زنگ سی‌شارپ - قسمت هفدهم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت قبل با مبحث بسیار مهم cast کردن آشنا شدید و دانستید که چگونه می‌توان نوع‌های مختلف را به یکدیگر تبدیل کرد. در این قسمت قصد داریم با عمل‌گرهای منطقی و بیتی و آرایه‌ها که یکی دیگر از مباحث مهم در برنامه‌نویسی هستند آشنا شویم.

عمل‌گرهای منطقی (Logical Operators)

در عمل‌گرهای منطقی ما با عملوندهایی از جنس بولین سروکار داریم که در نهایت نتیجه از جنس bool خواهد بود. عمل‌گرهای منطقی شامل AND و OR و XOR و NOT هستند که به‌ترتیب با علامت‌های & و | و ^ و ! مشخص می‌شوند.

در شکل زیر جدول درستی عمل‌گرهای منطقی را می‌بینید:

p	q	p & q	p | q	p ^ q	!p
False	False	False	False	False	True
True	False	False	True	True	False
False	True	False	True	True	True
True	True	True	True	False	False

بدیهی است که باید این جدول را به ذهن خود بسپارید. خروجی XOR زمانی برابر با true است که یکی و تنها یکی از عمل‌وندها true باشد.

به مثال زیر دقت کنید:

```
using System;
class Example
{
    static void Main()
    {
        bool p, q;

        p = true;
        q = false;

        if (p & q) Console.WriteLine("this won't execute");
        if (!(p & q)) Console.WriteLine("!(p & q) is true");
        if (p | q) Console.WriteLine("p | q is true");
```

```
        if (p ^ q) Console.WriteLine("p ^ q is true");
    }
}
```

با مقایسه‌ی برنامه‌ی بالا با جدول درستی، متوجه خواهید شد که چه اتفاقی در حال افتادن است. با اعمال عمل‌گرهای منطقی به‌روی عمل‌وندهای بولین، نتیجه‌ای از جنس بولین حاصل خواهد شد که موجب اجرا یا عدم اجرای دستور if می‌شود.

عمل‌گر Implication

یکی دیگر از عمل‌گرهای منطقی عمل‌گر implication است. این عمل‌گر نیز پاسخی از نوع بولین دارد. در این عمل‌گر تنها زمانی که عمل‌وند سمت چپ برابر با true و عمل‌وند سمت راست برابر با false است، پاسخ برابر با false می‌شود:

p	q	p implies q
True	True	True
True	False	False
False	False	True
False	True	True

نتیجه‌ای که از این عمل‌گر گرفته می‌شود این است: اگر مقدار p برابر با true است آنگاه مقدار q نیز باید true باشد. عملیات implication از طریق الگوی زیر انجام می‌شود:

```
!p | q
```

به مثال زیر توجه کنید:

```
using System;
class Example
{
    static void Main()
    {
        bool p, q;

        p = true;
        q = true;

        Console.WriteLine("p is " + p + ", q is " + q + "");
        Console.WriteLine("" + p + " implies " + q + " is " + (!p | q) + "");
        Console.WriteLine();

        p = true;
        q = false;

        Console.WriteLine("p is " + p + ", q is " + q + "");
        Console.WriteLine("" + p + " implies " + q + " is " + (!p | q) + "");
        Console.WriteLine();
```

```
        p = false;
        q = false;

        Console.WriteLine("p is " + p + ", q is " + q + "");
        Console.WriteLine("" + p + " implies " + q + " is " + (!p | q) + "");
        Console.WriteLine();

        p = false;
        q = true;

        Console.WriteLine("p is {0}, q is {1}", p, q);
        Console.WriteLine("{0} implies {1} is {2}", p, q, (!p | q));
        Console.WriteLine();
    }
}
```

در مثال بالا حالت‌های مختلف implication را مشاهده می‌کنید. همان‌طور که می‌بینید تنها زمانی که عمل‌وند سمت چپ برابر با true و عمل‌وند سمت راست برابر با false است، پاسخ برابر با false می‌شود.

