دانشکده مهندسی کامپیوتر

برنامه‌سازی پیشرفته
تمرین های سری یازدهم

مدرس: سید صالح اعتمادی
طرح تمرین: امید میرزاجانی

مهلت ارسال:
شنبه ۱۷ خرداد ۹۹

فهرست مطالب

۱ مقدمه

۱.۱ موارد مورد توجه

- توجه داشته باشید که برای کسب نمره‌ی قبولی درس کسب حداقل نصف نمره‌ی هر سری تمرین الزامی می‌باشد.
- مهلت ارسال پاسخ تمرین تا ساعت ۲۳:۵۹ روز اعلام‌شده است. توصیه می‌شود نوشتن تمرین را به روزهای نهایی موکول نکنید.
- هم‌کاری و هم‌فکری شما در حل تمرین مانعی ندارد، اما پاسخ ارسالی هر کس حتما باید توسط خود او نوشته شده باشد.
- مبنای درس، اعتماد بر پاسخ ارسالی از سوی شماست؛ بنابراین ارسال پاسخ در ریپازیتوری گیت شما به این معناست که پاسخ آن تمرین، توسط شما نوشته شده است. در صورت تقلب یا اثبات عدم نوشتار پاسخ حتی یک سوال از تمرین، برای هر دو طرف تقلب‌گیرنده و تقلب‌دهنده نمره‌ی **مردود** برای درس در نظر گرفته خواهد شد.
- توجه داشته باشید که پاسخ‌ها و کدهای مربوط به هر مرحله را بایستی تا قبل از پایان زمان مربوط به آن مرحله، در سایت Azure DevOps (طبق توضیحات کارگاه‌ها و کلاس‌ها) بفرستید. درست کردن `Pull request` و `Complete` کردن `Pull request` و انتقال به شاخه‌ی `master` پس از تکمیل تمرین فراموش نشود!
- پس از پایان مهلت ارسال تا ۲ روز به ازای هر روز تاخیر ۱۰ درصد از نمره مربوط به تمرین کسر خواهد شد و پس از ۲ روز نمره‌ای به تمرین تعلق نخواهد گرفت.
- بعضی از قسمت های تمرین نیاز به پیاده سازی بر روی هر چهار زبان `C#` ، `Python` ، `C++` و `Java` را دارند بعضی هم خیر. بنابراین روبروی هر سوال زبان های مورد نیاز برای پیاده سازی مشخص شده است.

۲ آماده سازی های اولیه

۱.۲ ساخت پروژه ی C#

برای ایجاد پروژه C# کافی است کد زیر را در ترمینال خود اجرا کنید:

```
mkdir A11_cs
cd A11_cs
dotnet new sln
mkdir A11_cs
cd A11_cs
dotnet new console
cd ..
dotnet sln add A11_cs\A11_cs.csproj
mkdir A11_cs.Tests
cd A11_cs.Tests
dotnet new mstest
dotnet add reference ..\A11_cs\A11_cs.csproj
cd ..
dotnet sln add A11_cs.Tests\A11_cs.Tests.csproj
```

۲.۲ قواعد نام گذاری

قواعد نام‌گذاری تمرین را از جدول ۱ مطالعه کنید.

`*` در کل یک دیرکتوری داخل Assignments به نام A۱۱ بسازید و داخل آن،یک دیرکتوری به نام A۱۱_cs داشته باشید و فایل های مربوطه را داخل دیرکتوری مربوطه بگذارید.

