Курс по програмиране на C#

ДАВИД холдинг АД | гр.“Казанлък“, ул.“Стара река“ 2, ДК „Арсенал“, ет.4

Упражнения

Валери Дачев и Александър Далемски

2013

# Съдържание

[Съдържание 1](#_Toc389173297)

[Дискриминанта на квардратно уварнение 5](#_Toc389173298)

[Задача 5](#_Toc389173299)

[Поздрав 5](#_Toc389173300)

[Задача 5](#_Toc389173301)

[Графичен потребителски интерфейс на калкулатор 5](#_Toc389173302)

[Задача 5](#_Toc389173303)

[Допълнителни задачи 5](#_Toc389173304)

[Проверка за високосна година 5](#_Toc389173305)

[Задача 5](#_Toc389173306)

[Форматиране на дробно число 5](#_Toc389173307)

[Задача 5](#_Toc389173308)

[Проверка за дробно число 5](#_Toc389173309)

[Задача 5](#_Toc389173310)

[Намиране на прости числа 5](#_Toc389173311)

[Задача 5](#_Toc389173312)

[Намиране на факториел 6](#_Toc389173313)

[Задача 6](#_Toc389173314)

[Допълнителни задачи 6](#_Toc389173315)

[Намиране на число от редицата на Фибоначи 6](#_Toc389173316)

[Задача 6](#_Toc389173317)

[Решаване на квадратно уравнение 6](#_Toc389173318)

[Задача 6](#_Toc389173319)

[Намиране на среден успех на клас 6](#_Toc389173320)

[Задача 6](#_Toc389173321)

[Конзолен калкулатор 6](#_Toc389173322)

[Задача 6](#_Toc389173323)

[Текстово-базирана приключенска игра 7](#_Toc389173324)

[Задача 7](#_Toc389173325)

[Допълнителни задачи 7](#_Toc389173326)

[Най-голям и най-малък елемент на масив 7](#_Toc389173327)

[Задача 7](#_Toc389173328)

[Сортиране на масив по метода на мехурчето 7](#_Toc389173329)

[Задача 7](#_Toc389173330)

[Допълнителни задачи 7](#_Toc389173331)

[Намиране на броя на инверсиите в масив 7](#_Toc389173332)

[Задача 7](#_Toc389173333)

[Сумиране на елементите на квадратна матрица 7](#_Toc389173334)

[Задача 7](#_Toc389173335)

[Температури през седмицата 8](#_Toc389173336)

[Задача 8](#_Toc389173337)

[Пресмятане на срочни оценки 8](#_Toc389173338)

[Задача 8](#_Toc389173339)

[Градове и пътища 8](#_Toc389173340)

[Задача 8](#_Toc389173341)

[Обръщане на масив 8](#_Toc389173342)

[Задача 8](#_Toc389173343)

[Изисквания 8](#_Toc389173344)

[Допълнителни задачи 8](#_Toc389173345)

[Златотърсачи 9](#_Toc389173346)

[Задача 9](#_Toc389173347)

[Изисквания 9](#_Toc389173348)

[Допълнителни задачи 10](#_Toc389173349)

[Двоично търсене в сортиран масив 10](#_Toc389173350)

[Задача 10](#_Toc389173351)

[Изисквания 10](#_Toc389173352)

[Допълнителни задачи 10](#_Toc389173353)

[Търсене на път в лабиринт 10](#_Toc389173354)

[Задача 10](#_Toc389173355)

[Изисквания 11](#_Toc389173356)

[Допълнителни задачи 11](#_Toc389173357)

[Сняг 11](#_Toc389173358)

[Задача 11](#_Toc389173359)

[Описание 11](#_Toc389173360)

[Изисквания 11](#_Toc389173361)

[Допълнителни задачи 11](#_Toc389173362)

[Метеорологична информация 12](#_Toc389173363)

[Задача 12](#_Toc389173364)

[Автомобили и водачи 12](#_Toc389173365)

[Задача 12](#_Toc389173366)

[Геометрични фигури 12](#_Toc389173367)

[Задача 12](#_Toc389173368)

[Изисквания 12](#_Toc389173369)

[Допълнителни задачи 13](#_Toc389173370)

[Софтуерна академия 14](#_Toc389173371)

[Задача 14](#_Toc389173372)

[Изисквания 14](#_Toc389173373)

[Библиотека 14](#_Toc389173374)

[Задача 14](#_Toc389173375)

[Изисквания 15](#_Toc389173376)

[Допълнителни задачи 15](#_Toc389173377)

[Ролева игра 15](#_Toc389173378)

[Задача 15](#_Toc389173379)

[Изисквания 15](#_Toc389173380)

[Допълнителни задачи 16](#_Toc389173381)

[Адресна книга 16](#_Toc389173382)

[Задача 16](#_Toc389173383)

[Изисквания 16](#_Toc389173384)

[Допълнителни задачи 16](#_Toc389173385)

[Приоритетна опашка 17](#_Toc389173386)

[Задача 17](#_Toc389173387)

[Изисквания 17](#_Toc389173388)

[Шаблонни алгоритми за сортиране 17](#_Toc389173389)

[Рационални числа 17](#_Toc389173390)

[Задача 17](#_Toc389173391)