عمل‌گرهای منطقی Short-Circuit یا اتصال کوتاه

سی‌شارپ ورژن مخصوصی برای AND و OR تدارک دیده که به Short-Circuit یا اتصال کوتاه معروف هستند. استفاده از Short-Circuit موجب می‌شود کد بهینه‌تر و پر سرعت‌تری داشته باشید. همان‌طور که می‌دانید در AND اگر عمل‌وند اول False باشد آن‌گاه جواب False است و در OR اگر عمل‌وند اول True باشد پاسخ True خواهد بود. کاری که Short-Circuit انجام می‌دهد این است که در این موارد، بقیه‌ی حالت‌ها را ارزیابی نکرده و مستقیماً جواب نهایی را می‌دهد در صورتی‌که در AND و OR معمولی همه‌ی حالت‌ها بررسی می‌شوند. علامت Short-Circuit AND به‌صورت && و علامت Short-Circuit OR به‌صورت || است که همچنین با عناوین Conditional AND و Conditional OR نیز شناخته می‌شوند.

به نمونه‌ی زیر توجه کنید:

```
using System;
class Example
{
    static void Main()
    {
        int i;
        bool someCondition = false;

        i = 0;

        // Here, i is still incremented even though the if statement fails.
        if (someCondition & (++i < 100))
            Console.WriteLine("this won't be displayed");
        Console.WriteLine("if statement executed: " + i); // displays 1
```

```
        // In this case, i is not incremented because the short-circuit
        // operator skips the increment.
        if (someCondition && (++i < 100))
            Console.WriteLine("this won't be displayed");
        Console.WriteLine("if statement executed: " + i); // still 1 !!
    }
}
```

در این برنامه متغیر i در ابتدا برابر صفر است. در دستور شرطی if به‌دلیل وجود عمل‌گر AND معمولی، هردو عمل‌وند آن ارزیابی می‌شوند بنابراین i یک واحد افزایش می‌یابد. در دستور if بعدی به‌دلیل وجود Short-Circuit AND عمل‌وند اول ارزیابی می‌شود که به‌دلیل False بودن، دیگر عمل‌وند دوم ارزیابی نشده و i افزایش پیدا نمی‌کند.

عمل‌گرهای بیتی (The Bitwise Operators)

عمل‌گرهای بیتی مستقیماً روی بیت‌های عمل‌وندشان کار می‌کنند و تنها برای مقادیر صحیح تعریف شده‌اند و نمی‌توانند برای نوع‌های bool، float و double به‌کار گرفته شوند. این عمل‌گرها برای اعمالی چون برنامه‌نویسی در سطوح پایین سیستم به‌کار می‌روند. عمل‌گرهای بیتی شامل AND و OR و XOR و NOT هستند که به‌ترتیب با نشانه‌های & و | و ^ و ~ نشان داده می‌شوند. تفاوت عمل‌گرهای بیتی با عمل‌گرهای منطقی در این است که عمل‌گرهای بیتی مستقیماً روی تک تک بیت‌های عمل‌وندشان کار می‌کنند. جدول زیر خروجی هر عمل‌گر را با استفاده از صفر و یک نشان می‌دهد:

p	q	p & q	p | q	p ^ q	~p
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

به نمونه‌ی زیر توجه کنید:

```
using System;
class Example
{
    static void Main()
    {
        int i = 25; // Bits of i: 00011001
        int j = 30; // Bits of j: 00011110

        int resutl = i & j; // (00011001) & (00011110) produce 00011000

        Console.WriteLine("First Number: {0}\nSecond Number: {1}", i, j);
        Console.WriteLine();
        Console.WriteLine("Result of AND Operation: " + resutl); // Result is 00011000 that means 24

        resutl = i | j; // (00011001) | (00011110) produce 00011111
        Console.WriteLine("Result of OR Operation: " + resutl); // Result is 00011111 that means 31
```

```
        resutl = i ^ j; // (00011001) | (00011110) produce 00000111
        Console.WriteLine("Result of XOR Operation: " + resutl); // Result is 00000111 that means 7

        resutl = ~i; // ~(00011001) produces 11100110
        Console.WriteLine("Result of NOT Operation: " + resutl); // Result is 11100110 that means -26
    }
}
```

