



زنگ سی‌شارپ – قسمت هشتم

نوشته‌ی مسعود درویشیان  

[لینک مستقیم این مطلب در وب‌تارگت](#)

در [قسمت قبل](#) با if های تودرتو و if-else-if ladder آشنا شدیم. قصد داریم با دستور switch و حلقه‌ی for آشنا می‌شویم که یکی دیگر از مباحث بسیار مهم و پایه‌ای در برنامه‌نویسی هستند.

ولی قبل از این که سراغ حلقه‌ها برویم لازم است با یک سری از عمل‌گرهای ریاضی آشنا شویم.

عمل‌گرهای افزایشی و کاهشی (Increment و Decrement)

افزایش مقدار نگهداری شده در یک متغیر، یکی از امور رایج در برنامه‌نویسی است. فرض کنید شما یک متغیر به اسم counter تعریف کردید و می‌خواهید هربار که عملیاتی در برنامه شما انجام شد، به متغیر counter شما یک واحد اضافه شود. شما قصد دارید که خط‌کدی را شبیه خط‌کد زیر اجرا کنید:

```
counter = counter + 1;
```

این عبارت از نظر یک دانشجوی ریاضی، یک عبارت نادرست است اما [همان‌طور که در قسمت سوم بیان شد](#) در سی‌شارپ علامت تک‌مساوی (=) برای مقایسه‌ی دو مقدار استفاده نمی‌شود و در واقع، علامت تک‌مساوی، علامت انتساب است. خط‌کد بالا این‌طور عمل می‌کند که یک کپی از مقدار سمت راست مساوی را در قسمت سمت چپ قرار می‌دهد. فرض کنید مقدار counter عدد ۲ باشد، در این خط‌کد ۲+۱ می‌شود و حاصل آن در سمت چپ مساوی قرار می‌گیرد که در سمت چپ، خود متغیر counter قرار دارد و این بدین معنی است که: مقدار counter با عدد ۱ جمع شده و مجدداً در خودش ریخته می‌شود و در نهایت مقدار counter برابر با ۳ است.

از آن‌جا که افزایش مقدار یک متغیر یکی از کارهای رایج در برنامه‌نویسی است، سی‌شارپ چندین راه میان‌بر برای این منظور دارد. برای مثال، دو خط‌کد زیر دقیقاً یک معنی را می‌دهند:

```
counter += 1;  
counter = counter + 1;
```

عمل گر += به طور هم زمان عملیات "اضافه کردن و اختصاص دادن" را انجام می دهد. این عمل گر، عمل وند سمت راست را با عمل وند سمت چپ جمع می کند و نتیجه ی آن را به عمل وند سمت چپ اختصاص می دهد. علاوه بر عمل گر += عمل گرهای = - و *= و /= نیز وجود دارند که برای منظوره های دیگری استفاده می شوند.

- عمل گر -= از لحاظ طریقه ی عمل کرد مشابه += است با این تفاوت که به جای جمع، عمل تفریق را انجام می دهد. برای مثال در دستور `counter -= 1` مقدار عمل وند سمت راست از مقدار عمل وند سمت چپ کم می شود و حاصل آن در عمل وند سمت چپ ذخیره می شود.

- عمل گر *= به طریقی مشابه عمل "ضرب کردن و اختصاص دادن" را انجام می دهد.

- عمل گر /= نیز به همین روال عمل می کند. برای مثال `counter /= 2` معادل با `counter = counter / 2` است.

به خاطر داشته باشید که نباید بین دو علامت این عملگرها فاصله بگذارید. فاصله گذاشتن قبل و بعد از آنها اختیاری است.

هنگامی که می خواهید دقیقاً ۱ واحد به متغیر خود اضافه کنید می توانید از دو عمل گر پیش وندی و پس وندی برای افزایش مقدار متغیر استفاده کنید. برای استفاده از عمل گر افزایشی پیش وندی (prefix increment operator) کافی است از دو علامت + قبل از اسم متغیر استفاده کنید. به عنوان مثال، در نمونه ی زیر متغیر `someValue` در نهایت مقدار ۷ را در خود نگه می دارد:

```
using System;
class Example
{
    static void Main()
    {
        int someValue = 6;
        ++someValue;

        Console.WriteLine(someValue);
    }
}
```

