



دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ) تمرین سری دهم

علی حیدری
استاد: سید صالح اعتمادی

مهلت ارسال: ۴ خرداد ۹۸

فهرست مطالب

۲	۱	آماده‌سازی
۲	۱.۱	نکات مورد توجه
۲	۲.۱	آماده‌سازی‌های اولیه
۲	۱.۲.۱	آماده‌سازی‌های مربوط به git
۳	۲.۲.۱	آماده‌سازی‌های مربوط به visual studio
۳	۲	پیاده‌سازی
۳	۱.۲	مقدمه و شرح سوال
۳	۲.۲	کلاس Vector
۳	۱.۲.۲	توضیحات کلی
۴	۲.۲.۲	پیاده‌سازی با انواع داده‌ی عام
۴	۳.۲.۲	پیاده‌سازی واسط‌ها
۴	۴.۲.۲	override کردن متد ToString()
۵	۵.۲.۲	سربارگذاری عمل‌گرها
۶	۶.۲.۲	نمایه‌گر
۶	۳.۲	کلاس Matrix
۶	۱.۳.۲	پیاده‌سازی با انواع داده‌ی عام
۷	۲.۳.۲	پیاده‌سازی واسط‌ها
۷	۳.۳.۲	override کردن متد ToString()
۸	۴.۳.۲	سربارگذاری عمل‌گرها
۹	۵.۳.۲	نمایه‌گر
۱۰	۴.۲	کلاس SquareMatrix
۱۰	۱.۴.۲	پیاده‌سازی
۱۰	۲.۴.۲	آزمون

۱۰	۳ ارسال
۱۰	۱.۳ مشاهده وضعیت اولیه فایل ها
۱۰	۲.۳ اضافه کردن فایل های تغییر یافته به stage
۱۱	۳.۳ commit کردن تغییرات انجام شده
۱۱	۴.۳ ارسال تغییرات انجام شده به Remote repository
۱۱	۵.۳ ساخت Pull Request
۱۱	۶.۳ ارسال Pull Request به بازبیننده

۱ آماده سازی

۱.۱ نکات مورد توجه

- توجه داشته باشید که برای کسب نمره ی قبولی درس کسب حداقل نصف نمره ی هر سری تمرین الزامی می باشد.
- مهلت ارسال پاسخ تمرین تا ساعت ۲۳:۵۹ روز اعلام شده است. توصیه می شود نوشتن تمرین را به روزهای نهایی موکول نکنید.
- هم کاری و هم فکری 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 تقلب گیرنده و تقلب دهنده نمره ی مردود برای درس در نظر گرفته خواهد شد.
- توجه داشته باشید که پاسخ ها و کدهای مربوط به هر مرحله را بایستی تا قبل از پایان زمان مربوط به آن مرحله، در سایت [Azure DevOps](#) (طبق توضیحات کارگاه ها و کلاس ها) بفرستید. درست کردن Pull request و Complete کردن Pull request و انتقال به شاخه ی 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 نمره ای به تمرین تعلق نخواهد گرفت.
- برای طرح سوال و پرسش و پاسخ از صفحه درس در [Quera](#) استفاده کنید.

۲.۱ آماده سازی های اولیه

قواعد نام گذاری تمرین را از جدول ۱ مطالعه کنید.

جدول ۱: قراردادهای نام گذاری تمرین

Naming conventions					
Branch	Directory	Solution	Project	Test Project	Pull Request
fb_A10	A10	A10	A10	A10Tests	HW10

۱.۲.۱ آماده سازی های مربوط به git

اگر چه در کارگاه git مفاهیم و روش کار با آن آموزش داده شد اما بار دیگر در این جا کارهایی را که باید در ابتدای تمرین انجام دهید را مرور می کنیم.

✓ ابتدا به شاخه ی master بروید.

```

1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A9)
2 $ git checkout master
3 Switched to branch 'master'
4 Your branch is up to date with 'origin/master'.
```

✓ تغییرات انجام شده در Remote Repository را دریافت کنید.

```

1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
2 $ git pull
3 remote: Azure Repos
4 remote: Found 8 objects to send. (90 ms)
5 Unpacking objects: 100% (8/8), done.
```

```

6 From https://9752XXXX.visualstudio.com/AP97982/_git/AP97982
7   e7fd3b5..2cc74de master -> origin/master
8 Checking out files: 100% (266/266), done.
9 Updating e7fd3b5..2cc74de
10 Fast-forward
11  .gitattributes          |      63 +
12  A9/A9.sln              |      37 +
13  A9/A9/A9.csproj        |      61 +
14  A9/A9/App.config       |       6 +
15  A9/A9/Program.cs       |      15 +
16  A9/A9/Properties/AssemblyInfo.cs |     36 +
17  .
18  .
19  .