معادل صفر و یک متغیرهای i و j در قسمت کامنت‌ها نوشته شده است. این صفر و یک‌ها طبق جدول با یکدیگر AND و OR و XOR و NOT می‌شوند و مقادیر جدیدی را تولید می‌کنند.

دو عمل‌گر بیتی دیگر با نام‌های Shift left و Shift right وجود دارند که به‌ترتیب با نمادهای >> و << نشان داده می‌شوند. توسط این دو عمل‌گر می‌توانید بیت‌ها به سمت چپ یا راست انتقال دهید.

فرم کلی این دو عمل‌گر به‌شکل زیر است:

```
value << num-bits
value >> num-bits
```

در این‌جا value عددی است که قصد داریم بیت‌های آن‌ها را به تعداد num-bits به چپ یا راست شیفت دهیم.

به نمونه‌ی زیر دقت کنید:

```
int result = 2 << 1; // 0010 becomes 0100 (Left shift)
Console.WriteLine(result);
```

در Shift right/Shift left بیت‌ها از مکان فعلی خودشان به‌اندازه یک بیت به چپ/راست انتقال می‌یابند. در مثال بالا بیت‌های عدد ۲ را به‌اندازه یک بیت به سمت چپ انتقال داده‌ایم که موجب شده است عدد ۲ به عدد ۴ تبدیل شود.

```
int result = 8 >> 1; // 1000 becomes 0100 (Right shift)
Console.WriteLine(result);
```

در این‌جا بیت‌های عدد ۸ را به‌اندازه یک بیت به سمت راست انتقال داده‌ایم که موجب شده است عدد ۸ به ۴ تبدیل شود. بنابراین نتیجه می‌گیریم که یه بیت شیفت به چپ مقدار را دو برابر و یک بیت شیفت به راست مقدار را نصف می‌کند. در پروژهایی با مقیاس بزرگ از این روش (به‌جای استفاده از ضرب و تقسیم معمولی) برای بالا بردن سرعت برنامه استفاده می‌کنند.

آرایه

آرایه مجموعه‌ای از متغیرها با نوعی مشخص است که همه‌گی با یک نام شناخته می‌شوند. در سی‌شارپ آرایه‌ها می‌توانند یک بعد یا بیشتر از یک بعد داشته باشند اما با این حال آرایه یک بعدی رایج‌ترین نوع آرایه است.

آرایه یک بعدی

آرایه یک بعدی لیستی از متغیرهای مرتبط به هم است. تصور کنید که می‌خواهید لیستی از نام کتاب‌های برنامه‌نویسی خودتان را داشته باشید. بنابراین شما باید آرایه‌ای از جنس string داشته باشید تا نام کتاب‌های خود را در آن قرار دهید. برای تعریف آرایه یک بعدی نیاز است که جنس ، اسم و اندازه آرایه را مشخص کنید.

آرایه یک بعدی از طریق الگوی زیر تعریف می‌شود:

```
type[] array-name = new type[size];
```

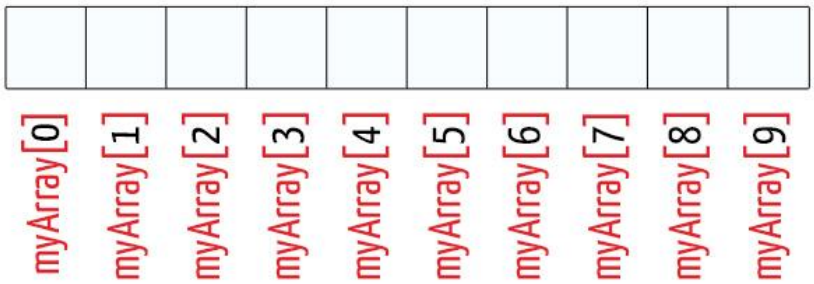
در این‌جا type مشخص کننده‌ی جنس آرایه و size مشخص کننده‌ی تعداد عناصری است که می‌توان در این آرایه قرار داد. array-name نیز نام آرایه را مشخص می‌کند.

