Университетско издателство „Паисий Хилендарски“

Ръководство по ПРОГРАМИРАНЕ(C#)

Николай Касъклиев
2016
Първо издание
ISBN 978-619-202-141-2

Ръководство по Програмиране(C#)
Първо издание, ISBN: 978-619-202-141-2
Университетско издателство „Паисий Хилендарски“

Пловдив, 2016 г.

Съдържание

Предговор

Настоящото ръководство цели да запознае читателите с основите на програмирането по един неформален и леснодостъпен начин с използване на много примери от практиката. То обхваща учебен материал, който се изучава в триместриален курс по Програмиране(C#) във Факултет по математика и информатика при Пловдивския университет „Паисий Хилендарски”. Представени са основните понятия в езиците за програмиране с примери на C#. В детайли се разглеждат основните типове данни, конструирането на изрази, управляващите конструкции, подпрограми и др., които формират основата на структурното програмиране.

Всяка глава съдържа теоретично въведение в изучаваната тематика и множество задачи, които илюстрират проблематиката по темата. За част от задачите се представя и подхода при решаването им, използваните структури от данни, алгоритъм и изходен код на решението на C#. Всяка глава съдържа и задачи за самостоятелна работа.

Ръководството е предназначено за студентите от бакалавърските програми на факултета - специалности Математика и Бизнес математика, но може да се използва и от всички студенти, които искат да се запознаят с основите на програмирането, чрез използване на един съвременен език за програмиране. Естественото място за използване на ръководството са семинарните упражнения, но успешно може да се използва и за самоподготовка.

Преподавателите от факултета могат да използват ръководството, като методическо помагало.

Ръководството е достъпно онлайн на адрес http://bell.uni-plovdiv.bg/.

Авторът, Николай Касъклиев, е преподавател по различни информатични дисциплини във Факултета по математика и информатика на Пловдивския университет „Паисий Хилендарски”. Основните му научни интереси са в областите: Програмиране, Операционни системи, Информационна сигурност и Разработка на мобилни приложения.

1. Увод

Компютрите са неизменна част от нашия живот. Използваме ги както за работа, така и за забавление и информационен източник в интернет ерата. Тяхното широко разпространение предполага и разполагането на тях(инсталиране) на съответните компютърни програми. Ако се замислим какво всъщност е компютъра ще установим, че той е съвкупност от два компонента – хардуер и софтуер. Друго кратко описание може да бъде следното: електронно устройство, което се използва за автоматизирана обработка на информация.

За да се случи тази обработка на електронното устройство трябва да се установи съответното управление. Да вземем следния пример. Искаме да използваме компютри, за да съхраняваме и обработваме различни данни за студенти в един факултет или университет. За целта ще бъде необходимо да изберем подходящи електронни устройства – компютри, но не само, и с подходящото им управление да реализираме съответните поставени задачи по обработката.

Управлението ще се осъществява с помощта на т.нар. компютърни програми, като може да имаме няколко различни програми предназначени за специфична обработка. В нашият случай, например - програма за съхраняване на данните, за тяхното извличане и редица програми за специфична обработка, като изчисляване на среден успех, подреждане на списъци, намиране на отлични или слаби оценки, формиране на специалности, курс и групи и стотици други. Другият вариант е всички тези програми да бъдат обединени в един общ пакет от програми, какъвто е например Microsoft Office.

Компютърът е машина (хардуер). Хардуерът, изграждащ компютъра се състои от централен процесор, дънна платка, видео карта, оперативна (RAM) памет, захранващ блок, твърд диск, компютърна кутия и входно-изходни устройства. За всички тези компоненти знаем достатъчно и не са предмет на разглеждане в настоящото ръководство.

За да извърши дадена задача компютърът трябва да бъде управляван по определен начин. Това управление става, чрез специално създадени от човек(написани на специален език) команди, които последствие се превеждат на разбираем за компютъра език. Именно това представлява компютърният софтуер. Софтуерът служи, за да накара компютъра да извърши конкретна задача, например да изпратим имейл, да преобразува данни от един вид в друг, да извършва сложни изчисления и хиляди други.

Какво по-конкретно е той?

- „Душата“ на машината.
- Създава се от програмистите по специални правила и с помощта на специфични средства.
- Има точно определена функционалност, с други думи позволява на потребителя да решава точно определени задачи.

1.1. Компютърна програма

Софтуерът по същество специфицира обработката на данните, за да може компютърът да изпълни определена обработка е необходимо да получи поредица от разбираеми команди плюс самите данни. За решаване на различните задачи се подават различни поредици от команди (наричат се още

инструкции) за решаване на определена задача от компютъра. Тази поредица се нарича ***програма***.

Процесът на създаване на програми се нарича ***програмиране*** и се осъществява от обучени специалисти наричани ***програмисти***. Множеството от команди (инструкции), с помощта на които се записват програмите формира т.н. ***език за програмиране.*** Такива са езиците C#, Java, C++, PHP, JavaScript, Visual Basic, Python и др.

Както всеки писмен език, така и езикът за програмиране се основава на дадена азбука. Азбуката е винаги крайно множество от символи.
Изреченията на даден език представляват последователности от символите на азбуката, конструирани по т.н. синтактични правила на езика (***синтаксис, syntax***).

Коректно написаните изречения на някой от езиците за програмиране се наричат програми на съответния език. Разбирането, което се влага в така конструираните правилни изречения, т.е. програмите и техните съставни части, се определя от т.н. семантични правила на езика (***семантика, semantics***). Взети заедно, азбуката, синтаксисът и семантиката формират езика за програмиране (ЕП) и както при естествените езици се описват и изучават с помощта на учебници, справочници и различни видове документация.

Когато за решаване на определена задача се използва компютър, моделът се описва с **алгоритми** и **данни**. При обработката на информацията се разграничават две основни страни на обработката: обект на обработката (данни) и процедура на обработката (алгоритъм). Според швейцарският професор Н. Вирт структурата от данни и алгоритъма са основните елементи на всяка компютърна програма[1]. Изборът на структурата от данни и съставянето на алгоритми за тяхната обработка ни дава възможността за създаване на програми с дадена функционалност.

ПРОГРАМА = ДАННИ + АЛГОРИТЪМ

Данни

Термините “информация” и “данни” са тясно свързани. Трябва да се има предвид, че данните са разновидност на информацията. Съществуват различни определения за понятието данни. Ето някои от тях: данните са систематизирана/извлечена информация във вид удобен за обработка от компютър; данните носят конкретна информация за конкретен обект; информацията, която е достъпна за въвеждане и обработка от компютъра е абстрактно отражение на реалния свят и се нарича данни.

Типът на данните е основно свойство на данните. Всеки тип определя допустимо множество от стойностни на съответните данни и операции, които могат да се извършат над тези данни. С типа на данните са свързани и начинът на представяне на данните в оперативната памет на компютъра. Когато две величини имат едно и също множество от допустими стойности, то това означава, че те са от един и същи тип. В езиците за програмиране стойностите на всяка константа, променлива и функция, принадлежат към точно определен тип. Типовете елементарни данни предварително се представят по определен вътрешен начин от компилатора, който превежда операторите в програмата от езика за програмиране на машинен език.

[1] Никлаус Вирт "Алгоритми + структури от данни = програми"

Основните категории типове са: числови (парични суми, брой….), символни, вкл. низове (имена на стоки, кодове на стоки …), логически (да/не, пр. Пол). Операциите върху тях изцяло зависят от типа, напр. аритметични за числовите, сравнение на низове за символните и т.н.

Преобразуването на типове е една от най-често прилаганите операции от програмиста. Защо е така ще демонстрираме с един прост пример. Както всички знаем, всеки студент притежава т.нар. Факултетен номер(ФН). Ако искаме да създадем компютърна програма, която съхранява такива данни в база данни (БД) е необходимо да помислим какъв тип данни да използваме. На пръв поглед е числов, но ако се замислим ще установим, че ФН няма да участва в операции прилагани върху числа, а с данните ще се използват операции, като въвеждане, извеждане, сортиране, сравнение и няколко други.

След направените анализи стигаме до извода, че подходящ тип на тези данни е стринг (символен низ). Така попадаме в ситуацията, ако искаме да създадем програма, която да е оптимизирана по отношение на представяне(бързина и малко заемана памет) е добре ФН да се съхранява в БД, като число, но когато данните трябва да се обработват числата да се преобразуват до стрингове за по-лесна обработка. Ето и един конкретен случай за необходима обработка. Всички знаем, че във ФН е закодирана информация за годината на постъпване и специалността на всеки студент. Несравнимо по-лесно е за програмиста да използва с един оператор метод за извличане на подниз от низ отколкото да извлича цифрите от числова стойност. Затова в такива ситуации много интензивно се прилагат методи за преобразуване на данните от един тип в друг.

Алгоритъм

Нека сега отделим внимание и на втория елемент от формулата. Алгоритъм наричаме последователност от операции прилагани върху данните решаващи поставената задача за обработка.

Други дефиниции са:

- Рецепта - формула;
- Метод, който при даден списък от коректно дефинирани команди за изпълнение на задача и зададено едно начално състояние преминава през точно дефинирана поредица от последователни състояния и завършва в едно крайно състояние по отношение на задачата;
- Поредица от елементарни действия описващи изпълнението на дадено изчисление.

Неформално понятието може да се дефинира като „процес, при който се извършва някаква крайна, предварително зададена поредица от операции“.

Класически пример за „алгоритъм“ е алгоритъмът на Евклид за намиране на най-големия общ делител (НОД) (чрез изваждане) на две цели числа, по-големи от 1. Ето поредицата от действия.

Вход: Числа a, b; **Изход:** НОД на a, b

Стъпка 1. Пригответе се за работа.

Стъпка 2. Въведете и запомнете числата a и b.

Стъпка 3. Ако $a \neq b$, изпълнете ст. 4, в противен случай – ст. 6.

Стъпка 4. Ако $a > b$, то изчислете $a - b$ и го запомнете като a,

в противен случай изчислете $b - a$ и го запомнете като b.

Стъпка 5. Изпълнете ст. 3.

Стъпка 6. Съобщете стойността на a (като резултат).

Стъпка 7. Прекратете работа.

Видовете стъпки според тяхното предназначение са:

- стъпки 2, 6 и 4 задават входните и изходните данни и междинните резултати на програмата;
- стъпки 3 и 4 описват условно изпълнение на едно от две различни действия;
- указанието от стъпка 5 изисква повторение на изпълнението на стъпки от 3 до 5;
- стъпки 1, 2 и 3 се изпълняват последователно по реда на тяхното срещане.

Както се вижда от примера за всяка компютърна програма можем да определим едно множество от основни алгоритми, наричат се още **базови**.

Базови алгоритми:

- Въвеждане и извеждане на данни
- Последователно изпълнение - Линейни
- Условно изпълнение - Разклонени
- Многократно изпълнение – Циклични

На всеки от базовите алгоритми е посветена отделна тема.

1.2. Езикът C#

C# е съвременен обектно-ориентиран език за програмиране от високо ниво с общо предназначение. Синтаксисът му е подобен на Java и C++. Езикът е част от .NET платформата на Microsoft и разработен в началото на 2002 г. NET е създадена за да осигури среда в която могат да се разработват всякакъв тип приложения за Windows. Последната версия наричана .NET Core 1.0 може да работи и на Linux и Mac компютърни системи. Базира се на концепцията за "виртуална машина" за изпълнение на програмата.

Програмите на C# представляват сорс код разпределен в един или няколко файла с тип .cs., в които се съдържат дефиниции на класове, структури, интерфейси и др. Тези файлове се компилират от компилатора на C# до изпълним код за изпълнение от CLR(Common Language Runtime) "виртуална машина" и в резултат се получават .exe или .dll файлове.

С използването на C# могат да се създават уеб страници, Windows Presentation Foundation (WPF) приложения, REST уеб услуги, компоненти за достъп до БД, класически десктоп приложения или дори Universal Windows Platform (UWP) приложения.

1.3. Структура на програма на C#

За запознаване със структурата на програма на този език нека разгледаме една кратка програма(конзолно приложение).

```
// Hello C# World! Програма на C#
using System;
namespace HelloCSharpWorld
{
    class Program
    {
        static void Main()
        {
             // извеждане на стринг на екрана
            Console.WriteLine("Hello C# World!");

            Console.WriteLine("Натиснете клавиш за изход.");
             // задържане на конзолния прозорец отворен
            Console.ReadKey();
        }
    }
}
```

В следващите редове накратко ще разясним програмата с нейните основни елементи.

Пространства от имена(namespace) и класове(class) се разглеждат по-късно, затова тук ще ги пропуснем.

Коментари

Първо нека дадем пояснение за първият ред от кода.

```
// Hello C# World! Програма на C#
```

Този ред съдържа т. нар. коментар и се указва с използването на символите //. Така останалата част от реда няма да се компилира и няма да предизвика някакво действие по време на изпълнение ма програмата. Коментарите се използват за поясняване на кода на програмите, когато се четат от разработчиците. Използват се още и когато разработчика иска временно да „изключи“ част от кода на програмата и да я тества без него.

В тези ситуации се използва и другия начин за коментар на цял блок(няколко реда) чрез символите `/* и */` съответно за начало и за край на блока.

```
/* Hello C# World! Програма на C#.
Това е моята първа програма и тя извежда на екрана  думите
 "Hello C#  World!".  */
```

Метод Main

Конзолните приложения на C# съдържат поне един метод – Main, който представлява входната точка за програмата.

```
static void Main()
      {
      // тук се декларират променливи, създават се обекти
      // и се извикват други методи
      }
```

В него се създават различни обекти и се извикват други методи. Повече за методите ще намерите в Глава 9.

Извеждане на екрана

Нашата първа програма при изпълнение предизвиква извеждане на екрана на два стринга(низ от символи), чрез използване на метода `Console.WriteLine();` За да можем да използваме този или други методи, които са предварително дефинирани за нас ние трябва да укажем това, чрез добавяне на `using` директива. В нашия случай метода `WriteLine` е дефиниран в класа `System`.

1.4. Новости в актуалната версия на C#

Както всеки език за програмиране така и този се развива и променя. Актуалната версия на езика е C# 6 и ще посочим накратко някои от новостите.

1) Използването на static в using декларациите позволява извикването на методите без да се посочва класа

В предходните версии

```
using System;
Console.WriteLine("Hello, World!");
```

В C# 6

```
using static System.Console;
WriteLine("Hello, World");
```

2) Автоматично имплементирани свойства, чрез инициализатор на свойства

В предходните версии

```
public class Student
{
public Student ()
{
Age = 20;
}
public int Age {get; set;}
}
```

В C# 6

```
public class Student
```

```
{
public int Age {get; set;} = 22;
}
```

3) Read-Only автоматични свойства – опростен синтаксис

В предходните версии

```
private readonly int teacherId;
public TeacherId
{
get
{
return teacherId;
}
}
```

В C# 6

```
public BookId {get;}
```

4) String заместване на използването на метода `string.Format`

В предходните версии

```
public override ToString()
    {
return string.Format("{0}, {1}", Name, Phone);
    }
```

В C# 6

```
public override ToString() => $"{ Name } { Phone }";
```

5) Dictionary инициализатори – опростен синтаксис

В предходните версии

```
var resDict = new Dictionary<int, string>();
resDict.Add(200,"OK");
resDict.Add(500,"ERROR");
```

В C# 6

```
var resDict = new Dictionary<int, string>()
{
[200] ="OK",
[500] =" ERROR"
};
```

1.5. Допълнителни източници на информация

Ръководството предоставя на читателя синтезирана и леснодостъпна информация за разглежданата тематика придружена от задачи за упражнение и техните решения. При съставянето му са използвани редица литературни източници и интернет ресурси (вж. Литература), които могат да се използват, като допълнителен източник.

В книгата на Светлин Наков и колектив ще намерите информация обхващаща основите на програмирането. В нея може да откриете подробно как да дефинирате и използвате променливи, как да работите с примитивни структури от данни, напр. числа, как да създавате изрази, условни конструкции и цикли, как да извеждате информация на конзолата, как да ползвате масиви, как да дефинирате и използвате методи, и да създавате и използвате обекти.

Книгата съдържа множество задачи и примерени програми, които са особено подходящи за студенти изучаващи дисциплината Програмиране.

Книгата C# 5.0 in a Nutshell е предназначена както за начинаещи програмисти така и за такива с повече опит. Тук ще може да се запознаете с езика C# и .NET Framework. Тематиката застъпена в ръководството е представена в глави две и три. Втора глава е посветена на синтаксиса на езика, на примитивните типове данни и символните низове, на масиви и изрази. В трета глава се разглеждат структури, изброим тип и методи.

В Programming in C#: A Primer, както и в другите книги се разглежда сходна тематика. Информацията по темите от това ръководство е разпределена в глави от 1 до 12. Така например Глава 6 е посветена на условно изпълнение, а Глава 9 на масиви. Книгата съдържа множество примерни програми.

На интернет страниците C# Programming Guide и C# Reference ще намерите абсолютно цялата информация синтаксис, семантика и примери касаеща езика – оператори, класове, методи, ключови думи и мн. др.

Deep C# е малка е-книга, където ще може да прочетете за типовете данни и променливите, за преобразуване на типове и за предаване на параметри на функции.

C# Yellow Book е книга специално написана за студенти от Department of Computer Science на University of Hull. От нея обърнете внимание на първа, втора, трета и четвърта глава.

Останалите книги имат сходно съдържание и също може да използвате като допълнителен източник на информация за Вашата подготовка.

2. Операции за вход и изход от конзолата

Както стана ясно от увода езикът позволява да се разработват много широк кръг приложения. За начинаещи програмисти е най-удачно да започнат пътя си като програмисти с програми, които комуникират – получават команди и показват резултати с опростен интерфейс(чрез команден ред). Интерфейсът с въвеждане на команди от командния ред се използва, когато широк набор от команди и голям брой настройки към тях могат да бъдат въведени по-бързо като текстови команди, отколкото, чрез графичен потребителски интерфейс. Типичен пример за това са командните за копиране на множество файлове от един и същи тип но намиращи се в различни директории(папки). Този тип въвеждане на данни, се използва и при системи, които нямат достатъчно ресурс за ползване на графичен интерфейс. Такива са различни видове контролери

2.1. Интерфейс с въвеждане на команди

Конзолата(shell), нарича се още команден ред, представлява специален прозорец достъпен в операционната система(ОС) или това е режим на работа с ОС, чрез който потребителя може да си комуникира със системните програми на самата ОС или с други програми. Комуникацията се изразява във въвеждане на текст и команди от стандартния вход (най-често това е клавиатурата) и/или извеждане на текст-резултати на стандартния изход (най-често е екрана на компютъра). Тези действия са известни още, като **операции за вход и изход**. Текстът, изписван на конзолата съдържа определена информация и представлява поредица от символи резултат от изпълнението на една компютърна програма.

За всяко конзолно приложение операционната система заделя входни изходни ресурси - устройства за вход и изход. По подразбиране това са клавиатурата и компютърния екран, но те могат да бъдат и файл или други устройства.

2.2. Класът System.Console

В C# за реализация на малки програми за конзолата се използва класът System.Console. Той има различни свойства и методи, които се използват за четене и извеждане на текст на конзолата, както и за неговото форматиране. Най-често за извеждане на текст на екрана се използват методите Console.Write() и Console.WriteLine(), а за четене от клавиатурата Console.ReadLine().

Извеждане на текст на конзолата

Работата с Console.Write() и Console.WriteLine() е лесна, тъй като, чрез тях могат да се извеждат всички примитивни типове и символни низове. Ето някои примери за извеждане на числови данни, символи и символни низове.

```
// извеждане на символ

Console.Write('K');

// извеждане на символен низ

Console.WriteLine("Аз обичам България.");
```

```
            // извеждане на цяло число

            Console.WriteLine(5);

            // извеждане на реалното число пи

            Console.WriteLine(3.14159265358979);
```

Ако стартирате програма с този код за извеждане ще забележите, че символът ‚К' е залепен за низа „Аз обичам България", което показва и каква е разликата между двата метода за извеждане. Докато Console.Write() извежда това което му е подадено като параметър без нищо допълнително, то Console.WriteLine() извежда ред и следващото извеждане ще започне на нов ред.

Форматиране на изхода

В практиката много често се налага да се извеждат низове форматирани по определен начин. Такива са например датите и валутните единици. В езика има лесни начини за реализиране на форматиран изход с използване на т. нар. спецификатори. Със следващите кратки примери ще демонстрираме как може да се постигне това.

```
    Console.WriteLine("Днес е {0:D}", DateTime.Now);
```

С използване на спецификатор D указваме датата да се изведе във формат 17 май 2016 г. Освен това обърнете внимание, че метода има два параметъра. Първият е низа за извеждане, а вторият е данни които да се изведат, като част от низа. За целта се използва форматиращ низ „**{индекс на параметъра}**".

Локализация

Нека разгледаме още една особеност. В примера по-горе, извеждането на екрана ще бъде в подходящ (искаме месецът да се изписва с дума и да е на български) за нас формат (17 май 2016 г.) само ако операционната система е настроена(регионално) за България. Ако не е така лесно може в програмата да се задава т. нар. локализация. Локализацията в .NET се нарича още "култура" и може да се задава чрез класа System.Globalization.CultureInfo. Ето пример.

```
Thread.CurrentThread.CurrentCulture = new CultureInfo("bg-
BG");
```

Задание: Проучете и останалите форматиращи низове.

Вход от конзолата

Въвеждането изисква наличието на код в програмата, който да осигурява запазване на въведеното по някакъв начин, за да може по-нататък то да се обработи. Тъй, като още не познаваме начините за реализация на такива операции тук само ще дадем само едно кратко изложение, с пример как да се запазват въведени текстове от конзолата.

За да позволим в нашата програма да се въвеждат данни от конзолата може да използваме методът Console.ReadLine(). При активирането му той става

текущия оператор за изпълнение от програмата и пред потребителя се показва маркер за вход. Потребителят въвежда някакъв низ в конзолата и натиска [Enter], така програмата прочита данните. Прочетените данни след това се записват за по-нататъшна обработка. Данните се записват в паметта и са достъпни чрез т.нар. променливи. Променливата е място за съхранение. Методът Console.ReadLine() връща като резултат въведения от потребителя символен низ, който трябва да се запише някъде. Как точно се записват различните по тип данни ще разгледаме в Глава 3. Тук ще дадем само един кратък пример за въвеждане на текст – име от клавиатурата и записването му в паметта като променлива с име name. Пред name виждате изписана думата `string`, което указва, че можем да запазим текст в нея.

За да се извърши записването използваме оператор ‚=‘ (присвояване).

```
Console.Write("Въведете име: ");
string name = Console.ReadLine();
```

Повече за променливите ще намерите в Глава 3. Където подробно ще изясним как да запазваме и обработваме данни от различен характер.

2.3. Задачи за упражнение

Задача 1. Да се напише програма която извежда на екрана думите „Hello Peter”.

За решаване на задачата е необходимо да използваме някой от методите за извеждане на конзолата.

За целта е необходимо да включим в кода на нашата програма референция към класа в който са дефинирани методите за извеждане. В нашият случай това е класа *System*.

В C# се използва директивата *using* за да укажем на компилатора да използва тип, който не е дефиниран в това пространство от имена(namespace).

След като сме извикали метода Write трябва да му предаден като параметър и какъв текст да изведе. Текстовете се дефинират като използваме кавички, например “текст”.

Следващата програма решава задачата.

```
using System;
namespace Console
{
    class Program
    {

     static void Main()
     {
     Console.Write("Hello Peter");
     }
   }
}
```

Задача 2. Да се напише програма, която извежда на екрана думите „ Hello Petar" и на следващ ред " How are you?"

В тази задача наред с извеждането на екрана се налага да направим и определено форматиране. Изисква се да се извеждата текстове на отделни редове.

За решаване ще използваме един от специалните символи за форматиране на извеждането "\n". При срещане на този сомвол в програмата следващото извеждане ще започне на нов ред.

Символи като този се наричат Escape sequence и представляват символни литерали, чрез които можем например да задаваме символи, които няма на клавиатурата и такива, които директно не могат да се задават в кода. Примери за символи, които не могат да се пишат директно в кода на програмата са: двойна кавичка, табулация, нов ред, наклонена черта и други. Точно такъв е нужен в нашата задача.

Други такива специални литерали са:

- \' – единична кавичка
- \" – двойна кавичка
- \\ – лява наклонена черта
- \n – нов ред
- \t – отместване (табулация)

Следващата програма решава задачата.

```
using System;
namespace Console
{
class Program
{

static void Main()
{
    Console.Write("Hello Petar");
    Console.Write("\n");
    Console.Write("How are you?");
    Console.Write("\n");
}
}
}
```

Второ решение с използване на метода Console.WriteLine()

```
using System;
namespace Console
{
```

```
    class Program
    {

    static void Main()
    {
    Console.WriteLine("Hello Petar");
    Console.WriteLine("How are you?");

    }
   }
}
```

Задача 3. Да се напише програма, която извежда на екрана думите „ Hello Petar" и на следващ ред днешна дата.

Вече добре познаваме начините за извеждане на екрана. За решаване на тази задача трябва да науча как в езика се представят данни различни от текст или число. В конкретният случай трябва да използваме данни от тип дата.

В езика в класа System е дефиниран и типа DateTime, който представя ден и час. Типът DateTime притежава и свойството Now, чиято стойност е текущия ден и час на компютъра на който се изпълнява програмата.

DateTime today = DateTime.Now - дефинира и инициализира променлива от този тип.

