دانشکده‌ی مهندسی کامپیوتر

برنامه‌سازی پیشرفته (سی‌شارپ)
تمرین سری دهم

علی حیدری
استاد: سید صالح اعتمادی

مهلت ارسال: ۴ خرداد ۹۸

فهرست مطالب

۱ آماده‌سازی ۲
۱.۱ نکات مورد توجه ۲
۲.۱ آماده‌سازی‌های اولیه ۲
۱.۲.۱ آماده‌سازی‌های مربوط به git ۲
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 ۳
۲.۲.۲ پیاده‌سازی با انواع داده‌ی عام ۴
۳.۲.۲ پیاده‌سازی واسط‌ها ۴
۴.۲.۲ override کردن متد `ToString()` ۴
۵.۲.۲ سربارگذاری عمل‌گرها ۵
۶.۲.۲ نمایه‌گر ۶
۳.۲ کلاس Matrix ۶
۱.۳.۲ پیاده‌سازی با انواع داده‌ی عام ۶
۲.۳.۲ پیاده‌سازی واسط‌ها ۷
۳.۳.۲ override کردن متد `ToString()` ۷
۴.۳.۲ سربارگذاری عمل‌گرها ۸
۵.۳.۲ نمایه‌گر ۹
۴.۲ کلاس SquareMatrix ۱۰
۱.۴.۲ پیاده‌سازی ۱۰
۲.۴.۲ آزمون ۱۰

۳ ارسال ۱۰
۱.۳ مشاهده‌ی وضعیت اولیه‌ی فایل‌ها . . . ۱۰
۲.۳ اضافه کردن فایل‌های تغییر یافته به stage . . . ۱۰
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 . . . ۱۱

۱ آماده‌سازی

۱.۱ نکات 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 ازپایان 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.
- برای طرح سوال و پرسش و پاسخ از صفحه درس در Quera استفاده کنید.

۲.۱ آماده‌سازی‌های اولیه

قواعد نام‌گذاری تمرین را از جدول ۱ مطالعه کنید.

جدول ۱: قراردادهای نام‌گذاری تمرین

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

۱.۲.۱ آماده‌سازی‌های مربوط به git

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

✓ ابتدا به شاخه‌ی `master` بروید.

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

✓ تغییرات انجام‌شده در `Remote Repository` را دریافت کنید.

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

```
From https://9752XXXX.visualstudio.com/AP97982/_git/AP97982
   e7fd3b5..2cc74de  master               -> origin/master
Checking out files: 100% (266/266), done.
Updating e7fd3b5..2cc74de
Fast-forward
 .gitattributes                                |    63 +
 A9/A9.sln                                     |    37 +
 A9/A9/A9.csproj                               |    61 +
 A9/A9/App.config                              |     6 +
 A9/A9/Program.cs                              |    15 +
 A9/A9/Properties/AssemblyInfo.cs              |    36 +
 .
 .
 .
```

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

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (master)
$ git checkout -b fb_A10
Switched to a new branch 'fb_A10'
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
$
```

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:

```
A10
+---Project
|       Matrix.cs
|       SquareMatrix.cs
|       Vector.cs
|
\---ProjectTests
        MatrixTests.cs
        VectorTests.cs
```

در فایل پایه دو پوشه وجود دارد شما باید فایل(های) موجود در پوشه‌ی Project را به پروژه‌ی اصلی (A10) و فایل(های) موجود در پوشه‌ی 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 بزنیم 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 برداری از هر نوع داده‌ای بسازید. مثلا برداری از `int` ها، `double` ها، `string` ها یا هر حتی کلاسی فرضی‌ای مانند `Cell` .
فرض کنید می‌خواهیم بردارهای زیر را بسازیم:

$$\vec{v_1} = [۱ \quad ۲ \quad ۳ \quad ۴ \quad ۵]$$
$$\vec{v_2} = [۱/۱ \quad ۲/۲ \quad ۳/۳ \quad ۴/۴ \quad ۵/۵]$$

کد مربوطه به صورت زیر خواهد بود:

```
Vector<int> v1 = new Vector<int>(5) { 1, 2, 3, 4, 5 };
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:

```
using System;

namespace A10
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Vector<int>[] vectors = new[]
            {
                new Vector<int>(5) {1, 2, 3, 4, 5},
                new Vector<int>(5) {1, 2, 0, 4, 5},
                new Vector<int>(6) {1, 2, 3, 4, 5, 6},
            };

            foreach (Vector<int> vector in vectors)
            {
                Console.WriteLine(vector.ToString());
                Console.WriteLine("---------------");
            }
        }
    }
}
```

خروجی:

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

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 مثلا `_Type` عملگر جمع یا ضرب لزوما سربارگزاری 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:

```
using System;

namespace A10
{
    public class Program
    {
        public static void Main(string[] args)
        {
          Vector<int> v1 = new Vector<int>(5) { 1, 2, 3, 4, 5 };
          Vector<int> v2 = new Vector<int>(5) { 5, 4, 3, 2, 1 };

          Console.WriteLine(v1 * v2);
        }
    }
}
```

خروجی:

```
35
```

توضیحات:

$$\vec{v_1} = [1 \quad 2 \quad 3 \quad 4 \quad 5]$$
$$\vec{v_2} = [5 \quad 4 \quad 3 \quad 2 \quad 1]$$
$$\vec{v_1}.\vec{v_2} = 1 \times 5 + 2 \times 4 + 3 \times 3 + 4 \times 25 \times 1 = 35$$

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:

```
using System;

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

```
            Vector<int> v2 = new Vector<int>(5) {3, 1, 1, 6, 1};

            Vector<int> v3 = v1 + v2;

            Console.Write(v3.ToString());
        }
    }
}
```

خروجی:

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

توضیحات:

$$\begin{aligned}\vec{v_۱} &= [۱ \quad ۳ \quad ۲ \quad ۵ \quad ۴]\\ \vec{v_۲} &= [۳ \quad ۱ \quad ۱ \quad ۶ \quad ۱]\\ \vec{v_۳} &= \vec{v_۱} + \vec{v_۲}\\ &= [(۱+۳) \quad (۳+۱) \quad (۲+۱) \quad (۵+۶) \quad (۴+۱)]\\ &= [۴ \quad ۴ \quad ۳ \quad ۱۱ \quad ۵]\end{aligned}$$

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. مثلا 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:

$$m_۱ = \begin{bmatrix} ۱ & ۲ & ۱ \\ ۲ & -۱ & ۱ \end{bmatrix} \qquad m_۲ = \begin{bmatrix} -۱/۱ & ۲/۵ \\ ۲/۴ & ۱/۸ \\ ۱/۳ & ۲/۰ \end{bmatrix}$$

کد مربوطه به صورت زیر خواهد بود:

```
Matrix<int> m1 = new Matrix<int>(2, 3)
{
    new Vector<int>(3) { 1, 2, 1},
    new Vector<int>(3) { 2, -1, 1},
};

Matrix<double> m2 = new Matrix<double>(3, 2)
{
    new Vector<double>(2) { -1.1, 2.5},
    new Vector<double>(2) { 2.4, 1.8},
    new Vector<double>(2) { 1.3, 2.0}
};
```

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:

```
using System;

namespace A10
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Matrix<int>[] matrices = new[]
            {
                new Matrix<int>(2, 3)
                {
                    new Vector<int>(3) {1, 2, 1},
                    new Vector<int>(3) {2, -1, 1},
                },

                new Matrix<int>(2, 3)
                {
                    new Vector<int>(3) {1, 2, 1},
                    new Vector<int>(3) {2, -1, 1},
                },

                new Matrix<int>(3, 3)
                {
                    new Vector<int>(3) {1, 2, 1},
                    new Vector<int>(3) {2, 0, 1},
                    new Vector<int>(3) {2, 0, 1}
                },
            };

            foreach (Matrix<int> matrix in matrices)
            {
                Console.WriteLine(matrix.ToString());
                Console.WriteLine("---------------");
            }
        }
    }
}
```

خروجی:

```
[
[1,2,1],
[2,-1,1]
]
---------------
```

```
[
[1,2,1],
[2,-1,1]
]
---------------
[
[1,2,1],
[2,0,1],
[2,0,1]
]
---------------
```

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:

```
using System;

namespace A10
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Matrix<int> m1 = new Matrix<int>(2, 3)
            {
                new Vector<int>(3) { 1, 2, 1},
                new Vector<int>(3) { 2, -1, 1},
            };

            Matrix<int> m2 = new Matrix<int>(3, 2)
            {
                new Vector<int>(2) { -1, 2},
                new Vector<int>(2) { 2, 1},
                new Vector<int>(2) { 1, 2}
            };

            var m3 = m1 * m2;

            Console.WriteLine(m3.ToString());

        }
    }
}
```

خروجی:

```
[
[4,6],
[-3,5]
]
```

توضیحات:

$$m_۱ = \begin{bmatrix} ۱ & ۲ & ۱ \\ ۲ & -۱ & ۱ \end{bmatrix} \qquad m_۲ = \begin{bmatrix} -۱ & ۲ \\ ۲ & ۱ \\ ۱ & ۲ \end{bmatrix}$$

$$m_۳ = m_۱.m_۲ = \begin{bmatrix} ۱ & ۲ & ۱ \\ ۲ & -۱ & ۱ \end{bmatrix} \begin{bmatrix} -۱ & ۲ \\ ۲ & ۱ \\ ۱ & ۲ \end{bmatrix} = \begin{bmatrix} ۴ & ۶ \\ -۳ & ۵ \end{bmatrix}$$

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:

```
using System;

namespace A10
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Matrix<int> m1 = new Matrix<int>(2, 3)
            {
                new Vector<int>(3) { 1, 2, 1},
                new Vector<int>(3) { 2, -1, 1},
            };

            Matrix<int> m2 = new Matrix<int>(2, 3)
            {
                new Vector<int>(3) { 0, 2, 1},
                new Vector<int>(3) { 1, 4, 1},
            };

            var m3 = m1 + m2;

            Console.WriteLine(m3.ToString());

        }
    }
}
```

خروجی:

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

توضیحات:

$$m_۱ = \begin{bmatrix} ۱ & ۲ & ۱ \\ ۲ & -۱ & ۱ \end{bmatrix} \qquad m_۲ = \begin{bmatrix} ۰ & ۲ & ۱ \\ ۱ & ۴ & ۱ \end{bmatrix}$$

$$m_۳ = m_۱ + m_۲ = \begin{bmatrix} ۱ & ۲ & ۱ \\ ۲ & -۱ & ۱ \end{bmatrix} + \begin{bmatrix} ۰ & ۲ & ۱ \\ ۱ & ۴ & ۱ \end{bmatrix} = \begin{bmatrix} ۱ & ۴ & ۲ \\ ۳ & ۳ & ۲ \end{bmatrix}$$

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:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
$ git status
On branch fb_A10
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    A10/

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.

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

حال دوباره وضعیت فایل‌ها و فولدرها را مشاهده می‌کنیم:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
On branch fb_A10
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   A10/A10.sln
    new file:   A10/A10/A10.csproj
    new file:   A10/A10/App.config
    new file:   A10/A10/Program.cs
    new file:   A10/A10/Properties/AssemblyInfo.cs
    new file:   A10/A10Tests/A10Tests.csproj
    new file:   A10/A10Tests/Properties/AssemblyInfo.cs
    new file:   A10/A10Tests/packages.config
    .
    .
    .
```

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:

```
Ali@DESKTOP-GS7PR56 MINGW64 /c/git/AP97982 (fb_A10)
$ git commit -m "Implement HW10"
[fb_A10 c1f21df] Implement HW10
 15 files changed, 595 insertions(+)
 create mode 100644 A10/A10.sln
 create mode 100644 A10/A10/A10.csproj
 create mode 100644 A10/A10/App.config
 create mode 100644 A10/A10/Program.cs
 create mode 100644 A10/A10/Properties/AssemblyInfo.cs
 create mode 100644 A10/A10Tests/A10Tests.csproj
 create mode 100644 A10/A10Tests/Properties/AssemblyInfo.cs
 create mode 100644 A10/A10Tests/packages.config
.
.
.
```

۴.۳ ارسال تغییرات انجام شده به Remote repository

گام بعدی ارسال تغییرات انجام شده به `Remote Repository` است.

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