



پروژه‌ی سوم درس ارزشهای رمزگذاری شده

استاد درس: دکتر بهرک

مهلت تحویل: 20 تیر



دستیار مربوطه: محمد محمودیان mahmoodian@ut.ac.ir

در این تمرین کامپیوتری هدف این است که شما با زبان برنامه‌نویسی Solidity و همچنین یک API¹ به نام web3.js آشنا شوید به طوریکه بتوانید یک نرم‌افزار غیرمتمرکز² ساده را پیاده سازی کنید. در نرم‌افزارهای غیرمتمرکز نیز، همانند سایر نرم-افزارها، لازم است یک قسمت سمت کاربر³ و یک قسمت سمت سرور⁴ پیاده سازی شود، که در این تمرین شما با پیاده سازی هر دو سمت آشنا می‌شوید.

1. دفترچه حساب زنجیره‌ی بلوکی⁵

هدف این است که یک سیستم غیرمتمرکز طراحی شود که بتواند بدهی و طلب افراد را ثبت کند؛ درواقع نمونه‌ی غیرمتمرکز از یک دفترچه حساب. برای شفاف شدن بیشتر، یک مثال از نحوه‌ی کار این سیستم در ادامه آورده می‌شود.

فرض کنید که علیرضا، الهام و شبنم سه دوست هستند که معمولاً با یکدیگر بیرون می‌روند و غذا می‌خورند. آخرین بار که علیرضا و الهام با یکدیگر ناهار خورده‌اند، هزینه آن را الهام پرداخت کرده است. بنابراین علیرضا 30 هزار تومان به الهام بدهی دارد. همچنین یک بار که الهام و شبنم برای شام بیرون رفته‌اند شبنم هزینه آن را پرداخت کرده است، پس الهام 30 هزار تومان به شبنم بدهکار است.

حال فرض کنید که شبنم به پول نیاز دارد و مبلغ 30 هزار تومان از علیرضا قرض بگیرد؛ در این لحظه هر سه نفر می‌توانند توافق کنند که هیچ یک از آن‌ها به دیگری بدهی ندارد و از تبادل پول خودداری کنند. درواقع هر زمانی که یک دور⁶ از بدهی‌ها وجود داشته باشد می‌توان آن دور را حذف کرد و از جابجا شدن پول جلوگیری کرد.

ما قرار است که یک سیستم غیرمتمرکز طراحی و پیاده سازی کنیم که مشخص کند هر شخص به چه کسی و چه مبلغی بدهکار است، بدون اینکه نیاز باشد یک سازمان سومی⁷ در میان باشد. به صورتی که مثلاً وقتی الهام هزینه ناهار علیرضا را پرداخت می‌کند از او بخواهد که در شبکه زنجیره‌ی بلوک بدهکار بودنش را تصدیق کند. نحوه‌ی تصدیق کردن بدین شکل است که فرد بدهکار یک تابع را صدا می‌زند و در آن شخص طلبکار و مبلغ بدهی را مشخص می‌کند؛ در کد برنامه این تصدیق‌نامه (تاییدیه) را

¹ Application Protocol Interface

² Decentralized Application(DApp)

³ Client side

⁴ Server side

⁵ Blockchain Splitwise

⁶ Cycle

⁷ Third party

IOU⁸ می‌نامیم. فضای موجود و عمومی و همچنین قابل دسترسی شبکه‌ی زنجیره‌ی بلوک می‌تواند به عنوان محل ذخیره سازی و یا به عبارتی سمت سرور برای لیست بدهی‌های هر شخص استفاده شود. برای راحتی کار با این سرور لازم است که یک محیط کاربری نیز طراحی شود که جلوتر در مورد آن صحبت خواهیم کرد. ☺

