<table>
<tr><td colspan="3">Politechnika Świętokrzyska w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki</td></tr>
<tr><td colspan="3">Algorytmy i struktury danych – PROJEKT
Informatyka - I rok, Rok akademicki - 2021/2022, semestr - II</td></tr>
<tr><td>I termin</td><td colspan="2">Temat projektu: Szyfr Cezara i szyfr Vigener’a</td></tr>
<tr><td>Grupa: 1ID14B</td><td rowspan="2">Wykonujący:
Marek Supierz,
Andrzej Mysior,
Adrian Nowak</td><td rowspan="2">Ocena:</td></tr>
<tr><td>Data oddania sprawozdania:
04.06.2022</td></tr>
</table>

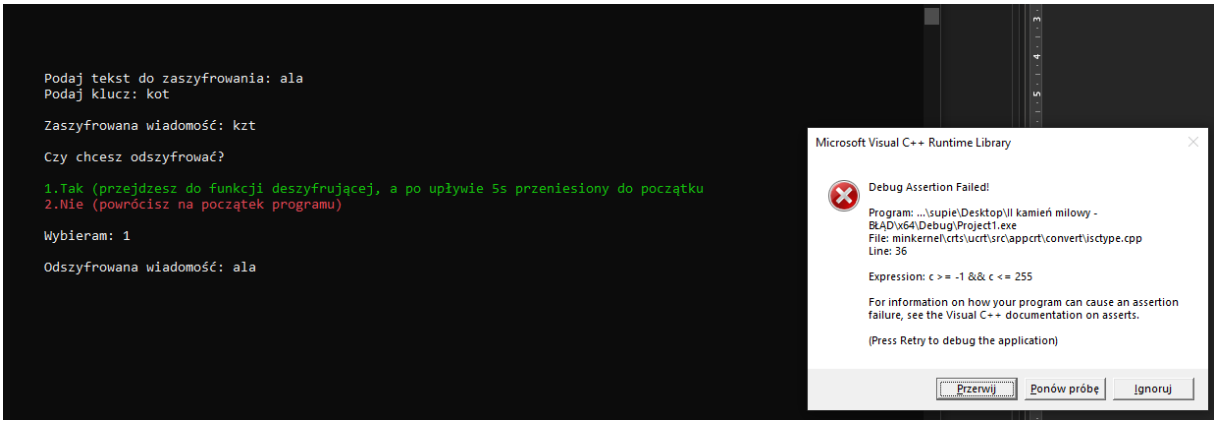
Prace przewidziane na I termin:

1. Dodanie ewentualnych ulepszeń,
2. Optymalizacja i poprawa błędów,
3. Stworzenie dokumentacji w programie Doxygen,
4. Sporządzenie sprawozdania

Github: https://github.com/PSK-projekty/AiSD
Trello: https://trello.com/b/BZRM1xfR/aisd

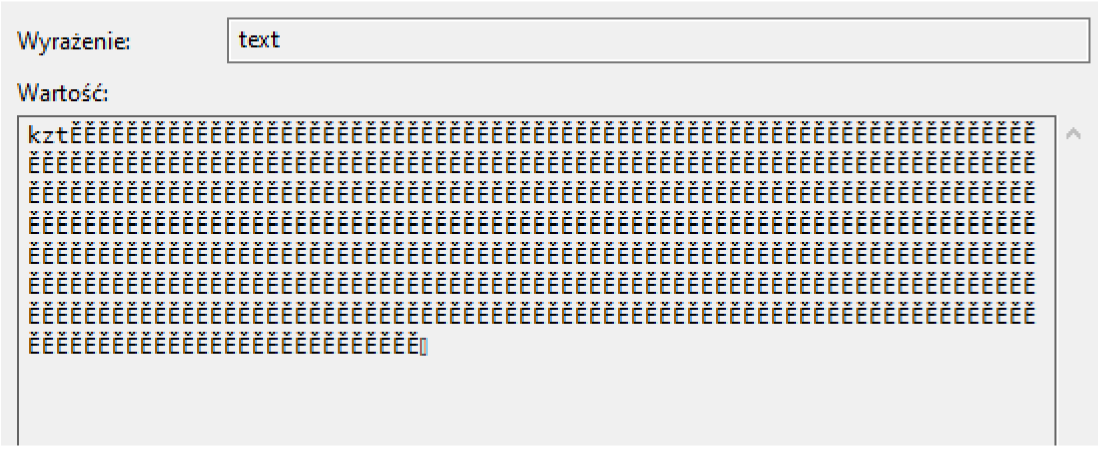
Dodanie ewentualnych ulepszeń

W funkcjach szyfrujących metodą Cezara dodano możliwość natychmiastowej deszyfracji w celu sprawdzenia działania. Niestety nie udało się tego zaimplementować w szyfrze Vigener’a. Objęta metoda działa poprawnie lecz występuje błąd:



Uniemożliwia on poprawne działanie programu, który po wypisaniu odszyfrowanej wiadomości ma odczekać 5 sekund a następnie wrócić do menu głównego.

W procesie debugowania można zauważyć, że zmienna ‘text’ jest przepełniona. Po zaczerpnięciu wiedzy, można stwierdzić, że błąd ten jest najprawdopodobniej spowodowany przez metody ‘islower’, ‘isupper’, ‘isalpha’.



Poprawa błędów

Naprawiono błędne działanie szyfru Vigener’a. Przyczyną wystąpienia błędu było pospieszne działanie programisty.
Nie udało się naprawić błędu związanego z przepełnieniem zmiennej ‘tekst’.

Stworzenie dokumentacji w programie Doxygen

Do kodu programu dołączono komentarze a następnie stworzono dokumentację w programie Doxygen.

Podsumowanie prac przewidzianych na I termin

Naprawiono algorytm odpowiadający za szyfrowanie i deszyfrowanie metodą Vigener'a. Częściowo udało się usprawnić sprawdzanie poprawności działania algorytmów poprzez wywołanie deszyfrowania, niestety dla szyfru Vigener'a nie udało się tego zaimplementować. Spełniono założenia etapu oraz projektu.

Podsumowanie projektu

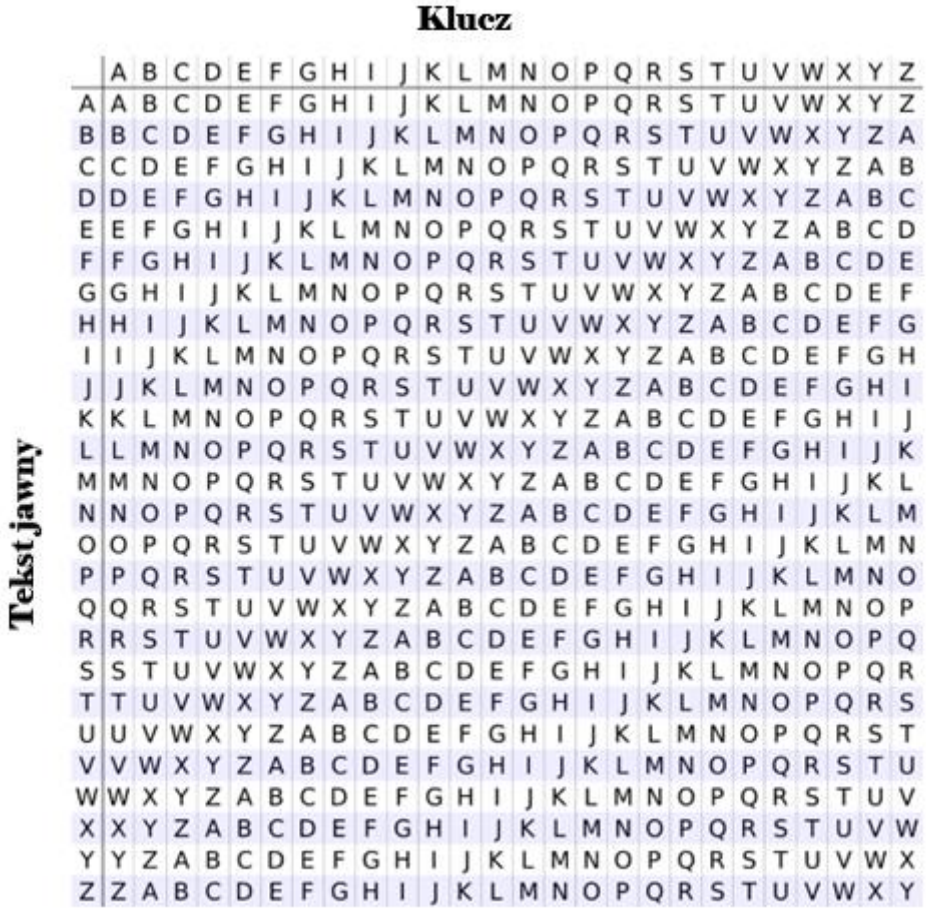
Opis tematyki projektu

Tematem projektu jest szyfr Cezara oraz szyfr Vigener'a. Szyfr sam w sobie jest systemem umownych znaków stosowanym w celu zatajenia wiadomości w taki sposób, aby była ona przynajmniej trudna i czasochłonna.