[Изисквания 17](#_Toc389173392)

[Допълнителни задачи 18](#_Toc389173393)

[Аритметични изрази 18](#_Toc389173394)

[Задача 18](#_Toc389173395)

[Изисквания 18](#_Toc389173396)

[Допълнителни задачи 18](#_Toc389173397)

[Оказион 18](#_Toc389173398)

[Задача 18](#_Toc389173399)

[Изисквания 19](#_Toc389173400)

[Допълнителни задачи 19](#_Toc389173401)

[Информация за разработчици 19](#_Toc389173402)

[Задача 19](#_Toc389173403)

[Изисквания 19](#_Toc389173404)

[Допълнителни задачи 20](#_Toc389173405)

[Калкулатор с използване на делегати 20](#_Toc389173406)

[Задача 20](#_Toc389173407)

[Допълнителни задачи 20](#_Toc389173408)

# Дискриминанта на квардратно уварнение

## Задача

Създайте програма, в която променливите **a**, **b** и **c** се инициализират с десетични числа – коефициенти в квадратно уравнение. Нека програмата изчисли дискриминантата, запише резултата в променлива **discriminant** и го покаже на екрана.

# Поздрав

## Задача

Създайте програма, която инициализира променливата **studentName** с имената на ученик и променливата **courseName** – с наименованието на курс. Нека програмата запише в променлива **message** текста „Здравейте **[име на ученик]** и добре дошли в **[наименование на курс]**!” и покаже съобщението на екрана.

# Графичен потребителски интерфейс на калкулатор

## Задача

Създайте интерфейса на визуално приложение – калкулатор. Използвайте различни визуални компоненти (бутони, текстови полета, менюта и др.) с подходящите наименования и текстове.

## Допълнителни задачи

* Реализирайте реалната функционалност на калкулатора; описание на необходимите стъпки може да намерите [тук](https://www.facebook.com/DavidAcademy/posts/608317065895085?comment_id=5462897&offset=0&total_comments=1).

# Проверка за високосна година

## Задача

Създайте програма, в която се въвежда **година** (цяло положително число) и се извежда съобщение дали тя е **високосна** или не.

# Форматиране на дробно число

## Задача

Създайте програма, в която се въвежда **дробно число** и след това то се извежда с избран от потребителя **брой знаци след десетичната запетая**.

# Проверка за дробно число

## Задача

Създайте програма, в която въвежда се **цяло положително число** и се извежда съобщение дали то е **просто** или не.

# Намиране на прости числа

## Задача

Създайте програма, в която се въвежда **цяло положително число** и се извеждат всички **прости** числа, **по-малки или равни** на него.

# Намиране на факториел

## Задача

Създайте програма, която изчислява факториел на дадено число ***n***.

## Допълнителни задачи

* Намирането на факториел от число оформете като функция ***GetFactorialIteratively***, която използва итерация (т.е. използва цикли).
* Променете програмата така, че тя да може да работи с големи числа като резултат от изчислението;
* Модифицирайте програмата така, че да дава възможност на потребителя да въведе цялото число ***n***, като проверявате за коректен вход;
* Модифицирайте програмата така, че да попита потребителя дали иска да продължи. Ако избере да продължи, изпълнението да започне отначало. За целта използвайте подходящ цикъл;
* Добавете функция ***GetFactorialRecursively***, която намира факториел рекурсивно на подадено число използвайки свойството:

n! = n . (n-1)!

* Дайте възможност на потребителя да избере начина на изчисляване – итеративно или рекурсивно;
* Реализирайте приложението като Windows Forms Application. Използвайте ***NumericUpDown*** за въвеждане на числото ***n***, ***Label*** за извеждане на резултата, ***Button*** за рекурсивно изчисление и ***Button*** за итеративно изчисление.

# Намиране на число от редицата на Фибоначи

## Задача

Създайте програма, в която се въвежда **цяло положително число n** и се извежда n-тото по ред число от **редицата на Фибоначи**.

# Решаване на квадратно уравнение

## Задача

Създайте програма, в която се въвеждат **коефициентите на квадратно уравнение** и се намират неговите **решения**, като се разгледат всички случаи.

# Намиране на среден успех на клас

## Задача

Създайте програма, в която се въвежда **поредица** от **оценки** (дробни числа между 2 и 6) на ученици и се извежда **средният успех** на класа.

# Конзолен калкулатор

## Задача

Създайте конзолно приложение-калкулатор, в което потребителят избира **аритметична операция** (**събиране**, **изваждане**, **умножение**, **деление**), въвежда двата **операнда** (дробни числа) и получава **резултата**; изпълнението да продължи до избиране на опция за **изход** вместо аритметична операция.

# Текстово-базирана приключенска игра

## Задача

Създайте проста текстово-базирана приключенска игра, в която потребителят въвежда **текстови команди** и след всяка команда получава **описание на резултата** от нея.

## Допълнителни задачи

* Реализирайте програмата с помощта на масиви, така че да избегнете влагането на условни преходи;
* Разделете програмата на подходящи функции, така че да я направите по-удобна за четене и модификация;
* Реализирайте сюжета на играта така, че до някои от етапите да може да се стигне по различни пътища.

