**Глава 4.1. По-сложни проверки**

В **настоящата** глава ще разгледаме **вложените проверки** в езика **C#**, чрез които нашата програма може да съдържа **условни конструкции**, в които има **вложени други условни конструкции**. Наричаме ги **"вложени"**, защото **поставяме if конструкция** в **друга if конструкция**. Ще разгледаме и **по-сложни логически условия** с подходящи примери.

* За **главната форма (Form)**, която съдържа всички контроли:
  + (name) = **FormPointAndRectangle**
  + age = **18**
* “**Mr.**” – мъж (пол “**m**”) на 16 или повече години.
* “**Master**” – момче (пол “**m**”) под 16 години.

**Втори** параграф. Ще разгледаме и **по-сложни логически условия** с подходящи примери. Ще разгледаме и **по-сложни логически условия** с подходящи примери.

**Трети** параграф. Ще разгледаме и **по-сложни логически условия** с подходящи примери. Ще разгледаме и **по-сложни логически условия** с подходящи примери.

**Видео**

Гледайте видео-урок по тази глава тук: <https://www.youtube.com/watch?v=z8XxYIyesz0>.

**Тестване на решението**

Тествайте решението си тук: <https://judge.softuni.bg/Contests/Practice/Index/508#7>

|  |  |
| --- | --- |
| https://csharp-book.softuni.bg/assets/alert-icon.png | **Добра практика** е да използваме **блокове**, които **заграждаме** с къдрави скоби **{ }** след **if** и **else**. Също така, препоръчително e при писане да **отместваме** кода **след if и else** с една табулация **навътре**, за да направим кода по-лесно четим. |

**Вложени проверки**

Доста често програмната логика налага използването на **if** или **if-else** конструкции, които се съдържат една в друга. Те биват наричани **вложени** **if** или **if-else** конструкции. Както се подразбира от названието **"вложени"**, това са **if** или **if-else** конструкции, които са поставени в други **if** или **else** конструкции.

if (condition1)

{

if (condition2)

{

// тяло;

}

else

{

// тяло;

}

}

Влагането на повече от три условни конструкции една в друга не се счита за добра практика и трябва да се избягва, най-вече чрез оптимизиране на структурата/алгоритъма на кода и/или чрез използването на друг вид условна конструкция, който ще разгледаме по-надолу в тази глава.

**Пример: обръщение според възраст и пол**

Според въведени **възраст** (десетично число) и **пол** (**m** / **f**) да се отпечата обръщение:

* “**Mr.**” – мъж (пол “**m**”) на 16 или повече години.
* “**Master**” – момче (пол “**m**”) под 16 години.
* “**Ms.**” – жена (пол “**f**”) на 16 или повече години.
* “**Miss**” – момиче (пол “**f**”) под 16 години.