Следващата програма решава задачата.

```
using System;
namespace Console
{
    class Program
    {
        static DateTime thisDate = DateTime.Now;

     static void Main()
     {
    Console.WriteLine("Hello Petar");
    Console.WriteLine("Today is {0}",thisDate); // ще изведе
дата и час

   }
  }
}
```

Второ решение

```
using System;
namespace Console
```

```
{
    class Program
    {
        static DateTime thisDate = DateTime.Now;

     static void Main()
     {
    Console.WriteLine("Hello Petar");
    Console.WriteLine("Today is {0:d}",thisDate); // ще изведе
само дата

     }
   }
}
```

*Задача 4. Да се напише програма, която извежда на екрана думите „ Здравей Петър” и на следващ ред днешна **дата във формат приет в България**.*

```
using System;
using System.Globalization;
using System.Threading;
namespace Console
{
    class Program
    {
        static DateTime thisDate = DateTime.Now;
    static void Main()
     {
   Thread.CurrentThread.CurrentCulture = new CultureInfo("bg-
BG");
    Console.WriteLine("Здравей Петър");
   Console.WriteLine("Днес е {0:d}",thisDate);
    }
   }
}
```

Задача 5. Да се напише програма, която извежда на екрана „ Здравей Петър” и на следващ ред “Днес разполагаш с бюджет от 20лв.”.

```
using System;
using System.Globalization;
using System.Threading;
namespace Console
{
```

```
    class Program
    {

     static void Main()
    {
    Thread.CurrentThread.CurrentCulture = new CultureInfo("bg-
BG");
    Console.WriteLine("Здравей Петър");
    Console.WriteLine("Днес разполагаш с бюджет от
{0:c}",20);

    }
  }
}
```

Втори вариант

```
using System;
using System.Globalization;
using System.Threading;
namespace Console
{
    class Program
    {
        static DateTime thisDate = DateTime.Now;
       static void Main()
     {
    Thread.CurrentThread.CurrentCulture = new CultureInfo("bg-
BG");
    Console.WriteLine("Здравей Петър");

    Console.WriteLine("Днес {0:d} разполагаш с бюджет от
{1:c}",thisDate,20);

   }
  }
}
```

Задача 6. Да се напише програма, която прочита от клавиатурата въведен текст(символен) низ и след това го извежда на екрана.

Указание:

За решаване на здачата използвайте метод ReadLine() и дефинирайте

```
using System;
```

```
using System.Globalization;
using System.Threading;

namespace Console
{
    class Program
    {

       static void Main()
     {
    string input=null; // низ в който ще запазим въведеното от
клавиатурата
    Console.WriteLine("Здравейте, моля въведете низ от
клавиатурата и натиснете ENTER.");
    input = Console.ReadLine();
    Console.WriteLine("Вие въведохте:{0} ",input);

    }
   }
}
```

2.4. Задачи за самостоятелна работа

Задача 1. Напишете програма, която извежда на екрана цитати от любими ваши литературни произведения. Всеки цитат да се изведе на отделен ред.

Задача 2. Напишете програма, която извежда на екрана списък с имената на членовете на вашето семейство. Всяко име да се изведе на отделен ред и за всеки член изведене освен името му и в скоби възрастта.

Например: Митко Петров (50)

Иванка Петрова (42)

Указание: използвайне методите за форматиране(нов ред) и метода за извеждане `WriteLine()` *с два параметъра*

```
Console.WriteLine("Моето име е {0}", „Никола”);
```

Задача 3. Напишете програма, която реализира въвеждане от клавиатурата на имената на три ваши любими филма и ги извежда на екрана в обратен ред на въвеждането.

Указание: използвайте този код като пример

```
Console.Write("Въведете филм: ");
string name = Console.ReadLine();
```

3. Типове данни и променливи

Във втора глава бяха разгледани възможностите на езика за извеждане информация на екрана. Тази информация може да бъде, както информативни съобщения за потребителя как да управлява изпълняваната програма, така и данни получени след определена обработка.

За илюстрация нека разгледаме следния пример. На програмист е поставена задача да състави програма представляваща прост калкулатор. Програмата трябва да извежда съобщения на екрана за избор на операция (пр. събиране) и въвеждане от клавиатурата на две числа, да осигурява прочитане на въведените от потребителя числа, да изчисли сбора на числата и получения резултат да бъде изведен с подходящо съобщение на екрана.

За решаване на задачата, наред с вече разгледаните начини за извеждане на екрана, програмиста трябва да използва средствата на езика C# осигуряващи въвеждане на информация от различен характер (числа или символи) от клавиатурата, съхраняване и обработка на тази информация. За тези операции се използват променливи върху които да приложим определени операции в случая аритметични. Преди да разгледаме променливите нека дадем яснота за системата от типове в езика.

3.1. Типове данни в C#

Типовете данни представляват множества от стойности, които имат еднакви характеристики. Например типът int задава множеството от цели числа в диапазона от -2,147,483,648 до 2,147,483,647. Типовете данни се характеризират с: име – например int, размер (колко памет заемат) – например 4 байта и стойност по подразбиране (default value) – например 0.

Естествената система от типове на C# е сходна с тези на някои други езици за програмиране от високо ниво (пр. Java). C# е силно (строго) типизиран.

Всеки тип данни трябва да бъде деклариран, за да може да се използва в програмата те са две категории – предварително декларирани (стандартен, базов, примитивен) или декларирани от потребителя, по-специални синтактични правила.

Примитивните типове в езика са предварително декларирани, като са им съпоставени стандартни имена, като int, float или double.

Потребителски в езика е например типът enum.

Примитивни типове

Константите от примитивен тип могат да се задават директно по начина близък до общоприетия, например:

102 (целочислена)	-2.16 или 0.01E-3 (реални)
true и false (логически)	'a' 'z' '%' 'я' (символни)

Когато програмата трябва да обработва прости данни, между които има наредба те могат да се моделират с някой примитивен тип данни. Например парични суми, мерки, поредни номера, азбука и т.н.

Примитивните типове данни в C# се разделят на следните **видове**:

- Целочислени типове **– sbyte, byte, short, ushort, int, uint, long, ulong**;
- Реални типове с плаваща запетая – **float, double**;

- Реални типове с десетична точност – **decimal**;
- Булев тип – **bool**;
- Символен тип – **char**;
- Символен низ (стринг) – **string**;
- Обектен тип – **object**.

Тези типове данни се наричат още built-in types.

Таблица на примитивните типове в C#

Име	Заемана памет	Диапазон
byte	8-bit	0:255
sbyte	8-bit	-128:127
short	16-bit	-32,768:32,767
int	32-bit	-2,147,483,648:2,147,483,647
long	64-bit	-9,223,372,036,854,775,808: 9,223,372,036,854,775,807
ushort	16-bit	0:65,535
uint	32-bit	0:4,294,967,295
ulong	64-bit	0:18,446,744,073,709,551,615
bool	1-bit	true/false
float	32-bit	±1.5e−45 до ±3.4e38
double	64-bit	±5.0e−324 to ±1.7e308
decimal	128-bit	-7.9×10^{28} to 7.9×10^{28}

Приложение

Тип **char** се използва за представяне на отделни символи. char се представя в паметта с един байт. Вътрешното представяне на символите се извършва с цели числа (проучете Unicode таблицата на кодовете).

Тип **int** е използва за представяне на цели числа. В паметта int се представя с 4 байта.

C# поддържа и типовете **short** и **long**. Те се използват съответно за малки и големи цели числа. Действителният размер на тези типове е зависим от конкретна реализация.

Типовете sbyte, short, int и long може да са със или без знак.

Булевият тип се декларира с ключовата дума **bool**. Той може да приема две стойности – true и false. Стойността по подразбиране е false. Използва се най-често за съхраняване на резултата от изчисляването на логически изрази.

Тип **float, double** и тип **decimal** представят числа с плаваща запетая, съответно с единична и двойна точност. Типът float се представя с 4 байта, типът double с 8 байта а decimal 16 байта.

Стойностите по подразбиране за реалните типове са съответно 0.0f, 0.0d и 0.0м.

3.2. Променливи

Променливата е място в компютърната памет (RAM) именувано от програмиста, където може да съхраним частица информация. Тази информация на по-късен етап може да бъде променена или изтрита. Ако си представим паметта на компютъра като един голям шкаф с множество чекмеджета то променливата е едно чекмедже с табелка с име, в което може да съхраните информация. Отделните чекмеджета могат да бъдат с различен размер, съответно да съхраняват повече или по-малко информация.

Променливата е място за съхранение на информация, което може да се променя. Тя осигурява възможност за:

- запазване на информация;
- извличане на запазената информация;
- промяна на запазената информация.

За да използваме променлива в нашата програма трябва първо да я декларираме.

Декларация

Декларирането на променлива предизвиква заделянето на място в паметта. Декларацията на променлива може да включва освен елементите от декларацията на една променлива и указание каква памет да бъде заделена за тази променлива, както и присвояване на начална стойност на променливата (**инициализация**).

В C# ако една променлива не е инициализирана при декларацията, тя ще притежава стойност по подразбиране. Всеки тип данни в C# има стойност по подразбиране, например за числовите типове тя е 0.

Синтаксис:

<тип данни> <име на променлива> [= <инициализиращ израз>];

Примери:

```
int x = 10;
```

```
string lastName = "Dimitrov";
char symbol = 'a';
float len=3.5f;
decimal decimalPI = 3.14159265358979238m;
```

Идентификатор

Името на всяка променлива се нарича още идентификатор. Идентификаторът е последователност от букви, цифри или знак за подчертаване (underscore). Идентификаторът задължително започва с буква или знак за подчертаване. Големите и малките букви се различават. В езика не са поставени ограничения за дължината на идентификаторите, но такива могат да съществуват в някои конкретни реализации. Друга важна особеност е , че идентификатора не може да бъде ключова дума от езика (вж. Таблица 1.)

abstract	**as**	**base**	**bool**
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	in (generic modifier)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

Таблица 1. *Ключови думи.*

Препоръки при именуване

Ето някои препоръки за именуване на променливите:

- Имената трябва да са описателни и да подсказват за какво служи дадената променлива. Например, за име на човек подходящо име е personName
- Трябва да се използват само латински букви. Макар, че кирилицата е позволена от компилатора, не е добра практика тя да бъде използвана в имената на променливите

- В C# е прието променливите да започват с малка буква и да съдържат малки букви, като всяка следваща дума в тях започва с главна буква. Например препоръчително име е firstName, а не firstname
- Името на променливите трябва да не е нито много дълго, нито много късо – само трябва да е ясно за какво служи променливата в контекста, в който се използва
- Да се внимава за главни и малки букви, тъй като C# прави разлика между тях. Например асе и Асе са две различни променливи

Присвояване на стойност на променлива

Присвояването на стойност на променлива представлява задаване на стойност, която да бъде записана в нея. Тази операция се извършва чрез оператора за присвояване '='. От лявата страна на оператора стои име на променлива, а от дясната страна – новата й стойност.

Примери:

```
int age; // декларация на променливата
age = 40; // присвояване на стойност 40
string name; // декларация на променливата
 name = "Стоян Минков"; // присвояване на стойност
```

3.3. Константи

Константите представляват стойност, която не може да бъде променяна по време на изпълнение на програмата. Типичен пример за константа е числото π, чиято стойност е 3,14 не може да бъде променяна.

Декларация:

```
const <тип данни> <име> = стойност;
```

Декларацията на константа включва задаване на ключовата дума const типа на данните името на константата и задължително да се инициализира със стойност.

Примери:

```
public const double Pi = 3.14159;
public const int SpeedOfLight = 300000;
```

Използването на константи има следните **предимства**:

- повишава четливостта на програмата
- улеснява евентуални модификации на програмата
- предотвратява нежелана промяна на стойности

Следващата програма демонстрира използването на константа при изчисляване на лицето на окръжност.

```
using System;
```

```
namespace Constants
{
    class Program
    {
        static void Main(string[] args)
        {
             // декларация на константа
            const double pi = 3.14159;

            double r;
            Console.WriteLine("Въведете радиус: ");
            r = Convert.ToDouble(Console.ReadLine());
            double areaCircle = pi * r * r;
  Console.WriteLine("Радиус: {0}, Лице: {1}", r, areaCircle);
            Console.ReadLine();
        }
    }
}
```

3.4. Изрази

Под израз (expression) в ЕП се разбира правило за изчисляване на някаква стойност от точно определен тип. Изразът се състои от един или повече операнда, свързани помежду си с някои от допустимите операции в езика.

При срещане на израз в една програма се извършва изчисление на израза и се формира една единствена стойност.

При конструиране на изрази е необходимо да се съчетават правилно операциите и операндите.

Изразът в езика C# представлява:

Последователност от операнди и операции, която се използва за една от следните цели:

- изчисляване на стойност въз основа на операндите (константи, променливи, изрази, израз в кръгли скоби): -2.16, money, 2*a-3, 2/(a+3), i++
- обозначаване на функции (методи) или други обекти: Math.Sqrt(x), myFunc('s',2)

Операциите с един операнд са унарни, а с два операнда бинарни. Операндите на бинарните операции се означават като ляв и десен операнд. Изразите се изчисляват чрез последователно изпълнение на включените в тях операции, като при това се спазва техния приоритет (вж. таблица 4.).

Приоритетът на операциите може да се промени чрез **скоби** ( ), които отделят подизраз от целия израз. При пресмятане на сложен израз, първо се изчисляват всички подизрази. Всеки подизраз се замества с резултата от пресмятането и изчисленията продължават. Ако има вложени скоби, изчисленията започват от най-вътрешните.

Операции

Много от операциите са ни познати от математиката, например аритметичните, като тук остава да посочим само конкретния оператор в езика.

Други, като логическите ще използваме по-нататък в ръководството при решаване на конкретни задачи за условни оператори. Затова ще направим само един преглед, чрез изброяване.

В Таблица 2. са дадени са дадени аритметичните операции в C# заедно с тяхното предназначение и употреба. В Таблица 3. са поместени логическите и операциите за сравнение. В следващите глави ще се запознаем (с примерни задачи) с повечето операции и там ще бъдат разяснени в детайли.

Оператор	**Предназначение**	**Употреба**
*****	умножение	оп1*оп2
/	деление	оп1/оп2
%	остатък при деление	оп1%оп2
+	събиране	оп1+оп2
-	изваждане	оп1-оп2

Таблица 2. *Аритметични операции.*

Оператор	**Предназначение**	**Употреба**
!	логическо отрицание	! оп
<	по-малко	оп1<оп2
<=	по-малко или равно	оп1<=оп2
>	по-голямо	оп1>оп2
>=	по-голямо или равно	оп1>=оп2
==	равенство	оп1==оп2
!=	неравенство	оп1!=оп2
&&	логическо И	оп1&&оп2
||	логическо ИЛИ	оп1||оп2

Таблица 3. *Операции за сравнение и логически операции.*

Операция	**Предназначение**
.	избор на член
[]	индексиране
()	извикване на функция, метод
++,– –	добавяне/изваждане на 1
~	побитово отрицание
!	логическо отрицание
+,–	унарен плюс/минус
*,&	адресни операции
new	заемане на памет, създаване на инстанция
*,/,%	умножение, деление, остатък
+,–	събиране и изваждане

<<,>>	изместване наляво/надясно
<,<=,>,>=	операции за сравнение
==,!=	операции за сравнение
&	побитово И
^	изключващо ИЛИ
|	побитово ИЛИ
&&	логическо И
||	логическо ИЛИ
?:	аритметичен IF
+=,=,–=,*=,%=,/=,	присвояване комбинирано с други операции

Таблица 4. *Приоритет на операциите.*

На този етап от таблиците може да се добием представа за голяма част от операциите в езика, но накратко ще представим операция присвояване, която ще използваме от тук до края на разглежданата тематика.

Операция за присвояване

Най-често използваната операция в програмирането е присвояването на стойност. В езика операцията се реализира с оператор ‚=‘.

Същност:

Операторът за присвояване (assignment) изисква изчислената стойност на някакъв израз да се съхрани на определено място в паметта, обикновено указано чрез име на дадена променлива (т.е. да се присвои на променливата).

Приложение:

При задаване на правилата за определяне на стойностите на множеството на входните данни и междинните резултати, а понякога участва и в определянето на изходните резултати.

Синтаксис:

```
<л-израз> <операция за присвояване> <д-израз>
```

, където <операция за присвояване> може да бъде:

- прост (simple assignment) =
- съставен (compound assignment):
  - *= /= %= += –= с аритметична (arithmetic) операция
  - <<= >>= с операция за изместване (shift)
  - &= |= ^= с побитова (bitwise) операция

Стойността на израза отдясно (<д-израз>) на операцията за присвояване се изчислява а резултатът от изчислението се присвоява на левия операнд(<л-израз>). Типът на резултата е типа на левия операнд.

Ако отляво и отдясно на знака за присвояване се среща една и съща променлива, то при изчисляване на израза отдясно се използва старата стойност на променливата (дясна стойност), а след изпълнение на оператора, тя получава вече новата изчислена стойност (лява стойност)

(C#) операторът за присвояване връща като резултат самия <л-израз> (с новоприсвоената му от оператора за присвояване стойност). Ето защо, е възможно:

```
int xcoord,ycoord,zcoord;
xcoord = ycoord = zcoord = 1.0/dist;
```

Операторите за присвояване са дясно асоциативни. Поради това и поради горното:

```
int a=1,b=2,d=4;
a += b += d;//a=7,b=6,d=4 - дясно асоциативни
```

Внимание, ако типовете на операндите не са еднакви се прилагат ***правилата за преобразуване на типове***.

Преобразуването става по два начина:

- автоматично (Implicit Conversion) - пример 1 цялото число 32 ще се преобразува до реалното 32.0
- чрез оператор от програмиста (Explicit Conversion) - пример 2.

Примери:

(1)

```
double ctemp, ftemp;
ftemp = (ctemp * 1.8) + 32;
```

(2)

```
int a; float b,c;
c = float(a)+b;// преобразуване на a от цяло в реално число
```

Задачи върху променливи

Задача 1. Да се напише програма, в която се декларира целочислена променлива с име A, инициализира се A със стойност 2 и се извежда на екрана стойността на A.

За решаване на задачата трябва да си припомним, първо, как се декларира променлива и второ как се писвоява стойност. Синтаксисът на деклариране е следният : <тип на данните> <име на променливата>;

Присвояването на стойност има следния синтаксис: <име на променливата> = <стойност> ;

Тъй като по условие променливата се инициализира с числото, то за тип на данните трчбва да изберем някой от целочислените типове.

След това трябва да запишем и съответния код на C# тов е оператора: int A = 2;

Извеждането на екрана ни е познато от първа глава. Така че получаваме следния код решаващ задачата.

```
using System;
namespace Console
{
    class Program
    {

       static void Main()
     {
    int A = 2;
```

```
    Console.Write("A=");
    Console.Write(A);
    Console.Write("\n");

     }
  }
}
```

Задача 2. Да се напише програма, която намира лицето на кръг с радиус 2см.

За решението трябва да си припомним как се изчислява лицето на кръг. Формулата е: $S = \pi . r^2$

След като вече припомнихме работата с променливи за решаване на тази задача в допълнение, трябва да си припомним и конструирането на изрази. Първоначално дефинираме две променливи от реален и целочислен тип, в които да съхраним стойностите за числото π и радисуа r.

И накрая да съставим израза, който ще изчислява лицето на кръга - pi * r * r. Изчисляването става директно като впишем израза като втори аргумент на метода WriteLine.

```
using System;

namespace Console
{
    class Program
    {

       static void Main()
     {
    float pi = 3.14F;
    int r = 2;
Console.WriteLine("Лицето на окръжността S={0}", pi * r * r);
     }
   }
}
```

Задача 3. Да се напише програма, която намира периметъра и лицето на правоъгълник със страни 2,5 см и 5,8 см.

Първоначални ще декларираме четири реални променливи a, b, p и s. За целта при декларацията използваме тип на данните double.

За решението трябва да си припомним как се изчисляват периметъра и лицето на правоъгълник. Периметър е число, което е дължината на страните на фигурата. Стандартното означение за периметър в математиката е с буквата P.

Ако дължината на страните на правоъгълник са a и b. Дължината на всичките му страни е $P = a + b + a + b$ или: $P = 2a + 2b$.

Лицето на правоъгълник се изчислява по формулата: $S = a.b$

И накрая да съставим изразите, които ще изчисляват лицето и периметъра на правиъгълник. Те са : p = 2 * (a + b); и s = a * b;

Извеждането на екрана ни е познато от първа глава. Така че получаваме следния код решаващ задачата.

```
using System;

namespace Console
{
    class Program
    {

       static void Main()
        {
            double a = 2.5;
            double b = 5.8;
            double p;
            double s;
            p = 2 * (a + b); // промяна на приоритета
            s = a * b;
    Console.WriteLine("Лицето на правоъгълника е S={0}", s);
  Console.WriteLine("Периметъра на правоъгълника е P={0}", p);
       }
     }
 }
```

Задача 4. Да се напише програма, която намира обема на куб със страна A, дължината на която се въвежда от клавиатурата.

При решаването на тази задача ще използваме методиката от решенията на предишните задачи, като в допълнение остава само да припомним начина за прочитане на данни от клавиатурата, а именно използване на метода ReadLine().

За да преобразуваме прочетения символен низ в реално число ще използваме и метода ToDouble(стринг) от класа Convert.

```
using System;
namespace Console
{    class Program
    {
      static void Main()
        {
           Console.Write("Въведете дължина на страната a=");
            double a = Convert.ToDouble(Console.ReadLine());
            double v = a * a * a;

    Console.WriteLine("Обема на куба е V={0}", v);
```

```
      }
   }
 }
```

Задача 5. Да се напише програма, която въвежда радиуса на окръжност и намира и извежда дължината на окръжността и лицето на кръга с дадения радиус.

```
using System;

namespace Console
{
    class Program
    {
        private const double PI = 3.14159265;

       static void Main()
        {
            double r;
            Console.Write("Въведете r= ");
             r = Convert.ToDouble(Console.ReadLine());;
            double p = 2 * PI * r;
            double s = PI * r * r;
            Console.WriteLine("Лицето на кръга е S={0}", s);
    Console.WriteLine("Дължината на окръжността  е P={0}", p);

     }
   }
 }
```

3.5. Изброим тип

Изброимият тип enum е потребителски тип (дефиниран от програмиста), стойностите на който са именувани целочислени константи. Изброим тип се дефинира чрез ключовата дума enum, следвана от списък от идентификатори, заградени във фигурни скоби. По подразбиране първият идентификатор получава стойност 0. Всеки следващ идентификатор получава стойност с единица по-голяма от стойността на предходния.

Използването на типа позволява на програмистите да пишат програми, които са по-разбираеми. Тъй като стойностите са константи употребата им позволява да се осигури/предотврати промяната на данни, които не бива да бъдат променяни изобщо.

Синтаксис:

```
enum  <име> [  { <изброяващ списък> }]
```

Където <изброяващ списък> е:

```
<име> = <константен израз> незад , ...
```

Множеството от стойности:

Съвкупността от идентификаторите, изброени в скобите при деклариране на типа. На всяка стойност от даден изброен тип се съпоставя целочислена стойност:

- по подразбиране: 0, 1, 2, … по реда на изброяването им
- чрез явно задаване в изброяващия списък, като стойностите могат да се повтарят. Ако цялото число, съответно на някоя изброена стойност не е зададена, то е с 1 по-голямо от целочислената стойност на предишното име от списъка

Примери:

```
enum { OK, CANCEL }; // OK има стойност 0, CANCEL има стойност
1
enum Size { SMALL = 10, MEDIUM = 100, LARGE = 1000 };
int SZ=LARGE; // SZ има стойност 1000
```

Операции и релации:

Стойностите от изброен тип се считат еквивалентни на съответните им целочислени стойности. Поради това за тях са валидни всички операции валидни и за целочислените типове.

Стойностите от изброен тип са подредени според наредбата на съответните им целочислени стойности. Поради това за тях валидни всички операции за сравнение.

При извеждане (напр. с System.WriteLine) на стойности от изброен тип се извежда техният целочислен еквивалент.

Стойностите от изброен тип не могат пряко да се въвеждат (напр. с System.ReadLine).

Примерна програма

```
public class EnumExample
{
    enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Fri;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Fri = {0}", y);
```

```
        }
    }
    /* Изход:
       Sun = 0
       Fri = 5
    */
```

3.6. Клас с математически функции Math

Езикът C# разполага с богат набор от библиотеки с функции за обработка на данни от различен характер. Класът **Math** дефинира множество функции на най-често използваните математически операции.
Общият брой на функциите надвишава 40 тук ще дадем само няколко примера.

Метод ***Math.Sin()*** изчислява синуса на ъгъл даден в радиани

```
using System;

class Program
{
    static void Main()
    {
     double sin = Math.Sin(2.5);
     Console.WriteLine(sin);
    }
}
```

Метод ***Math.Pow ()*** реализира повдигане на число(първи аргумент) на степен(втори аргумент)

```
double value1 = Math.Pow(2, 2);
double value2 = Math.Pow(3, 4);
double value3 = Math.Pow(4, 2);
double value4 = Math.Pow(5, 3);
```

Метод ***Math.Sqrt()***реализира изчисление на квадратен корен

```
double a = Math.Sqrt(4);
double b = Math.Sqrt(2);
double c = Math.Sqrt(30);
double d = Math.Sqrt(44);
```

3.7. Задачи за упражнение

Решенията на следващите задачи са аналогични като подход затова се опитайте да дадете и свои решения освен дадените.

Приложените решения може да използвате като помощно средство.

Задача 1. Да се напише програма, която прочита трицифрено число въведено от клавиатурата и извежда на отделни редове цифрите на стотиците, десетиците и единиците.

```
using System;

namespace Console
{
    class Program
    {
        static void Main()
        {
            int a;
            Console.Write("Въведете трицифрено число:");
            a = Convert.ToInt32(Console.ReadLine());
            int b;
            int c;
            int d;
            b = a / 100;
            c = a / 10 % 10;
            d = a % 10;
            Console.WriteLine("Стотици: {0}",b);
            Console.WriteLine("Десетици:{0}", c);
            Console.Write("Единици:{0}",d);
        }
      }
   }
```

Задача 2. Да се напише програма, която прочита от клавиатурата две цели числа и извежда на екрана резултата от делението им.

```
using System;

namespace Console
{
    class Program
    {
```

```
        static void Main()
         {
             int a;
             Console.Write("Въведете число a:");
             a = Convert.ToInt32(Console.ReadLine());
             int b;
             Console.Write("Въведете число b:");
             b = Convert.ToInt32(Console.ReadLine());

             float c = (float)a / b;
             Console.WriteLine("Частно: {0}",c);
          }
         }
    }
```

Задача 3. Да се напише програма, която чете от клавиатурата един символ и извежда на екрана в десетичен формат UNICODE кода на символа.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
         {
             char a;
             Console.Write("Въведете символ:");
             a = Console.ReadKey().KeyChar;
             int b;
              b = (int)a ;
             Console.WriteLine("\n Код: {0}",b);
            }
     }
 }
```

Задача 4. Да се напише програма, която прочита от клавиатурата цена на стока начислява 20% отстъпка и извежда на екрана стойността на отстъпката и продажната цена.