# Най-голям и най-малък елемент на масив

## Задача

Създайте програма, в която се въвежда **масив от цели числа** и се извеждат **най-голямото число**, **най-малкото число** и **средното аритметично** на всички числа в масива.

# Сортиране на масив по метода на мехурчето

## Задача

Създайте програма, в която се въвежда **масив от дробни числа**, масивът се сортира във възходящ ред по метода на мехурчето и след това се извеждат **сортираните числа**.

## Допълнителни задачи

* Потребителят да може да избере дължината на въвеждания масив;
* Потребителят да може да избере между възходящ и низходящ ред на сортирането.

# Намиране на броя на инверсиите в масив

## Задача

Създайте програма, в която се въвежда **масив от цели числа** и се извежда **броят на инверсиите**. Два елемента с индекси **i** и **j** се намират в **инверсия**, точно когато **i** < **j** и елементът с индекс **i** е по-голям от елемента с индекс **j**.

# Сумиране на елементите на квадратна матрица

## Задача

Създайте програма, в която се въвежда **двумерна квадратна матрица от дробни числа** и се извеждат **сумата на всички елементи по главния диагонал**, **сумата на всички елементи над него** и **сумата на всички елементи под него**. Главен диагонал наричаме елементите **с равни първи и втори индекси**.

# Температури през седмицата

## Задача

Създайте програма, в която се въвежда **правоъгълна матрица от температури** (дробни числа) със **7 реда** и **6 колони**. Всеки от редовете съответства на **ден от седмицата**, започвайки с понеделник. Колоните съответстват на **00:00**, **04:00**, **08:00**, **12:00**, **16:00** и **20:00 часа**. Всяка клетка от матрицата съдържа **температурата в съответния час и ден** за една седмица. Програмата трябва да пресметне и изведе **средната температура за всеки от седемте дни**, както и **средната температура за всеки от шестте момента от денонощието**.

# Пресмятане на срочни оценки

## Задача

Създайте програма, в която се въвеждат **текущите оценки** (дробни числа между 2 и 6) по един учебен предмет на учениците от един клас. Възможно е всеки от учениците да има **различен брой текущи оценки** по предмета. Програмата трябва да пресмята и извежда **срочната оценка** на всеки от учениците (като **средно аритметично от текущите му оценки**), както и **средния успех на целия клас** по предмета (като **средно аритметично от всички срочни оценки**).

# Градове и пътища

## Задача

В една местност има **n** на брой градове (номерирани от **0** до **n-1**) и някои от тях са свързани с пътища. Създайте програма, в която се въвежда **n** и след това за **всеки от градовете** последователно се въвеждат **номерата на градовете**, които са **свързани с него чрез директен път**. Програмата трябва да пресметне и изведе **двойките градове**, които са свързани с **най-кратък път**, минаващ **през точно един друг град**, **точно два други града** и т.н. За простота приемете, че пътищата са **еднопосочни** (наличието на път от град **i** до град **j** не гарантира наличие на обратен път от град **j** към град **i**).

# Обръщане на масив

## Задача

Напишете програма, която обръща масив от числа отзад-напред.

## Изисквания

* Обръщането на масива оформете като процедура ***ReverseArrayIteratively***, която използва итерация (т.е. използва цикли);
* Покажете резултата с функция ***ShowArray***.

## Допълнителни задачи

* Модифицирайте програмата така, че да чете масива от клавиатурата, като:
	+ Потребителят въвежда броя на числата;
	+ Всяко число потребителят въвежда на отделен ред. Проверявайте за коректен вход. Ако дадено число не е въведено коректно, подканете потребителя да го въведе отново.
* Модифицирайте програмата така, че потребителят да не въвежда броя на числата, а всяко ново число да се добавя към масива и въвеждането да приключва с въвеждането на празен ред;
* Добавете функция ***ReverseArrayRecursively***, която да обръща масива рекурсивно последователно разменя крайните две числа, след което предпоследните две числа и т.н.;
* Дайте възможност на потребителя да избере начина на изчисляване – итеративно или рекурсивно.

# Златотърсачи

## Задача

Създайте конзолна игра, в която играчът търси случайно разположени диаманти в друизмерно правоъгълно игрално поле.