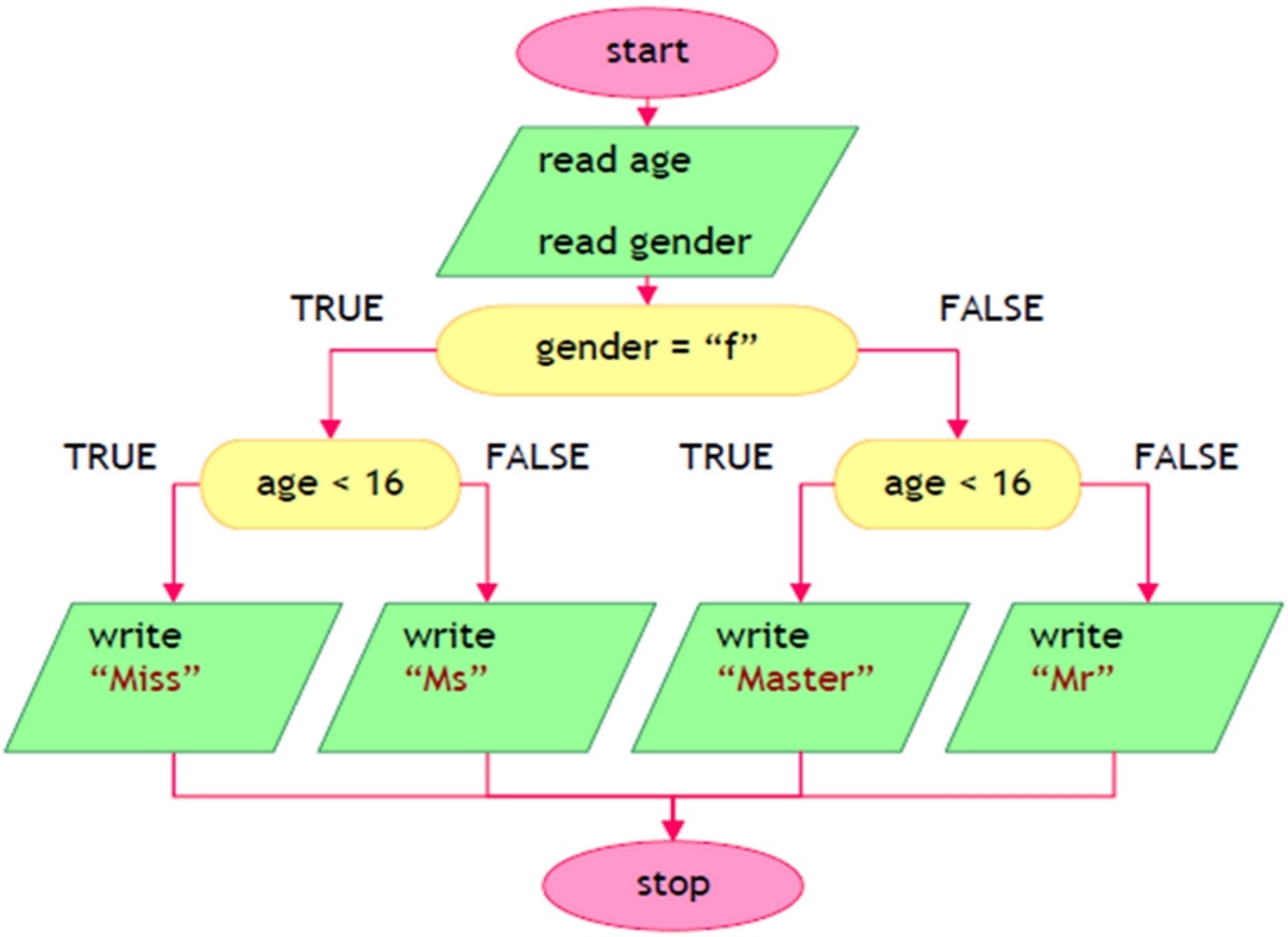
**Примерен вход и изход**

| **Вход** | **Изход** | **Вход** | **Изход** |
| --- | --- | --- | --- |
| 12 f | Miss | 17 m | Mr. |

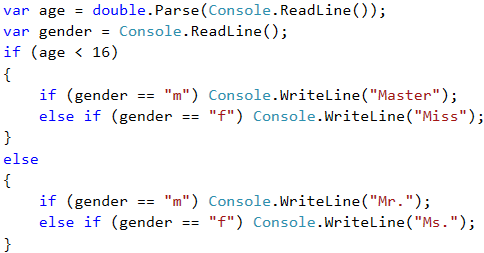
| **Вход** | **Изход** | **Вход** | **Изход** |
| --- | --- | --- | --- |
| 25 f | Ms. | 13.5 m | Master |

**Решение**

Можем да забележим, че **изходът** на програмата **зависи от няколко неща**. **Първо** трябва да проверим какъв **пол** е въведен и **после** да проверим **възрастта**. Съответно ще използваме **няколко** **if-else** блока. Тези блокове ще бъдат **вложени**, т.е. от **резултата** на първия ще **определим** кои от **другите** да изпълним.



