استاد: رامتین خسروی
طراحان: سهیل ذبیخش، زهرا یزدانی، امین ستایش
مهلت تحویل: دوشنبه ۱۸ فروردین ۱۳۹۹، ساعت ۲۳:۵۵

۱ مقدمه

- هدف این تمرین تمیز کردن کدتان در تمرین یک است به طوری که ساختار کلی و طراحی شما تغییر نکند.
- عملیات بهسازی باید در مرحله‌های کوچک اجرا شود و پس از هر تغییر با اجرای موارد آزمون از درستی کد خود مطمئن شوید. یک نمونه از این سبک تغییرات را می‌توانید در لینک لیست تغییرات در بخش نمونه‌ی بهسازی کد مشاهده کنید.
- دربخش نمونه‌ی بهسازی کد یک نمونه از تمیز کردن کد که مربوط به تمرین صفر است قرار داده شده است. لطفا به آن‌ها رجوع کنید.
- دو خلاصه‌ی قابل استفاده نیز برای شما در بخش خلاصه‌ها قرار داده شده است. توصیه می‌شود که از آن‌ها نیز استفاده کنید.

۲ کد تمیز

تعاریف زیادی از *کد تمیز*[۱] وجود دارد؛ اما احتمالاً یکی از بهترین تعریف‌ها متعلق به *بیارنه استروستراپ*[۲] خالق و توسعه‌دهنده‌ی زبان C++ است. وی در تعریف خود از کد تمیز، دو مورد را به عنوان معیارهای اساسی تمیزی کد برمی‌شمارد:

- منطق و الگوریتم کد باید آن‌قدر واضح و قابل‌فهم باشد که اشکالات و نقص‌های جزئی نتوانند از چشم برنامه‌نویس و آزمونگر کد دور بمانند. ضمن این که وضوح کد باید به حدی بالا باشد که برنامه‌نویس را از نوشتن یادداشت (کامنت[۳]) بی‌نیاز کند.
- کارایی[۴] برنامه‌ی نوشته‌شده باید در بهترین[۵] شکل ممکن باشد تا بعدها برنامه‌نویس دیگری به بهانه‌ی بهینه‌سازی[۶] برنامه‌ی سابق با ایجاد تغییرات نادرست سبب نامنظم شدن و کثیف شدن کد نشود.

در عمل، در اکثر مواقع شما بعد از یک طراحی نسبتاً خوب و پیاده‌سازی آن، برای مدتی طولانی از آن کد برای هدف خود استفاده می‌کنید و در طول این مدت تغییراتی در آن ایجاد می‌کنید و قابلیت‌های زیادی را به آن می‌افزایید.

پس از مدتی نه‌چندان طولانی، این تغییرات باعث می‌شوند که شما دیگر عملکرد کد را به‌وضوح متوجه نشوید و به تبع آن، توانایی تغییر و ارتقای کد را نیز از دست می‌دهید. همین زنجیره‌ی اتفاقات به ظاهر ساده در تاریخچه‌ی نسبتاً کوتاه توسعه‌ی نرم‌افزاری باعث نابود شدن شرکت‌های بسیاری در این عرصه شده است.

حال با توجه به خطرات و مشکلاتی که یک کد کثیف به همراه دارد، باید راه‌حلی برای رفع کثیف بودن کد و جلوگیری از کثیف شدن آن ارائه دهیم. *بهسازی*[۷] عملیاتی است که در طی آن ساختار یک نرم افزار به صورتی تغییر و بهبود می‌یابد که بدون تغییر کارکردها و تغییر رابط کاربری[۸] برنامه، ساختار درونی کد به طرز قابل توجهی تمیزتر می‌شود.

[1] clean code
[2] Bjarne Stroustrup
[3] comment
[4] performance
[5] optimal
[6] optimization
[7] refactoring
[8] interface

بنیادی‌ترین مفهوم یاری‌کننده‌ی یک برنامه‌نویس در طی عملیات بهسازی شناخت عناصری است که باعث کثیف شدن کدها می‌شوند و به اصطلاح به آن‌ها code smell گفته می‌شود.

در این تمرین از شما انتظار می‌رود کدی را که برای تمرین اول نوشته‌اید تمیز کنید؛ بنابراین **خوانایی و تمیز بودن کد** در این تمرین بیشترین اهمیت را دارد. در ادامه توضیحاتی درباره‌ی بهسازی کد ارائه می‌شود. پیشنهاد می‌کنیم که ابتدا صورت این تمرین را به طور کامل مطالعه کنید و سپس بهسازی کد خود را شروع کنید.

