



**ПРОФЕСИОНАЛНА ГИМНАЗИЯ**  
**ПО ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА “АПОСТОЛ АРНАУДОВ”**  
гр. Русе, ул. “Потсдам” № 3; п.к. 7005, тел. 082/84-60-96; e-mail: [info-1806301@edu.mon.bg](mailto:info-1806301@edu.mon.bg)

---

# ДИПЛОМЕН ПРОЕКТ

**ЗА ПРИДОБИВАНЕ НА**

**ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ**

**по професия код 481030 „Приложен програмист“**  
**специалност код 4810301 „Приложно програмиране“**

## **ТЕМА:**

# **РОЛЯТА НА МАТЕМАТИКАТА В ПРОГРАМИРАНЕТО**

**Ученик:** Даниел Ивайлов Стоянов

**Ръководител-консултант:** Милена Дамесова-Христова

гр. Русе

2025

# СЪДЪРЖАНИЕ

СЪДЪРЖАНИЕ .....	2
ВЪВЕДЕНИЕ .....	4
ГЛАВА ПЪРВА .....	6
ЗНАЧЕНИЕ И ПРИЛОЖЕНИЕ НА МАТЕМАТИКАТА В ПРОГРАМИРАНЕТО .....	6
1.1.    Въведение в програмирането .....	6
1.2.    Каква е разликата между Програмист, Софтуерен инженер и Софтуерен разработчик? .....	8
1.2.1.    Какво е програмист? .....	8
1.2.2.    Какво е софтуерен инженер? .....	8
1.2.3.    Какво е софтуерен разработчик? .....	9
1.3.    Разлика между различните програмните езици .....	9
1.4.    Въведение в математиката .....	10
1.5.    Връзката между математиката и програмирането .....	11
1.6.    Приложение на математиката в програмирането .....	11
1.6.1.    Алгоритми и структури от данни .....	11
1.6.2.    Логика и Булева алгебра .....	11
1.6.3.    Анализ на сложност и ефективност .....	11
1.6.4.    Линейна алгебра и компютърна графика .....	12
1.6.5.    Математически анализ и симулации .....	12
1.6.6.    Теория на числата и криптография .....	12
1.6.7.    Вероятности и статистика .....	12
ГЛАВА ВТОРА .....	13
БРОЙНИ СИСТЕМИ. ПОБИТОВИ ОПЕРАЦИИ, ТЕОРИЯ НА МНОЖЕСТВАТА. КОМБИНАТОРИКА .....	13
2.1. Бройни системи .....	13
2.2. Видове бройни системи .....	13
2.2.1.    Десетична бройна система (основа 10) .....	13
2.2.2.    Двоична бройна система (основа 2) .....	14
2.2.3.    Осмична бройна система (основа 8) .....	14
2.2.4.    Шестнадесетична бройна система (основа 16) .....	14
2.2.5.    Троична бройна система (основа 3) .....	14
2.2.6.    Петична бройна система (основа 5) .....	15
2.3.    Преобразуване на броични системи .....	15
2.3.1.    Двоична в десетична .....	15

2.3.2.	Десетична в двоична .....	15
2.3.3.	Двоична в шестнадесетична и обратно .....	16
2.4.	Побитови операции.....	17
2.4.1.	Оператор за побитово отрицание (NOT).....	17
2.4.2.	Побитово И (AND) .....	17
2.4.3.	Побитово ИЛИ (OR).....	18
2.4.4.	Побитово Изключващо ИЛИ (XOR).....	19
2.5.	Теория на множествата.....	20
2.5.1.	Основни концепции.....	20
2.6.	Комбинаторика.....	21
2.6.1.	Основни правила на комбинаториката .....	21
2.6.2.	Пермутации .....	22
2.6.3.	Вариации .....	22
2.6.4.	Комбинации .....	22
ГЛАВА ТРЕТА .....		23
ПРИЛОЖЕНИЕ – КОНВЕРТОР МЕЖДУ БРОЙНИ СИСТЕМИ .....		23
3.1.	Какво е C# и за какво е използван в приложението? .....	23
3.1.1.	Видове променливи използвани в приложението .....	23
3.1.2.	Цикли в C#.....	24
3.2.	WinForms .....	24
3.2.1.	Архитектура .....	24
3.2.2.	Характеристики.....	25
3.3.	Функционалност на приложението – ConverterApp .....	25
3.3.1.	Изглед на приложението при стартиране.....	25
3.3.2.	Функционалност и разбиране на ConverterApp .....	26
3.4.	Тестване на приложението с реални данни .....	31
ЗАКЛЮЧЕНИЕ.....		33
ИЗПОЛЗВАНА ЛИТЕРАТУРА.....		34

## ВЪВЕДЕНИЕ

Математиката стои в основата на програмирането, тъй като осигурява логическата и аналитичната рамка, необходима за разработването на алгоритми, анализ на данни и оптимизация на изчисленията. Без математически концепции програмирането би било значително по-сложно, а създаването на ефективни софтуерни решения – почти невъзможно.

Една от основните връзки между математиката и програмирането е концепцията за алгоритмите. Алгоритмите представляват набор от инструкции за решаване на даден проблем, а тяхното създаване изисква математическо мислене и логически анализ. Дискретната математика, включително логиката, комбинаториката и теорията на графите, играе ключова роля при структурирането на алгоритми и разработката на ефективен код.

Математиката също е в основата на структурите от данни, които са неизменна част от програмирането. Масиви, списъци, дървета и хеш таблици са математически модели, които позволяват съхранение и бързо извличане на информация. Алгоритми като двоично търсене, сортиране и динамично програмиране разчитат на математически анализ за оценка на тяхната сложност и ефективност.

Теорията на сложността, базирана на математиката, позволява на програмистите да оценяват и сравняват алгоритми по отношение на времева и пространствена сложност. Оптимизацията на кода, особено в големи софтуерни системи, зависи от познания по линейна алгебра, числени методи и вероятностна теория.

Математиката е от съществено значение в различни области на програмирането. В областта на машинното обучение и изкуствения интелект се използват вероятностни модели, статистически анализ и линейна алгебра. Компютърната графика разчита на геометрия и матрични трансформации. Криптографията, която гарантира сигурността на данните, е базирана на теории от областта на числовите системи и абстрактната алгебра.

Развитието на съвременните технологии и софтуерни решения е неразривно свързано с математиката. Познанията по математика не само улесняват програмирането, но и допринасят за по-ефективното и надеждно разработване на

алгоритми и приложения. Следователно, разбирането на математическите основи е критично за всеки успешен софтуерен инженер.

## ГЛАВА ПЪРВА

### ЗНАЧЕНИЕ И ПРИЛОЖЕНИЕ НА МАТЕМАТИКАТА В ПРОГРАМИРАНЕТО

#### 1.1. Въведение в програмирането

Програмирането е процесът на създаване на компютърни програми – програмен код или алгоритъм, който може да бъде разчетен и изпълнен от компютър. Програмиране се наричат също академичната дисциплина и компютърната наука – дял от математиката и информатиката – които се занимават с алгоритмите и методите и средствата на програмирането.

```
private void buttonCalculate_Click
(object sender, EventArgs e)
{
    try
    {
        var num1 = decimal.Parse(this.textBox1.Text);
        var num2 = decimal.Parse(this.textBox2.Text);
        var sum = num1 + num2;
        textBoxSum.Text = sum.ToString();
    }
    catch (Exception)
    {
        textBoxSum.Text = "error";
    }
}
```

*Снимка 1: Прост код написан на програмния език C#*

При програмирането се използват различни езици за програмиране, на които се записват изходните кодове на програмата. Тези езици имат различно предназначение, като някои от тях са езици на ниско, други на високо ниво, някои са уеб езици за програмиране, някои са скриптови, други функционални или процедурни, както и такива, които включват работа с обекти – обектно ориентирани езици за програмиране. Алгоритмите се делят най-общо на прости, сложни, и такива които изискват теореми или създаването на нови теореми (в математиката). Алгоритмите са специфична област на изучаване в програмирането и компютърните науки, в математиката, както и за оптимизацията на програмния код и добрата работа на процесора.

