

Cifra de César

Aluno : Waldyr Turquetti Gonçalves RA:2097842

Resumo — O Seguinte Trabalho é referente a matéria Organização de Computadores Ministrado pelo Prof. Dr. Erikson Freitas de Moraes, apontando o que é a Cifra de Cesar, a lógica da implementação de uma, qual foi a lógica usada na criação do Algoritmo e como se utiliza tal. Todo o Algoritmo foi desenvolvido no ambiente de Programação MARS, com a linguagem ASSEMBLY.

1 INTRODUÇÃO

Na Criptografia, a Cifra de César é um tipo de criptografia que consiste na substituição de um caractere por outro que está uma quantidade (Fator) para frente. Sendo muito utilizado por Caio Júlio César (13 de julho de 100 a.C. – 15 de março de 44 a.C.) no envio de mensagens Militares com o objetivo de proteger o conteúdo se caísse em mãos inimigas. Ainda que o uso deste esquema por César tenha sido o primeiro a ser registrado, é sabido que outras cifras de substituições foram utilizadas anteriormente.

A Cifra funcionava da seguinte maneira: Cada letra do alfabeto é substituído pela letra três posições à frente, ou seja, o “A” é substituído pelo “D”, o “B” pelo “E”, o “C” pelo “F”, e assim sucessivamente. Qualquer código que tenha esse padrão é considerado uma Cifra de César, também conhecida como Código de César.[3]

2 DESENVOLVIMENTO

Como foi especificado no enunciado do trabalho o programa deveria decifrar e codificar letras maiúsculas e minúsculas, números, pontuações e caracteres especiais, então para isso foi utilizado a Tabela ASCII (Figura 1.1) [2]. E assim ficaria mais fácil de mover o ponteiro, pois assim como em C se somarmos uma constante inteira no caractere, ele avança o número de vezes igual a constante, apontando para outro caractere. Sendo assim de exemplo: Suponha que o nosso fator é igual a 3, e o caractere que queremos modificar seria o “R”, como observamos na Tabela Ascii o “R” é equivalente ao número decimal 82, sendo assim quando codificamos o “R” ele passa a ser o “U” que tal é equivalente ao número 85, como representado na Figura 1.2. Do mesmo jeito funciona para decodificar só que ao invés de avançar (Ir para Frente) retorna-se (Vai para Trás).

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figura 1.1 – Tabela ASCII

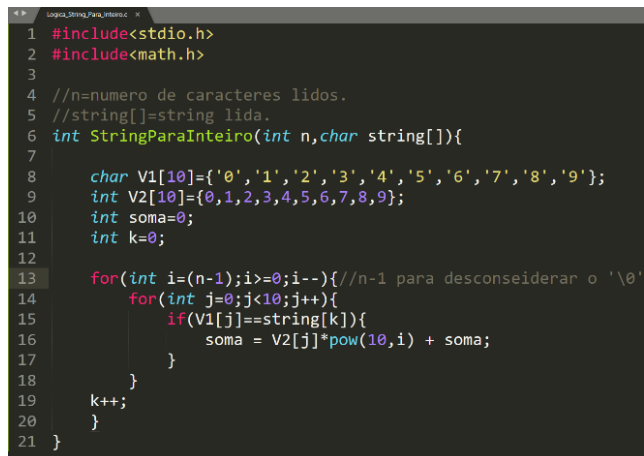
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figura 1.2 – Representação da Codificação do Caractere “R”. Onde “R” é o antigo Caractere e “U” é o novo Caractere.

Vale ressaltar que nesse algoritmo, foi descartados todos os caracteres representados do 0 ao 32 e o 127, por serem caracteres não vistos quando se escreve um texto em algum editor. Outro detalhe é que quando escreves frases, tanto para codificar quanto para decodificar o espaço é ignorado, ou seja, o espaço não é codificado ou decodificado, Exemplo: Supondo que fosse desejado codificar a frase “oi tudo bem?” com um fator=3, a saída seria “rl wxgr ehpb”, pois ignoramos o espaço e codificamos só os caracteres vistos quando digitamos a frase. Outra Preocupação foi em casos em que pudesse ocorrer de estourar o valor da Tabela Ascii, Como se quisemos codificar o caractere ‘~’ (em decimal 126) e com um fator igual a 2 aparecendo algo não desejado, então para isso quando vai estourar o valor é feito que o ponteiro aponte para o 33 novamente e assim continuar percorrendo a tabela ascii.

Agora que foi mostrado todas as restrições do código irá ser falado sobre a lógica utilizada na implementação. No código codifica ou decodifica caractere por caractere substituindo o antigo caractere pelo novo diretamente no endereço dele através de dois loops, um dentro do outro, onde o primeiro (o loop de fora) é o responsável por andar pela palavra sendo sua condição de parada o ‘\0’, e o segundo loop é o responsável por achar o número em decimal que o caractere da palavra original vale sendo esse valor somado com o conteúdo de um registrador temporário (esse conteúdo é igual a 32) que em seguida ocorre outra soma com o fator. Logo após isso fazemos uma verificação para saber se esse valor ultrapassa ou não o 126, se não ultrapassar simplesmente somasse o fator com o caractere original virando o novo caractere e em seguida usa um Store Byte (sb) para substituir o novo pelo o antigo, caso ultrapasse pegamos esse valor e subtraímos 126 dele, após isso pegamos o primeiro caractere do Vetor Ascii (!) e somamos o resto dessa subtração e fazemos outra verificação e se ultrapassar subtraímos 32, e logo em seguida, igual ao primeiro caso usamos o Store Byte. Outro detalhe importante é que transformamos a string do fator em um número, e logo colocamos esse valor em um registrador, verificamos se esse fator é maior que 126, caso for, utilizamos a pseudo Instrução rem que simplesmente pega o resto da divisão, essa divisão é do fator por 126 e salva esse resto no registrador original do fator, caso não for maior não é feito nada, isso é feito para evitar problemas de estourar o vetor mais de uma vez.