```

✓ یک شاخه‌ی جدید با نام fb_A10 بسازید و تغییر شاخه دهید.

```

1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
2 $ git checkout -b fb_A10
3 Switched to a new branch 'fb_A10'
4 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
5 $

```

توصیه می‌شود پس از پیاده‌سازی هر کلاس تغییرات انجام شده را commit و push کنید.

۲.۲.۱ آماده‌سازی‌های مربوط به visual studio

ساختار فایل پایه‌ای که در اختیار شما قرار می‌گیرد به صورت زیر است:

```

1 A10
2 +---Project
3 |       Matrix.cs
4 |       SquareMatrix.cs
5 |       Vector.cs
6 |       _Vector.cs
7 |
8 \---ProjectTests
9       MatrixTests.cs
10      VectorTests.cs

```

در فایل پایه دو پوشه وجود دارد شما باید فایل(های) موجود در پوشه Project را به پروژه‌ی اصلی (A10) و فایل(های) موجود در پوشه ProjectTests را به پروژه‌ی تست (A10Tests) اضافه کنید.

۲ پیاده‌سازی

۱.۲ مقدمه و شرح سوال

هدف تمرین: پیاده‌سازی کلاس Matrix (ماتریس) با استفاده از مفاهیم IEnumerable ، IEquatable و سربارگذاری عملگرها نکته‌ای که در رابطه با این تمرین وجود دارد این است که می‌خواهیم با استفاده از مفهوم انواع داده‌ای عام و واسط‌ها کدی بنویسیم که قابل استفاده‌ی مجدد باشد و بتوانیم از یک منطق چندبار استفاده کنیم.

۲.۲ کلاس Vector

۱.۲.۲ توضیحات کلی

همان طور که می‌دانید هر ماتریس از تعدادی بردار تشکیل شده است پس برای پیاده‌سازی کلاس Matrix (ماتریس) باید کلاس Vector (بردار) را پیاده‌سازی کنیم. لازم است بعد از پیاده‌سازی کامل کلاس Vector کلیه تست‌های مربوط به این کلاس پاس شوند.

۲.۲.۲ پیاده‌سازی با انواع داده‌ی عام

کلاس Vector را با استفاده از انواع داده‌ی عام (Generics) پیاده‌سازی کنید. همان‌طور که می‌دانید این کار باعث می‌شود تا انعطاف‌پذیری کلاس ساخته شده بالا برود و بتوانید بر داری از هر نوع داده‌ای بسازید. مثلاً بر داری از `int` ها، `double` ها، `string` ها یا هر حتی کلاسی فرضی‌ای مانند `Cell`. فرض کنید می‌خواهیم بردارهای زیر را بسازیم:

$$\vec{v}_1 = [1 \ 2 \ 3 \ 4 \ 5]$$

$$\vec{v}_2 = [1/1 \ 2/2 \ 3/3 \ 4/4 \ 5/5]$$

کد مربوطه به صورت زیر خواهد بود:

```
1 Vector<int> v1 = new Vector<int>(5) { 1, 2, 3, 4, 5 };
2 Vector<double> v2 = new Vector<double>(5) { 1.1, 2.2, 3.3, 4.4, 5.5 };
```

۳.۲.۲ پیاده‌سازی واسط‌ها

واسط‌های زیر را برای کلاس Vector پیاده‌سازی کنید:

۱. `IEnumerable<_Type>`

با پیاده‌سازی این واسط می‌توان بر روی نوع داده‌ی عامی که در کلاس Vector وجود دارد پیمایش کرد.

۲. `IEquatable<Vector<_Type>>`

با پیاده‌سازی این واسط می‌توان هر شی از این کلاس با نوع داده‌ای مشخص را با شی دیگری از همین کلاس با همان نوع داده‌ای از لحاظ برابری مقایسه کرد.