След прочитане на входните данни от конзолата ще трябва да се изпълни следната примерна програмна логика:



**Тестване на решението:**

Тествайте решението си тук: <https://judge.softuni.bg/Contests/Practice/Index/508#0>

**Пример: квартално магазинче**

Предприемчив българин отваря по едно **квартално магазинче** в **няколко града** с различни **цени** за следните **продукти**:

| **продукт / град** | **Sofia** | **Plovdiv** | **Varna** |
| --- | --- | --- | --- |
| coffee water beer sweets peanuts | 0.50 0.80 1.20 1.45 1.60 | 0.40 0.70 1.15 1.30 1.50 | 0.45 0.70 1.10 1.35 1.55 |

По даден **град** (стринг), **продукт** (стринг) и **количество** (десетично число) да се пресметне цената.

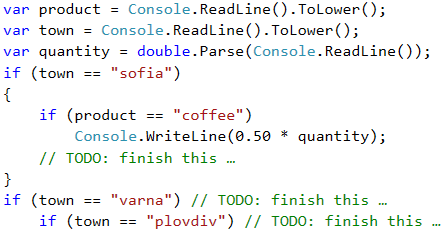
**Примерен вход и изход**

| **Вход** | **Изход** | **Вход** | **Изход** |
| --- | --- | --- | --- |
| coffee Varna 2 | 0.9 | peanuts Plovidv 1 | 1.5 |

| **Вход** | **Изход** | **Вход** | **Изход** |
| --- | --- | --- | --- |
| beer Sofia 6 | 7.2 | water Plovdiv 3 | 2.1 |

**Решение**

**Прехвърляме** всички букви в **долен регистър** с функцията **.ToLower()**, за да сравняваме продукти и градове **без значение** от малки/главни букви.



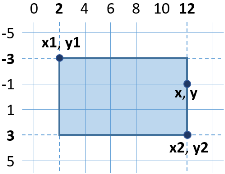
**Тестване на решението:**

Тествайте решението си тук: <https://judge.softuni.bg/Contests/Practice/Index/508#1>

**По-сложни логически условия**

**Пример: точка върху страна на правоъгълник**

Да се напише програма, която проверява дали **точка {x, y}** се намира **върху някоя от страните на правоъгълник {x1, y1} - {x2, y2}**. Входните данни се четат от конзолата и се състоят от 6 реда: десетичните числа **x1**, **y1**, **x2**, **y2**, **x** и **y** (като се гарантира, че **x1 < x2** и **y1 < y2**). Да се отпечата "**Border**" (точката лежи на някоя от страните) или "**Inside / Outside**" (в противен случай).



**Примерен вход и изход**

| **Вход** | **Изход** | **Вход** | **Изход** |
| --- | --- | --- | --- |
| 2 -3 12 3 12 -1 | Border | 2 -3 12 3 8 -1 | Inside / Outside |

**Насоки и подсказки**

**Обработка на входните данни**

Прочитайки внимателно условието разбираме, че очакваме **два** реда с входни данни. Първият параметър е **реално число**, за което е добре да изберем подходящ тип на променливата. За по-голяма точност в изчисленията ще се спрем на **decimal** като тип за бюджета, а за сезона - **string**.

https://csharp-book.softuni.bg/assets/chapter-4-2-images/02.Trip-01.png

|  |  |
| --- | --- |
| https://csharp-book.softuni.bg/assets/alert-icon.png | Винаги преценявайте какъв тип стойност се подава при входните данни, както и към какъв тип трябва да бъдат конвертирани тези данни, за да работят правилно създадените от вас програмни конструкции! |

**Пример**: Когато в задачата е необходимо да направите парични изчисления, използвайте **decimal** за по-голяма точност.

**Изчисления**

Нека си създадем и инициализираме нужните за логиката и изчисленията променливи.

https://csharp-book.softuni.bg/assets/chapter-4-2-images/02.Trip-02.png

Подобно на примера в предната задача, можем да инициализираме променливите с някои от изходните резултати - с цел спестяване на допълнително инициализиране.

