

استفاده از عملگر == برای مقایسه اعداد اعشاری عموماً جواب نخواهد داد و کار صحیحی نیست. از این جهت که اعداد، اساساً به صورت یک سری صفر و یک ذخیره شده و امکان ذخیره سازی کامل و دقیق قسمت اعشاری وجود ندارد. برای مثال نوع‌های double و float امکان ذخیره سازی دقیق عدد یک دهم را ندارند. عدد 10/1 به صورت 0.000110011001100... ذخیره می‌شود (در حالت باینری) و مقایسه دقیق مقادیر ثابت 0.00011 یا 0.00011001100 با آن میسر نیست؛ چون دقت نهایی این اعداد متفاوت است. در زبان C#، نوع double، مطابق استاندارد IEEE-754 تهیه شده‌است و تنها 15 رقم اعشار دقت دارد و ذخیره سازی اعداد اعشاری در آن، به یک گرد سازی نهایی ختم خواهد شد. بنابراین به دلیل وجود این rounding error طبیعی، امکان استفاده از عملگری مانند == جهت مقایسه‌ی اعداد اعشاری همیشه پاسخ صحیحی را به همراه نخواهد داشت. برای نمونه مثال زیر را بررسی کنید:

```
double d1 = 12.14;
double d2 = 12.13;
double d3 = d1 - d2; // Should be 0.01
bool check = (d3 == 0.01); // should be true
```

که یک چنین خروجی را به همراه دارد (حاصل آن برخلاف تصور مساوی 0.01 نیست):

C# CompareTwoFloat CompareTwoFloat.Program

```
1 namespace CompareTwoFloat
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             double d1 = 12.14;
8             double d2 = 12.13;
9             double d3 = d1 - d2; // Should be 0.01
10            bool check = (d3 == 0.01); // should be true
11        }
12    }
13 }
```

100 %

Locals

Name	Value
args	{string[0]}
d1	12.14
d2	12.13
d3	0.00999999999999997868
check	false

روشی که برای حل این مساله پیشنهاد شده است، تقریق دو عدد از هم و مقایسه‌ی نتیجه‌ی آن با epsilon است؛ بجای مقایسه با صفر:

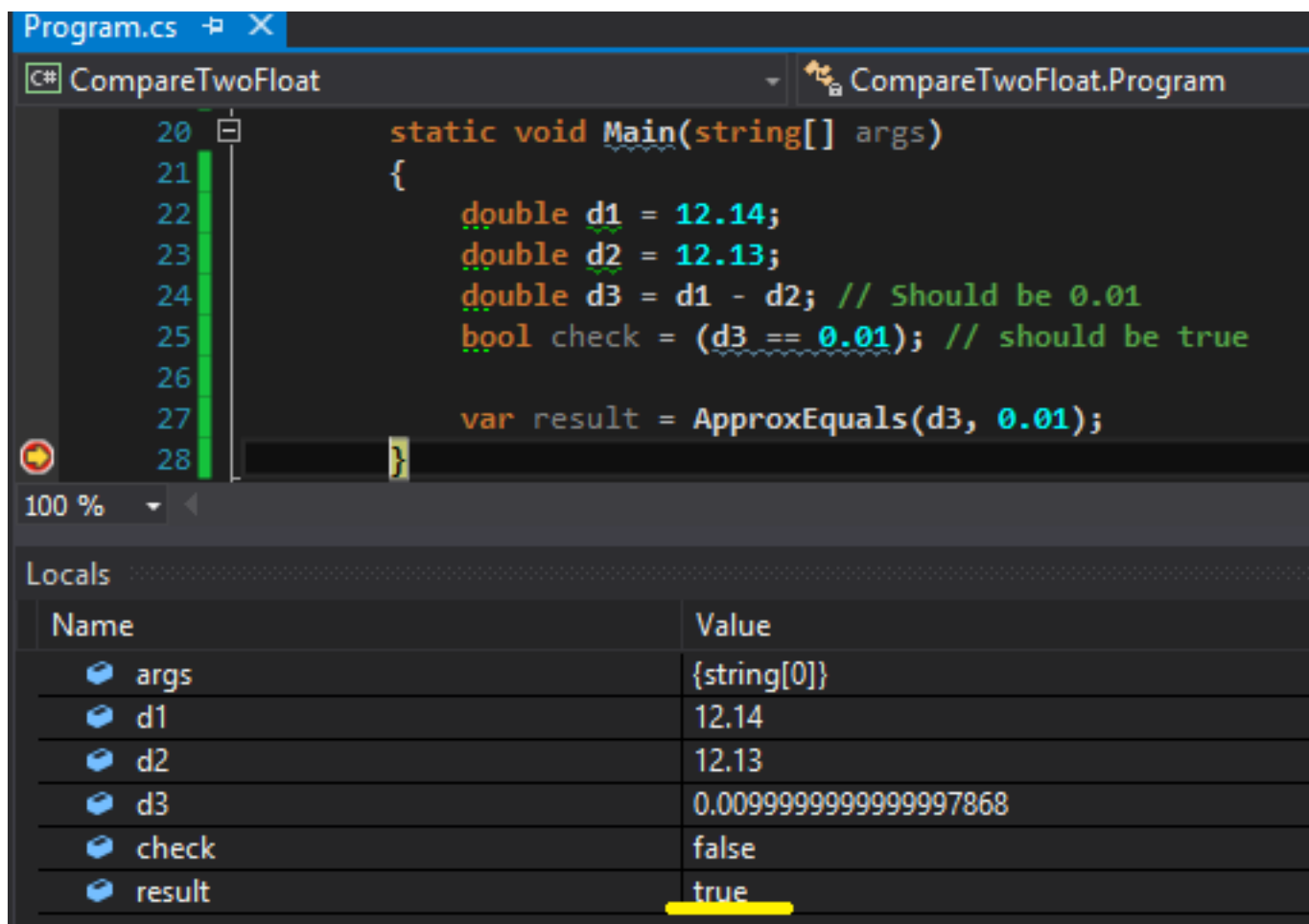
```
public static bool ApproxEquals(double d1, double d2)
{
    const double epsilon = 2.2204460492503131E-16;

    if (d1 == d2)
        return true;

    double tolerance = ((Math.Abs(d1) + Math.Abs(d2)) + 10.0) * epsilon;
    double difference = d1 - d2;

    return (-tolerance < difference && tolerance > difference);
}
```