Szyfr Cezara jest szyfrem przesuwnym, w którym każdemu znakowi w szyfrogramie, czyli wyniku szyfrowania wiadomości, odpowiada dokładnie jeden znak tekstu jawnego przesunięty o określoną, stałą liczbę znaków w alfabecie. Tak więc tekst 'Ala ma kota' przesunięty o 3 wygląda następująco: 'Dod pd nrwd'. Aby odszyfrować taki tekst musimy znać wartość o jaką został przesunięty tekst jawny. Jeżeli tekstem jawnym będzie 'z' a przesunięcie będzie równe 1, otrzymamy literę 'a'. Najłatwiej jest to zobrazować na tablicy ASCII. Bez „zapętlenia" każda próba przekroczenia zakresu powodowała by błąd.

LP.	Hex	Znak	LP.	Hex	Znak	LP.	Hex	Znak	LP.	Hex	Znak	LP.	Hex	Znak
1	**0x0**	NUL	27	**0x1A**	SUB	53	**0x34**	4	79	**0x4E**	N	105	**0x68**	h
2	**0x1**	SOH	28	**0x1B**	ESC	54	**0x35**	5	80	**0x4F**	O	106	**0x69**	i
3	**0x2**	STX	29	**0x1C**	FS	55	**0x36**	6	81	**0x50**	P	107	**0x6A**	j
4	**0x3**	ETX	30	**0x1D**	GS	56	**0x37**	7	82	**0x51**	Q	108	**0x6B**	k
5	**0x4**	EOT	31	**0x1E**	RS	57	**0x38**	8	83	**0x52**	R	109	**0x6C**	l
6	**0x5**	ENQ	32	**0x1F**	US	58	**0x39**	9	84	**0x53**	S	110	**0x6D**	m
7	**0x6**	ACK	33	**0x20**	spacja	59	**0x3A**	:	85	**0x54**	T	111	**0x6E**	n
8	**0x7**	BEL	34	**0x21**	!	60	**0x3B**	;	86	**0x55**	U	112	**0x6F**	o
9	**0x8**	BS	35	**0x22**	"	61	**0x3C**	<	87	**0x56**	V	113	**0x70**	p
10	**0x9**	HT	36	**0x23**	#	62	**0x3D**	=	88	**0x57**	W	114	**0x71**	q
11	**0x0A**	LF	37	**0x24**	$	63	**0x3E**	>	89	**0x58**	X	115	**0x72**	r
12	**0x0B**	VT	38	**0x25**	%	64	**0x3F**	?	90	**0x59**	Y	116	**0x73**	s
13	**0x0C**	FF	39	**0x26**	&	65	**0x40**	@	91	**0x5A**	Z	117	**0x74**	t
14	**0x0D**	CR	40	**0x27**	'	66	**0x41**	A	92	**0x5B**	[	118	**0x75**	u
15	**0x0E**	SO	41	**0x28**	(	67	**0x42**	B	93	**0x5C**	\	119	**0x76**	v
16	**0x0F**	SI	42	**0x29**	)	68	**0x43**	C	94	**0x5D**	]	120	**0x77**	w
17	**0x10**	DLE	43	**0x2A**	*	69	**0x44**	D	95	**0x5E**	^	121	**0x78**	x
18	**0x11**	DC1	44	**0x2B**	+	70	**0x45**	E	96	**0x5F**	_	122	**0x79**	y
19	**0x12**	DC2	45	**0x2C**	,	71	**0x46**	F	97	**0x60**	`	123	**0x7A**	z
20	**0x13**	DC3	46	**0x2D**	-	72	**0x47**	G	98	**0x61**	a	124	**0x7B**	{
21	**0x14**	DC4	47	**0x2E**	.	73	**0x48**	H	99	**0x62**	b	125	**0x7C**	|
22	**0x15**	NAK	48	**0x2F**	/	74	**0x49**	I	100	**0x63**	c	126	**0x7D**	}
23	**0x16**	SYN	49	**0x30**	0	75	**0x4A**	J	101	**0x64**	d	127	**0x7E**	~
24	**0x17**	ETB	50	**0x31**	1	76	**0x4B**	K	102	**0x65**	e	128	**0x7F**	DEL
25	**0x18**	CAN	51	**0x32**	2	77	**0x4C**	L	103	**0x66**	f			
26	**0x19**	EM	52	**0x33**	3	78	**0x4D**	M	104	**0x67**	g			