2. آماده‌سازی جهت شروع

- 1) نصب نرم‌افزارهای پیشنهادی: لازم است که بر روی سیستم خود Node.js را نصب کنید. می‌توانید از این [لینک](#) استفاده کنید. لازم است که علاوه بر Node.js ، npm نیز بر روی سیستم شما نصب شود. به کمک npm می‌توانید سایر پکیج‌های مورد نیاز را نصب کنید. نسخه‌ی ویندوز Node.js همراه خود npm را نیز دارد، اما در سیستم‌عامل‌های دیگر ممکن است لازم باشد آن را جداگانه نصب کنید.
- 2) نصب شبیه‌ساز شبکه اتریوم و اجرای آن: برای اینکار در ترمینال خود از دستور `npm install -g ganache-cli` استفاده کنید. با این کار یک سرور به نام ganache بر روی سیستم شما نصب می‌شود که با اجرا شدن آن تعدادی اکانت اتریوم (پیشفرض 10 عدد) با موجودی‌های تعیین شده (100 اتر) ایجاد می‌شود که از آن‌ها می‌توانید برای انجام این تمرین کامپیوتری استفاده کنید. می‌توان از طریق پروتکل http به این سرور وصل شد، آدرس پیشفرض آن localhost یا همان 127.0.0.1 است و پورت پیشفرض آن نیز 8545 است. برای اجرا شدن این سرور کافایت دستور ganache-cli را در ترمینال خود وارد کنید. برای متوقف کردن سرور نیز می‌توانید از Ctrl+C استفاده کنید.
- 3) از صفحه‌ی درس پوشه ItCProject2Starter را دانلود کنید.
- 4) در جستجوگر وب خود صفحه‌ی مقابل را باز کنید. <https://remix.ethereum.org>. در قسمت 'Run' محیط اجرایی⁹ را بر روی 'Web3 Provider' تنظیم کنید. و آدرس آن را نیز بر روی <http://localhost:8545> که آدرس پیشفرض نیز هست، تنظیم کنید. محیطی که در آن هستید جایی است که لازم است شما قرارداد هوشمند¹⁰ خود را بنویسید و آن را کامپایل کنید. و در شبکه‌ی شبیه‌سازی شده مستقر (ثبت)¹¹ کنید.
- 5) در پوشه‌ای که از سایت درس دانلود کرده اید یک فایل به نام index.html وجود دارد. آن را در مرورگر خود باز کنید. باید صفحه‌ای را تحت عنوان 'Blockchain Splitwise' بتوانید ببینید. توصیه می‌شود که قسمت javascript console مرورگر خود را نیز باز کنید و خطاهای احتمالی را مشاهده کنید. (برای نحوه‌ی باز کردن این قسمت می‌توانید از این [لینک](#) کمک بگیرید). احتمالاً هشدارهایی را می‌بینید که می‌توانید آن‌ها را نادیده بگیرید.
- 6) فایل‌های درون پوشه‌ی دانلود شده را در محیط برنامه نویسی مورد علاقه خود باز کنید. (مانند Atom, Sublime,..) شما برای کامل شدن سمت کاربر لازم است که فایل script.js را کامل کنید. توصیه می‌شود که سایر فایل‌های موجود را نیز بررسی کنید.
- 7) قبل از شروع توصیه می‌شود که راجع به web3.js و برنامه نویسی solidity اطلاعات بیشتری کسب کنید. همچنین فکر کردن راجع به نحوه‌ی کار سیستم کلی و همچنین نحوه‌ی ذخیره سازی اطلاعات و ساختارهای داده‌ای که قرار است استفاده کنید، قبل از کدزنی می‌تواند مفید باشد.

⁸ I Owe You

⁹ Environment

¹⁰ Smart contract

¹¹ Deploy

8) در قسمت بعد کارهایی که قرار است انجام دهید با جزئیات بیشتری توضیح داده می‌شود، بعد از اینکه قرارداد هوشمند خود را نوشتید و آن را بر روی شبکه ثبت کردید لازم است که آدرس^{۱۲} آن و همچنین چکیده^{۱۳} آن را در فایل `script.js` به روزرسانی کنید. چکیده را می‌توانید از قسمت 'Compile' و آدرس را نیز می‌توانید از قسمت 'Deployed Contracts' در بخش 'Run' در صفحه Remix بیابید و کپی کنید.

