استاد: **رامتین خسروی** طراحان: **بردیا اقبالی**، **امیرحسین حبیب‌وند**، **ژیوار صورتی**

مهلت تحویل: شنبه ۲۵ آبان ۱۳۹۸، ساعت ۲۳:۵۵

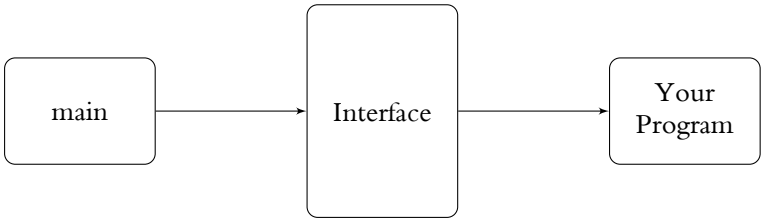
این تمرین اولین تلاش شما برای طراحی شئ‌گرای یک سیستم است. از آنجایی که استفاده از مفاهیم شئ‌گرایی در پیاده‌سازی سایر تمارین درس لازم است، پیشنهاد می‌شود به این پروژه زمان کافی اختصاص دهید.

۱ UTRELLO

یکی از مهم‌ترین عوامل برای پیش‌برد مؤثر یک پروژه مدیریت درست وظایف در تیم اجرایی است. در این تمرین قصد راه‌اندازی یک سیستم برای مدیریت وظایف و همینطور کنترل آن‌ها را داریم. این سسیتم از کاربرها، لیست‌ها و وظیفه‌ها تشکیل شده است. هر وظیفه متعلق به یک لیست است و می‌تواند به یک کاربر واگذار گردد. علاوه بر این برای هر وظیفه اولویت و زمان تخمینی آن در نظر گرفته می‌شود، تا در تصمیم‌گیری برای واگذاری و انجام وظیفه‌ها بهتر عمل کنیم.

توجه: در این تمرین نیازی به پیاده‌سازی تابع main نیست. برنامه شما باید با فایل‌های تابع main آماده شده توسط دستیاران آموزشی قابل اجرا باشد. برای این منظور شما باید توابع کلاس رابط[1] که در کنار صورت پروژه قرار گرفته است را پیاده کنید.

حالت کلی قرار گیری این کلاس به صورت زیر است. در تابع main یک شئ از کلاس رابط ایجاد می‌شود و در ادامه توابع این کلاس رابط صدا زده می‌شوند. توجه کنید که جزییات پیاده‌سازی شما نباید در کلاس رابط باشد. در صورت لزوم می‌توانید به کلاس رابط فیلدها و توابع مورد نیاز خود را اضافه کنید. توجه کنید که به هیچ‌وجه نباید امضای توابع از پیش تعیین شده کلاس رابط را تغییر دهید.



۱.۱ کلاس رابط

به معرفی هر کدام از توابع کلاس رابط می‌پردازیم:

۱.۱.۱ سازنده

این تابع زمانی که یک شئ از کلاس رابط می‌سازیم فراخوانی می‌شود.

امضا

```
UTrelloInterface::UTrelloInterface()
```

نمونه فراخوانی

```
UTrelloInterface interface;
```

[1] interface class

۲.۱.۱ توابع اضافه کردن

از این توابع برای اضافه کردن موجودیت[۲] به سیستم استفاده می‌کنیم. توابع این بخش در صورت موفقیت رشته Success را برمی‌گردانند. خطاهای احتمالی هر تابع در توضیحات تابع آمده است.

