CBrute

این کتابخانه خیلی پیچیده نیست و کار راه بندازه. کار من رو که راه انداخت.

توسعه این کتابخانه متوقف شده و حال و حوصله ادامه دادن ندارم(تو #C البته).

فهرست

مقدمه

در این کتاب قصد دارم نحوه استفاده از کتابخانه CBrute را به شما آموزش بدهم. ابزار های زیادی برای ساخت پسورد لیست‌های متنوع وجود دارد؛ اما باید در نظر داشته باشید که CBrute فقط برای ساخت پسورد لیست مورد استفاده قرار نمی‌گیرد(هر چند که هدف اصلی ساخت رمزهای عبور است). در CBrute به جای این‌که منتظر بمانید تا پسورد لیست تولید شده و سپس شروع به کرک کردن کنید، می‌توانید بلافاصله بعد از تولید هر پسورد از آن استفاده کرده و در صورت لزوم فعالیت را متوقف کنید و سپس بعداً آن را ادامه دهید.

CBrute از قابلیت مولتی تردینگ نیز پشتیبانی کرده و می‌توانید سرعت کار خود را با Thread ها بیشتر کنید. می‌توانید CBrute را یک ابزار کمکی در نظر بگیرید که به وسیله این ابزار، می‌توانید ابزارهای پیچیده‌تری برای کرک کردن رمزها بسازید.

به هر حال، قبل از این‌که شروع به مطالعه این کتاب کنید لطفاً به نکات زیر توجه کنید:

- تصور من این است که شما یک برنامه نویس C# هستید و برنامه نویسی را تا حد قابل قبولی درک کرده‌اید.
- دقت کنید که در این کتاب الگوریتم‌های مورد استفاده در کتابخانه CBrute را توضیح نمی‌دهم(حوصله ندارم فعلاً).
- انسان‌ها ممکن است خطا کنند. در صورتی‌که ایرادی در کتاب مشاهده کردید، حلال کنید. حتی اگه خطای فنی پیدا کردید! مثلاً اگه یک جایی نوشته بود که پایتون سریعتر از C هستش به دل نگیرید(البته تا این حدم کتاب داغون نیست! فقط خواستم منظورم رو برسونم)

هشدار: در برخی از قسمت‌های این کتاب از کلمه ترد استفاده شده. منظور از ترد در این کتاب بیشتر نمونه‌هایی از کلاس BruteForce است که در ترد دیگری اجرا می‌شوند. مثلاً وقتی می‌گوییم که یک ترد توانست رمز را پیدا کند یعنی یک نمونه از کلاس BruteForce موجود در آن ترد توانست رمز را پیدا کند.

CBrute چیست؟

CBrute یک کتابخانه برای تولید پسوردهای پیچیده است که به زبان C# نوشته شده و استفاده کردن از این کتابخانه نیز ساده است. با استفاده از این کتابخانه می‌توانید ابزارهای مختلفی را برای کرک کردن رمزهای عبور طراحی کنید.

برای مثال، تصور کنید که شما با یک حالت خاص برای حدس زدن رمز عبور مواجه شده‌اید که در آن مطمئن هستید که خانه دوم همیشه یکی از حالت های {1,"abc",4} است و می‌دانید که خانه یکی مانده به آخر نیز همیشه {"pass"} است. همچنین می‌دانید که رمز مورد نظر شما حداقل 4 خانه و حداکثر 8 خانه است. حتی ممکن است بدانید که رمز هیچوقت 6 خانه ندارد و طول رمزهای قابل تولید همیشه یکی از اعضای مجموعه {4,5,7,8} است.

شاید کمی پیچیده به نظر برسد ولی در CBrute می‌توانید چنین پسورد لیستی را تولید کنید. شما می‌توانید بدون ذخیره کردن رمزهای تولید شده، بلافاصله بعد از تولید هر رمز از آن استفاده کنید. حتی می‌توان این کار ها را به صورت مولتی تردینگ انجام داده و نظارت دقیقی بر کل فعالیت تولید رمزها داشته باشید؛ به صورتی‌که حتی بتوانید در فرایند تولید رمزها وقفه ایجاد کنید، ازسرگیری کنید یا آن را ذخیره کرده و سپس متوقف کنید تا بعداً ادامه آن را انجام دهید. البته باید بدانید در حالتی‌که از تردها استفاده می‌کنید نمی‌توانید از قابلیت extraLengths استفاده کنید.

شاید وقتی اسم جایگشت به گوشتان می‌خورد ابتدا تصور کنید که نمی‌توان برای جایشگت یک رشته(در CBrute رشته‌ای از کاراکترها را فراموش کنید و به جای آن، یک آرایه‌ای از اشیاء را تصور کنید) کمترین طول و بیشترین طول پسورد را تعیین کرد؛ ولی در CBrute این نیز شدنی است.(کافی است به مجموعه و زیر مجموعه گیری فکر کنید :)

به طور کلی من سعی کرده‌ام که این کتابخانه را طوری طراحی کنم که علاوه بر ساده بودن، منعطف نیز باشد و امیداورم موفق عمل کرده باشم. هر چند از نظر سرعت نمی‌توان آن را با ابزارهایی که با C نوشته شده‌اند مقایسه کرد...

چرا #C؟

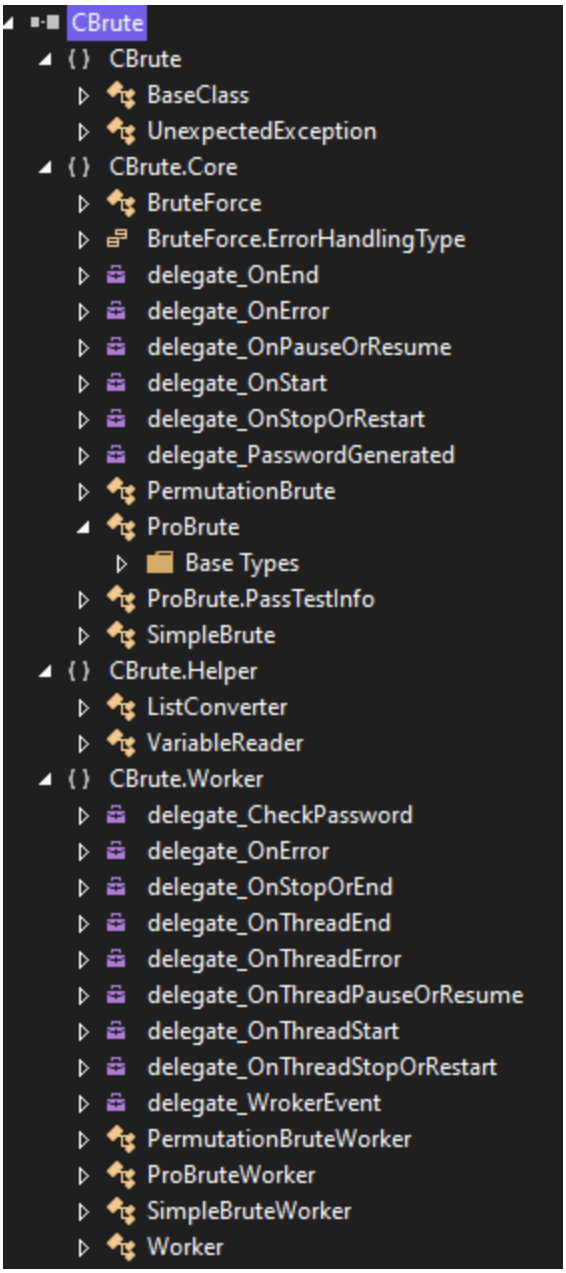
احتملاً از خودتان بپرسید چرا از یک زبان مانند C یا C++ استفاده نکردی؟ زبان هایی مانند C# یا JAVA برای کرک کردن مناسب نیستند!

در پاسخ باید بگویم که مهم‌ترین دلیلی که من از C# استفاده کردم این است که هنوز به اندازه کافی با زبان C++ آشنا نیستم و نیاز به تجربه بیشتری دارم. در زبان C++ خبری از مدیریت حافظه خودکار نیست و باید با خیلی حواس جمع بود تا یک وقت یک کد آسیب پذیر ننویسید. این در حالی است که در C# کدها و حافظه توسط CLR مدیریت می‌شوند.

دلیل دیگری که از زبان C# برای تولید پسوردها استفاده کردم این است که پیاده‌سازی الگوریتم‌ها در این زبان برای من ساده‌تر است.

ساختار کتابخانه CBrute

کتابخانه CBrute یک کتابخانه کوچک است و ساختار پیچیده‌ای ندارد. در تصویر زیر میتوانید ساختار این کتابخانه را مشاهده کنید:



فضای نام CBrute

این فضای نام کل کتابخانه را تشکیل می‌دهد. شامل کلاس‌هایی است که نمی‌توان آن‌ها را در دسته‌های خاصی قرار داد.

کلاس CBrute.BaseClass

تمامی کلاس‌های موجود در CBrute، از این کلاس مشتق می‌شوند(به جز خود کلاس :). برای مباحث مرتبط با چند ریختی می‌تواند مفید باشد. این کلاس فقط یک خاصیت دارد.

خاصیت های کلاس CBrute.BaseClass

BaseClass.Tag

```
public object Tag { get; set; }
//    Member of CBrute.BaseClass
```

این خاصیت از نوع Object است. از آنجایی که object می‌تواند به هر نوع داده‌ای اشاره کند، می‌توانید هر مقداری که دلتان می‌خواهد را در این خاصیت قرار دهید.

این خاصیت بیشتر در زمینه مولتی تردینگ کاربرد دارد و با استفاده از آن می‌توانید یک object خاص را به هر ترد اختصاص دهید. به عنوان مثال، هر ترد می‌تواند یک فایل را باز کند و روی آن کار کند. با استفاده از این خاصیت می‌توانید به یک استریم اشاره کرده و روی آن کار کنید.

کلاس CBrute.UnexpectedException

زمانی که خطای غیر منتظره‌ای رخ بدهد، از این کلاس برای پرتاب استثناء استفاده می‌کنیم.

سازنده های کلاس CBrute.UnexpectedException

UnexpectedException.UnexpectedException(int, string)

```
public UnexpectedException(int code, string file)
//    Member of CBrute.UnexpectedException
```

سازنده کلاس UnexpectedException.

پارامتر ها

- code: کد خطا را مشخص می‌کند.
- file: سورس کدی که باعث ایجاد خطا شده.

اطلاعات بیشتر

زمانی که خطایی رخ بدهد که که اصلاً انتظارش را نداریم، این استثناء پرتاب می‌شود. با استفاده از کدی که در اختیارتان قرار می‌دهد می‌توانید متوجه شوید که خطا کجا رخ داده و دلیلش چیست.

کد ها

کد شماره 0x1: به این معنی است که در فایل SimpleBrute.cs و در تابع GetPosByPass یک خطا رخ داده است. دلیل این خطا این است:

عنصری در پسورد وجود دارد که در TestArray وجود ندارد. در صورتی که با استفاده از تابع getPosByPassE باید این حالت مدیریت می‌شد.

کد شماره 0x2: به این معنی است که در فایل SimpleBrute.cs و در تابع StartBrute یک خطا رخ داده است. دلیل این خطا این است:

زمانی که شروع به تولید رمزها می‌کنیم، انتظار داریم که بعد از اتمام تولید رمزها و قبل از خارج شدن از حلقه‌ای که در حال تولید رمزها است، با استفاده از رویداد OnEnd به کار تابع خاتمه دهیم. وقتی این خطا رخ دهد به این معنی است که ما از حلقه‌ای که در حال تولید رمزها است خارج شده‌ایم و رویداد OnEnd هم رخ نداده است.

کد شماره 0x3: به این معنی است که در فایل ProBrute.cs و در تابع GetPassByPos یک خطا رخ داده است. دلیل این خطا این است:

زمانی که قصد داریم از طریق موقعیت یک پسورد به خود آن پسورد برسیم، ابتدا باید بدانیم رمزی که در آن موقعیت وجود دارد طولش چقدر است. این کار با استفاده از تابع getPassLen انجام می‌شود. طبق بررسی‌هایی که در تابع getPassByPosE در فایل ProBrute.cs انجام می‌شود، انتظار نداریم که تابع getPassLen یک عدد منفی را برگرداند(یعنی اینکه نتواند طول یک رمز در یک موقعیت را تشخیص دهد). اگر getPassLen یک عدد کمتر از 0 برگرداند، خطایی با کد 0x3 رخ می‌دهد.

کد شماره 0x4: به این معنی است که در فایل ProBrute.cs و در تابع StartBrute یک خطا رخ داده است. دلیل این خطا این است:

زمانی که شروع به تولید رمزها می‌کنیم، انتظار داریم که بعد از اتمام تولید رمزها و قبل از خارج شدن از حلقه‌ای که در حال تولید رمزها است، با استفاده از رویداد OnEnd به کار تابع خاتمه دهیم. وقتی این خطا رخ دهد به این معنی است که ما از حلقه‌ای که در حال تولید رمزها است خارج شده‌ایم و رویداد OnEnd هم رخ نداده است.

کد شماره 0x5: به این معنی است که در فایل PermutationBrute.cs و در تابع StartBrute یک خطا رخ داده است. دلیل این خطا این است:

زمانی که شروع به تولید رمزها می‌کنیم، انتظار داریم که بعد از اتمام تولید رمزها و قبل از خارج شدن از حلقه‌ای که در حال تولید رمزها است، با استفاده از رویداد OnEnd به کار تابع خاتمه دهیم. وقتی این خطا رخ دهد به این معنی است که ما از حلقه‌ای که در حال تولید رمزها است خارج شده‌ایم و رویداد OnEnd هم رخ نداده است.

فضای نام CBrute.Helper

شامل دو کلاس کاربردی است که در فرایند تولید رمزهای عبور می‌توانند مفید باشند.

کلاس CBrute.Helper.ListConverter

یک کلاس استاتیک که شامل یک سری از توابع است که برای تبدیل، ترکیب و تکه‌تکه کردن آرایه‌ها به کار می‌روند. بیشتر زمانی کاربرد دارد که قصد دارید که یک آریه از objectها را به یک رشته واحد تبدیل کنید. در CBrute پسوردها به صورت آرایه‌ای از objectها تولید می‌شوند. اکثر توابع موجود در این کلاس به صورت Extension methods هستند.

توابع کلاس CBrute.Helper.ListConverter

ListConverter.Append(object[], object[])

```
public static object[] Append(this object[] array, object[] newArray)
//    Member of CBrute.Helper.ListConverter
```

با استفاده از این تابع می‌توانید یک آرایه را به انتهای یک آرایه دیگر اضافه کنید.

پارامتر ها

- array: آرایه‌ای که قصد دارید newArray را به انتهای آن اضافه کنید.
- newArray: آرایه‌ای که به انتهای array اضافه می‌شود.

مقادیری که برمی‌گرداند

- یک آرایه جدید که حاصل ترکیب array و newArray است را برمی‌گرداند.

مثال(ListConverter.Append(object[], object[]))

این مثال با ترکیب دو آرایه با یکدیگر، روش استفاده از تابع Append را به شما آموزش می‌دهد.

```
// .Net 6
using CBrute.Helper;
object[] array1 = { "C",'B' };
object[] array2 = { "rut", 'e' };
object[] result = array1.Append(array2);
foreach (object o in result) Console.Write(o);
Console.ReadKey();
```

خروجی:

```
CBrute
```

ListConverter.ConvertObjectArrayToString(object[])

```
public static string ConvertObjectArrayToString(this object[] arr)
//    Member of CBrute.Helper.ListConverter
```

با استفاده از این تابع می‌توانید یک آرایه از objectها را به یک رشته واحد تبدیل کنید. این آرایه با عناصر null مشکلی ندارد.

پارامتر ها

- arr: آرایه‌ای که قصد دارید آن را به رشته تبدیل کنید.

مقادیری که برمی‌گرداند

- یک رشته که حاصل ترکیب عناصر arr به صورت متوالی است. اگر عنصری null باشد، در رشته به صورت <<null>> نمایش داده می‌شود.

اطلاعات بیشتر

زمانی که رمزی تولید می‌شود، می‌توانید با استفاده از این آرایه آن را به یک رشته تبدیل کنید تا در فرایند مورد نظر از آن استفاده کنید. پیشنهاد می‌شود که همیشه از این تابع استفاده کنید.

