Променливи и типове данни

Примитивни типове данни - символен низ (string), цели числа (int), реални числа (double), символи (char), логически (bool) .

Деклариране на променлива: <тип> <идентификатор>;

Пример: string **myStr**; double **sum**;

int **a, b, c;** char **sign**;

bool **flag**;

Правила за валиден идентификатор (име) на променлива:

1. **Може да съдържа:**
   - Букви (латински или Unicode букви)
   - Цифри (но **не може да започва с цифра**)
   - Подчертаване _

2. **Не може да съдържа:**
   - Специални символи като &, #, !, @ (освен ако не е в началото), -, + и др.
   - интервали

Инициализация (начална стойност на променлива) и присвояване на стойност (=).

sum=7.8; flag = true; myStr = “Hello, world”; sign = ‘A’; a=14;

Дробни и цели числа

Задача 1. Изведете решението на следния израз **(a + 7,2) / 3**, където стойността на променливата **a** е дробно число и се въвежда от конзолата. Резултатът да е закръглен до втория знак след десетичната запетая. Math.Round(a, <бр. знаци след запетаята>)

Пример: Вход: a = 2,5 Изход: 3,23
a = 1,2 2,8

Задача 2. Изведете решението на следния израз **a** $\sqrt{\mathbf{10}}$, където стойността на променливата **a** е дробно число и се въвежда от конзолата. Резултатът да се присвоява на отделна променлива, чиято стойност се извежда, като изходни данни. Math.Sqrt(a)

Пример: Вход: a = 4 Изход: 12.6491106406735
a = 3,6 11.3841995766062

Задача 3. Изведете решението на следния израз **(a - 4,2) / 7**, където стойността на променливата **a** е дробно число и се въвежда от конзолата. Резултатът да се присвоява на отделна променлива, закръглен до втория знак след десетичната запетая.

Math.Round(a, <бр. знаци след запетаята>)

Пример: Вход: a = 2,5 Изход: -0,24
a = 12,1 1,13

Задача 4. Да се въведе трицифрено цяло число от стандартен вход на конзолата, а като резултат програмата трябва да извежда последната цифра на въведеното число, повдигната на степен 5. Math.Pow(a, <степен>)

Пример: Вход: 132	Изход: 32 (2^5)
453	243 (3^5)

Задача 5. Да се въведе трицифрено число от конзолата, а като резултат програмата трябва да извежда частното от делението на това числото с първата му цифра.

Пример: Вход: 345 Изход: 115 (345/3)
612 102 (612/6)

Задача 6. Да се въведе двуцифрено цяло число от стандартен вход на конзолата, а като резултат програмата да извежда сбора на цифрите му повдигнати на квадрат.

Пример: Вход: 12 Изход: 9 ($1+2=3 - 3^2$)
45 81 ($4+5=9 - 9^2$)

Задача 7. **Условие:**

В ресторант трима приятели решават да си разделят сметката. Сметката включва:

- Цена за храна (реално число)
- Цена за напитки (реално число)
- Такса обслужване (процент, цяло число)

Програмата трябва да изчисли:

1. **Общата сума на сметката** след добавяне на таксата обслужване.
2. **Колко трябва да плати всеки приятел**, ако сметката се дели по равно.

Логически изрази, условен оператор, по-сложни проверки

1. **Логически (булеви) изрази -** изрази, които връщат **стойност true или false**.

Видове логически оператори:

Оператор	Значение	Пример
==	равно ли е	x == 5
!=	различно ли е	a != b
>	по-голямо	x > 0
<	по-малко	x < 10
>=	по-голямо или равно	x >= 5
<=	по-малко или равно	y <= 20

---

2. Логически оператори (съставни проверки) - използват се за комбиниране на условия:

Оператор	Име	Пример	Обяснение
&&	И (AND)	x > 0 && x < 10	вярно, ако и двете са верни
||	ИЛИ (OR)	x<0 || x>1	вярно, ако поне едно е вярно
!	НЕ (NOT)	!(x == 5)	обръща стойността

---

3. Условен оператор if.

Основен синтаксис:

```
if (условие)
{
   // код, ако условието е вярно
}
```

С else:

```
if (x > 0)
{
   Console.WriteLine("Положително число");
}
else
{
   Console.WriteLine("Нула или отрицателно");
}
```

С else if:

```
if (x > 0)
{
   Console.WriteLine("Положително");
}
else if (x < 0)
{
   Console.WriteLine("Отрицателно");
}
else
{
   Console.WriteLine("Нула");
}
```

По-сложни проверки: Можеш да комбинираш условия с логическите оператори:

```
if (x >= 10 && x <= 20)
{
   Console.WriteLine("x е между 10 и 20 включително");
}

if ((x < 0 || x > 100) && x % 2 == 0)
{
   Console.WriteLine("x е четно и извън интервала [0;100]");
}
```

Задача 8. Въведи цяло число и провери:

- Дали е положително
- Дали е четно
- Дали е в интервала [10; 100]

Задача 9. Проверка на трицифрено число
Да се въведе **цяло число** и да се провери дали:

- Числото е **трицифрено положително**
- **Последната му цифра е четна**

Ако и двете условия са изпълнени, програмата да изведе:

Числото е валидно.

В противен случай:

Невалидно число.

Насоки:

- Трицифрено положително число: x >= 100 && x <= 999
- Последна цифра: x % 10
- Четно: (x % 10) % 2 == 0

Задача 10. Валидно работно време.

Потребителят въвежда **час от денонощието** (цяло число от 0 до 23) и ден от седмицата (като текст, напр. "събота").

Да се провери дали:

- Часът е в **работно време** – от **10 до 18 ч** включително
- Денят е **делничен** – не е "събота" или "неделя"

Ако условията са изпълнени, изведи:

Отворено

Иначе:

Затворено

Насоки:

- Работно време: hour >= 10 && hour <= 18
- Делничен ден: day != "събота" && day != "неделя"

Циклични алгоритми и оператори

1. **Какво е цикъл?**

Цикъл е структура, която **повтаря дадено действие** многократно, докато е изпълнено определено условие. Използва се, когато трябва:

- Да повтаряме **еднотипни действия**
- Да **обхождаме диапазони** (напр. от 1 до 100)
- Да **сумираме, броим, намираме** нещо в последователност

---

2. **Видове цикли в C#**
    - **Цикъл с брояч (for) -** използва се, когато знаем **колко пъти** ще се повтаря действието.

Пример:

```
for (int i = 1; i <= 10; i++)
{
        Console.WriteLine(i);
}
```

Синтаксис:

```
for (начална стойност; условие; промяна)
{
        // тяло на цикъла
}
```

- **Цикъл while –** използва се, когато **не знаем предварително колко пъти** ще се изпълни цикълът.

Пример:
```
int i = 1;
while (i <= 10)
{
        Console.WriteLine(i);
         i++;
}
```

Синтаксис:
```
while (условие)
{
        // тяло на цикъла
}
```

❗ Ако условието не е вярно още отначало, **тялото може да не се изпълни нито веднъж**.

---

3. Разлики между for и while

Характеристика	**for**	**while**
Използване на брояч	Да – вградена в синтаксиса	По избор – създаваме го сами
Знаем броя повторения	Да	Обикновено – не
Прозрачност	По-прегледен за обхождане	По-гъвкав за непредсказуеми ситуации

Задача 11. Сумиране на числата от 1 до 100

Задача 12. Въвеждане на положителни числа, докато се въведе 0

Задача 11.
Да се отпечатат **всички числа от 1 до 100**, които са кратни на 7 и не са кратни на 5.