建一个新数组用于存储
    for (int i = 0; i < len;i++){
        newNum[(i+shift)%len] = num[i];
    }
    //循环算法
    delete num;//释放数组空间
    return newNum;//返回 newNum 数组
}
```

- print 函数功能的实现

```cpp
//输出数组函数
void printF(int*num,int length){
    //循环体输出数组
    for (int i = 0; i < length;i++){
        cout << num[i];
        if (i!=length-1){
            cout << " ";//各个数之间打印空格
        }
    }
}
```

六、 Q86。

1、 题目要求。

输入一个矩阵 m n，并逐个输入 m*n 个数。接着给定一个坐标 x，以螺旋的方式取得对应位置的数。默认取数坐标起始为 0

对矩阵

1 2 3

4 5 6

7 8 9

来说，螺旋的顺序为 1 2 3 6 9 8 7 4 5。

Input:

3 3

1 2 3

4 5 6

7 8 9

5

Output:

8

2、 考察点。

- 数组的使用
- 动态规划算法的入门

3、 程序思路。

- 数组的处理

```
//数组的处理
for (; i<m - n + 1 + j; i++) {
    num[k] = matrix[i][j];
    k++;
    if (k == m*n)
        break;

    if (i == m - n + j) {
        j--;
        l = 3;
        break;
    }
}
```

七、 Q87。

1、 题目要求。

给定一个三角数组，形式如下。三角中每个数字代表权值。从顶点开始，到底部任意一点结束为一条路径。路径只能向左下或右下的点延伸。找到一条权值和最少路径，输出路径的最小权值。

[

[1],

[1,2],

[1,2,3],

[1,2,3,4]

]

Input： n

a11

a21 a22

a31 a32 a33

……

an1 an2 …… ann

Output：sum_value

Input：

3

1

2 3

1 6 3

Output：

4

2、 考察点。

- 数组的使用
- 动态规划算法的入门

3、 程序思路。

- 动态规划算法

```
1.  //动态规划函数
2.  int sum(int**data,int **value,int x,int y,int len){
3.      //cout << " x: " <<x<< " y" <<y<< " data[x][y]" << data[x][y] << "  valu
    e[x][y]"<<value[x][y]<< endl;
4.      if (value[x][y]!=0){
5.          return value[x][y];
6.      }
7.      else if (x==0&&y==0){
8.          value[x][y] = data[x][y];
9.          return value[x][y];
10.     }
11.     else if (x==y){//在矩阵的对角线上,只能往左上走
12.         value[x][y] = sum(data,value,x-1,y-1,len)+data[x][y];
13.     }
14.     else if (x !=0 && y==0){//最左边一个，只能往右上走
15.         value[x][y] = sum(data, value, x-1, y, len)+data[x][y];
16.     }
17.     else{
18.         //取左上与右上值小的
19.         int left = sum(data, value, x - 1, y-1, len);//左上
20.         int right = sum(data, value, x-1, y , len);//右上
21.         value[x][y] =(left > right ? right : left) + data[x][y];
22.     }
23.     return value[x][y];
24. }
```

- 数组的建立

```
//申请空间
int ** a = new int *[row];
int **value = new int*[row];
for (int i = 0; i < row; i++){
    a[i] = new int[column];
    value[i] = new int[column];
}
//初始化三角
for (int i = 0; i < row;i++){
    for (int j = 0; j < i + 1;j++){
        cin >> a[i][j];
    }
}
```

八、 Q88。

1、 题目要求。

输入 n 个字符串，字符串由小写字母与大写字母组成。利用 sort 函数，对字符串进行排序，并按顺序输出。排序规则：(1)短字符串排在长字符串之前(2)长度相同的情况下按字母顺序排序，字母顺序为 aAbBcCdD...xXyYzZ

示例：

Input: 5

abC

Abd

b

bcd

c

Output：b

c

abC

Abd

bcd(无换行)

2、 考察点。

- 字符串的处理
- 控制台的输入输出

3、 程序思路。

- 字符串长度相同时候的处理