مثال(ListConverter.ConvertObjectArrayToString(object[]))

مثال قبلی را با اندکی تغییر در کد زیر مشاهده می‌کنید:

```
// .Net 6
using CBrute.Helper;
object[] array1 = { "C",'B' };
object[] array2 = { "rut", 'e', null! };// Changes
object[] result = array1.Append(array2);
Console.Write(result.ConvertObjectArrayToString());// Changes
Console.ReadKey();
```

خروجی:

```
CBrute<<null>>
```

ListConverter.ConvertStringArrayToString(object[])

```
public static string ConvertStringArrayToString(this object[] arr)
//    Member of CBrute.Helper.ListConverter
```

با استفاده از این تابع می‌توانید یک آرایه از objectها را به یک رشته واحد تبدیل کنید. این آرایه با عناصر null مشکل دارد.

پارامتر ها

- arr: آرایه‌ای که قصد دارید آن را به رشته تبدیل کنید.

مقادیری که برمی‌گرداند

- یک رشته که حاصل ترکیب عناصر arr به صورت متوالی است.

اطلاعات بیشتر

زمانی که رمزی تولید می‌شود، می‌توانید با استفاده از این تابع آن را به یک رشته تبدیل کنید تا در فرایند مورد نظر از آن استفاده کنید.

ListConverter.ConvertToObjectArray(string[])

```
public static object[] ConvertToObjectArray(this string[] arr)
//    Member of CBrute.Helper.ListConverter
```

برای تبدیل یک آرایه از رشته‌ها به یک آرایه از objectها استفاده می‌شود. هرچند با استفاده از تابع الحاقی cast نیز می‌توانید چنین کاری را انجام دهید... به هر حال پیاده سازی شده، گیر ندید!

پارامتر ها

- arr: آرایه‌ای که قصد دارید آن را به یک آرایه از نوع object تبدیل کنید.

مقادیری که برمی‌گرداند

- یک آرایه از objectها که هر عنصر در آن معادل آدرس عنصر متناظرش در آریه arr است.

اطلاعات بیشتر

از آنجایی که تمامی کلاس‌های CBrute که کار تولید رمزها را انجام می‌دهند فقط از Object arrayها پشتیبانی می‌کنند، این تابع می‌تواند مفید باشد. جهت مشاهده مثال به قسمت بعدی مراجعه کنید.

ListConverter.ConvertToStringArray(object[])

```
public static string[] ConvertToStringArray(this object[] arr)
//    Member of CBrute.Helper.ListConverter
```

این تابع برای تبدیل یک آرایه از نوع object به یک آرایه از نوع string استفاده می‌شود.

پارامتر ها

- arr: آرایه‌ای که قصد دارید آن را به یک آرایه از نوع string تبدیل کنید.

مقادیری که برمی‌گرداند

- یک آرایه از stringها که هر عنصر در آن معادل اجرای تابع ToString در اعضای آریه arr است.

مثال(ListConverter.ConvertToStringArray(object[]))

در مثال زیر می‌توانید روش استفاده از هر دو تابع قبل را مشاهده کنید:

```
// .Net 6
using CBrute.Helper;
object[] objects = { new Tuple<string>("item1"), "null", 5, 54.34 };
string[] strings = objects.ConvertToStringArray();
foreach (string s in strings) Console.WriteLine(s);
Console.ReadKey();
```

خروجی:

```
(item1)
null
5
54.34
```

ListConverter.Merge(object[][])

```
public static object[] Merge(params object[][] arrays)
//    Member of CBrute.Helper.ListConverter
```

با استفاده از این تابع می‌توانید چندین آرایه را با هم ترکیب کنید. تابع Append از این تابع استفاده می‌کند.

پارامتر ها

- arrays: یک آرایه از آرایه‌هایی است که قصد دارید آن‌ها را با یکدیگر به صورت متوالی ترکیب کنید.

مقادیری که برمی‌گرداند

- یک آرایه یک بعدی برمی‌گرداند که حاصل ادغام آرایه‌هایی است که برای تابع ارسال کردید.

مثال(ListConverter.Merge(object[][]))

در مثال زیر چندین آرایه را با یکدیگر ادغام می‌کنیم:

```
// .Net 6
using CBrute.Helper;
object[] array1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
object[] array2 = { 9, 8, 7, 6, 5, 4, 3, 2, 1 };
object[] array3 = { "C", 'B', "rute", '+' };
object[] result = ListConverter.Merge(array1, array2, array3);
Console.WriteLine(result.ConvertStringArrayToString());
Console.ReadKey();
```

خروجی:

```
123456789987654321CBrute+
```

ListConverter.Split(object[], int)

```
public static object[][] Split(this object[] list, int count)
//    Member of CBrute.Helper.ListConverter
```

با استفاده از این تابع می‌توانید یک آرایه یک بعدی را به count قسمت مساوی تقسیم کنید. در صورتی که باقی مانده‌ای وجود داشته باشد، یک آرایه جدید به آرایه‌ها اضافه می‌شود.

پارامتر ها

- list: آرایه‌ای که می‌خواهید آن را تکه‌تکه کنید.
- count: آرایه list قرار است که بین count آریه دیگر تقسیم شود.

مقادیری که برمی‌گرداند

یک آرایه دو بعدی برمی‌گرداند که در کمترین حالت ممکن count سطر دارد. در صورتی که باقی مانده‌ای وجود داشته باشد یک آرایه جدید به آرایه‌ها اضافه می‌شود تا عناصر باقی‌مانده را ذخیره کند. در این حالت آرایه برگشتی count+1 سطر دارد.

مثال(ListConverter.Split(object[], int))

در مثال زیر، هدف این است که یک آرایه را به سه قسمت تقسیم کنیم ولی...:

```
// .Net 6
using CBrute.Helper;
object[] array = { " 1", " 2", " 3", " 4", " 5", " 6", " 7", " 8", " 9", " 10", "11"
};
object[][] result = array.Split(3);
foreach (object[] row in result)
{
    for (int i = 0; i < row.Length; i++)
    {
        Console.Write(row[i]);
        if (i != row.Length - 1) Console.Write(", ");
    }
    Console.WriteLine();
}

Console.ReadKey();
```

خروجی:

```
 1,  2,  3
 4,  5,  6
 7,  8,  9
 10, 11
```

کلاس CBrute.Helper.VariableReader

با استفاده از این کلاس استاتیک می‌توانید کارهایی که در تولید رمزهای عبور متداول هستند را با سرعت بیشتری انجام دهید.

این کلاس فقط یک تابع به نام VariableReader.generateStringArray(string) دارد که یک عبارت رشته‌ای را دریافت کرده و با تجزیه و تحلیل آن عبارت، یک لیست از رشته‌ها را به شما تحویل می‌دهد.

```
public static string[] generateStringArray(string expression)
//    Member of CBrute.Helper.VariableReader
```

ورودی‌هایی که می‌پذیرد

حالت اول: در این حالت، رشته با علامت "-" می‌شود. متغیرهایی که بعد از این علامت می‌توانند وجود داشته باشند را به صورت زیر در نظر بگیرید:

“-X,Y,Z,T”

متغیرهایی که سبز و قرمز هستند اجباری بوده و موارد آبی اختیاری هستند. البته Y در یک حالت فقط اختیاری است که در ادامه متوجه می‌شوید.

در صورتی که X یک عدد باشد، Y(اختیاری) نیز باید یک عدد باشد. خروجی تابع در این حالت اعداد X تا Y است. متغیر Z(عدد) نیز در این حالت تعیین می‌کند که حداقل طول هر رشته چقدر باشد و در صورت لزوم قسمت سمت چپ رشته را پر می‌کند. متغیر T نیز کاراکتری است که به صورت پیش‌فرض ‘ ‘ یا همان Space در نظر گرفته شده. در صورتی که می‌خواهید قسمت سمت چپ رشته با یک کارکتر خاص پر شود، آن کاراکتر را به جای T قرار دهید. اگر فقط از متغیر X استفاده شده بود، اعداد 0 تا X تولید می‌شوند.

در صورتی که X یک کاراکتر باشد، Y اجباری است و هر دو باید یک کاراکتر باشند. تابع بازه از کاراکترها را برمی‌گرداند که شما خواستید. مثلاً”a,z-“ کاراکترهای a تا z را تولید می‌کند. برای فهمیدن کاربرد Z,T پاراگراف قبلی را بخوانید.

توجه کنید که بیش از اندازه از کاراکتر ”,“(کاما) استفاده نکنید و همچنین X همیشه باید از Y کوچکتر باشد.

برای مثال کد زیر را در نظر بگیرید:

```
// .Net 6
using CBrute.Helper;
string[] strings = VariableReader.generateStringArray("-5");//0,5
Console.Write("-5 => ");
for (int i = 0; i < strings.Length; i++)
{
    Console.Write(strings[i]);
    if (i != strings.Length - 1) Console.Write(", ");
}
Console.WriteLine("\n--5,5,3,* :");
strings = VariableReader.generateStringArray("--5,5,3,*");//-5 to 5
foreach (string s in strings) Console.WriteLine(s);
Console.ReadKey();
```

خروجی:

```
-5 => 0, 1, 2, 3, 4, 5
```

```
--5,5,3,* :
*-5
*-4
*-3
*-2
*-1
**0
**1
**2
**3
**4
**5
```

حالت دوم: رشته فقط شامل "@" است. در این حالت آرایه برگشتی شامل برخی از کاراکترهای قابل چاپ است.

حالت سوم: رشته با "file=>X" شروع می‌شود. در این حالت حتماً باید به جای X یک مسیر معتبر از یک فایل وجود داشته باشد. تابع خطوط فایل را خوانده و به صورت یک آرایه از رشته‌ها برمی‌گرداند.

حالت چهارم: رشته با "_" شروع می‌شود. بعد از کاراکتر "_" حتماً باید یک کلمه یا یک جمله بیاید. تابع تمام کاراکترهای انگلیسی موجود در کلمه یا جمله را کوچک و بزرگ کرده و برمی‌گرداند.

برای مثال(حالت چهارم) کد زیر را در نظر بگیرید:

```
// .Net 6
using CBrute.Helper;
object[][] strings =
VariableReader.generateStringArray("_flutter").ConvertToObjectArray().Split(8);
foreach (var row in strings)
{
    foreach (var col in row)
        Console.Write(col + " ");
    Console.WriteLine();
}
Console.ReadKey();
```

خروجی:

```
flutter flutteR fluttEr fluttER flutTer flutTeR flutTEr flutTER
fluTter fluTteR fluTtEr fluTtER fluTTer fluTTeR fluTTEr fluTTER
flUtter flUtteR flUttEr flUttER flUtTer flUtTeR flUtTEr flUtTER
flUTter flUTteR flUTtEr flUTtER flUTTer flUTTeR flUTTEr flUTTER
fLutter fLutteR fLuttEr fLuttER fLutTer fLutTeR fLutTEr fLutTER
fLuTter fLuTteR fLuTtEr fLuTtER fLuTTer fLuTTeR fLuTTEr fLuTTER
fLUtter fLUtteR fLUttEr fLUttER fLUtTer fLUtTeR fLUtTEr fLUtTER
fLUTter fLUTteR fLUTtEr fLUTtER fLUTTer fLUTTeR fLUTTEr fLUTTER
Flutter FlutteR FluttEr FluttER FlutTer FlutTeR FlutTEr FlutTER
FluTter FluTteR FluTtEr FluTtER FluTTer FluTTeR FluTTEr FluTTER
FlUtter FlUtteR FlUttEr FlUttER FlUtTer FlUtTeR FlUtTEr FlUtTER
FlUTter FlUTteR FlUTtEr FlUTtER FlUTTer FlUTTeR FlUTTEr FlUTTER
FLutter FLutteR FLuttEr FLuttER FLutTer FLutTeR FLutTEr FLutTER
FLuTter FLuTteR FLuTtEr FLuTtER FLuTTer FLuTTeR FLuTTEr FLuTTER
FLUtter FLUtteR FLUttEr FLUttER FLUtTer FLUtTeR FLUtTEr FLUtTER
```

```
FLUTter FLUTteR FLUTtEr FLUTtER FLUTTer FLUTTeR FLUTTEr FLUTTER
```

به هر حال دقت کنید که اگر رشته نامعتبری را برای تابع ارسال کنید، تابع یک استثناء پرتاب می‌کند.

فضای نام CBrute.Core

این فضای‌نام شامل کلاس‌ها، شمارنده‌ها و delegateهایی است که برای تولید رمزهای عبور و مدیریت رویدادها به کار گرفته می‌شوند. این فضای‌نام مغز، قلب، دل و... کتابخانه CBrute است!

Delegateهای موجود در فضای‌نام CBrute.Core

CBrute.Core.delegate_OnStart

```
public delegate void delegate_OnStart(CBrute.Core.BruteForce sender)
//    Member of CBrute.Core
```

این Delegate برای مدیریت رویداد OnStart مورد استفاده قرار می‌گیرد.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.

CBrute.Core.delegate_OnPauseOrResume

```
public delegate void delegate_OnPauseOrResume(CBrute.Core.BruteForce sender, long
generated, long total)
//    Member of CBrute.Core
```

این Delegate برای مدیریت رویدادهای OnPause و OnResume مورد استفاده قرار می‌گیرد.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- generated: تعداد رمزهایی که تولید شده.
- total: مجموع رمزهای قابل تولید.

CBrute.Core.delegate_OnStopOrRestart

```
public delegate void delegate_OnStopOrRestart(CBrute.Core.BruteForce sender, object[]
pass)
//    Member of CBrute.Core
```

از این Delegate برای مدیریت رویدادهای OnStop و OnRestart استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- pass: آخرین رمزی که تولید شده.

CBrute.Core.delegate_OnEnd

```
public delegate void delegate_OnEnd(CBrute.Core.BruteForce sender, object[] pass,
bool result)
//    Member of CBrute.Core
```

این Delegate برای مدیریت رویداد OnEnd مورد استفاده قرار می‌گیرد.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- pass: آخرین رمزی که تولید شده. در صورتی که result با True برابر باشد این پارامتر نشان دهنده رمزی است که پیدا شده.
- result: اگر True باشد یعنی رمز عبور پیدا شده، در غیر اینصورت یعنی فرایند شکست خورده.

CBrute.Core.delegate_OnError

```
public delegate void delegate_OnError(CBrute.Core.BruteForce sender, System.Exception
e)
//    Member of CBrute.Core
```

برای مدیریت خطا توسط رویداد OnError استفاده می‌شود. بیشتر در مولتی تردینگ کاربرد دارد.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- e: شامل اطلاعاتی راجع‌به خطای رخ داده است.

CBrute.Core.delegate_PasswordGenerated

```
public delegate bool delegate_PasswordGenerated(CBrute.Core.BruteForce sender,
object[] pass, long generated, long total)
//    Member of CBrute.Core
```

این Delegate برای مدیریت مهمترین رویداد، یعنی PasswordGenerated استفاده می‌شود. این رویداد حتماً باید مدیریت شود، در غیر اینصورت خطا رخ می‌دهد.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- pass: آخرین رمز عبوری که تولید شده.

- generated: تعداد رمزهایی که تا این لحظه تولید شده‌اند.
- total: مجموع رمزهای قابل تولید.

مقادیری که برمی‌گرداند

- در صورتی که توانستید رمزعبور مورد نظر خود را پیدا کنید، True را برگردانید و در غیر اینصورت False.

شمارنده CBrute.Core.BruteForce.ErrorHandlingType

این شمارنده روش مدیریت خطاهایی که ممکن است حین عملیات کرکینگ رخ بدهد را مشخص می‌کند.

مقادیر

- BruteForce.ErrorHandlingType.Event: می‌خواهید خطاها را از طریق رویداد OnError مدیریت کنید.
- BruteForce.ErrorHandlingType.TryCath: می‌خواهید خطاها را از طریق بلاک‌های try/catch مدیریت کنید.

اطلاعات بیشتر

در صورتی که قصد دارید از روش مولتی تردینگ اختصاصی استفاده کنید، پیشنهاد می‌شود که از Event استفاده کنید. در صورتی که نیازی به مولتی تردینگ ندارید از TryCatch استفاده کنید. در CBrute برای مولتی تردینگ از Event استفاده شده است ولی به صورت پیش‌فرض روش مدیریت خطا روی TryCatch تنظیم شده است.

دلیل استفاده از Event در مولتی تردینگ این است که اگر خطایی رخ دهد، کل برنامه از بین نمی‌رود ولی تردی که خطا در آن رخ داده‌است از بین می‌رود. درضمن، در CBrute می‌توانید به راحتی از قابلیت Try again نیز استفاده کنید.

کلاس CBrute.Core.BruteForce

یک کلاس انتزاعی است برای تمامی کلاس‌هایی که قصد دارند فرایند تولید پسوردها را انجام دهند. برای چند ریختی مفید است.