## Изисквания

* Създайте метод **GetFieldBounds**, в който потребителят въвежда две цели числа **x ∈ (10; maxX)** и **y ∈ (10; maxY)**, **maxX** и **maxY** се подават в параметър. Потребителят да може да откаже въвеждането с въвеждане на празен низ, а методът връща информация дали потребителят е отказал въвеждане или не, както и въведените числа (ако са въведени такива);
* Създайте метод **CreateField**, който създава и връща двумерен масив с размери **m** и **n**, елементите на който запълнете с произволни стойности от изброим списък с елементи:
	+ „**OurGuy**“ – точно един на произволно място в полето;
	+ „**SomeGuy**“ – брой съответстващ на 5% от местата в полето на произволни места в полето;
	+ „**Diamond**” – брой съответстващ на 10% от местата в полето на произволни места в полето (броят им се връща от метода);
	+ Останалите места в полето попълнете с едно от следните:
		- „**Ground**“ – с вероятност 40%;
		- „**Grass**“ – с вероятност 30%;
		- „**Tree**“ – с вероятност 20%;
		- „**Stone**“ – с вероятност 10%;
* Създайте метод **DrawField**, който „рисува“ на екрана съдържанието на масива от предишната задача. За всеки елемент от масива поставя съответния оцветен символ:
	+ „**Ground**“ – символ „▒“ (U+2592) оцветен в светло кафяво;
	+ „**Grass**“ – символ „▓“ (U+2593) оцветен в светло зелено;
	+ „**Tree**“ – символ „♣“ (U+2663) оцветен в тъмно зелено;
	+ „**Stone**“ – символ „٥“ (U+0665) оцветен в тъмно сиво и в курсив;
	+ „**Diamond**” – символ „♦“ (U+2666) оцветен в светло сиво;
	+ „**SomeGuy**“ – символ „☻” (U+263B) оцветен в цвят по избор;
	+ „**OurGuy**“ – символ „☺“ (U+263A) oцветен в цвят по избор;
* Създайте метод **MoveGuy**, който приема като параметър „поле“, двойка координатите и посока (стойност от изброим списък “Up”, “Down”, “Left” или “Right”), и мести „guy” намиращ се на съответните координати в съответната посока, само като там има „ground”, „grass” или „diamond”. След себе си „guy“ оставя „ground”.
* Създайте метод **MoveSomeGuys**, който приема като параметър „поле“ и премества всички „some guys“ в произволна посока.
* Създайте метод **MoveOurGuy**, който приема като параметър „поле“ и посока, и мести „our guy” в определената посока.
* Създайте метод **ProcessInput**, който приема като параметър „полето“ и проверява дали потребителят е натиснал клавиш:
	+ Ако е натиснал стрелка, методът връща стойност съответстваща на посоката на стрелката в референтен параметър;
	+ Ако е натиснал Escape, методът връща информация за това в изходен параметър;
	+ Методът връща информация дали потребителят е натиснал клавиш или не;
* Създайте програма съдържаща всички изработени по-горе методи, като основната програма трябва да:
	+ изчисти екрана;
	+ подкани потребителя да въведе размер на полето не по-голям от този на екрана (ако потребителят не въведе такъв, програмата трябва да прекрати работа);
	+ създаде „поле“ с въведените от потребителя размери;
	+ до натискане на „Escape” или свършване на „диамантите“:
		- рисува „полето“;
		- премества всички „other guys”;
		- обработва входа от потребителя и, ако е нужно, премества “our guy”;
		- изчаква 100 ms;
	+ извежда събрания от потребителя процент „диаманти“.

## Допълнителни задачи

* Преработете играта, така че квадратчетата да са от **структурен тип** и да съдържат информация за **терена** и наличието на **допълнителни обекти** върху тях (**хора** и **диаманти**); хората направете **обекти**, съдържащи информация за **координатите**, **цвета** им и **броя на събраните от тях диаманти**.

# Двоично търсене в сортиран масив

## Задача

Напишете програма, която използва [двоично търсене](http://en.wikipedia.org/wiki/Binary_search_algorithm) за намиране на число в сортиран масив от числа.

## Изисквания

* Двоичното търсене в сортиран масив оформете във функция ***BinSearchArrayRecursively***, която използва рекурсия намиране на число;
* Функцията ***BinSearchArrayRecursively*** трябва връща ***boolean***, указващ дали числото е намерено, и ***int*** – индекса на неговия пореден номер.

## Допълнителни задачи

* Добавете функция ***CheckIfSorted***, която проверява дали подаденият масив е сортиран, и я използвайте. Ако масивът не е сортиран, изведете съобщение за грешка и не извършвайте търсене;
* Добавете аналогична функция ***BinSearchArrayIteratively***, която осъществява двоично търсене с използване на цикли.

# Търсене на път в лабиринт

## Задача

Напишете програма, която [намира път в лабиринт рекурсивно](http://www.cs.bu.edu/teaching/alg/maze/).

## Изисквания

* Лабиринтът се представя като двумерен масив от символи с размери ***m*** x ***n***, в който масив символът интервал (***‘ ‘***) означава път, символът звезда (***‘\*’***) – стена, а символът диез (‘***#***’) – стъпка. Първоначалните координати са (***x0***,***y0***), а крайните координати – (***xt***,***yt***) (от думата target). Движението може да става само с една позиция в някоя от четирите посоки (нагоре, надолу, наляво и надясно), като не може да се стъпва на стена или място, на което вече има стъпки;
* Търсенето на път оформете във функция ***FindMazePath***, която използва рекурсия за намиране на пътя в подаден лабиринт.

## Допълнителни задачи

* Добавете функция ***ShowMaze***, която показва текущото състояние на лабиринта (с пътя, стената и стъпките);
* Извиквайте функцията ***ShowMaze*** при всеки ход;
* Добавете функция ***GenerateMaze***, която да създава лабиринт по зададени размери ***m*** и ***n***, като използвате [търсене в дълбочина](http://en.wikipedia.org/wiki/Maze_generation_algorithm#Depth-first_search). Използвайте нейния резултат, за да намерите път с функцията ***FindMazePath***.

# Сняг

## Задача

Напишете програма, която показва “сняг” в конзолата, който се натрупва. Снегът представлява множество от “снежинки” (представени със символа “\*”).