Програмирането включва писането на код, тестването му и поддържането на изходния код. Програмата е изпълнима от компютъра, но за да бъде полезна за потребителите, тя трябва да отговаря на поставените изисквания към софтуера. За разлика от системния софтуер,

софтуерът за потребителите има други етапи на създаване, като се отделя внимание на функционалните и потребителски изисквания („Какво иска потребителят?“). Потребителят може да е голям клиент или множество по-малки клиенти, или дори фирма, заявила разработка на софтуер, затова във всеки конкретен случай се дефинират софтуерни изисквания (requirements) според нуждите на потребителите. След това се пристъпва към най-общото разграфяване по етапи и дизайн като решение на софтуерно програмния процес: от една страна, самият софтуер се проектира най-общо като потребителски дизайн и интерфейс, от друга страна, се разпределят неговите функционалности на изпълнението за написване като код, в необходимата последователност.

Програмният код, който се пише на даден език за програмиране, се пише на базата на избраните според изискванията и спецификациите алгоритмични параметри, той може да бъде и модификация на съществуващ вече код или да бъде базиран на код на друг език, с цел решаването на даден математически проблем или програмно изискване. Отделно готовият софтуерен продукт подлежи на тестване на потребителския интерфейс. Макар че обикновено се възприема, че софтуерният продукт се тества от тестови екип по осигуряване на качеството, в действителност това е основното тестване на даден софтуер или програмен продукт, това може и да е уебсайт или конкретна онлайн функционалност към уебсайт, а основното тестване се прави от софтуерния мениджърски екип и дори от топ мениджъра, които проверяват дали писаното като код и тествано от тестовия екип е действително работещо. Компании от софтуерната индустрия, които се опитват да прехвърлят основното тестване към тестващия екип в действителност възлагат мениджърска работа и отговорности към тестващите, като отказът от този тип отговорност от страна на софтуерния мениджмънт понякога става причина такива компании да излязат от пазара или се налага да се реструктурират, независимо колко изгодно на пръв поглед изглежда да се прехвърля отговорното тестване, което предхожда пускането на софтуера към клиентите на тестовия екип.

В действителност най-солидните компании като Майкрософт например разчитат на код, писан и от самия основател на компанията, така че може да се каже, че тестването на софтуер за клиенти е мениджърски минимум, който разбира се не отменя работата на софтуерни или тестови екипи. В този смисъл от значително предимство е основателят или президентът (дори това да е жена) на една софтуерна или програмистка компания да е програмист, да може да програмира на практика, да може да прави софтуер, да има личен опит в създаването на софтуерни продукти, да разбира значението на това един софтуер да е работещ и да е тестван, преди да бъде подаден на потребителите, да е програмист или поне инженер в някоя близка област, която да му дава възможност да разбира значението на термини като „изисквания“ и „спецификации“, а не да ги поставя в полето за неясни софтуерни термини, това са термини от инженерните науки като цяло.

В сферата на софтуерното инженерство, програмирането е основната част от процеса на разработка на софтуер, но създаването на стабилен работещ софтуер изисква много повече от способността да се пише код.

В някои специализирани приложения или специфични ситуации, програма може да бъде написана или модифицирана чрез директно зареждане на нужните инструкции на машинен код и тяхното изпълнение.

## **1.2. Каква е разликата между Програмист, Софтуерен инженер и Софтуерен разработчик?**

### **1.2.1. Какво е програмист?**

Програмистът е действителен специалист по писането на предимно нов код. Включително ползването на различни известни алгоритми или при необходимост създаването на такива. Когато пише алгоритмите за една програма програмистът е и математик.

Към него спадат следните задачи:

- Писане на код
- Писане на алгоритми

### **1.2.2. Какво е софтуерен инженер?**

Софтуерният инженер може да пише компютърни програми, както и програмистът, но неговата задача е по-различна: да разпредели възложените за изпълнение задачи върху няколко екипа или дори множество екипи. Като той може да разпределя времето между отделните екипи и в някаква степен да координира работата им или да работи в рамките на един екип.

Работата на софтуерния инженер обикновено включва:

- Анализ на изискванията – анализът на изискванията е процес, който се извършва от самите програмисти, или от екип софтуерни инженери и други специалисти, участващи в процеса, на които е възложено изпълнение на софтуер, без да имат пряка връзка с потребителите, тоест на тях са зададени писани изисквания, които трябва съответно да бъдат „разчетени“
- Спецификация – инженерните спецификации се отнасят до параметрите, които трябва да отговарят на съществуващите стандарти
- Софтуерен дизайн или архитектура на софтуера
- Писане на код
- Компилиране на код
- Тестване
- Софтуерна документация
- Интеграция на софтуер (пример интеграция на Текстови редактор с останалите програми от офис пакет)
- Основна поддръжка



### 1.2.3. Какво е софтуерен разработчик?

Софтуерният инженер възлага на софтуерния разработчик на базата на съществуващ код и съществуващ софтуер да допълва и доизгражда дадена програма и софтуер.

## 1.3. Разлика между различните програмните езици

Различните езици за програмиране поддържат различни стилове на програмиране (парадигма на програмиране). Тези парадигми са свързани с областите на приложение на писания програмен код и софтуерен продукт.

Например езиците за програмиране на добър софтуер, включително на операционни системи са C и C++, те отговарят едновременно за добрата работа на програмите с процесора и харддиска, тоест възможността за запаметяване на потребителска входова информация върху диска, при добра работа и функциониране на процесора и на цялата система, тук се отнася и въпросът за съвместимостта, тоест някои програми се пишат специфично за някои системи, за да могат да работят най-добре върху тях. Друг възможен подход, при желание за бързо написване и възможност за портативност, тоест преносимост, на програмата, това е Java, този език позволява програмата да се стартира и като приложение и онлайн, но това не означава много бърза работа.

Модерното програмиране по неясни причини разчита в много голяма степен на огромните възможности на съвременния хардуер до степен, че игнорира тези базови езици и пише портативни програми и приложения буквално на какво ли не, включително и операционни системи, такива се пишат на php, на други скриптови и онлайн езици, на спесици от парадигми, и въобще непрофесионално програмиране, което разчита на работата на заводите по електроника в света. Това в много голяма степен се вижда и в Android мобилните системи, но го има и в мобилния Уиндоус, който борави с някои принципи за стабилност, но е силно повлиян от тези тенденции на negliжирано писане на софтуер. За много програмисти, дори в големи компании, теми като програмни обекти остават неясни, макар че те са твърде ясни, също така въпреки огромните заявки Android всъщност не може да работи добре с AI, а тези при Уиндос са от повторителен характер (тоест нямат действителни характеристики на мислене), което по някаква причина се отразява и на Android. Например причината за недоброто качество и липсата на ефективна работа на най-новите Уиндоус платформи се дължи на прекомерната употреба на php, което иначе трябва да става на C или на C++, опциите за теглене на приложения от сайт на Майкрософт не обясняват, защо и останалото трябва да се пише в тромав стил (дори и да е на C, то е пренесен php код), нито как терминът „спагети код“, който преди обозначава най-неработещият код на Майкрософт сега трябва да се отнася за най-доброто от Майкрософт или най-доброто на софтуерния пазар (виж китайски спагети, noodles или Moodle, както и странните рекламни спагетни слогани и аспирации за включване в софтуерната област, и внос на имидж)

Според стила основните сред тях са: обектно ориентирано, императивно, функционално и декларативно. Според друг източник обектно ориентираното програмиране спада към императивния стил, а функционалното програмиране спада към декларативния:

- Императивни

- Процедурни
- Рефлексивни
- Декларативни
  - Функционални
  - Логически
- Структурни
  - Обектно ориентирани
  - Функционални + Логически
  - Хибридни (декларативни + императивни)

Според предназначението си езиците за програмиране биват с общо предназначение и специализирани. Популярни съвременни езици за програмиране с общо предназначение са C, C++, Java, C#, Object Pascal, PHP, Perl. Специализирани езици за програмиране са например SQL (за заявки към системи за управление на бази от данни), JavaScript (за реализиране на динамично поведение в уеб сайтове от страна на клиента) и т.н.