فیلد های کلاس CBrute.Core.BruteForce

BruteForce.JunkArray

```
public static readonly object[] JunkArray
//    Member of CBrute.Core.BruteForce
```

فرض کنید می‌خواهید رمزی را پیدا کنید که سه خانه دارد. حالت‌های خانه اول اعداد {1,2,3}، حالت‌های خانه دوم رشته‌های {"drag","null"} و خانه سوم نیز عدد {5} است. برای این کار شما باید از کلاس ProBrute و PassTestInfo استفاده کنید. در این حالت، نیازی به TestArray ندارید ولی باید بدانید که TestArray ها هیچوقت نمی‌توانند null باشند؛ بنابراین می‌توانید به جای null از BruteForce.JunkArray استفاده کنید.

نگران ProBrute و PassTestInfo نباشید، در ادامه همه چیز را یاد می‌گیرد. شکیبا باشید...

BruteForce.startPos

```
protected long startPos
//    Member of CBrute.Core.BruteForce
```

موقعیت شروع تولید رمزها در این فیلد قرار دارد. مثلاً تصور کنید که می‌خواهید مجموعه‌ای از پسوردها که حد اکثر تعداد آن‌ها 1024 است را تولید کنید؛ ولی تصمیم گرفته‌اید که تولید رمزها از پسوردی با شماره 100 شروع شده و تا 1024 ادامه یابد. برای این کار باید startPos با 100 برابر باشد.

BruteForce.endPos

```
protected long endPos
//    Member of CBrute.Core.BruteForce
```

موقعیتی که در آن تولید رمزها باید متوقف شود. مثلاً تصور کنید که می‌خواهید مجموعه‌ای از پسوردها که حد اکثر تعداد آن‌ها 1024 است را تولید کنید؛ در صورتی که قصد دارید تولید رمزها از پسوردی با شماره 100 شروع شده و تا 555 ادامه یابد. در این حالت باید 100 را در startPos قرار داده و 555 را در endPos قرار دهید.

BruteForce.min

```
protected int min
//    Member of CBrute.Core.BruteForce
```

کمترین طول رمز عبور.

BruteForce.max

```
protected int max
//    Member of CBrute.Core.BruteForce
```

بیشترین طول رمز عبور.

BruteForce.started

```
protected bool started
//    Member of CBrute.Core.BruteForce
```

دقیقاً یک قدم قبل از اینکه تولید رمزها شروع شود، این فیلد باید True شود و پس از اتمام(با خطا، توقف یا پایان معمولی) باید False شود.

BruteForce.needToRestart

```
protected bool needToRestart
```

```
//     Member of CBrute.Core.BruteForce
```

در CBrute، این امکان وجود دارد که حتی در زمانی‌که رمزها در حال تولید هستند، مقدار خاصیت‌های StartPos و EndPos تغییر کند. تغییر خاصیت StartPos مانند این است که فرایند تولید رمزها مجدداً شروع شده است؛ بنابراین، نیاز است که فرایند Restart شود. این کار با True کرد فیلد needToRestart انجام می‌شود. دقت کنید که همیشه قبل از اینکه فرایند Restart شود باید خطاهایی که ممکن است رخ بدهند را دوباره بررسی کنیم.

BruteForce.pause

```
protected bool pause
//     Member of CBrute.Core.BruteForce
```

در صورتی‌که نیاز به یک وقفه در فرایند تولید رمزها دارید، این فیلد باید مقدارش True باشد و در غیر اینصورت اگر نیاز به ادامه دادن فرایند تولید رمزها دارید(در صورتی که قبلاً وقفه ایجاد شده باشد)، این فیلد باید False شود.

BruteForce.stopped

```
protected bool stopped
//     Member of CBrute.Core.BruteForce
```

در صورتی که فرایند تولید رمزها باید متوقف شود، این فیلد باید True شود.

BruteForce.total

```
protected long total
//     Member of CBrute.Core.BruteForce
```

مجموع تعداد رمزهای قابل تولید. این تعداد طبق شرایطی که کاربر تعریف کرده است محاسبه می‌شود. دقت کنید که این فیلد نشان دهنده تمام رمزهای قابل تولید نیست. معمولاً به این‌صورت محاسبه می‌شود: endPos-startPos+1. ولی در کلاس‌هایی که از ExtraLengths پشتیبانی می‌کنند، قضیه کمی پیچیده‌تر است. مقدار این فیلد همیشه توسط تابع CalculateTotal محاسبه می‌شود.

BruteForce.threadID

```
protected int threadID
//     Member of CBrute.Core.BruteForce
```

زمانی که از Worker‌ها استفاده می‌کنید، این فیلد نشان دهنده ID هر ترد است.

خاصیت های کلاس CBrute.Core.BruteForce

BruteForce.StartPos

```
public long StartPos { get; set; }
//     Member of CBrute.Core.BruteForce
```

موقعیت شروع تولید رمزها را تعیین می‌کند. این خاصیت روی فیلد startPos تأثیر می‌گذارد. نباید کوچکتر یا مساوی 0 باشد و همچنین نباید بزرگتر از EndPos باشد. اگر در زمان تولید رمزها در ترد دیگری این خاصیت را تغییر دهید، رویداد OnRestart رخ می‌دهد.

BruteForce.EndPos

```
public long EndPos { get; set; }
//     Member of CBrute.Core.BruteForce
```

موقعیت آخرین رمز قابل تولید را مشخص می‌کند. این خاصیت روی فیلد endPos تأثیر می‌گذارد. در صورتی که قصد دارید تمام رمزهای قابل تولید را بسازید، مقدار این خاصیت می‌تواند یک عدد کوچکتر یا مساوی 0 باشد. این خاصیت نباید از تعداد تمام پسوردهای قابل تولید بیشتر باشد. مثلاً اگر می‌خواهید رمز عبوری را که سه خانه دارد و هر خانه آن سه حالت دارد را پیدا کنید، EndPos نباید بیشتر از 3x3x3 باشد! می‌توانید این خاصیت را حتی در زمان تولید رمزها عوض کنید.

BruteForce.MinimumPassLength

```
public int MinimumPassLength { get; }
//     Member of CBrute.Core.BruteForce
```

کمترین طول رمز عبور را تعیین می‌کند.

BruteForce.MaximumPassLength

```
public int MaximumPassLength { get; }
//     Member of CBrute.Core.BruteForce
```

بیشترین طول رمز عبور را تعیین می‌کند.

BruteForce.Pause

```
public bool Pause { get; set; }
//     Member of CBrute.Core.BruteForce
```

این خاصیت برای ایجاد وقفه در یک فرایند یا ازسرگیری آن به کار می‌رود. این خاصیت روی فیلد pause تأثیر دارد.

در صورتی‌که در ترد دیگری می‌خواهید در عملیات تولید رمزها وقفه ایجاد کنید، مقدار این خاصیت را روی True تنظیم کرده و در صورتی‌که می‌خواهید فرایند ادامه یابد(در صورتی‌که قبلاً وقفه ایجاد شده باشد) مقدار این خاصیت را False کنید.

BruteForce.Started

```
public bool Started { get; }
//     Member of CBrute.Core.BruteForce
```

نشان می‌دهد که آیا فرایند تولید رمزها شروع شده است یا خیر.

BruteForce.ThreadID

```
public int ThreadID { get; }
//     Member of CBrute.Core.BruteForce
```

در صورتی‌که Object مورد بررسی در ترد دیگری باشد، این خاصت تعیین کننده ID آن ترد است. این خاصیت را با ManagedThreadId اشتباه نگیرید!

BruteForce.WaitForPauseCheckingMillisecond

```
public int WaitForPauseCheckingMillisecond { get; set; }
//     Member of CBrute.Core.BruteForce
```

تابع waitUntilPause، طی بازه‌های زمانی که توسط این خاصت مشخص شده است مقدار pause را بررسی می‌کند. هر چه این خاصیت مقدارش بیشتر باشد، رویداد OnResume با وقفه بیشتری رخ می‌دهد.

توابع کلاس CBrute.Core.BruteForce

BruteForce.CalculateTotal()

```
protected abstract long CalculateTotal()
//     Member of CBrute.Core.BruteForce
```

یک تابع انتزاعی که باید توسط تمام کلاس‌هایی که از BruteForce مشتق می‌شوند پیاده‌سازی شود. وظیفه این تابع این است که مجموع تعداد رمزهایی که قرار است تولید بشوند را محاسبه کند.

مقادیری که برمی‌گرداند

کلاسی که این تابع را پیاده‌سازی می‌کند باید مجموع رمزهایی که قرار است تولید بشوند را محاسبه کند. برای اینکه بهتر متوجه شوید به مثال زیر توجه کنید:

فرض کنید قصد دارید رمزهایی را تولید کنید که از قواعد زیر پیروی می‌کنند:

- هر خانه یکی از حالت‌های {"1","2","reza","null","5.5"} را می‌پذیرد.
- کمترین طول هر پسورد 4 است.
- بیشرین طول هر پسورد 8 است.
- نمی‌خواهیم پسوردهایی با طول‌های 5 و 7 تولید بشود.
- شروع تولید رمزها از موقعیت 20000 باشد.
- پایان تولید رمزها آخرین موقعیت قابل تولید باشد.

در این حالت باید با توجه به حداقل و حداکثر طول هر پسورد و اندازه‌هایی که می‌خواهیم تولید نشوند و موقعیت شروع و پایان، مجموع رمزهای قابل تولید را محاسبه کنیم. برای مثال در این حالت قرار است که 390625 رمز متمایز تولید شود. این در حالی است که اگر قرار باشد تمامی رمزها را بدون در نظر گرفتن طول‌های اضافی و موقعیت شروع و پایان تولید کنیم، باید 488125 رمز را تولید کنیم.

لازم به ذکر است که ما قصد داشتیم که تولید رمزها از رمزی با موقعیت 20000 شروع شود، ولی این اتفاق رخ نمی‌دهد زیرا رمزی که در موقعیت 20000 وجود دارد یک رمز 7 عضوی است. برای همین نیاز است که رمزهای 7 عضوی را رد کنیم و از 97501 تولید رمزها را ادامه دهیم. البته شما نیازی نیست نگران این موارد باشید، CBrute این کارها را برای شما انجام می‌دهد.

BruteForce.NeedErrorsChecking()

```
protected abstract void NeedErrorsChecking()
//     Member of CBrute.Core.BruteForce
```

قبل از اینکه فرایند تولید رمزها Restart بشود، نیاز داریم که دوباره خطاهایی که ممکن است رخ بدهند را پیش‌بینی کنیم.

BruteForce.resetStopPause()

```
protected void resetStopPause()
//     Member of CBrute.Core.BruteForce
```

این تابع فیلدهای pause و stopped را False می‌کند.

BruteForce.waitUntilPause()

```
protected bool waitUntilPause()
//     Member of CBrute.Core.BruteForce
```

این تابع زمانی استفاده می‌شود که فیلد pause مقدار True را داشته باشد. زمانی که این فیلد True باشد، باید این تابع فراخوانی شود. این تابع هر WaitForPauseCheckingMillisecond (میلی ثانیه) یک بار بررسی می‌کند که آیا pause همچنان True است یا خیر.

مقادیری که برمی‌گرداند

- در صورتی که در حین بررسی فیلد pause، فیلد stopped با True برابر شود، کار تابع پایان یافته و مقدار برگشتی True است و نشان دهنده این است که فرایند تولید رمزها باید متوقف شود. در صورتی که pause با False برابر باشد نیز کار تابع پایان یافته و مقدار برگشتی نیز False است. False بودن مقدار برگشتی نیز نشان دهنده فراخوانی کردن اداره کننده رویداد OnResume است.

BruteForce.Stop()

```
public void Stop()
//     Member of CBrute.Core.BruteForce
```

اگر قصد دارید که در ترد دیگری یک فرایند تولید رمز را متوقف کنید، از این تابع استفاده کنید.

BruteForce.StartBruteForceEvent()

```
protected abstract void StartBruteForceEvent()
//     Member of CBrute.Core.BruteForce
```

باید طوری پیاده‌سازی شود که در صورتی که در عملیات تولید رمزها خطایی رخ داد، با استفاده از رویداد OnError مدیریت شود.

BruteForce.StartBrute()

```
protected abstract void StartBrute()
//     Member of CBrute.Core.BruteForce
```

باید طوری پیاده‌سایز شود که در صورتی که در عملیات تولید رمزها خطایی رخ داد، با استفاده از بلاک‌های try/catch قابل مدیریت باشد.

تابع StartBruteForceEvent این تابع(StartBrute()) را داخل یک بلاک try قرار داده و در صورتی که خطایی رخ دهد، داخل بلاک catch، اداره‌کننده رویداد OnError را فراخوانی می‌کند.

BruteForce.Start(CBrute.Core.BruteForce.ErrorHandlingType)

```
public void Start([CBrute.Core.BruteForce.ErrorHandlingType handlingType =
BruteForce.ErrorHandlingType.TryCath])
//     Member of CBrute.Core.BruteForce
```

این تابع فرایند تولید رمزها را شروع می‌کند.

پارامتر ها

- handlingType: این پارامتر روش مدیریت خطاها را مشخص می‌کند. به صورت پیش‌فرض روی BruteForce.ErrorHandlingType.TryCath تنظیم شده.

رویداد های مشترک CBrute.Core.BruteForce

تمام رویدادهای کلاس‌هایی که BruteForce را بسط می‌دهند مشترک است. برای جلوگیری از اضافه‌کاری در این قسمت آن رویدادها را شرح می‌دهیم.

OnStart

```
public event CBrute.Core.delegate_OnStart OnStart
//     Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که تا شروع فرایند تولید رمزها، فقط یک مرحله باقی مانده است. می‌توانید کارهایی که نیاز دارید تا قبل از شروع فرایند انجام شود را توسط این رویداد انجام دهید.

OnPause

```
public event CBrute.Core.delegate_OnPauseOrResume OnPause
//     Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که در فرایند تولید رمزها وقفه ایجاد شده است. دقت کنید که به ازای هر بار تغییر StartPos و EndPos نیز این رویداد رخ می‌دهد.

OnResume

```
public event CBrute.Core.delegate_OnPauseOrResume OnResume
//    Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که فرایند Pause شده، از سر گرفته شود. دقت کنید که به ازای هر بار تغییر StartPos و EndPos نیز این رویداد رخ می‌دهد.

OnStop

```
public event CBrute.Core.delegate_OnStopOrRestart OnStop

//    Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که فرایند تولید رمزها توسط تابع Stop متوقف شده است.

OnRestart

```
public event CBrute.Core.delegate_OnStopOrRestart OnRestart
//    Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که شما مقدار StartPos را در زمان تولید رمزها تغییر داده‌اید.

OnEnd

```
public event CBrute.Core.delegate_OnEnd OnEnd
//    Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که فرایند تولید رمزها به پایان رسیده است.

OnError

```
public event CBrute.Core.delegate_OnError OnError
//    Member of CBrute.Core.SimpleBrute
```

این رویداد زمانی رخ می‌دهد که خطایی در فرایند تولید رمزها رخ داده است. فقط زمانی کار می‌کند که روش مدیریت خطا در حین فراخوانی تابع Start را روی Event تنظیم کرده باشید.

PasswordGenerated

```
public event CBrute.Core.delegate_PasswordGenerated PasswordGenerated
//    Member of CBrute.Core.SimpleBrute
```

زمانی رخ می‌ده که یک رمز تولید شده است و آماده تست است. در صورتی که رمز عبور را پیدا کرده‌اید مقدار True و در غیر اینصورت مقدار False را برگردانید. این رویداد نباید null باشد!

کلاس CBrute.Core.SimpleBrute

این کلاس ساده ترین روش تولید رمزهای عبور را پیاده‌سازی کرده است. با استفاده از این کلاس می‌توانید مانند هر ابزار دیگری شروع به تولید رمزهای عبور بکنید.

برای جلوگیری از اضافه‌کاری، برخی از توابع و خاصیت‌ها و رویدادها که بین تمامی کلاس‌های BruteForce مشترک هستند را دیگر تعریف نمی‌کنیم و شما باید این بخش را مطالعه کنید!

توابع استاتیک کلاس CBrute.Core.SimpleBrute

SimpleBrute.GetMax(object[], int, int, int[])

```
public static long GetMax(object[] test, int min, int max, [int[] extraLengths =
null])
//    Member of CBrute.Core.SimpleBrute
```

با استفاده از این تابع می‌توانید تعداد رمزهای قابل تولید برای یک شرایط خاص را به دست بیاورید.

پارامتر ها

- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- min: کمترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.
  - min نباید بزرگتر از max باشد.
- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.
- extraLengths: در صورتی که قصد دارید برخی از اعداد بین min و max را نادیده بگیرید، آن‌ها را در این آرایه قرار دهید. دقت کنید نباید از اعدادی استفاده کنید که در بازه min و max نیست! همچنین نباید طوری این آرایه را پر کنید که هیچ رمزی قابل تولید نباشد! مثلاً اگر min برابر با 3 باشد و max برابر با 5 باشد، نباید این آرایه شامل عناصر 3,4,5 باشد!