## Описание

Работа на програмата:

1. Изчиства се конзолата при стартиране;
2. От горната част на екрана по широчината на целия екран започват да “вали сняг” – произволен брой снежинки;
3. С времето всички снежинки, под които няма нищо, се “падат” с един ред надолу;
4. Снежинките, които са стигнали дъното на екрана или под които има друга снежинка, остават на мястото си;
5. Движението на снежинките трябва да се забави с определен период от време, за да изглежда падането плавно;
6. Изпълнението на програмата продължава автоматично, докато не бъде натиснат клавиш.

## Изисквания

* Изрисуването на снежинките трябва да става без да се изчиства екрана, за да се избегне премигването му;
* Екранът трябва да бъде представен като двумерен масив от символи, размерът на който зависи от размерите на конзолата;
* Разбийте програмата на отделни функции – за инициализация на масива, за обработка на масива, за изрисуването на екрана и т.н..

## Допълнителни задачи

* Преди започване на “снега”, създайте произволен земен “релеф” (представен със символа символа “@”).
* Ако по време на изпълнение на програмата е натиснат клавиш “стрелка нагоре” или “стрелка надолу”, съответно да се увеличава или намалява скоростта на снега;
* Ако по време на изпълнение на програмата е натиснат клавиш “стрелка наляво” или “стрелка надясно”, да се симулира вятър, като летящите снежинки (тези, под които няма нищо), се преместват съответно наляво или надясно.

# Метеорологична информация

## Задача

Създайте програма, в която се въвежда **метеорологична информация** за **всеки кръгъл час** от едно денонощие (**реална температура**, **посока** и **скорост на вятъра**, **„привидна“ температура**, **атмосферно налягане**, **облачност/валежи/мъгла**; програмата да определя и извежда минималните и максималните стойности на количествените величини.

# Автомобили и водачи

## Задача

Създайте програма, в която се декларират класове за **свиделство за правоуправление на МПС** (**номер**, **дата на издаване**, **издаващ орган**, **валидност**), **автомобил** (**регистрационен номер**, **марка**, **модел**, **дата на регистриране**) и **шофьор** (**имена**, **свидетелство**, **автомобили**). Предоставете на потебителя възможност да регистрира обекти от съответните класове, както и да разглежда вече регистрираните обекти.

# Геометрични фигури

## Задача

Създайте програма, декларираща типове за геометрични обекти (точки, вектори, равнинни фигури и обемни тела).

## Изисквания

* Създайте структура **Point2D**, която описва точка в двумерното пространство:
	+ Изберете подходящо вътрешно представяне на точката;
	+ Добавете конструктор с два параметъра – планарни координати на точката (x, y);
	+ Добавете конструктор с три параметъра – един от изброим списък указващ координатната система и два за самите координати (планарни (x, y) или полярни (r, phi));
	+ Добавете метод, който връща планарните координати (x, y) на точката;
	+ Добавете метод, който връща полярните координати (r, phi) на точката;
	+ Добавете метод, който връща разстоянието до друга точка **Point2D**;
* Създайте клас **Vector2D**, който описва вектор в двумерното пространство:
	+ Изберете подходящо вътрешно представяне на вектора;
	+ Добавете конструктор с два параметъра – компоненти на вектора в декартова координатна система (x, y);
	+ Добавете конструктор с три параметъра – един от изброим списък указващ координатната система и два за самите компоненти (планарни (x, y) или полярни (r, phi));
	+ Добавете методи за транслация и ротация на вектора;
	+ Добавете метод, който прибавя/изважда друг вектор;
	+ Добавете метод, който връща точка получена от преместването на подадена точка с настоящия вектор;
* Създайте програма съдържаща класове за различни основни равнинни фигури и обемни тела:
	+ Добавете подходящите конструктори;
	+ Добавете полета описващи характеристиките на съответните фигури;
	+ Добавете методи за изчисляване на повърхнина и обем на тези, които имат такива;
	+ Такива класове могат да бъдат:
		- **Point2D** и **Vector2D** съответно за точка и вектор в двумерното пространство;
		- **Parallelogram** за успоредник, **Ellipse** за елипса;
		- **Point3D** и **Vector3D** съответно за точка и вектор в тримерното пространство;
		- **Cone** за конус, **Parallelepiped** за паралелепипед, **Pyramid** за пирамида, **Ellipsoid** за елипсоид.