Szyfr Vigener'a jest rozwinięciem szyfru Cezara. Tekst jawny nie jest szyfrowany przez przesunięcie a przez klucz. Każdą literę tekstu jawnego szyfrujmy korzystając z tablicy Vigener'a.



Aby zaszyfrować wiadomość należy wziąć literę z tekstu oraz literę z klucza. Na ich przecięciu znajdzie się zaszyfrowana litera.
Proces ten jest dobrze pokazany na stronie: https://studio.code.org/s/vigenere/lessons/1/levels/1
Aby odszyfrować tekst należy znać klucz i postępować odwrotnie do szyfrowania.

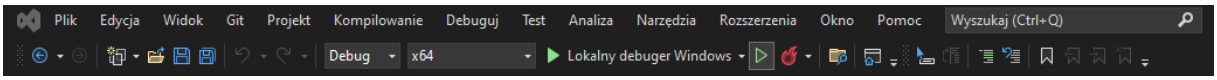
Celem projektu było stworzenie programu umożliwiającego szyfrowanie i deszyfrowanie dwoma wyżej opisanymi metodami.

Użyty język, biblioteki, OS, IDE

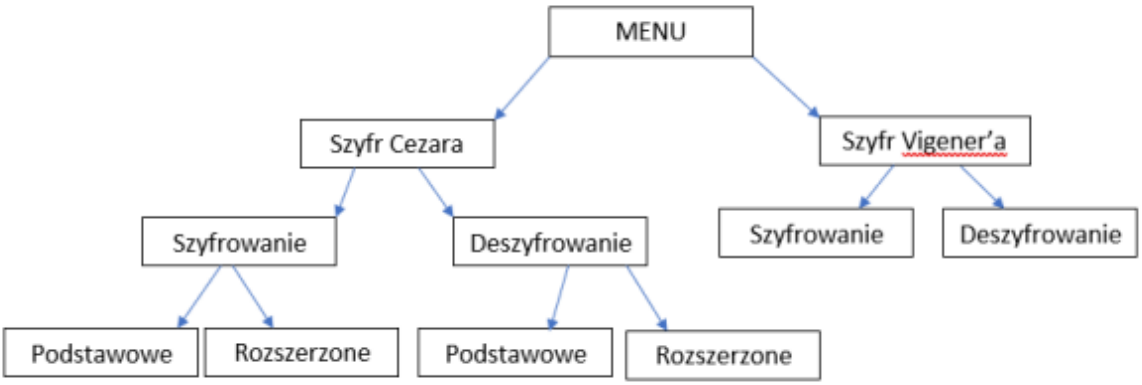
Jedynym językiem użytym w projekcie jest język C.
Użyto bibliotek: stdio.h, windows.h, string.h, ctype.h
Pracowano i testowano pod systemem Windows 10
Posługiwano się środowiskiem programistycznym Visual Studio 2022

Instrukcja kompilacji/uruchomienia, opis działania i instrukcja obsługi

Aby uruchomić program przy użyciu Visual Studio 2022 należy otworzyć plik ‘Szyfry.sln’ a następnie uruchomić przy pomocy kombinacji klawiszy Ctrl+F5, lub poprzez naciśnięcie zielonego trójkąta na górnym pasku.