Избирането на език за програмиране е свързано с много съображения като например фирмена политика, оперативна съвместимост, наличност на библиотеки или лични предпочитания.

```

1 //-----C# code \-----
2 using System;
3
4
5 0 references
6 class Program
7 {
8     0 references
9     static void Main()
10     {
11         Console.WriteLine("Hello, World!");
12     }
13
14 //-----Java code \-----
15
16 public class Main
17 {
18     public static void main(String[] args)
19     {
20         System.out.println("Hello, World!");
21     }
22 }

```

Снимка 2: Разлика между код написан на C# и Java, който отпечатва „Hello World!“

## 1.4. Въведение в математиката

Математиката е една от фундаменталните науки, която има широко приложение в различни области. Тя е в основата на множество академични дисциплини, включително икономика, физика, квантова механика, управление на бизнес и програмиране.

Логическата структура на математиката я прави тясно свързана с програмирането, което също изисква аналитично мислене и прилагане на строго определени правила и принципи.

## **1.5. Връзката между математиката и програмирането**

Математиката и компютърните науки имат сходни методологии за решаване на проблеми. Сложните проблеми се разглеждат и анализират чрез разлагането им на по-малки и прости подзадачи. След това всяка подзадача се решава отделно, а резултатите се комбинират за получаване на окончателното решение.

Както в математиката, така и в програмирането се анализират входните данни, информацията, функциите и получените резултати. Тези анализи позволяват да се направят локални изводи, които в крайна сметка водят до глобално решение на поставения проблем.

Програмирането и всички процеси, свързани с работата на компютрите, са пряко зависими от математиката. Изчислителните процеси на компютъра се основават на математически принципи, вариращи от най-простите математически операции и изрази до сложни изчисления, формули и концепции.

Математиката е също така основа за развитието на различни технологии, включително 3D графики, обработка на изображения, аудио и видео компресия, криптиране, GPS и други информационни технологии. Освен това математическите концепции подпомагат разработването на алгоритми, абстрактното мислене и разбирането на същността на различни проблеми, което е от съществено значение при разработката на софтуер.

## **1.6. Приложение на математиката в програмирането**

### **1.6.1. Алгоритми и структури от данни**

- Алгоритмите са сърцето на програмирането и често се изграждат върху математически принципи.
- Графи, дървета, хеш-таблици и масиви са примери за структури от данни, които използват математически модели.

### **1.6.2. Логика и Булева алгебра**

- Програмирането използва логически оператори (AND, OR, NOT), които са част от Булевата алгебра.
- Условни конструкции (if, while, for) разчитат на логически изрази.

### **1.6.3. Анализ на сложност и ефективност**

- Теорията на сложността (Big-O нотация) оценява бързодействието на алгоритмите.
- Изчислителната сложност е важен клон на математиката, който помага за оптимизация.

#### 1.6.4. Линейна алгебра и компютърна графика

- Векторите и матриците са основа за компютърната графика, машинното обучение и обработката на изображения.
- Трансформации (мащабиране, завъртане) в 3D графиката разчитат на линейна алгебра.

#### 1.6.5. Математически анализ и симулации

- Числените методи и диференциалните уравнения се използват в симулации и научни изчисления.
- Финансовото моделиране и статистическите анализи също разчитат на математиката.

#### 1.6.6. Теория на числата и криптография

- Шифроването на данни използва простите числа и модулната аритметика.
- RSA и други криптографски алгоритми се основават на математически принципи.

#### 1.6.7. Вероятности и статистика

- Машинното обучение и изкуственият интелект използват вероятностни модели.
- Статистически анализи се прилагат в обработка на големи данни (Big Data).

Прост пример за приложение на математиката в програмирането е използване на цикъл, в който се въвежда по едно число на всяка итерация и се търси дали е четно. Ако се намери, програмата излиза от цикъла, ако ли не – продължава докато не намери четно число.

```
var n = 0;
while (true)
{
    Console.Write("Enter even number: ");
    n = int.Parse(Console.ReadLine());
    if (n % 2 == 0)
    {
        break; // even number -> exit from the loop
    }
    Console.WriteLine("The number is not even.");
}
Console.WriteLine("Even number entered: {0}", n);
```

Снимка 3: Намиране на четно число в цикъл while

## ГЛАВА ВТОРА

### БРОЙНИ СИСТЕМИ. ПОБИТОВИ ОПЕРАЦИИ, ТЕОРИЯ НА МНОЖЕСТВАТА. КОМБИНАТОРИКА

#### 2.1. Бройни системи

Бройната система е метод за представяне на числа чрез фиксиран набор от символи и правила за комбиниране. Различните бройни системи се използват в математиката, компютърните науки и електрониката. Всяка бройна система се основава на определена база (основа), която определя броя на уникалните цифри в нея. Числата в различните системи могат да бъдат представени чрез степенни множители на базата.

Бройните системи имат дълга история и са се развивали в зависимост от нуждите на различните цивилизации. Най-ранните записани числа са открити в древните цивилизации на Шумер и Египет, където се използвали йероглифни символи. Римската бройна система е един от ранните примери за не-десетична система, докато индийско-арабската система, която използваме днес, въвежда позиционното представяне на числата.

С развитието на компютърните технологии бройните системи играят решаваща роля в дигиталния свят. Компютрите работят с двоична бройна система (база 2), защото електронните схеми могат лесно да представят два състояния – включено (1) и изключено (0). Освен това, шестнадесетичната и осмичната бройни системи се използват за по-лесно представяне на двоични числа в програмирането и цифровите технологии.

#### 2.2. Видове бройни системи

Бройните системи се класифицират според своята основа (база), която определя броя на уникалните цифри в системата.

##### 2.2.1. Десетична бройна система (основа 10)

- Най-използваната система в ежедневието.
- Използва цифрите 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Всяко число може да бъде представено като сбор от степените на 10.

**Пример:**

$$345_{10} = 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0.$$

Снимка 4: Пример за десетична бройна система

### 2.2.2. Двоична бройна система (основа 2)

- Използва се в компютърните науки и електрониката.
- Съдържа само две цифри: 0 и 1.
- Числата се представят като комбинация от степени на 2.

**Пример:**

$$1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{10}.$$

*Снимка 5: Пример за двоична бройна система*

### 2.2.3. Осмична бройна система (основа 8)

- Използва цифрите 0, 1, 2, 3, 4, 5, 6, 7.
- Използва се в програмирането като по-кратка форма на двоичната система.

**Пример:**

$$745_8 = 7 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 485_{10}.$$

*Снимка 6: Пример за осмична бройна система*

### 2.2.4. Шестнадесетична бройна система (основа 16)

- Използва цифрите 0-9 и буквите A-F (където A = 10, B = 11, ..., F = 15).
- Използва се широко в компютърните науки, особено за представяне на цветове и адреси в паметта.