## Допълнителни задачи

* Възползвайте от принципите за наследяване и капсулиране:
	+ Капсулирайте данните на всички обекти: **всички полета да бъдат частни** и промяната на стойностите им да се извършва през **конструктори** и **методи** (публични или защитени);
	+ Създайте базови класове **Object2D** (характеристики: *периметър* и *лице*) и **Object3D** (характеристики: *обем* и *пълна повърхнина*) и реализирайте изчисляването им на базата на специфичните характеристики наследени класове;
	+ Създайте колекции от базовите класове и реализирайте логика за въвеждане/извеждане на характеристиките им (използвайки upcasting и downcasting);
* Приложете наученото за свойства, индексатори и статични членове:
	+ Използвайте **свойства** и **индексатори** вместо методи, където е уместно, за да капсулирате обектите;
	+ Създайте **статични методи** за въвеждане на нов екземпляр на съответния клас от клавиатурата;
	+ Възползвайте се от **статичните полета, конструктори и свойства**, за да споделите информация между отделните екземпляри на един клас (например екземпляр на класа **Random**);
	+ Изнесете поведението, което не е обвързано с конкретен екземпляр на класа, в **статични методи**;
* Приложете наученото за виртуални методи и предефинирането им:
	+ Добавете презаписани конструктори за базовите класове **Object2D** и **Object3D**;
	+ Добавете нов базов клас **GeometryObject** съдържащ конструктор с параметър „наименование“ и виртуален метод за генериране на произволни параметри на обекта;
	+ Изведете общите за различните видове обекти виртуални методи в съответните базови класове;
	+ Създайте нов клас **GeometryStore**, в който да съхранявате геометрични обекти, индексатори за тях по номер и наименование, както и метод за добавяне на създаден вече обект;
* Приложете наученото за абстракция, интерфейси и шаблони:
	+ Създайте абстрактен базов клас **GeometryObject** с характеристика *име*, виртуален метод за въвеждане от клавиатурата и абстрактен метод за генериране на случайни стойности на характеристиките;
	+ Заменете класовете **Object2D** и **Object3D** с интерфейсите **IObject2D** със стойства *периметър* и *лице* и **IObject3D** със свойства *пълна повърхнина* и *обем*;
	+ Използвайте **List<GeometryObject>** за съхранение на списъка от геометрични обекти в класа **GeometryStore**;
	+ Реализирайте логика за извеждане само на равнинните фигури и само на тримерните тела (използвайки съответните интерфейси);
* Приложете наученото за изключения:
	+ В началото на конструкторите и методите с параметри, да се проверяват за коректност техните стойности и да се предизвикват съответните изключения;
	+ При въвеждане на данни за даден обект, да се проверява тяхната коректност и да се предизвикват различни по вид изключения според ситуацията;
	+ При изчисляване на параметрите за даден обект, да се проверява наличието на всички необходими данни и, ако липсват такива, да се предизвиква изключение;
	+ В основната програма, да се прихванат различните изключения, да се изведе разбираемо съобщение и да се реагира адекватно на различните видове изключения;

# Софтуерна академия

## Задача

Създайте програма за управление на софтуерна академия.

## Изисквания

* Създайте клас **Student**, който описва учащ;
* Създайте клас **Course**, който описва курс;
* Създайте клас **Academy**, който описва софтуерна академия;
* Класът **Student**:
	+ Съдържа пореден номер, имена и списък от курсове;
	+ Декларира метод за записване на курс, подаден като параметър;
* Класът **Course**:
	+ Съдържа наименование, e-mail и списък от записани учащи;
	+ Декларира метод за записване на учащ, подаден като параметър;
* Класът **Academy**:
	+ Съдържа наименование, списък от курсове и учащи;
	+ Декларира методи за добавяне на курс/учащ и записване.

# Библиотека

## Задача

Реализирайте приложение за регистриране и разглеждане на списък с произведения в библиотека.

## Изисквания

Създайте класове за следните видове произведения:

* **Печатно произведение** (*заглавие*; *език*; *издателство*):
	+ **Периодично печатно произведение** (*година*; *брой*):
		- **Вестник** (*вид*: ежедневник, седмичник, двуседмичник; *гл. редактор*; *водеща новина за броя*);
		- **Списание** (*тема*; *авторски колектив*; *описание на корицата на броя*);
	+ **Самостоятелно печатно произведение** (*дата на издаване*; *автор/автори*; *брой страници*):
		- **Научна статия** (*научна област*; *препоръчана литература*);
		- **Книга** (*номер на изданието*; *твърди/меки корици*; *наличие на илюстрации*):
			* **Художествена литаратура** (*жанр*; *целева аудитория*);
			* **Техническа литература** (*научна област*; *ниво на аудиторията*: начинаещи, напреднали, експерти).

## Допълнителни задачи

* Приложете наученото за свойства, индексатори и статични членове:
	+ Използвайте **свойства** и **индексатори** вместо методи, където е уместно, за да капсулирате обектите;
	+ Създайте **статични методи** за въвеждане на нов екземпляр на съответния клас от клавиатурата;
	+ Възползвайте се от **статичните полета, конструктори и свойства**, за да споделите информация между отделните екземпляри на един клас (например екземпляр на класа **Random**);
	+ Изнесете поведението, което не е обвързано с конкретен екземпляр на класа, в **статични методи**;
* Приложете наученото за виртуални методи и предефинирането им:
	+ Създайте нов конструктор на базовия клас на всички произведения, който да приема като параметър низ – **ISBN**. Добавете свойство за това колко бройки от произведението са налични за ползване;
	+ Изведете общите за различните видове произведения виртуални методи в съответните базови класове;
	+ Създайте клас **Library**, в който се съхраняват произведения, индексатори за тях по номер и **ISBN**, както и метод за добавяне;
	+ Добавете метод в класа **Library** за търсене на произведения;
	+ Добавете метод в класа **Library** за показване на информацията за подадено издание.

# Ролева игра

## Задача

Реализирайте походова конзолна ролева игра.