```
using System;

namespace Console
{
    class Program
```

```
    {
       static void Main()
        {
            double regularPrice;
            double discount;
            double salePrice;
            Console.Write("Въвдете цена: ");
            regularPrice=Convert.ToDouble(Console.ReadLine());
            // изчислява  20% отстъпка
            discount = regularPrice * 0.2;

            // изчислява продажната цена
            salePrice = regularPrice - discount;

            Console.Write("Цена: ");
            Console.Write(regularPrice);
            Console.Write("\n");
            Console.Write("Отстъпка: ");
            Console.Write(discount);
            Console.Write("\n");
            Console.Write("Продажна цена: ");
            Console.Write(salePrice);
            Console.Write("\n");
           }
      }
    }
```

Задача 5. Програма демонстрираща изброим тип

```
using System;

class Program
{
    enum Importance
    {
     None,
     Trivial,
     Regular,
     Important,
     Critical
    };

    static void Main()
    {
```

```
     Importance value = Importance.Critical;


     if (value == Importance.Trivial)
     {
         Console.WriteLine("НЕ");
     }
     else if (value == Importance.Critical)
     {
         Console.WriteLine("ДА");
     }
    }
}
```

Задача 6. Да се напише програма, която изчислява $y = \frac{a^3+b^4}{\sqrt{a}}$

```
using System;
namespace Console
{
    class Program
    {
       static void Main()
        {
            double a;
            double b;
            Console.Write("Въведете a=");
            a = Convert.ToDouble(Console.ReadLine());
            Console.Write("Въведете b=");
            b = Convert.ToDouble(Console.ReadLine());
  double y = (Math.Pow(a, 3) + Math.Pow(b, 4)) / Math.Sqrt(a);
            Console.Write("y=");
            Console.Write(y);
            Console.Write("\n");

     }
    }
   }
```

3.8. Задачи за самостоятелна работа

Задача 1. Напишете програма, в която са декларирани променливи, като за всяка променлива изберете подходящ тип на данните. Всяка променлива да съдържат една от следните стойности: 255, -120, 2.34, 1 и 1245678987.

Задача 2. Напишете програма, в която са декларирани променливи, като за всяка променлива изберете подходящ тип на данните. Всяка променлива да съдържат една от следните стойности: „Аз програмирам“, true, ‚K’, 1, 1.1245678987567845 и - 658974568789. Изведете на екрана стойностите на всяка променлива на отделен ред.

Задача 3.Напишете програма, която изчислява и извежда на екрана лицето на триъгълник при въведени от клавиатурата дължините на страна и височина към нея.

Задача 4. Напишете програма, която изчислява и извежда на екрана обема на цилиндър при въведени от клавиатурата дължините на радиус на основата и височина на цилиндъра.

Задача 5. Напишете програма, която изчислява и извежда на екрана обема на сфера при въведен от клавиатурата радиус.

Задача 6. Да се напише програма, която прочита четирицифрено число въведено от клавиатурата и извежда на отделни редове цифрите на хилядните, стотиците, десетиците и единиците.

4. Оператори за избор

По подразбиране операторите в една прогама се изпълняват последователно. Всяка програма на C# (за конзолни приложения) започва изпълнението си от първия оператор на функцията main(), след това последователно се изпълняват всички оператори от тялото на тази функция. Изпълнението завършва с последния оператор от main(). С изключение на най-простите програми за извеждане на екрана последователното изпълнение на операторите не съответства на логиката на алгоритмите на задачите. Да разгледаме отново същинската част от кода на задачата за делението на две числа.

```
    ...
            int a;
            Console.Write("Въведете число a:");
            a = Convert.ToInt32(Console.ReadLine());
            int b;
            Console.Write("Въведете число b:");
            b = Convert.ToInt32(Console.ReadLine());

            float c = (float)a / b;
            Console.WriteLine("Частно: {0}",c);
    ...
```

Така реализираната така задача има един голям недостатък, а именно не е предвиден механизъм за проверка за коректност на входните данни. Как ще работи програмата, ако за стойност на променливата b потребителя въведе 0? Правилното решение е програмиста да реализира в кода на програмата проверка ***АКО*** е въведена некоректна стойност изпълнението да не продължи с изчисляване на стойността на променливата **c** а да се изведе съобщение на екрана например. Проверката за некоректна стойност изисква да се прави сравнение на различни типове данни. В програмирането това се осигурява с използването на различни оператори за сравняване.

Оператори за сравняване

В C# има няколко оператора за сравнение, които се използват за сравняване на двойки цели числа, числа с плаваща запетая, символи, низове и други типове данни. Припомнете си Таблица 3 от Глава 3.

Операторите за сравнение могат да сравняват изрази операндите в които са от произволен тип, например две числа, два числови израза, символи или две булеви стойности. Резултатът от сравнението е булева стойност, т.е. true или false.

Ето един пример:

```
using System;

namespace Console
{
    class Program
```

```
    {
        static void Main(string[] args)
        {

        Console.Write("Сравнение на различни типове данни: ");

            int length = 3;
            Console.WriteLine(length >= 2); // извежда True

            char gender = 'm';
            Console.WriteLine(gender <= 'f'); // извежда False

            double height = 1.70;
            Console.WriteLine(height != 1.60); // извежда True

            string name ="Мариан";
            string secondName = "Мариан";
            bool match = name == secondName;
            Console.WriteLine(match);  // извежда True
            Console.ReadLine();

        }
    }
}
```

В програмата се извършва сравнение между числа(цели и реални), между символи и между символни низове. При сравнението на числа те се сравняват по големина, а при сравнението на символи се сравнява тяхната подредба на Unicode кодовете на съответните символи. Сомволните низове се сравняват аналогично за всеки символ от низа.

4.1. Оператор if

Операторът се използва ако искаме по време на изпълнение на програмата да предизвикаме изпълнение на едни или други действия в зависимост от някакво условие.

Синтаксис:

```
if ( булев израз )
{
  тяло // прост или съставен оператор;
}
```

Семантика:

Изразът заграден в кръгли скоби представлява булев израз. Ако неговата изчислена стойност е true, условието се счита за вярно и операторът/ите в тялото се изпълняват.

Булевият израз може да бъде променлива от булев тип или булев логически израз. Булевият израз не може да бъде цяло число.

Тялото на оператора представлява една или повече операции. Когато операциите са повече от една те се поместват в оператор блок **{ … }.**

По тази причина трябва да сме особено внимателни с това колко наистина операции ще се извършват и да броим операторите. Една препоръка е да броите операторите , като използвате , **;** ‘ за индикатор. Както знаем в езика всеки оператор(всяка операция) завършва с , **;** ‘

Условният оператор може да бъде използван и в друга форма с клаузата else.

Синтаксис:

```
if(израз)
{
   // true
}
else
{
  // false
}
```

Така, ако стойността на израза е различна от нула (е true), условието се счита за вярно и операторът/ите след if се изпълняват. В противен случай се изпълнява операторът/ите след else.

Съществува вариант и условния оператор if да се използва като вложен на друг условен оператор if

Синтаксис:

```
if(израз)
{
   // true
}
else if(израз)

{
  // true
}
else

{
  // false
}
```

Приложението на оператора ще демонстрираме с един кратък пример.

Задача 1. Да се напише програма, която определя дали даден човек е тинейджър по въведена от клавиатурата възраст(години).

За да решим задачата е необходимо да декларираме една променлива, в която ще запишем числовата стойност за възрастта въведена от клавиатурата. Съгласно условието променливат е целочислена, в кода по-долу това е age.

След това ще преобразуваме въведения низ до цяло число и ще го присвоим на променливата age.

Остава да да направим сравнение на променливата и стойност 18(тинейджърска възраст) с използване на оператор if и да изведем подходящо съобщение.

Ето и кода на програмата, която решава задачата.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int age;
            Console.Write("Въведете вашата възраст: ");
            age=Convert.ToInt32(Console.ReadLine());
            if (age <= 18)
            {
                Console.Write("Вие сте тинейджър. ");
       }
   }
 }
```

4.2. Задачи за упражнения

Задача 2. Да се напише програма, която определя дали дадено число въведено от клавиатурата е четно или нечетно.

Решението на задачата се свежда до използване на условен оператор if и алгоритъм за определяне на това дали едно цяло число е четно или нечетно като използваме оператор , % '. Нарича се деление по модул.

Синтаксис:

<лява стойност> % <дясна стойност>

Семантика:

Операторът % изчислява остатъка от делението на първия операнд на втория.

Пример:

```
Console.WriteLine(5 % 2);        // ще изведе на екрана 1
Console.WriteLine(4 % 2);  // ще изведе на екрана 0
```

Както се вижда от примера остатъкът от делението на 5 и 2 е 1, а на 4 и 2 е 0. Така ако направим няколко експеримента установяваме, че остатъка при деление на 2 на четните числа е 0, а на нечетните е 1.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int a;
            Console.Write("Въведете число: ");
            a=Convert.ToInt32(Console.ReadLine());

            if (a % 2 == 0)
            {
                Console.WriteLine("Числото е четно!");

            }
            else
            {
                Console.Write("Числото е нечетно!");

            }
     }
   }
 }
```

При решаването на следващите задачи използваме вече утвърдения подход от предишната задача. Решението на задачата се свежда до използване на условен оператор if и подходящо сравнение на въведени от клавиатурата стойности на променливи.

Задача 3. Да се напише програма, която при число въведено от клавиатурата го увеличава с 3 ако е положително и увеличава с 2 ако е отрицателно.

```
using System;

namespace Console
{
```

```
    class Program
    {
       static void Main()
        {
            int a;
            Console.Write("Въведете число: ");
            a=Convert.ToInt32(Console.ReadLine());

            if (a > 0)
            {
                Console.WriteLine("Числото е положително!");

            a += 3;
          Console.WriteLine("Добавяме 3 . Новото а е {0}", a);

            }
            else
            {
           Console.WriteLine("Числото е отрицателно!");
            a += 2;
          Console.WriteLine("Добавяме 2. Новото а е {0}", a);
            }
         }
       }
    }
```

Задача 4. Да се напише програма, която прочита от клавиатурата резултат от тестове (в точки) и извежда на екрана оценката по шестобалната система по предварително утвърдена конвенция.

```
using System;

namespace Console
{
    class Program
    {

       static void Main()
        {
            int testScore;
            Console.Write("Въведете брой точки: ");
            testScore = Convert.ToInt32(Console.ReadLine());

            if (testScore >= 90)
            {
```

```
                Console.WriteLine("Оценката е Отличен!");

            }
            else if (testScore >= 80)
            {
                Console.WriteLine("Оценката е Много добър!");

            }
            else if (testScore >= 70)
            {
                Console.WriteLine(" Оценката е Добър. ");

            }
            else if (testScore >= 60)
            {
                Console.WriteLine("Оценката е Среден.");

            }
            else
            {
                Console.WriteLine("Оценката е Слаб.");

            }
           }
    }
  }
```

Задача 5. Да се напише програма, която определя дали даден човек има право да гласува, при въведени от клавиатурата възраст и гражданство.

```
using System;

namespace Console
{
    class Program
    {

       static void Main()
        {
            int age;
      char choice;
      bool citizen;
      Console.WriteLine("Въведете ваашата възраст: ");
       age=Convert.ToInt32(Console.ReadLine());
```

```
      Console.WriteLine("Български граждадин ли сте (Y/N): ");
            choice=Console.ReadKey().KeyChar;
            if (choice == 'Y')
            {
                citizen = true;

            }
            else
            {
                citizen = false;
            }
            if (age >= 18)
            {
                if (citizen == true)
                {
           Console.WriteLine("Вие имате право да гласувате.");
                }
                else
                {
        Console.WriteLine("Вие нямате право да гласувате. ");
                }
            }
            else
            {
       Console.WriteLine(" Вие нямате право да гласувате. ");
            }
     }
  }
}
```

Задача 6. Да се напише програма проверява дали година въведена от клавиатурата е високосна.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int a;

            Console.WriteLine("Въведете година: ");
```

```
            a=Convert.ToInt32(Console.ReadLine());
           if (a > 0)
        {
        Console.WriteLine("Числото е валидна година.");

       if (((a % 4 == 0) && (a % 100 != 0)) || (a % 400 == 0))
        {
             Console.WriteLine("Годината е високосна.");

        }
       else
       {
            Console.WriteLine("Годината  не е високосна.");
       }
       }
     else
     {
          Console.WriteLine("Невалидна година.");

     }
  }
 }
}
```

Второ решение:

Как то знаем в съвременните езици за програмиране голяма част от често използваните алгоритми върху данните са предварително реализирани за нас в конкретни библиотеки с функции. В C# в класа DateTime е наличен за използване от програмистите метода DateTime.IsLeapYear(), който връща булева стойност. Ако стойността е true предадената като аргумент година е високосна.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        static void Main()
        {
            int a;

            Console.WriteLine("Въведете година: ");
            a=Convert.ToInt32(Console.ReadLine());
```

```
            if (DateTime.IsLeapYear(a))
            {
                  Console.WriteLine("Годината е високосна.");

         }
        else
        {
            Console.WriteLine("Годината  не е високосна.");
        }
}
  }
 }
```

Задача 7. Да се напише програма, която намира най-голямото от три различни числа въведени от клавиатурата.

При решаването на тази задача ще използваме логическите оператори “И” и “ИЛИ” за да съставим един булев израз като правим няколко сравнения.

Логическо “И” на англ. "AND"

Нарича се още конюнкция или логическо произведение. Ако имаме два булеви израза и всеки може да бъде със стойност true или false. В C# се реализира с оператор **&&** и ще даде следните резултати:

БИ1	БИ2	БИ1 && БИ2
false	false	false
false	true	false
true	false	false
true	true	true

Логическо “ИЛИ” на англ. "OR"

Нарича се още дизюнкция или логическо събиране. В C# се реализира с оператор **||** и ще даде следните резултати:

БИ1	БИ2	БИ1 || БИ2
false	false	false
false	true	true
true	false	true
true	true	true

Логическо “НЕ“ на англ. "NOT"

Нарича се логическо отрицание или инвертор. В C# се реализира с оператор **!** и ще даде следните резултати:

БИ	**! БИ**
false	true
true	false

Ето и кода на програмата, която решава задачата.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int a,b,c;

            Console.WriteLine("Въведете A: ");
            a=Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Въведете B: ");
            b = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Въведете C: ");
            c = Convert.ToInt32(Console.ReadLine());


            if (a > b && a > c)
            {
                Console.Write("A е най-голямото число!");

            }
            else if (b > a && b > c)
            {
                Console.Write("B е най-голямото число!");

            }
            else
            {
                Console.Write("C е най-голямото число!");
```

```
            }
}
     }
    }
```

Задача 8. Да се напише програма, която при зададени дължини на трите страни от клавиатурата определя дали триъгълника е равностранен, равнобедрен или разностранен.

```
using System;

namespace Console
{
    class Program
    {

       static void Main()
        {
            int a,b,c;

            Console.WriteLine("Въведете A: ");
            a=Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Въведете B: ");
            b = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Въведете C: ");
            c = Convert.ToInt32(Console.ReadLine());

 bool x = a <= 0 || b <= 0 || c <= 0 || a + b <= c || a + c <=
b || b + c <= a;
            if (x)
            {
       Console.WriteLine("Не съществува такъв триъгълник!");

            }
            else if (a == b && b == c)
            {
           Console.WriteLine("Триъгълника е равностранен!");

            }
            else if (a == b || b == c || a == c)
            {
            Console.WriteLine("Триъгълника е равнобедрен!");
```

```
            }
            else
            {
             Console.WriteLine("Триъгълника е разностранен!");

            }

            }
        }
    }
```

Задача 9. Да се напише програма, която при въведено от клавиатурата цяло число в интервала от 1 до 9999 извежда на екрана числото с думи.

Указание: използвайте оператор остатък от целочислено деление %

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {
            int enterNumber = 0;

  Console.WriteLine( "Въведете цяло число  (от 1 до 9999): ");
            enterNumber = Convert.ToInt32(Console.ReadLine());

            if (enterNumber <= 0 || enterNumber > 9999)
Console.WriteLine( "Числото не е в диапазона от 1 - 9999!\n");
            else
            {
                Console.WriteLine( "\nВие въведохте: ");

                if ((enterNumber / 1000) % 10 == 1)
Console.WriteLine( "Хиляда ");
                else if ((enterNumber / 1000) % 10 == 2)
Console.WriteLine( "Две хиляди ");
                else if ((enterNumber / 1000) % 10 == 3)
Console.WriteLine( "Три хиляди ");
                else if ((enterNumber / 1000) % 10 == 4)
Console.WriteLine( "Четири хиляди ");
                else if ((enterNumber / 1000) % 10 == 5)
Console.WriteLine( "Пет хиляди ");
```

```
                else if ((enterNumber / 1000) % 10 == 6)
Console.WriteLine( "Шест хиляди ");
                else if ((enterNumber / 1000) % 10 == 7)
Console.WriteLine( "Седем хиляди ");
                else if ((enterNumber / 1000) % 10 == 8)
Console.WriteLine( "Осем хиляди ");
                else if ((enterNumber / 1000) % 10 == 9)
Console.WriteLine( "Девет хиляди ");


                if ((enterNumber / 100) % 10 == 1)
Console.WriteLine( "сто ");
                else if ((enterNumber / 100) % 10 == 2)
Console.WriteLine( "двеста ");
                else if ((enterNumber / 100) % 10 == 3)
Console.WriteLine( "триста ");
                else if ((enterNumber / 100) % 10 == 4)
Console.WriteLine( "четиристотин ");
                else if ((enterNumber / 100) % 10 == 5)
Console.WriteLine( "петстотин ");
                else if ((enterNumber / 100) % 10 == 6)
Console.WriteLine( "шестстотин ");
                else if ((enterNumber / 100) % 10 == 7)
Console.WriteLine( "седемстотин ");
                else if ((enterNumber / 100) % 10 == 8)
Console.WriteLine( "освемстотин ");
                else if ((enterNumber / 100) % 10 == 9)
Console.WriteLine( "деветстотин ");

                if ((enterNumber / 10) % 10 == 1)
                {
                    if (enterNumber % 10 == 0)
Console.WriteLine( "десет ");
                    else if (enterNumber % 10 == 1)
Console.WriteLine( "единадесет ");
                    else if (enterNumber % 10 == 2)
Console.WriteLine( "дванадестет ");
                    else if (enterNumber % 10 == 3)
Console.WriteLine("тринадесет ");
                    else if (enterNumber % 10 == 4)
Console.WriteLine("четиринадесет ");
                    else if (enterNumber % 10 == 5)
Console.WriteLine("петнадесет ");
```

```
                    else if (enterNumber % 10 == 6)
Console.WriteLine("шестнадесет ");
                    else if (enterNumber % 10 == 7)
Console.WriteLine("седемнадесет ");
                    else if (enterNumber % 10 == 8)
Console.WriteLine("осемнадесет ");
                    else if (enterNumber % 10 == 9)
Console.WriteLine("деветнадесет ");
                }

                if ((enterNumber / 10) % 10 == 2)
Console.WriteLine( "двадесет ");
                else if ((enterNumber / 10) % 10 == 3)
Console.WriteLine( "тридесет ");
                else if ((enterNumber / 10) % 10 == 4)
Console.WriteLine("четиридесет ");
                else if ((enterNumber / 10) % 10 == 5)
Console.WriteLine("петдесет ");
                else if ((enterNumber / 10) % 10 == 6)
Console.WriteLine("шестдесет ");
                else if ((enterNumber / 10) % 10 == 7)
Console.WriteLine("седемдесет ");
                else if ((enterNumber / 10) % 10 == 8)
Console.WriteLine("осемдесет ");
                else if ((enterNumber / 10) % 10 == 9)
Console.WriteLine("деветдесет ");

                if ((enterNumber / 10) % 10 != 1)
                {
                    if (enterNumber % 10 == 0)
Console.WriteLine( "");
                    else if (enterNumber % 10 == 1)
Console.WriteLine( "и едно");
                    else if (enterNumber % 10 == 2)
Console.WriteLine( "и две");
                    else if (enterNumber % 10 == 3)
Console.WriteLine( "и три ");
                    else if (enterNumber % 10 == 4)
Console.WriteLine( "и четири");
                    else if (enterNumber % 10 == 5)
Console.WriteLine( "и пет  ");
                    else if (enterNumber % 10 == 6)
Console.WriteLine( "и шест  ");
```

```
                    else if (enterNumber % 10 == 7)
Console.WriteLine( "и седем  ");
                    else if (enterNumber % 10 == 8)
Console.WriteLine( "и осем  ");
                    else Console.WriteLine( "и девет  ");
                }
            }

        }
    }
}
```

4.3. Оператор за избор на варианти switch.

Вече разгледахме възможността последователността при изпълнението на програмата да бъде променяно в зависимост от проверката на определени условия. За целта използвахме няколко вложени оператора if. Често този подход води до трудно предвидими резултати, а програмистите по-трудно откриват логическите си грешки.

В C# съществува алтернативен начин за избор от множество възможности, чрез оператор switch.

Синтаксис:

```
switch (израз)
{
case константа:
// оператори[break]
case константа:
// оператори[break]
…
default:
// оператори

}
```

Семантика:

Стойността след ключовата дума case нарича се етикет case label. След него задължително следва ":". Всеки етикет case е израз, чиято стойност е цяло число (или символна константа). Два етикета не бива да имат еднаква стойност.

При изпълнение на switch се изчислява изразът в скобите и получената стойност се сравнява последователно с етикетите case. При съвпадение се изпълняват операторите след съответния етикет. В противен случай не следват никакви действия от switch.

Обикновено се смята, че се изпълняват само операторите за съответния етикет case. В действителност се изпълняват всички оператори между първия оператор за съответния случай и последния оператор в switch.

default клаузата не е задължителна и операторите след нея ще се изпълнят само ако няма съвпадение с нито един етикет.

Операторът дава възможност на програмистът да „инструктира“ компилатора, че иска да се изпълнят само операторите за съответния етикет. За целта се използва оператора за безусловен преход break.Обикновено break е последният оператор за съответния етикет case, break предизвиква излизане от switch, а изпълнението продължава с първия оператор след switch.

Пример:

```
int value = 1;
switch (value)
{
    case 1:
Console.WriteLine("Вариант едно"); // ще се изпълни този
        break;
    case 2:
        Console.WriteLine("Вариант две");
        break;
    default:
        Console.WriteLine("По подразбиране");
        break;
}
```

Възможно е обаче и в някои ситуации да се налага да извършваме една операция при наколко стойности на етикета(case). Това може да укажем така:

```
int i = int.Parse(System.Console.ReadLine());
            switch (i)
            {
                case 1:
                case 5:
                case 10:
                   {
                     System.Console.WriteLine("Малко число.");
                      break;
                   }
                case 30:
                case 40:
                case 50:
                   {
            System.Console.WriteLine("Средно голямо число.");
                       break;
                   }
                default:
                   {
                       System.Console.WriteLine("Друго.");
                       break;
                   }
            }
```

4.4. Задачи за упражнения

Задача 1. Да се напише програма, която прочита от клавиатурата номер на пръст и звежда на екрана наименованието му.

При решаването на задачата трябва първо да декларираме целочислена променлива, която ще използваме за въвеждане на номера на пръста.

Следващата стъпка е проверка за коректност на въведената стойност. В нашият случай стойността трябва да бъде в интервата от 1 до 5. За целта може да използваме условен оператор.

И чрез включване на оператора switch да опишем оператори за всяка стойност [1, 5] на променливата.

Ето и кода на програмата, която решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {
            int Finger = 0;

            Console.WriteLine("Въведете номер на пръста: ");
            Finger = Convert.ToInt32(Console.ReadLine());


            if (Finger <= 0 || Finger > 5)
   Console.WriteLine("Числото не е в диапазона от 1 до 5!\n");
            else
            {
                switch (Finger)
                {
                    case 1:
                        Console.WriteLine("Палец");
                        break;
                    case 2:
                        Console.WriteLine("Показалец");
                        break;
                    case 3:
                        Console.WriteLine("Среден");
                        break;
                    case 4:
                        Console.WriteLine("Безимен");
                        break;
```

```
                    case 5:
                        Console.WriteLine("Кутре");
                        break;
                    default:
                    Console.WriteLine("Грешка");break;
                }

            }

        }
    }
}
```

Задача 2. Да се напише програма, която реализира прост калкулатор (,/,+,-) за цели числа.*

При решаването на задачата ще използваме четири променливи. Две целочислени за операндите, една реално число за резултат и една – символ за вида на операцията. С изпозването на switch ще опишем операциите съответно събиране, изваждане, умножение и деление в зависимост от стойността на променливата от символен тип.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int num1;

            int num2;

            string operand;

            float answer;

            Console.Write("Моля въведете цяло число: ");

            num1 = Convert.ToInt32(Console.ReadLine());

           Console.Write("Въведете операция (+, -, /, *): ");

            operand = Console.ReadLine();
```

```
            Console.Write("Въведете второ цяло число: ");

             num2 = Convert.ToInt32(Console.ReadLine());

             switch (operand)
             {

                 case "-":

                     answer = num1 - num2;

                     break;

                 case "+":

                     answer = num1 + num2;

                     break;

                 case "/":

                     answer = num1 / num2;

                     break;

                 case "*":

                     answer = num1 * num2;

                     break;

                 default:

                     answer = 0;

                     break;

             }

 Console.WriteLine(num1.ToString() + " " + operand + " " +
num2.ToString() + " = " + answer.ToString());
             Console.ReadLine();
```

```
      }
   }
}
```

Задача 3. Да се напише програма, която реализира въвеждане от клавиатурата модела на вашата кола и извежда на екрана производителя (Програмата да работи с предварително заложено множество от производители и модели).

Указание: Използвайте метод за сравняване на стрингове.