جدول ۱: قراردادهای نام‌گذاری تمرین

Naming conventions		
Branch	Directory	Pull Request
`fb_A11`	`A11`	`A11`

۳ ماشین حساب

هدف از این تمرین این است که شما به کمک برنامه نویسی شی گرا،OOP، مساله ای نسبتاً پیچیده را به قسمت های کوچک تر و آسان تر تبدیل کنید.
برای این منظور سعی کردیم تمرین به گونه ای باشد که شما بیشترین مواجهه را با این مفاهیم در حداقل حجم داشته باشید. در معمول پروژه های واقعی تراکم استفاده از این مفاهیم کمتر است. ولی با این وجود این تمرین برای یادگیری این مفاهیم بسیار مناسب است. توجه داشته باشید که حجم کدی که شما باید بنویسید نسبت به تمرین های قبلی خیلی کمتر است. منتها پیچیدگی بیشتری دارد. لذا تمرکزتان را روی فهمیدن کد و امتحان و دیباگ کردن پیاده سازی های مختلف بگذارید تا خوب متوجه بشوید.
در این تمرین یک کتابخانه پردازنده عبارات ریاضی را بصورت شیء گرا پیاده سازی میکنیم. این کتابخانه قابلیت دریافت یک فایل عبارات ریاضی و محاسبه جواب آن را دارد. برای راحتی کار شما اجزاء عبارت ورودی بصورت یک جزء در هر خط داده میشود. همچنین ترتیب عبارات ریاضی به صورت preorder است. یعنی اول عملگر، بعد عملوندها. مثلا عبارت ۲+۳ در فایل به این صورت داده میشود:

```
Add
۳
۲
```

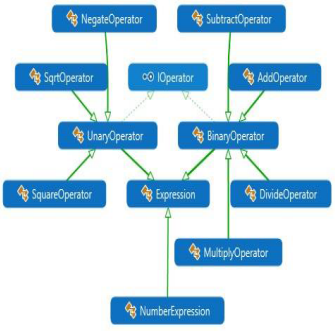
اگر به جای عدد ۳ یک عبارت دیگر داشته باشیم (مثلا ۸*۵) یعنی کل عبارت ورودی باشد ۲+(۸*۵) در فایل به صورت زیر نشان داده میشود:

```
Add
Multiply
۵
۸
۲
```

برای عملگرهایی که فقط یک عملوند دارند هم به همین ترتیب. مثلا ۸*(۵-) میشود:

```
Multiply
Negate
۵
۸
```

ممکن است یک مقداری ناآشنا باشد ولی اصلا چیز پیچیده ای نیست. چند تا مثال برای خودتان بنویسید برایتان جا می افتد. مزیت این روش نمایش عبارت های ریاضی راحتی پردازش آن است. چون ابتدا عملگر را از ورودی/فایل میگیرید بعد عملوند ها را. وقتی عملگر را میخوانید بستگی که چه عملگری باشد تعداد عملوند ها مشخص است.
از مزایای برنامه نویسی شیء گرا پخش پیچیدگی برنامه در اشیاء مختلف است. به صورتی که هر شیء کار بسیار ساده ای انجام میدهد و به راحتی قابل فهم و تست کردن است. برای نشان دادن کلاس ها و روابط آنها از نمودار کلاس Diagram Class در شکل زیر استفاده کردیم.



هر عبارتی که این ماشی حساب، آن را پردازش میکند، طبیعتاً یا یک عملگر است، یا عدد.
عملگر ها نیز به دو شاخه ی عملگرهای یگانه و عملگر های دوتایی تقسیم میشود. منظور این است که برای مثال:

- عملگرهای +،-،*،/، برای محاسبه به دو عدد سمت راست و چپش نیاز دارند.
- عملگرهای ،منفی،Sqrt،Square برای محاسبه تنها به یک عدد دیگر نیاز دارند.

کلاس Expression را با پیاده سازی کامل در اختیار شما قرار دادیم. برای این تمرین کافی است که بقیه کدها را کامل کنید. شما کافی است که متد های Evaluate ، ToString را تغییر دهید تا به ترتیب، مقدار عبارت، و نمایش آن عبارت باشد.

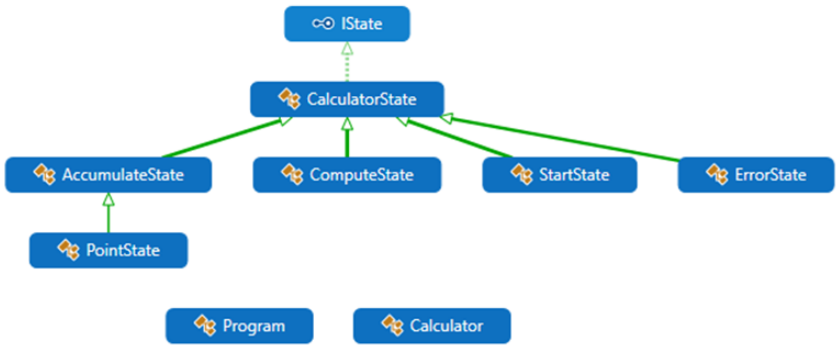
۴ StatePattern

هدف از این تمرین آشنایی شما با الگوهای برنامه‌سازی شیءگرا است. بطور خاص State Pattern .علاوه بر این مفاهیم زیر نیز مرور می‌شود