مقادیری که برمی‌گرداند

- تعداد رمزهای قابل تولید را برای شرایطی که تعریف کردید را محاسبه کرده و برمی‌گرداند.

مثال(SimpleBrute.GetMax(object[], int, int, int[]))

این مثال حداکثر تعداد رمزهای قابل تولید را برای شرایط زیر را محاسبه می‌کند:

- رمزها از آرایه test تشکیل می‌شوند.
- کمترین طول هر رمز 4 و بیشترین طول هر رمز 8 است.
- در حالت دوم نمی‌خواهیم رمزهایی با طول 5 و 7 تولید بشوند.

```
// .Net 6
using CBrute.Core;
object[] test = { "1", "2", "reza", "null", "5.5" };
int min = 4, max = 8;
int[] extraLengths = { 5, 7 };
Console.Write("Without extraLengths: ");
Console.WriteLine(SimpleBrute.GetMax(test, min, max));
Console.Write("With extraLengths: ");
Console.WriteLine(SimpleBrute.GetMax(test, min, max, extraLengths));
Console.ReadKey(true);
```

توجه کنید که ما اینجا نمی‌توانیم از StartPos و EndPos استفاده کنیم. ما فقط قصد داریم بدانیم چند پسورد قرار است تولید بشود.

خروجی به صورت زیر است:

```
Without extraLengths: 488125
With extraLengths: 406875
```

SimpleBrute.GetPassByPos(long, object[], int, int)

```
public static object[] GetPassByPos(long pos, object[] test, int min, int max)
//    Member of CBrute.Core.SimpleBrute
```

می‌توانید با استفاده از موقعیت یک رمز در بین تمام حالت‌های ممکن، به خود آن رمز برسید.

پارامتر ها

- pos: موقعیت رمزی که می‌خواهید آن را دریافت کنید. به موارد زیر توجه کنید:
  - pos نباید کوچکتر یا مساوی 0 یا بزرگتر از تمام حالت‌های ممکن باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.

- min: کمترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.
  - min نباید بزرگتر از max باشد.
- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.

مقادیری که برمی‌گرداند

- رمزی که در موقعیت pos وجود دارد را محاسبه کرده و برمی‌گرداند.

اطلاعات بیشتر

این تابع و برادرش را از عمد برای این پیاده سازی کردم که بتوانم کاری کنم تا CBrute از مولتی تردینگ پشتیبانی کند. با این حال شما با استفاده از این تابع و برادرش GetPosByPass می‌توانید فرایند ذخیره و بازیابی عملیات را به راحتی انجام دهید. کافی است که یا رمز را ذخیره کنید، یا موقعیت آن‌را و بعداً یا از طریق رمز به موقعیت برسید، یا برعکس... همچنین باید بدانید که تولید تمامی پسوردها با این توابع اصلاً عاقلانه نیست زیرا بسیار کند انجام می‌شود. این توابع یک‌بار مصرف هستند و نباید بیش از حد استفاده شوند؛ به جای این‌کار باید از روش‌های استانداردی که هر کلاس BruteForce در اختیارتان قرار می‌دهد استفاده کنید. (همان تابع ()Start)

مثال(SimpleBrute.GetPassByPos(long, object[], int, int))

اگر به خاطر داشته باشید، در یکی از مثال‌ها گفتیم: رمزی که در موقعیت 20000 قرار دارد یک رمز 7 خانه‌ای است. در این مثال قصد داریم رمزی که در آن موقعیت است را به دست بیاوریم:

```
// .Net 6
using CBrute.Core;
using CBrute.Helper;

object[] test = { "1", "2", "reza", "null", "5.5" };
int min = 4, max = 8;
object[] password = SimpleBrute.GetPassByPos(20000, test, min, max);
Console.WriteLine($"The length of {nameof(password)} is {password.Length}");
Console.WriteLine($"The {nameof(password)} is
{password.ConvertStringArrayToString()}");//Consider the number 5.5 as an element.
Console.ReadKey(true);
```

خروجی:

```
The length of password is 7
The password is 1115.55.55.55.5
```

SimpleBrute.GetPosByPass(object[], object[], int)

```
public static long GetPosByPass(object[] pass, object[] test, int min)
//    Member of CBrute.Core.SimpleBrute
```

با استفاده از یک پسورد می‌توانید به موقعیت آن پسورد در بین تمام حالت‌های ممکن برسید.

پارامتر ها

- pass: رمزی که می‌خواهید موقعیت آن‌را محاسبه کنید. به موارد زیر توجه کنید:
  - pass نباید null باشد یا شامل عناصر null باشد.
  - pass نباید خالی باشد.
  - pass نباید شامل عناصری باشد که در test نیست.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عنصار null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- min: کمترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.

مقادیری که برمی‌گرداند

- موقعیت رمزی که دریافت کرده است را محاسبه کرده و برمی‌گرداند.

اطلاعات بیشتر

لطفاً تابع قبلی را بررسی کنید و بخش اطلاعات بیشتر را بخوانید.

مثال(SimpleBrute.GetPosByPass(object[], object[], int))

در این مثال می‌خواهیم بررسی کنیم که آیا دو تابع GetPosByPass و GetPassByPos درست کار می‌کنند یا خیر:

```
// .Net 6
using CBrute.Core;
object[] test = { "1", "2", "reza", "null", "5.5","0" };
int min = 1, max = 9;
long endPos = SimpleBrute.GetMax(test, min, max);
for (long pos = 1; pos <= endPos; ++pos)
{
    object[] pass = SimpleBrute.GetPassByPos(pos, test, min, max);
    long temp = SimpleBrute.GetPosByPass(pass, test, min);
    if (temp != pos) throw new Exception("NOOOOO!");
}
Console.WriteLine(":)");
```

```
Console.ReadKey(true);
```

خروجی:

```
:)
```

خاصیت های موجود در کلاس CBrute.Core.SimpleBrute

SimpleBrute.Test

```
public object[] Test { get; }
//     Member of CBrute.Core.SimpleBrute
```

حالت‌هایی که ساختار هر پسورد را تشکیل می‌دهند را برمی‌گرداند.

SimpleBrute.RealPos

```
public long RealPos { get; }
//     Member of CBrute.Core.SimpleBrute
```

در حالی که رمزها توسط کلاس SimpleBrute در حال تولید هستند، این خاصیت موقعیت واقعی آخرین رمز تولید شده را برمی‌گرداند.
تفاوت این موقعیت با پارامتر generated(این پارامتر را در بخش delegateها مشاهده کردید) این است که پارامتر generated تعداد رمزهای تولید شده را برمی‌گرداند ولی RealPos موقعیت واقعی آخرین رمز تولید شده در بین تمام حالت‌های ممکن را برمی‌گرداند.
این یعنی می‌توانید این موقعیت را به تابع GetPassByPos تحویل داده و رمز موجود در آن موقعیت را به دست آورید.

SimpleBrute.ExtraLengths

```
public int[] ExtraLengths { get; }
//     Member of CBrute.Core.SimpleBrute
```

اندازه‌هایی را برمی‌گرداند که قرار نیست تولید شوند. می‌تواند null باشد!

سازنده‌های کلاس CBrute.Core.SimpleBrute

SimpleBrute.SimpleBrute(long, long, int, int, object[], int[], int)

```
public SimpleBrute(long startPos, long endPos, int minPassLen, int maxPassLen,
object[] test, [int[] extraPassLengths = null], [int threadID = -1])
//     Member of CBrute.Core.SimpleBrute
```

سازنده کلاس SimpleBrute.

پارامتر ها

- startPos: موقعیتی که تولید رمزها باید از آنجا شروع شود. به موارد زیر توجه کنید:

- startPos نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر endPos باشد.

- endPos: موقعیتی که تولید رمزها باید در آنجا متوقف شود. برای تولید تمام رمزها می‌توانید مقداری کوچکتر یا مساوی 0 را در این پارامتر قرار دهید. به موارد زیر توجه کنید:
  - endPos نباید بزرگتر از تمامی حالت‌های ممکن باشد.
- minPassLen: کمترین طول پسوردها. به موارد زیر توجه کنید:
  - minPassLen نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر maxPassLen باشد.
- maxPassLen: بیشترین طول پسوردها. به موارد زیر توجه کنید:
  - maxPassLen نباید کوچکتر مساوی 0 باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- extraPassLengths: در صورتی که قصد دارید برخی از اعداد بین min و max را نادیده بگیرید، آن‌ها را در این آرایه قرار دهید. دقت کنید نباید از اعدادی استفاده کنید که در بازه min و max نیست! همچنین نباید طوری این آرایه را پر کنید که هیچ رمزی قابل تولید نباشد! مثلاً اگر min برابر با 3 باشد و max برابر با 5 باشد، نباید این آرایه شامل عناصر 3,4,5 باشد!
- threadID: در صورتی که می‌خواهید از روش خودتان برای پیاده‌سازی مولتی تردینگ استفاده کنید، این پارامتر نشان دهنده ID تردی است که نمونه در حال ایجاد قرار است در آن اجرا شود.

یک مثال ساده

در این مثال قصد داریم یک رمز عبور که از شرایط زیر پیروی می‌کند را پیدا کنیم:

رمز عبور بین 4 تا 6 خانه است ولی مطمئن هستیم که 5 خانه نیست. رمز عبور می‌تواند ترکیبی از تاریخ تولد، شماره تلفن و... باشد. می‌خواهیم تمام حالت‌های ممکن را تست کنیم. (رمز:reza0918813)

این کار به راحتی توسط کلاس SimpleBrute قابل انجام است:

```
// .Net 6
using CBrute.Core;
using CBrute.Helper;

object[] testArray = { "13", "81", "31", "18", "0939", "0918", "0938", "1382", "reza"
};
int min = 4, max = 6;
int[] extraLengths = { 5 };
long startPos = 1, endPos = 0;
SimpleBrute SB = new SimpleBrute(startPos, endPos, min, max, testArray,
extraLengths);
SB.PasswordGenerated += (BruteForce sender, object[] pass, long generated, long
total) =>
```

```
    pass.ConvertObjectArrayToString().Equals("reza09188113");//PasswordGenerated
SB.OnStart += (sender) => Console.WriteLine("OnStart...");//OnStart
SB.OnEnd += (sender, pass, result) =>//OnEnd
{
    if (result)
    {
        Console.WriteLine("Password found");
        Console.WriteLine(pass.ConvertObjectArrayToString());
    }
    else Console.WriteLine("WTF?");
};
SB.Start();
Console.ReadKey(true);
```

خروجی:

```
OnStart...
Password found
reza09188113
```

چی شد؟

در این مثال ساده ما قصد داشتیم رمزی که مقدارش reza09188113 است را پیدا کنیم. ابتدا مواردی که برای نمونه سازی از کلاس SimpleBrute نیاز است را آماده کردیم. این موارد شامل startPos، endPos، testArray، extraLengths ، min و max بودند.

بعد از آماده‌سازی این موارد یک نمونه از کلاس SimpleBrute را ایجاد کردیم و با استفاده از رویداد PasswordGenerated بررسی کردیم که آیا رمز تولید شده با reza09188113 برابر است یا خیر. همانطور که می‌دانید این رمز 4 خانه دارد که به ترتیب عبارتند از reza، 0918، 81 و 13.

با استفاده از رویداد OnEnd هم بررسی کردیم که آیا رمز پیدا شده است یا خیر.

کلاس CBrute.Core.ProBrute

زمانی که اطلاعات بیشتری راجع به رمزی که می‌خواهید پیدا کنید دارید، استفاده از این کلاس بهترین انتخاب است. با استفاده از این کلاس می‌توانید بدون در نظر گرفتن تعداد اعضای هر رمز، تعیین کنید که هر خانه از رمز چه حالت‌هایی را بپذیرد. به عنوان مثال می‌توانید حالتی را در نظر بگیرید که می‌دانید که خانه یکی مانده به آخر فقط شامل یکی از حالت‌های {"_","12"} بوده و خانه سوم همیشه یکی از حالت‌های {"666","555"} است!

اگر از ابهاماتی که ممکن است پیش بیاید صرف‌نظر کنیم، این کار به راحتی در CBrute قابل انجام است. به زودی راجع‌به ابهامات بیشتر می‌خوانید.

برای جلوگیری از اضافه‌کاری، برخی از توابع و خاصیت‌ها و رویدادها که بین تمامی کلاس‌های BruteForce مشترک هستند را دیگر تعریف نمی‌کنیم و شما باید این بخش را مطالعه کنید!

کلاس CBrute.Core.ProBrute.PassTestInfo

قبل از اینکه توابع و خاصیت‌های کلاس ProBrute را شرح بدهم، ابتدا باید شما را با کلاس ProBrute.PassTestInfo آشنا کنم. این کلاس یک کلاس کمکی مهم است که فقط توسط کلاس ProBrute استفاده می‌شود. با استفاده از این کلاس می‌توانید حالت‌هایی که هر خانه از رمز می‌تواند بپذیرد را تعیین کنید. این کار از طریق فاصله از ابتدا و انتها انجام می‌شود.

سازنده کلاس CBrute.Core.ProBrute.PassTestInfo

ProBrute.PassTestInfo.PassTestInfo(int, object[])

```
public PassTestInfo(int position, object[] testList)
//    Member of CBrute.Core.ProBrute.PassTestInfo
```

سازنده کلاس ProBrute.PassTestInfo

پارامتر ها

- position: با استفاده از این پارامتر می‌توانید موقعیت از ابتدا یا انتهای یک رمز را مشخص کنید. برای فاصله از انتها از اعداد منفی و برای فاصله از ابتدا از اعداد غیر منفی استفاده کنید. برای مثال 1- خانه آخر و 0 خانه اول رمز است.
- testList: آرایه‌ای از objectها که نشان دهنده حالت‌هایی است که قرار است در جایگاه position استفاده شوند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.

اطلاعات بیشتر

برای اینکه بهتر بتوانید وظیفه این کلاس را درک کنید، فرض کنید می‌خواهید رمزی پیدا کنید که خانه آخر آن حالت‌های {1,2,3} و خانه یکی مانده به آخر حالت‌های {“nfsd”,”sdkfjirj”} را می‌پذیرد. همچنین می‌دانید خانه اول فقط رشته ”test“ است. در این حالت باید سه نمونه از کلاس PassTestInfo بسازید:

```
ProBrute.PassTestInfo[] testInfos =
{
    new(0, new object[]{"test" }),
    new(-1, new object[]{1, 2, 3 }),
    new(-2, new object[]{"nfsd", "sdkfjirj" })
};
```

خاصیت های کلاس ProBrute.PassTestInfo

ProBrute.PassTestInfo.Position

```
public int Position { get; }
```

```
//     Member of CBrute.Core.ProBrute.PassTestInfo
```

فاصله از ابتدا و انتهای پسورد را مشخص می‌کند.

ProBrute.PassTestInfo.Test

```
public object[] Test { get; }
//     Member of CBrute.Core.ProBrute.PassTestInfo
```

آرایه‌ای از objectها که نشان دهنده حالت‌هایی است که قرار است در جایگاه position استفاده شوند.

توابع کلاس ProBrute.PassTestInfo

ProBrute.PassTestInfo.GetPosition(int)

```
public int GetPosition(int length)
//     Member of CBrute.Core.ProBrute.PassTestInfo
```

این تابع با استفاده از طول یک پسورد، Position را در آن محاسبه کرده و ایندیکسی از پسورد را برمی‌گرداند.

پارامتر ها

- length: طول پسوردی که قرار است بررسی شود.

مقادیری که برمی‌گرداند

- این تابع بر اساس پارامتر length و خاصیت Position (این پارامتر فاصله از ابتدا یا انتها را مشخص می‌کند) یک ایندکس از پسورد را برمی‌گرداند. در صورتی که تابع موفق نشود عدد 1- را برمی‌گرداند.