Lógica Utilizada para transformar a string em um número inteiro na Figura 2.



```

1 #include<stdio.h>
2 #include<math.h>
3
4 //n=numero de caracteres lidos.
5 //string[]=string lida.
6 int StringParaInteiro(int n,char string[]){
7
8     char V1[10]={'0','1','2','3','4','5','6','7','8','9'};
9     int V2[10]={0,1,2,3,4,5,6,7,8,9};
10    int soma=0;
11    int k=0;
12
13    for(int i=(n-1);i>=0;i--){//n-1 para desconsiderar o '\0'
14        for(int j=0;j<10;j++){
15            if(V1[j]==string[k]){
16                soma = V2[j]*pow(10,i) + soma;
17            }
18        }
19        k++;
20    }
21 }

```

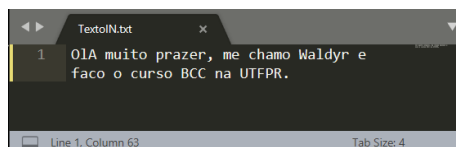
Figura 2: Representação em C da lógica utilizada para transformar a string do fator em um número inteiro.

OBS: A lógica utilizada nesse algoritmo segue o mesmo princípio da codificação e decodificação.

3 UTILIZAÇÃO

O projeto foi feito para ser utilizado com arquivos, onde possui quatro arquivos o “TextoIN.txt”, o “TextoOUT.txt”, o “TextoOpcao.txt” e o “TextoFator.txt” (OBS: É imprescindível que os arquivos tenha esse nome obrigatoriamente) onde cada um se refere respectivamente a frase de entrada, a frase de saída (podendo variar entre Codificado e decodificado), a opção entre codificar ou decodificar a frase colocado no TextoIN sendo 1- Para codificar e 2- Para decodificar (OBS: Nesse arquivo poderá apenas ter um caractere) qualquer caractere diferente irá printar no TextoOut: “Opcao Invalida” e valor do fator (OBS: valor do fator tem sempre que ser positivo, não foi feito tratamento para valores negativos e no máximo um valor de até 9 dígitos).

Exemplo:



```

1 Olá muito prazer, me chamo Waldyr e
  faco o curso BCC na UTFPR.

```

```

1 RoD pXlwr sud}hu/ ph fkdpr Zdog|u h
  idfr r fxuvr EFF qd XWISU1

```

```

1 1

```

```

1 3

```

Fazendo o Cruzamento de testes com o Fernando Mo-crosky:

- 1) Entrada: Canada
Fator:3
Opção:Codificar
Saída: Fdqgdg
- 2) Entrada: rk | | c
Fator:2
Opção: decodificar
Saída: pizza

Outra observação importante é que **não é permitido em nenhum momento na hora de escrever nos arquivos apertar o Enter**, pois a leitura do ‘\n’ causara problemas na finalização da leitura acarretando em não finalizar o programa. Para terminar as restrições de uso o código.asm, o programa MARS e os arquivos txt tem que estar no mesmo diretório, para que não ocorrer problemas.

Depois que escrever nos arquivos é necessário abrir o programa MARS e abrir o código.asm, clicar em “Assembly the corrente file and clear bleackpoints” e em seguida o “Run the current program”. E pronto o resultado estará em TextoOUT.txt .

5 REFERÊNCIAS

- [1] IDE Assembly MARS: <http://courses.missouristate.edu/ken-vollmar/mars/>
- [2] Imagem da Tabela Ascii: <https://pt.wikipedia.org/wiki/Ficheiro:ASCII-Table.svg>
- [3] Prof ° Fernando Souza “Criptografia e a Cifra de Cesar: <http://www.fernandosilva.pro.br/portal/index.php/en/para-pensar/curiosidades/item/214-criptografia-e-a-cifra-de-c%C3%A9sar>

4 RESULTADOS

Três exemplos de funcionamento a seguir:

- 1) TextoIN.txt:Eu gosto de cerveja!!
TextoFator.txt:1029
TextoOpcao.txt:1
TextoOUT.txt: Z, |&*+& yz xz)-z!v66
- 2) TextoIN.txt: s46BA 8; I<74 Fc Fc
TextoFator.txt: 50323
TextoOpcao.txt:2
TextoOUT.txt: Bacon eh vida s2 s2
- 3) TextoIN.txt:Oh Palmeiras Tem Mundial
TextoFator.txt: 5012
TextoOpcao.txt:6
TextoOUT.txt: Opcao Invalida