- Inheritance
- Abstract Class
- Virtual Method
- Interface
- Polymorphism
- Lambda Expressions
- Writing Testable Code

هدف دیگر این تمرین این است که از کد پیچیده وحشت نکنید. سعی کنید با خواندن کدها هر کلاس را به تنهایی متوجه بشوید که چکار میکند، روابط بین کلاس‌ها را هم متوجه بشوید. ممکن است که نتوانید تصور کنید کل مجموعه تمام کلاس‌ها با هم چکار می‌کنند، ولی اگر هر کلاس را متوجه بشوید و ارتباطش با کلاس‌های دیگر را هم متوجه بشوید، می‌توانید هر کلاس را درست پیاده‌سازی کنید. علاوه بر این به اهمیت و فایده یونیت تست پی خواهید برد که چقدر به دیباگ کردن و درست کردن کد کمک می‌کند. برای تمرین می‌توانید ابتدا از یونیت تست‌ها استفاده نکنید و ببینید آیا می‌توانید تمرین را انجام دهید؟

موضوع این تمرین نوشتن یک ماشین حساب بسیار ساده است که چهار عملگر اصلی را انجام می‌دهد. برای نحوه کار ماشین حساب، از یک ماشین حساب ساده توی موبایل یا کامپیوترتان استفاده کنید. این ماشین حساب هم مثل آن‌ها عمل می‌کند. با این تفاوت که فقط از صفحه‌کلید ورودی دریافت می‌کند نه با کلیک یا لمس. به عنوان مثال یک ماشین حساب باز کنید. دکمه صفر را چند بار فشار دهید. آیا صفحه نمایش تغییری می‌کند؟ نقطه (ممیّز) را فشار دهید. چه اتفاقی می‌افتد؟ چند بار دیگر هم فشار دهید. آیا صفحه نمایش تغییری می‌کند؟ حالا چند بار صفر را فشار دهید. آیا صفحه نمایش این دفعه فرقی می‌کند؟ چرا؟ در ابتدا ماشین حساب در وضعیت شروع (StartState) بود. ولی بعد از این‌که نقطه را فشار دادید از وضعیت شروع خارج شد و به وضعیت نقطه (PointState) وارد شد. در اینجا اتفاقات کلیدهای ورودی هستند. اگر با این دید به کد نگاه کنید، بهتر متوجه می‌شوید. کد را چندین بار مطالعه کنید اما از آن وحشت نکنید! قسمت‌هایی که نیاز به پیاده‌سازی شما دارد با علامت # و عدد کنار آن مانند: ۱# ، ۲# ، ... مشخص شده است. این ماشین حساب باید مثل یک ماشین حساب معمولی کار بکند. اگر مطمئن نیستید در یک حالتی باید چکار کند، یک ماشین حساب ساده بردارید و امتحان کنید. نمودار روابط کلاس ها در شکل



موجود است.
مثل تمرین قبل دقت، کنید که تعداد خطهای کد که شما باید اضافه کنید کم است. از دیباگ کردن برنامه برای فهم این که چطوری کار می‌کند استفاده کنید. لطفا در کامنت‌ها توضیح اضافه در مورد روابط کلاس‌ها و این که هر کلاسی چرا عضو کلاس دیگر است یا از آن ارث میرد یا ... بدهید. اگر فقط روی جاهایی که کد پاک شده تمرکز کنید کارتان سخت خواهد شد. باید همه کلاس‌ها و روابطشان را متوجه بشوید تا بتونید جاهای خالی را پر کنید. بعد از اینکه از درست کار کردن برنامه اطمینان پیدا کردید، عملگر توان را به ماشین حساب اضافه کنید. نشانه عملگر توان کاراکتر ^ است.

۱.۴ تست Zero

بعد از درست کردن پروژه‌ها و اضافه کردن فایل‌ها این تست بدون هیچ تغییری از طرف شما، پاس می‌شود. متن این تست و بخصوص متد RunTest را با دقت مطالعه کرده و خط به خط اجرا/دیباگ کنید و از تسلط بر نحوه انجام تست اطمینان حاصل کنید. به طور خلاص هدف این تست این است که بعد از باز کردن ماشین‌حساب هر چند بار دکمه صفر فشار داده شود صفحه نمایش عدد صفر را نشان داده و تغییری نمی‌کند.