**Пример:**

$$2F3_{16} = 2 \times 16^2 + 15 \times 16^1 + 3 \times 16^0 = 755_{10}.$$

*Снимка 7: Пример за шестнадесетична бройна система*

### 2.2.5. Тройна бройна система (основа 3)

- Използва цифрите 0, 1, 2.
- По-рядко срещана, но използвана в някои математически и компютърни приложения.

**Пример:**

$$102_3 = 1 \times 3^2 + 0 \times 3^1 + 2 \times 3^0 = 11_{10}.$$

Снимка 8: Пример за троична бройна система

### 2.2.6. Петична бройна система (основа 5)

- Използва цифрите 0, 1, 2, 3, 4.
- Среща се в някои древни цивилизации.

**Пример:**

$$243_5 = 2 \times 5^2 + 4 \times 5^1 + 3 \times 5^0 = 73_{10}.$$

Снимка 9: Пример за петична бройна система

## 2.3. Преобразуване на броични системи

### 2.3.1. Двоична в десетична

*Пример:* Имаме числото 101011 в двоична бройна система. За да го превърнем в десетична бройна система, трябва да сумираме тегловните коефициенти, умножени по цифрата на съответната позиция.

$$101011 = 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 43$$

Снимка 10: Преобразуване двоична в десетична

### 2.3.2. Десетична в двоична

За да превърнем число от десетична в двоична бройна система, трябва да го разделяме на 2, докато частното стане нула като записваме остатъците вдясно (ако числото не може да се дели на 2, записваме единица, а ако може – нула).

*Пример:* Преобразуване на 87 от десетична в двоична бройна система.

- $87 \div 2 = 43 \Rightarrow 1$
- $43 \div 2 = 21 \Rightarrow 1$
- $21 \div 2 = 10 \Rightarrow 1$
- $10 \div 2 = 5 \Rightarrow 0$
- $5 \div 2 = 2 \Rightarrow 1$
- $2 \div 2 = 1 \Rightarrow 0$

- $1 \div 2 = 0 \Rightarrow 1$

След това, за да получим двоичното число, вземаме получените единици и нули, като резултатът от последното деление става най-старшият разряд на двоичното число, съответно – резултатът от първото деление – най-младшият: на числото 87 в десетична бройна система съответства 1010111 в двоична.

За да получим двоичен еквивалент на десетична дроб, последователно умножаваме по основата на бройната система (в случая 2) до достигане на желаната точност. //При всяко умножаване цялата част на получения резултат е съответен разряд от двоичния еквивалент. В някои случаи точна десетична дроб не може да се получи и е налице остатък. В този случай дробта се закръгля според изискваната от конкретния случай точност (максимално допустимата за случая грешка).

### 2.3.3. Двоична в шестнадесетична и обратно

Най-лесно практически се осъществява чрез следната таблица за съответствие, използваща т.нар. тетради (четирицифрени двоични числа):

0000 – 0

0001 – 1

0010 – 2

0011 – 3

0100 – 4

0101 – 5

0110 – 6

0111 – 7

1000 – 8

1001 – 9

1010 – A

1011 – B

1100 – C

1101 – D

1110 – E

1111 – F

*Пример:* 0011101001110010<sub>(2)</sub>



- Разделяме числото от най-младшия разряд към най-старшия („от дясно наляво“) на тетради (полубайтове – четири бита) и според таблицата на съответствие между двете бройни системи получаваме 0011|1010|0111|0010 = 3A72. В случай, когато броят двоични цифри не е кратен на четири, за практическо удобство двоичното число се допълва откъм най-старшата цифра/бит („отляво“) с необходимия брой нули.

Редица от 0 и 1 се нарича двоичен или бинарен код.

## 2.4. Побитови операции

### 2.4.1. Оператор за побитово отрицание (NOT)

Операторът побитово отрицание (bitwise NOT), наричан също допълнение, е унитарна операция (операция с един операнд) която извършва логическо отрицание за всеки бит. Така се сформира допълнението за дадената двоична стойност. Битовете, които са били със стойност 0, приемат стойност 1, а битовете със стойност 1 стават 0. Например:

```
NOT 1010 (десетично 10)
    = 0101 (десетично 5)
```

Снимка 11: Побитово отрицание NOT

Побитовото допълнение е равно на допълнението на две на дадена стойност, минус едно. Ако се използва аритметично допълнение на две, тогава:

```
NOT x = -x -1
```

Снимка 12: Аритметично допълнение на две

В таблицата по-долу се вижда как работи побитовото отрицание върху бит A.

A	NOT
0	1
1	0

Снимка 13: Таблица на побитово отрицание

### 2.4.2. Побитово И (AND)

Побитовото И (AND) използва две двоични числа с еднаква дължина и прави логическо И на всяка двойка съответстващи битовете. Резултатът на дадена позиция е 1, ако първият бит е 1 и вторият е 1, в противен случай резултатът е 0. Използваме умножение на битовете :  $1 \times 0 = 0$  и  $1 \times 1 = 1$ . Например:

```
0101 (десетично 5)
AND 0011 (десетично 3)
= 0001 (десетично 1)
```

Снимка 14: Побитово И (AND) <sub>1</sub>

Този оператор може да се използва, за да определим дали даден бит е вдигнат (има стойност 1), или е свален (има стойност 0). Например, ако искаме за дадена поредица от битове 0110 (десетично 6), да разберем състоянието на втория бит, може да използваме оператор побитово „И“ за гореспоменатата поредица и поредица, където само втория бит е 1:

```
0110 (десетично 6)
AND 0010 (десетично 2)
= 0010 (десетично 2)
```

Снимка 15: Побитово И (AND) <sub>2</sub>

Понеже резултатът 0010 не е нула, знаем че вторият бит в оригинала е бил вдигнат. Това често бива наричано битово маскиране. (Аналогично на това как бояджийското тиксо покрива/маскира местата, които не трябва да бъдат променяни или не представляват интерес. В този случай нулите маскират битовете, които не ни интересуват.) Ако запазим резултата, това свойство може да се използва, за да се свалят избрани битове.

За побитово „И“ се използва знак „&“. В долната таблица е описано действието на тази операция върху битовете A и B:

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Снимка 16: Таблица на побитово И (AND)

### 2.4.3. Побитово ИЛИ (OR)

Побитовото ИЛИ взима две числа с еднаква дължина и прави логическа дизюнкция ИЛИ на всяка двойка съответстващи битове. Резултатът на дадена позиция е 1, ако първият или вторият бит има стойност 1 или ако и двата бита са равни на едно. В противен случай резултатът е 0. Например:

```
0101 (десетично 5)
OR 0011 (десетично 3)
= 0111 (десетично 7)
```

Снимка 17: Побитово ИЛИ (OR) <sub>1</sub>

Побитово „ИЛИ“ може да се използва за задаване на стойност 1 на определени битове, например ако имаме специфичен бит (флаг) от регистър, където всеки бит представлява единична булева стойност. Например 0010 (2 десетично) се състои от четири флага като първият, третият и четвъртият флаг имат стойност 0, а вторият има стойност 1. Чрез побитово „ИЛИ“ може да се зададе стойност 1 на четвъртия бит:

```
0010 (десетично 2)
OR 1000 (десетично 8)
= 1010 (десетично 10)
```

Снимка 18: Побитово ИЛИ (OR) <sub>2</sub>

Тази техника е много ефективен начин за съхраняване на булеви стойности, използвайки възможно най-малко памет.

Побитовото „ИЛИ“ се означава със знак „|“. В таблицата долу е представена функцията на оператора върху битове A и B.