۱.۲.۱.۱ کاربر: از این تابع برای اضافه کردن کاربر به سیستم استفاده می‌کنیم. دقت کنید که اسم هر کاربر در سیستم رشته‌ای یکتاست. در صورتی که اسم تکراری برای کاربر وارد شده باشد، رشته User already exists را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::addUser(std::string username)
```

نمونه فراخوانی
```
interface.addUser("Zhivar");
```

نمونه خروجی
```
Success
User already exists
```

۲.۲.۱.۱ لیست: از این تابع برای اضافه کردن لیست به سیستم استفاده می‌کنیم. دقت کنید که اسم هر لیست در سیستم رشته‌ای یکتاست. در صورتی که اسم تکراری برای لیست وارد شده باشد، رشته List already exists را برمی‌گرداند.

امضا
```
std:string UTrelloInterface::addList(std::string name)
```

نمونه فراخوانی
```
interface.addList("Code");
```

نمونه خروجی
```
Success
List already exists
```

۳.۲.۱.۱ وظیفه: از این تابع برای اضافه کردن وظیفه به یک لیست استفاده می‌کنیم. نام هر وظیفه در سیستم رشته‌ای یکتاست. زمان تقریبی و اولویت هر کدام یک عدد مثبت هستند. وظیفه با عدد اولویت بزرگ‌تر اولویت بالاتری دارد. وظیفه در حالت انجام‌نشده به سیستم اضافه می‌شود. همینطور وظیفه تازه ایجاد شده به کسی واگذار نشده است.

در صورتی که لیستی با نام وارد شده در سیستم موجود نباشد، رشته List does not exist و در صورتی که نام وظیفه تکراری باشد رشته Task already exists را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::addTask(std::string list, std::string name,
    unsigned int estimatedTime, unsigned int priority, std::string description)
```

[2] entity

نمونه فراخوانی
```
interface.addTask("Code", "Do Everything", 12, 1, "Write the whole code");
```

نمونه خروجی
```
Success
Task already exists
List does not exist
```

۳.۱.۱ توابع حذف

از این توابع برای حذف موجودیت‌های سیستم استفاده می‌کنیم. توابع این بخش هم در صورت موفقیت رشته Success را برمی‌گردانند. خطاهای احتمالی هر تابع در توضیحات تابع آمده است.

۱.۳.۱.۱ لیست: از این تابع برای حذف لیست استفاده می‌کنیم. در صورتی که لیست مورد نظر در سیستم موجود نباشد، رشته List does not exist را برمی‌گرداند. توجه کنید که با حذف لیست، وظیفه‌های لیست هم حذف می‌شوند.

امضا
```
std::string UTrelloInterface::deleteList(std::string list)
```

نمونه فراخوانی
```
interface.deleteList("Code");
```

نمونه خروجی
```
Success
List does not exist
```

۲.۳.۱.۱ وظیفه: از این تابع برای حذف وظیفه استفاده می‌کنیم. در صورتی که وظیفه مورد نظر در سیستم موجود نباشد، رشته Task does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::deleteTask(std::string task)
```

نمونه فراخوانی
```
interface.deleteTask("Upload Assignment");
```

نمونه خروجی
```
Success
Task does not exist
```

۴.۱.۱ توابع مربوط به وظیفه‌ها

از این توابع برای ایجاد تغییرات روی وظیفه‌ها استفاده می‌کنیم. توابع این بخش هم در صورت موفقیت رشته Success را برمی‌گردانند. همچنین در صورتی که وظیفه‌ای با نام وارد شده وجود نداشته باشد رشته Task does not exist را برمی‌گردانند. باقی خطاهای احتمالی هر تابع در توضیحات تابع آمده است.

۱.۴.۱.۱ واگذار کردن: با استفاده از این تابع می‌توانیم یک وظیفه را به یک کاربر واگذار کنیم. در صورتی که کاربر مورد نظر در سیستم موجود نباشد، رشته User does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::assignTask(std::string task, std::string user)
```

نمونه فراخوانی
```
interface.assignTask("Do Everything", "Bardia");
```

نمونه خروجی
```
Success
Task does not exist
User does not exist
```

۲.۴.۱.۱ ویرایش: با استفاده از این تابع می‌توانیم مواردی که در زمان ساخت برای وظیفه تعیین کرده‌ایم را تغییر دهیم.