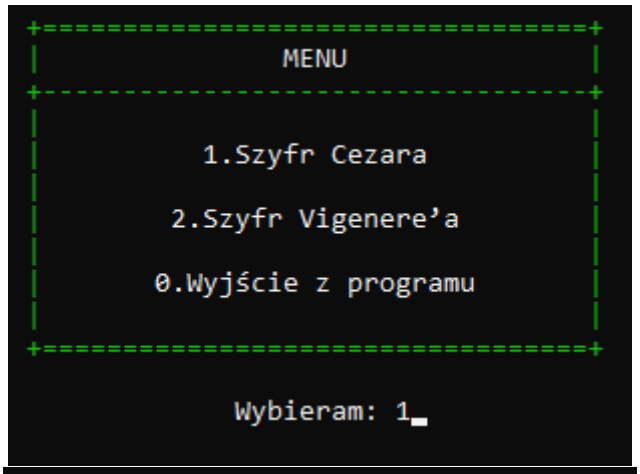
Program opiera się na zagnieżdżonych instrukcjach switch. Przechodząc przez kolejne możliwości użytkownik dostaje się do danej funkcji.

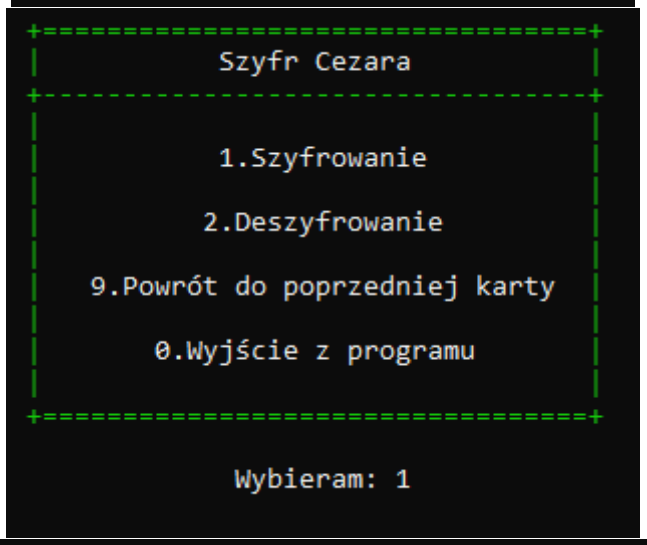


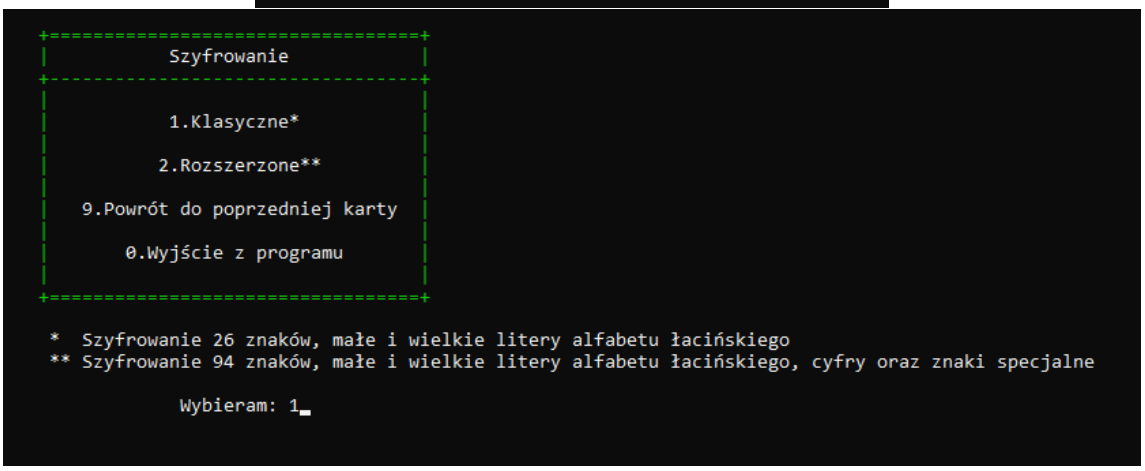
Szyfr Cezara został rozbity na dwie możliwości. „Podstawowy” to klasyczny szyfr Cezara pracujący na wielkich i małych literach alfabetu łacińskiego. „Rozszerzony” to szyfr Cezara pracujący na tablicy ASCII od ! do ~.