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

Снимка 19: Таблица на побитово ИЛИ (OR)

#### 2.4.4. Побитово Изключващо ИЛИ (XOR)

Изключващото ИЛИ (XOR) взима две числа с еднаква дължина и прави логическо изключващо ИЛИ на всяка двойка съответстващи битове. Резултатът е равен на 1, ако само първият бит е 1 или ако само вторият бит е 1, но ако и двата бита са равни на 1 или и ако и двата бита са равни на 0, резултатът е 0. В следващия пример се прави сравнение между два бита и резултатът е 1, ако двата бита са различни и 0, ако са еднакви:

```
0101 (десетично 5)
XOR 0011 (десетично 3)
= 0110 (десетично 6)
```

Снимка 20: Побитово Изключващо ИЛИ (XOR) <sub>1</sub>

Изключващото „ИЛИ“ може да се използва за обръщане на битове (също така наричано toggle или flip. Всеки бит може да се обърне след XOR с 1. Например в числото 0010 (2 десетично) може да обърнем втория и четвъртия бит след изключващо ИЛИ с число, чиито битове на позиция две и четири имат стойност 1.

```

    0010 (десетично 2)
XOR 1010 (десетично 10)
    = 1000 (десетично 8)

```

Снимка 21: Побитово Изключващо ИЛИ (XOR) 2

Програмистите на асемблер често използват XOR като по-лесен начин да променят стойността на някой регистър на нула. Ако се направи изключващо „ИЛИ“ на две еднакви числа, резултатът е 0. В повечето архитектури тази операция изисква много по-малко цикли и/или памет, отколкото задаването на нулева стойност и запазването ѝ в регистъра.

Таблицата на побитовото изключващо „ИЛИ“ за битове A и B изглежда по следния начин:

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Снимка 22: Таблица на побитовото Изключващо ИЛИ (XOR)

## 2.5. Теория на множествата

Теория на множествата е дял от математиката, която изучава множествата, като съвкупност от обекти. Въпреки че всякакъв вид обект може да бъде поместен в множество, теорията на множествата се прилага най-често за обекти, които са свързани с математиката. Термините от теорията на множествата може да се използват в определенията на почти всички математически обекти.

### 2.5.1. Основни концепции

В основата на теорията на множествата е фундаменталната бинарна релация между даден обект  $o$  и дадено множество  $A$ . Ако  $o$  е елемент на  $A$ , това се изписва като  $o \in A$ . Тъй като множествата са обекти, релацията може да свързва и множества.

Производна бинарна релация между две множества е релацията подмножество. Ако всички елементи на множеството  $A$  са също елементи на множеството  $B$ , то  $A$  е подмножество на  $B$ :  $A \subseteq B$ . Например,  $\{1,2\}$  е подмножество на  $\{1,2,3\}$ , но  $\{1,4\}$  не е. От това определение се вижда, че всяко множество е подмножество на самото себе си. В случаите, в които това трябва да се избегне, се използва понятието чисто подмножество.

Подобно на бинарните операции върху числа в аритметиката, теорията на множествата използва бинарни операции върху множества:

- Обединение на множествата  $A$  и  $B$  ( $A \cup B$ ) е множеството на всички обекти, които са елементи на  $A$ ,  $B$  или на двете. Например, обединението на  $\{1, 2, 3\}$  и  $\{2, 3, 4\}$  е множеството  $\{1, 2, 3, 4\}$ .

- Сечение на множествата  $A$  и  $B$  ( $A \cap B$ ) е множеството от обекти, които са елементи и на  $A$ , и на  $B$ . Например, сечението на  $\{1, 2, 3\}$  и  $\{2, 3, 4\}$  е множеството  $\{2, 3\}$ .
- Разлика на множествата  $U$  и  $A$  ( $U \setminus A$ ) е множеството от всички елементи на  $U$ , които не са елементи на  $A$ . Например, разликата  $\{1,2,3\} \setminus \{2,3,4\}$  е  $\{1\}$ , докато разликата  $\{2,3,4\} \setminus \{1,2,3\}$  е  $\{4\}$ . Когато  $A$  е подмножество на  $U$ , разликата  $U \setminus A$  се нарича и допълнение на  $A$  в  $U$ . В този случай изборът на  $U$  е ясен от контекста и понякога се използва нотацията  $A^c$ , особено когато  $U$  е универсално множество.
- Симетрична разлика на множествата  $A$  и  $B$  е множеството от обектите, които са елементи или само на  $A$ , или само на  $B$ . Например, симетричната разлика на множествата  $\{1,2,3\}$  и  $\{2,3,4\}$  е множеството  $\{1,4\}$ . Симетричната разлика на две множества е разликата на обединението и сечението им:  $(A \cup B) \setminus (A \cap B)$ .
- Декартово произведение на множествата  $A$  и  $B$  ( $A \times B$ ) е множеството от всички възможни наредени двойки  $(a,b)$ , където  $a$  е елемент на  $A$ , а  $b$  е елемент на  $B$ . Например, декартовото произведение на  $\{1, 2\}$  и  $\{\text{червено}, \text{бяло}\}$  е  $\{(1, \text{червено}), (1, \text{бяло}), (2, \text{червено}), (2, \text{бяло})\}$ .

Основни множества с голямо значение в математиката са празното множество, което не съдържа никакви елементи, както и множествата на естествените и реалните числа.

## 2.6. Комбинаторика

Комбинаториката е сред най-старите и силно развити дялове на математиката и по-специално на дискретната математика. Основен обект, с който се занимава комбинаториката, е комбинаторната конфигурация. В областта на комбинаториката са се оформили две проблемни области: изброителна комбинаторика и структурна комбинаторика.

### 2.6.1. Основни правила на комбинаториката

- Правило за събиране

Ако елементът  $a$  може да бъде избран по  $m$  начина, а елементът  $b$  по  $n$  различни начина, изборът на „ $a$  или  $b$ “ може да се извърши по  $m + n$  начина. Правилото за събиране може да се обобщи за повече от две множества. Трябва броят на всички обекти да е равен на сбора от броя им в отделните групи.

- Правило за умножение

Ако елементът  $a$  може да бъде избран по  $m$  начина и при всеки избор на  $a$  елементът  $b$  може да бъде избран по  $n$  начина, то изборът на наредената двойка  $(a,b)$  може да стане по  $m \cdot n$  начина. Правилото за умножение може да се обобщи за намиране броя на наредени тройки обекти, наредени четворки обекти.

## 2.6.2. Пермутации

**Пермутация без повторение** наричаме конфигурация от  $n$ -елементно множество, от което трябва да изберем всичките  $n$  елемента, като редът е от значение. Броят на конфигурациите е  $n!$  (ен факториел) и има стойност  $n! = n \cdot (n - 1) \cdot (n - 2) \dots 3 \cdot 2 \cdot 1$ .

Прост пример за една комбинаторна задача е: „По колко начина може да се нареди едно тесте от  $n$  карти?“. Отговорът е  $n!$ .

## 2.6.3. Вариации

Вариациите без повторение на  $n$  елемента от  $k$ -ти клас ( $k < n$ ) се наричат такива съединения, всяко от които съдържа по  $k$  различни елемента от дадените  $n$  и се различават едно от друго или по елементите, или по реда на елементите. Разликата между вариациите и пермутациите на елементите на някакво множество е единствено в това, че в една вариация не е задължително да участват всички елементи на множеството. Ясно е, че всяка пермутация е вид вариация (от  $n$  елемента  $n$ -ти клас), докато обратното не е вярно.

- Формула за броя на вариациите

Броят на различните вариации от  $n$  елемента от  $k$ -клас се означава с:

$$V_n^k \cdot V_n^k = \frac{n!}{(n - k)!}$$

Снимка 23: Формула Вариации

## 2.6.4. Комбинации

Комбинации без повторение от  $n$ -елемента от  $k$ -ти клас се наричат такива съединения, всяко от които съдържа по  $k$  различни елемента от дадените  $n$  и се различават едно от друго с поне 1 елемент.