امضا
```
std::string UTrelloInterface::editTask(std::string task, unsigned int estimatedTime,
    unsigned int priority, std::string description)
```

نمونه فراخوانی
```
interface.editTask("Do Everything", 12, 10, "Write the whole code");
```

نمونه خروجی
```
Success
Task does not exist
```

۳.۴.۱.۱ انتقال: با استفاده از این تابع می‌توانیم یک وظیفه را به یک لیست دیگر منتقل کنیم. در صورتی که لیست مقصد در سیستم موجود نباشد، رشته List does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::moveTask(std::string task, std::string list)
```

نمونه فراخوانی
```
interface.moveTask("Have fun", "Code");
```

نمونه خروجی
```
Success
List does not exist
Task does not exist
```

۴.۴.۱.۱ انجام: با استفاده از این تابع می‌توانیم یک وظیفه را به حالت انجام‌شده ببریم.

امضا
```
std::string UTrelloInterface::completeTask(std::string task)
```

نمونه فراخوانی
```
interface.completeTask("Do Everything");
```

نمونه خروجی
```
Success
Task does not exist
```

۵.۱.۱ توابع گزارش‌گیری

از این توابع برای گزارش‌گیری از سیستم استفاده می‌کنیم. مقدار بازگشتی این توابع یک عدد صحیح مثبت می‌باشد.

شما باید برای این توابع آزمون واحد[۳] بنویسید و درستی کار آن‌ها را بسنجید.

۱.۵.۱.۱ کل کار: این تابع زمان تخمینی اتمام همه وظیفه‌ها را برمی‌گرداند. که این مقدار برابر با بیشینه[۴] زمان مورد نیاز توسط کاربران برای انجام وظایفشان است. زمانی که کاربران برای انجام وظایفشان نیاز دارند، برابر با مجموع زمان تخمینی همه‌ی وظیفه‌های واگذار شده به آن کاربر است. دقت کنید که این تابع وظایف واگذار نشده را در نظر نمی‌گیرد.

امضا
```
int UTrelloInterface::printTotalEstimatedTime()
```

نمونه فراخوانی
```
interface.printTotalEstimatedTime();
```

نمونه خروجی
```
12
0
```

۲.۵.۱.۱ کار باقی‌مانده: این تابع زمان تخمینی اتمام همه وظیفه‌های انجام‌نشده را برمی‌گرداند. که این مقدار برابر با بیشینه[۵] زمان مورد نیاز توسط کاربران برای انجام وظایف باقی‌مانده‌شان است. زمانی که کاربران برای انجام وظایف باقی‌مانده‌شان نیاز دارند، برابر با مجموع زمان تخمینی همه‌ی وظیفه‌های واگذار شده به آن کاربر است که انجام‌نشده. دقت کنید که این تابع وظایف واگذار نشده را در نظر نمی‌گیرد.

امضا
```
int UTrelloInterface::printTotalRemainingTime()
```

[3] Unit test
[4] maximum
[5] maximum

نمونه فراخوانی
```
interface.printTotalRemainingTime();
```

نمونه خروجی
```
15
0
```

۳.۵.۱.۱ حجم کار کاربر: این تابع مجموع زمان همه وظیفه‌های یک کاربر در همه لیست‌ها را برمی‌گرداند. در صورتی که کاربر مورد نظر در سیستم موجود نباشد، عدد ۰ را برمی‌گرداند.

امضا
```
int UTrelloInterface::printUserWorkload(std::string user)
```

نمونه فراخوانی
```
interface.printUserWorkload("Amir");
```

نمونه خروجی
```
8
0
```

۶.۱.۱ توابع خروجی

توابع این بخش جزییات موجودیت‌های سیستم را برمی‌گردانند.

۱.۶.۱.۱ وظیفه: این تابع جزییات وظیفه مورد نظر را در قالب زیر برمی‌گرداند.

عبارات داخل {{}} با فیلدهای وظیفه جایگزین می‌شوند و بقیه عبارات برای هر وظیفه ثابت می‌باشند. در صورتی که وظیفه به کسی واگذار شده نام آن کاربر در قالب زیر و در غیر این صورت در خط آخر عبارت Unassigned را برمی‌گرداند.

قالب خروجی
```
{{ name }}
{{ description }}
Priority: {{ priority }}
Estimated Time: {{ estimatedTime }}
Assigned to {{ assigneeName }} OR Unassigned
```

در صورتی که وظیفه مورد نظر در سیستم موجود نباشد، رشته Task does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::printTask(std::string task)
```