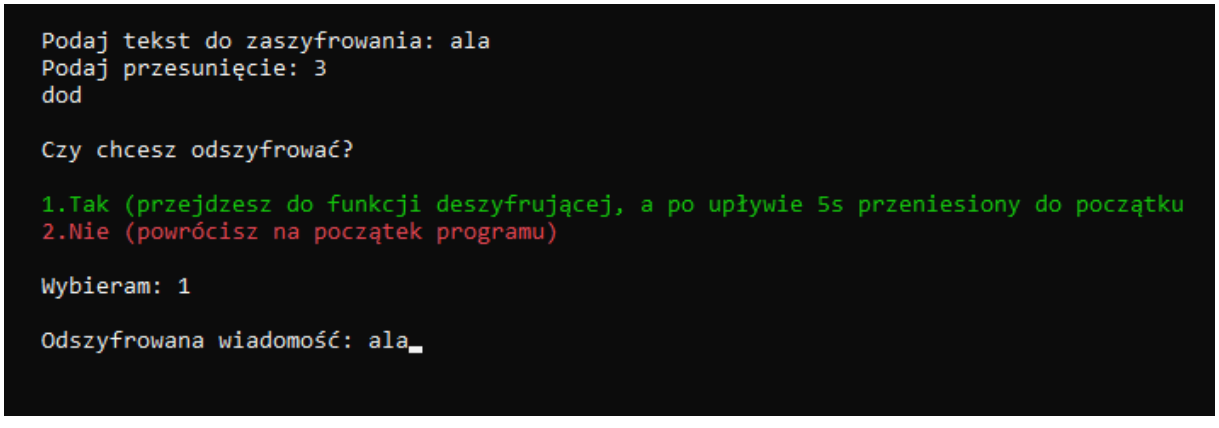
Obsługa programu ogranicza się do wybierania opcji, a ostatecznie do podania tekstu oraz klucza lub przesunięcia.

Zrzuty ekranu z przykładowym działaniem









Opis użytych algorytmów i najważniejsze fragmenty implementacji

Klasyczny szyfr Cezara - szyfrowanie

```
void PSC_encrypt(char text[], int shift) {

    for (int i = 0; text[i] != 0; ++i) {
        if (text[i] >= 'A' && text[i] <= 'Z') {

            text[i] -= 'A';
            text[i] += shift;
            text[i] = text[i] % 26;
            text[i] += 'A';

        }

        if (text[i] >= 'a' && text[i] <= 'z') {

            text[i] -= 'a';
            text[i] += shift;
            text[i] = text[i] % 26;
            text[i] += 'a';

        }
    }
    printf("\t%s", text);
}
```

Funkcja przyjmuje 2 parametry, tekst i przesunięcie podane przez użytkownika. W ciele funkcji znajduje się pętla iterująca aż do ostatniego elementu w zmiennej text. W pętli znajdują się dwie instrukcje if sprawdzające czy znak jest wielką czy małą literą. W instrukcjach if odbywa się szyfrowanie. Najpierw odejmowana jest dolna granica przedziału, następnie dodawane przesunięcie. W kolejnym kroku na zmiennej text wykonywana jest operacja dzielenia z resztą przez 26 czyli ilość wielkich lub małych liter. Na końcu dodawana jest dolna granica przedziału.

Klasyczny szyfr Cezara - deszyfrowanie

```
void PDC_decrypt(char text[], int shift) {

    for (int i = 0; text[i] != 0; ++i) {

        if (text[i] >= 'A' && text[i] <= 'Z') {
            text[i] -= shift;

            if (text[i] < 'A')
                text[i] += 'Z' - 'A' + 1;
        }

        if (text[i] >= 'a' && text[i] <= 'z') {
            text[i] -= shift;

            if (text[i] < 'a')
                text[i] += 'z' - 'a' + 1;
        }
    }
    printf("\n\tOdszyfrowana wiadomość: %s", text);
}
```

Funkcja przyjmuje 2 parametry, tekst i przesunięcie podane przez użytkownika. W ciele funkcji znajduje się pętla iterująca aż do ostatniego elementu w zmiennej text.W pętli znajdują się dwie instrukcje if sprawdzające czy znak jest wielką czy małą literą. W instrukcjach if odbywa się deszyfrowanie. Od tekstu odejmowane jest przesunięcie a jeżeli jest to litera na brzegu przedziału od górnej granicy odejmowana jest dolna a na końcu dodawane 1.

Rozszerzony szyfr Cezara działa w ten sam sposób.