## Изисквания

* Играта се развива в **правоъгълна мрежа от символи**, подобно на *Златотърсачи*;
* Създайте **базов клас** за **единица (играч или чудовище)**, който съдържа информация за *координатите на единицата*, *точките живот* и *нанасяните от единицата щети*;
* Създайте **наследени класове** за **играч** и **различни видове чудовища**;
* В началото на играта, поставете играча и известен брой чудовища в игралното поле;
* Реализирайте **безкраен цикъл**, в който **всяка единица получава ход**, в който може да се **придвижва** или **атакува** противник, отнемайки му точки живот; чудовищата управлявайте **програмно**;

## Допълнителни задачи

* Добавете случаен елемент в играта (например случайно количество щети при нападение, случайно генериран терен на игралното поле, и/или различни характеристики за чудовища от един и същи вид);
* Добавете екипировка за играча;
* Приложете наученото за свойства, индексатори и статични членове:
	+ Използвайте **свойства** и **индексатори** вместо методи, където е уместно, за да капсулирате обектите;
	+ Създайте **статични методи** за въвеждане на нов екземпляр на съответния клас от клавиатурата;
	+ Възползвайте се от **статичните полета, конструктори и свойства**, за да споделите информация между отделните екземпляри на един клас (например екземпляр на класа **Random**);
	+ Изнесете поведението, което не е обвързано с конкретен екземпляр на класа, в **статични методи**;
* Приложете наученото за виртуални методи и предефинирането им;
* Приложете наученото за абстракция, интерфейси и шаблони;
* Реализирайте събития за важни ситуации в играта и походящи обработчици, които да извършват промени в поведението на програмата или да извеждат на екрана подходящи съобщения; примери за такива ситуации са:
	+ Раняване на чудовище;
	+ Раняване на играч;
	+ Смърт на чудовище;
	+ Смърт на играч;
	+ Опит за придвижване в неразрешена посока;
	+ Попадане в капан и т.н.

# Адресна книга

## Задача

Да се реализира адресна книга с информация за имената, адресите и телефоните на хора. Да се предостави възможност за добавяне на редове в края на книгата през конзолата, както и за извеждане на всички редове в конзолата.

## Изисквания

* Да се декларира клас AddressEntry, който ще представлява един ред от адресната книга.
* Да се реализират необходимите методи за въвеждане и извеждане на данните за реда от/в конзолата.
* Редовете в адресната книга да се съхраняват в масив.

## Допълнителни задачи

* Размерът на масива да се въвежда от потребителя в началото на програмата.
* Да се реализират опции за изтриване на ред от адресната книга и за вмъкване на ред на конкретно място в адресната книга.
* Да се реализира сортирането на редовете в адресната книга по азбучен ред на имената.
* **Ново:** да се замени масивът с шаблонен списък от обекти и да се преправят реализациите на операциите за добавяне и премахване на ред от книгата.

# Приоритетна опашка

## Задача

Реализирайте шаблонен тип за строго типизирана приоритетна опашка.

## Изисквания

* При добавяне на елемент в опашката се указва приоритет (цяло число); елементът се добавя непосредствено след последния елемент с по-висок или равен приоритет;
* От опашката се винаги се вади само челният елемент;
* Може да се прочете челният елемент, без да бъде изваден;
* Имплементирайте интерфейса **IEnumerable<T>**, като изброявате елементите в посока от челния елемент към края на опашката.

# Шаблонни алгоритми за сортиране

Реализирайте статични шаблонни методи за сортиране на аргумент от тип **IList<T>** чрез следните алгоритми:

* Метод на мехурчето;
* Метод на пряката селекция;
* Сортиране чрез вмъкване;
* Quicksort.

# Рационални числа

## Задача

Напишете програма, която декларира структура за рационално число със съответните аритметични операции и създава няколко пробни израза с рационални числа.

## Изисквания

* Структурата за рационално число да бъде с наименование **Rational**;
* Да се реализира конструктор с параметри числител и знаменател;
* Да се презапишат аритметичните операции събиране, изваждане, умножение и деление за параметри рационални числа;
* Да се презапишат преобразувания на типове както следва:
	+ неявно преобразуване от int към рационално число;
	+ неявно преобразуване от рационално число към **float**, **double** и **decimal**;
	+ явно преобразуване от рационално число към **int** (като се раздели целочислено числителя на знаменателя);
* Да се реализира метод за получаване на реципрочното на рационално число;
* Да се предефинират методите **ToString()** и **Equals()**;
* Да се имплементират интерфейсите **IComparable**, **IComparable<Rational>**, **IEquatable<Rational>** и **IConvertible**.

## Допълнителни задачи

* След всяка аритметична операция числителят и знаменателят на полученото рационално число да нямат общи делители.
* Да се реализира в главната функция на програмата цикъл, във всяка итерация на който се избира от потребителя операция (събиране, изваждане, умножение, деление, намиране на реципрочно), въвеждат се необходимия брой цели или рационални числа и се извежда резултатът. Рационалните числа се въвеждат под формата на две цели числа, разделени с наклонена черта. Изпълнението на цикъла да приключи при въвеждане на празен низ при избора на операция.