نمونه فراخوانی
```
interface.printTask("Have fun");
```

نمونه خروجی
```
Task does not exist

Do Everything
Write the whole code
Priority: 10
Estimated Time: 12
Assigned to Bardia

Have fun
Just do it
Priority: 2
Estimated Time: 10
Unassigned
```

در توابع زیر هر وظیفه در این قالب برمی‌گردد.

قالب خروجی
```
{{ taskPriority }} | {{ taskName }} | {{ assigneeName }} OR Unassigned | {{ estimatedTime }}h
```

۲.۶.۱.۱ لیست: تمام وظیفه‌های یک لیست را به ترتیب ساخته شدن برمی‌گرداند. در خط اول در صورتی که لیست مورد نظر در سیستم موجود نباشد، رشته List does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::printList(std::string list)
```

نمونه فراخوانی
```
interface.printList("Code");
```

قالب خروجی
```
List {{ listName }}
{{ task }}
.
.
.
```

نمونه خروجی
```
List does not exist

List Code
10 | Do Everything | Bardia | 12h
2 | Destroy code formatting | Zhivar | 1h
2 | Have fun | Unassigned | 10h
```

۳.۶.۱.۱ همه لیست‌ها: تمام وظیفه‌های تمام لیست‌ها را به ترتیب ساخته شدن لیست و در هر لیست وظیفه‌ها را به ترتیب ساخته شدن برمی‌گرداند. قالب خروجی هر لیست مانند تابع تابع قبلی است.

امضا
```
std::string UTrelloInterface::printAllLists()
```

نمونه فراخوانی
```
interface.printAllLists();
```

نمونه خروجی
```
List Code
10 | Do Everything | Bardia | 12h
2 | Destroy code formatting | Zhivar | 1h
2 | Have fun | Unassigned | 10h

List Description
1 | Write Description | Amir | 3h

List Misc
```

۴.۶.۱.۱ وظیفه‌های کاربر: تمام وظیفه‌های کاربر را به ترتیب ساخته شدن برمی‌گرداند. در صورتی که کاربر مورد نظر در سیستم موجود نباشد، رشته User does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::printUserTasks(std::string user)
```

نمونه فراخوانی
```
interface.printUserTasks("Amir");
```

نمونه خروجی
```
User does not exist

1 | Write Description | Amir | 3h
```

۵.۶.۱.۱ وظیفه‌های انجام‌نشده کاربر: تمام وظیفه‌های انجام‌نشده کاربر را به ترتیب ساخته شدن برمی‌گرداند. در صورتی که کاربر مورد نظر در سیستم موجود نباشد، رشته User does not exist را برمی‌گرداند.

امضا
```
std::string UTrelloInterface::printUserUnfinishedTasks(std::string user)
```

نمونه فراخوانی
```
interface.printUserUnfinishedTasks("Amir");
```

نمونه خروجی
```
User does not exist

1 | Write Description | Amir | 3h
```

۶.۶.۱.۱ وظیفه‌های واگذارنشده: تمام وظیفه‌های واگذارنشده سیستم را به ترتیب اولویت و در صورت مساوی بودن به ترتیب دلخواه برمی‌گرداند.

