

Тема:

*Инкриптиране и декриптиране в ASCII
таблица*

Изготвил: Яни Живков

Клас: 9а

Дата на предаване: 06.18.2025г.



Описание на заданието

Програмата трябва да приема от потребителя текст, да го преобразува по зададена логика (например чрез изместване на всеки символ с определен брой позиции в ASCII кода) и да показва получения шифриран резултат. Също така трябва да може да върне оригиналния текст чрез декриптиране.

Основната обработка трябва да бъде реализирана чрез *клас*, например „Encryptor“, който съдържа два метода: един за *инкриптиране* и един за *декриптиране*. Всеки метод трябва да приема низ и да го обработва като масив от символи, прилагайки преобразуване на всеки символ чрез операция с неговия ASCII код.

За да се демонстрира използването на *масив*, програмата трябва да преобразува входния текст в масив от символи, след което с *цикъл* да премине през всеки символ, да го модифицира и да запише резултата в нов масив, който после се сглобява обратно в текст.

Потребителят трябва да избере чрез меню дали иска да шифрира или дешифрира съобщение, след което да въведе текста. Програмата трябва да обработи текста и да изведе резултата. Изместването (например +3 или друго число) може да бъде зададено фиксирано или въведено от потребителя.

Съдържание

1. Въведение - Какво са C#, Метод и Масив?
2. Описание на кода
3. Логика на криптирането с ASCII
4. Работа с Дебъгера
5. Заключение
6. Изпозвана литература

1. Въведение

1.1 Какво е C#?

C# (произнася се “си шарп”) е съвременен език за програмиране, създаден от Microsoft през началото на 2000-те години. Той е официално част от .NET платформата и се опитва да бъде лесно за използване, но в същото време достатъчно мощен за разработка на големи и сложни приложения. C# комбинира характеристиките на няколко други езика като Java и C++, но избира да бъде по-сигурен и по-структуриран от тях. Това го прави оптимален за начинаещи програмисти, но запазва достатъчно гъвкавост, за да задоволи професионални изисквания. Солидно качество на C# е уважаването му. Когато пишеш програма на C#, езикът ти помага да избегнеш често срещаните грешки. Например, той няма да ти позволи да работиш с променливи, която няма начална доза или да “достъпиш” обекти, които не съществуват в паметта. Това го прави по-стабилна платформа за работа. C# позволява съдействие на различен софтуер: той може да е десктоп приложение за Windows, уеб сайт или уеб услуга, мобилно приложение чрез различни платформи. Синтаксисът на C# е лесен за разбиране, подреден. Примерите в документацията често са изчерпателно моделирани и обяснени или напълно чисто описани термини, което го прави много по-лесен за разбиране. Освен това средата за разработка, в която се разработва – Visual Studio Service – предоставя много удобна за функции – автоматично дописване на недописаните продукти, откриване на грешки в кода, премахване на грешки, и т.н. и т.н. • Много учебни заведения, университети и други използват първо C# като основен език, под който тренират. Той комбинира практически и теоретически знания много добре. Това е още една предимство – това е от Microsoft, което може да се обновява. Трябва да споменем, че C# е, в същността си, обектно-ориентиран език. Това означава, че работи с обекти и класове – концепции, които позволяват на програмирането да стане по-организиран и логичен подход. Тази черта по принцип е отлична, когато проектите поемат по-озбилен траектория или увеличат обемите. По принцип, в наши дни, като се има предвид бързото развитие на технологиите, C# е преди всичко стабилен избор на програмисти по целия свят. Практичен, достатъчно бърз и

сравнително лесен за четене и разбиране на синтаксиса, този език единствено единствено има жизнеспособно бъдеще.

1.2 Какво е Метод?

Метод (method) е съставна част от програмата, която решава даден проб-лем, може да приема параметри и да връща стойност.

В методите се извършва цялата обработка на данни, която програмата трябва да направи, за да реши поставената задача. Методите съдържат логиката на програмата и те са мястото, където се извършва реалната работа. Затова можем да ги приемем като строи-телен блок на програмата. Съответно, имайки множество от простички блокчета – отделни методи, можем да създаваме големи програми, с които да решим по-сложни проблеми. Ето например как изглеж-да един метод за намиране лице на правоъгълник (1фиг.):

```
static double GetRectangleArea(double width, double height)
{
    double area = width * height;
    return area;
}
```

(1фиг.)