۴.۲.۲ override کردن متد ToString()

متد `ToString()` به به گونه‌ای پیاده‌سازی کنید که هر شی از نوع کلاس Vector را به فرمتی که در مثال‌ها آمده نمایش دهد. برنامه‌ی نمونه:

```
1 using System;
2
3 namespace A10
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Vector<int>[] vectors = new[]
10             {
11                 new Vector<int>(5) {1, 2, 3, 4, 5},
12                 new Vector<int>(5) {1, 2, 0, 4, 5},
13                 new Vector<int>(6) {1, 2, 3, 4, 5, 6},
14             };
15
16             foreach (Vector<int> vector in vectors)
17             {
18                 Console.WriteLine(vector.ToString());
19                 Console.WriteLine("-----");
20             }
21         }
22     }
23 }
```

خروجی:

```

1 [1,2,3,4,5]
2 -----
3 [1,2,0,4,5]
4 -----
5 [1,2,3,4,5,6]
6 -----

```

۵.۲.۲ سربارگذاری عمل‌گرها

عمل‌گرهای `*`، `+`، `==` و `!=` را برای کلاس `Vector` سربارگذاری کنید.

۱. عمل‌گر `*` را به گونه‌ای پیاده‌سازی کنید که بتوان دو شی از کلاس `Vector` با نواع داده یک‌سان را به روش ضرب برداری در هم ضرب کرد. در صورت مهیا نبودن شرایط ضرب برداری از `Exception` مناسب استفاده کنید. دقت کنید که با توجه به اینکه برای نوع داده‌ای عام مثلاً `_Type` عملگر جمع یا ضرب لزوماً سربارگذاری نشده است. از این جهت کامپایلر اجازه جمع و ضرب این انواع را نمی‌دهد. برای رفع این اشکال می‌توانید از کلمه کلیدی `dynamic` استفاده کنید. جزئیات بیشتر روش استفاده از این کلمه کلیدی را جستجو کنید.

برنامه‌ی نمونه:

```

1 using System;
2
3 namespace A10
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Vector<int> v1 = new Vector<int>(5) { 1, 2, 3, 4, 5 };
10            Vector<int> v2 = new Vector<int>(5) { 5, 4, 3, 2, 1 };
11
12            Console.WriteLine(v1 * v2);
13        }
14    }
15 }

```

خروجی:

```

1 35

```

توضیحات:

$$\begin{aligned}
 \vec{v}_1 &= [1 \quad 2 \quad 3 \quad 4 \quad 5] \\
 \vec{v}_2 &= [5 \quad 4 \quad 3 \quad 2 \quad 1] \\
 \vec{v}_1 \cdot \vec{v}_2 &= 1 \times 5 + 2 \times 4 + 3 \times 3 + 4 \times 2 + 5 \times 1 = 35
 \end{aligned}$$

۲. عمل‌گر `+` را به گونه‌ای پیاده‌سازی کنید که بتوان دو شی از کلاس `Vector` با نواع داده یک‌سان را به روش جمع برداری در هم ضرب کرد. در صورت مهیا نبودن شرایط ضرب برداری از `Exception` مناسب استفاده کنید.

برنامه‌ی نمونه:

```

1 using System;
2
3 namespace A10
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Vector<int> v1 = new Vector<int>(5) { 1, 3, 2, 5, 4 };

```

```

10     Vector<int> v2 = new Vector<int>(5) {3, 1, 1, 6, 1};
11
12     Vector<int> v3 = v1 + v2;
13
14     Console.WriteLine(v3.ToString());
15 }
16 }
17 }

```

خروجی:

```
1 [4,4,3,11,5]
```

توضیحات:

$$\begin{aligned}
 \vec{v}_1 &= [1 \quad 3 \quad 2 \quad 5 \quad 4] \\
 \vec{v}_2 &= [3 \quad 1 \quad 1 \quad 6 \quad 1] \\
 \vec{v}_3 &= \vec{v}_1 + \vec{v}_2 \\
 &= [(1+3) \quad (3+1) \quad (2+1) \quad (5+6) \quad (4+1)] \\
 &= [4 \quad 4 \quad 3 \quad 11 \quad 5]
 \end{aligned}$$

۳. عملگر `==` را به گونه‌ای پیاده‌سازی کنید که بتوان برابری دو شی از کلاس `Vector` با نوع داده یک‌سان را بررسی کرد.

۴. عملگر `!=` را به گونه‌ای پیاده‌سازی کنید که بتوان نابرابری دو شی از کلاس `Vector` با نوع داده یک‌سان را بررسی کرد.