- Формула за броя на комбинациите

Броят на различните комбинации без повторение от  $n$ -елемента от  $k$ -ти клас се означава с  $C(n, k)$  или  $C_n^k$ .

Броят на комбинациите от  $n$ -елемента от  $k$ -ти клас е:

$$C_n^k = \frac{V_n^k}{P_k} = \frac{n \cdot (n - 1) \cdot (n - 2) \dots (n - k + 1)}{k(k - 1) \dots 3 \cdot 2 \cdot 1}$$

Снимка 24: Формула Комбинации

## ГЛАВА ТРЕТА

### ПРИЛОЖЕНИЕ – КОНВЕРТОР МЕЖДУ БРОЙНИ СИСТЕМИ

Това приложение е разработено с помощта на C# и Windows Forms и представлява удобен инструмент за конвертиране на числа между различни бройни системи. То поддържа преобразувания между двоична (BIN), десетична (DEC) и шестнадесетична (HEX) система.

#### 3.1. Какво е C# и за какво е използван в приложението?

C# е обектно-ориентиран програмен език, разработен от Microsoft като част от платформата .NET. Той се използва за разработка на десктоп, уеб и мобилни приложения, както и за игри, облачни услуги и микросървиси.

**Основни характеристики:**

- Статично типизиран език.
- Управление на паметта чрез Garbage Collector.
- Поддръжка на асинхронно програмиране.
- Вграден механизъм за обработка на изключения.

##### 3.1.1. Видове променливи използвани в приложението

В C# променливите се използват за съхраняване на стойности в паметта по време на изпълнение на програмата. Те се разделят на различни типове според съдържанието и начина на съхранение.

- Цели числа (int, long, short, byte) – числови стойности без дробна част;
- Дробни числа (float, double, decimal) - числови стойности с плаваща запетая;
- Символи (char) - Представя един символ;
- Текст (string) - Последователност от символи;
- Булеви стойности (bool) - Приема стойности **true** или **false**;
- Неинициализирани стойности (nullable types) - Позволява стойности **null** за стойностни типове.

Пример за работа с различни видове променливи:

```

using System;

0 references
class Program
{
    0 references
    static void Main()
    {
        int number = 10;
        double price = 15.99;
        char symbol = 'A';
        string text = "Примерен текст";
        bool isAvailable = true;

        Console.WriteLine($"Число: {number}, Цена: {price}, Символ: {symbol}, Текст: {text}, Статус: {isAvailable}");
    }
}

```

Снимка 25: Видове променливи в C#

### 3.1.2. Цикли в C#

Циклите се използват за многократно изпълнение на код при изпълнение на дадено условие.

- **for** - Изпълнява блок код определен брой пъти. Използва се за преброяване на елементи.
- **while** - Изпълнява блок код, докато условието е вярно. Използва се за четене от вход.
- **do-while** - Изпълнява блок поне веднъж, след което проверява условието. Обикновено се използва за менюта в конзолни приложения.
- **foreach** - Обхожда елементите на колекция. Използва се за обработка на масиви и списъци.

## 3.2. WinForms

Windows форми е графична (GUI) библиотека от класове в състава на Microsoft .NET Framework, която предоставя платформа за писане на клиентски приложения за настолни компютри, лаптопи и таблети.

Дизайнерът Windows Forms се използва за създаването на приложения с графичен потребителски интерфейс. Този дизайнер позволява да се добавят и оформят различни менюта и бутони. Полетата показващи данни, могат да бъдат свързвани с различни източници на данни, като бази данни или заявки. Тези полета се добавят чрез изтегляне от прозореца Data Sources върху създадения формуляр. Потребителският интерфейс задължително се свързва с програмен код, за създаването на приложение.

### 3.2.1. Архитектура

Изработените с помощта на Windows форми приложения се задействат при настъпване на определено събитие или при определено действие от страна на потребителя, като например попълване на текстово поле или посочване и щракване на бутон. Windows формите предоставят достъп до стандартните вградени контроли на Windows User Interface, като комбинира Windows API и т.нар. managed code.



### 3.2.2. Характеристики

Всички визуални елементи в библиотеката Windows Forms са получени от класа Control. Това осигурява минималната необходима информация за всеки елемент от потребителския интерфейс, като например местоположение, размер, цвят, шрифт, текст, както и чести събития, като посочване и щракване и влачене и пускане.

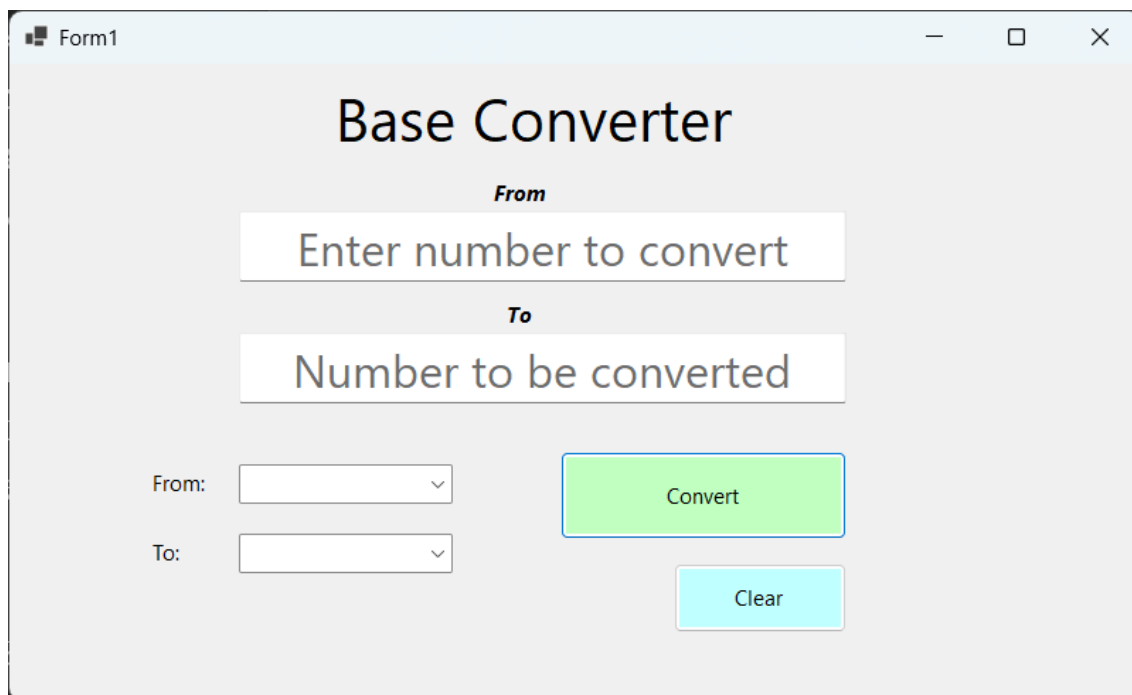
### 3.3. Функционалност на приложението – ConverterApp

Разработеното приложение представлява настолен софтуерен инструмент, насочен към осигуряване на интуитивна и ефективна среда за преобразуване между различни бройни системи. То е създадено с идеята да облекчи работата на потребители, които често боравят с числови стойности, представени в двоичен, десетичен или шестнадесетичен формат – независимо дали става дума за образователни цели, програмиране или технически анализи.

В основата на приложението стои стремежът към яснота, достъпност и бърза ориентация в процеса на конвертиране между различните представяния на числова информация. То съчетава функционална практичност с опростен визуален интерфейс, който позволява на потребителя да се фокусира изцяло върху задачата, без да бъде затормозен от излишна сложност.