Има много причини, които ни карат да използваме методи. Ще разгледаме някои от тях и с времето ще се убедите, че методите са нещо, без което не можем, ако искаме да програмираме сериозно.

При създаването на една програма, е добра практика да използваме методи, за да я направим добре структурирана и лесно четима не само за нас, но и за други хора.

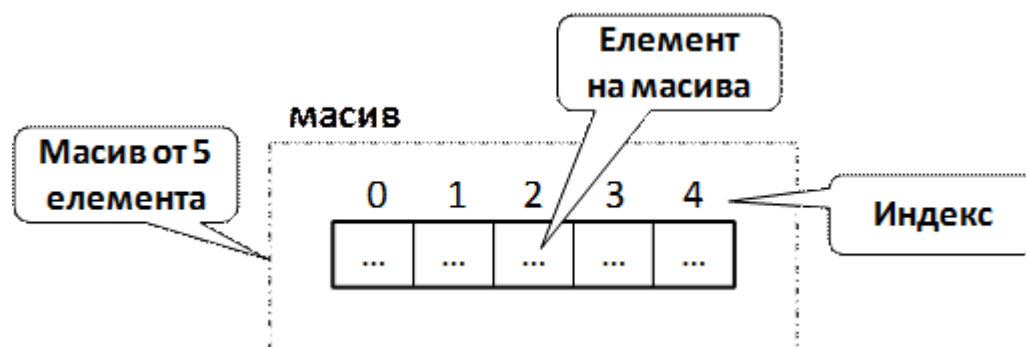
Довод за това е, че за времето, през което съществува една програма, средно само 20% от усилията, които се заделят за нея, се състоят в създаване и тестване на кода. Останалата част е за поддръжка и добавяне на нови функционалнос-ти към началната версия. В повечето случаи, след като веднъж кодът е написан, той не се поддържа и

модифицира само от създателя му, но и от други програмисти. Затова е важно той да е добре структуриран и лесно четим.

Друга много важна причина, заради която е добре да използваме методи е, че по този начин избягваме повторението на код. Това е пряко свързано с концепцията за преизползване на кода.

1.3 Какво е Масив?

Масивите са неизменна част от повечето езици за програмиране. Те представляват съвкупности от променливи, които наричаме елементи (2фиг.):



(2фиг.)

Елементите на масивите в C# са номерирани с числата 0, 1, 2, ... N-1. Тези номера на елементи се наричат индекси. Броят елементи в даден масив се нарича дължина на масива.

Всички елементи на даден масив са от един и същи тип, независимо дали е примитивен или референтен. Това ни помага да представим група от еднородни елементи като подредена свързана последователност и да ги обработваме като едно цяло.