متغیر `someValue` در ابتدا مقدار ۶ را در خود نگه می دارد ولی پس از این که عمل گر ++ روی آن اعمال شد، مقدار متغیر ۱ واحد افزایش پیدا می کند. برای استفاده از ++ پس وندی، از دو علامت پلاس (+) بعد از اسم متغیر استفاده می کنیم. به نمونه ی زیر دقت کنید:

```
using System;
```

```

class Example
{
    static void Main()
    {
        int anotherValue = 56;
        anotherValue++;

        Console.WriteLine(anotherValue);
    }
}

```

در این مثال مقدار متغیر **anotherValue** برابر با ۵۶ است و پس از اعمال عمل گر ++ پس‌وندی، مقدار آن برابر با ۵۷ می‌شود. ++ پیش‌وندی و پس‌وندی را می‌توانید به‌روی متغیرها اعمال کنید و اعمال آن‌ها به‌روی اعداد ثابت نادرست است. برای مثال، ++56 نادرست است زیرا عدد ۵۶ ثابت است و تغییر نمی‌کند ولی اگر عدد ۵۶ را در یک متغیر قرار دهید می‌توانید از این عمل گر برای افزایش مقدار آن استفاده کنید. برای نمونه، `int val = 56` و سپس می‌توانید بنویسید `val++` یا `++val` و مقدار متغیر را افزایش دهید.

هنگامی که از ++ پیش‌وندی و پس‌وندی برای افزایش مقدار یک متغیر استفاده می‌کنید، به‌ظاهری متوجه تفاوتی نمی‌شوید زیرا هر دوی آن‌ها یک واحد به متغیر اضافه می‌کنند ولی این دو عمل گر متفاوت عمل می‌کنند. هنگامی که از ++ پیش‌وندی استفاده می‌کنید، ابتدا تغییرات روی متغیر مربوطه اعمال می‌شود، نتایج محاسبه شده و ذخیره می‌شود سپس متغیر مورد استفاده قرار می‌گیرد.

به مثال زیر توجه کنید:

```

using System;
class Example
{
    static void Main()
    {
        int b, c;

        b = 4;
        c = ++b;
        Console.WriteLine("{0} {1}", b, c);
    }
}

```

هنگامی که برنامه‌ی بالا را اجرا می‌کنید می‌بینید که در خروجی "5 5" چاپ می‌شود. در این مثال، ابتدا مقدار ۴ به **b** اختصاص داده می‌شود سپس مقدار **b** یک واحد اضافه شده و برابر با مقدار ۵ می‌شود و در نهایت مقدار **b** به **c** اختصاص داده می‌شود. روند این پروسه به‌این دلیل بود که از ++ پیش‌وندی استفاده کردیم.

در مقابل، وقتی که از ++ پس‌وندی استفاده می‌کنید ابتدا متغیر مورد استفاده قرار می‌گیرد سپس نتایج محاسبه شده و ذخیره می‌گردد. برای مثال در نمونه‌ی زیر، ابتدا b شامل مقدار ۴ است سپس در خط بعد مقدار b به c اختصاص داده می‌شود و پس از این که مقدار b به c اختصاص داده شد، مقدار b یک واحد افزایش پیدا می‌کند و برابر با ۵ می‌شود:

```
b = 4;  
c = b++;  
Console.WriteLine("{0} {1}", b, c);
```

خروجی این برنامه برابر با "5 4" است. به عبارت دیگر، اگر $b = 4$ باشد آن‌گاه مقدار ++b هم‌چنان برابر با ۴ است و هنگامی که مقدار b به c اختصاص داده شد سپس مقدار b یک واحد افزایش پیدا می‌کند.

علاوه بر عمل‌گر پیش‌وندی و پس‌وندی افزایشی، می‌توانید از عمل‌گر پیش‌وندی و پس‌وندی کاهشی (--) نیز استفاده کنید. عمل‌گر ماینس ماینس (--) دقیقاً یک واحد را از متغیر کم می‌کند و طریقه‌ی عمل کرد آن به صورت پیش‌وندی و پس‌وندی، به شکلی مشابه با پلاس پلاس (++) است با این تفاوت که به جای عمل افزایش، عمل کاهش را انجام می‌دهد.

کلیه حقوق مادی و معنوی برای وب‌سایت [وب‌تارگت](#) محفوظ است.

استفاده از این مطلب در سایر وب‌سایت‌ها و نشریات چاپی تنها با ذکر و درج لینک منبع مجاز است.