```
using System;


namespace Console
{
    class Program
    {
       static void Main()
        {
            string carModel;
            string carManufacturer;

     Console.Write("Моля въведете модел (на английски): ");

            carModel = System.Console.ReadLine();

            if ((String.Compare(carModel, "Patriot") == 0) ||
                (String.Compare(carModel, "Liberty") == 0) ||
                (String.Compare(carModel, "Wrangler") == 0))
            {
                carManufacturer = "Jeep";
            }
            else if (String.Compare(carModel, "Focus") == 0)
            {
                carManufacturer = "Ford";
            }
            else if (String.Compare(carModel, "Corolla") == 0)
            {
                carManufacturer = "Toyota";
            }
            else
            {
                carManufacturer = "Неизвестен производител.";
```

```
            }

     Console.Write("Производител: " + carManufacturer);

     }
  }
}
```

Второ решение с използване на switch

```
using System;


namespace Console
{
    class Program
    {


       static void Main()
        {
            string carModel;
            string carManufacturer;

System.Console.Write("Моля въведете модел (на английски): ");

            carModel = System.Console.ReadLine();

            switch (carModel)
            {
                case "Patriot":
                case "Liberty":
                case "Wrangler":
                    carManufacturer = "Jeep";
                    break;
                case "Focus":
                    carManufacturer = "Ford";
                    break;
                case "Corolla":
                    carManufacturer = "Toyota";
                    break;
                default:
            carManufacturer = " Неизвестен производител.";
                    break;
```

```
            }

    System.Console.Write("Производител: " + carManufacturer);

      }
   }
}
```

4.5. Задачи за самостоятелна работа

Задача 1. Да се напише програма, която използва оператор if, който проверява стойността на две целочислени променливи и разменя техните стойности, ако стойността на първата променлива е по-голяма от тази на втората.

Задача 2. Напишете програма, която за дадена цифра между 0 и 9, въведена от клавиатурата, извежда името на цифрата на български език.

Задача 3. Напишете програма, която за даден месец от годината, въведен като символен низ от клавиатурата, го извежда на екрана с цифра [1,12].

Задача 4. Да се напише програма, която прочита от клавиатурата резултат от тестове (в точки) и извежда на екрана оценката по шестобалната система по предварително утвърдена конвенция. За упътване вижте решението на задача 4, но използвайте оператора switch.

5. Цикли

В програмите често се налага да се изпълнят едни и същи оператори над множество от обекти. В програмирането такива алгоритми са наричат циклични. За реализацията на такива алгоритми в ЕП се използват т.нар. оператори за цикъл.

Цикъл (на англ. loop) е основна конструкция в програмирането и съществува във всички съвременни езици за програмиране. Той позволява многократно изпълнение на дадена част от кода на програмата. В зависимост от вида на използвания цикъл, програмният код в него се изпълнява или фиксиран брой пъти или докато е в сила дадено условие.

Цикъл, който никога не завършва изпълнението си се нарича безкраен цикъл (infinite loop). Такъв тип цикъл се използва много рядко и в тези случаи е необходимо в него по някакъв начин да се укаже кога да спре, обикновено се използва операторът break, за да бъде прекратено неговото изпълнение преждевременно.

За илюстрация нека разгледаме следния пример. Трябва да се състави програма за обслужване на банка, в която една от функционалностите е да начислява лихва (олихвява) N на брой банкови депозити. Операцията, която трябва да се извърши е увеличаване на сумата по депозита със сумата на начислената лихва, като тази операция се повтаря N на брой пъти. Очевидно не е ефективно да се пише един и същи код за всеки депозит, затова се използва някой от операторите за цикъл.

Езикът C# позволява да се използват няколко оператори за цикъл. Чрез тях се изпълняват многократно съвкупност от оператори наречени **тяло** на цикъла. То се изпълнява докато е удовлетворено някакво условие. В зависимост от това дали броят на изпълненията, наричат се още итерации, може да се определи еднозначно или не в C# могат да се използват операторите за цикъл for, foreach, while и do…while.

Ще започнем изучаването с най-често използваният от тях.

5.1. Оператор за цикъл for

Оператор **for** се използва най-често за обхождане на структура с фиксирана дължина, например масив и броят на итерациите може да бъде определен.

Синтаксис:

```
for ( <инициализиращ_оператор>; <израз1>; <израз2> )
{
<оператор>; //тяло на цикъла
}
```

Където:

Инициализиращ_оператор може да бъде декларация или израз. В общия случай той инициализира една или повече променливи, но може и да липсва. Примери:

```
for ( int i = 0; ...
for (; /* липсва инициализиращ_оператор */ ...
```

Израз$_1$ е ***условието за край***, което се проверява при всяко изпълнение/итерация на цикъла. Ако неговата стойност е различна от нула (true), се изпълнява тялото на for (един съставен или прост оператор).

Ако при първото изчисление на израз1 се получи нулева стойност, тялото на цикъла не се изпълнява. Пример: for (...; i < 20; ...)

Израз$_2$ (***стъпка***) се изчислява след всяка итерация на for. В общия случай той променя стойностите на променливите, инициализирани в инициализиращия оператор. Ако при първото изчисление на израз1,

Семантика:

1. Изпълнява се <инициализиращ_оператор>

2. Изчислява се <израз1> и ако е true, то: изпълнява се <оператор>, изчислява се <израз2> и отново се повтаря ст. 2, а ако <израз1> се оцени като false, цикълът се прекратява

неговата стойност е нула, израз2 не се изчислява. Пример: for (i=0; i < 20; i=i+2)

Пример:

```
for (int i = 0; i <= 20; i++)

{

          Console.Write(i + ",");

}
```

В резултат от изпълнение на този код на екрана ще се изведат числата [0,20] разделени със запетая: 0,1,2,3,4,….,20

5.2. Задачи за упражнение

Задача 1. Да се напише програма реализираща обратно отброяване от 10 до 1.

За решението ще използваме една особеност за управление на итерациите – случая когато знаем техния брой. За реализирането на обратно отброяване може да използваме детерминиран цикъл, тъй като знаем предварително броя на итерациите.

За целта в цикъла ще използваме променлива, която първоначално ще инициализираме с 10 и ще променяме, като я намаляваме с 1. Така променливата ще приема последователно следните стойности: 10, 9, 8, 7 … ,1. Остава само да изведем с подходящ метод за извеждане на нейната стойност.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
```

```
    {


       static void Main()
        {
            for (int n = 10; n > 0; n--)
            {
                Console.Write(n);
                Console.Write(", ");
            }
            Console.Write("Start!\n");

         }
    }
 }
```

Задача 2. Да се напише програма реализираща обратно отброяване от 10 до 1 и прекъсване при достигане на стойност 3.

За решението ще използваме подхода от предишната задача, като допълним с проверка в тялото на оператора за цикъл за достигане на стойността по условие. За целта ще използваме условен оператор if с проверка за достигане на 3 оператор ==.

За да може да прекъснем изпълнението на цикъла ще използваме оператор за прекъсване break.

Следващата програма решава задачата.

```
using System;


namespace Console
{
    class Program
    {


       static void Main()
        {
            for (int n = 10; n > 0; n--)
            {
                Console.Write(n);
                Console.Write(", ");
                if (n == 3)
                {
           Console.Write("Отброяването прекъснато на {0}!",n);
                    break;
                }

            }
            Console.Write("Start!\n");
```

```
          }
        }
    }
```

Задача 3. Да се напише програма реализираща обратно отброяване от 10 до 1 и пропускане на извеждането при достигане на стойност 5.

За решението ще използваме подхода от предишната задача, като допълним с проверка в тялото на оператора за цикъл за достигане на стойността по условие. За целта ще използваме условен оператор if с проверка за достигане на 5 оператор ==.

За да може да пропуснем извеждането на стойността ще използваме оператор continue.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {
            for (int n = 10; n > 0; n--)
            {
                if (n == 5)
                    continue;
                Console.Write(n);
                Console.Write(", ");
            }
 Console.Write("\n 5 се пропуска в отброяването. Start!\n");

     }
    }
  }
```

Задача 4. Да се напише програма, която извежда на екрана следното

1
22
333
4444
55555

Решение. Използвайте вложен цикъл. Обърнете внимание точно кой програмен код е тялото на всеки от циклите.

Извеждането трябва да стане на стойността на променливата от първия цикъл.

Следващата програма решава задачата.

```
using System;


namespace Console
{
    class Program
    {


       static void Main()
        {


            int i, j;
            i = 0;
            j = 0;

            for (i = 1; i <= 5; i++)
            {
                for (j = 1; j <= i; j++)
                {
                    Console.Write(i);
                }
                Console.Write("\n");
            }
      }
     }
   }
```

Задача 5. Да се напише програма, която извежда на екрана следното

1
12
123
1234
12345

Решение. Използвайте вложен цикъл. Обърнете внимание точно кой програмен код е тялото на всеки от циклите.

Извеждането трябва да стане на стойността на променливата от вложения цикъл.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {


            int i, j;
            for (i = 1; i <= 5; i++)
            {
                for (j = 1; j <= i; j++)
                {
                    Console.Write(j);
                }
                Console.Write("\n");
            }
      }
   }
}
```

*Задача 6. Да се напише програма, която по дадено естествено число **n**, намира факториела му. Внимание определете кое е най-голямото **n** , за което решението по-долу е вярно.*

За решението първо ще припомним как се намира факториела на дадено естествено число.

Факториел е функция на естествено число n, равна на произведението на първите n естествени числа. Например:

5! = 5*4*3*2*1 = 120

На следваща стъпка декларираме две променливи: за въвеждане на числото от клавиатурата **n** и за натрупване на произведението.

Ще използваме и цикъл for, в който управляващата променлива ще се променя от 1 до **n.** На всяка итерация променливата за произведението ще умножаваме с управляващата променлива.

Следващата програма решава задачата.

```
using System;


namespace Console
{
    class Program
    {
```

```
       static void Main()
        {
            Console.Write("Въведете n= ");
            int n;
            n = Convert.ToInt32(Console.ReadLine());
            if (n <= 0)
            {
                Console.Write("Некоректни входни данни! \n");
                return;
            }
            int fact = 1; // натрупване на произведението
            for (int i = 1; i <= n; i++)
            {
                fact = fact * i; // тяло на цикъла
            }

            Console.WriteLine("{0}! = {1} ",n,fact);

     }
   }
 }
```

Задача 7. Напишете програма извеждаща на екрана таблицата за умножение за произволно цяло число въведено от клавиатурата.

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {


            int number;

            Console.Write("Въведете число : ");
            number = Convert.ToInt32(Console.ReadLine());
            Console.Write("\n");
            Console.Write("Таблица за умножение");
            Console.Write("\n\n");

            for (int i = 1; i <= 10; i++)
            {
                Console.Write(number);
```

```
                Console.Write(" x ");
                Console.Write( i);
                Console.Write( " = ");
                Console.Write( number * i);
                Console.Write( "\n");
                Console.Write( "-------------");
                Console.Write( "\n");
            }
     }
   }
 }
```

Задача 8. Програма демонстрираща оператора ***foreach***

```
using System;
using System.Collections;


namespace Console
{
    class Program
    {


       static void Main()
        {


            ArrayList list = new ArrayList();
            list.Add("Иван");
            list.Add("Петър");
            list.Add("Симеон");

            foreach (string name in list)
                Console.WriteLine(name);
        }
     }
 }
```

5.3. Оператор за цикъл с предусловие while.

Друг оператор за цикъл е ***while***. Основна характеристика при него е, че броя на итерациите може да бъде различен при всяко изпълнение на програмата.

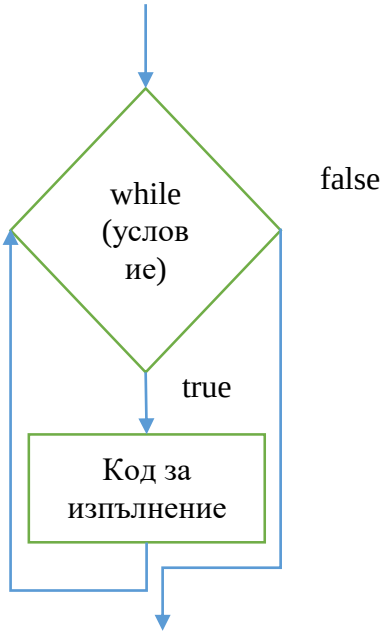
Синтаксис:

```
while (условие за край)
```

```
{
//тяло на цикъла
}
```

Семантика:

Условието за край се проверява ***първо*** и при всяко изпълнение/итерация на цикъла. Ако неговата стойност е различна от нула (true), се изпълнява тялото (един съставен или прост оператор). Ако условието за край е вярно още при първата проверка (false) тялото на цикъла няма да се изпълни нито веднъж. Семантиката на while цикъла може да се опише чрез следната схема:



Понеже условието за край на цикъла while се намира в неговото начало, той често се нарича цикъл с предусловие (pre-test loop). Ако условието за край никога не стане true, то той ще се изпълнява безкрайно.

Пример:

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Итерации с цикъл While ");
            int i = 0;
            while (i < 10)
            {
```

```
                    Console.WriteLine("Итерация {0}",i+1);

                    i++;
                }

                Console.ReadLine();

            }
        }
    }
```

При стартиране на програмата на екрана ще се изведе следното:

```
Итерации с цикъл While
Итерация 1
Итерация 2
Итерация 3
Итерация 4
Итерация 5
Итерация 6
Итерация 7
Итерация 8
Итерация 9
Итерация 10
```

5.4. Задачи за упражнения

Задача 1. Да се напише програма реализираща обратно отброяване от стойност въведена от клавиатурата

За решението ще използваме случай когато не знаем броя на итерациите по време на компилация, а техния брой ще бъде определен по време на изпълнение на програмата. В такива случаи не може да използваме детерминиран цикъл, тъй като не знаем предварително броя на итерациите. Затова ще използваме оператор за цикъл while.

За целта в цикъла ще използваме променлива, началната стойност на която ще въведем от клавиатурата и която ще намаляване до достигане на стойност 0, като я намаляваме с 1.

```
using System;


namespace Console
{
    class Program
    {


       static void Main()
        {
```

```
            int n;
            Console.Write("Въведете начална стойност : ");
            n=Convert.ToInt32(Console.ReadLine());

            while (n > 0)
            {
                Console.Write(n);
                Console.Write(", ");
                --n;
            }

            Console.Write("Start !\n");
     }
   }
}
```

Задача 2. Да се напише програма, която намира най-големият общ делител (НОД) (чрез изваждане) на две естествени числа въведени от клавиатурата.

За решението ще използваме алгоритъм на Евклид за намиране на НОД.

Стъпките на този алгоритъм вече бяха дадени в увода на ръководството, но все пак ще го припомним накратко. Алгоритъмът е следния: От по-голямото се вади по-малкото число и присвоява получения резултат. Това се изпълнява докато числата са различни. Когато се изравнят двете числа, то това е НОД(a,b).

Пример:
a=16; b=12
a=16-12= 4; b=12
a=4; b=12-4=8
a=4; b=8-4=4
a=4; b=4
НОД(a,b) = 4

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {


            int a, b;
            Console.Write("Въведете a : ");
            a=Convert.ToInt32(Console.ReadLine());
            Console.Write("Въведете b : ");
            b = Convert.ToInt32(Console.ReadLine());
```

```
            while (a != b)
            {
                if (a > b)
                {
                    a -= b;
                }
                else
                {
                    b -= a;
                }
            }
 Console.Write("Най-големият общ делител на числата е:{0}",a);

     }
   }
 }
```

Задача 3. Да се напише програма, която намира факториела на дадено естествено число. За целта да се използва операторът while.

Упътване. Припомнете си решението на задачата, чрез използване на оператор за цикъл for.

Следващата програма решава задачата.

```
using System;


namespace Console
{
    class Program
    {
       static void Main()
        {
            int n;
            Console.Write("Въведете n= ");
            n=Convert.ToInt32(Console.ReadLine());
            if (n <= 0)
            {
                Console.Write("Некоректни входни данни! \n");
                return ;
            }
            int fact = 1;
            int i = 1;
            while (i <= n)
            {
                fact = fact * i;
                i++;
            }
            Console.Write(n);
```

```
            Console.Write("! = ");
            Console.Write(fact);
            Console.Write("\n");

     }
    }
  }
```

Задача 4. Да се напише програма, която въвежда от клавиатурата редица от цели числа и намира средноаритметичното им. Въвеждането да продължава до въвеждане на 0.

Упътване: използвайте за условие за край на цикъла въвеждане на 0 от клавиатурата. while (число != 0)

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {

            int count = 0;
            double average = 0;
            Console.Write("Въведете число: ");
            int number;
            number=Convert.ToInt32(Console.ReadLine());
            while (number != 0)
            {
                count++;
                average = average + number;
                Console.Write("Въведете следващо число:  ");
                number = Convert.ToInt32(Console.ReadLine());
            }
            if (count != 0)
            {
                average = average / count;
        Console.WriteLine("Средна стойност = {0}",average);

            }
          }
       }
    }
```

Задача 5. Да се напише програма, която чете от клавиатурата положителни числа и намира минималния, максималния елемент и средната стойност на

въведените числа. Въвеждането продължава до въвеждане на отрицателно число.

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {


            double value;
            double sum;
            double average;
            double minimum;
            double maximum;
            int count;
            sum = 0.0F;
            count = 0;
            Console.Write("Въвдете число: ");
            value = Convert.ToDouble(Console.ReadLine());
            minimum = value;
            maximum = value;
            while (value >= 0.0)
            {
                sum += value;
                count++;
                if (value > maximum)
                {
                    maximum = value;
                }
                else if (value < minimum)
                {
                    minimum = value;
                }
                Console.Write("Въведете следващо число: ");
                value = Convert.ToDouble(Console.ReadLine());
            }
            if (count == 0)
            {
                Console.WriteLine("Няма въведени данни!");

            }
            else
            {
                average = sum / count;
```

```
                Console.Write("Вие въведохте  ");
                Console.Write(count);
                Console.Write(" числа.");
                Console.Write("\n");
                Console.Write("Средната стойност е : ");
                Console.Write(average);
                Console.Write("\n");
                Console.Write("Най-малкото е : ");
                Console.Write(minimum);
                Console.Write("\n");
                Console.Write("Най-голямото е : ");
                Console.Write(maximum);
                Console.Write("\n");
            }


        }
    }
}
```

*Задача 6. Да се напише програма която чете от клавиатурата цяло число **n** и проверява дали то е просто число.*

```
using System;


namespace Console
{
    class Program
    {


       static void Main()
        {


           Console.Write("Въведете цяло положително число: ");

            int num = int.Parse(Console.ReadLine());

            int divider = 2;

            int maxDivider = (int)Math.Sqrt(num);

            bool prime = true;

            while (prime && (divider <= maxDivider))
            {

                if (num % divider == 0)
```

```
                {
                    prime = false;
                }
                divider++;
            }
            Console.WriteLine("Дали е просто ? " + prime);

        }
    }
}
```

5.5. Оператор за цикъл с постусловие do … while.

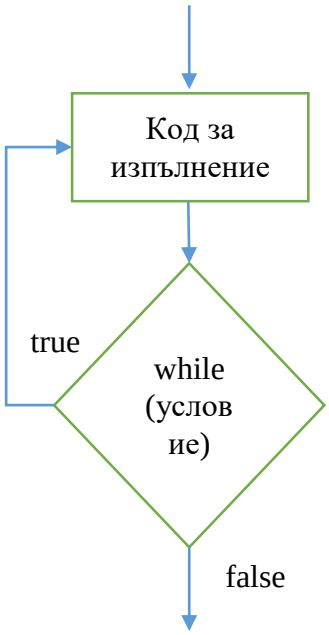
do … while цикълът е аналогичен на while цикъла, но при него проверката на условието за край се извършва след изпълнението на операциите в тялото на цикъла. Този тип цикли се наричат цикли с пост условие (post-test loop).

Синтаксис:

```
do
{
// тяло на цикъла
}
while (условие за край);
```

Семантика:

Условието за край се проверява ***след*** всяко изпълнение/итерация на цикъла. Ако неговата стойност е различна от нула (true), се изпълнява тялото (един съставен или прост оператор). Ако условието за край е вярно още при първата проверка (false) тялото на цикъла ще се изпълни само веднъж. Семантиката на do … while цикъла може да се опише, чрез следната схема:



Пример: програма, която намира сумата на естествените числа от 0 до 5.

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
                int sum = 0;
                int i = 0;
                do
                {
                    sum += i;
                    i++;
                } while (i < 5);

                Console.WriteLine("Сумата е {0}", sum);


                Console.ReadLine();

            }
        }
    }
```

При стартиране на програмата на екрана ще се изведе следното:

```
Сумата е 10
```

5.6. Задачи за упражнение

Задача 1. Да се напише програма, която намира сумата на естествените числа в интервала 1..30.

За решаването на задачата използваме подхода от примера.

Декларираме три променливи съответно за начало и за край на интервала и за сумата, която ще натрупваме на всяка итерация на цикъла.

Внимание! Натрупването трябва да започне от 0, тъй като на първата итерация имаме добавяне на управляващата за цикъла променлива.

```
using System;


namespace Console
{
    class Program
    {
       static void Main()
        {

           const int max=30;
            int sum = 0;
            int num = 1;
            do
            {
                sum = sum + num;
                num++;
            } while (num <= max);
            Console.Write("Сумата е:");
            Console.Write(sum);
            Console.Write("\n");


         }
       }
    }
```

Задача 2. Да се напише програма, която намира произведението на целите числа от m до n, където m и n са дадени естествени числа и m ≤ n. За целта да се използва операторът do/while.

```
using System;


namespace Console
{
    class Program
    {
```

```
      static void Main()
      {

       Console.Write("Въведете m= ");
int m;
m=Convert.ToInt32(Console.ReadLine());

if (m <= 0)
{
Console.Write("Некоректни входни данни! \n");
return;
}
Console.Write("Въведете n= ");
int n;
 n=Convert.ToInt32(Console.ReadLine());

if (n <= 0)
{
    Console.Write("Некоректни входни данни! \n");
return ;
}
if (m > n)
{
    Console.Write("Некоректни входни данни! \n");
return ;
}
int prod = 1;
int i = m;
do
{
prod = prod * i;
i++;
} while (i <= n);
Console.WriteLine("Произведението е {0}",prod);

          }
       }
    }
```

Задача 3. Да се напише програма демонстрираща начин за непрекъснато изпълнение на оператори до посочване на отговор за прекъсване.

```
using System;


namespace Console
{
    class Program
    {
```

```
        static void Main()
         {

             char ans;
             do
             {
                 Console.Write("Извършват се някакви операции !
\n ");
                 Console.Write("Желаете ли да продължите
изпълнението (произволен символ за Да или N за Не)?\n");
                 ans = Console.ReadKey().KeyChar;
             } while (ans != 'N');



           }
        }
    }
```

Задача 4. Да се напише програма, която преобразува символите на въведен от клавиатурата низ от малки в главни букви или обратно. Въвеждането продължава до въвеждане на точка.

```
using System;


namespace Console
{
    class Program
    {


        static void Main()
         {


             char ch;

Console.Write("Въведете текст и (точка) . за изход).\n");
             do
             {
                 ch = Console.ReadKey().KeyChar;
                 if (char.IsLower(ch))
                 {
                     ch = char.ToUpper(ch);
                 }
                 else
                 {
```

```
                    ch = char.ToLower(ch);
                }

                Console.Write(ch);
            } while (ch != '.');



        }
    }
}
```

Задача 5. Да се напише програма, която демонстрира валидиране на входните данни. Данните представляват число за ден от месеца в интервала [1 , 31].

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int number;
            bool isValid = false;

            do
            {
         Console.Write("Въведете ден от месеца от 1 до 31: ");
                number=Convert.ToInt32(Console.ReadLine()); ;
                isValid = (number >= 1) && (number <= 31);

                if (isValid)
                {
           Console.WriteLine("Вие въведохте коректно деня.");

                    break;
                }

                else
                {
Console.WriteLine("Некоректни данни. Моля опитайте отново.");

                }
            } while (!isValid);



        }
```

```
            }
        }
```

5.7. Задачи за самостоятелна работа

Задача 1. Напишете програма извеждаща на екрана сумата на естествените числа в интервала [10,100].

Задача 2. Напишете програма извеждаща на екрана произведението на четните числа в интервала [5,15].

Задача 3. Да се напише програма, която прочита от клавиатурата редица от 10 естествени числа и за всяко проверява, дали цифрата 3 участва в записа му.

Задача 4. Да се напише програма, която въвежда от клавиатурата редица от 20 цели числа и намира и извежда на екрана средноаритметичното им.

Задача 5. Напишете програма, която прочита от клавиатурата дневните разходи за 31 дни на едно семейство. Да се изведат на екрана:

А) общата сума на разходите за месеца;

Б) средната стойност на дневните разходи.

6. Масиви

В тази тема се разглеждат масивите като средство за обработка на поредица от данни от едни и същи тип. Ще разясним какво представляват масивите, как можем да ги декларираме, инициализираме със стойности и използваме масиви в нашите програми. Ще обърнем особено внимание на едномерните и многомерните масиви.

Дефиниция:

Масивът (array**)** представлява наредена крайна съвкупност от еднотипни елементи, до които е осигурен пряк достъп. Масивът обикновено е статичен тип данни.

Характеристики:

- ***краен*** означава, че елементите са краен брой;
- ***статичен*** означава че размерът на масива трябва да се знае по време на компилирането, но това не означава че всички места в масива трябва да съдържат смислени данни;
- ***подреден*** означава, че относителното място на всеки елемент в масива е предварително дефиниран;
- ***еднотипни компоненти*** означава, че елементите са от един и същ тип данни;
- ***прекия достъп*** до елемент, означава, че при достъп до елемент на масива не е необходимо да се обръщаме към предишните елементи.

Приложение:

Масивите се използват най-често за моделиране на подредени по някакъв признак еднотипни данни, които се подлагат на еднотипна обработка.

Пример:

За студентите от даден курс често се налага да знаем кой студент е с най-висок среден успех, кои са студентите с отличен успех или какъв е средният успех на целия курс. В този случай е подходящо да се използва масив, като наредбата на компонентите (оценки на студентите) се извършва, например по факултетния номер на студента.

Масивите са неизменна част от повечето езици за програмиране. Те представляват съвкупности от обекти, които наричаме елементи или компоненти. Броят елементи в даден масив се нарича **дължина на масива или размерност** (*dimension*). Всички елементи на даден масив са от един и същи тип, който наричаме **базов**. Масивите могат да бъдат едномерни, двумерни и многомерни според това как са подредени елементите им.

Достъп:

Мястото на елемента в масива е строго определено от набора на всичките му индекси по всички размерности на масива.

Желаният елемент се посочва чрез неговия *индекс* (или индекси), което дава относителното му място в съвкупността.

Конструиране:

Конструирането на масивите, т.е. изграждането на съвкупността от стойностите на неговите елементи най-често се извършва поелементно, чрез

обхождане (*traversing*) на масива. Това означава че ние последователно се обръщаме към елементите (чрез техните индекси) и записваме необходимите данни в тях.

Ще започнем изучаването с едномерните масиви в C#.

6.1. Едномерен масив

Масивът е наредено множество от еднотипни обекти с пряк достъп.

Декларация:

```
<тип> [константен израз] <идентификатор>;
```

Дефиницията на масив съдържа спецификатор на тип, идентификатор и размерност.

Размерността определя броя на елементите на масива и се загражда в квадратни скоби [брой елементи].

Стойността трябва да бъде поне единица и да е константен израз, т.е. израз, чиято стойност може да се изчисли по време на компилация. Това означава, че в дефиницията на масив променлива не може да се използва като размерност.

В C# се поддържат статични масиви, което означава, че задължително указваме броя на елементите при декладацията на масива.

Примери:

```
int[] numbers; // декларация на масив
numbers = new int[10];  // от 10 цели числа
double prices = new double[20];    // от 20 реални числа
char[] chararray= new char[30]; //масив от 30 символа
```

Друг пример: Масив от низове

```
string[] strArray = new string[] { "Иван Петров", "Ненчо
Стойков", "Стоян Манов" };
```

Обръщението към отделните елементи на масива се извършва с посочване на неговия индекс (позиция в масива). Можем да се обърнем към всеки елемент на масива директно т.е. не е необходимо да извършваме допълнителни действия за да достигнем до всеки отделен елемент, което представлява свойството пряк достъп.

Пример:

```
int[] staticIntArray = new int[3];
staticIntArray[0] = 1;
staticIntArray[1] = 3;
staticIntArray[2] = 5;
```

Номерирането на индексите на елементите на масив започва от 0. За масив от 10 елемента правилните стойности на индекса са от 0 до 9 включително, а не от 1 до 10. Тази особеност е причина за програмни грешки.

Следващата програма използва цикъл **for**, за да **обходи** елементите на масива, като инициализира съответния елемент със стойността на неговия индекс.