3. نیازمندی‌ها

همانطور که قبلاً توضیح داده شد، این تمرین کامپیوتری از دو بخش مهم تشکیل می‌شود: قرارداد هوشمند که به زبان `solidity` نوشته می‌شود و در شبکه‌ی آنلاین زنجیره‌ی بلوک ثبت می‌شود و قسمت سرور شما را تشکیل می‌دهد و همچنین یک محیط کاربری که به زبان `JavaScript` نوشته می‌شود و در مرورگر شما اجرا می‌شود و از طریق `web3.js` به سرور شما وصل می‌شود و اطلاعاتی را که میخواهید نمایش می‌دهد.

1) توابع سمت کاربر

- I. `getUsers()`: این تابع یک لیست از آدرس‌های شبکه‌ی اتریوم را برمی‌گرداند. شما می‌توانید آدرس تمام افراد موجود در شبکه را برگردانید و یا اینکه فقط افرادی را برگردانید که تا به حال فعالیت مالی (طلب یا بدهی) داشته‌اند. این تابع می‌تواند به عنوان تابع کمکی در دیگر توابع مورد استفاده قرار بگیرد.
- II. `getTotalOwed(user)`: این تابع آدرس یک کاربر را می‌گیرد و کل مبلغی را که آن کاربر بدهکار است برمی‌گرداند.
- III. `getLastActive(user)`: این تابع زمان آخرین فعالیت کاربر را برمی‌گرداند. منظور از آخرین فعالیت، فرستادن IOU و یا ذکر شدن به عنوان بستانکار در یک IOU توسط یک کاربر دیگر است. فرمت زمانی باید به صورت ثانیه‌های گذشته از مبدأ تاریخ (1 ژانویه 1970) باشد.
- IV. `add_IOU(creditor, amount)`: این تابع به عنوان تاییدیه فرستاده می‌شود که نشان می‌دهد شخصی که آن را صدا می‌زند مبلغی معادل `amount` را به فرد `creditor` بدهکار است. این تابع چیزی را برنمی‌گرداند. در این تابع لازم است که مشکل وجود حلقه نیز برطرف شود که جلوتر در مورد آن توضیح خواهیم داد.

2) توابع سمت سرور

یک فایل به نام `BlockchainSplitwise.sol` در پوشه‌ی `starter` وجود دارد که لازم است همه‌ی آن را خودتان بنویسید و شامل توابع زیر باشد.

- I. `lookup(address debtor, address creditor) public view returns (unit32 ret)`: این تابع مبلغی را که فرد `debtor` به شخص `creditor` بدهکار است، برمی‌گرداند. برای این کار کافی است که در بدنه‌ی تابع متغیر `ret` را مقداردهی کنید و نیازی به نوشتن `return` در انتهای تابع نیست. کلمه `view` یعنی اینکه این تابع قرار نیست متغیرهای موجود در شبکه را تغییر دهد، بلکه صرفاً قرار است مقادیر این متغیرها را بخواند.

¹² Contract address

¹³ ABI

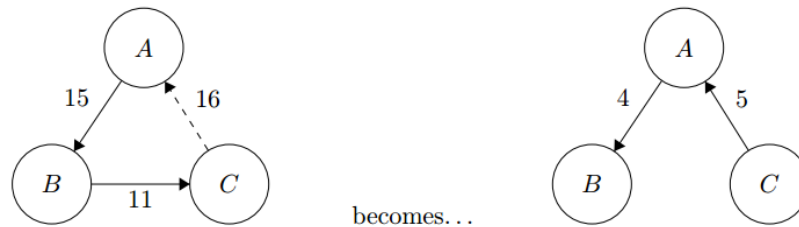
II. `add_IOU(address creditor, uint32 amount, ...)` : این تابع تاییدیه نهایی در سرور را انجام می‌دهد و نشان می‌دهد که `msg.sender` مبلغ `amount` را به فرد `creditor` بدهکار است. `amount` لازم است که عدد مثبتی باشد. شما می‌توانید بسته به نیازتان آرگومان‌های دیگری برای این تابع تعریف کنید.