اطلاعات بیشتر همراه با مثال

قصد داریم رمزی را پیدا کنیم که خانه دوم آن همیشه یکی از حالت‌های {1,2,3,4} است و خانه یکی مانده به آخر آن فقط یک حالت {55} دارد. همچنین خانه آخر و اول نیز یکی از حالت‌های {2,4,6} هستند. بقیه خانه را فعلاً کاری نداریم...

```
// .Net 6
using CBrute.Core;
ProBrute.PassTestInfo[] testInfos =
{
    new(0, new object[]{2,4,6}),//First cell of password
    new(1, new object[]{1,2,3,4}),//Second cell of password
    new(-2, new object[]{ 55 }),//Penultimate cell of password
    new(-1,new object[]{2,4,6})//Last cell of password
};
Console.WriteLine($"*------- Index({testInfos[0].GetPosition(8)})");
Console.WriteLine($"-*------ Index({testInfos[1].GetPosition(8)})");
Console.WriteLine($"------*- Index({testInfos[2].GetPosition(8)})");
Console.WriteLine($"-------* Index({testInfos[3].GetPosition(8)})");
```

```
Console.ReadKey(true);
```

خروجی:

```
*------- Index(0)
-*------ Index(1)
------*- Index(6)
-------* Index(7)
```

همانطور که مشاهده می‌کنید، ما تمامی نمونه‌های کلاس ProBrute.PassTestInfo را فقط روی پسوردهای 8 سلولی(منظورم همون خونه‌اس یا هر چیزی که خوشتون میاد اسمش باشه، گیر ندید) تست کردیم. در حالت اول ما اولین خانه پسورد را مورد بررسی قرار دادیم که شاخصش در پسورد برابر با 0 است.

در حالت دوم نیز حالت‌های خانه دوم را تعیین کردیم و در حالت سوم، حالت‌های خانه یکی مانده به آخر را تعیین کردیم. حالت آخر نیز آخرین سلول از پسورد را تعیین می‌کند.

ولی این موارد فقط برای پسوردهایی با طول 8 عنصر عالی کار می‌کند، اگر طول پسورد 3 بود چه می‌شد؟ این حالت باعث ایجاد ابهام می‌شود و در کلاس ProBrute می‌تواند باعث ایجاد خطا شود. البته می‌توانید کاری کنید که ProBrute به ابهامات توجهی نکند.

اجازه بدهید که با هم نتیجه را مشاهده کنیم:

```
// .Net 6
using CBrute.Core;
ProBrute.PassTestInfo[] testInfos =
{
    new(0, new object[]{2,4,6 }),//First cell of password
    new(1, new object[]{1, 2, 3, 4 }),//Second cell of password
    new(-2, new object[]{ 55 }),//Penultimate cell of password
    new(-1,new object[]{2,4,6 })//Last cell of password
};
Console.WriteLine($"*-- Index({testInfos[0].GetPosition(3)})");
Console.WriteLine($"-*- Index({testInfos[1].GetPosition(3)}) ambiguity");
Console.WriteLine($"-*- Index({testInfos[2].GetPosition(3)}) ambiguity");
Console.WriteLine($"--* Index({testInfos[3].GetPosition(3)})");
Console.ReadKey(true);
```

خروجی:

```
*-- Index(0)
-*- Index(1) ambiguity
-*- Index(1) ambiguity
--* Index(2)
```

همانطور که مشاهده می‌کنید، برای خانه دوم و یکی مانده به آخر یک نتیجه را نشان می‌دهد. تابع کاملاً درست کار می‌کند اما این مشکل شماست. باید بدانید که در صورتی که ابهامی ایجاد بشود، حالت‌های آن خانه را اولین عنصر در آرایه PassTestInfoها تعیین می‌کند. بنابراین حواستان را جمع کنید!

توابع استاتیک کلاس CBrute.Core.ProBrute

ProBrute.GetMax(object[], CBrute.Core.ProBrute.PassTestInfo[], int, int, int[])

```
public static long GetMax(object[] test, CBrute.Core.ProBrute.PassTestInfo[]
testInfos, int min, int max, [int[] extraLengths = null])
//    Member of CBrute.Core.ProBrute
```

با استفاده از این تابع می‌توانید تعداد رمزهای قابل تولید برای یک شرایط خاص را به دست بیاورید.

پارامتر ها

- test: از این آرایه زمانی استفاده می‌شود که ProBrute قصد دارد حالت یکی از خانه‌های پسورد را تغییر دهد ولی حالت‌های این خانه در آرایه testInfos تعیین نشده است. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد. در صورتی‌که به این پارامتر نیاز ندارید از JunkArray استفاده کنید.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- testInfos: با استفاده از این آرایه می‌توانید حالت‌های هر خانه از پسورد را تعیین کنید. ProBrute.PassTestInfo را مطالعه کنید. می‌تواند null باشد.
- min: کمترین طول هر پسورد. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.
  - min نباید بزرگتر از max باشد.
- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.
- extraLengths: در صورتی‌که قصد دارید برخی از اندازه‌های بین min و max را نادیده بگیرید، آن‌ها را در این آرایه قرار دهید. دقت کنید نباید از اعدادی استفاده کنید که در بازه min و max نیست و همچنین نباید طوری این آرایه را پر کنید که هیچ رمزی قابل تولید نباشد! مثلاً اگر min برابر با 3 و max برابر با 5 باشد، نباید این آرایه شامل عناصر 3,4,5 باشد!

مقادیری که برمی‌گرداند

- تعداد رمزهای قابل تولید را برای شرایطی که تعریف کردید را محاسبه کرده و برمی‌گرداند.

مثال(ProBrute.GetMax(object[], CBrute.Core.ProBrute.PassTestInfo[], int, int, int[]))

لطفاً ابتدا این مثال را بخوانید و سپس این موارد را نیز به آن اضافه کنید:

- برای خانه‌هایی که حالتی را در testInfos برایشان تعیین نکردیم، حالت‌های {10,20,30,40} را در نظر می‌گیریم.
- در این مثال که اکنون مشاهده می‌کنید، یک‌بار از extraLengths استفاده کردیم و یک بار از آن استفاده نکردیم.
- کمترین طول رمزها را 3 و بیشترین طول رمزها را 10 در نظر می‌گیریم.

- با استفاده از خاصیت استاتیک IgnoreTrivialErrors می‌توانیم خطای ناشی از ابهامات را نادیده بگیریم.

```
// .Net 6
using CBrute.Core;
ProBrute.IgnoreTrivialErrors = true;//To avoid ambiguities
ProBrute.PassTestInfo[] testInfos =
{
    new(0, new object[]{2,4,6 }),//First cell of password
    new(1, new object[]{1, 2, 3, 4 }),//Second cell of password
    new(-2, new object[]{ 55 }),//Penultimate cell of password
    new(-1,new object[]{2,4,6 })//Last cell of password
};
object[] testArray = { 10, 20, 30, 40 };
int min = 3, max = 10;
int[] extraLengths = { 6, 7 };
Console.WriteLine($"With {nameof(extraLengths)} : {ProBrute.GetMax(testArray,
testInfos, min, max, extraLengths)}");
Console.WriteLine($"Withou {nameof(extraLengths)} : {ProBrute.GetMax(testArray,
testInfos, min, max)}");
Console.ReadKey(true);
```

خروجی:

```
With extraLengths : 193752
Withou extraLengths : 196632
```

ProBrute.GetPassByPos(long, object[], CBrute.Core.ProBrute.PassTestInfo[], int, int)

```
public static object[] GetPassByPos(long pos, object[] test,
CBrute.Core.ProBrute.PassTestInfo[] testInfos, int min, int max)
//    Member of CBrute.Core.ProBrute
```

می‌توانید با استفاده از موقعیت یک رمز در بین تمام حالت‌های ممکن، به خود آن رمز برسید.

پارامتر ها

- pos: موقعیت رمزی که می‌خواهید آن را دریافت کنید. به موارد زیر توجه کنید:
  - pos نباید کوچکتر یا مساوی 0 یا بزرگتر از تمام حالت‌های ممکن باشد.
- test: از این آرایه زمانی استفاده می‌شود که ProBrute قصد دارد حالت یکی از خانه‌های پسورد را تغییر دهد ولی حالت‌های این خانه در آرایه testInfos تعیین نشده است. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد. در صورتی‌که به این پارامتر نیاز ندارید از JunkArray استفاده کنید.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.

- testInfos: با استفاده از این آرایه می‌توانید حالت‌های هر خانه از پسورد را تعیین کنید. ProBrute.PassTestInfo را مطالعه کنید. می‌تواند null باشد.
- min: کمترین طول هر پسورد. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.
  - min نباید بزرگتر از max باشد.
- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.

مقادیری که برمی‌گرداند

- رمزی که در موقعیت pos وجود دارد را محاسبه کرده و برمی‌گرداند.

اطلاعات بیشتر

این تابع و برادرش را از عمد برای این پیاده سازی کردم که بتوانم کاری کنم تا CBrute از مولتی تردینگ پشتیبانی کند. با این حال شما با استفاده از این تابع و برادرش GetPosByPass می‌توانید فرایند ذخیره و بازیابی عملیات را به راحتی انجام دهید. کافی است که یا رمز را ذخیره کنید، یا موقعیت آن‌را و بعداً یا از طریق رمز به موقعیت برسید، یا برعکس... همچنین باید بدانید که تولید تمامی پسوردها با این توابع اصلاً عاقلانه نیست زیرا بسیار کند انجام می‌شود. این توابع یک‌بار مصرف هستند و نباید بیش از حد استفاده شوند؛ به جای این‌کار باید از روش‌های استانداردی که هر کلاس BruteForce در اختیارتان قرار می‌دهد استفاده کنید. (همان تابع ()Start)

مثال(ProBrute.GetPassByPos(long, object[], CBrute.Core.ProBrute.PassTestInfo[], int, int))

قصد داریم رمزهایی را با شرایط زیر تولید کنیم:

- تمامی خانه‌های پسورد به صورت پیش‌فرض یکی از حالت‌های {1,2,3,4,5} را می‌پذیرند.
- خانه دوم همیشه عدد 9 است.
- خانه یکی مانده به آخر یکی از حالت‌های {8,7} است.
- کمترین طول پسورد 1 و بیشترین طول پسورد 3 است.

```
// .Net 6
using CBrute.Core;
using CBrute.Helper;

ProBrute.IgnoreTrivialErrors = true;//To avoid ambiguities
ProBrute.PassTestInfo[] testInfos =
{
    new(1, new object[]{ "9" }),
    new(-2, new object[]{ "8","7" }),
};
object[] testArray = "12345".ToCharArray().Cast<object>().ToArray();
int min = 1, max = 3;
long maxNumber = ProBrute.GetMax(testArray, testInfos, min, max);
```

```
for(long pos = 1; pos <= maxNumber; ++pos)
{
    object[] password = ProBrute.GetPassByPos(pos, testArray, testInfos, min, max);
    Console.WriteLine($"{pos} = {password.ConvertStringArrayToString()}");
}
Console.ReadKey(true);
```

خروجی:

```
1 = 1
2 = 2
3 = 3
4 = 4
5 = 5
6 = 89
7 = 79
8 = 191
9 = 192
10 = 193
11 = 194
12 = 195
13 = 291
14 = 292
15 = 293
16 = 294
17 = 295
18 = 391
19 = 392
20 = 393
21 = 394
22 = 395
23 = 491
24 = 492
25 = 493
26 = 494
27 = 495
28 = 591
29 = 592
30 = 593
31 = 594
32 = 595
```

به پسورد شماره 8 دقت کنید، اینجا یک ابهام وجود دارد! انتظار داریم که خانه یکی مانده به آخر، یکی از حالت‌های 8 یا 7 باشد و خانه دوم نیز همیشه باید 9 باشد. از آنجایی که طول پسورد ما در این قسمت 3 است، خانه یکی مانده به آخر با خانه دوم تفاوتی ندارد!

همانطور که مشاهده می‌کنید، از آنجایی در آرایه testInfos، اول حالت‌های خانه دوم را روی 9 تنظیم کرده‌ایم، اولویت با این عنصر از آرایه است و عنصر بعدی(یعنی خانه یکی مانده به آخر) نادیده گرفته می‌شود.

ProBrute.GetPosByPass(object[], object[], CBrute.Core.ProBrute.PassTestInfo[], int, int)

```
public static long GetPosByPass(object[] pass, object[] test,
CBrute.Core.ProBrute.PassTestInfo[] testInfos, int min, int max)
```

```
//    Member of CBrute.Core.ProBrute
```

با استفاده از یک پسورد می‌توانید به موقعیت آن پسورد در بین تمام حالت‌های ممکن برسید.

پارامتر ها

- pass: رمزی که می‌خواهید موقعیت آن‌را محاسبه کنید. به موارد زیر توجه کنید:
  - pass نباید null باشد یا شامل عناصر null باشد.
  - pass نباید خالی باشد.
  - pass نباید شامل عناصری باشد که از شرایط تعریف شده پیروی نکند.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- test: از این آرایه زمانی استفاده می‌شود که ProBrute قصد دارد حالت یکی از خانه‌های پسورد را تغییر دهد ولی حالت‌های این خانه در آرایه testInfos تعیین نشده است. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد. در صورتی‌که به این پارامتر نیاز ندارید از JunkArray استفاده کنید.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- testInfos: با استفاده از این آرایه می‌توانید حالت‌های هر خانه از پسورد را تعیین کنید. ProBrute.PassTestInfo را مطالعه کنید. می‌تواند null باشد.
- min: کمترین طول هر پسورد. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.
  - min نباید بزرگتر از max باشد.
- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.

مقادیری که برمی‌گرداند

- موقعیت رمزی که دریافت کرده است را محاسبه کرده و برمی‌گرداند.

اطلاعات بیشتر

لطفاً تابع قبلی را بررسی کنید و بخش اطلاعات بیشتر را بخوانید.

مثال(ProBrute.GetPosByPass(object[], object[], CBrute.Core.ProBrute.PassTestInfo[], int, int))

در این مثال می‌خواهیم بررسی کنیم که آیا دو تابع GetPosByPass و GetPassByPos درست کار می‌کنند یا خیر:

```
// .Net 6
```

```
using CBrute.Core;

ProBrute.IgnoreTrivialErrors = true;//To avoid ambiguities
ProBrute.PassTestInfo[] testInfos =
{
    new(1, new object[]{ "9" }),
    new(-2, new object[]{ "8","7" }),
    new(3, new object[]{"9999"})
};
object[] testArray = "12345".ToCharArray().Cast<object>().ToArray();
int min = 1, max = 12;
long maxNumber = ProBrute.GetMax(testArray, testInfos, min, max);
for (long pos = 1; pos <= maxNumber; ++pos)
{
    object[] password = ProBrute.GetPassByPos(pos, testArray, testInfos, min, max);
    if (pos != ProBrute.GetPosByPass(password, testArray, testInfos, min, max))
        throw new Exception("NOOOOOO!");
}
Console.WriteLine(":)");
Console.ReadKey(true);
```

خروجی:

```
:)
```

خاصیت‌های کلاس CBrute.Core.ProBrute

ProBrute.IgnoreTrivialErrors

```
public static bool IgnoreTrivialErrors { get; set; }
//     Member of CBrute.Core.ProBrute
```

اگر این خاصیت را True کنید، برای ابهامات موجود در PassTestInfo‌ها هیچ استثنائی پرتاب نمی‌شود.

ProBrute.Test

```
public object[] Test { get; }
//     Member of CBrute.Core.ProBrute
```

این آرایه برای خانه‌هایی استفاده می‌شود که در testInfos مشخص نشده‌اند.

ProBrute.TestInfos

```
public CBrute.Core.ProBrute.PassTestInfo[] TestInfos { get; }
//     Member of CBrute.Core.ProBrute
```

ProBrute.PassTestInfo را مطالعه کنید. می‌تواند null باشد!

ProBrute.ExtraLengths

```
public int[] ExtraLengths { get; }
```

```
//     Member of CBrute.Core.ProBrute
```

اندازه‌هایی را برمی‌گرداند که قرار نیست تولید شوند. می‌تواند null باشد!

ProBrute.RealPos

```
public long RealPos { get; }
//     Member of CBrute.Core.ProBrute
```

در حالی که رمزها توسط کلاس ProBrute در حال تولید هستند، این خاصیت موقعیت واقعی آخرین رمز تولید شده را برمی‌گرداند. تفاوت این موقعیت با پارامتر generated(این پارامتر را در بخش delegateها مشاهده کردید) این است که: پارامتر generated تعداد رمزهای تولید شده را برمی‌گرداند ولی RealPos موقعیت واقعی آخرین رمز تولید شده در بین تمام حالت‌های ممکن را برمی‌گرداند. این یعنی می‌توانید این موقعیت را به تابع GetPassByPos تحویل داده و رمز موجود در آن موقعیت را به دست آورید.

سازنده‌های کلاس CBrute.Core.ProBrute

ProBrute.ProBrute(long, long, int, int, object[],ProBrute.PassTestInfo[], int[], int)

```
public ProBrute(long startPos, long endPos, int minPassLen, int maxPassLen, object[]
test, [CBrute.Core.ProBrute.PassTestInfo[] testInfos = null], [int[] extraPassLengths
= null], [int threadID = -1])
//     Member of CBrute.Core.ProBrute
```

سازنده کلاس ProBrute.