امضا
```
std::string UTrelloInterface::printUnassignedTasksByPriority()
```

نمونه فراخوانی
```
interface.printUnassignedTasksByPriority();
```

نمونه خروجی
```
2 | Have fun | Unassigned | 10h
```

۷.۶.۱.۱ وظیفه‌های انجام‌نشده: تمام وظیفه‌های انجام‌نشده سیستم را به ترتیب اولویت و در صورت مساوی بودن به ترتیب دلخواه برمی‌گرداند.

امضا
```
std::string UTrelloInterface::printAllUnfinishedTasksByPriority()
```

نمونه فراخوانی
```
interface.printAllUnfinishedTasksByPriority();
```

نمونه خروجی
```
1 | Write Description | Amir | 3h
2 | Destroy code formatting | Zhivar | 1h
```

توابع زیر لیستی از کاربران سیستم را برمی‌گردانند. برای هر کاربر کافیست اسم آن را در یک خط برگرداند.

قالب خروجی
```
{{ name }}
```

۸.۶.۱.۱ کاربران به ترتیب مقدار کار: این تابع کاربران به ترتیب مقدار کارشان به ترتیب صعودی و در صورت مساوی بودن به ترتیب دلخواه برمی‌گرداند. مقدار کار کاربر مجموع زمان تخمینی همه وظایف واگذار شده به آن کاربر است.

امضا
```
std::string UTrelloInterface::printUsersByWorkload()
```

نمونه فراخوانی
```
interface.printUsersByWorkload();
```

نمونه خروجی
```
Zhivar
Amir
Bardia
```

۹.۶.۱.۱ کاربران به ترتیب کارایی: این تابع کاربران را به ترتیب کارایی آن‌ها به ترتیب نزولی و در صورت مساوی بودن به ترتیب دلخواه برمی‌گرداند. کارایی کاربر برابر با مجموع زمان تخمینی وظیفه‌های انجام‌شده واگذارشده به آن کاربر است.

امضا
```
std::string UTrelloInterface::printUsersByPerformance()
```

نمونه فراخوانی
```
interface.printUsersByPerformance();
```

نمونه خروجی
```
Bardia
Zhivar
Amir
```

۲ آزمون واحد

شما باید برای توابع گزارش‌گیری با استفاده از چهارچوب Catch2 آزمون واحد بنویسید. برای آشنایی با چهارچوب Catch2 می‌توانید فایل آپلود شده در سایت درس را مطالعه کنید.

۳ نکات پایانی

- در این تمرین طراحی درست موجودیت‌های سیستم و روابط آن‌ها با استفاده از مفاهیم آموخته شده در کلاس درس، از اهمیت بالایی برخوردار است.
- از صحت قالب خروجی‌های برنامه خود مطمئن شوید. توجه کنید که داور این تمرین به خطوط خالی اضافی حساس نیست.
- رعایت سبک برنامه‌نویسی درست، نام‌گذاری‌ها و تمیزی کد شما تاثیر زیادی در نمره تمرین دارد.

۴ نحوه‌ی تحویل

تمامی پرونده[۶]های برنامه‌ی خود را در قالب یک پرونده با پسوند zip و نام A4-SID.zip در صفحه CECM درس بارگذاری کنید که SID شماره دانشجویی شماست؛ برای مثال اگر شماره دانشجویی شما ۸۱۰۱۹۹۹۹۹ باشد، نام پرونده‌ی شما باید A4-810199999.zip باشد.

- برنامه‌ی شما باید در سیستم‌عامل لینوکس و با مترجم g++ با استاندارد c++11 ترجمه و در زمان معقول برای ورودی‌های آزمون اجرا شود.
- برنامه شما باید حتما Makefile داشته باشد و کلاس‌ها در فایل‌های جدا پیاده‌سازی شوند. در غیر این صورت نمره بخش‌های مربوطه را از دست خواهید داد.

[6] file

- برنامه شما با تابع main که توسط دستیاران آموزشی آماده شده است تست می‌شود. به همین منظور تابع main خود را بارگذاری نکنید.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.