شما می‌توانید بسته به نیازتان توابع دلخواه دیگری را در هریک از دو سمت کاربر و سرور تعریف کنید و از آن‌ها استفاده کنید. در سمت کاربر می‌توانید توابع سمت سرور را به صورت `BlockchainSplitwise.functionname(arguments)` صدا بزنید.

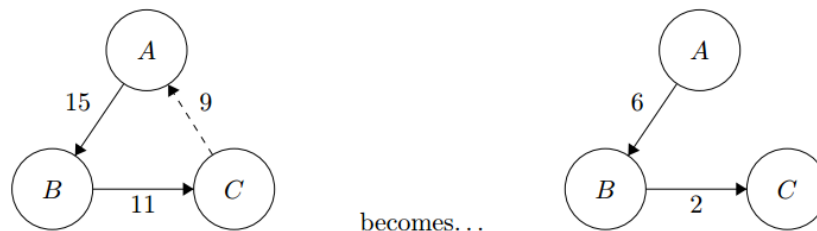
4. حل مشکل وجود حلقه‌ی بدهی‌ها

برای تحلیل راحت‌تر، از مدل گراف جهت‌دارِ وزن‌دار برای مسئله‌ی خود استفاده می‌کنیم، بدین صورت که هر یک از کاربرها یک گره محسوب می‌شوند و هریک از یال‌ها وزنی دارد که مقدار وزن آن نشان‌دهنده‌ی مبلغ بدهی است. جهت یال‌ها از بدهکار به سمت بستانکار است. مثلاً وقتی که شخص `A` مبلغ `X` را به شخص `B` بدهکار است آنگاه: $A \xrightarrow{X} B$. نرم‌افزار ما لازم است به گونه‌ای کار کند که اگر حلقه‌ای ایجاد شد آن را از بین ببرد برای این کار لازم است که کوچکترین یال در حلقه پیدا شود و مقدار آن از تک تک یال‌ها کم شود به صورتیکه یکی از یال‌ها به ارزش صفر برسد.

به عنوان مثال اگر $A \xrightarrow{15} B$ و $B \xrightarrow{11} C$ آنگاه اگر $C \xrightarrow{16} A$ را اضافه کند لازم است که نرم‌افزار ما گراف را به صورت زیر تغییر دهد.



به عنوان یک مثال دیگر اگر $C \xrightarrow{9} A$ را اضافه کند آنگاه گراف جدید به صورت زیر خواهد بود.



لازمه‌ی انجام این کار این است که شما در تابع `add_IOU` در قسمت کاربر قبل از فرستادن دستور به سمت سرور چک کنید که آیا با اضافه شدن این تراکنش جدید حلقه‌ای به وجود می‌آید یا نه و در صورت وجود، حداقل یکی از آن‌ها را حذف کنید. نگران

موارد پیچیده نباشید و یا اینکه لازم نیست به دنبال پیدا کردن راه‌های بهینه برای حذف حلقه باشید. بنابراین لازم است در توابع سمت سرور این فرض را در نظر بگیرید که هیچ حلقه‌ای وجود ندارد و قسمت حذف حلقه در سمت کاربر انجام می‌گیرد.

در کدهای سمت کاربر یک تابع کمکی به نام `doBFS(start, end, getNeighbors)` برای شما قرار داده شده است که الگوریتم `bread-first search` را انجام می‌دهد و خروجی آن لیستی از گره‌ها است که گره `start` را به گره `end` وصل می‌کند، البته در صورتیکه مسیری بین این دو گره وجود داشته باشد. دقت شود که آرگومان سوم، خود یک تابع است که یک گره را می‌گیرد و همسایه‌های آن را به صورت یک لیست برمی‌گرداند و لازم است خودتان آن را بنویسید. در صورتیکه مسیری بین دو گره وجود نداشته باشد، خروجی تابع یک لیست خالی خواهد بود. استفاده از این تابع اختیاری است و شما می‌توانید از هر الگوریتم دیگری برای پیدا کردن مسیر استفاده کنید.