```
     using System;
```

```
namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {
            const int SIZE = 30;
            int[] A = new int[SIZE];

            {
                for (int i = 0; i < SIZE; ++i)
                {
                    A[i] = i;
                }


            }
        }

    }
}
```

Масив може да се инициализира още при дефинирането си чрез списък от стойности, отделени със запетаи и заградени във фигурни скоби.

Пример:

```
const int SZ =3;
int[ SZ ]  M = { 0, 1, 2 };
```

За масив, който се инициализира при дефиниране, не е необходимо да се задава размерността. Компилаторът определя броя на неговите елементи по броя на заградените в скоби елементи (Виж примера за масива от низове по-горе).

6.2. Задачи за упражнение

Задача 1. Да се напише програма, която въвежда последователно n числа, след което ги извежда в обратен ред.

За тази задача е необходимо да се декларира масив с определена размерност. Тъй като условието не указва изрично каква е тази размерност, можем да си изберем един максимален брой елементи, но потребителя да въведе броя ***n*** от клавиатурата. Въвеждането се реализира с обхождане на масива последователно и прочитане на стойност от клавиатурата.

Тъй като в условието не е зададен типа на числата е препоръчително той да бъде реален.

За извеждане в обратен ред е достатъчно да използваме цикъл, който да променя индексите в намаляващ ред от ***n*** до 0.

Решение:

```
using System;

namespace Console
{
    class Program
    {
        static void Main()
        {
            double[] x = new double[100];
            Console.Write("Брой на елементите n= ");
            int n;
            // въвеждане на стойност за n
            n=Convert.ToInt32(Console.ReadLine());

            if (n < 0 || n > 100)
            {
                Console.Write("Некоректни входни данни! \n");
                return ;
            }
            // n е цяло число от интервала [1, 100]
            // въвеждане на стойности за елем. на масива x
            for (int i = 0; i <= n - 1; i++)
            {
                Console.Write("x[");
                Console.Write(i);
                Console.Write("]= ");
                x[i]=Convert.ToDouble(Console.ReadLine());;

            }
         // извеждане на елементите на x в обратен ред
            for (int i = n - 1; i >= 0; i--)
            {
                Console.Write(x[i]);
                Console.Write("\n");
            }


}        }
    }
```

Задача 2. Дадени са редицата от цели числа a_0, a_1, …, a_{n-1} ($n \geq 1$) и цялото число x. Да се напише програма, която намира колко пъти x се съдържа в редицата.

Аналогично на предходната задача броят на числата не е указан затова може да приложим същия подход.

За да се намери броят на срещанията на *x* в масива е необходимо да използваме променлива, в която да отразяваме(увеличаваме с 1) всяко срещане.

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {
            int[] a = new int[20];
            Console.Write("Брой елементи n= ");
            int n;
           // въвеждане на дължината на редицата
            n = Convert.ToInt32(Console.ReadLine());

            if (n < 1 || n > 20)
            {
                Console.Write("Некоректни входни данни! \n");
                return ;
            }
            // въвеждане на редицата
            int i;
            for (i = 0; i <= n - 1; i++)
            {
                Console.Write("a[");
                Console.Write(i);
                Console.Write("]= ");
                a[i] = Convert.ToInt32(Console.ReadLine());

            }
            // въвеждане на стойност за x
            int x;
            Console.Write("Въведете x= ");
            x = Convert.ToInt32(Console.ReadLine()); ;

         // намиране на броя br на срещанията на x в редицата
            int br = 0;
            for (i = 0; i <= n - 1; i++)
            {
                if (a[i] == x)
                {
                    br++;
                }
            }
            Console.Write("Брой срещания: ");
            Console.Write(br);
            Console.Write("\n");
```

```
        }
      }
    }
```

Задача 3. Да се напише програма, която установява, дали редицата от цели числа $a_0, a_1, ..., a_{n-1}$ е монотонно намаляваща.

Една редица наричаме монотонно намаляваща такава, при която $n>k \Rightarrow a_n \leq a_k$.

За да решим задачата е необходимо да проверяваме всеки две съседни числа и да отброяваме, ако първото е по-голямо от второто. Когато броят е равен на n-1 имаме монотонно намаляваща.

Ако a[i]>=a[i+1]
отброяваме - брояч+1

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int[] a = new int[20];
            Console.Write("Брой елементи n= ");
            int n;
           // въвеждане на дължината на редицата
            n = Convert.ToInt32(Console.ReadLine());

            if (n < 1 || n > 20)
            {
                Console.Write("Некоректни входни данни! \n");
                return ;
            }
            // въвеждане на редицата
            int i;
            for (i = 0; i <= n - 1; i++)
            {
                Console.Write("a[");
                Console.Write(i);
                Console.Write("]= ");
                a[i] = Convert.ToInt32(Console.ReadLine());

            }
            int br = 0;
            for (i = 0; i <= n - 2; i++)
            {
                if (a[i] >= a[i + 1])
                {
                    br++;
                }
```

```
            }
            if (br == n - 1)
            {
                Console.Write("Да \n");
            }
            else
            {
                Console.Write("Не \n");
            }


     }
     }
    }
```

Задача 4. Напишете програма реализираща въвеждането от клавиатурата на елементите на масив A с размерност 50 и извеждаща на екрана най-големия елемент.

Намирането на най-големия елемент на масив може да се реализира по следния алгоритъм.

Използваме една променлива, в която ще запазваме текущия максимум. Първоначално нейната стойност ще бъде първия елемент на масива.

На следваща стъпка първим обхождаме на останалите елементи и за всеки проверяваме дали е по-голям от текущия мяксимален.

Ако е така то този елемент става максимален. В противен случай текущия максимум не се променя.

```
using System;

namespace Console
{
    class Program
    {
       static void Main()
        {
            int[] array = new int[50];
int i;
for (i = 0; i < 50; i++)
{
Console.Write("A [");
Console.Write(i);
Console.Write("]= ");
array[i]=Convert.ToInt32(Console.ReadLine());
}
int max = array[0];
      for (i = 1; i < 50; i++)
      {
            if (array[i] > max)
            {
```

```
            max = array[i];
            }
       }
      Console.Write("Най-голям елемент:");
      Console.Write(max);
      }
     }
  }
```

6.3. Двумерен масив

Двумерният масив представлява съвкупност от еднотипни елементи подредени в редове и колони. При него имаме два индекса. Първият е за номера на реда, вторият е за номера на колоната. Пример за двумерен масив е матрицата. Достъпът до елементите на двумерния масив се осъществява чрез името на масива и двата индекса на масива: за номера на реда и за номера на колоната. Освен за съхраняване и обработка на числови стойности двумерните масиви могат да се използват и със символни низове, например за съхраняване на имена.

Декларация:

Декларацията е сходна с тази на едномерен масив:

```
int[,] a = new int[3,5]; //цели числа - 3 реда 5 колони
string[,] array = new string[2,2]; //низове - 2 реда 2 колони
```

Достъпът до елементите се осъществява, чрез двата индекса. За демонстрация нека разгледаме следния пример. Задачата е да се реализира инициализиране на двумерен масив 5x2 и да се изведат на екрана стойностите. За **обхождането** на масива се използват цикъл и вложени в него друг цикъла. Първият управлява индекса на реда, а втория на колоната.

Пример:

```
using System;
namespace Console
{
   class Program
   {
      static void Main(string[] args)
      {
         /*  5 реда и 2 колони*/
int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
         int i, j;

         /* извеждане на елементите */
         for (i = 0; i < 5; i++)
         {
            for (j = 0; j < 2; j++)
            {
         Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);
            }
```

```
            }
            Console.ReadKey();
        }
    }
}
```

6.4. Задачи за упражнения

Задача 1. Да се напише програма, която използва двумерен масив 3 реда и 3 колони за съхраняване на имена на книги.

За решаване на задачата е нужно да декларираме двумерен масив от тип string. Условието изисква масивът да е с размерност 3 реда и 3 колони, затова декларацията ще бъде подобна на тази: string[,] twodimarray = new string[3, 3];.

Както вече знаем, за да обхождаме двумерен масив е необходимо да използваме два цикъла, които ще управляват съответно индексите по редове и колони. Чрез такива два цикъла ще въвеждаме и извеждаме елементите на масива с книгите.

Ето програмата, която решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j;
            string[,] Books = new string[3, 3];
            for (i = 0; i < 3; i++)
            {
                for (j = 0; j < 3; j++)
                {
 Console.Write("\nВъведете име на книга за {0} ред и {1}
колона:\t", i + 1, j + 1);
                    Books[i, j] = Console.ReadLine();
                }
            }

            Console.WriteLine("Списък:\n\n");

            Console.Write("\t1\t2\t3\n\n");

            for (i = 0; i < 3; i++)
            {
                Console.Write("{0}.\t", i + 1);


                for (j = 0; j < 3; j++)
                {
                    Console.Write("{0}\t", Books[i, j]);
```

```
                }
                Console.Write("\n");
            }

            Console.ReadLine();
        }
    }
}
```

Задача 2. Напишете програма, която създава два двумерни масива, елементите на първия масив се въвеждат от клавиатурата и след това се копират във втория.

Вече знаем как се въвеждат елементи на двумерни масиви с използване на двойка цикли за обхождане на редовете и колоните.

В конкретната задача за осъществяване на копиране на елементите във втори масив е нужно само да обхождаме първия и всеки елемент да го присвояваме на елемент със същия индекс от втория масив. С използване на цъкъл for алгоритъма за копиране ще има следния код:

```
for (ред = 0; ред < брой_редове; ред ++)
     {
          for (колона = 0; колона < брой_колони; колона ++)
            {
          втори_масив[ред, колона] = първи_масив[ред, колона];
            }
     }
```

Ето програмата, която решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {

            int[,] arr1 = new int[3, 3];
            int[,] arr2 = new int[3, 3];
            int i, j;

  Console.Write("Въведете 9 елемента на двумерен масив:\t");

            for (i = 0; i < 3; i++)
            {
                for (j = 0; j < 3; j++)
                {

        arr1[i, j] = Convert.ToInt32(Console.ReadLine());
```

```
                }
            }
            //копиране
            for (i = 0; i < 3; i++)
            {
                for (j = 0; j < 3; j++)
                {
                    arr2[i, j] = arr1[i, j];
                }
            }

 Console.WriteLine("\n\nЕлементите на втория масив са:\n\n");
            //извеждане
            for (i = 0; i < 3; i++)
            {
                Console.WriteLine();
                for (j = 0; j < 3; j++)
                {
                    Console.Write("\t{0}", arr2[i, j]);
                }
            }
            Console.ReadLine();

        }
    }
}
```

Задача 3. Напишете програма, която реализира умножението на две матрици.

Указание: използвайте умножаване на ред по стълб.

Ако две матрици A и B са от вид A_{nxm} и B_{mxp} (броят на стълбовете на първата матрица е равен на броя на редовете на втората) произведението A.B може да бъде извършено.

Резултатът е трета матрица C от вида C_{nxp}. Cij е произведението на i - тия ред на A по j - тия стълб на B.

Ето програмата, която решава задачата.

```
using System;
namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j;
            int[,] a = new int[2, 2];
```

```
            Console.WriteLine("Въведете матрица 2*2 ");
            for (i = 0; i < 2; i++)
            {
                for (j = 0; j < 2; j++)
                {
                    a[i, j] = int.Parse(Console.ReadLine());
                }
            }
            Console.WriteLine("Въведената матрица е:");
            for (i = 0; i < 2; i++)
            {
                for (j = 0; j < 2; j++)
                {
                  Console.Write(a[i,j]+"\t");
                }
                Console.WriteLine();
            }
            int[,] b = new int[2, 2];
            Console.WriteLine("Въведете втора матрица 2*2 ");
            for (i = 0; i < 2; i++)
            {
                for (j = 0; j < 2; j++)
                {
                    b[i, j] = int.Parse(Console.ReadLine());
                }
            }
            Console.WriteLine("Втората матрица е:");
            for (i = 0; i < 2; i++)
            {
                for (j = 0; j < 2; j++)
                {
                    Console.Write(b[i, j] + "\t");
                }
                Console.WriteLine();
            }
```

```
            Console.WriteLine("Умножение на матриците:");
            int[,] c = new int[2, 2];
            for (i = 0; i < 2; i++)
            {
                for (j = 0; j < 2; j++)
                {

                    c[i,j]=0;
                     for (int k = 0; k < 2; k++)
                     {
                         c[i, j] +=  a[i, k] * b[k, j];
                     }
                 }
            }
            for (i = 0; i < 2; i++)
            {
                for (j = 0; j < 2; j++)
                {
                    Console.Write(c[i, j]+"\t");
                }
                Console.WriteLine();
            }

            Console.ReadKey();
        }
    }
}
```

6.5. Задачи за самостоятелна работа

Задача 1. Напишете програма реализираща въвеждането от клавиатурата на дневните разходи на едно семейство за 30 дни. Да се изведе на екрана най-високия дневен разход и да се изведе на екрана общата сума на разходите за 30-те дни.

Задача 2. Напишете програма реализираща произведение на вектор с число.

Задача 3. Напишете програма реализираща скаларното произведение на два вектора.

Задача 4. Напишете програма, която проверява дали редицата от числа a_0, a_1, ..., a_{n-1}. *се състои от различни елементи.*

Указание: Вижте решението на задачата за монотонно намаляваща редица, но проверката е за различни двойки числа.

Задача 5. Напишете програма, която изчислява и извежда на екрана сумата от елементите на всеки стълб на матрица a[nxm].

7. Символен тип и символен низ

Досега в нашите програми ние използвахме главно числови стойности. Понякога по различни причини в програмите се налага да се използват и единични символи, за да се представи или кодира някаква информация. Подходящ пример е представянето на пола на човек с една буква – М за мъж и Ж за жена. В тази ситуация в програмирането се използват променливи от символен тип.

Символният тип представя единичен символ. В езика C# той се декларира с ключовата дума **char**. Всеки символ се представя с код от Unicode таблицата. Unicode е международен стандарт, който съпоставя цяло число на всеки знак от човешките познати писмености (всички езици и техните азбуки). Минималната стойност, която може да заема типът char, е 0, а максималната – 65535. Стойностите от тип char представляват букви или други символи и се ограждат в апострофи.

Декларация:

Декларацията на променлива от символен тип по нищо не се различава от декларация на променлива от другите примитивни типове.

Пример за деклариране на променлива от символен тип.

```
char symbol = 'x';
```

При декларацията може да бъде направена и инициализация. За инициализирането може да се използват стойности в различни формати – шестнадесетичен, преобразувано цяло число или Unicode. Следващата задача демонстрира декларация.

Задача 1. Да се напише програма, която демонстрира различна форма на записване на константи от тип ***char***

```
using System;
using System.Globalization;
using System.Collections;


namespace ConsoleOOP
{
    class Program
    {


       static void Main()
        {
            char[] chars = new char[4];

            chars[0] = 'X';
            chars[1] = '\x0078';    // шестнадесетичен
            chars[2] = (char)68;    // преобразувано от int
            chars[3] = '\u0058';    // Unicode
```

```
            foreach (char c in chars)
            {
                Console.Write(c + " ");
            }


        }
       }
    }
```

И тук, както в други езици за програмиране могат да се използват някои специални символи за форматиране, за представяне на някои препинателен знак и др. Малка част са дадени в следващата таблица.

Символ	UTF-16	Описание
\'	\u0027	апостроф
\"	\u0022	кавички
\\	\u005c	наклонена черта
\0	\u0000	символ за край ма низ
\a	\u0007	alarm (звуков сигнал)
\t	\u0009	хоризонтален tab
\v	\u000b	вертикален tab

Използването на тези символи може да става в програми от всякакъв вид, т.е. не се ограничава използването им само до конзолни приложения.

7.1. Операции

Над символен тип са позволени следните операции:

Намиране на кода на символ

Пример:

```
char ch = 'x';
int code = (int)ch; //стойност 120
```

Намиране на символ по неговия код

Пример:

```
int code = 117;
char c=(char)code; //стойност u
```

Аритметични операции

Всички аритметични операции допустими над целочислен тип са приложими над символен тип. Операциите се извършват над кодовете на символите. Резултатът от операциите е цяло число.

Примери:

```
    int letters = 'z' - 'a' + 1; // стойност 26
    int e = 'a' + 4;//стойност 101
```

7.2. Символни низове

В практиката често се налага обработката на текстове: четене на данни от текстови файлове, търсене на ключови думи в текст, заместване на думи в параграф, валидиране на входни данни и др. Във всички тези случаи можем да запишем текста в символни низове и последствие да го обработим с помощта на вградените методи за работа с низове или да реализираме наши алгоритми.

Символният низ, нарича се още стринг е последователност от символи, записана в паметта, като другите типове променливи. За разлика от типа char в променливите от тип стринг съхраняваме повече от един символ.

В различните езици за програмиране символните низове се реализират по различни начини. Програмистите също могат да изберат различен подход за реализацията.

Първият подход е да използват масив от тип char. Например в C# - char[], но обработката тук е по-сложна – трябва да се обръщаме към всеки елемент(символ) поотделно с неговия индекс.

Вторият подход е да използваме вградените средства на езиците за работа с низове от символи реализирани като класове. Такъв е типа string в езиците C++, Java или C#. Този подход е препоръчителен, тъй като може да използваме множество стандартни методи за обработка.

Тук ще разясним само втория подход, за да улесним читателя, а в следващата глава ще разгледаме подробно масивите и техните характеристики, вкл. и масиви от символи.

C# класът System.String позволява обработка на символни низове по много лесен и интуитивен начин. За декларация на низове се използва служебната дума string, която е референтен тип в C#.

Типът String е различен от останалите типове данни, които разгледахме дотук(т.нар. стойностни). Всъщност е клас и като такъв той отговаря на принципите на обектно-ориентираното програмиране. Стойностите му се записват в динамичната памет, а променливите от тип string имат стойност референция към обект в динамичната памет.

Декларация:

Пример за декларация на символен низ:

```
    string  sayHello = "Hello C# strings";
```

Инициализацията на променлива от тип string може да бъде освен директно със символен низ, както е в горния пример, така и чрез присвояване – резултат от израз, който връща символен низ.

Конкатенация(свързване) на символни низове

В израз стрингове могат да се свързват с оператор „+“ или метод Concat(). Например contact в следващия пример:

```
string phone = "08884581xxx";
string contact = "Свържете се с мен на номер: " + phone;
```

Пример с използване на метод Concat().

```
string name1 = "Иван";
string name2 = "Петров";
string fullname=string.Concat(name1, name2);
Console.WriteLine(fullname); // резултатът е ИванПетров
```

Както се вижда от примера символните низове буквално се залепват един за друг без разделител между тях. В някои ситуации това може да бъде проблем.

В C# проблемът се решава с използване на метода Join(сепаратор, масив от низове).

Пример:

```
string[] names = new string[2] { "Стоян", "Петков" };
            string sep=",";
string fullname = string.Join(sep, names);

Console.WriteLine(fullname); // резултатът е: Стоян,Петков
```

Освен операция свързване, върху стрингове могат да се прилагат и други операции. За запознаване с някои от най-често използваните ще дадем няколко кратки примера.

Сравняване на символни низове за еднаквост

Ако в програмата се изисква да сравним два символни низа и да установим дали стойностите им са еднакви или не, лесен начин е използване на методът **Equals(string, string),** или неговия еквивалент оператора „**==**“, оператора “**!=**“ използваме за проверка за различие. Методът връща булев резултат със стойност true, ако низовете имат еднакви стойности, и false, ако те са различни. Методът Equals работи, като проверява побуквено стойностите на двата низа. Внимание, отчита се разлика между малки и главни букви.

```
string name1 = "Иван";
string name2 = "иван";
bool eq=String.Equals(name1, name2); // eq = false

string name3 = "Стоян";
string name4 = "Стоян";
bool eq2 = name3 == name4; // eq2 = true
```

Търсене на подниз в низ

Често в програмите се стига до ситуация, при която се налага да се провери дали в един символен низ се съдържа друг. Всички знаем за търсещите системи в интернет, където ние задаваме ключова дума(низ), а резултатът от търсенето е уеб страници(в заглавие, в описанието или в параграфите), в които се среща търсения низ. В тази ситуация може да използваме метода String.Contains()

Пример:

```
            string name1 = "Иван Петров";
            string name2 = "Иван";
            if (name1.Contains(name2))
            {
```

```
                Console.WriteLine("Да съдържа се.");

    }
```

Заместване на низ в низ

С метода String.Replace() може да се замества низ в друг низ , колкото и пъти да се среща. Ситуацията е доста често срещана при въвеждане на тест потребителя може да сбърка част от низа. Ето един нагледен пример за заместване на сбъркано име в текст.

```
string corruptedString = "Ивал написа следното. Аз съм Ивал, "
+" на 25 години съм и уча Програмиране.";

Console.WriteLine("Сгрешен низ:{0}",corruptedString);


string correctString = corruptedString.Replace("Ивал", "Иван");
Console.WriteLine(correctString);
```

7.3. Задачи за упражнение

Задача 1. Да се напише програма, която въвежда малка буква от латинската азбука и извежда съответната ѝ главна буква.

За да решим задачата трябва да декларираме променлива от символен тип, в която ще съхраним въведената от клавиатурата малка латинска буква.

Прочитането става с метод Console.ReadKey(), но след това трябва да се обърнем и към свойството KeyChar на този метод за да получим Unicode кода на символа. На следващо място трябва да обърнем и внимание на това дали потребителя е въвел действително малка латинска буква. Затова е важно да се направи проверка с условен оператор и да прекъснем изпълнението ако не е въведено коректно буква. Проверката е по код със стойност между кодовете на символите 'a' и 'z' и така:

ако (въведеният_символ < 'a' или въведеният_символ > 'z')

то е некоректно въведена буквата

Ако е коректно въвеждането остава да приложим малко аритметични действия и преобразуване и да получим главната буква. Ето пример:

Ако е въведена малка буква s (код 115) и извадим кода на ‚a‘(97) и съберем с кода на ‘A’(65) получаваме 115-97+65=83 – код на ‘S’.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {
            char c;
```

```
            c = Console.ReadKey().KeyChar;
            if (c < 'a' || c > 'z')
            {
                Console.Write("Некоректен символ! \n");
                return ;
            }
            Console.Write((char)(c - 'a' + 'A'));
            Console.Write('\n');

        }
    }
  }
```

Задача 2. Да се напише програма, която извежда цифрите, главните и малките букви на латинската азбука и кодовете им. Всеки символ и кодът му да са на един и същ ред. За разделител между символ и код да се използва специалния символ за хоризонтална табулация.

За да решим задачата трябва да реализираме три цикъла, които да се управляват със исканите по условие цифри, главните и малките букви на латинската азбука. Така ще обходим всички символи последователно.

На следваща стъпка в тялото на всеки цикъл трябва да изведем символа и след табулация (използвайки '\t') да изведем и кода, който може да получим ако преобразуваме символа до цяло число, чрез оператор: (int) ‘символ’.

Следващата програма решава задачата.

```
using System;


namespace Console
{
    class Program
    {


       static void Main()
        {
            char ch;
            Console.WriteLine("Таблица с кодовете \n");
            for (ch ='0'; ch <= '9'; ch++)
            {
                Console.Write(ch);
                Console.Write('\t');
                Console.Write((int)ch);
                Console.Write('\n');
            }
            for (ch = 'A'; ch <= 'Z'; ch++)
            {
                Console.Write(ch);
                Console.Write('\t');
```

```
                Console.Write((int)ch);
                Console.Write('\n');
            }
            for (ch = 'a'; ch <= 'z'; ch++)
            {
                Console.Write(ch);
                Console.Write('\t');
                Console.Write((int)ch);
                Console.Write('\n');
            }

        }
       }
    }
```

Задача 3. Да се напише програма, която извежда върху екрана следната таблица:

A B C D E F
B C D E F G
C D E F G H
D E F G H I
E F G H I J

За решаването на задачата проучете внимателно десетичните кодове на символите от условието. Тъй като са последователни с използване на цикли може да се изведат на редове с изместване за веки следващ ред.

```
using System;

namespace Console
{
    class Program
    {


       static void Main()
        {
           char ch;
           for (int i = 0; i < 5; i++)
{
for (ch = (char)(65 + i); ch <= (char)(65 + i + 5); ch++)
{
Console.Write(ch);
Console.Write(" ");
```

```
}
Console.Write("\n");
}


        }
       }
    }
```

Задача 4. Да се напише програма, която прочита от клавиатурата списък с 5 имена. Да се въведе и име за търсене. Програмата да изведе на екрана съобщение дали името се съдържа в списъка и на коя позиция.