پارامتر ها

- startPos: موقعیتی که تولید رمزها باید از آنجا شروع شود. به موارد زیر توجه کنید:
  - startPos نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر endPos باشد.
- endPos: موقعیتی که تولید رمزها باید در آنجا متوقف شود. برای تولید تمام رمزها می‌توانید مقداری کوچکتر یا مساوی 0 را در این پارامتر قرار دهید. به موارد زیر توجه کنید:
  - endPos نباید بزرگتر از تمامی حالت‌های ممکن باشد.
- minPassLen: کمترین طول پسوردها. به موارد زیر توجه کنید:
  - minPassLen نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر maxPassLen باشد.
- maxPassLen: بیشترین طول پسوردها. به موارد زیر توجه کنید:
  - maxPassLen نباید کوچکتر مساوی 0 باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد. در صورتی‌که به این پارامتر نیاز ندارید از JunkArray استفاده کنید.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.

- پیشنهاد می‌شود که فقط شامل رشته باشد.

- testInfos: با استفاده از این آرایه می‌توانید حالت‌های هر خانه از پسورد را تعیین کنید. ProBrute.PassTestInfo را مطالعه کنید. می‌تواند null باشد.
- extraLengths: در صورتی‌که قصد دارید برخی از اندازه‌های بین min و max را نادیده بگیرید، آن‌ها را در این آرایه قرار دهید. دقت کنید نباید از اعدادی استفاده کنید که در بازه min و max نیست و همچنین نباید طوری این آرایه را پر کنید که هیچ رمزی قابل تولید نباشد! مثلاً اگر min برابر با 3 و max برابر با 5 باشد، نباید این آرایه شامل عناصر 3,4,5 باشد!

یک مثال ساده

می‌خواهیم پسوردهایی تولید کنیم که از شرایط زیر پیروی می‌کنند:

- خانه اول همیشه یکی از حالت‌های 6980 یا 8590 است.
- خانه یکی مانده به آخر همیشه یکی از حالت‌های {1,2,3,4} است.
- بقیه خانه‌ها یکی از حالت‌های {@,#,$,'space',%} را می‌پذیرند.
- حداقل اندازه پسورد 4 کاراکتر و حداکثر اندازه 14 کاراکتر است. بنابراین حداقل طول هر رمز 1 خانه و حداکثر طول هر رمز 11 خانه است.
- نیازی به پسوردهایی با طول 7 نیز نداریم.

قصد داریم رمز "#3@%%%%%8590" را پیدا کنیم.

```
// .Net 6
using CBrute.Core;
using CBrute.Helper;

ProBrute.IgnoreTrivialErrors = true;//To avoid ambiguities
ProBrute.PassTestInfo[] testInfos =
{
    new(0, new object[]{ "6980","8590" }),
    new(-2, new object[]{ "1","2","3","4" })
};
object[] testArray = "@#$ %".ToCharArray().Cast<object>().ToArray();
int min = 1, max = 11;
int[] extraLengths = { 7 };
long startPos = 1, endPos = 0;
ProBrute PB = new(startPos, endPos, min, max, testArray, testInfos, extraLengths);
PB.PasswordGenerated += (sender, pass, generated, total) =>
  pass.ConvertObjectArrayToString().Equals("8590% %%%%%@3#");
PB.OnStart += (sender) => Console.WriteLine("OnStart...");
PB.OnEnd += (sender, pass, result) =>
{
    if (result) Console.WriteLine(pass.ConvertObjectArrayToString());
    else Console.WriteLine("WTF?");
};
PB.Start();
Console.ReadKey(true);
```

خروجی:

```
8590% %%%%%@3#
```

کلاس CBrute.Core.PermutationBrute

با استفاده از این کلاس، می‌توانید جایگشت رشته‌های مختلف را تولید کنید. یکی از مهم ترین ویژگی‌های این کلاس این است که حتی از Minimum password length و Maximum password length نیز پشتیبانی می‌کند. اگر مفهوم جایگشت را درک کرده باشید شاید برایتان عجیب باشد که چگونه می‌توان کمترین طول پسورد و بیشترین طول پسورد را برای یک آرایه از رشته‌ها مانند {"oo","bb","cc"} تعیین کرد؟

من این‌کار را از طریق زیرمجموعه‌گیری انجام دادم. برای مثال برای آرایه {"oo","bb","cc","dd"} می‌توانید کمترین طول را 2 و بیشترین طول را 4 در نظر بگیرید. در این حالت PermutationBrute ابتدا زیر مجموعه‌های 2 عضوی را به دست آورده و تمام جایگشت‌های هر کدام را تولید می‌کند و سپس این‌کار را روی تمام زیرمجموعه‌های 3 و 4 عضوی نیز انجام می‌دهد. شما نمی‌توانید از عددی بیشتر از 4 استفاده کنید زیرا منطقی نیست!

برای جلوگیری از اضافه‌کاری، برخی از توابع و خاصیت‌ها و رویدادها که بین تمامی کلاس‌های BruteForce مشترک هستند را دیگر تعریف نمی‌کنیم و شما باید این بخش را مطالعه کنید!

توابع استاتیک کلاس CBrute.Core.PermutationBrute

PermutationBrute.GetMax(object[], int, int)

```
public static long GetMax(object[] test, int min, int max)
//    Member of CBrute.Core.PermutationBrute
```

با استفاده از این تابع می‌توانید تعداد رمزهای قابل تولید برای یک شرایط خاص را به دست بیاورید.

پارامتر ها

- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- min: کمترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.
  - min نباید بزرگتر از max باشد.
- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.
  - در جایگشت، max نباید از طول test بیشتر باشد.

مقادیری که برمی‌گرداند

- تعداد رمزهای قابل تولید را برای شرایطی که تعریف کردید را محاسبه کرده و برمی‌گرداند.

مثال(PermutationBrute.GetMax(object[], int, int))

این مثال حداکثر تعداد رمزهای قابل تولید را برای شرایط زیر را محاسبه می‌کند:

- رمزها از آرایه { "1", "2", "reza", "null", "5.5", "6", "7", "8" } تشکیل می‌شوند.
- کمترین طول هر رمز 4 و بیشترین طول هر رمز 8 است.

```
// .Net 6
using CBrute.Core;
object[] test = { "1", "2", "reza", "null", "5.5", "6", "7", "8" };
int min = 4, max = 8;
long result = PermutationBrute.GetMax(test, min, max);
Console.WriteLine($"PermutationBrute.GetMax({{ \"1\", \"2\", \"reza\", \"null\",
\"5.5\", \"6\", \"7\", \"8\" }}, {min}, {max}) = {result}");
Console.ReadKey(true);
```

خروجی:

```
PermutationBrute.GetMax({ "1", "2", "reza", "null", "5.5", "6", "7", "8" }, 4, 8) =
109200
```

PermutationBrute.GetPassByPos(long, object[], int, int)

```
public static object[] GetPassByPos(long pos, object[] test, int min, int max)
//    Member of CBrute.Core.PermutationBrute
```

می‌توانید با استفاده از موقعیت یک رمز در بین تمام حالت‌های ممکن، به خود آن رمز برسید.

پارامتر ها

- pos: موقعیت رمزی که می‌خواهید آن را دریافت کنید. به موارد زیر توجه کنید:
  - pos نباید کوچکتر یا مساوی 0 یا بزرگتر از تمام حالت‌های ممکن باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- min: کمترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - min نباید کوچکتر یا مساوی 0 باشد.

- min نباید بزرگتر از max باشد.

- max: بیشترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - max نباید کوچکتر یا مساوی 0 باشد.
  - در جایگشت، max نباید از طول test بیشتر باشد.

مقادیری که برمی‌گرداند

- رمزی که در موقعیت pos وجود دارد را محاسبه کرده و برمی‌گرداند.

اطلاعات بیشتر

این تابع و برادرش را از عمد برای این پیاده سازی کردم که بتوانم کاری کنم تا CBrute از مولتی تردینگ پشتیبانی کند. با این حال شما با استفاده از این تابع و برادرش GetPosByPass می‌توانید فرایند ذخیره و بازیابی عملیات را به راحتی انجام دهید. کافی است که یا رمز را ذخیره کنید، یا موقعیت آن‌را و بعداً یا از طریق رمز به موقعیت برسید، یا برعکس... همچنین باید بدانید که تولید تمامی پسوردها با این توابع اصلاً عاقلانه نیست زیرا بسیار کند انجام می‌شود. این توابع یک‌بار مصرف هستند و نباید بیش از حد استفاده شوند؛ به جای این‌کار باید از روش‌های استانداردی که هر کلاس BruteForce در اختیارتان قرار می‌دهد استفاده کنید. (همان تابع ()Start)

مثال(PermutationBrute.GetPassByPos(long, object[], int, int))

در این مثال می‌خواهیم تمامی پسوردهایی که از شرایط زیر پیروی می‌کنند را تولید کنیم:

- رمزها از آرایه { "1", "2", "reza", "null", "5.5", "6", "7", "8" } تشکیل می‌شوند.
- کمترین طول هر رمز 4 و بیشترین طول هر رمز 5 است.

```
// .Net 6
using CBrute.Core;
using CBrute.Helper;

object[] test = { "1", "2", "reza", "null", "5.5", "6", "7", "8" };
int min = 4, max = 5;
long maxNumber = PermutationBrute.GetMax(test, min, max);
for(long pos = 1; pos <= maxNumber; ++pos)
{
    object[] password = PermutationBrute.GetPassByPos(pos, test, min, max);
    Console.WriteLine($"{pos}. {password.ConvertObjectArrayToString()}");
}
Console.ReadKey(true);
```

خروجی:

```
1. 125.56
2. 1265.5
.
.
.
8399. rezanull867
```

```
8400. rezanull876
```

PermutationBrute.GetPosByPass(object[], object[], int)

```
public static long GetPosByPass(object[] pass, object[] test, int min)
//    Member of CBrute.Core.PermutationBrute
```

با استفاده از یک پسورد می‌توانید به موقعیت آن پسورد در بین تمام حالت‌های ممکن برسید.

پارامتر ها

- pass: رمزی که می‌خواهید موقعیت آن‌را محاسبه کنید. به موارد زیر توجه کنید:
  - pass نباید null باشد یا شامل عناصر null باشد.
  - pass نباید خالی باشد.
  - pass نباید شامل عناصری باشد که در test نیست.
  - در جایگشت طول pass نمی‌تواند از طول test بیشتر باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عنصار null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- min: کمترین طول هر رمز عبور. به موارد زیر توجه کنید:
  - min نباید کوچکتر مساوی 0 یا بزرگتر از طول test باشد.

مقادیری که برمی‌گرداند

- موقعیت رمزی که دریافت کرده است را محاسبه کرده و برمی‌گرداند.

اطلاعات بیشتر

لطفاً تابع قبلی را بررسی کنید و بخش اطلاعات بیشتر را بخوانید.

مثال(PermutationBrute.GetPosByPass(object[], object[], int))

اجازه بدهید تا بررسی کنیم که آیا توابع GetPassByPos و GetPosByPass درست کار می‌کنند یا خیر.

```
// .Net 6
using CBrute.Core;
object[] test = { "1", "2", "reza", "null", "5.5", "6", "7", "8" };
```

```
int min = 1, max = 7;
long maxNumber = PermutationBrute.GetMax(test, min, max);
for(long pos = 1; pos <= maxNumber; ++pos)
{
    object[] password = PermutationBrute.GetPassByPos(pos, test, min, max);
    long getPosByPassVal = PermutationBrute.GetPosByPass(password, test, min);
    if (getPosByPassVal != pos) throw new Exception("NOOOOOOO!");
}
Console.WriteLine(":)");
Console.ReadKey(true);
```

خروجی:

```
:)
```

فیلد های CBrute.Core.PermutationBrute

PermutationBrute.MillisecondsTimeout

```
public static int MillisecondsTimeout
//     Member of CBrute.Core.PermutationBrute
```

وقتی می‌خواهیم زیر مجموعه‌های یک مجموعه را به دست بیاوریم، ممکن است که حافظه RAM به شدت اشغال شود و باعث کاهش عملکرد سیستم بشود. برای جلوگیری از پر شدن RAM یک زمان محدود را در اختیار تردی که زیرمجموعه‌ها را به دست می‌آورد می‌گذاریم تا اگر در آن زمان موفق به انجام کل کار نشد، یک خطا رخ بدهد و حافظه آزاد بشود. شما می‌توانید هر مقداری که دوست دارید را در این فیلد قرار بدهید تا اگر در صورتی که در این زمان تعیین شده کار انجام نشد، حافظه آزاد شود.

خاصیت های CBrute.Core.PermutationBrute

PermutationBrute.Test

```
public object[] Test { get; }
//     Member of CBrute.Core.PermutationBrute
```

حالت‌هایی که ساختار هر پسورد را تشکیل می‌دهند را برمی‌گرداند.

PermutationBrute.RealPos

```
public long RealPos { get; }
//     Member of CBrute.Core.PermutationBrute
```

در حالی که رمزها توسط کلاس PermutationBrute در حال تولید هستند، این خاصیت موقعیت واقعی آخرین رمز تولید شده را برمی‌گرداند. تفاوت این موقعیت با پارامتر generated(این پارامتر را در بخش delegateها مشاهده کردید) این است که پارامتر generated تعداد رمزهای تولید شده را برمی‌گرداند ولی RealPos موقعیت واقعی آخرین رمز تولید شده در بین تمام حالت‌های ممکن را برمی‌گرداند. این یعنی می‌توانید این موقعیت را به تابع GetPassByPos تحویل داده و رمز موجود در آن موقعیت را به دست آورید.

سازنده های کلاس CBrute.Core.PermutationBrute

PermutationBrute.PermutationBrute(long, long, int, int, object[], int)

```
public PermutationBrute(long startPos, long endPos, int minPassLen, int maxPassLen,
object[] test, [int threadID = -1])
//    Member of CBrute.Core.PermutationBrute
```

پارامتر ها

- startPos: موقعیتی که تولید رمزها باید از آنجا شروع شود. به موارد زیر توجه کنید:
  - startPos نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر endPos باشد.
- endPos: موقعیتی که تولید رمزها باید در آنجا متوقف شود. برای تولید تمام رمزها می‌توانید مقداری کوچکتر یا مساوی 0 را در این پارامتر قرار دهید. به موارد زیر توجه کنید:
  - endPos نباید بزرگتر از تمامی حالت‌های ممکن باشد.
- minPassLen: کمترین طول پسوردها. به موارد زیر توجه کنید:
  - minPassLen نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر maxPassLen باشد.
- maxPassLen: بیشترین طول پسوردها. به موارد زیر توجه کنید:
  - maxPassLen نباید کوچکتر مساوی 0 یا بزرگتر از طول پارامتر test باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- threadID: در صورتی که می‌خواهید از روش خودتان برای پیاده‌سازی مولتی تردینگ استفاده کنید، این پارامتر نشان دهنده ID تردی است که نمونه در حال ایجاد قرار است در آن اجرا شود.

یک مثال ساده

در این مثال قصد داریم یک رمز عبور که از شرایط زیر پیروی می‌کند را پیدا کنیم:

رمز عبور بین 4 تا 6 خانه است. رمز عبور می‌تواند ترکیبی از تاریخ تولد، شماره تلفن و... باشد. می‌خواهیم تمام حالت‌های ممکن را تست کنیم.(رمز:reza09188113null82)

از آنجایی که هیچ عنصر تکراری در این پسورد وجود ندارد(هر عنصر رنگش متفاوت است)، می‌توان خیلی سریعتر با استفاده از PermutationBrute به نتیجه رسید.

```
// .Net 6
using CBrute.Core;
using CBrute.Helper;
using System.Reflection;
```

```
object[] testArray = { "13", "81", "31", "18", "0939", "0918", "0938","82","0992",
"1382", "reza","null" };
int min = 4, max = 6;
long startPos = 1, endPos = 0;
PermutationBrute PEB = new PermutationBrute(startPos, endPos, min, max, testArray);
PEB.OnStart += (sender) => Console.WriteLine("OnStart...");
PEB.PasswordGenerated += (sender, pass, generated, total) =>
pass.ConvertObjectArrayToString().Equals("reza09188113null82");
PEB.OnEnd += (sender, pass, result) =>
{
    if (result) Console.WriteLine(pass.ConvertObjectArrayToString());
    else Console.WriteLine("WTF!?");
};
PEB.Start();
Console.ReadKey(true);
```

خروجی:

```
reza09188113null82
```

فضای نام CBrute.Worker

این فضای نام شامل کلاس‌هایی است که برای موازی سازی فرایند تولید رمزها استفاده می‌شوند. تمامی کلاس‌های Worker در CBrute از قابلیت‌های وقفه و ازسرگیری و توقف و... پشتیبانی می‌کنند. شاید Worker نام مناسبی برای این نوع از کلاس‌ها نباشد ولی نیازی نیست که خودتان را درگیر این مسئله بکنید زیرا نام‌ها فقط برای صدا زدن هستند :)

Delegate های موجود در فضای نام CBrute.Worker

CBrute.Worker.delegate_OnThreadStart

```
public delegate void delegate_OnThreadStart(CBrute.Worker.Worker sender,
CBrute.Core.BruteForce brute)
//    Member of CBrute.Worker
```

برای مدیریت کردن رویداد OnThreadStart استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- brute: نمونه‌ای از کلاس BruteForce که رویداد OnStart در آن نمونه رخ داده است.