آخرین چیزی که به نظر می‌رسد باید انجام دهیم، پرداخت بدهی است به این صورت که فرض کنید آخرین بار علیرضا مبلغ 40 هزار تومان به شبنم بدهکار است حال اگر پول خود را به شبنم پرداخت کند لازم است که شبنم در شبکه یک تاییدیه بفرستد که مبلغ 40 هزار تومان به علیرضا بدهکار است با این کار یک حلقه‌ی کوچک به وجود می‌آید که انتظار می‌رود نرم‌افزار ما آن را تشخیص دهد و حذف کند و نتیجه‌ی آن این باشد که هیچ یک از علیرضا و شبنم به دیگری بدهکار نیست. در واقع برای این حالت خاص لازم نیست که شما کار خاصی را انجام دهید، در صورتیکه قسمت قبل را به خوبی پیاده‌سازی کرده باشید، این حالت نیز خودبه‌خود انجام می‌شود.

5. توصیه‌های نهایی

- دست شما باز است که هر طور که می‌خواهید قرارداد هوشمند خود را بنویسید فقط لازم است که توابع خواسته شده را حتماً پیاده‌سازی کنید. همچنین سعی کنید قرارداد هوشمند شما کمترین میزان حافظه و کمترین میزان انجام محاسبات را به خود اختصاص دهد، چرا که اینکار باعث می‌شود که استفاده از قرارداد شما ارزان‌تر تمام شود.
- از بدهی منفی حتماً خودداری کنید زیرا این کار پیچیدگی را افزایش می‌دهد.
- توصیه می‌شود که ابتدا قرارداد هوشمند خود را کامل کنید، سپس سراغ بخش کاربری بروید. در سایت `Remix` می‌توانید تمام توابع قرارداد را صدا بزنید و آن‌ها را تست کنید. (از طریق پنل سمت راست و پایین صفحه) و همچنین می‌توانید از قسمت `'Accounts'` بین آدرس‌های مختلف جابجا شوید. برای کپی کردن هریک از آدرس‌هایی که فکر می‌کنید لازم دارید، می‌توانید از آیکن کنار آن‌ها استفاده کنید. همچنین از طریق قسمت `'debug'` می‌توانید تراکنش‌های انجام شده را دیباگ کنید و با جزئیات بیشتری اتفاقات درون آن را ببینید. (توجه کنید که برای استفاده از این قسمت لازم است که از قسمت `Environment` اتصال خود به سرور محلی اتریوم را قطع کنید و به `JavaScript VM` وصل شوید)
- هر بار که سرور شبیه‌ساز شبکه اجرا می‌شود اکانت‌های متفاوتی ساخته می‌شود، در صورتیکه علاقه دارید هربار اکانت‌های یکسانی ساخته شود می‌توانید از دستور `ganache-cli -d` استفاده کنید.
- برای انجام این تمرین کامپیوتری به حجم زیادی از کدزنی نیاز ندارید. کد نوشته شده‌ی ما شامل حدود 40 خط در سمت سرور و 70 خط در سمت کاربر است.