При решаването на задачата трябва да използваме низ от 5 елемента от тип string за да съхраним имената в списък. В друга променлива от същия тип ще съхраним и търсеното име.

За да може да посочим и позицията на търсеното име в списъка ще използваме още една променлива от целочислен тип. Много е важно тя да бъде инициализирана със стойност, която е извън диапазона на индексите на масива с имената, или да не е в интервала [0,4]. В практиката е наложено стойността за тази променлива да е -1;

На следваща стъпка трябва да обходим масива с имената и на всяка итерация да проверяваме за съвпадение на иманата. За целта може да използваме оператор за сравняване '==' или метода Equals(string, string).

Ако попаднем на такова съвпадение променяме стойността на променливата с текущия индекс.

Накрая трябва да правим проверка, дали променливата с индекса е различна от -1. Ако е така значи името е намерено в списъка.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        static void Main(string[] args)
        {

            string[] names = new string[5];
            string searched = "";

            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Въведете име ");
                names[i] = Console.ReadLine();
            }
            Console.WriteLine("Въведете търсеното име ");
            searched = Console.ReadLine();
            int index = -1;
```

```
            for (int i = 0; i < 5; i++)
            {
                if (names[i] == searched)
                    index = i;
            }
            if (index!=-1)
            {
Console.WriteLine("Името е намерено на позиция {0}", index+1);
            }
            else
            {
                Console.WriteLine("Името не е намерено.");
            }

        }

    }
}
```

7.4. Задачи за самостоятелна работа

Задача 1. Напишете програма, която декларира и инициализира 4 променливи от символен тип. Декларацията на всяка променлива да бъде в различна форма на стойността при инициализацията. Указание: вижте Задача 1.

Задача 2. Напишете програма, която въвежда от клавиатурата стойностите на 4 променливи от символен тип. В програмата реализирайте различни аритметични операции, като използвате символните променливи.

Задача 3. Напишете програма, която прочита символен низ от клавиатурата, обръща го отзад напред и го извежда на екрана. Например: "Потоп".

Задача 4. Напишете програма, която брои колко пъти даден подниз се съдържа в текст. Например нека търсим подниза "по" в текста: „По дрехите го посрещат, по ума го изпращат“.

Задача 5. Напишете програма, която реализира заместване на сбъркана дума в текст.

8. Алгоритми с масиви

8.1. Сортиране

Сортирането е подреждане на елементите на масив по определен признак. Обикновено се подреждат (сортират) числа по големина в нарастващ или намаляващ ред. Друг често срещан случай са списъци с имена. Ако ги съхраняваме в един масив от низове те също могат лесно да се сортират. Когато масивът е съставен от низове (имена) имената се подреждат по азбучен ред.

Съществуват различни начини на сортиране. За целите на обучението най-често се използва метода на пряката размяна по популярен под името "Метод на мехурчето" .

8.1.1. Метод на „мехурчето“

При този метод се сравняват стойностите на два съседни елемента като последователно се обхожда масива. Ако сортираме във възходящ ред и стойността на текущия елемент е по-голяма от стойността на следващия елемент то те си разменят местата.

Ето как се реализира метода на C#

```
int[] arr = { 80, 11, 50, 77, 64, 35, 24, 9 };

int temp = 0;
//обхождане
for (int write = 0; write < arr.Length; write++)
 {
   for (int sort = 0; sort < arr.Length - 1 - write; sort++)
    {
     if (arr[sort] > arr[sort + 1]) //сравняване на съседни елементи
      { // размяна
       temp = arr[sort + 1];
       arr[sort + 1] = arr[sort];
       arr[sort] = temp;
     }
   }
 }

   for (int i = 0; i < arr.Length; i++)
      Console.Write(arr[i] + " ");
```

8.1.2. Метод за сортиране "Пряк избор"

Ако сортираме във възходящ ред масивът се преглежда N-1 пъти, при първото обхождане елементът с най-малката стойност заема първата позиция в масива. При второто обхождане намираме отново най-малкия елемент, от вече останалите N-1 елемента и го поставяме на втора позиция. По този начин преглеждаме последователно N-2, N-3 ... 2 елементи, до достигане до най-големия елемент, който вече е сортиран и се намира на последната позиция в масива.

Ето как се реализира метода на C#

```
int[] arr = { 80, 11, 50, 77, 64, 35, 24, 9 };

            int pos_min, temp;

            for (int i = 0; i < arr.Length - 1; i++)
            {
                pos_min = i;// pos_min е текущия индекс

                for (int j = i + 1; j < arr.Length; j++)
                {
                    if (arr[j] < arr[pos_min])
                    {
                        // pos_min ще запази индекса на
                        // намерения по-малък елемент
                        pos_min = j;
                    }
                }

                    // размяна
                    temp = arr[i];
                    arr[i] = arr[pos_min];
                    arr[pos_min] = temp;

            }

            for (int i = 0; i < arr.Length; i++)
                Console.Write(arr[i] + " ");
```

8.1.3. Метод за сортиране чрез вмъкване

Чрез сравняващо сортиране сортираният масив се допълва с по един елемент всеки път. Масивът с елементи, които ще бъдат сортирани се разделя на две части: частта със сортираните елементи и частта с несортираните.

При всяка стъпка се взема първия елемент от несортирания масив и се вмъква на правилната позиция в сортираната част от масива. Сортирането продължава докато елементите от несортираната част на масива се изчерпят.

Ето как се реализира метода на C#

```
int[] arr = { 80, 11, 50, 77, 64, 35, 24, 9 };

            for (int i = 0; i < arr.Length - 1; i++)
            {
                int j = i + 1;

                while (j > 0)
                {
                    if (arr[j - 1] > arr[j])
                    {
                        int temp = arr[j - 1];
                        arr[j - 1] = arr[j];
```

```
                        arr[j] = temp;

                    }
                    j--;
                }
            }

            for (int i = 0; i < arr.Length; i++)
                    Console.Write(arr[i] + " ");
```

8.2. Търсене

Търсенето представлява проверка дали даден елемент е част от дадено множество. При масивите търсенето се изразява в проверката дали зададена стойност съвпада с елемент на масив от същия тип както зададената стойност.

Важно е да се уточни, че тук не търсим броя на съвпаденията, а дали има такова съвпадение. Тогава търсене на едно съвпадение, ще има за резултат позицията(индекса) на съвпадението в масива.

8.2.1. Последователно (линейно) търсене

Алгоритъмът се състои в последователно сравняване на всички елементи със зададената стойност за търсене. Ако се намери такова съвпадение, като резултат се връща позицията, ако се стигне до края на масива и не е намерено съвпадение се извежда съобщение или ако се реализира, чрез метод - някаква стойност, която да е извън множеството на допустимите позиции.

Ето една примерна реализация на алгоритъма на C# върху масив от 20 елемента.

```
Console.WriteLine("Въведете търсения елемент \n");
string se = Console.ReadLine();
 int y = Int32.Parse(se);
 for (int i = 0; i < 20; i++)
 {
 if (a[i] == y)
 {
Console.WriteLine("Елем.{0}е намерен на поз.{1}\n", y, i + 1);
Console.ReadLine();
  }
  }
 Console.WriteLine("Елементът не е намерен");
```

8.2.2. Двоично (бинарно) търсене

Характеризира се с бързина в сравнение с последователното търсене. Изисква елементите на масива да са предварително сортирани.

Алгоритъмът представлява обхождане на масива, като се започва със сравнение на елемента от масива, който е в средата. Ако средният елемент на масива съвпада с търсения то резултата е позицията в масива, където е съвпадението. В противен случай, определяме дали търсения елемент е по-малък или по-голям от средния елемент и продължаваме да търсим по същия

начин съответно в половината с по-малките или по-големите елементи от него. На всяка итерация се елиминира половина от оставащите елементи за сравнение.

Ето една примерна реализация на алгоритъма, чрез итеративен метод на C#

```
int target = 11;

int[] mynumbers = { 10, 12, 23, 24, 45 };
int mid=0, first = 0, last = mynumbers.Length - 1;
bool found = false;
while (!found && first <= last)
{
    mid = (first + last) / 2;
    if (target == mynumbers[mid])
        found = true;
    else
    {
        if (target > mynumbers[mid])
        {
            first = mid + 1;
        }
        if (target < mynumbers[mid])
        {
            last = mid - 1;
        }
    }
 }
  if(found==true)
   {
    // намерен е, индекса е mid
   }
  else
  {
    // не е намерен
  }
```

8.1. Задачи за самостоятелна работа

Задача 1. Напишете програма, която прочита от клавиатурата елементите на целочислен масив с размерност 50 и извежда на екрана сортиран списък на въведените елементи. Сортирането да се извърши по метода на мехурчето.

Задача 2. Напишете програма, която прочита от клавиатурата елементите на масив от реални числа с размерност 40 и извежда на екрана сортиран списък на въведените елементи. Сортирането да се извърши по метода на прекия избор.

Задача 3. Напишете програма, която прочита от клавиатурата елементите на масив от символи с размерност 30 и извежда на екрана сортиран списък на въведените елементи. Сортирането да се извърши по метода на сортиране, чрез вмъкване.

Задача 4. Напишете програма, която прочита от клавиатурата елементите на масив цели числа с размерност 40 и стойност за търсене(също цяло число). Програмата да извежда на екрана текстово съобщение, дали стойността за търсене се съдържа в масива или не. Търсенето да се реализира като последователно.

Задача 5. Напишете програма, която прочита от клавиатурата елементите на масив реални числа с размерност 40 и стойност за търсене(също реално число). Програмата да извежда на екрана индекса на съвпадение в масива. Търсенето да се реализира като двоично.

9. Подпрограми

Когато пишем дадена програма, целта ни е с нея да решим конкретна задача. За да го направим ефективно и да улесним работата си разделяме поставената ни задача на подзадачи, реализираме решения на тези подзадачи и накрая ги обединяваме в една програма. Тези подзадачи се наричат подпрограми.

Същност:

Подпрограмите са средство за отделяне и идентифициране на подалгоритъм от даден алгоритъм при програмно описание на този алгоритъм.

Предназначение:

- **групиране на съвкупност от действия**, които са част от даден алгоритъм, обикновено логически самостоятелна част (подалгоритъм). В повечето случаи на тази съвкупност от действия се съпоставя име – **име на подпрограмата;**
- **указване изпълнението** на цялата съвкупност от действия само чрез съпоставеното име.

Видове:

В някои езици за програмиране подпрограмите са познати под наименованията функции (functions) или процедури (procedures).

Функциите служат за описание на подалгоритми, чието множество от изходни данни се състои от един елемент, който се нарича *резултат на функцията.*

Процедурите служат за описание на произволен подалгоритъм.

Предимства:

Използването на подпрограми дава следните основни предимства:

- Лаконичност при програмна реализация на алгоритмите, тъй като там където е необходимо да се изпълнява цялата група от действия, е достатъчно да се укаже само името на групиращата подпрограма, без да се налага тази последователност от действия отново да се задава;
- Повишаване надеждността на програмите, тъй като алгоритъма на задачата се разбива на малки части (подпрограми), които самостоятелно се проектират, програмират, тестват и настройват и по този начин вероятността да се допусне грешка е значително по-малка;
- Многократно използване на едни и същи подпрограми като елементи при разработване на различни програми. Това от своя страна отново води до повишаване надеждността на програмите, в следствие на възможността за използване на предварително готови и проверени програмни части.

Подпрограми и C#

В C# се наричат методи (methods).

В началото нека направим и едно важно уточнение. В езика C# не съществуват функции или процедури, които познаваме от други езици за програмираме а се използват методи на класове, това е така защото езика в

същността си е Обектно ориентиран и всяка програма съдържа поне един клас, в който дефинираме данните и обработката им. За да изясним по разбираем начин идеите, които стоят зад методите, но не е изучаван тип данни Клас тук ще използваме понятието функция – основна единица за обобщение на обработката на данни в повечето езици за програмиране.

Функцията е подпрограма, която може да бъде извикана от друга функция или от главната функция/метод main за извършване на определена задача. Например, когато имаме код, който се повтаря на много места във програмата, би било добре ако го оформим във функция и да извикваме нея, когато ни потрябва вместо да копираме кода отново и отново.

Функциите се характеризират с това, че извършват определена обработка на данни, които обикновено са им предадени като параметри и след като завършат тази обработка връщат(но не винаги) като резултат обработените данни или една стойност.

В едно конзолно приложение обикновено има следните категории функции:

- За въвеждане на данни;
- За извеждане на данни;
- За всякакви изчисления, в зависимост от решаваната задача;
- За сортиране на данни;
- За търсене.

9.1. Декларация

Може да декларираме функции по същия начин както декларираме и главната функция, която познаваме още от първата ни програма.

Синтаксис:

```
[public] [static] <тип на върнатия резултат> <име>([списък с формални параметри])
        {
        // тяло на функцията
        }
```

На този етап [public] [static] ще ги пропуснем в разясняването и ще бъдат дискутирани в Глава 10.

<тип на върнатия резултат> може да бъде всеки от допустимите типове в езика.

<име> се определя от програмиста и трябва да бъде така подбрано, че недвусмислено да говори за предназначението на функцията.

Добре е, когато се избира името на функцията, да се следват препоръки, като:

- Името да започва с главна буква.
- Трябва да се прилага т. нар. правило PascalCase, т.е. всяка нова дума, която се добавя като част от името, трябва да започва с главна буква.
- Имената е препоръчително да бъдат съставени от глагол или от глагол и съществително име.

Важно е да отбележим, че тези правила не са задължителни, а препоръчителни и всеки програмист може да именува функциите според собствения си стил.

Тялото на функцията съдържа всички оператори, които решават задачата.

Ето един прост пример.

```
public void DoSomething()
{
    Console.WriteLine("Извършване на някакви действия.");
}
```

В примера използваме тип void (на върнат резултат), защото в случая не искаме от функцията да връща стойност, а само да изведе на екрана дадено съобщение. По същата причина тази функция не притежава списък с параметри.

В следващия пример ще декларираме функция, която изчислява и връща като резултат сбора(цяло число) на две цели числа. Тук параметрите са два – number1 и number2 цели числа, които трябва да се съберат. Когато функцията връща резултат най-често в края на тялото ѝ се добавя ключовата дума **return** и стойността, която да се върне.

```
public int AddNumbers(int number1, int number2)
{
    int result = number1 + number2;
    return result;
}
```

Локални променливи

Когато декларираме променлива в тялото на един метод, тя се нарича локална променлива (local variable) за метода.

Областта, в която съществува и е достъпна една локална променлива, започва от реда, на който сме я декларирали и стига до затварящата фигурна скоба "}" на тялото на метода. Тази област се нарича **област на видимост** на променливата. Ако направим опит да декларираме друга променлива със същото име в рамките на един метод, ще получим грешка при компилация.

Обърнете внимание на променливата result в горния пример, тя именно е локална за тази функция. Помислете и дайте отговор, дали променливите number1 и number2 са също локални.

9.2. Параметри

След като можем да отделяме различни части от алгоритъма в подпрограми за да разделим о обработката на данните трябва да можем и да използваме определен механизъм за взаимодействие между различните части на програмите. Този механизъм наричаме механизъм на предаване на параметри.

Видове взаимодействие за определяне на входните и изходните данни при подпрограмите:

1. с външната среда (потребителя и периферните устройства за вход и изход)

2. с вътрешната среда (с други части на програмата)

Средство осигуряващо втория вид взаимодействие: "механизъм на предаване на параметри"

Механизмът на предаване на параметри осигурява многократното използване на подпрограмите (свойството масовост), давайки възможност

подпрограмите да се изпълняват за различни множества от входни данни и да се получават различни изходни резултати.

Същността на механизма на предаване на параметри се състои в това, че задаването на множествата от входни и изходни данни на подпрограмите се разделя на два етапа:

1) При описание на съвкупността от действия (деклариране и дефиниране), които подпрограмата групира се използват т.нар. **формални параметри** – служат за определяне на множеството от входни и изходни данни на подпрограмата само в най-общ вид (брой и тип), като им съпоставят формални (условни) имена (без да определят конкретните им стойности)

Поради специфичното предназначение на функциите, при тях обикновено множеството от изходните данни се състои от един елемент и неговият тип определя типа на резултата на самата функция. Така че при функциите списъкът формални параметри в общия случай определя само множеството от входни данни

2) При активирането (изпълнение) на подпрограмата се използват т.нар. **фактически параметри**, които задават конкретните стойности на входните и изходните данни

Възможността формалните параметри на подпрограмите да се заместват с различни фактически параметри при всяко извикване на подпрограмата се нарича механизъм на предаване на параметри (аналогично на дефиниране и намиране на стойност на функция в математиката)

Видове формални параметри:

- **Формални параметри, които се предават по адрес** – това са параметрите от референтен тип (например масиви)**.** Те служат главно за деклариране на *изходни* или *входно-изходни данни*, въпреки че могат да се използват и за деклариране на входни данни;

- **Формални параметри, които се предава по стойност** – това са параметрите, от всички останали типове. Те служат за деклариране само на *входни данни.*

Примери:

Параметри по стойност:

```
public float addf(float a, float b)
     { float c=a+b; a = a + 10;    return c; }
```

Параметри по референция(адрес):

```
(1)
static void UpdateArray(int[] array)
      {
        array [0] = 5;
      Console.Write("UpdateArray() има параметър масив ");

      }
(2)
static void Method(out int i)
```

```
        {
            i = 22;
        }
```

Сравнение между видовете формалните параметри

- Формални параметри, които се предават по адрес (ФПА)
- Формални параметри, които се предава по стойност (ФПС)

1. Синтактичните особености:

а) ФПА се декларират в C# с ключови думи *out* и *ref*

б) Фактическите параметри съответни на ФПС могат да бъдат произволни изрази, докато тези съответни на ФПА – само променливи от съвместим тип

2. Семантични особености:

а) ФПА могат да служат като изходни и като входно-изходни, докато ФПС служат само за задаване на входни данни

б) Измененията, които ф-ята извършва върху ФПС не се отразяват върху съответните им фактически параметри, докато при ФПА се отразяват

3. Начина на реализацията им в езика C#:

а) За фактическите параметри съответни на ФПС при извикване на ф-ята се заделя памет в стека. Измененията, които ф-ята извършва с техните стойности се записват в тази памет. След приключване на изпълнението, тази памет се освобождава.

б) За фактическите параметри съответни на ФПА в стековата памет се записва само адреса на съответния фактически параметър. Ето защо измененията, които ф-ята извършва с техните стойности се записват в първоначално заделената за тях памет.

9.3. Извикване на функция

За да накараме функцията да извърши операциите в тялото ѝ трябва тя да бъде активирана/извикана и да и се прехвърлят(предадат) реални данни за обработка.

Осъществява се с Оператор за извикване на функция.

Синтаксис:

<име>(<списък фактически параметри> незад)

- <име> е име на предварително декларирана метод
- <списък фактически параметри> е последователност от изрази (фактическите параметри, т.е. тези които се подават на метода), разделени с "," и трябва донякъде да съответстват по реда си, по броя и типа на формалните параметри от описанието
- ако съответните формални параметри имат стойности по подразбиране, които съвпадат с тези на фактическите, то последните могат да се пропуснат, но само ако са в края на списъка

Семантика:

Извикването на функцията има типа на резултата на функцията и предизвиква неговото изпълнение, като конкретните стойности на входните и

изходните данни се определят от зададените при това извикване фактически параметри

Пример:

```
int result = AddNumbers(10, 5);// Функцията се извиква с
                               // фактически параметри 10 и 5
Console.WriteLine(result);
```

9.4. Рекурсивни функции

Рекурсия

Произход на думата “рекурсия” (recursion):
На английски:recur – връщам се, повтарям се, случвам се отново (на повтарящи се интервали)

Има латински произход: (re = обратно) + (currere = бягам)

Примери на рекурсивни явления:
ако в телевизионно студио камерата снима изображението на телевизор, който излъчва същото предаване, то на екрана на домашния телевизор се вижда картина, която се съдържа в себе си отново и отново ако някакъв предмет е поставен между две огледала, намиращи се едно срещу друго, то и в двете огледала отражението представлява рекурсивно изображение.

Един обект се нарича рекурсивен, ако се съдържа в себе си или е дефиниран чрез себе си.

Рекурсията широко се използва в математиката за дефиниране на математически понятия или функции:

А) Рекурсивна дефиниция на понятието **естествено число**:

1. 1 е естествено число
2. Ако N е естествено число, то N+1 е естествено число

Б) Рекурсивна дефиниция на **функцията за** N! (N факториел):

1. 0!=1
2. N!=(N-1)!N, $\forall N > 0$

В) Рекурсивна дефиниция на **редицата от числата на Фибоначи** f_0, f_1, …, f_{N-1}, f_N, f_{N+1},…:

1. $f_0 = 0$, $f_1 = 1$
2. $f_{N+1} = f_N + f_{N-1}$, $\forall N > 0$

Пример за първите няколко ЧФ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,...

Рекурсията в информатиката

В информатиката рекурсията се използва за:

- описание на данни – **рекурсивни структури от данни** т.е. структури, които се съдържат в себе си:

  Рекурсивен тип данни T е съставен тип, който съдържа компоненти от типа T (напр. масив от масиви, списък, стек и т.н.)

- за описание на обработката на данни – **рекурсивни алгоритми**, т.е. алгоритми, които се описват чрез себе си

Тук се разглеждат само възможностите за компютърно моделиране на рекурсивни алгоритми.

Рекурсивни функции

Средството в ЕП, което осигурява възможност за представяне на рекурсивни алгоритми са подпрограмите.

Определение:

Рекурсивна подпрограма(функция) е такава подпрограма, която директно или косвено извиква себе си.

Пример:

```
long fact( int n )
        {
                if ( n == 0 )
                        return 1;

                else
                        return ( n * fact( n - 1 ) );
        }
```

Семантика:

При всяко рекурсивно обръщение в програмния стек се разпределя памет за параметрите и локалните променливи. Използването на рекурсия води до увеличаване бързината на съставяне на програми, обикновено рекурсивните алгоритми са по-кратки .

Но рекурсията не води до ефективни програми, защото при нея се използва повече памет и е необходимо повече време за изпълнение на програмата.

Приложение

Рекурсията е техника за програмиране, чиято употреба води до по-ефективни решения на определени задачи. Примери за такива задачи са определяне на числата на Фибоначи, изчисляване на факториела, двоично търсене и др. Понякога нейното използване може да опрости значително кода и да подобри четимостта му, но в други случаи итеративните решения може да бъдат по-удачни.

За илюстрация нека се върнем пак на редицата от числата на Фибоначи.

Всеки член на редицата се получава като сума на предходните два. Сами може да се убедите за по-високата ефективност от следващите примери за рекурсивна и нерекурсивни функции за изчисляване на N-ия елемент от редицата от числата на Фибоначи.

Нерекурсивно решение

```
     public static int Fibonacci(int n)
         {
          int a = 0;
          int b = 1;

          for (int i = 0; i < n; i++)
          {
              int temp = a;
              a = b;
              b = temp + b;
          }
```

```
          return a;
        }
```

Рекурсивно решение

```
static long Fib(int n)

{

       if (n <= 2)

       {

              return 1;

       }

       return Fib(n - 1) + Fib(n - 2);

}
```

Съставяне на рекурсивни алгоритми

Основните стъпки при съставяне на рекурсивни алгоритми са:

1. Определят се параметрите от които зависи задачата (параметризация);
2. Извършва се тъй наречения рекурсивен анализ, т.е. задачата се разбива на подзадачи, като всяка подзадача се описва чрез един или повече случаи на същата задача, но за други (обикновено по-малки) стойности на параметрите (редукция);
3. Осигурява се край на рекурсията, т.е. намира се такава подзадача, чието решение не е рекурсивно т.е. води до завършване на алгоритъма. Определят се стойностите на параметрите на алгоритъма, на които тази подзадача съответства (условие за край).

9.5. Задачи за упражнение

Задача 1. Да се напише програма, която въвежда стойности на естествените числа a, b, c и d и намира и извежда най-големият общ делител(НОД) на числата a и b, след това на c и d и накрая на a, b, c и d.

За решаване на задачата е необходимо да се декларира функция, която има два параметъра цели числа и връща като резултат цяло число.

```
int gcd(int x, int y)
        {
          // тяло изчислява се НОД
        }
```

За намиране на НОД ще използваме алгоритъм, чрез изваждане.

В решението ще използваме и друг начин за въвеждане от клавиатурата и конвертиране на въведените низове до числа, чрез метода:

```
int.TryParse(string s,   out int result)
```

Методът има два параметъра – символен низ и върнатия резултат.

Обърнете внимание на ключовата дума `out`. Използването и означава че параметъра ще се предава по адрес, следователно може да го променим.

Ето пример

```
  string sa=Console.ReadLine();
  int a; // променливата има стойност по подразбиране
  int.TryParse(sa, out a); // тук стойността на a се променя
```

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        static int gcd(int x, int y)
        {
            while (x != y)
            {
                if (x > y)
                {
                    x = x - y;
                }
                else
                {
                    y = y - x;
                }
            }
            return x;
        }

        static void Main(string[] args)
        {


            Console.Write("a, b, c, d= ");
 string sa=Console.ReadLine(),    sb=Console.ReadLine(),
            sc=Console.ReadLine(), sd=Console.ReadLine();

            int a, b, c, d;
            int.TryParse(sa, out a);
            int.TryParse(sb, out b);
            int.TryParse(sc, out c);
            int.TryParse(sd, out d);
```

```
            int r = gcd(a, b);
            Console.Write("gcd{");
            Console.Write(a);
            Console.Write(", ");
            Console.Write(b);
            Console.Write("}= ");
            Console.Write(r);
            Console.Write("\n");
            int s = gcd(c, d);
            Console.Write("gcd{");
            Console.Write(c);
            Console.Write(", ");
            Console.Write(d);
            Console.Write("}= ");
            Console.Write(s);
            Console.Write("\n");
            Console.Write("gcd{");
            Console.Write(a);
            Console.Write(", ");
            Console.Write(b);
            Console.Write(", ");
            Console.Write(c);
            Console.Write(", ");
            Console.Write(d);
            Console.Write("}= ");
            Console.Write(gcd(r, s));
            Console.Write("\n");



        }
    }
}
```

Задача 2. Да се напише програма, която реализира функция за отпечатване на символ определен брой пъти.