CBrute.Worker.delegate_OnThreadPauseOrResume

```
public delegate void delegate_OnThreadPauseOrResume(CBrute.Worker.Worker sender,
CBrute.Core.BruteForce brute, long generated, long total)
//    Member of CBrute.Worker
```

برای مدیریت کردن رویدادهای OnThreadPause و OnThreadResume استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- brute: نمونه‌ای از کلاس BruteForce که رویداد OnPause یا OnResume در آن نمونه رخ داده است.
- generated: تعداد رمزهایی که تولید شده در ترد.
- total: مجموع رمزهای قابل تولید در ترد.

CBrute.Worker.delegate_OnThreadStopOrRestart

```
public delegate void delegate_OnThreadStopOrRestart(CBrute.Worker.Worker sender,
CBrute.Core.BruteForce brute, object[] pass)
//    Member of CBrute.Worker
```

برای مدیریت کردن رویدادهای OnThreadStop و OnThreadRestart استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- brute: نمونه‌ای از کلاس BruteForce که رویداد OnStop یا OnRestart در آن نمونه رخ داده است.
- pass: آخرین پسورد تولید شده در ترد.

CBrute.Worker.delegate_OnThreadEnd

```
public delegate bool delegate_OnThreadEnd(CBrute.Worker.Worker sender,
CBrute.Core.BruteForce brute, object[] pass, bool result)
//    Member of CBrute.Worker
```

برای مدیریت کردن رویداد OnThreadEnd استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- brute: نمونه‌ای از کلاس BruteForce که رویداد OnStop یا OnRestart در آن نمونه رخ داده است.
- pass: آخرین پسورد تولید شده در ترد.
- result: اگر True باشد یعنی رمز عبور پیدا شده، در غیر اینصورت یعنی فرایند شکست خورده.

مقادیری که برمی‌گرداند

- اگر قصد دارید فعالیت تردهای دیگر هم به پایان برسد، True را برگردانید. بهترین کار این است که همیشه مقدار پارامتر result را برگردانید.

CBrute.Worker.delegate_OnThreadError

```
public delegate void delegate_OnThreadError(CBrute.Worker.Worker sender,
CBrute.Core.BruteForce brute, System.Exception e, ref bool tryAgain)
//    Member of CBrute.Worker
```

برای مدیریت رویداد OnThreadError استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- brute: نمونه‌ای از کلاس BruteForce که رویداد OnStop یا OnRestart در آن نمونه رخ داده است.
- e: شامل اطلاعاتی راجع‌به خطای رخ داده است.
- tryAgain: این مقدار را بسته به نیاز خود تعیین دهید. با استفاده از ارجاع ارسال شده است.

مقادیری که برمی‌گرداند

- اگر قصد دارید یک بار دیگر حالتی که خطا ایجاد کرده است را امتحان کنید، مقدارش را True بگذارید و اگر نمی‌توانید خطا را مدیریت کنید مقدارش را False کنید.

CBrute.Worker.delegate_WrokerEvent

```
public delegate void delegate_WrokerEvent(CBrute.Worker.Worker sender)
//    Member of CBrute.Worker
```

برای مدیریت برخی از رویدادهای مرتبط با Worker‌ها استفاده می‌شود. OnStart ,OnPause

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.

CBrute.Worker.delegate_OnError

```
public delegate void delegate_OnError(CBrute.Worker.Worker sender, System.Exception
lastEx)
//    Member of CBrute.Worker
```

برای مدیریت رویداد وحشتناک OnError است.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- lastEX: آخرین خطایی که رخ داده است!

CBrute.Worker.delegate_CheckPassword

```
public delegate bool delegate_CheckPassword(CBrute.Worker.Worker sender,
CBrute.Core.BruteForce brute, object[] pass, long generated, long total)
//    Member of CBrute.Worker
```

برای پارامتر check در تمامی سازنده‌های کلاس‌های Worker استفاده می‌شود. با استفاده از این delegate می‌توانید رمزهای عبور تولید شده توسط ترد را بررسی کنید. دقت کنید که بهتر است از منابع مشترک استفاده نکنید زیرا به عملکرد تردهای دیگر صدمه می‌زند.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- brute: نمونه‌ای از کلاس BruteForce که رویداد PasswordGenerated در آن نمونه رخ داده است.
- pass: آخرین پسورد تولید شده در ترد.
- generated: تعداد رمزهایی که تولید شده در ترد.
- total: مجموع رمزهای قابل تولید در ترد.

مقادیری که برمی‌گرداند

- در صورتی که توانستید رمزعبور مورد نظر خود را پیدا کنید، True را برگردانید و در غیر اینصورت False.

CBrute.Worker.delegate_OnStopOrEnd

```
public delegate void delegate_OnStopOrEnd(CBrute.Worker.Worker sender, bool result)
//    Member of CBrute.Worker
```

برای مدیریت رویدادهای OnEnd و OnStop استفاده می‌شود.

پارامتر ها

- sender: شیئی که باعث ایجاد رویداد شده.
- result: نشان‌دهنده موفق بودن یا نبودن فرایند است.

کلاس CBrute.Worker.Worker

یک کلاس انتزاعی برای تمامی کلاس‌هایی که قصد دارند فرایند تولید پسوردها را به صورت موازی انجام دهند. این کلاس‌ها وابسته به کلاس‌های موجود در فضای نام CBrute.Core هستند. برای چند ریختی مفید است.

کلاس‌هایی که از Worker مشتق می‌شوند از extraLengths ها پشتیبانی نمی‌کنند!

فیلد های کلاس CBrute.Worker.Worker

Worker.check

```
protected CBrute.Worker.delegate_CheckPassword check
//    Member of CBrute.Worker.Worker
```

تمامی کلاس‌هایی که از Worker مشتق می‌شوند باید توسط این فیلد درستی پسورد تولید شده را بررسی کنند. برای این‌کار شما باید رویداد PasswordGenerated تمامی نمونه‌های کلاس BruteForce(منظورم آن کلاس‌هایی است که از BruteForce مشتق می‌شوند!) را توسط این فیلد مدیریت کنید.

Worker.endCounter

```
protected System.Threading.CountdownEvent endCounter
//    Member of CBrute.Worker.Worker
```

برای اینکه رویداد OnEnd رخ بدهد، نیاز داریم مطمئن شویم که تمامی تردهای دیگر نیز کارشان تمام شده است. هر بار که رویداد OnThreadEnd رخ می‌دهد باید توسط این فیلد گزارش شود. این فیلد همیشه مقدار پایه‌اش تعداد تردهای موجود است.

Worker.errorCounter

```
protected System.Threading.CountdownEvent errorCounter
//    Member of CBrute.Worker.Worker
```

برای اینکه رویداد OnError رخ بدهد، نیاز داریم مطمئن شویم که تمامی تردهای دیگر نیز با خطا روبه‌رو شده‌اند. هر بار که رویداد OnThreadErorr رخ می‌دهد باید توسط این فیلد گزارش شود. این فیلد همیشه مقدار پایه‌اش تعداد تردهای موجود است.

Worker.foundedCounter

```
protected System.Threading.CountdownEvent foundedCounter
//    Member of CBrute.Worker.Worker
```

تعداد پسوردهای پیدا شده توسط تردهای مختلف را تعیین می‌کند. زمانی که یک ترد رمزی را پیدا می‌کند باید یک سیگنال بفرستد.

Worker.pauseCounter

```
protected System.Threading.CountdownEvent pauseCounter
//    Member of CBrute.Worker.Worker
```

برای اینکه رویداد OnPause رخ بدهد، نیاز داریم مطمئن شویم که تمامی تردهای دیگر نیز موقتاً متوقف شده‌اند. هر بار که رویداد OnThreadPause رخ می‌دهد باید توسط این فیلد گزارش شود. این فیلد همیشه مقدار پایه‌اش تعداد تردهای موجود است.

Worker.resumeCounter

```
protected System.Threading.CountdownEvent resumeCounter
```

```
//     Member of CBrute.Worker.Worker
```

برای اینکه رویداد OnResume رخ بدهد، نیاز داریم مطمئن شویم که تمامی تردهای دیگر نیز فعالیت را ادامه می‌دهند. هر بار که رویداد OnThreadResume رخ می‌دهد باید توسط این فیلد گزارش شود. این فیلد همیشه مقدار پایه‌اش تعداد تردهای موجود است.

Worker.stopCounter

```
protected System.Threading.CountdownEvent stopCounter
//     Member of CBrute.Worker.Worker
```

برای اینکه رویداد OnStop رخ بدهد، نیاز داریم مطمئن شویم که تمامی تردهای دیگر نیز فعالیت را متوقف کرده‌اند. هر بار که رویداد OnThreadStop رخ می‌دهد باید توسط این فیلد گزارش شود. این فیلد همیشه مقدار پایه‌اش تعداد تردهای موجود است.

Worker.startPos

```
protected long startPos
//     Member of CBrute.Worker.Worker
```

برای تعیین موقعیت شروع تولید رمزها استفاده می‌شود.

Worker.endPos

```
protected long endPos
//     Member of CBrute.Worker.Worker
```

موقعیت آخرین رمز قابل تولید. startPos و endPos محدوده تولید رمزها را مشخص می‌کنند.

Worker.min

```
protected int min
//     Member of CBrute.Worker.Worker
```

کمترین طول پسوردها را مشخص می‌کند. نباید بیشتر از max باشد. نباید کوچکتر مساوی 0 باشد.

Worker.max

```
protected int max
//     Member of CBrute.Worker.Worker
```

بیشترین طول پسوردها را مشخص می‌کند. نباید کوچکتر مساوی 0 باشد.

Worker.started

```
protected bool started
//     Member of CBrute.Worker.Worker
```

در صورتی‌که فرایند آماده شروع شدن است، این فیلد باید True شود.

Worker.stoped

```
protected bool stoped
//    Member of CBrute.Worker.Worker
```

در صورتی‌که فرایند تولید پسوردها باید متوقف بشود، این فیلد باید True شود.

Worker.pause

```
protected bool pause
//    Member of CBrute.Worker.Worker
```

در صورتی‌که می‌خواهید در فرایند تولید پسوردها وقفه ایجاد کنید، مقدار این فیلد را True کنید. در صورتی که می‌خواهید فرایند تولید رمزها ادامه یابد مقدار این فیلد را False کنید.

Worker.threadCount

```
protected int threadCount
//    Member of CBrute.Worker.Worker
```

تعداد تردهایی که می‌خواهید کار بین آن‌ها تقسیم بشود. نمی‌تواند منفی باشد یا از تمام حالت‌های ممکن برای تولید رمزها بیشتر باشد.

Worker.list

```
protected System.Collections.Generic.List<CBrute.Core.BruteForce> list
//    Member of CBrute.Worker.Worker
```

لیستی از نمونه‌های کلاس BruteForce که هر کدام از آن‌ها باید در ترد جداگانه‌ای اجرا شود. با توجه به threadCount باید این لیست را پر کنید.

خاصیت های کلاس CBrute.Worker.Worker

Worker.EndPos

```
public long EndPos { get; }
//    Member of CBrute.Worker.Worker
```

موقعیت آخرین رمزی‌که قرار است تولید شود را برمی‌گرداند.

Worker.MaximumPassLength

```
public int MaximumPassLength { get; }
//    Member of CBrute.Worker.Worker
```

بیشترین طول پسوردها را مشخص می‌کند.

Worker.MinimumPassLength

```
public int MinimumPassLength { get; }
//     Member of CBrute.Worker.Worker
```

کمترین طول پسوردها را مشخص می‌کند.

Worker.Pause

```
public bool Pause { get; set; }
//     Member of CBrute.Worker.Worker
```

با استفاده از این خاصیت می‌توانید در فرایند تولید رمزها وقفه ایجاد کنید یا فرایند Pauseشده را ادامه دهید. برای ایجاد وقفه در فرایند تولید رمزها، باید این خاصیت را True کنید. برای ازسرگیری فرایند نیز این خاصیت را False کنید.

Worker.Started

```
public bool Started { get; }
//     Member of CBrute.Worker.Worker
```

زمانی که فرایند تولید رمزها در حال انجام باشد این خاصیت True را برمی‌گرداند.

Worker.StartPos

```
public long StartPos { get; }
//     Member of CBrute.Worker.Worker
```

موقعیت اولین رمزی که قرار است تولید بشود را برمی‌گرداند.

Worker.ThreadCount

```
public int ThreadCount { get; }
//     Member of CBrute.Worker.Worker
```

تعداد تردهایی که قرار است تولید بشوند را برمی‌گرداند.

Worker.this[int]

```
public CBrute.Core.BruteForce this[int threadID] { get; }
//     Member of CBrute.Worker.Worker
```

با استفاده از این Indexer می‌توانید به نمونه‌هایی از کلاس BruteForce که در تردهای دیگری هستند دسترسی داشته باشید. کمترین مقدار قابل قبول 0 و بیشترین مقدار قابل قبول ThreadCount-1 است. هدف از پیاده‌سازی این Indexer این بود که بتوانید در صورت نیاز StartPos و EndPos هر ترد را تغییر بدهید.

توابع کلاس CBrute.Worker.Worker

Worker.theEnd()

```
protected void theEnd()
//     Member of CBrute.Worker.Worker
```

این تابع برای از بین بردن تمامی تردهای باقی مانده استفاده می‌شود. زمانی از این تابع استفاده می‌شود که یک ترد موفق به کشف رمز شده است و کاربر دستور پایان را در رویداد OnThreadEnd صادر کرده باشد.

Worker.stopThreads()

```
protected void stopThreads()
//     Member of CBrute.Worker.Worker
```

این تابع تمامی تردهای باقی مانده را متوقف می‌کند.

Worker.Stop()

```
public void Stop()
//     Member of CBrute.Worker.Worker
```

این تابع برای متوقف کردن فرایند تولید رمزها استفاده می‌شود. مدتی بعد از فراخوانی این تابع رویداد OnStop رخ می‌دهد.

Worker.pauseThreads()

```
protected void pauseThreads()
//     Member of CBrute.Worker.Worker
```

این تابع برای True کردن خاصیت Pause تمامی نمونه‌های کلاس BruteForce در آرایه list استفاده می‌شود.

Worker.initializeCounters()

```
protected void initializeCounters()
//     Member of CBrute.Worker.Worker
```

این تابع تمامی CountdownEvent ها را مقدار دهی می‌کند. مقداری که برای آن‌ها در نظر گرفته شده است ThreadCount است.

Worker.getRanges(long, long, int)

```
protected static long[] getRanges(long start, long end, int threadCount)
//     Member of CBrute.Worker.Worker
```

این تابع برای تقسیم کار بین تردها استفاده می‌شود.

پارامتر ها

- start: موقعیتی که تولید رمزها از آن شروع می‌شود.
- end: موقعیت آخرین رمز قابل تولید.
- threadCount: تعداد تردهایی که می‌خواهید کار را بین آن‌ها تقسیم کنید.

مقادیری که برمی‌گرداند

- این تابع محدوده بین start و end را بین چندین ترد که تعداد آن‌ها توسط threadCount تعیین شده، تقسیم می‌کند. یک آرایه تک بعدی برمی‌گرداند که هر عنصر در ایندکس زوج نشان‌دهنده startPos و هر ایندکس فرد نشان‌دهنده endPos است.

Worker.DoWork(bool)

```
public abstract bool DoWork(bool waitForWork)
//     Member of CBrute.Worker.Worker
```

این تابع برای شروع فرایند تولید رمزها به صورت موازی استفاده می‌شود.

پارامتر ها

- waitForWork: در صورتی که قصد دارید منتظر بمانید تا فرایند تولید رمزها به پایان برسد، مقدار این پارامتر را True کنید.

مقادیری که برمی‌گرداند

- در صورتی‌که تابع کار خود را با موفقیت انجام داده و فرایند تولید رمزها را به صورت موازی شروع کند، مقدار True را برمی‌گرداند.