- (6) در قسمت سمت کاربر هنگامی که شما از تابع `lookup` استفاده می کنید مقدار بازگشتی آن به فرمت `BigNumber` است که به فرم استاندارد با سه پارامتر `c, e, s` است، برای تبدیل آن به عدد معمولی می توانید از دستور `bn.toNumber()` و یا از عملگر `+` قبل از `bn` استفاده کنید که منظور از `bn` مقدار بازگشتی از تابع `lookup` است.
- (7) برای دیباگ کردن سمت کاربر می توانید از تابع آماده ی `log` و یا از تابع `console.log` استفاده کنید که تابع اول خروجی را در صفحه ی اصلی سایت نشان می دهد و تابع دوم خروجی را در قسمت `console` مرورگرتان نشان می دهد.
- (8) در صورت مشاهده ی خطایی مبنی بر عدم اتصال به `localhost:8545`، از درحال اجرا بودن شبیه ساز شبکه اتریوم (`ganache-cli`) مطمئن شوید.
- (9) در زبان `solidity` یک تابع کارآمد به نام `require` وجود دارد که می تواند برای چک کردن های مربوط به قسمت های امنیتی از آن استفاده کنید.

6. بخش امتیازی

در بخش قبل برای کامپایل کردن و ثبت کردن قرارداد بر روی شبکه ی بلاکچین از سایت `remix` استفاده کردیم، در این بخش قرار است ابزاری را معرفی کنیم که این کار را هوشمندانه تر و کاربردی تر انجام دهد. پکیجی به نام `truffle`. برای نصب آن کافیست که در ترمینال خود دستور `npm install -g truffle` را وارد کنید. سه تا از دستورهای ترافل که در این بخش از آن استفاده می کنیم، عبارتند از:

- `truffle compile`
- `truffle migrate`
- `truffle test`

نحوه ی کار ترافل بدین شکل است که لازم است سه پوشه با نام های `contracts`، `migrations` و `test` در پوشه ی اصلی که پروژه ما در آن است موجود باشد. در پوشه اول (`contracts`) کدهای `solidity` قرار می گیرد، قراردادهای اصلی که لازم است کامپایل و ثبت شوند. در پوشه دوم و سوم فایل های به فرمت `.js`. قرار می گیرند که اصطلاحاً اسکریپت نام دارند و به زبان `Node.js` هستند که شباهت زیادی به `JavaScript` دارد. فایل های پوشه ی `migrations` لازم است به صورتی باشند که اسم آنها با عدد شروع شود.

با اجرای دستور اول، قراردادهای موجود در پوشه اول کامپایل می شوند و یک پوشه به نام `build` ساخته می شود و چکیده ی قراردادهای به فرمت `.json`. در آن ثبت می شود، می توانید `abi` هر قرارداد را آن جا ببینید و در صورت لزوم آن را کپی کنید.

با اجرای دستور دوم، اسکریپت های موجود در پوشه دوم به ترتیب اسم توسط `Node.js` کامپایل و اجرا می شوند، این اسکریپت ها به نحوی نوشته می شوند که با اجرا شدن آنها، قراردادهای اصلی (موجود در پوشه اول) کامپایل و در صورت درستی در شبکه ثبت شوند. درواقع هدف اسکریپت های موجود در پوشه دوم انجام کاری است که ما تا الان با `remix` انجام می دادیم.

اجرای دستور سوم باعث می شود که تمام فایل های موجود در پوشه تست کامپایل و اجرا شوند. هدف از فایل های موجود در این پوشه این است که قراردادهای اصلی ما را در شبکه ثبت کند و سپس توابع آنها را صدا بزند و خروجی های آنها را نمایش دهد، در واقع با این کار به راحتی می توان چک کرد که آیا توابعی که ما در قراردادمان نوشته ایم به درستی کار می کنند یا نه.

چند نکته.