۶.۲.۲ نمایه‌گر

نمایه‌گر (Indexer) را برای کلاس `Vector` به گونه‌ای پیاده‌سازی کنید که بتوان با استفاده از آن به درایه‌های بردار بر اساس شماره‌ی درایه دسترسی داشت و آن‌ها را دریافت و یا مقداردهی کرد.

۳.۲ کلاس Matrix

۱.۳.۲ پیاده‌سازی با انواع داده‌ی عام

کلاس `Matrix` را با استفاده از انواع داده‌ی عام (Generics) پیاده‌سازی کنید. همان‌طور که می‌دانید این کار باعث می‌شود تا انعطاف‌پذیری کلاس ساخته شده بالا برود و بتوانید برداری از هر نوع داده‌ای بسازید. مثلاً برداری از `int` ها، `double` ها، `string` ها یا هر حتی کلاسی فرضی‌ای مانند `Cell`. لازم است بعد از پیاده‌سازی کامل کلاس `Matrix` کلیه تست‌های مربوط به این کلاس پاس شوند. فرض کنید می‌خواهیم ماتریس‌های زیر را بسازیم:

$$m_1 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 1 \end{bmatrix} \qquad m_2 = \begin{bmatrix} -1/1 & 2/5 \\ 2/4 & 1/8 \\ 1/3 & 2/0 \end{bmatrix}$$

کد مربوطه به صورت زیر خواهد بود:

```

1 Matrix<int> m1 = new Matrix<int>(2, 3)
2 {
3     new Vector<int>(3) { 1, 2, 1},
4     new Vector<int>(3) { 2, -1, 1},
5 };
6
7 Matrix<double> m2 = new Matrix<double>(3, 2)
8 {
9     new Vector<double>(2) { -1.1, 2.5},
10    new Vector<double>(2) { 2.4, 1.8},
11    new Vector<double>(2) { 1.3, 2.0}
12 };

```

۲.۳.۲ پیاده‌سازی واسط‌ها

واسط‌های زیر را برای کلاس `Matrix` پیاده‌سازی کنید:

۱. `IEnumerable<_Type>`

با پیاده‌سازی این واسط می‌توان بر روی نوع داده‌ی عامی که در کلاس `Matrix` وجود دارد پیمایش کرد.

۲. `IEquatable<Matrix<_Type>>`

با پیاده‌سازی این واسط می‌توان هر شی از این کلاس با نوع داده‌ای مشخص را با شی دیگری از همین کلاس با همان نوع داده‌ای از لحاظ برابری مقایسه کرد.

۳.۳.۲ override کردن متد `ToString()`

متد `ToString()` به به گونه‌ای پیاده‌سازی کنید که هر شی از نوع کلاس `Matrix` را به فرمتی که در مثال‌ها آمده نمایش دهد.
برنامه‌ی نمونه:

```
1 using System;
2
3 namespace A10
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Matrix<int>[] matrices = new[]
10             {
11                 new Matrix<int>(2, 3)
12                 {
13                     new Vector<int>(3) {1, 2, 1},
14                     new Vector<int>(3) {2, -1, 1},
15                 },
16
17                 new Matrix<int>(2, 3)
18                 {
19                     new Vector<int>(3) {1, 2, 1},
20                     new Vector<int>(3) {2, -1, 1},
21                 },
22
23                 new Matrix<int>(3, 3)
24                 {
25                     new Vector<int>(3) {1, 2, 1},
26                     new Vector<int>(3) {2, 0, 1},
27                     new Vector<int>(3) {2, 0, 1}
28                 },
29             };
30
31             foreach (Matrix<int> matrix in matrices)
32             {
33                 Console.WriteLine(matrix.ToString());
34                 Console.WriteLine("-----");
35             }
36         }
37     }
38 }
```

خروجی:

```
1 [
2 [1,2,1],
3 [2,-1,1]
4 ]
5 -----
```

```

6  [
7  [1,2,1],
8  [2,-1,1]
9  ]
10 -----
11 [
12 [1,2,1],
13 [2,0,1],
14 [2,0,1]
15 ]
16 -----

```

۴.۳.۲ سربارگذاری عملگرها

عملگرهای `*`، `+`، `==` و `!=` را برای کلاس `Matrix` سربارگذاری کنید.