۳ معیارهای تمیزی کد

عواملی در کد وجود دارند که ممکن است باعث کثیف شدن آن شوند؛ در ادامه برخی از این عوامل توضیح داده شده‌اند. توجه کنید که معیار نمره‌دهی در این تمرین همین عوامل خواهد بود و به ازای هر یک از موارد زیر که در کد شما وجود داشته باشد مقداری از نمره‌ی شما کاسته خواهد شد. ساختار کلی کد و طراحی شما نباید تغییر کند و فقط ساختار درونی کد شما که شامل مواردی که در ادامه آمده است، می‌تواند تغییر کند. با این حال می‌توانید مشکلات کد خود را رفع کنید. تغییرات را مرحله‌به‌مرحله و در قدم‌های کوچک اعمال کنید و پس از هر مرحله با اجرای موارد آزمون اطمینان پیدا کنید که عملکرد برنامه با مشکل مواجه نشده باشد.

این عوامل خلاصه‌ای از کتاب Clean Code[9] هستند. عبارت مقابل هر بخش شماره‌ی فصل مرتبط با آن بخش را در کتاب نشان می‌دهد. نسخه‌ی الکترونیکی این کتاب در سایت درس قابل دسترسی است.

۱.۳ نام‌گذاری فصل ۲

- استفاده از نام‌های نامرتبط کار درستی نیست؛ مثلاً استفاده از متغیرهایی با نام‌های a و b که هیچ توضیحی ارائه نمی‌دهند و خواننده را گیج می‌کنند. (فصل ۱۷، N1)
- نام متغیر باید کاربرد و مکان استفاده از متغیر را نشان دهد. اسامی کلاس‌ها[10]، ساختارها[11] و اشیا[12] باید عبارت‌های اسمی[13] باشند. اسامی کلاس‌ها باید با حرف بزرگ[14] شروع شوند؛ مانند: Customer، AddressParser و Account.
- نام تابع باید وظیفه‌ی تابع و تأثیرات جانبی[15] احتمالی تابع بر محیط را توضیح دهد. اسامی توابع باید عبارت‌های امری[16] باشند و با حرف کوچک شروع شوند؛ مانند: مثل get، set، get_flagged_cells و deletePage.

۲.۳ توابع فصل ۳

- یک تابع باید یک کار واحد را به خوبی انجام دهد. یعنی فقط یک کار را به صورت بهینه و بدون هیچ اثر جانبی انجام دهد.
- توابع باید تا حد امکان **کوتاه** باشند. طول توابع به‌ندرت باید به ۲۰ خط برسد.
- هر تابع باید حداکثر به یک سطح پایین‌تر دسترسی داشته باشد؛ مثلاً حرکت با یک حلقه روی لیستی از اشیا و تغییر ویژگی[17]های هر کدام از اشیا دسترسی تابع به دو سطح پایین‌تر محسوب می‌شود. این عملیات باید در تابعی جداگانه

[9] Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship* (1 ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.
[10] classes
[11] structures
[12] objects
[13] noun phrase
[14] capital
[15] side-effects
[16] verb phrase
[17] property

پیاده‌سازی شود.

- تعداد آرگومان‌های تابع تا حد امکان کم (ترجیحاً ۱ یا ۲ و حداکثر ۳ تا) باشد. گاهی می‌توان از آرگومان‌هایی از نوع اشیا یا ساختارها برای بسته‌بندی چند آرگومان مرتبط و کاهش تعداد آرگومان‌های توابع استفاده کرد؛ مثلاً به جای دو متغیر از نوع `double` از یک شیء از نوع `Point` استفاده کنیم.

- آرگومان‌های تابع نباید به عنوان خروجی تابع استفاده می‌شوند. یک تابع فقط می‌تواند از طریق مقدار بازگشتی خود بر محیط بیرون تأثیر بگذارد و نباید از طریق تغییر آرگومان‌ها بر محیط تأثیری داشته باشد. (فصل ۱۷، F2)

- استفاده از پرچم[۱۸]ها (معمولاً آرگومان از نوع بولی[۱۹]) برای تعیین نحوه‌ی عملکرد تابع کار درستی نیست. مثالی از این کار ارسال یک متغیر به نام flag به تابع فقط برای اجرای یک بخش کد در حالتی خاص است. چنین تابعی در واقع حاصل ادغام دو تابع مختلف است که باید به صورت جدا از هم پیاده‌سازی شوند و در زمان مناسب فراخوانی[۲۰] شوند. (فصل ۱۷، F3)

- انجام بیش از یک کار در یک تابع درست نیست. هر تابع باید فقط یک کار را انجام دهد و این کار را به شیوه درستی پیاده و اجرا کند. همچنین نباید در کنار انجام این کار تأثیری در متغیرها و دیگر اجزای برنامه داشته باشد. (فصل ۱۷، G30)

۳.۳ یادداشت‌ها (کامنت[۲۱]ها) فصل ۴

- در این تمرین استفاده از یادداشت به هیچ نحوی قابل قبول نیست. حتی اگر توضیحی نباشند یا فقط برای جدا کردن تکه های کد باشند.