# Аритметични изрази

## Задача

Напишете програма, която декларира класове за аритметични изрази, които съхраняват в себе си информацията за аритметичен израз и предоставят методи за извеждането му на екрана и пресмятане на стойността му.

## Изисквания

* Да се реализират базов клас за аритметичен израз и три наследени класа за константа, унарна операция и бинарна операция.
* Аритметичните операции, които се разглеждат, са събиране, изваждане, умножение, деление и намиране на противоположното число (унарен минус).
* Константите и стойностите на изразите да бъдат от тип int.
* Да се презапишат съответните оператори за създадените класове.
* Да се направи виртуален метод за пресмятане на стойността на израз.
* Да се предефинира методът ToString() така, че да извежда представяне на аритметичния израз в символен низ.
* Да не се използва преобразуване на типове надолу по йерархията (downcasting).
* Да се създадат няколко аритметични израза и да се изпечатат на екрана заедно със стойностите си.

## Допълнителни задачи

* Да се създаде интерфейс за аритметичен израз и с него да се замени базовия клас на йерархията.
* Да се променят класовете на шаблонни класове с един типов параметър, който определя типа на стойностите на аритметичните изрази.
* Да се реализира възможност за въвеждане на израз от конзолата, пресмятането на стойността му и извеждането й на екрана (прост калкулатор).

# Оказион

## Задача

Да се напише програма, която реализира оказион с обяви за продажба на артикули чрез търг или директно закупуване. Да се поддържа списък от потребители и списък от обявени за продан артикули. Да се предостави възможност за вписване и отписване от системата с потребител; регистриране на обява за продан на артикул; разглеждане на активните, регистрираните и спечелените обяви; наддаване по търг; директно закупуване на артикул.

## Изисквания

* Да се използва обектно-ориентирано програмиране с подходящи класове, наследяване и интерфейси.
* Да се спази принципът за капсулиране.
* Изключенията в програмата да бъдат обработвани по подходящ начин, а при въвеждането на данни от клавиатурата да се осигурява валидността им.
* Данните да се съхраняват в оперативната памет.
* Всеки потребител се характеризира с уникално потребителско име.
* Всяка обява се характеризира със състояние (активна/прикючена), дата на регистриране, описание на артикула и потребител, който я е регистрирал.
* Приключените обяви се характеризират и с дата на приключване и потребител, който я е спечелил.
* Обявите за директна продажба се характеризират с фиксирана цена за закупуване.
* Обявите за търг се характеризират с начална цена на търга и краен срок.
* В програмата да се реализира главно меню със следните опции:
	+ Вписване в системата;
	+ Разглеждане на списъка с активни обяви;
	+ Разглеждане на списъка с обяви, регистрирани от вписания потребител (изисква вписан потребител);
	+ Разглеждане на списъка от приключени обяви, които вписаният потребител е спечелил (изисква вписан потребител);
	+ Наддаване по търг (изисква вписан потребител);
	+ Директно закупуване на артикул (изисква вписан потребител);
	+ Отписване на текущия потребител от системата (изисква вписан потребител);
	+ Край на работата с програмата.

## Допълнителни задачи

* Да се реализира в главното меню опция за регистриране на потребител с парола, като вписването в системата изисква съответната парола.
* Да се реализира зареждане на списъците с потребители и обяви от текстови файлове с фиксирани имена и подходящ формат за съхранение на данните, както и записването на данните от оперативната памет обратно в тези файлове при приключване на работата с програмата.
* За обявите за търг да се реализира максимална цена на търга, при която предложилият я потребител автоматично печели търга преди изтичането на крайния му срок.
* Да се използват подходящи събития в реализацията на програмата.

# Информация за разработчици

## Задача

Създайте C# библиотека, която да декларира атрибут за информация за разработчик, разработил определена програмна единица.

## Изисквания

* Декларирайте Атрибута **DeveloperInfoAttribute**, който съдържа информация за име, e-мейл адрес и телефон за контакт на разработчика.
* Името и е-мейл адресът са задължителни, докато телефонният номер не е.
* Атрибутът може да се прилага повече от веднъж върху всякакви програмни единици.
* Поставете така направения атрибут на всички възможни програмни единици, като попълните информация за поне няколко разработчика.

## Допълнителни задачи

* Създайте аналитично приложение, което да:
	+ Изчита от клавиатурата наименованието на \*.dll или \*.exe файл;
	+ Опитва да зареди файла като assembly;
	+ Има добавена референция към проекта, в който е деклариран атрибута **DeveloperInfoAttribute**;
	+ Намира програмните единици, които имат атрибут **DeveloperInfoAttribute** и извежда информацията от атрибута;
	+ Има добавена референция към проекта „Геометрични фигури“;
	+ Намира всички класове наследяващи класа **Object2D**, извежда тяхното наименование и техните **DeveloperInfoAttribute** атрибути, ако имат такива (иначе надпис „No developer info for this class.”).

# Калкулатор с използване на делегати

## Задача

Създайте конзолно приложение – калкулатор, като използвате подходящи делегати и делегатни екземпляри за реализиране на различните аритметични операции.

## Допълнителни задачи

* Преработете конзолния калкулатор в приложение с графичен потребителски интерфейс и поведение, което е сходно на вградения в Windows калкулатор.