۱. عملگر `*` را به گونه‌ای پیاده‌سازی کنید که بتوان دو شی از کلاس `Matrix` با نواع داده یک‌سان را به روش ضرب ماتریسی در هم ضرب کرد. در صورت مهیا نبودن شرایط ضرب ماتریسی از `Exception` مناسب استفاده کنید.

برنامه‌ی نمونه:

```

1  using System;
2
3  namespace A10
4  {
5      public class Program
6      {
7          public static void Main(string[] args)
8          {
9              Matrix<int> m1 = new Matrix<int>(2, 3)
10             {
11                 new Vector<int>(3) { 1, 2, 1},
12                 new Vector<int>(3) { 2, -1, 1},
13             };
14
15             Matrix<int> m2 = new Matrix<int>(3, 2)
16             {
17                 new Vector<int>(2) { -1, 2},
18                 new Vector<int>(2) { 2, 1},
19                 new Vector<int>(2) { 1, 2}
20             };
21
22             var m3 = m1 * m2;
23
24             Console.WriteLine(m3.ToString());
25
26         }
27     }
28 }

```

خروجی:

```

1  [
2  [4,6],
3  [-3,5]
4  ]

```

توضیحات:

$$m_1 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 1 \end{bmatrix} \quad m_2 = \begin{bmatrix} -1 & 2 \\ 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$m_3 = m_1.m_2 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 6 \\ -3 & 5 \end{bmatrix}$$

۲. عملگر + را به گونه‌ای پیاده‌سازی کنید که بتوان دو شی از کلاس Matrix با نواع داده یک‌سان را به روش جمع ماتریسی در هم ضرب کرد. در صورت مهیا نبودن شرایط ضرب ماتریسی از Exception مناسب استفاده کنید.

برنامه‌ی نمونه:

```
1 using System;
2
3 namespace A10
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Matrix<int> m1 = new Matrix<int>(2, 3)
10            {
11                new Vector<int>(3) { 1, 2, 1},
12                new Vector<int>(3) { 2, -1, 1},
13            };
14
15            Matrix<int> m2 = new Matrix<int>(2, 3)
16            {
17                new Vector<int>(3) { 0, 2, 1},
18                new Vector<int>(3) { 1, 4, 1},
19            };
20
21            var m3 = m1 + m2;
22
23            Console.WriteLine(m3.ToString());
24
25        }
26    }
27 }
```

خروجی:

```
1 [
2 [1,4,2],
3 [3,3,2]
4 ]
```

توضیحات:

$$m_1 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 1 \end{bmatrix} \quad m_2 = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 4 & 1 \end{bmatrix}$$

$$m_3 = m_1 + m_2 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 1 \\ 1 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 2 \\ 3 & 3 & 2 \end{bmatrix}$$

۳. عملگر == را به گونه‌ای پیاده‌سازی کنید که بتوان برابری دو شی از کلاس Matrix با نوع داده یک‌سان را بررسی کرد.

۴. عملگر != را به گونه‌ای پیاده‌سازی کنید که بتوان نابرابری دو شی از کلاس Matrix با نوع داده یک‌سان را بررسی کرد.

۵.۳.۲ نمایه‌گر

نمایه‌گر (Indexer) را برای کلاس Matrix به گونه‌ای پیاده‌سازی کنید که بتوان با استفاده از آن به درایه‌های ماتریس بر اساس شماره‌ی درایه دسترسی داشت و آن‌ها را دریافت و یا مقداردهی کرد.

۴.۲ کلاس SquareMatrix

۱.۴.۲ پیاده‌سازی

کلاس `SquareMatrix` را با استفاده از مفاهیم ارث‌بری پیاده‌سازی کنید. این کلاس باید تمام قابلیت‌های کلاس پدر را داشته باشد. راهنمایی: در این قسمت نیازی به زدن کد زیاد و عجیب غریبی نیست. اگر به مفاهیم ارث‌بری مسلط باشید پیاده‌سازی این بخش آسان است.

۲.۴.۲ آزمون

برای کلاس `SquareMatrix` تست‌هایی مشابه با کلاس `Matrix` بنویسید.