به نمونه‌ی زیر توجه کنید:

```
int[] myArray = new int[10];
```

در این‌جا ما آرایه‌ای با نام myArray از جنس int تعریف کرده‌ایم که ۱۰ عنصر را می‌تواند در خود نگه‌داری کند. توجه داشته باشید که عنصرهای این آرایه همه‌گی باید از جنس int باشند.

آرایه‌ای که تعریف کردیم را در شکل زیر می‌توانید ببینید:



برای دسترسی به هر خانه‌ی آرایه نیاز است که شماره index آن خانه را صدا بزنید. در سی‌شارپ شماره index اول آرایه‌ها برابر با صفر است. در این‌جا که اندازه‌ی myArray برابر با ۱۰ است، index های آن از شماره ۰ تا ۹ هستند. بنابراین برای دسترسی به خانه‌ی اول آرایه باید اسم آرایه و در جلوی آن شماره index خانه‌ی اول آرایه که در براکت باز و بسته قرار دارد را بنویسید.

دسترسی به خانه‌ی اول:

```
myArray[0]
```

دسترسی به خانه‌ی آخر:

```
myArray[9]
```

برای مقداردهی به خانه‌های آرایه بدین صورت عمل می‌کنیم:

```
myArray[0] = 20;
myArray[1] = 13;
myArray[2] = 16;
.
.
.
```

به مثال زیر توجه کنید:

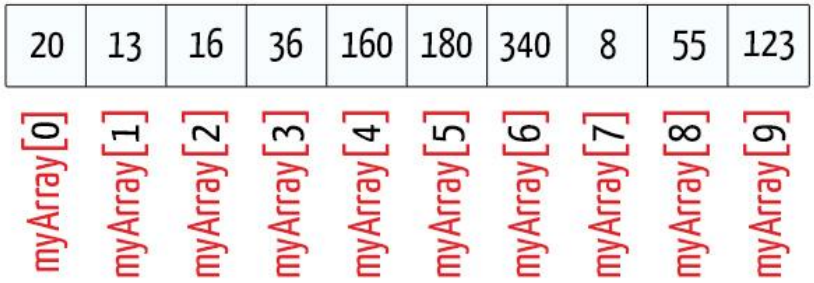
```
using System;
class Example
{
    static void Main()
    {
        int[] myArray = new int[10];

        myArray[0] = 20;
        myArray[1] = 13;
        myArray[2] = 16;
        myArray[3] = 36;
        myArray[4] = 160;
        myArray[5] = 180;
        myArray[6] = 340;
        myArray[7] = 8;
        myArray[8] = 55;
        myArray[9] = 123;

        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("myArray[" + i + "]: " + myArray[i]);
        }
    }
}
```

در این مثال ابتدا خانه‌های آرایه را با مقادیر دل‌خواه پر کرده‌ایم و سپس از طریق حلقه‌ی for یکی یکی مقدار هر خانه‌ی آرایه را نمایش داده‌ایم:

```
myArray[0]: 20
myArray[1]: 13
myArray[2]: 16
myArray[3]: 36
myArray[4]: 160
myArray[5]: 180
myArray[6]: 340
```

```
myArray[7]: 8
myArray[8]: 55
myArray[9]: 123
```

شکل آرایه بعد از پر شدن:



در قسمت بعد با مثال‌های بیشتری از آرایه آشنا خواهید شد.