```
for (int i = 0; i < str1.length();i++){
    if (str1[i]==str2[i]){
        continue;
    }
    else{
        if ((str1[i] <= 90 && str2[i] <= 90) || (str1[i] >=97 && str2[i] >=97)){
            if (str1[i]<str2[i]){
                return 1;
            }
            else{
                return -1;
            }
        }
        int v1, v2;
        v1 = str1[i],v2=str2[i];
        if (str1[i]<=90){//为大写字母
            v1 = str1[i] + 32;
        }
        if (str2[i] <= 90){//为大写字母
            v2 = str2[i] + 32;
        }
```

九、 Q89。

1、 题目要求。

描述：给定一个二叉树，每一个二叉树的节点的值都是一个字符串，给定节点

node1 和 node2，求节点 node1 和 node2 的路径长度，如果 node1 和 node2 不是从根

节点出发的同一条路径上的节点则返回-1.此处一个子节点和父节点之间的路径长

度定位 1.二叉树的输入使用二叉树的数组形式表示，在输入中如果表示 null（空指针），则使用字符串“NULL”表示。如二叉树

aa

bb cc

dd ee ff

gg hh i j k l

使用数组表示：

aa bb cc dd NULL ee ff gg hh i j k l

（注意这棵树中 bb 没有右节点）

在输入的时候，对于 bb 的右节点，因为没有子节点，则输入“NULL”表示。

输入：

step1：二叉树的节点个数（满二叉树，其中非叶节点空节点

也算进去）m，

step2：然后依次输入 m 个字符串，如果节点表示为 null，使用字符串“NULL”表示。

step3：节点 node1 的值（不会为“NULL”值）

step4：节点 node2 的值（不会为“NULL”值）

输出：

节点 node1 和节点 node2 之间的路径长度（如果两个节点不在从根节点出发的一条

路径上，则返回-1）

注：此处的“NULL”在输入和输出时都表示字符串

示例 1：

输入：

7

aa bb cc dd NULL ee ff

aa

ee

输出：

2（没有回车）

示例 2：

输入：

7

aa bb cc dd NULL ee ff

bb

ee

输出：

-1（没有回车）

2、 考察点。

- 二叉树的生成
- 链表的结构生成
- 结构体的构造

3、 程序思路。

- 二叉树的结构体的构造

```
//二叉树结构体
struct TreeNode
{
    string val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(string x) :val(x), left(NULL), right(NULL) {}
};
```

- 二叉树的计算

```
//计算长度
int calculateLen(TreeNode *r, string first, string second) {
    if (first.compare(second) == 0) {
        return 0;
    }
    string v = r->val;
    if (v.compare(first) == 0) {
        int res = findChild(r, second, 0);
        return res;
    }
    if (v.compare(second) == 0) {
        int res = findChild(r, first, 0);
```

```
13.         return res;
14.     }
15.     //cannot find
16.     int lres = -1;
17.     int rres = -1;
18.     if (r->left != NULL) {
19.         lres = calculateLen(r->left, first, second);
20.     }
21.     if (r->right != NULL) {
22.         rres = calculateLen(r->right, first, second);
23.     }
24.     return (lres >= rres ? lres : rres);
25. }
26. //查找孩子
27. int findChild(TreeNode *r, string cmp, int len) {
28.     string v = r->val;
29.     if (v.compare(cmp) == 0) {
30.         return len;
31.     }
32.     else {
33.         int l = len + 1;
34.         int leftres = -1;
35.         int rightres = -1;
36.         if (r->left != NULL) {
37.             leftres = findChild(r->left, cmp, l);
38.         }
39.         if (r->right != NULL) {
40.             rightres = findChild(r->right, cmp, l);
41.         }
42.         if (leftres == -1 && rightres == -1) {
43.             return -1;
44.         }
45.         else {
46.             return (leftres >= rightres ? leftres : rightres);
47.         }
48.     }
49.     return 0;
50. }
```

十、 Q90。

1、 题目要求。

自定义一个链表的结构，不能使用已有库

定义一些链表的操作 ADD ,REVERSE ,REMOVEREPEAT ,DEL，PRINT，SIZE

ADD：往链表追加一个元素，如果之前一个元素都没有，则该元素作为链表的第一个元素

如： ADD 1，将 1 添加至链表

REVERSE：将链表反转

REMOVEREPEAT：剔除链表中重复的元素，保留一个，如 REMOVEREPEAT n，则剔除链表中所有值为 n 的元素，保留最后一个,如果没有该元素，则不操作

DEL：删除链表中的所有指定的元素，如 DEL 1，删除链表中值为 1 的所有元素，不存在元素则不进行任何操作

PRINT：将链表从第一个依次输出,如果为空，则输出“NULL”

SIZE：输出链表的元素个数

输入：

先输入整数 n，表示将会有 n 条指令输入

然后依次输入 n 条指令（指令只会在给定的指令范围内）

输出：

根据指令的要求，如果指令要求输出，则将结果输出到控制台（注

意最后一次输出没有回车）

注：此处的“NULL”输出时都表示字符串

示例：

输入

12

ADD 1

ADD 1

ADD 2

ADD 1

SIZE

PRINT

REMOVEREPEAT 1

PRINT

ADD 1

PRINT

DEL 1

PRINT

输出：

4

1 1 2 1

2 1

2 1 1

2（没有回车）

2、 考察点。

- 链表的构造
- 链表的操作
- 结构体的建立

3、 程序思路。

- 链表结构体的构造