Например:

C

CC

CCC

CCCC

Упътване: за решаването на задачата реализирайте функция с два параметъра: символ за отпечатване и брой пъти за отпечатване.

```
static void Print_Char(char display_char, int count)
```

Функцията ще извикваме в тялото на цикъл с променен втори параметър. Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
     static void Main(string[] args)
     {
     int i,number_of_rows;
     char ch;
     Console.Write( "Въведете символ, който да се изведе на
екрана: ");
    ch = Console.ReadKey().KeyChar;
    Console.Write(System.Environment.NewLine);
     Console.Write( "Въведете брой редове: ");
    number_of_rows = Convert.ToInt32(Console.ReadLine());

     for (i = 1; i <= number_of_rows; ++i)
     {
     Print_Char(ch, i);

     }
     }
     static void Print_Char(char display_char, int count)
     {
     int j;
     for (j = 1; j <= count; ++j)
     Console.Write( display_char);
    Console.Write(System.Environment.NewLine);
     return;
     }

    }
}
```

Задача 3. Да се напише програма, която въвежда стойности на променливи a, b, c и d, след което разменя стойностите на a и b и на c и d съответно.

За решаването на задачата ще реализираме функция с параметри, които се предават по адрес.

За целта използваме ключовата дума `ref` при декларацията и при извикването на функцията.

```
static void swaptwo(ref int  x, ref int y)
```

и

```
swaptwo(ref a, ref b);
```

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
     static void swaptwo(ref int  x, ref int y)
     {
     int work = x;
     x = y;
     y = work;

     }
    static void Main(string[] args)
    {
        Console.Write("a= ");
        int a = Convert.ToInt32(Console.ReadLine());
        Console.Write("b= ");
        int b = Convert.ToInt32(Console.ReadLine());
        Console.Write("c= ");
        int c = Convert.ToInt32(Console.ReadLine());
        Console.Write("d= ");
        int d = Convert.ToInt32(Console.ReadLine());


        swaptwo(ref a, ref b);
        swaptwo(ref c, ref d);
 Console.WriteLine("A=" + a + "B=" + b + "C=" + c + "D=" + d);
    }

    }
}
```

Задача 4. Да се напише програма, която въвежда елементи на масив от клавиатурата и с функция проверява дали дадено число x се съдържа в масива.

```
using System;

namespace Console
{
    class Program
    {
     static int broj (int [] a, int n, int x)
     { int br = 0;
     for (int i=0; i<n; i++)
     { if (a[i] == x) br++; }
     return br;
     }
```

```
    static void Main(string[] args)
     {
     int [] arr=new  int[20];
     int x, br;
     for (int i=0; i<20; i++)
     { Console.WriteLine( "Елемент ["+ (i + 1) +"] = ");
     arr[i]=Convert.ToInt32(Console.ReadLine()); }
     Console.Write( "Стойност за търсене X= ");
        x=Convert.ToInt32(Console.ReadLine());
     br = broj (arr,20,x);
     if (br == 0) Console.WriteLine( "X не е намерена.");
     else Console.WriteLine( "Намерена "+ br + " пъти. ");
     }
    }
}
```

*Задача 5. Да се напише рекурсивна програма за намиране на факториела m! (m е дадено естествено число). В тази програма е описана рекурсивната функция fact, която приложена към естествено число, връща факториела на това число. Стойността на функцията се определя посредством обръщение към самата функция в оператора return n * Factorial(n-1);.*

```
using System;

namespace Console
{
    class Program
    {
        static long Factorial(int n)
        {
            if (n == 0)
                return 1;
            return n * Factorial(n - 1);
        }
    static void Main(string[] args)
     {

     int x;

     Console.Write( "Число X= ");
        x=Convert.ToInt32(Console.ReadLine());
      Console.WriteLine( "Факториелът на 0} е
{1}",x,Factorial(x));

     }
    }
}
```

Задача 6. Като се използва рекурсивната дефиниция на функцията за степенуване да се напише програма, която по дадени a реално и k – цяло число, намира A на степен k.

```
using System;

namespace Console
{
    class Program
    {
       static double pow(double x, int n)
        {
            if (n == 0) return 1;
            if (n > 0 ) return x * pow(x, n - 1);
            return 1.0 / pow(x, -n);
        }
    static void Main(string[] args)
     {

     double a;
     Console.Write( "A= ");
    a=Convert.ToDouble(Console.ReadLine());
    int k;
    Console.Write("k= ");
    k = Convert.ToInt32(Console.ReadLine());
Console.WriteLine( "{0} на степен {1} е {2}",a,k,pow(a,k));
          }
    }
}
```

9.6. Задачи за самостоятелна работа

Задача 1. Да се напише програма, която включва функции за:

А) Въвеждане от клавиатурата на елементите на целочислен масив;

Б) Намира и връща сумата на елементите на масива.

Задача 2. Да се напише програма, която включва функции за:

А) Въвеждане от клавиатурата на елементите на целочислен масив;

Б) Намира и връща средната стойност на елементите на масива.

Задача 3. Да се напише програма, която включва функция, която намира колко пъти дадено число се среща в даден масив. Да се напише и функция за въвеждане на 30 елемента на масив и търсеното число.

Задача 4. Да се напише програма, която включва функция за намиране на най-големия елемент на масив.

Задача 5. Да се напише програма, която включва функции за:

А) Въвеждане от клавиатурата на елементите на двумерен масив от реални числа;

Б) Извежда на екрана сумата на елементите на всеки ред на масива.

10. Структури

10.1. Структура от данни запис

Структурата (запис, structure, record) е съставен тип, представляващ подредена крайна съвкупност от елементи, които могат да се различават по своя тип. До компонентите е осигурен директен достъп.

Основни понятия и характеристики:

Компонентите се наричат полета или членове (fields, members) на структурата

Записът е един от малкото комбинирани АТД (т.е. чиито компоненти от своя страна могат да са от различни АТД)

В различните ЕП структурата се реализира или като статичен тип, или като динамичен, но в определени граници.

Приложение

За моделирането на обекти, които имат много на брой характеристики (и за обръщение към този обект само с едно име) – служи за групиране на смислово свързани данни, които в общия случай представляват характеристики на едно понятие.

Пример:

Във фирмите служителите се описват от няколко признака (например име, адрес, социално-осигурителен номер, заплата), някои от които на свой ред могат съдържат също няколко признака (например адресът се състои от улица, номер, град, окръг и пощенски код).

Достъп

Директният достъп до компонентите се осъществява чрез съпоставените им уникални имена, а не чрез позицията им (както е при масивите)

Конструиране

Стойността на структурата може да се конструира по два начина:

чрез последователно изброяване на стойностите на всички полета на записа, като се ограждат в някакъв вид “скоби”, чрез достъп до всички полета на записа за задаване на стойностите им.

В паметта на компютъра всяка величина от тип запис заема толкова място, колкото е необходимо, за да бъдат съхранени всички нейни компоненти.

C# структура

В C# структурите служат не само като модел на запис на различни множества от стойности, те всъщност са специален вид класове. Все пак, за да се прави разлика тук ще бъде разгледано само първото им предназначение.

10.2. Декларация

Синтаксис:

За деклариране на тип структура и променливи от този тип използваме следния синтаксис:

```
struct  <име>незад   { <декл. на полета>незад}  ;
```

, където:

<име> е името на типа-структура

<декл. на полета> е списък от типове и имена на променливи, разделени с “;”, които представляват полетата на структурата (членове-данни). Полетата

могат да бъдат от всеки тип освен void, непълен тип (затова не могат да бъдат от типа на структурата, която се декларира в момента) или функция

Примери:

(1)

```
struct Child {
      public double height;
      public double weight;
      public int years;
      public int months;
      public char gender;
} ;
```

(2)

```
struct Date {  //дата
           public int Day;
           public int Mon;
           public int Year;
     };
     enum  Merka{Kg, M, L, Br} ;
struct Stoka { //стока
       public string Ime;
       public Merka Mer;
       public double EdCena;
       public Date GodnaDo; //вложен запис
     } ;
```

Множеството от стойности

Множеството от стойности на структурата е съвкупността от всички възможни комбинации от стойности на неговите полета. Стойността на всяка променлива от даден тип структура съдържа всички декларирани полета.

Достъп до поле

Операторът за достъп до полета на структура има следната форма:

```
<име_на_структура>.<име_на_поле>
```

За демонстрация нека разгледаме следната задача.

Задача 1. Да се напише програма, която инициализира заглавие, автор, кратко описание и идентификатор на една книга и извежда на екрана цялата информация за книгата.

За решаване на задачата наред с декларацията на структурата, която трябва да съдържа четири полета, три символни низа и едно цяло число. Следва декларация на променлива от тип структура.

Декларацията не се различава, като синтаксис с декларация на променлива от другите типове.

```
<име_на_структура>  <име_на_променлива>;
```

Трябва да използваме и оператор за достъп до поле за да инициализираме всяко едно поле.

Ето и решението.

```
struct Book
{
   public string title;
   public string author;
   public string subject;
   public int book_id;
};

public class testStructure
{
   public static void Main(string[] args)
   {
      Book book1;
      book1.title = "C# Programming";
      book1.author = "John Petersen";
      book1.subject = "C# Programming Tutorial";
      book1.book_id = 6495407;
      Console.WriteLine( "Заглавие : {0}", book1.title);
      Console.WriteLine("Автор : {0}", book1.author);
      Console.WriteLine("Описание : {0}", book1.subject);
   Console.WriteLine("Идентификатор :{0}", book1.book_id);
   }
}
```

10.3. Операции

Въвеждане и извеждане: структурите могат да се въвеждат и извеждат само поелементно.

Присвояване – на променлива от тип структура може да бъде присвоена променлива от същия тип структура:

```
Point p1, p2;
p1.x=3;
p1.y=2;
p2=p1; /*копира полетата на p1 в p2*/
```

, което е еквивалентно на:

```
p2.x=p1.x;
p2.y=p1.y;
```

Всички останали операции със структури се извършват поелементно (могат да се предефинират от програмиста).

Релации със структури не са дефинирани стандартно (програмистът може сам да предефинира избрана релация за някой тип структура)

Структурите могат да се подават като параметри на функции и да бъдат техен резултат.

Особености на структурите в C#

В този език структурите се доближават до класовете и могат да съдържат и методи и свойства. Повече за методи и свойства ще представим в следващата глава.

Пример:

```
using System;
struct SimpleStruct
{
    private int xval;
    public int X
    {
        get
        {
             return xval;
       }
        set
        {
            if (value < 100)
                xval = value;
        }
    }
    public void DisplayX()
    {
    Console.WriteLine("Стойност: {0}", xval);
    }
}

class TestClass
{
    public static void Main()
    {
        SimpleStruct ss = new SimpleStruct();
        ss.X = 5;
        ss.DisplayX();
    }
}
```

10.4. Задачи за упражнение

Задача 1. Дефинирайте следните три структури: банков акаунт, телефонен номер и пълно име(трите имена).

Указание: припомнете си как се декларира структура.

```
public struct BankAccount
{
    public string Currency { get; set; }
    public double Amount { get; set; }
}

public struct PhoneNumber
{
    public int CountryCode { get; set; }
```

```
        public int RegionCode { get; set; }
        public int SubscriberNunber { get; set; }

    }

    public struct FullName
    {
        public string FirstName { get; set; }
        public string MiddleName { get; set; }
        public string LastName { get; set; }
    }
```

Задача 2. Напишете програма, която дефинира структура Кутия. Структурата съдържа следните полета: ширина, височина и дължина.

Инициализирайте полетата със стойности съответно 7, 5 и 6 см. Изведете на екрана обема на кутията.

Указание: вижте декларацията на структурата от тип Книга и използвайте оператор за достъп до поле.

Следващата програма решава задачата.

```
    using System;

    namespace Console
    {
        class Program
        {
            struct Box
            {
                public double length;
                public double width;
                public double height;

            };

            static void Main(string[] args)
            {


                Box Box1 ; // Декларация на променлива
                double volume = 0.0; // Обем на кутията

                // box 1 инициализация
                Box1.height = 5.0;
                Box1.length = 6.0;
                Box1.width = 7.0;
                // Изчисляване на обема
            volume = Box1.height * Box1.length * Box1.width;
```

```
        Console.WriteLine("Обема на кутията е : {0}", volume);


            }

        }
    }
```

Задача 2. Да се дефинира масив от 50 студента елементи от тип структура. Структурата да съдържа елементите име, факултетен номер, специалност и среден успех. Да се въведат данните в масива. Да се изведат за справка записаните в масива данни.

Указание: вижте декларацията на структурата от тип Книга и използвайте оператор за достъп до поле.

Ето и декларацията:

```
public struct Student
        {
            public string name;
            public string facNumber;
            public string specialnost;
            public double uspeh;
        };
```

На второ място е необходимо да се декларира масив от 50 елемента, където да съхраним данните за студентите. Особеното на този масив, е че трябва да бъде от тип структура.

Ето и декларацията:

```
Student[] students = new Student[50];
```

Остава само да обходим масива използвайки цикъл и да въвеждаме и извеждаме полетата.

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
        public struct Student
        {
            public string name;
            public string facNumber;
            public string specialnost;
            public double uspeh;
        };



        static void Main(string[] args)
```

```
            {


                Student[] students = new Student[50];
             Console.Write("Въведете брой студенти, n <=50 ");
                int n = int.Parse(Console.ReadLine());
                for (int i = 0; i < n; i++)
                {
                    Console.Write("Име :");
                    students[i].name = Console.ReadLine();
                    Console.Write("Факултетен номер :");
                   students[i].facNumber = Console.ReadLine();
                    Console.Write("Специалност :");
                 students[i].specialnost = Console.ReadLine();
                    Console.Write("Ср. успех :");
     students[i].uspeh = Convert.ToDouble(Console.ReadLine());
                }
            Console.Write("Вие сте въвели следните данни:");
                Console.Write("\n");
Console.Write("<Име\tФак. номер\tСпециалностт\tСр. успех>");
                Console.Write("\n");
                for (int i = 0; i < n; i++)
                {
                    Console.Write(students[i].name);
                    Console.Write('\t');
                    Console.Write(students[i].facNumber);
                    Console.Write('\t');
                    Console.Write(students[i].specialnost);
                    Console.Write('\t');
                    Console.Write(students[i].uspeh);
                    Console.Write("\n");
                    Console.Write("\n");

                }
            }
        }
    }
```

Задача 4. Да се напише програма за въвеждане от клавиатурата на информация за не повече от 100 книги. За всяка книга се въвежда информация за:

Име на книга;

Автор;

Година на издаване;

Цена;

Да се изведе на екрана списък с издадените книги след 2005 г.

За решението използвайте указанията от предходните задачи.

В допълнение трябва да се реализира и метод за въвеждане на информация за книгите. Методът следва да има параметър масив от тип *Book*

```
public static void input_book(Book[] f, int n)
```

Следващата програма решава задачата.

```
using System;

namespace Console
{
    class Program
    {
       public struct Book
        {
            public string title; // поле за име
            public string author; // поле за автор
            public int year; // поле за година
            public double price; // поле за цена

        };

        public static void input_book(Book[] f, int n)
        {
            for (int i = 0; i < n; i++)
            {

                Console.Write("Въведете заглавие: ");
                f[i].title = Console.ReadLine();
                Console.Write("Автор:");
                f[i].author = Console.ReadLine();
                Console.Write("Година на издаване :");
               f[i].year= Convert.ToInt32(Console.ReadLine());
                Console.Write("Цена:");
             f[i].price = Convert.ToDouble(Console.ReadLine());
            }
```

```
        }

        static void Main(string[] args)
        {
            Book[] b = new Book[100]; // масив от тип структура
            int i;
            int n;
            do
            {
                Console.Write("Въведете брой на книгите n=");
                n = int.Parse(Console.ReadLine());
            } while (n < 2 || n > 100);
            input_book(b, n);
 Console.Write(" Списък на книги издадени след 2005г.:");
            Console.Write("\n");
            for (i = 0; i < n; i++)
            {
                if (b[i].year > 2005)
                {
                    Console.Write(b[i].title);
                    Console.Write(" ");
                    Console.Write(b[i].author);
                    Console.Write(" ");
                    Console.Write(b[i].price);
                    Console.Write("\n");
                }

            }

        }
    }
}
```

10.5. Задачи за самостоятелна работа

Задача 1. Напишете програма, която дефинира структура Цилиндър. Структурата съдържа следните полета: радиус на основата и височина. Инициализирайте полетата със стойности въведени от клавиатурата. Изведете на екрана обема на цилиндъра.

Задача 2. Да се дефинира масив от 80 студента елементи от тип структура. Структурата да съдържа елементите име, факултетен номер, специалност и възраст. Да се въведат данните в масива. Да се изведат за справка записаните в масива данни за студенти по-млади от 25 години.

Задача 3. Да се напише програма за въвеждане от клавиатурата на информация за не повече от 100 музикални албума. За всяка албум се въвежда информация за:

Заглавие;

Изпълнител;

Година на издаване;

Цена;

Да се изведе на екрана списък с издадените албуми през 2015 г.

Задача 4. Да се напише програма за обработване на данни за 100 артикула в един магазин. За всеки артикул се въвеждат данни за: наименование, производител, цена и количество. Да се реализират: функция за въвеждане от клавиатурата на данните за артикулите, функция за извеждане на данните за артикулите, функция, която връща средната цена на артикулите.

Задача 5. Да се напише програма за обработване на данни за 50 автомобила в един автосалон. За всеки автомобил се въвеждат данни за: модел, производител, цена и количество. Да се реализират: функция за въвеждане от клавиатурата на данните за автомобилите, функция за извеждане на данните за артикулите, функция, която връща средната цена на автомобилите, функция за начисляване на отстъпка от 5% на всички автомобили и функция за извеждане на екрана на модела и производителя на автомобила с най-ниската цена.

11. Обектно ориентирано програмиране и C#

Тази глава, съдържа тематика, която не е част от курса Програмиране(C#), но може да се проучи от читателя, тъй като езика в същността си се базира на Обектно ориентираната парадигма за програмиране.

По един или друг начин някои неща, които използвахме в програмите досега може и да са били разбрани, но може и да са останали загадка. Така например, защо нашите програми винаги включват декларация като:

```
namespace Console
{
    class Program
    {
    }
}
```

Досега използвахме тези декларации без да обръщаме внимание на тях, но е добре да се запознаем с тях.

В тази глава ще направим един кратък преход с увод подходящ за дисциплината Обектно ориентирано програмиране (ООП), който ще ни даде една основа за продължаване на изучаването на съвременните методи на програмиране.

Ще въведем читателя в основните понятия и принципи на ООП, ще научим как се дефинират класове, техните полета и методи. Тематиката ще илюстрираме с няколко примера.

11.1. ООП

Целта на всяка една програма, която създаваме, е да реши даден проблем или да реализира някаква идея. За да предложим решение, ние първо създаваме опростен модел на реалността, който не отразява всички факти от нея, а се фокусира само върху тези, които имат значение за намирането на решение на нашата задача. След това, използвайки модела, намираме решение (т.е. създаваме алгоритъма) на нашия проблем и това решение го описваме чрез средствата на даден език за програмиране.

В днешно време най-често използваният тип езици за програмиране са обектно-ориентираните. И тъй като обектно-ориентираното програмиране е близко до начина на мислене на човека, то ни дава възможността с лекота да описваме модели на заобикалящата ни среда.

Една от причините за това е, че ООП ни предоставя средство, за описание на съвкупността от понятия, които описват обектите във всеки модел.

Това средство се нарича клас (class). Понятието клас и дефинирането на собствени класове, е вградена възможност на езиците от високо ниво използващи ОО подход.

Обектно-ориентираното програмиране моделира обектите от реалния свят със средствата на програмния език.

Обектите в ООП се характеризират с:

- атрибути (свойства)
- операции (възможни действия)

Основни принципи на ООП са:

- Капсулиране на данните
- Наследяване

- Полиморфизъм
- Абстракция

Обектно-ориентираното програмиране позволява преизползване на програмния код (code reuse).

11.2. Основни понятия

Клас – категория обекти с общи свойства и операции, които могат да се извършват върху тях.

Обект – единичен обект от даден клас, инстанция на клас.

Интерфейс – спецификация на съвкупност от действия, които даден обект предлага.

Свойство – видима за външния свят характеристика (properties) на даден обект.

Метод – операция, която всички обекти от даден клас могат да извършват.

Капсулиране на данните – събиране на данните за даден обект и операциите над тях в едно цяло (клас), като се ограничава директния достъп до тези данни и операции.

Абстракция на данните – възможността да работим с данни без да се интересуваме от тяхното вътрешно представяне, а само от операциите над тях (абстрактни класове).

Абстракция на действията – възможността да изпълняваме действия, за които не знаем точно как са реализирани (интерфейсите).

Наследяване – възможността един клас (наследник) да придобие свойства и действия от друг клас (родител).

Полиморфизъм – възможността да разглеждаме обекти от клас-наследник като обекти от базов клас, като класът наследник може да е предефинирал някои от действията на базовия клас.

11.3. Още за клас и обект

Клас (class) в ООП наричаме описание (спецификация) на даден клас обекти от реал-ността. Класът представлява шаблон, който описва видо-вете състояния и поведението на конкретните обекти (екземплярите), които биват създа-вани от този клас (шаблон).

Обект (object) наричаме екземпляр създаден по дефиницията (описанието) на даден клас. Когато един обект е създаден по описанието (шаблона), което един клас дефинира, казваме, че обектът е от тип "името на този клас".

Например, ако имаме клас Dog, описващ някакви характеристики на куче от реалния свят, казваме, че обектите, които са създадени по описанието на този клас (например кученцата "Шаро" и "Рекс") са от тип класа Dog.

Друг пример е, ако имаме клас Car, който описва някои от характеристиките на автомобил (марка, модел, цвят, изминати километри и регистрационен номер), може да създадем обекти (2) на класа автомобил. Например моят автомобил Марка: Дачия, Модел: Логан, Цвят: Бял, Изминати километри: 180х. и Рег.Номер: РВ 45 56, и автомобила на мой колега Марка: БМВ, Модел: 320, Цвят: Бял, Изминати километри: 50х. и Рег.Номер: РВ 12 34

Всеки клас съдържа дефиниция на това какви данни трябва да се съдържат в един обект, за да се опише състоянието му. Обектът (конкретния екземпляр от този клас) съдържа самите данни. Тези данни дефинират състоянието му (state).

В примера за автомобила програмиста решава, че това са: марка, модел, цвят, изминати километри и регистрационен номер.

Освен състоянието, в класа също се описва и поведението (behavior) на обектите. Поведението се изразява в действията, които могат да бъдат извършвани от обектите. Средството на ООП, чрез което можем да описваме поведе-нието на обектите от даден клас, е декларирането на методи в класа.

В примера за автомобила програмиста решава, че това са: ускорение, спиране, промяна на изминатите километри или цвета и други.

ООП и .Net

- В .NET Framework обектно-ориентираният подход е залегнал на най-дълбоко архитектурно ниво
- Всички .NET приложения са обектно-ориентирани
- Всички .NET езици са обектно-ориентирани
- Понятието клас от ООП има две реализации – класове и структури
- Няма множествено наследяване
- Класовете могат да имплементират няколко интерфейса едновременно

Namespace

Пространства от имена(Namespaces) се използват широко в C# програмите. Първо класовете използват пространства от имена за да обединят множество класове. На второ място може да декларираме собствени пространства от имена, за да ограничаваме достъпа до класовете и методите в големи проекти.

Декларация:

```
namespace <име>
{

}
```

За да използваме дадено пространство от имена използваме директивата **using**

```
using System;
```

В .NET **класовете** са класическа реализация на понятието клас от обектно-ориентираното програмиране и много приличат на класовете в C++ и Java.
Класовете имат членове (class members) :

- полета, константи, методи, свойства, индексатори, събития, оператори, конструктори, деструктори
- вложени типове (вложени класове, структури, интерфейси, делегати, ...)

Членовете имат видимост - public, private, protected, internal
Членовете могат да бъдат статични (общи) или за дадена инстанция.

11.4. Дефиниране на клас

Синтаксис:

```
<мод_за_видимост> class  <име_на_класа>
{
    // член променливи
    < мод_за_видимост > <тип данни> променлива 1;
     ...
    < мод_за_видимост > < тип данни > променлива N;

    // методи

 < мод_за_видимост > < тип данни > Метод1 (списък_параметри)
    {
        // тяло на метода
    }
    ...
   < мод_за_видимост > < тип данни > Метод1 N (списък_параметри)
    {
        // тяло на метода
    }
}
```

Пример:

Декларация на клас Кутия с метод за изчисляване на обема

```
class Box
    {
       public double length;
       public double width;
       public double height;
    }
    class Boxtest
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box(); // Декларация на обект кутия
             double volume = 0.0; // Обем на кутията