۳ ارسال

در اینجا یک‌بار دیگر ارسال تمرین را با هم مرور می‌کنیم:

۱.۳ مشاهده‌ی وضعیت اولیه‌ی فایل‌ها

ابتدا وضعیت فعلی فایل‌ها را مشاهده کنید:

```
1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
2 $ git status
3 On branch fb_A10
4 Untracked files:
5   (use "git add <file>..." to include in what will be committed)
6
7   A10/
8
9 nothing added to commit but untracked files present (use "git add" to track)
```

همان‌طور که مشاهده می‌کنید فولدر `A10` و تمام فایل‌ها و فولدرهای درون آن در وضعیت `Untracked` قرار دارند و همان‌طور که در خط آخر خروجی توضیح داده شده برای `commit` کردن آن‌ها ابتدا باید آن‌ها را با دستور `git add` وارد `stage` کنیم.

۲.۳ اضافه کردن فایل‌های تغییر یافته به stage

حال باید فایل‌ها و فولدرهایی را که در `stage` قرار ندارند را وارد `stage` کنیم. برای این کار از دستور `git add` استفاده می‌کنیم.

```
1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
2 $ git add A10/*
```

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
2 On branch fb_A10
3 Changes to be committed:
4   (use "git reset HEAD <file>..." to unstage)
5
6   new file:   A10/A10.sln
7   new file:   A10/A10/A10.csproj
8   new file:   A10/A10/App.config
9   new file:   A10/A10/Program.cs
10  new file:   A10/A10/Properties/AssemblyInfo.cs
11  new file:   A10/A10Tests/A10Tests.csproj
12  new file:   A10/A10Tests/Properties/AssemblyInfo.cs
13  new file:   A10/A10Tests/packages.config
14  .
15  .
16  .
```

همان‌طور که مشاهده می‌کنید فولدر `A10` و تمام فولدرها و فایل‌های درون آن (به جز فایل‌هایی که در `gitignore` معین کرده‌ایم) وارد `stage` شده‌اند.

۳.۳ commit کردن تغییرات انجام شده

در گام بعدی باید تغییرات انجام شده را commit کنیم. فراموش نکنید که فقط فایل‌هایی را می‌توان commit کرد که در stage قرار داشته باشند. با انتخاب یک پیام مناسب تغییرات صورت گرفته را commit می‌کنیم:

```
1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
2 $ git commit -m "Implement HW10"
3 [fb_A10 c1f21df] Implement HW10
4 15 files changed, 595 insertions(+)
5 create mode 100644 A10/A10.sln
6 create mode 100644 A10/A10/A10.csproj
7 create mode 100644 A10/A10/App.config
8 create mode 100644 A10/A10/Program.cs
9 create mode 100644 A10/A10/Properties/AssemblyInfo.cs
10 create mode 100644 A10/A10Tests/A10Tests.csproj
11 create mode 100644 A10/A10Tests/Properties/AssemblyInfo.cs
12 create mode 100644 A10/A10Tests/packages.config
13 .
14 .
15 .
```

۴.۳ ارسال تغییرات انجام شده به Remote repository

گام بعدی ارسال تغییرات انجام شده به Remote Repository است.

```
1 Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
2 $ git push origin fb_A10
3 Enumerating objects: 25, done.
4 Counting objects: 100% (25/25), done.
5 Delta compression using up to 8 threads
6 Compressing objects: 100% (22/22), done.
7 Writing objects: 100% (25/25), 9.56 KiB | 890.00 KiB/s, done.
8 Total 25 (delta 4), reused 0 (delta 0)
9 remote: Analyzing objects... (25/25) (5 ms)
10 remote: Storing packfile... done (197 ms)
11 remote: Storing index... done (84 ms)
12 To https://9752XXXX.visualstudio.com/AP97982/_git/AP97982
13 * [new branch]      fb_A10 -> fb_A10
```

۵.۳ ساخت Pull Request

با مراجعه به سایت [Azure DevOps](#) یک Pull Request جدید با نام HW10 بسازید به طوری که امکان merge کردن شاخه‌ی fb_A10 را بر روی شاخه‌ی master را بررسی کند. (این کار در صورتی انجام می‌شود که کد شما کامپایل شود و همچنین تست‌های آن پاس شوند) در نهایت با انتخاب گزینه‌ی set auto complete در صفحه‌ی Pull Request مربوطه تعیین کنید که در صورت وجود شرایط merge این کار انجام شود. دقت کنید که گزینه‌ی Delete source branch نباید انتخاب شود.

۶.۳ ارسال Pull Request به بازبیننده

در نهایت Pull Request ساخته شده را برای بازبینی، با بازبیننده‌ی خود به اشتراک بگذارید.