Приложението е резултат от комбинирането на базови концепции в програмирането с приложна насоченост, и цели да покаже как чрез сравнително елементарни алгоритми и потребителски интерфейс може да се създаде завършен, полезен и надежден софтуерен продукт.

#### 3.3.1. Изглед на приложението при стартиране

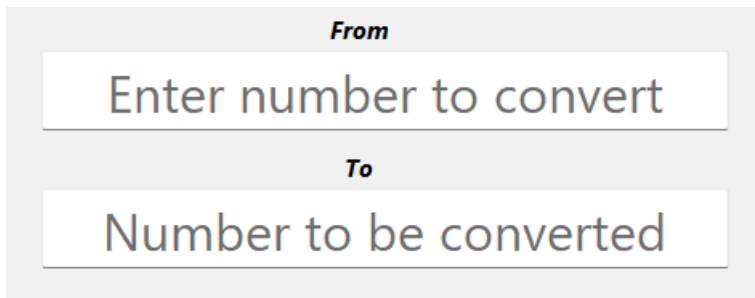
The screenshot shows a Windows Forms application window titled "Form1". The main content area has a light gray background. At the top, the text "Base Converter" is displayed in a large, bold, black font. Below this, the word "From" is centered in a smaller, italicized font. Underneath "From" is a white rectangular input field with the placeholder text "Enter number to convert". Below that, the word "To" is centered in a smaller, italicized font. Underneath "To" is another white rectangular input field with the placeholder text "Number to be converted". At the bottom left, there are two labels: "From:" and "To:", each followed by a small white dropdown menu with a downward arrow. To the right of these labels is a large green rectangular button with the text "Convert" in black. Below the "Convert" button is a smaller light blue rectangular button with the text "Clear" in black.

Снимка 26: Изглед на приложението

### 3.3.2. Функционалност и разбиране на ConverterApp

Приложението на практика е доста просто изработено. При стартиране на приложението, виждаме две текстови полета, две падащи полета и два бутона.

- Текстови полета – Въвеждане и Извеждане



Снимка 27: Текстови полета – Въвеждане и Извеждане

#### textBox1 –

Това е първото текстово поле в интерфейса, разположено непосредствено под надписа "From". То представлява входно поле, в което потребителят въвежда числото, което желае да преобразува от една бройна система в друга.

- Надпис вътре в полето (placeholder): "Enter number to convert"
- Тип: TextBox
- Функция: Приема стойности в зависимост от избраната начална бройна система (бинарна, десетична, шестнадесетична).
- **Пример:** Ако потребителят иска да преобразува шестнадесетично число 1A в десетично, то трябва да въведе 1A в това поле.

#### Кодова логика, свързана с textBox1:

```
if (String.IsNullOrEmpty(textBox1.Text))
{
    MessageBox.Show("Error! There is not entered number to convert!");
}
```

Снимка 28: Грешка при въвеждане на грешна стойност

Това е първата проверка, която се извършва при натискане на бутона „Convert“. Ако полето е празно или съдържа само интервали, се показва съобщение за грешка и не се извършва преобразуване. Това гарантира, че се работи само с валидни входни стойности.

#### textBox2 –

Това е второто текстово поле в интерфейса, намиращо се под надписа "To". То не е предназначено за въвеждане от потребителя, а автоматично се попълва с резултата от извършеното преобразуване.

- Надпис вътре в полето (placeholder): "Number to be converted"
- Тип: TextBox

- **Функция:** Показва изчислената стойност според избраната целева бройна система.

Във всеки случай, `textBox2.Text` служи като контейнер за крайния резултат. Той се попълва автоматично и не подлежи на директна редакция от потребителя в нормалната логика на приложението.

`textBox1` и `textBox2` са директно свързани – стойността, въведена в първото, служи като изходна за изчисления, а второто визуализира крайния резултат.

Те работят в синхрон със стойностите, избрани в комбо боксовете, и зависят от действията върху бутоните `Convert` и `Clear`.

Крайната цел: максимално просто и ясно взаимодействие – въвеждаш число ➤ избираш посока ➤ получаващ резултат.

- Падащи менюта – Въвеждане и Извеждане

### **comboBox1 – Избор на входна бройна система ("From") –**

Това е първото от двете падащи менюта, обозначено с текстовия етикет "From:", който се намира вляво под първото текстово поле. То служи за избор на изходната бройна система, т.е. тази, в която е написано въведеното от потребителя число.

- **Тип:** `ComboBox`
- **Разположение:** Под текстовото поле за вход
- **Възможни стойности (най-вероятно):**
  - "Binary" (двоична)
  - "Decimal" (десетична)
  - "Hexadecimal" (шестнадесетична)

Изборът в `comboBox1` определя типа на въведеното число в `textBox1`. При натискане на бутона „Convert“ програмата чете избраната стойност от `comboBox1` и според нея решава кой алгоритъм да използва.

```
if (comboBox1.SelectedIndex == 1 && comboBox2.SelectedIndex == 0)
{
    ...
}
```

Снимка 29: `comboBox` проверка

Горният пример показва, че ако потребителят е избрал:

From: Decimal (`SelectedIndex == 1`)

To: Binary (`SelectedIndex == 0`)

Тогава приложението ще извърши преобразуване от десетична в двоична бройна система.

### **comboBox2 – Избор на целева бройна система ("To") –**

Второто падащо меню, обозначено с текстовия етикет "То:", се намира точно под първото (comboBox1). То служи за избор на целева бройна система, т.е. тази, в която числото ще бъде преобразувано и показано в textBox2.

- **Тип:** ComboBox
- **Разположение:** Под comboBox1, вляво от бутоните
- **Възможни стойности (идентични с comboBox1):**
  - "Binary"
  - "Decimal"
  - "Hexadecimal"

#### Роля в логиката:

Комбинацията от comboBox1 (From) и comboBox2 (To) определя конкретната посока на преобразуване, която трябва да се извърши.

#### Гъвкавост и валидност:

Възможни са всички 6 комбинации между 3-те системи (без преобразуване към същата система). За всяка от тях в кода има отделна логика. Примерните комбинации са:

From	To	Действие
Binary	Decimal	Двоично → Десетично
Binary	Hexadecimal	Двоично → Шестнадесетично
Decimal	Binary	Десетично → Двоично
Decimal	Hexadecimal	Десетично → Шестнадесетично
Hexadecimal	Decimal	Шестнадесетично → Десетично
Hexadecimal	Binary	Шестнадесетично → Двоично

Снимка 30: Всички възможни комбинации за преобразуване

- Бутони – „Преобразувай“ и „Изчисти“

Бутонът „Convert“ е разположен вдясно от падащите менюта, непосредствено под текстовите полета. Отличава се със светлозелен фон, което го откроява визуално като основен елемент за действие.

- **Текст на бутона:** Convert
- **Име в кода:** button1
- **Цел:** Стартира процеса на конвертиране от една бройна система в друга

#### Поведение и логика:

При натискане на бутона, се задейства методът:

```
private void button1_Click(object sender, EventArgs e)
```

В този метод се съдържа цялата основна логика на приложението за преобразуване между бройните системи. Логиката се ръководи от избора в comboBox1 (From) и comboBox2 (To), както и стойността в textBox1.

