C# > основни аритметични оператори и примерени кодове

```
        // 1. Деклариране на променливи
        int a = 10;
        int b = 3;
        double x = 10.0;
        double y = 3.0;
        // 2. Основни операции
        Console.WriteLine($"Събиране: {a} + {b} = {a + b}");
        Console.WriteLine($"Умножение: {a} * {b} = {a * b}");
        // 3. Специфики на делението
        Console.WriteLine($"Целочислено деление: {a} / {b} = {a / b}");
// Резултат: 3
        Console.WriteLine($"Реално деление: {x} / {y} = {x / y:F2}");
// Резултат: 3.33
        // 4. Остатък при деление (Modulo)
        Console.WriteLine($"Остатък от {a} / {b} е: {a % b}"); //
Полезно за проверка на четност
        // 5. Инкрементиране
        a++;
        Console.WriteLine($"Стойността на 'a' след a++ е: {a}");
```

Задачи за упражнение:

1. **Сума на три числа:** Напишете програма, която чете три цели числа от конзолата и извежда тяхната сума и средно аритметично
2. **Лице на правоъгълник:** Въведете две страни (double) и изчислете лицето (a * b).
3. **Четно или нечетно:** Използвайте оператора % 2, за да проверите дали въведено число е четно.

примерен код за комбинирани оператори **+=** и ***=**

Комбинираните оператори (Compound Assignment Operators) съкращават изписването на кода, като извършват аритметична операция и присвояване в една стъпка.

```
        int score = 10;
        Console.WriteLine($"Начален резултат: {score}");
        // Използване на += (еквивалентно на: score = score + 5)
        score += 5;
        Console.WriteLine($"След += 5 (добавяне): {score}");
        // Използване на *= (еквивалентно на: score = score * 2)
        score *= 2;
        Console.WriteLine($"След *= 2 (умножение): {score}");
        // Приложение с текст (Конкатенация)
        string message = "Здравей";
        message += ", Програмист!"; // message = message + ", Прог-ст!"
        Console.WriteLine(message);
```

1. **Чистия код:** Намалява излишното повтаряне на името на променливата.
2. **Оптимизация:** В някои езици и компилатори тези операции са малко по-бързи за обработка.

Примерен код: Приоритет и Присвояване

```
        // 1. Оператори за присвояване
        int x = 20;
        x += 5;  // x = 25
        x -= 10; // x = 15
        x *= 2;  // x = 30
```

```
        x /= 3;  // x = 10
        Console.WriteLine($"Крайна стойност на x: {x}");
        // 2. Приоритет на операциите (Precedence)
        // Правило: Умножение/Деление са преди Събиране/Изваждане
        int result1 = 5 + 2 * 10;
        Console.WriteLine($"Без скоби (5 + 2 * 10): {result1}"); //=25
        // Скобите имат най-висок приоритет
        int result2 = (5 + 2) * 10;
        Console.WriteLine($"Със скоби ((5 + 2) * 10): {result2}");//=70
        // 3. Комбиниран пример
        double complex = 10 + 20 / 2.0 * 3;
        // 1. 20 / 2.0 = 10.0
        // 2. 10.0 * 3 = 30.0
        // 3. 10 + 30.0 = 40.0
        Console.WriteLine($"Сложен израз (10 + 20 / 2 * 3): {complex}");
```

Кратка таблица на приоритетите
операторите се изпълняват в този ред:

1. **Скоби** () — променят естествения ред.
2. **Инкрементиране/Декрементиране** ++, --.
3. **Умножение, Деление и Остатък** *, /, %.
4. **Събиране и Изваждане** +, -.
5. **Присвояване** =, +=, *=, и т.н. (изпълняват се последни).

логически оператори (**&&** и **||**)

Примерен код: Вход и Парсване

```
// 1. Използване на int.Parse() - бърз и директен метод
        Console.Write("Въведете вашата възраст: ");
        string inputAge = Console.ReadLine();
        int age = int.Parse(inputAge);
// 2. Използване на Convert.ToInt32() - по-безопасен при null стойности
        Console.Write("Въведете любимо число: ");
        int luckyNumber = Convert.ToInt32(Console.ReadLine());
// 3. Аритметика с получените данни
        int sum = age + luckyNumber;
        Console.WriteLine($"\nСлед {luckyNumber} години ще бъдете на
{sum}!");
// 4. Преобразуване към double за дробни числа
        Console.Write("Въведете цена на продукт (напр. 5,50): ");
        double price = double.Parse(Console.ReadLine());
        Console.WriteLine($"Цената с 20% ДДС е: {price * 1.20:F2} лв.");
```

Основни разлики:

- **int.Parse():** Използвайте, когато сте сигурни, че входът е число. Ако потребителят остави празно поле, програмата ще даде грешка (Exception).
- **Convert.ToInt32():** Ако входът е null, този метод връща 0 вместо да прекъсва програмата.
- **double.Parse():** Важен е за финансови изчисления. Внимавайте с разделителя (точка или запетая) в зависимост от регионалните настройки на Windows.