متد فوق را در فایل [MathUtils](#) کتابخانه‌ی JSON.NET می‌تواند مشاهده کنید.
با این خروجی:



The screenshot shows a C# program named 'CompareTwoFloat' with the following code:

```
static void Main(string[] args)
{
    double d1 = 12.14;
    double d2 = 12.13;
    double d3 = d1 - d2; // Should be 0.01
    bool check = (d3 == 0.01); // should be true

    var result = ApproxEquals(d3, 0.01);
}
```

The 'Locals' window at the bottom displays the values of the variables:

Name	Value
args	{string[0]}
d1	12.14
d2	12.13
d3	0.00999999999999997868
check	false
result	true

در این حالت می‌توان نتیجه گرفت که d3 و 0.01 بسیار بسیار نزدیک به هم هستند؛ یا تقریباً مساوی.

نظرات خوانندگان

نویسنده: سوین
تاریخ: ۱۶:۴۷ ۱۳۹۳/۰۸/۰۳

با سلام
من خودم در یه نرم افزار به این مشکل برخوردی بودم و برای حل این مشکل ، قبل از اعمال جبری نوع های double اون ها رو تبدیل به decimal کرده و مقایسه یا تفریق می کنم .

نویسنده: عثمان رحیمی
تاریخ: ۱۹:۵۸ ۱۳۹۳/۰۸/۰۳

سلام .
آیا این قسمت

```
if (d1 == d2)  
    return true;
```

در تابع ApproxEquals اصلا اجرا می شود ، یعنی true برگردانده می شود ؟ اگر بله چه زمانی ؟
و چه فرقی با

```
(d3 == 0.01)
```

دارد مگه هر دو بررسی دو مقدار double نیستند ؟ پس وقتی (d3 == 0.01) نتیجه ی مورد نظر را ندهد if هم true را (تقریبا هیچ وقت) بر نمیگرداند.

نویسنده: وحید نصیری
تاریخ: ۲۱:۴۴ ۱۳۹۳/۰۸/۰۳

زمانیکه d1 و d2 حاصل هیچ نوع عملیات ریاضی خاصی نباشند. برای مثال اگر 0.33 را با 0.33 مقایسه کنید. اما مقایسه (double)0.33333 == (double)1/3 هرچند صحیح به نظر می رسد اما حاصل false است چون دقت اعشار دو طرف یکی نیست. سمت چپ حداکثر دقت را دارد و سمت راست یک عدد ثابت غیر محاسباتی است. همچنین در بسیاری از محاسبات، نتیجه ی نهایی در یک double [جای داده می شود](#) ؛ مانند d3 در تصاویر فوق. علت اینجا است که مطابق استاندارد IEEE 754، نوع double یک عدد **binary floating-point** است و علت اینکه d3 حاصل از محاسبات در اینجا دقیقا مساوی 0.01 نشده این است که تمام بیت های حاصل از عملیات ریاضی محاسبه ی آن در double ای که در کل 64 بیتی است، [جای نمی گیرد](#) و نتیجه ی نهایی، خیلی جزئی کمتر است از 0.01 (rounding error).

[اطلاعات بیشتر](#)

نویسنده: وحید نصیری
تاریخ: ۲۲:۰۰ ۱۳۹۳/۰۸/۰۳

- نوع double در دات نت 64 بیتی و نوع decimal دارای 128 بیت است. نوع double توسط CPU به صورت مستقیم پشتیبانی می شود اما نوع decimal خیر. به همین جهت کار کردن با double چندین برابر سریعتر است از decimal.
- نوع double به صورت باینری ذخیره می شود؛ اما نوع decimal دقیقا در مبنای 10. به همین جهت نوع decimal برای کارهای رومزه تجاری دارای اعشار، بسیار مناسب تر است.