۲.۴ تست Accumulation

برای پیدا کردن شهود نسبت به هدف این تست برنامه calc.exe و یا یک ماشین‌حساب دستی باز کنید. هر چند بار عدد صفر را بزنید، آیا تغییری در صفحه نمایش مشاهده می‌کنید؟ حال، ابتدا دکمه یک را زده و بعد دکمه صفر را به دفعات فشار دهید. آیا متوجه تفاوت رفتار می‌شوید؟ برای پاس شدن این تست لازم است از دیباگر استفاده کرده و تغییرات لازم را در کد ایجاد کرده تا تست پاس بشود.

۳.۴ تست AccumulateState

هدف این تست مانند بخش قبل است. با این تفاوت که وقتی ماشین‌حساب اجرا می‌شود، هر عددی غیر از صفر باید رفتاری متفاوت از عدد صفر داشته باشد. ولی همانطور که در تست قبل مشاهده کردید، این رفتار متفاوت فقط در ابتدا است. بعد از اینکه عددی غیر از صفر وارد شد، دیگر عدد صفر با دیگر اعداد تفاوتی نمی‌کند.

۴.۴ تست PointState

هدف این تست،آزمون وارد کردن درست اعداد اعشاری در ماشین حساب می‌باشد.

۵.۴ تست PointsOnlyState

برای پاس شدن این تست لازم است از دیباگر استفاده کرده و تست را دیباگ کنید. این تست زمانی پاس می‌شود که توی ماشین‌حساب هر چند بار دکمه نقطه فشار داده شود، فقط یک صفر و یک نقطه روی صفحه نمایش نشان داده شود. برای نمونه و مقایسه می‌توانید از یک ماشین‌حساب دستی یا برنامه calc.exe استفاده کنید.

۶.۴ تست ExtraPoint

مجددا برای فهمیدن هدف این تست از یک ماشین‌حساب استفاده کنید. آیا بعد از اینکه مثلا عدد یک و یک دهم را وارد کردید فشار دادن دکمه نقطه تغییری در ماشین‌حساب ایجاد می‌کند؟ این نشان‌دهنده این است که بعد از وارد کردن نقطه، ماشین‌حساب به حالتی وارد می‌شود که وارد کردن نقاط بعدی صفحه نمایش را تغییر نمی‌دهد. مجددا با کمک گرفتن از دیباگر تغییرات لازم در برنامه را انجام دهید.

۷.۴ تست StartState

حال که اعداد به درستی به ماشین‌حساب وارد می‌شوند، لازم است که بتوانیم محاسبات بین اعداد را انجام دهیم. برای گام اول،این تست زمانی پاس می‌شود که اگر دکمه بعلاوه فشار داده شد، ماشین‌حساب به حالت ComputeState تغییر حالت دهد.

۸.۴ تست Sum

بعد از این مقدمات، لازم است که ماشین حساب یک حساب ساده را بتواند انجام دهد. به این معنی که بعد از فشار دادن دکمه مساوی، نتیجه محاسبه نمایش داده شود.

۹.۴ تست ErrorState

چنانچه بعد از نمایش نتیجه یک محاسبه، دکمه مساوی مجددا فشار داده شود، لازم است حالت ماشین‌حساب به ErrorState تغییر پیدا کند و صفحه نمایش تغییری نکند.

۱۰.۴ تست Multiply

حال که عملگر جمع به درستی پیاده‌سازی شد، نوبت عملگر ضرب می‌باشد. تغییرات لازم برای پاس شدن این تست را اعمال کنید.

۱۱.۴ تست MultipleSum

چنانچه تست‌های قبل به درستی پیاده‌سازی شده باشند، این تست نیز بدون هیچ تغییری باید پاس بشود.

۱۲.۴ تست Divide

این تست درستی اجرای عملگر تقسیم را راست‌آزمایی می‌کند.

۱۳.۴ تست StartingPoint

چنانچه تست‌های قبل به درستی پیاده‌سازی شده باشند، این تست نیز بدون هیچ تغییری باید پاس بشود.

۱۴.۴ تست Power

حال که چهار عمل اصلی را برای ماشین‌حساب پیاده سازی کردید، نوبت پیاده‌سازی یک عملگر جدید می‌باشد. تغییرات لازم را برای پاس شدن تست توان اعمال کنید.

قهرمان باشید!