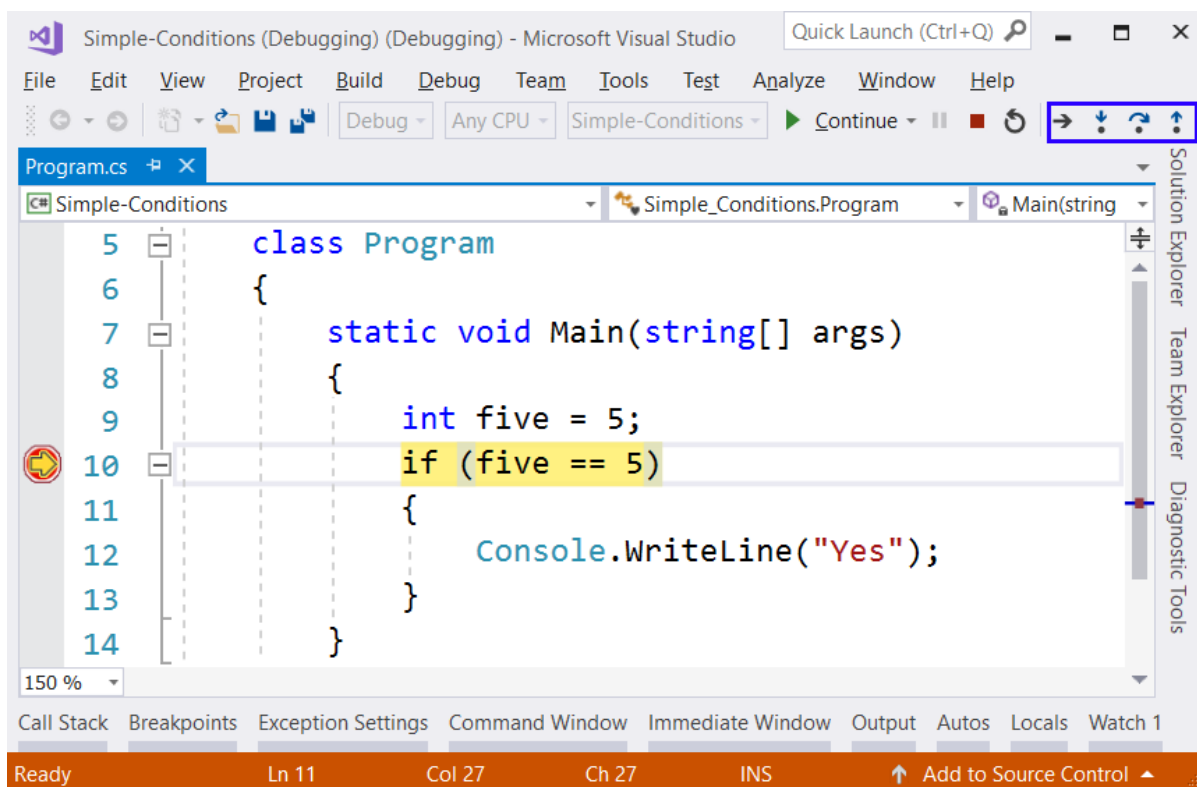
Масивите могат да бъдат от различни размерности, като най-често използвани са едномерните и двумерните масиви. Едномерните масиви се наричат още вектори, а двумерните – матрици.

Те по подразбиране са нулево-базирани, т.е. номерацията на елементите започва от 0. Първият елемент има индекс 0, вторият 1 и т.н. Ако един масив има N елемента, то последният елемент се намира на индекс N-1.

Достъпът до елементите на масивите е пряк и се осъществява по индекс. Всеки елемент може да се достъпи с името на масива и съответния му индекс (пореден номер), поставен в квадратни скоби. Можем да осъществим достъп до даден елемент както за четене така и за писане т.е. да го третираме като най-обикновена променлива.

1.4 Какво е Дебъгер?

Инструментът, който програмистите използват за откриване и отстраняване на грешки в кода, се нарича дебъгер. Вместо да се опитваш да разбереш къде точно е проблемът чрез съобщение даните на програмата, с дебъгер просто можеш да наблюдаваш как точно се изпълнява кода стъпка по стъпка. Може да виждаш, къде тръгва на грешната позиция, да поставяш breakpoints, да виждаш стойностите на променливите и дали програмата си върши работата така, както трябва. Това е изключително полезно другаде, когато кодът е дълъг, сложен или работи с много данни (3фиг.).



(3фиг.)

Повечето ив средите за разработка като Visual Studio, IntelliJ, PyCharm и Eclipse, имат вградени дебъгери. Те предлагат визуални инструменти, които правят откриването на грешки по-лесно и по-интуитивно. Следователно може да се заключи, че дебъгерът е неотделима част от програмирането. Той не само прави намирането на грешки по-лесно, а помага и намирането на работната логика на програмата. Без съвременния дебъгер програмирането би било безсмислено.

2. Описание на кода

2.1 Клас: Encryptor

Този клас е сърцевината на приложението. В него се намират методите за:

- стартиране на конзолното меню (Start)
- криптиране на текст (Encrypt)
- декриптиране на текст (Decrypt)

2.2 Метод Start()

Методът Start е говори с потребителя чрез конзолата и задейства всичко. Той съдържа:

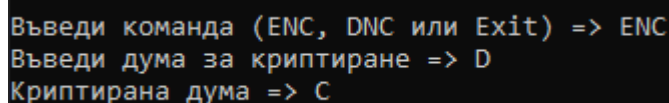
1. Инструкции – обяснява на потребителя какви команди са налични:
 - ENC – за криптиране.
 - DNC – за декриптиране.
 - Exit – за изход.
2. Главен цикъл while (true) – безкраен цикъл, който позволява на потребителя многократно да въвежда команди, докато не въведе „Exit“.
3. Обработка на командите:
 - Exit – когато потребителят въведе „Exit“, програмата прекъсва цикъла.
 - ENC – при въвеждане на тази команда потребителят трябва да въведе дума, която ще бъде криптирана с метода Encrypt.
 - DNC – при тази команда потребителят трябва да въведе вече криптирана дума, която ще бъде декриптирана чрез метода Decrypt.
 - Невалидна команда – при всяка друга не валидна въведена стойност се извежда съобщение за грешка.