Разглеждайки отново условието на задачата забелязваме, че основното разпределение за това къде ще почиваме се определя от **стойността на подадения бюджет**, т.е. основната ни логика се разделя на два случая:

* Ако бюджетът е **по-малък** от дадена стойност.
* Ако е **по-малък** от друга стойност, или е **повече** от дадена гранична стойност.

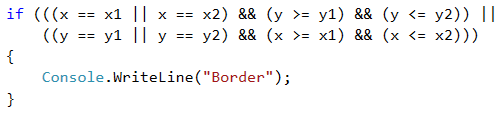
Спрямо това как си подредим логическата схема (в какъв ред ще обхождаме граничните стойности), ще имаме повече или по-малко проверки в условията. **Помислете защо!**

След това е необходимо да направим проверка за стойността на **подадения сезон**. Спрямо нея ще определим какъв процент от бюджета ще бъде похарчен, както и къде ще почива програмистът - в **хотел** или на **къмпинг**.

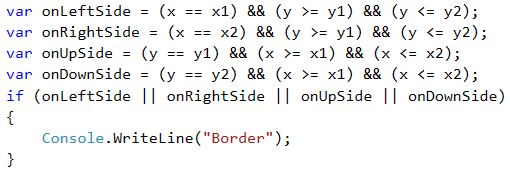
**Решение**

Точка лежи върху някоя от страните на правоъгълник, ако:

* x съвпада с x1 или x2 и същевременно y е между y1 и y2 или
* y съвпада с y1 или y2 и същевременно x е между x1 и x2



Предходната проверка може да се опрости по този начин:



**Тестване на решението**

Тествайте решението си тук: <https://judge.softuni.bg/Contests/Practice/Index/508#5>

**Какво научихме от тази глава?**

**Вложени проверки:**

if (condition1)

{

if (condition2)

// тяло;

else

// тяло;

}

**По-сложни проверки с &&, ||, ! и ():**

if ((x == left || x == right) && y >= top && y <= bottom)

Console.WriteLine(...);

**Switch-case проверки**

switch (селектор)

{

case стойност1:

конструкция;

break;

case стойност2:

case стойност3:

конструкция;

break;

....

default:

конструкция;

break;

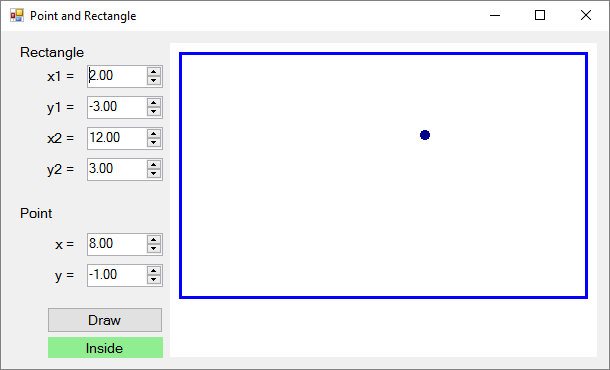
}

**Упражнения: графично приложения с по-сложни проверки**

В тази глава научихме как можем да правим **проверки с нетривиални условия**. Нека сега приложим тези знания, за да създадем нещо интересно: настолно приложение, което визуализира точка и правоъгълник. Това е прекрасна визуализация за една от задачите от упражненията.

**Задача: \* точка и правоъгълник – графично (GUI) приложение**

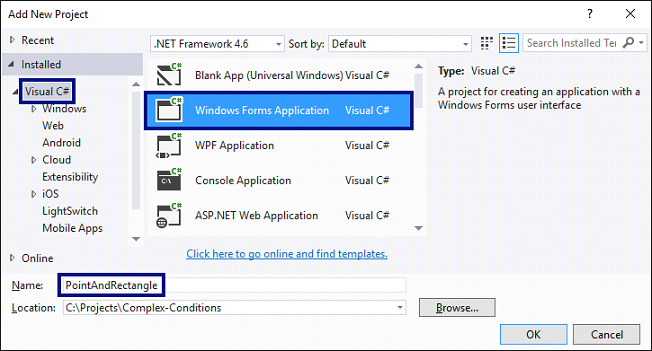
Задачата, която си поставяме е да се разработи графично (**GUI**) приложение за **визуализация на точка и правоъгълник**. Приложението трябва да изглежда приблизително по следния начин:



От контролите вляво се задават координатите на **два от ъглите на правоъгълник** (десетични числа) и координатите на **точка**. Приложението **визуализира графично** правоъгълника и точката и изписва дали точката е **вътре** в правоъгълника (**Inside**), **вън** от него (**Outside**) или на някоя от стените му (**Border**). Приложението **премества и мащабира** координатите на правоъгълника и точката, за да бъдат максимално големи, но да се събират в полето за визуализация в дясната страна на приложението.

|  |  |
| --- | --- |
| https://csharp-book.softuni.bg/assets/alert-icon.png | Внимание: това приложение е значително **по-сложно** от предходните графични приложения, които разработвахме до сега, защото изисква ползване на функции за чертане и нетривиални изчисления за преоразмеряване и преместване на правоъгълника и точката. Следват инструкции за изграждане на приложението стъпка по стъпка. |

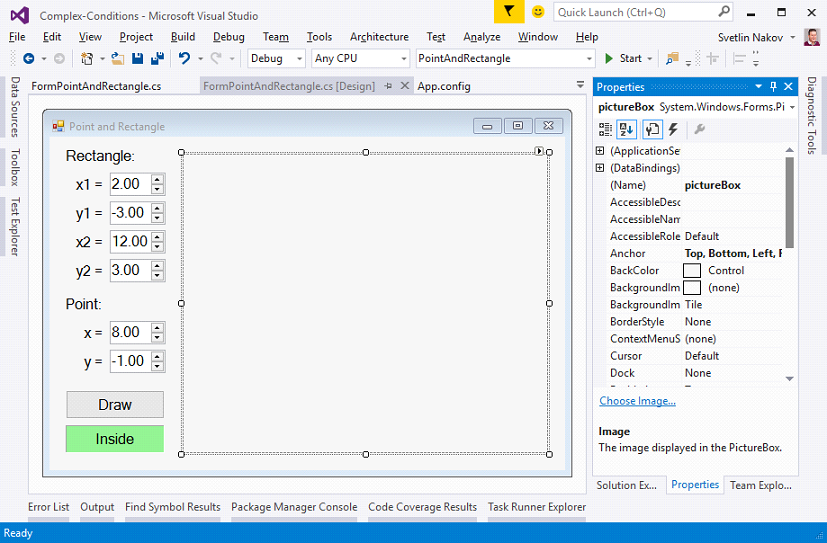
Създаваме нов проект **Windows Forms Application** с подходящо име, например “Point-and-Rectangle”:



**Нареждаме контролите** във формата, както е показано на фигурата по-долу:

* 6 кутийки за въвеждане на число (**NumericUpDown**).
* Надписи (**Label**) пред всяка кутийка за въвеждане на число.
* Бутон (**Button**) за изчертаване на правоъгълника и точката.
* Текстов блок за резултата (**Label**).

Нагласяме **размерите** и **свойствата** на контролите, за да изглеждат приблизително като на картинката:



Задаваме следните препоръчителни настройки на контролите:

* За **главната форма (Form)**, която съдържа всички контроли:
  + (name) = **FormPointAndRectangle**
  + **Text** = **Point and Rectangle**
  + **Font.Size** = **12**
  + **Size** = **700**, **410**
  + **MinimumSize** = **500**, **400**
  + **FormBorderStyle** = **FixedSingle**
* За **полетата за въвеждане на число (NumericUpDown)**:
  + (name) = **numericUpDownX1**; **numericUpDownY1**; **numericUpDownX2**; **numericUpDownY2**; **numericUpDownX**; **numericUpDownY**
  + **Value** = **2**; **-3**; **12**; **3**; **8**; **-1**
  + **Minimum** = **-100000**
  + **Maximum** = **100000**
  + **DecimalPlaces** = **2**
* За **бутона (Button)** за **визуализация** на правоъгълника и точката:
  + (name) = **buttonDraw**
  + **Text** = **Draw**
* За **текстовия блок за резултата (Label)**:
  + (name) = **labelLocation**
  + **AutoSize** = **False**
  + **BackColor** = **PaleGreen**
  + **TextAlign** = **MiddleCenter**
* За **полето с чертежа (PictureBox)**:
  + (name) = **pictureBox**
  + **Anchor** = **Top**, **Bottom**, **Left**, **Right**

Следва да хванем следните **събития**, за да напишем C# кода, който ще се изпълни при настъпването им:

* Събитието **Click** на бутона **buttonDraw** (извиква се при натискане на бутона).
* Събитието **ValueChanged** на контролите за въвеждане на числа **numericUpDownX1**, **numericUpDownY1**, **numericUpDownX2**, **numericUpDownY2**, **numericUpDownX** и **numericUpDownY** (извиква се при промяна на стойността в контролата за въвеждане на число).
* Събитието **Load** на формата **FormPointAndRectangle** (извиква се при стартиране на приложението, преди да се появи главната форма на екрана).
* Събитието **Resize** на формата **FormPointAndRectangle** (извиква се при промяна на размера на главната формата).

Всички изброени по-горе събития ще изпълняват едно и също действие – **Draw()**, което ще визуализира правоъгълника и точката и ще показва дали тя е вътре, вън или на някоя от страните. Кодът трябва да изглежда така:

private void buttonDraw\_Click(object sender, EventArgs e)

{

Draw();

}

private void FormPointAndRectangle\_Load(object sender, EventArgs e)

{

Draw();

}

private void FormPointAndRectangle\_Resize(object sender, EventArgs e)

{

Draw();

}

private void numericUpDownX1\_ValueChanged(object sender, EventArgs e)

{

Draw();

}

/\*TODO: implement the same way event handlers numericUpDownY1\_ValueChanged,

numericUpDownX2\_ValueChanged, numericUpDownY2\_ValueChanged,

numericUpDownX\_ValueChanged and

numericUpDownY\_ValueChanged\*/

private void Draw()

{

// TODO: implement this a bit later …

}

Нека започнем от по-лесната част: **печат на информация къде е точката спрямо правоъгълника** (Inside, Outside или Border).

**Стартираме приложението** и го **тестваме**. Въвеждаме различни данни, за да видим дали се държи коректно.

Ако имате проблеми с примерния проект по-горе, **гледайте видеото** в началото на тази глава. Там приложението е направено на живо стъпка по стъпка с много обяснения. Или питайте във **форума на СофтУни**: <https://softuni.bg/forum>.

**Глава 4.2. По-сложни проверки - изпитни задачи**

В предходната глава се запознахме с вложените условни конструкции в езика C#. Чрез тях програмната логика в дадена програма може да бъде представена посредством **if конструкции**, които се съдържат една в друга. Разгледахме и условната конструкция **switch-case**, която позволява избор измежду списък от възможности. Следва да упражним и затвърдим наученото досега, като разгледаме няколко по-сложни задачи, давани на изпити. Преди да преминем към задачите, ще си припомним условните конструкции:

**Вложени проверки**

if (condition1)

{

if (condition2)

// тяло;

else

// тяло;

}

|  |  |
| --- | --- |
| https://csharp-book.softuni.bg/assets/alert-icon.png | Не е добра практика нивото на влагане да бъде повече от три, тоест не трябва да бъдат влагани повече от три условни конструкции една в друга. |

**Switch-case проверки**

Когато работата на програмата ни зависи от стойността на една променлива, вместо да правим последователни проверки с множество **if-else** блокове, можем да използваме условната конструкция **switch-case**.