Worker.Dispose()

```
public void Dispose()
//     Member of CBrute.Worker.Worker
```

این تابع برای از بین بردن ترد ها با متوقف کردن آن‌ها استفاده می‌شود. **حواستان باشد که یک وقت نمونه‌ای که ساختید توسط زباله روب از بین نرود.**

رویداد های مشترک CBrute.Worker.Worker

تمام رویدادهای کلاس‌هایی که Worker را بسط می‌دهند مشترک است. برای جلوگیری از اضافه‌کاری در این قسمت آن رویدادها را شرح می‌دهیم.

OnStart

```
public event CBrute.Worker.delegate_WrokerEvent OnStart
//     Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد قبل از اینکه فرایند تولید رمزها به صورت موازی شروع بشود رخ می‌دهد.

OnPause

```
public event CBrute.Worker.delegate_WrokerEvent OnPause
//    Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که خاصیت Pause تغییر کرده و روی True تنظیم شده است؛ به عبارتی دیگر، این رویداد زمانی رخ می‌دهد که در فرایند تولید رمزها یک وقفه ایجاد شده باشد.

OnResume

```
public event CBrute.Worker.delegate_WrokerEvent OnResume
//    Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که خاصیت Pause تغییر کرده و روی False تنظیم شده است؛ به عبارتی دیگر، این رویداد زمانی رخ می‌دهد که فرایند تولید رمزها ادامه یافته باشد.

OnStop

```
public event CBrute.Worker.delegate_OnStopOrEnd OnStop
//    Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که فرایند تولید رمزها به صورت کامل متوقف شده باشد.

OnEnd

```
public event CBrute.Worker.delegate_OnStopOrEnd OnEnd
//    Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که فرایند تولید رمزها به پایان رسیده باشد.

OnError

```
public event CBrute.Worker.delegate_OnError OnError
//    Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که کل تردها با خطا روبه‌رو شوند.

OnThreadStart

```
public event CBrute.Worker.delegate_OnThreadStart OnThreadStart
//    Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد قبل از اینکه یک ترد تولید رمزها را شروع کند رخ می‌دهد.

OnThreadPause

```
public event CBrute.Worker.delegate_OnThreadPauseOrResume OnThreadPause
```

```
//     Member of CBrute.Worker.SimpleBruteWorker
```

زمانی که یک ترد pause می‌شود، این رویداد رخ می‌دهد.

OnThreadResume

```
public event CBrute.Worker.delegate_OnThreadPauseOrResume OnThreadResume
//     Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که یک ترد Pause شده فعالیت خود را ادامه بدهد.

OnThreadStop

```
public event CBrute.Worker.delegate_OnThreadStopOrRestart OnThreadStop
//     Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌هد که یک ترد کاملاً متوقف شده باشد.

OnThreadEnd

```
public event CBrute.Worker.delegate_OnThreadEnd OnThreadEnd
//     Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که کار یک ترد تمام شده باشد.

OnThreadError

```
public event CBrute.Worker.delegate_OnThreadError OnThreadError
//     Member of CBrute.Worker.SimpleBruteWorker
```

این رویداد زمانی رخ می‌دهد که یک ترد با خطا روبه‌رو شده باشد.

کلاس CBrute.Worker.SimpleBruteWorker

این کلاس برای موازی کردن فرایند تولید رمزها به وسیله SimpleBrute پیاده‌سازی شده است.

برای جلوگیری از اضافه‌کاری، برخی از توابع و خاصیت‌ها و رویدادها که بین تمامی کلاس‌های Worker مشترک هستند را دیگر تعریف نمی‌کنیم و شما باید این بخش را مطالعه کنید!

خاصیت های کلاس CBrute.Worker.SimpleBruteWorker

SimpleBruteWorker.Test

```
public object[] Test { get; }
//     Member of CBrute.Worker.SimpleBruteWorker
```

حالت‌هایی که ساختار هر پسورد را تشکیل می‌دهند را برمی‌گرداند.

سازنده‌های کلاس CBrute.Worker.SimpleBruteWorker

SimpleBruteWorker.SimpleBruteWorker(long, long, int, int, object[], int, CBrute.Worker.delegate_CheckPassword)

```
public SimpleBruteWorker(long startPos, long endPos, int minimumPasswordLength, int
maximumPasswordLength, object[] test, int threadCount,
CBrute.Worker.delegate_CheckPassword passChecker)
//    Member of CBrute.Worker.SimpleBruteWorker
```

سازنده کلاس CBrute.Worker.SimpleBruteWorker.

پارامتر ها

- startPos: موقعیتی که تولید رمزها باید از آنجا شروع شود. به موارد زیر توجه کنید:
  - startPos نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر endPos باشد.
- endPos: موقعیت آخرین رمز قابل تولید. برای تولید تمام رمزها می‌توانید مقداری کوچکتر یا مساوی 0 را در این پارامتر قرار دهید. به موارد زیر توجه کنید:
  - endPos نباید بزرگتر از تمامی حالت‌های ممکن باشد.
- minimumPasswordLength: کمترین طول پسوردها. به موارد زیر توجه کنید:
  - minimumPasswordLength نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر maximumPasswordLength باشد.
- maximumPasswordLength: بیشترین طول پسوردها. به موارد زیر توجه کنید:
  - maximumPasswordLength نباید کوچکتر مساوی 0 باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- threadCount: تعداد تردهایی که می‌خواهید کار بین آن‌ها تقسیم بشود. به موارد زیر توجه کنید:
  - threadCount نباید کوچکتر مساوی 0 یا بزرگتر از تعداد تمام حالت‌های ممکن باشد.
- passChecker: با استفاده از این Delegate می‌توانید رمزهایی که توسط همه تردها تولید می‌شوند را بررسی کنید. حواستان باشد که در این تابع چه کاری می‌کنید زیرا چندین ترد به صورت همزمان این تابع را اجرا می‌کنند.

یک مثال ساده

خب در این مثال قصد داریم که یک پسورد لیست با شرایط زیر را تولید کنیم:

- حالت‌های تشکیل دهنده هر خانه از پسورد: {1,2,3,4,5,6,7,8,9, 0}
- کمترین اندازه هر رمز 4 و بیشترین اندازه 8 است.

- فایل خروجی passlist.txt نام دارد

با فشردن کلید p می‌توانید در فعالیت وقفه ایجاد کنید یا آن را ادامه دهید. با استفاده از x می‌توانید فعالیت را متوقف کنید.

هشدار: این برنامه RAM را به شدت درگیر می‌کند!

```
using CBrute.Helper;
using CBrute.Worker;

namespace Examples
{
    public static class Program
    {
        static readonly object locker = new();
        static bool started = false;
        static readonly object[] testArray =
"A1234567890B".ToCharArray().Cast<object>().ToArray();
        private static readonly int min = 4;
        private static readonly int max = 6;
        private static readonly long startPos = 1;
        private static readonly long endPos = 0;
        static readonly SimpleBruteWorker SBW = new(startPos, endPos, min, max,
testArray, Environment.ProcessorCount * 2, PassCheck);
        [STAThread]
        public static void Main()
        {
            SBW.OnThreadStart += (worker, brute) => brute.Tag ??= new
StreamWriter(new MemoryStream()) { AutoFlush = false };
            SBW.OnPause += (sender) => Console.WriteLine("SBW_OnPause");
            SBW.OnResume += (sender) => Console.WriteLine("SBW_OnResume");
            SBW.OnStop += SBW_OnStop;
            SBW.OnEnd += SBW_OnEnd;
            SBW.DoWork(false);
            started = true;
            while (true)
            {
                lock (locker)
                    if (!started) break;
                if (Console.KeyAvailable)
                {
                    var key = Console.ReadKey(true);
                    while (Console.KeyAvailable) Console.ReadKey(true);
                    switch (key.KeyChar)
                    {
                        case 'p':
                            SBW.Pause = !SBW.Pause;
                            break;
                        case 'x':
                            SBW.Stop();
                            break;
                    }
                }
            }
```

```
            Console.WriteLine("The end...");
            Console.ReadKey(true);
        }

        private static void SBW_OnStop(Worker sender, bool result)
        {
            for (int threadID = 0; threadID < SBW.ThreadCount; ++threadID)
            {
                StreamWriter temp = (SBW[threadID].Tag as StreamWriter)!;
                SBW[threadID].Tag = null;
                temp.Close();
                GC.Collect();
            }
            lock (locker) started = false;
            Console.WriteLine("SBW_OnStop");
        }

        private static bool PassCheck(Worker sender, CBrute.Core.BruteForce brute,
object[] pass, long generated, long total)
        {
            StreamWriter SW = (brute.Tag as StreamWriter)!;
            SW.WriteLine(pass.ConvertObjectArrayToString());
            SW.Flush();
            return false;
        }
        private static void SBW_OnEnd(Worker sender, bool result)
        {
            using (StreamWriter writer = new($"passlist.txt", false))
            {
                for (int threadID = 0; threadID < SBW.ThreadCount; ++threadID)
                {
                    StreamWriter threadStream = (SBW[threadID].Tag as StreamWriter)!;
                    threadStream.BaseStream.Seek(0, SeekOrigin.Begin);
                    StreamReader reader = new(threadStream.BaseStream);
                    do writer.WriteLine(reader.ReadLine());
                    while (!reader.EndOfStream);
                    threadStream.Close();
                    SBW[threadID].Tag = null;
                    reader.Close();
                    GC.Collect();
                }
            }
            lock (locker) started = false;
        }
    }
}
```

چی شد؟

ساده‌ترین برنامه‌ای که می‌توانید برای ساخت پسوردلیست بسازید، همین برنامه است ولی بهینه‌ترین نیست!

در ابتدا ما باید پیش‌نیازهای تولید پسوردها را آماده کنیم و سپس باید یک نمونه از کلاس SimpleBruteWorker را بسازیم. منظور از پیش‌نیاز، همان testArray و min و max و... است. در مرحله بعدی باید برخی از رویدادهای نمونه‌های ساخته شده را مدیریت کنیم.

ما اداره‌کننده رویدادهای Pause و Resume را به ساده‌ترین شکل ممکن پیاده‌سازی کردیم ولی رویدادهای End و Stop را با کدهای بیشتری مدیریت کردیم. در یک حلقه بی‌نهایت ما ورودی‌های کاربر را در ترد اصلی مدیریت می‌کنیم. هر ترد باید پسوردها را در یک MemoryStream بنویسد تا کار سریعتر انجام بشود.

در نهایت، پس اتمام کار(رویداد OnEnd) باید تمامی تردها را به ترتیب به دست آورده و اطلاعاتی که در حافظه نوشته شده است را به فایل passlist.txt انتقال دهیم.

دقت کنید که همیشه پس از اتمام کار باید استریم‌های باز را ببندیم و منابع سیستم را آزاد کنیم.

کلاس CBrute.Worker.ProBruteWorker

این کلاس برای موازی کردن فرایند تولید رمزها به وسیله ProBrute پیاده‌سازی شده است.

برای جلوگیری از اضافه‌کاری، برخی از توابع و خاصیت‌ها و رویدادها که بین تمامی کلاس‌های Worker مشترک هستند را دیگر تعریف نمی‌کنیم و شما باید این بخش را مطالعه کنید!

خاصیت های کلاس CBrute.Worker.ProBruteWorker

ProBruteWorker.Test

```
public object[] Test { get; }
//    Member of CBrute.Worker.ProBruteWorker
```

حالت‌هایی که ساختار هر پسورد را تشکیل می‌دهند را برمی‌گرداند.

ProBruteWorker.TestInfos

```
public CBrute.Core.ProBrute.PassTestInfo[] TestInfos { get; }
//    Member of CBrute.Worker.ProBruteWorker
```

یک آرایه از کلاس PassTestInfo.

سازنده های کلاس CBrute.Worker.ProBruteWorker

ProBruteWorker.ProBruteWorker(long, long, int, int, object[], CBrute.Core.ProBrute.PassTestInfo[], int, CBrute.Worker.delegate_CheckPassword)

```
public ProBruteWorker(long startPos, long endPos, int minimumPasswordLength, int
maximumPasswordLength, object[] test, CBrute.Core.ProBrute.PassTestInfo[] testInfos,
int threadCount, CBrute.Worker.delegate_CheckPassword passChecker)
//    Member of CBrute.Worker.ProBruteWorker
```

پارامتر ها

- startPos: موقعیتی که تولید رمزها باید از آنجا شروع شود. به موارد زیر توجه کنید:
  - startPos نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر endPos باشد.
- endPos: موقعیتی که تولید رمزها باید در آنجا متوقف شود. برای تولید تمام رمزها می‌توانید مقداری کوچکتر یا مساوی 0 را در این پارامتر قرار دهید. به موارد زیر توجه کنید:
  - endPos نباید بزرگتر از تمامی حالت‌های ممکن باشد.
- minimumPasswordLength: کمترین طول پسوردها. به موارد زیر توجه کنید:
  - minimumPasswordLength نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر maximumPasswordLength باشد.
- maximumPasswordLength: بیشترین طول پسوردها. به موارد زیر توجه کنید:
  - maximumPasswordLength نباید کوچکتر مساوی 0 باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد. در صورتی‌که به این پارامتر نیاز ندارید از JunkArray استفاده کنید.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.
- testInfos: با استفاده از این آرایه می‌توانید حالت‌های هر خانه از پسورد را تعیین کنید. ProBrute.PassTestInfo را مطالعه کنید. می‌تواند null باشد.
- threadCount: تعداد تردهایی که می‌خواهید کار بین آن‌ها تقسیم بشود. به موارد زیر توجه کنید:
  - threadCount نباید کوچکتر مساوی 0 یا بزرگتر از تعداد تمام حالت‌های ممکن باشد.
- passChecker: با استفاده از این Delegate می‌توانید رمزهایی که توسط همه تردها تولید می‌شوند را بررسی کنید. حواستان باشد که در این تابع چه کاری می‌کنید زیرا چندین ترد به صورت همزمان این تابع را اجرا می‌کنند.

یک مثال ساده

شما می‌توانید مثال قبلی را به راحتی از کلاس SimpleBruteWorker به ProBruteWorker تغییر بدهید و پارامترهای مورد نیاز را وارد کنید و برنامه را اجرا کنید.

کلاس CBrute.Worker.PermutationBruteWorker

این کلاس برای موازی کردن فرایند تولید رمزها به وسیله PermutationBrute پیاده‌سازی شده است.

برای جلوگیری از اضافه‌کاری، برخی از توابع و خاصیت‌ها و رویدادها که بین تمامی کلاس‌های Worker مشترک هستند را دیگر تعریف نمی‌کنیم و شما باید این بخش را مطالعه کنید!

خاصیت های کلاس CBrute.Worker.PermutationBruteWorker

PermutationBruteWorker.Test

```
public object[] Test { get; }
//    Member of CBrute.Worker.PermutationBruteWorker
```

حالت‌هایی که ساختار هر پسورد را تشکیل می‌دهند را برمی‌گرداند.

سازنده های کلاس CBrute.Worker.PermutationBruteWorker

PermutationBruteWorker.PermutationBruteWorker(long, long, int, int, object[], int, CBrute.Worker.delegate_CheckPassword)

```
public PermutationBruteWorker(long startPos, long endPos, int minimumPasswordLength,
int maximumPasswordLength, object[] test, int threadCount,
CBrute.Worker.delegate_CheckPassword passChecker)
//    Member of CBrute.Worker.PermutationBruteWorker
```

پارامتر ها

- startPos: موقعیتی که تولید رمزها باید از آنجا شروع شود. به موارد زیر توجه کنید:
  - startPos نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر endPos باشد.
- endPos: موقعیتی که تولید رمزها باید در آنجا متوقف شود. برای تولید تمام رمزها می‌توانید مقداری کوچکتر یا مساوی 0 را در این پارامتر قرار دهید. به موارد زیر توجه کنید:
  - endPos نباید بزرگتر از تمامی حالت‌های ممکن باشد.
- minimumPasswordLength: کمترین طول پسوردها. به موارد زیر توجه کنید:
  - minimumPasswordLength نباید کوچکتر مساوی 0 یا بزرگتر از پارامتر maximumPasswordLength باشد.
- maximumPasswordLength: بیشترین طول پسوردها. به موارد زیر توجه کنید:
  - maximumPasswordLength نباید کوچکتر مساوی 0 یا بزرگتر از طول پارامتر test باشد.
- test: حالت‌هایی که پسوردها را تشکیل می‌دهند. به موارد زیر توجه کنید:
  - test نباید null باشد یا شامل عناصر null باشد.
  - test نباید خالی باشد.
  - test نباید شامل عناصر تکراری باشد.
  - پیشنهاد می‌شود که فقط شامل رشته باشد.

یک مثال ساده

شما می‌توانید مثال اول را به راحتی از کلاس SimpleBruteWorker به PermutationBruteWorker تغییر بدهید و پارامترهای مورد نیاز را وارد کنید و برنامه را اجرا کنید.