Задача за упражнение:

Създайте **"Калкулатор за плочки"**: Потребителят въвежда дължина и ширина на стая (double), а програмата изчислява лицето и добавя 10% резерв за фира, като използва **+=**. Може да използвате **int.TryParse()**, за да не "гърми" програмата при грешен вход?

int.TryParse() е най-професионалният начин за четене на числа, защото предотвратява срив (crash) на програмата, ако потребителят въведе текст вместо цифри.

Примерен код с TryParse

```
        Console.Write("Въведете цяло число: ");
        string input = Console.ReadLine();
// TryParse връща true/false и извежда резултата в променливата 'number'
        bool isValid = int.TryParse(input, out int number);
        if (isValid)
        {
            Console.WriteLine($"Успешно преобразуване! Числото на
квадрат е: {number * number}");
        }
        else
        {
            Console.WriteLine("Грешка! Въвели сте невалиден формат
(текст или празен вход).");
        }
```

Защо това е важно?

1. **Защита от грешки:** За разлика от **int.Parse()**, който спира програмата при грешка, **int.TryParse** (Microsoft Learn) просто ни казва дали операцията е успешна.
2. **Параметърът out:** е ключовата дума **out** позволява на метода да "изнесе" готовата стойност директно в променливата number.
3. **Логическо мислене:** Това е преход към **условни конструкции (if-else)**.

Напишете програма, която изисква от потребителя да въведе **количество стока**.
Ако входът е валидно число, изпишете го.
Ако не е – изведете съобщение: *"Моля, използвайте само цифри!"*.

Тест по C#: Вход/Изход

1. Какъв ще бъде резултатът от следния код?

```
int result = 10 + 5 / 2;
Console.WriteLine(result);
```

- а) 7
- б) 12
- в) 12.5

2. Кой оператор се използва за намиране на остатък при деление?

- а) `/`
- б) `#`
- в) `%`

3. Ако `int x = 5;`, каква ще бъде стойността на `x` след изпълнение на `x *= 3 + 1;`?

- а) 16
- б) 20
- в) 19

(Подсказка: Сметнете първо израза след оператора за присвояване)

4. Кой от методите ще предизвика грешка (Exception), ако потребителят въведе празен ред в конзолата?

- а) `int.TryParse()`
- б) `int.Parse()`
- в) Нито един от двата

5. Какво прави ключовата дума `out` в метода `int.TryParse(input, out int number)`?

- а) Изтрива променливата от паметта.
- б) Показва резултата на екрана.
- в) Позволява на метода да запише преобразуваната стойност директно в променливата.

задача, за напреднали да комбинират всичко до момента: приоритет, остатък от деление и работа с Console.ReadLine().

Задача: „Машина за ресто“
Условие:
Напишете програма, която приема цяло число от конзолата – **сума в лева**. Програмата трябва да изчисли колко най-малко банкноти от **50**, **20** и **10** лева са необходими, за да се изплати тази сума.
Логически подсказки за учениците:

1. Първо пресметнете колко банкноти от 50 лв. се съдържат в сумата (чрез деление /).
2. Вземете остатъка (чрез %), който остава след "вземането" на 50-левките.
3. Повторете същата логика за 20 и 10 лв.

Примерен код (Решение):

```
        Console.Write("Въведете сума за теглене (лв.): ");
        if (int.TryParse(Console.ReadLine(), out int amount))
        {
            // Изчисляваме банкнотите от 50 лв.
            int bills50 = amount / 50;
            amount %= 50; // Остатъкът след 50-левките
            // Изчисляваме банкнотите от 20 лв.
            int bills20 = amount / 20;
            amount %= 20; // Остатъкът след 20-левките
            // Изчисляваме банкнотите от 10 лв.
            int bills10 = amount / 10;
            amount %= 10; // Остатъкът (дребни монети под 10 лв.)

            Console.WriteLine("\nВашето ресто ще бъде в:");
            Console.WriteLine($"- Банкноти по 50 лв.: {bills50}");
            Console.WriteLine($"- Банкноти по 20 лв.: {bills20}");
            Console.WriteLine($"- Банкноти по 10 лв.: {bills10}");
            Console.WriteLine($"- Остатък в монети: {amount} лв.");
        }
        else
        {
            Console.WriteLine("Невалидна сума!");
        }
```

Извод защо тази задача е полезна?

- **Реално приложение:** Показва как математическите оператори се използват в софтуер за банкомати или касови апарати.
- **Критично мислене:** Трябва да се разбере, че редът на пресмятане е от най-голямата към най-малката номинална стойност.

- **Упражняване на %:** Това е най-добрият начин да разберат оператора за остатък в практическа ситуация.

1. **б) 12** (Поради целочисленото деление: $5/2 = 2$. След това $10 + 2 = 12$).
2. **в) %** (Modulo).
3. **б) 20** (Операторът `*=` има по-нисък приоритет, така че първо се събира $3 + 1 = 4$, после $5 * 4 = 20$).
4. **б) int.Parse()** (Той изисква задължително валидно число).
5. **в)** Записва стойността в променливата.