- I. لازم است در هنگام استفاده از `truffle` سرور `ganache-cli` در حال اجرا باشد.
- II. برای آشنایی بیشتر با `truffle` می‌توانید از دستور `truffle --help` استفاده کنید و با سایر قابلیت‌های آن آشنا شوید از جمله می‌توان به `truffle console` اشاره کرد که برای استفاده‌ی `developers` مناسب است.
- III. اگر در یک پوشه خالی دستور `truffle unbox` را اجرا کنید یک نمونه از یک پروژه انجام شده را همراه با سه پوشه اصلی و فایل‌های درون آن می‌توانید مشاهده کنید. حتما توصیه می‌شود که از این فایل‌ها کمک بگیرید.
- IV. با انجام نکته سوم خواهید دید که در پوشه `migrations` دوفایل با نام‌های `1_initial_migration.js` و `2_deploy_contracts.js` وجود دارد فایل اول را می‌توانید دقیقا به همان صورت استفاده کنید اما لازم است که در فایل دوم تغییراتی ایجاد کنید که با قرارداد هوشمند شما سازگار باشد. همچنین برای دیدن آدرس قرارداد ثبت شده (که در مراحل قبل آدرس آن را از `remix` کپی می‌کردید و به فایل `script.js` انتقال می‌دادید) می‌توانید از قطعه کد زیر کمک بگیرید.

```
deployer.deploy(BlockchainSplitwise).then(function() {  
    console.log("BlockchainSplitwise ADDRESS: " + BlockchainSplitwise.address);  
    console.log("\n\n-----\nDEPLOYMENT DONE!\n-----\n\n");  
});
```

- V. با انجام قسمت سوم همچنین خواهید دید که در پوشه‌ی `contracts` سه فایل موجود است، لازم است که فایل `Migrations.sol` را به همان شکل به پوشه‌ی متناظر در پروژه‌ی خود انتقال دهید.
 - VI. در آخر نیز اگر بعد از انجام قسمت سوم محتویات پوشه `test` را ببینید، خواهید دید که در اسکریپت نوشته شده به زبان جاوا اسکریپت، از مفهومی به نام `promise`، استفاده‌ی زیادی شده، همچنین از دستور `it()` استفاده می‌شود، توصیه می‌شود برای درک بهتر کدها، مقداری با این مفاهیم آشنا شوید.
- در این قسمت از شما خواسته می‌شود که سه پوشه اصلی را بسازید و فایل‌های لازم را در آن بنویسید. در مورد فایل موجود در پوشه `test`، لازم است که یک فایل به نام `test.js` بسازید و آن را کامل کنید در این فایل لازم است که کلیه توابعی که در قرارداد خود آورده اید تست شوند و از صحت اجرای آن‌ها اطمینان حاصل شود. میزان پیچیدگی تست‌ها به خودتان بستگی دارد، اما انتظار می‌رود که تست‌ها هوشمندانه و قابل اطمینان باشند. از آنجاییکه این بخش به صورت امتیازی است، نمره‌ی اختصاص داده شده به آن به صورت نسبی و در مقایسه با عملکرد بقیه دانشجویان خواهد بود. در ادامه دو قابلیت برداشت^{۱۴} و پرداخت^{۱۵} را به قرارداد هوشمندتان اضافه کنید. همچنین لازم است یک تابع جهت نمایش دادن موجودی قرارداد نوشته شود. تست‌های مربوط به این سه تابع را نیز به تست‌های قبلی اضافه کنید. توجه کنید که در مورد تابع برداشت لازم است از `modifier` مناسب استفاده کنید و فقط شخص سازنده قرارداد هوشمند اجازه برداشت داشته باشد.

¹⁴ Withdraw

¹⁵ Deposit

7. بارگزاری فایل ها

توجه کنید که برای این پروژه:

- 1- لازم است یک گزارش کتبی مختصر تهیه کنید. بطوریکه روند انجام، نتایج حاصله و حتی در صورت وجود، مشکلات موجود و نحوه برطرف کردن آنها در گزارش ارائه شود.
 - 2- لازم است یک ویدئو کوتاه (مشابه ویدئو آموزشی قرارداده شده در سایت) تهیه کنید و چند سناریو مختلف اجرا کنید و مراحل انجام کار، نتایج حاصله و کلیه کارهایی که برای این پروژه انجام داده اید به طور کامل توضیح دهید.
 - 3- در کنار گزارش کتبی و ویدئویی، ارسال کدها نیز الزامی می باشد.
- کلیه فایل ها و پوشه های خود را در قالب یک فایل فشرده به صورت StudentID.zip آپلود کنید.