برای آشنایی بیشتر با یادداشت‌های مفید و مضر به فصل ۴ کتاب مراجعه کنید.

۴.۳ قالب‌بندی[۲۲] فصل ۵

- **دندانه‌گذاری**[۲۳] در کد اهمیت بالایی دارد و حتماً هر محدوده[۲۴] باید یک دندانه داخل‌تر باشد. همچنین هر تابع باید حداکثر یک یا دو دندانه داخل رفته باشد.

- در نام‌گذاری توابع و متغیرها باید از یک روش واحد نام‌گذاری[۲۵] استفاده شده باشد؛ مثلاً یا همه‌ی متغیرها به صورت camelCase یا همه به شکل snake_case نام‌گذاری شده باشند. این موارد شامل اسم کلاس‌ها که باید به صورت PascalCase باشند نمی‌شود. در هر صورت، دیگر قوانین نام‌گذاری نیز باید رعایت شوند.

- **سازگاری**[۲۶] یکی دیگر از نکات مهم در کد نویسی است. سعی کنید که همیشه از یک الگو و روند در پیاده‌سازی و نام‌گذاری‌های خود استفاده کنید. (فصل ۱۷، G11)

[18] flag
[19] boolean
[20] Call
[21] comment
[22] formatting
[23] indentation
[24] scope
[25] naming convention
[26] consistency

۵.۳ مشکلات دیگر

اشکالات دیگری نیز ممکن است در کد شما دیده شود که باید آن‌ها را برطرف کنید، عبارتند از:

- **کد تکراری**[۲۷]: از مهم‌ترین نکاتی که باید در این تمرین رعایت کنید جلوگیری از تکرار کد است و کد تکراری به هیچ وجه قابل قبول نیست. (G5)
- **کدهای مرده**[۲۸]: کدهایی که دیگر در هیچ قسمتی از برنامه فراخوانی نمی‌شوند نباید در متن برنامه وجود داشته باشند. (G9)
- استفاده از **اعداد جادویی**[۲۹]: اعداد و ثابت‌ها نباید به طور مستقیم در کد استفاده شوند؛ بلکه باید در ثابت[۳۰]ها ذخیره شوند و از این متغیرها در کد استفاده شود؛ مثلاً عدد π را باید در ابتدای برنامه در ثابتی به نام PI ذخیره کنیم و از این ثابت در بقیه کد استفاده کنیم. (G25)

۴ دو خلاصه‌ی قابل استفاده

دو نمونه خلاصه‌ی قابل استفاده را در لینک‌های زیر می‌توانید مشاهده کنید.

- Clean Code
- Refactoring

۵ نمونه‌ی بهسازی کد

یک نمونه از بهسازی کد را می‌توانید در لینک‌های زیر مشاهده کنید. این کد مربوط به تمرین صفر است:

- کد اولیه
- کد نهایی
- مقایسه‌ی دو کد
- لیست تغییرات

۶ نکات پایانی

- درستی کد شما نباید در تمیز کردن از بین برود. کد نهایی شما با موارد آزمون تمرین ۱ نیز آزموده خواهند شد و اگر در آزمونی که قبلاً با موفقیت گذرانده شکست بخورد، نمره‌ی شما کاسته خواهد شد. موارد آزمون تمرین ۱ را می‌توانید از سایت درس دریافت کنید.
- در این تمرین اجازه‌ی رفع مشکلات[۳۱] کد اولیه خود را دارید، اما نمره‌ای بابت آن دریافت نمی‌کنید.
- پیشنهاد می‌کنیم فصل ۱۷ کتاب Clean Code را که مربوط به Code Smells است به طور کامل مطالعه کنید. همچنین، مطالعه‌ی بخش‌هایی از کتاب Refactoring[۳۲] احتمالاً برای شما مفید خواهد بود.

[27] duplication
[28] dead codes
[29] magic numbers
[30] constant
[31] debug
[32] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code* (1 ed). Addison-Wesley, Boston, MA, USA.

۷ نحوه‌ی تحویل

پرونده‌ی[۳۳] برنامه‌ی خود را با نام A1C-SID.cpp در صفحه‌ی CECM درس بارگذاری کنید که SID شماره‌ی دانشجویی شماست؛ برای مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۹۸۹۹۹ باشد، نام پرونده‌ی شما باید A1C-810198999.cpp 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.
- در این تمرین اجازه **ندارید** از مفاهیم شیءگرایی استفاده کنید.
- از صحت قالب[۳۴] ورودی‌ها و خروجی‌های برنامه‌ی خود مطمئن شوید. توصیه می‌کنیم حتماً برنامه‌ی خود را با ورودی و خروجی نمونه بیازمایید و از ابزارهایی مانند `diff` برای اطمینان از درستی عملکرد برنامه‌ی خود برای ورودی نمونه استفاده کنید.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.

[33] file
[34] format