Szyfr Vigener'a - szyfrowanie

```
void SV_encrypt(char* text, char* key) {

    char encrypted_text[500];
    int key_length = strlen(key);

    printf("\n\tZaszyfrowana wiadomość: ");

    for (int i = 0; i < strlen(text); i++) {

        if (islower(text[i]))
            encrypted_text[i] = ((int)text[i] - 97 + (int)tolower(key[i %
key_length]) - 97) % 26 + 97;

        else
            encrypted_text[i] = ((int)text[i] - 65 + (int)toupper(key[i %
key_length]) - 65) % 26 + 65;

        if (isalpha(text[i]))
            printf("%c", encrypted_text[i]);

        else
            printf("%c", text[i]);

     }
}
```

Ponownie, funkcja przyjmuje dwa parametry. W ciele funkcji znajduję się pętla iterująca do ostatniego elementu. W niej sprawdzane jest czy dana litera jest wielka czy mała. Następnie odbywa się szyfrowanie.

Formuła do szyfrowania wygląda następująco:
zaszyfrowana_wiadomość[i] = (tekst[i] + klucz[i mod dlugosc_klucza]) mod 26.
Zapis [i mod dlugosc_klucza] ma na celu obsługę przypadku gdy klucz jest krótszy od szyfrowanej wiadomości (po przekroczeniu długości klucza wraca do początku). Aby lepiej zrozumieć zasadę działania prześledźmy ją na przykładzie:

tekst = ala
klucz = kot
prawidlowy_wynik = kzt

Przypisujemy literom alfabetu kolejne numery: a=0 b=1 c=2 ... z=25
Postępujemy zgodnie z formuła

(0+10) mod 26 = 10 -> k
(11+14) mod 26 = 25 -> z
(0+19) mod 26 = 19 -> t

Aby moc pracować na liczbach musimy zamienić typ zmiennej z char na int, uzyskamy to w następujący sposób
zaszyfrowana_wiadomosc[i] = ((int)tekst[i] + (int)(key[i mod dlugosc_klucza])) % 26;

Ponieważ pracujemy na tablicy ASCII trzeba odjąć dolna granice przedziału w jakim znajdują się małe lub wielkie litery.

Szyfr Vigener'a - deszyfrowanie

```
void DV_decrypt(char* text, char* key) {

    char decrypted_text;
    int key_length = strlen(key);
    printf("\n\tOdszyfrowana wiadomosc: ");

    for (int i = 0; i < strlen(text); i++) {

        if (islower(text[i]))
            decrypted_text = (((text[i] - 97) - (tolower(key[i % key_length])
- 97)) + 26) % 26 + 97;
        else
            decrypted_text = (((text[i] - 65) - (toupper(key[i % key_length])
- 65)) + 26) % 26 + 65;

        if (isalpha(text[i]))
            printf("%c", decrypted_text);

        else
            printf("%c", text[i]);
    }

}
```

Funkcja ta działa niemalże identycznie do funkcji szyfrującej. Jedynymi różnicami jest odejmowanie klucza od tekstu oraz dodatkowe odejmowanie i dodawanie dolnego krańca przedziału

Podział prac w zespole

Prace zostały podzielone w następujący sposób:

Marek Supierz: programowanie, przygotowanie i zarządzanie repozytorium, testowanie, poprawa błędów, wsparcie przy komentarzach

Andrzej Mysior: Testowanie, przeniesienie harmonogramu do Trello, wsparcie programistyczne, komentarze, przygotowanie dokumentacji

Adrian Nowak: Testowanie, wyszukiwanie potrzebnych zasobów, wsparcie programistyczne, sporządzenie sprawozdania

Podsumowanie

Udało się stworzyć program który poprawnie wykonuje operacje szyfrowanie i deszyfrowania metodami Cezara i Vigener'a. Możliwymi dalszymi pracami było by stworzenie możliwości wywołania funkcji deszyfrującej metodą Vigener'a bez konieczności wcześniejszego wyjścia do menu głównego. Tak jak ma to miejsce w szyfrze Cezara. Można także zmniejszyć ilość używanych plików co wiązało by się z ograniczeniem używanej pamięci na dysku oraz zmniejszyło ilość wywołań funkcji pobierającej dane. Ciekawym doświadczeniem było by porównanie czasów potrzebnych do złamania tych prostych algorytmów z nieco bardziej współczesnymi.

Nie udało się w pełni stworzyć możliwości szybkiego deszyfrowania metodą Vigener'a. (zostało to opisane wyżej)