            // box 1 инициализация
            Box1.height = 5.0;
            Box1.length = 6.0;
            Box1. width = 7.0;
            // Изчисляване на обема
            volume = Box1.height * Box1.length * Box1. width;
            Console.WriteLine("Volume of Box1 : {0}",  volume);
         }
      }
```

11.5. Член променливи

Модификатори

Модификатор, наричаме ключова дума с помощта, на която даваме допълнителна информация на компилатора за кода, за който се отнася модификаторът.

В C# има четири модификатора за достъп. Те са public, private, protected и internal.

Ниво на достъп **public**

Използвайки модификатора public, се указва на компилатора, че елементът, пред който е поставен, може да бъде достъпен от всеки друг клас, независимо дали е от текущото пространство от имена или извън него.

Ниво на достъп **private**

Модификаторът private служи за указание, че елементът, за който се отнася, не може да бъде достъпван от никой друг клас освен от класа, в който е дефиниран. Това ниво на достъп се използва по подразбиране.

Ниво на достъп **internal**

Модификаторът internal се използва, за да се ограничи достъпът до елемента само от файлове от същото асембли.

Ниво на достъп **protected**

Модификаторът указва видимост само за класа и неговите наследници.

Полета (член-променливи)

Както вече стана дума, когато декларираме клас, описваме обект от реалния свят. За описанието на този обект, се фокусираме само върху тези от характеристиките му, които имат отношение към проблема, който ще решава нашата програма.

Тези характеристики на реалния обект ги имплементираме в деклараци-ята на класа, като декларираме набор от специален тип променливи, наре-чени полета (наричат се още атрибути). Атрибутите на класа се дефинират като собствени променливи в тялото му (наречени член-променливи), в които съхраняваме данните за отделните характеристики.

Когато създадем обект по описанието на нашия клас, стойностите на полетата ще съдържат кон-кретните характеристики, с които даден екзем-пляр (нарича се още инстанция) от класа (обект) се отличава от всички останали обекти от същия клас.

Характеристики:

- Член-променливите (полетата, атрибутите) съдържат данни за инстанцията на класа
- Могат да бъдат от произволен тип
- Могат да имат зададена видимост
- При деклариране може да им се задава стойност

Пример:

```
class Student
{
    private string FirstName;
    private string LastName;
    private string StudentID;
    private int Course = 1;
```

```
    private string Speciality;
    protected Course[] Courses;
    private string Remarks = " ";
}
```

Стойности по подразбиране на член-променливите

Тип на поле	Стойност по подразбиране
bool	false
byte	0
char	'\0'
decimal	0.0M
double	0.0D
float	0.0F
int	0
референция към обект	null

11.6. Методи на класове

Методите вече бяха разгледани, но нека припомним най-важното в контекста на разглеждане на класовете.

Метод (method) е съставна част от програмата, която решава даден проблем, може да приема параметри и да връща стойност.

Чрез методите се извършва цялата обработка на данни, която програмата трябва да направи, за да реши поставената задача.

Методите съдържат логиката на програмата и те са мястото, където се извършва реалната работа - изчисления. Затова можем да ги приемем като строителни блокчета на програмата.

Съответно, имайки множество от простички елементи – отделни методи, можем да създаваме големи програми, с които да решим по-сложни проблеми.

Използвайки методи ние описваме самата обработка на данните, а когато извикваме даден метод ние го правим многократно с различни данни за обработка.

Предимствата при използването на методи са:

- **По-добро структуриране и по-добра четимост**

При създаването на една програма, е добра практика да използваме методи, за да я направим добре структурирана и лесно четима не само за нас, но и за други хора.

- **Избягване на повторението на код**

Друга много важна причина, заради която е добре да използваме методи е, че по този начин избягваме повторението на код. Това е пряко свързано с концепцията за преизползване на кода.

- **Преизползване на кода**

Добър стил на програмиране е, когато използваме даден фрагмент програмен код повече от един или два пъти в програмата си, да го дефинираме като отделен метод, за да можем да го изпълняваме многократно с различни входни данни.

Методи в C# - общи характеристики:

- В C# няма глобални функции, както в някои други езици (като C++ и Delphi)
- Методи могат да се дефинират само като членове на тип (клас или структура)
- Методите дефинират операции за типа, в който са дефинирани
- Могат да приемат параметри и да връщат стойност
- Има няколко вида предаване на параметри
- Методите могат да имат зададена видимост
- Могат да бъдат статични (да не са обвързани с инстанция на типа)
- Имат синтаксис, близък до C++ и Java

Деклариране на метод, представлява регистриране на метода в нашата програма. То става чрез следната декларация:

```
[модификатор] <тип данни> <име на метод>([<списък с параметри>])
```

Задължителните елементи в декларацията на един метод са:

- Тип на връщаната от метода стойност – `<тип данни>`

- Име на метода – `<име на метод>`

- Списък с параметри на метода – < `списък с параметри` > – може да е празен списък или да съдържа поредица от декларации на параметри.

Примери:

```
(1)
float Div(int number1, int number2)

{

      int result = number1 / number2;

      return result;
```

```
}
(2)
public int FindMax(int num1, int num2)
   {
          int result;

      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
(3)
class Car
    {
        public string model;
        public int speed;

     public void ShowCuurentSpeed()   // Метод без параметри
        {
Console.WriteLine("Текущата скорост {0} е {1}.", model, speed);
        }

     public void Accelarate(int kph)   // Метод с 1 параметър
        {
            speed += kph;        }
    }
}
```

Много често в нашите програми се налага да въвеждаме и обработваме данни за хора ето един прост пример: Дефиниране на клас Човек с полета Име и Възраст и метод предефиниращ ToString

```
(4)
public class Person
{
  public string  Name;  // полета
  public int     Age;

  public Person()       // конструктор
  { this.Name = "?"; this.Age = -1;   }
  public Person(string name, int age)  // конструктор с
параметри
  { this.Name = name; this.Age = age; }

  public override string ToString()    // метод
  { return this.Name; }

}    // край на декларацията на класа
```

Пример за деклариране на обект от класа и промяна на възрастта

```
Person  p;
p = new Person("Иван Петров", 40);
p.Age = p.Age + 1;
Декларация на деструктор
~Person()  // деструктор
  {
    ...
  }
Пример за наследяване
public class Student : Person
{
  private int  m_ID;
  public Student(string name, int age, int id)  // конструктор
    :base(name, age)
  {
    this.m_ID = id;
  }
}
```

11.7. Конструктори

В обектно-ориентираното програмиране, когато създаваме обект от даден клас, е необходимо да извикаме елемент от класа, наречен конструктор.

Конструктор на даден клас, наричаме псевдометод, който няма тип на връщана стойност, носи името на класа и който се извиква чрез ключовата дума new. Ролята на конструктора е да инициализира заделената за обекта памет, в която ще се съхраняват неговите полетата(член променливи).

Декларацията на конструктора се реализира така:

```
[<модификатор>] <име_на_клас>([<списък_параметри>])
```

Както вече знаем, конструкторите наподобяват методите, но не връщат стойност затова декларацията е подобна на тази на метод.

Пример:

```
using System;

namespace ConsoleOOP
{
    class Car
    {
        private string make;
        private string model;

//конструктор без параметри

        public Car()
        {
            make = "Audi";
            model = "A5";
        }
```

```
        }
    }
```

11.8. Задачи за упражнение

Задача 1. Да се напише програма реализираща клас Car с полета name и speed. В класа са реализирани: метод за извеждане на екрана на модела и текущата скорост на колата и метод за увеличаване на скоростта на автомобила(ускорение). В Main да се реализира извеждане на екрана на началното състояние, ускоряване на автомобила (10км/ч) и извеждане на актуализираната скорост. Полетата са достъпни в Main те са public.

```
/// Реализация на класа
using System;

namespace ConsoleOOP
{
    class Car
    {
        public string model;
        public int speed;
        public void ShowCuurentSpeed()
        {
Console.WriteLine("Текущата скорост на {0} е {1}.", model,
speed);
        }
        public void Accelarate(int kph)
        {
            speed += kph;        }
    }
}
```

……

Реализация на главната функция

```
using System;

namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            Car myCar = new Car();
            myCar.model = "Opel";
            myCar.speed = 55;
            Console.WriteLine("Начално състояние.");
            myCar.ShowCuurentSpeed();
            Console.WriteLine("Ускоряване.");
            myCar.Accelarate(10);
            Console.WriteLine("Текуща скорост.");
            myCar.ShowCuurentSpeed();
```

```
        }
   }
}
```

Задача 2. Да се напише програма разширяваща програмата в Задача 1., като в Main се реализира ускоряване на автомобила няколко пъти с 8км/ч и извеждане на актуализираната скорост.

```
/// Реализация на класа – без промяна
……
///реализация на главната функция
using System;

namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            Car myCar = new Car();
            myCar.model = "Bmw";
            myCar.speed = 40;
            Console.WriteLine("Начално състояние.");
            myCar.ShowCuurentSpeed();
            Console.WriteLine("Ускоряване неколкократно.");
            for (int i = 0; i <= 5; i++)
            {
                myCar.Accelarate(8);
                myCar.ShowCuurentSpeed();
            }


        }
    }
}
```

Задача 3. Да се напише програма реализираща клас Car с полета make, model и speed. В класа са реализирани: метод за извеждане на екрана на марката, модела и текущата скорост на колата и метод за увеличаване на скоростта на автомобила(ускорение) определен брой пъти. В Main да се реализира извеждане на екрана на началното състояние, ускоряване на автомобила 3 пъти с 8км/ч и извеждане на актуализираната скорост. Използва се ключовата дума this за достъп до полетата на класа.

```
/// Реализация на класа
using System;

namespace ConsoleOOP
{
```

```
    class Car
    {
        public string make;
        public string model;
        public int speed;
        public void ShowCuurentSpeed()
        {
Console.WriteLine("Текущата скорост на {0},{1} е
{2}.",this.make, this.model, this.speed);        }
        public void Accelarate(int kph)
        {
            this.speed += kph;
        }
/// Нов метод със същото име но различен брой параметри
Overloading
 public void Accelarate(int kph, int times)
        {
            for (int i = 0; i <= times; i++)
            {
                this.speed += kph;
            }
        }
    }
}

///реализация на главната функция
using System;

namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            Car myCar = new Car();
            myCar.make = "Mercedes";
            myCar.model = "S500";
            myCar.speed = 60;
            Console.WriteLine("Начално състояние");
            myCar.ShowCuurentSpeed();
            Console.WriteLine("Ускоряване 3 тъти");

                myCar.Accelarate(8,3);
                myCar.ShowCuurentSpeed();
          }
    }
}
```

Демонстрация на употребата на this

```
using System;
```

```
using System.Collections.Generic;

class Program
{
    class Employee
    {
        private string name;
        private string alias;
        private decimal salary = 1000.00m;

        // конструктор
        public Employee(string name, string alias)
        {
            this.name = name;
            this.alias = alias;
        }
        // метод за извеждане на екрана
        public void printEmployee()
        {
            Console.WriteLine("Име: {0}\nИме(съкратено): {1}",
name, alias);
            // предаване на обект като параметър на метода
CalcTax, чрез this
            Console.WriteLine("Данък: {0:C}",
Tax.CalcTax(this));
        }

        public decimal Salary
        {
            get { return salary; }
        }
    }

    class Tax
    {
        public static decimal CalcTax(Employee em)
        {
            return 0.1m * em.Salary;
        }
    }



    static void Main()
    {


        // създаване на обект
        Employee em1 = new Employee("Moni Panov", "MoPan");

        // показване на данните
```

```
        em1.printEmployee();


    }
}
```

Задача 4. Да се напише програма реализираща клас Car (с реализация на Default конструктор) с полета make, model и speed. В класа са реализирани: метод за извеждане на екрана на марката, модела и текущата скорост на колата и метод за увеличаване на скоростта на автомобила(ускорение) определен брой пъти. В Main да се реализира извеждане на екрана на началното състояние, ускоряване на автомобила 4 пъти с 10км/ч и извеждане на актуализираната скорост. Използва се ключовата дума this за достъп до полетата на класа.

```
/// Реализация на класа
using System;


namespace ConsoleOOP
{
    class Car
    {
        private string make;
        private string model;
        private int speed;
       /// <summary>
       /// Default конструктор името съвпада с името на класа
няма параметри
       /// </summary>
        public Car()
        {
            this.make = "Audi";
            model = "A5";
            speed = 80;
        }
        public void ShowCuurentSpeed()
        {
            Console.WriteLine("Текуща скорост {0},{1} -
{2}",make, model, speed);
        }
        public void Accelarate(int kph)
        {
            speed += kph;
        }
        public void Accelarate(int kph, int times)
        {
            for (int i = 0; i <= times; i++)
            {
```

```
                speed += kph;
            }
        }
    }
}

///реализация на главната функция
using System;


namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
    Car myCar = new Car();// изпълнява се Default конструктора

            Console.WriteLine("Начало");
            myCar.ShowCuurentSpeed();
            Console.WriteLine("Ускорение 4 пъти");

                myCar.Accelarate(10,4);
                myCar.ShowCuurentSpeed();



        }
    }
}
```

Задача 5. Да се напише програма реализираща клас Product (с реализация на Custom конструктор) с полета make, model и price. В класа са реализирани: метод за извеждане на екрана на марката, модела и текущата цена и метод за промяна на цената. В Main да се реализира извеждане на екрана на началното състояние и промяна на цената с такава въведена от клавиатурата.

```
/// Реализация на класа
using System;


namespace ConsoleOOP
{
    class Product
    {
        public string make;
        public string model;
```

```
        public decimal price;
       /// <summary>
       /// Custom конструктор
       /// </summary>
        public Product(string manufacturer, string type,
decimal cost)
        {
            make = manufacturer;
            model = type;
            price = cost;
        }
        public void ShowData()
        {
            Console.WriteLine("Детайли за продукт: {0},{1} ,
{2}",make, model, price);
        }
        public void ChangePrice(decimal newPrice)
        {
            price = newPrice;
        }

    }
}
///реализация на главната функция
using System;


namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
       Product myProduct = new Product("AEG", "S45", 50.5M);

            Console.WriteLine("Таблица на продуктите.");
            myProduct.ShowData();
            Console.WriteLine("Промяна на цената.");
            Console.WriteLine("Въведете новата цена.");
        decimal newPrice = decimal.Parse(Console.ReadLine());
            Console.WriteLine("Актуализирани данни.");
            myProduct.ChangePrice(newPrice);
            myProduct.ShowData();


        }
    }
}
```

Задача 6. Да се напише програма решаваща задача 5, като използвате принципа капсулиране. За достъп до всяко поле се декларира свойство (property) даващо достъп до private полето.

```
/// Реализация на класа
using System;


namespace ConsoleOOP
{
    class Product
    {
        private string make;

        public string Make
        {
            get { return make; }
            set { make = value; }
        }
        private string model;

        public string Model
        {
            get { return model; }
            set { model = value; }
        }
        private decimal price;

        public decimal Price
        {
            get { return price; }
            set { price = value; }
        }
       /// <summary>
       /// Custom конструктор
       /// </summary>
        public Product(string manufacturer, string type,
decimal cost)
        {
            Make = manufacturer;
            Model = type;
            Price = cost;
        }
        public void ShowData()
        {
            Console.WriteLine("Детайли за продукта: {0},{1} ,
{2}",Make, Model, Price);
        }
        public void ChangePrice(decimal newPrice)
        {
            Price = newPrice;
```

```
        }

    }
}
```

Задача 7. Да се напише програма реализираща класове Product и Computer втория наследник на първия илюстрираща принципа на ООП-наследяване.

```
/// Реализация на класа
using System;


namespace ConsoleOOP
{
    class Product
    {
        private string make;

        public string Make
        {
            get { return make; }
            set { make = value; }
        }
        private string model;

        public string Model
        {
            get { return model; }
            set { model = value; }
        }
        private decimal price;

        public decimal Price
        {
            get { return price; }
            set { price = value; }
        }
        public Product()
        {

        }
       /// <summary>
       /// Custom конструктор
       /// </summary>
public Product(string manufacturer, string type, decimal cost)
        {
            Make = manufacturer;
            Model = type;
            Price = cost;
        }
        public void ShowData()
```

```
        {
            Console.WriteLine("Детайли за продукта: {0},{1} ,
{2}",Make, Model, Price);
        }
        public void ChangePrice(decimal newPrice)
        {
            Price = newPrice;
        }

    }
class Computer:Product
{
// класа Computer притежава всички свойства и методи на
Product
}


///реализация на главната функция
using System;


namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            Computer myPc = new Computer();
            myPc.Make = "Acer";
            myPc.Model = "Aspire";
            myPc.Price = 1000;
            Console.WriteLine("Продукт");
            myPc.ShowData();
            Console.WriteLine("Промяна на цена");
            Console.WriteLine("Въвдете новата цена");
         decimal newPrice = decimal.Parse(Console.ReadLine());
            Console.WriteLine("Нова цена");
            myPc.ChangePrice(newPrice);
            myPc.ShowData();


        }
    }
}
```

Задача 8. Да се напише програма разширяваща решението на задача 7 с допълнение на свойства на класа Computer реализация на нов метод за извеждане на екрана.

```
/// Реализация на класа
using System;
```

```
namespace ConsoleOOP
{
    class Product
    {
        private string make;

        public string Make
        {
            get { return make; }
            set { make = value; }
        }
        private string model;

        public string Model
        {
            get { return model; }
            set { model = value; }
        }
        private decimal price;

        public decimal Price
        {
            get { return price; }
            set { price = value; }
        }
        public Product()
        {

        }
       /// <summary>
       /// Custom конструктор
       /// </summary>
public Product(string manufacturer, string type, decimal cost)
        {
            Make = manufacturer;
            Model = type;
            Price = cost;
        }
        public virtual void ShowData() // Виртуален метод
        {
            Console.WriteLine("Детайли на продукт: {0},{1} ,
{2}",Make, Model, Price);
        }
        public void ChangePrice(decimal newPrice)
        {
            Price = newPrice;
        }

    }
```

```
class Computer:Product
{

// Автоматични свойства
    public string CPU
    { get; set; }
    public int RAM
    { get; set; }

//Предефиниран метод
    public override void ShowData()
    {
        Console.WriteLine("Computer details:Make {0}, Model
{1} , CPU {2}, RAM {3} at Price {4}", Make, Model,
CPU,RAM,Price);
    }
}
}

///реализация на главната функция
using System;


namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            Computer myPc = new Computer();
            myPc.Make = "Toshiba";
            myPc.Model = "Satelitte";
            myPc.Price = 1000;
            myPc.CPU = "Intel";
            myPc.RAM = 2048;
            Console.WriteLine("Продукт");
            myPc.ShowData();
            Console.WriteLine("Промяна на цена");
            Console.WriteLine("Въведете нова цена");
            decimal newPrice =
decimal.Parse(Console.ReadLine());
            Console.WriteLine("Нова цена");
            myPc.ChangePrice(newPrice);
            myPc.ShowData();


        }
    }
}
```

Задача 9. Да се напише програма за изчисление на лицето и обиколката на правоъгълник и окръжност. Да се използва декларация на абстрактен клас Фигура.

```
using System;

namespace Program
{

    //Абстрактен клас
    abstract class Shape
    {

        protected float r, a, h;

        //Абстрактните методи съдържат само дефиниция
        public abstract float Area();
        public abstract float Circumference();

    }


    class Rectangle : Shape
    {
        public void GetLB()
        {
            Console.Write("Ширина  :  ");

            a = float.Parse(Console.ReadLine());

            Console.Write("Височина: ");

            h = float.Parse(Console.ReadLine());
        }


        public override float Area()
        {
            return a * h;
        }

        public override float Circumference()
        {
            return 2 * (a + h);
        }

    }


    class Circle : Shape
    {
```

```
        public void GetRadius()
        {

            Console.Write("Въведете радиус  :  ");
            r = float.Parse(Console.ReadLine());
        }

        public override float Area()
        {
            return 3.14F * r * r;
        }
        public override float Circumference()
        {
            return 2 * 3.14F * r;

        }
    }
    class MainClass
    {
        public static void Calculate(Shape s)
        {

            Console.WriteLine("Площ : {0}", s.Area());
            Console.WriteLine("Обиколка : {0}",
s.Circumference());

        }
        static void Main()
        {

            Rectangle R = new Rectangle();
            R.GetLB();
            Calculate(R);

            Console.WriteLine();

            Circle C = new Circle();
            C.GetRadius();
            Calculate(C);

            Console.Read();

        }
    }
}
```

Задача 10. Да се напише програма за демонстрация на използването на интерфейси. За целта да се реализира интерфейс IEmployee с методи за добавяне, изтриване, търсене, актуализиране и изчисляване на заплата. Да

се реализира клас FullTimeEmployee имплементиращ методите на интерфейса и полета за идентификационен номер, име и фамилия.

```
/// Интерфейс - съдържа само дефиниции
using System;

namespace ConsoleOOP
{
    interface IEmployee
    {
        String ID
        {
            get;
            set;
        }

        String FirstName
        {
            get;
            set;
        }

        String LastName
        {
            get;
            set;
        }


        String Update();

        String Add();

        String Delete();

        String Search();

        String CalculateSalary();


    }
}

/// Клас за имплементация
using System;

namespace ConsoleOOP
{
    class FullTimeEmployee:IEmployee
    {
          protected String id;
```

```
        protected String lname;
        protected String fname;

        public FullTimeEmployee()
        {

        }

        public String ID
        {
            get

            {
                return id;
            }
            set
            {
                id = value;
            }
        }

        public String FirstName
        {
            get
            {
                return fname;
            }
            set

            {
                fname = value;
            }
        }

        public String LastName
        {
            get
            {
                return lname;
            }
            set
            {
                lname = value;
            }
        }



        public String Add()
        {
            return "Служител " +
```

```
                            fname +" "+lname+ " добавен.";
        }

        public String Delete()
        {
            return " Служител  " +
                         fname + " " + lname + " изтрит.";
        }

        public String Search()
        {
            return " Служител " +
                        fname + " " + lname + " намерен.";
        }

        public String Update()
        {
            return " Служител " +
                        fname + " " + lname +" актуализиран.";
        }



        public String CalculateSalary()
        {
            return " Служител " +
                 fname + " " + lname + " изчисляване на
заплата, чрез  " +
                  "Interface.";
        }

    }
}


///реализация на главната функция
using System;


namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {


            FullTimeEmployee emp = new FullTimeEmployee();


            emp.ID = "1234";
```

```
            emp.FirstName = "Ivan";
            emp.LastName = "Petrov";

          Console.WriteLine(emp.Add().ToString());


Console.WriteLine(emp.CalculateSalary().ToString());


        }
    }
}
```

Задача 11. Да се разшири програмата за демонстрация на използването на интерфейси. Да се реализира клас PartTimeEmployee имплементиращ методите на интерфейса и полета за идентификационен номер, име и фамилия.

```
/// Клас за имплементация
using System;

namespace ConsoleOOP
{
    class PartTimeEmployee:IEmployee
    {
        protected String id;
        protected String lname;
        protected String fname;
          public PartTimeEmployee()
        {

        }

        public String ID
        {
            get

            {
                return id;
            }
            set
            {
                id = value;
            }
        }

        public String FirstName
        {
            get
            {
                return fname;
```

```
            }
            set

            {
                fname = value;
            }
        }

        public String LastName
        {
            get
            {
                return lname;
            }
            set
            {
                lname = value;
            }
        }



            public String Add()
        {
            return "Служител " +
                           fname +" "+lname+ " добавен.";
        }

        public String Delete()
        {
            return " Служител  " +
                          fname + " " + lname + " изтрит.";
        }

        public String Search()
        {
            return " Служител " +
                        fname + " " + lname + " намерен.";
        }

        public String Update()
        {
            return " Служител " +
                        fname + " " + lname +" актуализиран.";
        }



        public String CalculateSalary()
        {
            return " Служител на намалено работно време " +
```

```
                    fname + " " + lname + " заплата. " ;
        }

    }
}

///реализация на главната функция
using System;


namespace ConsoleOOP
{
    class Program
    {
        static void Main(string[] args)
        {


            FullTimeEmployee emp = new FullTimeEmployee();
            PartTimeEmployee emp2 = new PartTimeEmployee();

            emp.ID = "1234";
            emp.FirstName = "Ivan";
            emp.LastName = "Petrov";

            emp2.ID = "1235";
            emp2.FirstName = "Milena";
            emp2.LastName = "Petrova";


          Console.WriteLine(emp.Add().ToString());
          Console.WriteLine(emp.CalculateSalary().ToString());
          Console.WriteLine("Нов служител на намалено работно
време");
          Console.WriteLine(emp2.Add().ToString());
          Console.WriteLine(emp2.Update().ToString());


        }
    }
}
```

11.9. Задачи за самостоятелна работа

Задача 1. Да се напише програма реализираща клас Car с полета make, model и speed. В класа са реализирани: метод за извеждане на екрана на марката, модела и текущата скорост на колата и метод за увеличаване на скоростта на автомобила(ускорение) определен брой пъти. В Main да се реализира

извеждане на екрана на началното състояние, ускоряване на автомобила 5 пъти с 5км/ч и извеждане на актуализираната скорост.

Задача 2. Да се напише програма реализираща клас Car с полета model, price и Kms_Driven. В класа са реализирани: метод за извеждане на екрана на модела, цената и изминатите километри на автомобила и методи за промяна на цената и изминатите километри на автомобила. В Main да се реализира въвеждане на три автомобила с техните начални данни и промяна съответно на цената и изминатите километри, да се изведат на екрана новите данни за трите автомобила.

ЛИТЕРАТУРА

1. Наков С., и колектив, Въведение в програмирането със C#, Издателство: Фабер, Велико Търново, 2011, ISBN: 978-954-400-527-6.
2. Albahari J., C# 5.0 in a Nutshell: The Definitive Reference, O'Reilly Media, 2012, ISBN-13: 978-1449320102.
3. Balagurusamy E., Programming in C#: A Primer, McGraw-Hill Education, 2010, ISBN-10: 0-07-070207-1.
4. C# Programming Guide, https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx, Последно посетен на 26.05.2016 г.
5. C# Reference, https://msdn.microsoft.com/bg-bg/library/618ayhy6.aspx, Последно посетен на 26.05.2016 г.
6. Deep C#, http://www.i-programmer.info/ebooks/deep-c.html, Последно посетен на 26.05.2016 г.
7. Greene J., Head First C#, O'Reilly Media, 2013, ISBN-13: 978-1449343507.
8. Miles R. C# Yellow Book, University of Hull, 2015, ASIN: B00HNSGM9A.
9. Murach M., Murach's C# 2012, Mike Murach & Associates, 2013, ISBN-13: 978-1890774721.
10. Smyth N., C# Essentials, Payload Media, 2012, http://www.techotopia.com/index.php/C_Sharp_Essentials.