```
struct linkedList
{
    string val;
    linkedList *next;
    linkedList(string x) :val(x), next(NULL){}
};
```

- 链表的添加

```
//添加数字功能算法
linkedList* add(linkedList *l, string v) {
    linkedList *t = l;
    if (t == NULL) {
        t = new linkedList(v);
        t->next = NULL;
        return t;
    }
    else {
        linkedList *h = t;
        while (t->next != NULL) {
            t = t->next;
        }
        t->next = new linkedList(v);
        return h;
    }
}
```

- 链表的翻转

```
//翻转链表功能的算法
linkedList* reverse(linkedList *l) {
    if (l == NULL){
        return NULL;
    }
    if (l->next == NULL){
        return l;
    }
    if (l->next->next == NULL){
        linkedList *tmp = l->next;
        l->next = NULL;
        tmp->next = l;
        return tmp;
    }
    linkedList *p = l;
    linkedList *q = l->next;
    linkedList *r = NULL;
    p->next = NULL;
    while (q->next != NULL) {
        r = q->next;
        q->next = p;
        p = q;
        q = r;
    }
    q->next = p;
    l = q;
    return l;
}
```

- 链表的删除

```
//删除重复数据的方法
linkedList* removerepeat(linkedList *l, string v) {
    linkedList *tmp = l;
    linkedList *t = l;
    linkedList *pre = NULL;
    int count = 0;
    while (tmp != NULL) {
        if (tmp->val.compare(v) == 0) {
            count++;
        }
```

```
        tmp = tmp->next;
    }
    while (t != NULL) {
        if (t->val.compare(v) == 0) {
            if (pre == NULL) {
                if (count != 1) {
                    l = l->next;
                    t = t->next;
                    count--;
                }
                else {
                    pre = t;
                    t = t->next;
                }
            }
            else {
                if (count != 1) {
                    pre->next = t->next;
                    t = t->next;
                    count--;
                }
                else {
                    pre = t;
                    t = t->next;
                }
            }
        }
        else {
            pre = t;
            t = t->next;
        }
    }
    return l;
}
```

十一、Q91。

1、 题目要求。

输入学生信息（包括姓名，学号，科目，成绩），学号为唯一，成绩可能发生更新，若信息中缺少某个学生的某个科目，则将该学生的该科目得分计算为 0。

将平均得分小于等于总体平均分的学生按平均分从高到低进行排序,并输出学号。同平均分学生按学号排序。

Input： n

name，id，subject，score

……

Output：id1

id2

id3(无换行)

示例：

Input: 5

student1 1 cpp 70

student1 1 java 80

student1 1 cpp 80

student2 2 cpp 70

student3 3 java 90

Output：3

2(无换行)

2、 解题思路。

- 结构体的构造
- 排序算法

3、 程序思路。

- 结构体的构造

```
//学生信息结构体
struct info
{
    string name;
    int id;
    string subject;
    int score;
    info(string n, int i, string s, int sc) :name(n), id(i), subject(s), score(sc){}
};
//排序结构体
struct sort
{
    int id;
    double mean;
    sort(int i, double m) :id(i), mean(m){}
};
```

- 排序算法

```
//sort
for (int i = 0; i < sort_len - 1; i++) {
    for (int j = 0; j < sort_len - i - 1; j++) {
        if (sortlist[j]->mean < sortlist[j + 1]->mean) {
            sort *tmp = sortlist[j];
            sortlist[j] = sortlist[j + 1];
            sortlist[j + 1] = tmp;
        }
        else if (sortlist[j]->mean == sortlist[j]->mean) {
            if (sortlist[j]->id < sortlist[j + 1]->id) {
                sort *tmp = sortlist[j];
                sortlist[j] = sortlist[j + 1];
                sortlist[j + 1] = tmp;
```

```
            }
        }
    }
}
```