2.3 Метод: Encrypt()

Методът за криптиране получава дума, която обработва:

1. Създаване на символен масив – с дължината на въведения текст.
2. Цикъл през всеки символ на думата – всеки символ се трансформира, като от неговата ASCII стойност се извади $(i + 1)$, където i е позицията му в думата.

Пример: ако първият символ е 'D' (ASCII 68), новата стойност ще бъде $68 - 1 = 67$, което отговаря на символа 'C' (4фиг.).



```
Въведи команда (ENC, DNC или Exit) => ENC
Въведи дума за криптиране => D
Криптирана дума => C
```

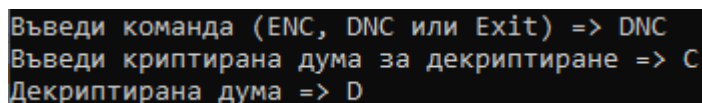
(4фиг.)

3. Връщане на резултата – преобразуваният масив от символи е върнат като нова криптирана дума.

2.4 Метод: Decrypt()

Методът за декриптиране работи по обратния начин:

1. Създаване на масив – със същата дължина като въведения криптиран низ.
2. Цикъл през всеки символ – този път се извършва добавяне на $(i + 1)$ към ASCII стойността на всеки символ (5фиг.).



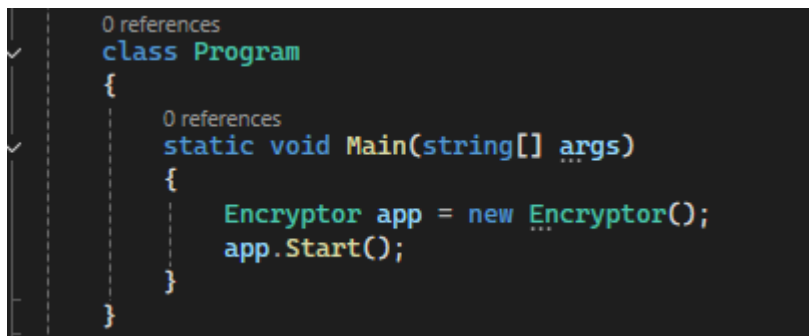
```
Въведи команда (ENC, DNC или Exit) => DNC
Въведи криптирана дума за декриптиране => C
Декриптирана дума => D
```

(5фиг.)

3. Връщане на резултата – новият низ се връща като декриптирана дума.

2.5 Клас: Program

Това е класът, съдържащ метода Main, който е първата точка на всяка C# програма (6фиг.).

A screenshot of a code editor showing the C# code for the Program class. The code is as follows:

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Encryptor app = new Encryptor();
        app.Start();
    }
}
```

The code is color-coded: 'class' is green, 'Program' is blue, 'static' is blue, 'void' is blue, 'Main' is blue, 'string[]' is blue, 'args' is blue, 'Encryptor' is green, 'app' is blue, 'new' is blue, and 'Start()' is blue. There are two '0 references' labels above the class and method definitions.

(6фиг.)

В Main:

- Създава се нов обект от тип Encryptor.
- Извиква се методът Start който стартира всичко.

3. Логика на криптирането с ASCII

ASCII таблицата е стандартна таблица, съдържаща символите, които компютрите използват, като на всеки символ е съответствана конкретна стойност (От 0 до 127). Програмата се възползва от това като променя символите чрез математически операции със техните ASCII стойности (7фиг.).

Пример:

- Вход: "TEST"
- ASCII стойности: T(84), E(69), S(83), T(84)
- Криптиране:
 - Символ 0: $84 - 1 = 83 \rightarrow 'S'$
 - Символ 1: $69 - 2 = 67 \rightarrow 'C'$
 - Символ 2: $83 - 3 = 80 \rightarrow 'P'$
 - Символ 3: $84 - 4 = 80 \rightarrow 'P'$
- Резултат: "SCPP"

Обратно, декриптирането на "SCPP" ще върне "TEST".

ASCII Code Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

(7фиг.)

4. Използване на Дебъдера

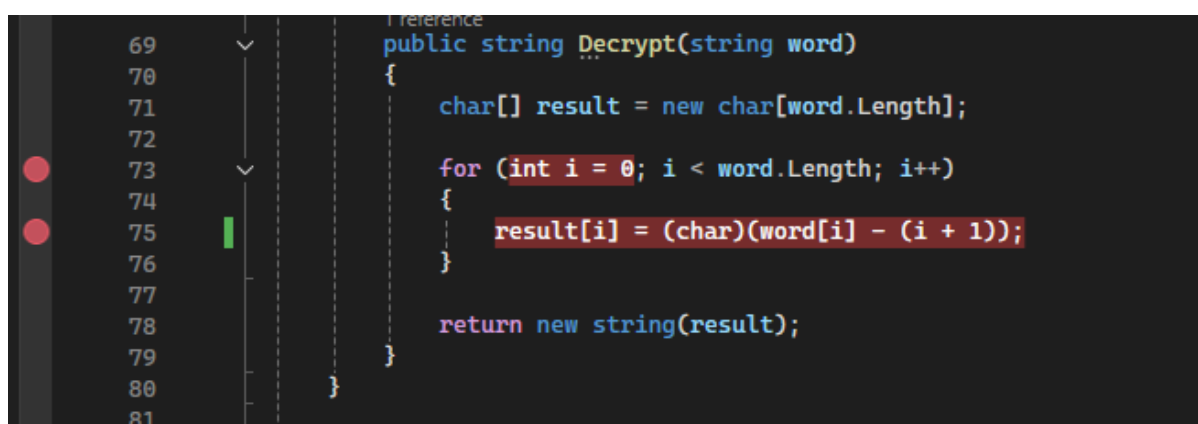
При разработката на програмата възникна проблем със декриптирането на думи. След въвеждане на криптирана дума, резултатът от декриптирането не съвпаднаше с оригиналната дума. За да открия и коригирам причината за грешката използвах дебъгер

Методът Decrypt трябва да прави обратното на Encrypt, тоест да добавя $(i + 1)$ към всеки символ, за да върне оригиналната стойност. По погрешка бях копирали същия код от Encrypt, където стойността се изважда.

Това водеше до ситуация, при която и криптирането, и декриптирането използват една и съща операция – изваждане:

За да открия къде е проблемът, стартирах кода в Debug режим и направих (8фиг.):

1. Поставяне на breakpoint – в метода Decrypt, вътре в for цикъла.
2. Проследих стойностите – след стартиране, наблюдавах стойностите на:
 - `word[i]` – текущия символ.
 - $(i + 1)$ – стойността която трябва да бъде добавена.
 - `result[i]` – изходния символ.
3. Забелязах, че стойността на `result[i]` продължава да намалява.



The screenshot shows a code editor with a debugger interface. On the left, a list of line numbers from 69 to 81 is visible. A red circle breakpoint is set at line 75. The code in the editor is as follows:

```
public string Decrypt(string word)
{
    char[] result = new char[word.Length];

    for (int i = 0; i < word.Length; i++)
    {
        result[i] = (char)(word[i] - (i + 1));
    }

    return new string(result);
}
```

(8фиг.)

5. Заключение

Като цяло, чрез този проект запознах с начина, по който се използва ASCII таблицата, за преобразуване на символите, това как да създавам методи за криптиране и декриптиране на текст и още много. Имах възможност, ако нещо липсваше, да усетя още веднъж на лично ниво, колко важни са определени неща като логиката на операциите, защото дори грешка относно знака на математическа операция, променя резултата напълно.

Научих и как се използва дебъгера във Visual Studio и чрез breakpoint мога да следя какво точно се случва във всяка стъпка на моята програма. Това доведе до откриване и отстраняване на грешката в моя метод за декриптиране. Осъзнах също и как е полезно да структурирам кода си правилно, да разделям логиката на методи и класове и да проверявам внимателно моите input данни. Целият процес ме направи по-добър в писането и тестването на C# програми.

6.Използвана литература

- Уикипедия
- Книгата на Наков за C# програмиране
- GitHub
- W3Schools