### Алгоритми и конвертиране:

След валидацията следват **6 условни клона**, които обработват различните комбинации на „From“ и „To“:

- **Decimal → Binary**

```
//Decimal to binary
if (comboBox1.SelectedIndex == 1 && comboBox2.SelectedIndex == 0)
{
    long number = Convert.ToInt64(textBox1.Text);
    long binary = 0;
    long I = 0;

    while (number != 0)
    {
        long d = (long)number % 2;
        binary = binary + (long)d * (long)Math.Pow(10, I);
        I = I + 1;
        number = number / 2;
    }
    textBox2.Text = binary.ToString();
}
```

Снимка 31: Преобразуване - Decimal to Binary

- **Binary → Decimal**

```
//Binary to decimal
else if (comboBox1.SelectedIndex == 0 && comboBox2.SelectedIndex == 1)
{
    long num, binary, decimal_val = 0, base_val = 1, rem;

    num = long.Parse(textBox1.Text);

    binary = num;
    while (num > 0)
    {
        rem = num % 10;
        decimal_val = decimal_val + rem * base_val;
        num = num / 10;
        base_val = base_val * 2;
    }
    textBox2.Text = decimal_val.ToString();
}
```

Снимка 32: Преобразуване - Binary to Decimal

- Decimal → Hex и Hex → Decimal

```
//Decimal to hex
else if (comboBox1.SelectedIndex == 1 && comboBox2.SelectedIndex == 2)
{
    long decimalNumber = Convert.ToInt64(textBox1.Text);
    string hexadecimalNumber = decimalNumber.ToString("X");
    textBox2.Text = hexadecimalNumber;
}
//Hex to Decimal
else if (comboBox1.SelectedIndex == 2 && comboBox2.SelectedIndex == 1)
{
    string hex_value = textBox1.Text;
    long int_value = Convert.ToInt64(hex_value, 16);
    textBox2.Text = int_value.ToString();
}
```

Снимка 33: Decimal to Hex / Hex to Decimal

- Hex → Binary и Binary → Hex

```
//Hex to Binary
else if (comboBox1.SelectedIndex == 2 && comboBox2.SelectedIndex == 0)
{
    string hexValue = textBox1.Text;
    string binaryValue = Convert.ToString(Convert.ToInt32(hexValue, 16), 2);
    textBox2.Text = binaryValue.ToString();
}
//Binary to Hex
else if (comboBox1.SelectedIndex == 0 && comboBox2.SelectedIndex == 2)
{
    string binaryNumber = textBox1.Text;
    long decimalNumber = Convert.ToInt64(binaryNumber, 2);
    string hexadecimalNumber = Convert.ToString(decimalNumber, 16).ToUpper();
    textBox2.Text = hexadecimalNumber;
}
```

Снимка 34: Hex to Binary / Binary to Hex

### **Бутонът „Clear“ –**

Бутонът „Clear“ е разположен под бутона „Convert“. Има светлосин фон, което визуално подсказва, че изпълнява вторична функция, свързана с нулиране или почистване на формата.

- Текст на бутона: Clear
- Име в кода: button2
- Цел: Изтрива съдържанието на двете текстови полета

### **Функционалност:**

Когато потребителят натисне бутона, се активира методът:

```
private void button2_Click(object sender, EventArgs e)
{
    textBox1.Clear();
    textBox2.Clear();
}
```

Снимка 35: Бутон Clear

### 3.4. Тестване на приложението с реални данни

- От Двоична към Десетична бройна система

The screenshot shows a Windows application window titled 'Form1' with the title bar containing standard minimize, maximize, and close buttons. The application is titled 'Base Converter'. It features two input fields: the top one is labeled 'From' and contains the binary value '01010010010'; the bottom one is labeled 'To' and contains the decimal value '658'. Below these fields are two dropdown menus: 'From:' is set to 'Binary-2' and 'To:' is set to 'Decimal-10'. To the right of the dropdowns are two buttons: a green 'Convert' button and a light blue 'Clear' button.

Снимка 36: Тестване от Двоична към Десетична

- От Десетична към Двоична бройна система

The screenshot shows the same 'Form1' application window. The 'From' input field now contains the decimal value '2355' and the 'To' input field contains the binary value '100100110011'. The 'From:' dropdown menu is now set to 'Decimal-10' and the 'To:' dropdown menu is set to 'Binary-2'. The green 'Convert' button and the light blue 'Clear' button remain visible.

Снимка 37: Тестване от Десетична към Двоична

- **От Двоична към Шестнадесетична бройна система**

The screenshot shows a Windows application window titled "Form1" with a title bar containing standard minimize, maximize, and close buttons. The application is titled "Base Converter" in a large, bold font. Below the title, there are two input fields. The first field, labeled "From" above it, contains the binary number "100100110011". The second field, labeled "To" above it, contains the hexadecimal number "933". Below these fields, there are two dropdown menus. The "From:" dropdown is set to "Binary-2" and the "To:" dropdown is set to "Hex-16". To the right of these dropdowns are two buttons: a green "Convert" button and a light blue "Clear" button.

Снимка 38: Тестване от Двоична към Шестнадесетична

- **От Шестнадесетична към Двоична бройна система**

The screenshot shows the same "Base Converter" application window. In this instance, the "From" input field contains the hexadecimal number "A17" and the "To" input field contains the binary number "101000010111". The "From:" dropdown menu is now set to "Hex-16" and the "To:" dropdown menu is set to "Binary-2". The "Convert" and "Clear" buttons remain visible.

Снимка 39: Тестване от Шестнадесетична към Двоична



## **ЗАКЛЮЧЕНИЕ**

Програмните езици и математиката са неразривно свързани, като те формират основата на съвременните софтуерни приложения и технологии. Възможността за ефективно управление на данни и изпълнение на алгоритми се дължи на основните математически концепции, като линейна алгебра, булева логика, вероятности и криптография. В дипломната работа е направен подробен анализ на значението на математиката в програмирането и е разгледано приложението на тези теории в контекста на C# и съвременните разработки в областта на софтуерното инженерство.

Проектът, разгледан в документацията, демонстрира как тези теоретични концепции могат да бъдат приложени практически чрез създаването на конвертор за бройни системи. Използването на C# и WinForms осигурява стабилна и гъвкава среда за разработка на приложение, което отговаря на поставените изисквания. Тестването на приложението с реални данни потвърждава неговата надеждност и ефективност. Също така, архитектурата на приложението позволява бъдещо разширение с нови функции или адаптиране към други видове приложения. Работата по този проект показва как фундаментални математически принципи могат да се интегрират в реални софтуерни решения, които отговарят на нуждите на бизнеса и индустрията.

## ИЗПОЛЗВАНА ЛИТЕРАТУРА

1. **Колисниченко, Д.** (2016). Въведение в .NET, София, Издателство „Асеновци трейд ЕООД“.
2. **Тепляков, С.** (2017). *Шаблони за дизайн на платформата .NET*, София, Издателство „Асеновци трейд ЕООД“.
3. **Уогнър, Б.** (2022). *Ефективно програмиране със C#*, София, Издателство ИК „Алекс-Софт“.
4. Microsoft. (2025). *C# Programming Guide*. Получено от <https://learn.microsoft.com/en-us/dotnet/csharp/>
5. **Бейкер, Дж.** (2018). *Алгоритми и структури от данни*, София, Издателство "Математика и информатика".
6. **Таненбаум, А.** (2019). *Структури от данни и алгоритми*, София, Издателство "Програма".
7. **Уикипедия.** (2025). C#. Получено от [https://bg.wikipedia.org/wiki/C\\_Sharp](https://bg.wikipedia.org/wiki/C_Sharp)
8. **TutorialsPoint.** (2025). *C# File I/O*. Получено от [https://www.tutorialspoint.com/csharp/csharp\\_file\\_io.htm](https://www.tutorialspoint.com/csharp/csharp_file_io.htm)
9. SoftUni. (2025). *Основи на програмирането с C#*. Получено от <https://softuni.bg/courses/learn/csharp>