Universidade de Brasília - UNB

Faculdade Gama - FGA

Disciplina: Estruturas Matemáticas para Computação

Tema: Criptografia

Professora: Fabiana Mendes

Criptografia

Alunos: Cleiton da Silva Gomes 10/0097022
Cristóvão de Lima Frinhani 10/0097235
Fagner Rodrigues 09/0112750
Lucas dos Santos Ribeiro Leite 10/0034462
Ruyter Parente da Costa 10/0122981
Thiago Silveira Honorato 10/0053351
Vanessa Barbosa Martins 10/0131182

Brasília, 11 de junho de 2012

SUMÁRIO

INTRODUÇÃO

As mensagens cifradas ajudaram a definir o rumo da humanidade ao longo de séculos. Desde os Hieróglifos Egípcios, a briga das Rainhas da Inglaterra e da Escócia e mais recentemente nas 1ª e 2ª guerras mundiais, as mensagens cifradas foram usadas para esconder segredos e dificultar que os inimigos desvendassem as táticas a serem usadas na guerra.

A Criptografia é uma técnica que consiste cifrar uma mensagem ou um arquivo usando um conjunto de cálculos. O arquivo cifrado torna-se incompreensível até que seja desencriptado. Os cálculos usados para encriptar ou desencriptar o arquivo são chamados de chaves. Apenas alguém que tenha a chave poderá ler o arquivo criptografado. [1][2]

O código foi feito na linguagem c, CRIP_EMC, e é basicamente é formado por três procedimentos junto com a função principal, é claro. O funcionamento da função principal (main) e desses procedimentos serão detalhados no decorrer desse documento.

A função principal, basicamente, recebe a palavra que o usuário digitar através da variável frase [TAMANHO] (tipo char). Essa variável guarda a frase que será criptografada. No processo de criptografia que entram os procedimentos.

No código existem três procedimentos. Os nomes deles são: CRIPT_01(char frase[]), CRIPT_02(char frase[]) e CRIPT_03(char frase[]). O argumento do procedimento é a variável já mencionada declarada no main que guarda a frase que o usuário digitar.

O procedimento CRIPT_01(char frase[]) possui um laço do tipo for para percorrer cada caractere e fazer a criptografia do mesmo.

A figura 01 é um trecho do código CRIP_EMC que mostrar o procedimento CRIPT_01(char frase[]) e o laço for.

```
//Procedimento que faz a criptografia 01
void CRIPT_01(char frase[]){

    //Laço para percotrrer os 20 primeiros caracteres
     for(  i = inicio; i <= parada ; i++)
     {
```

Figura 01 – laço for

As variáveis i, inicio, parada são do tipo globais de acordo com o trecho de código abaixo (figura 02).

```
//Variáveis globais
//Essas variáveis serão incrementadas na função principal e utilizadas nas funções secundárias
int j = 0,inicio = 0,parada = 1; int i;
```

Figura 02 – variáveis globais

O motivo dessas variáveis serem globais é para serem acessas por qualquer função/procedimento e serem incrementadas. Mas, porque serem incrementadas? Simples. Devem ser incrementadas para que seja possível haver outro laço, pois a variável parada, por exemplo, começa em 1 de acordo com a figura 02 e se não for incrementada não pode haver outro laço for dentro do procedimento CRIPT_01(char frase[]).

O processo de criptografia que ocorre dentro do procedimento CRIPT_01(char frase[]) será explicado a seguir. Supondo que o código irá criptografar o caractere a, como o código faria isso? A resposta pode ser entendida através da figura 03:

```
 //Condições para gerar criptografias
 if(frase[i] == 'a' || frase[i] == 'A')
     frase[i] = '%';
```

Figura 03 – condição para gerar criptografia

Todas as letras do alfabeto, números e vários caracteres possuem a condição especifica para criptografar. Só para ficar mais claro, se o usuário digitar: EMC, então temos que no primeiro laço dentro do procedimento o caractere E será criptografado e impresso, depois no segundo laço M será criptografado e impresso e por fim, C será criptografado e impresso. O processo de impressão pode ser visualizado na figura 04.

```
//Impressão das letras criptogradas no primeiro laço
printf("%c",frase[i]);
```

Figura 04 – processo de impresso no procedimento

O trecho de código da figura 05 é simplesmente a condição para a quebra do laço dentro do procedimento CRIPT_01(char frase[]) e incremento das variáveis globais. Pode-se ver j sendo incrementado. Isso ocorre a cada laço.

```
//Incremendo de j
j++;

//Condição para quebrar o laço e imprimir o resultado
if( parada == j){
    //Incremento dessas variáveis para que seja possivel haver um novo loop
    //nesse ou em outro procedimento
    inicio += 1;
    parada += 1;
    break;
}
```

Figura 05 - incremento de j e condição para quebrar o laço e incrementar as variáveis globais

Os outros procedimentos (CRIPT_02(char frase[]) e CRIPT_03(char frase[])) funcionam da mesma maneira que o procedimento CRIPT_01(char frase[]). A diferença está nas condições de criptografia. Em CRIPT_02(char frase[]) a condição para criptografar os caracteres são diferentes . O mesmo ocorre em CRIPT_03(char frase[]). Exemplo disso pode ser visualizado abaixo:

```
//Condição para criptografia no procedimento CRIPT_01(char frase[])

if(frase[i] == 'a' || frase[i] == 'A')

        frase[i] = '%';

//Condição para criptografia no procedimento CRIPT_02(char frase[])

if(frase[i] == 'a' || frase[i] == 'A')

        frase[i] = '&';

//Condição para criptografia no procedimento CRIPT_03(char frase[])

if(frase[i] == 'a' || frase[i] == 'A')

        frase[i] = '-';
```

A figura 06 faz parte da função principal e indica quantos caracteres a pessoa digitou e possui o loop infinito, caso o usuário fique digitando uma frase que contenha mais de 300 caracteres.

```
 //Contando o tamanho da frase que o usuário digitou
 int TAMANHO_FRASE = strlen(frase);
 //Condição para o usuário digitar no máximo 300 caracteres
while(TAMANHO_FRASE > 300){
        printf("\nVoce ultrapassou o numero de caracteres\n");
        printf("\nDigite novamente\n");
        fflush(stdin);
        gets(frase);
}
```

Figura 06 – contando o tamanho da frase e laço while caso o usuário ultrapasse o tamanho da frase

A função principal basicamente recebe a frase digitada pelo usuário, como já mencionado, e chama os procedimentos para fazer a criptografia das frases. A chamada dos procedimentos é feita de maneira "esquisita" como pode ser visto na figura 07. O motivo disso é para dificultar ao máximo que alguém consiga descriptografar, se é que isso seja possível, pois existe a chamada de 300 procedimentos na função principal e cada procedimento faz uma criptografia diferente.

```
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_03(frase);
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_01(frase);
CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_03(frase);    CRIPT_01(frase);    CRIPT_02(frase);
CRIPT_01(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_01(frase);
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_01(frase);

CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_03(frase);
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_01(frase);
CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_03(frase);    CRIPT_01(frase);    CRIPT_02(frase);
CRIPT_01(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_01(frase);
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_01(frase);

CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_03(frase);
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_01(frase);
CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_03(frase);    CRIPT_01(frase);    CRIPT_02(frase);
CRIPT_01(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_01(frase);
CRIPT_02(frase);    CRIPT_01(frase);    CRIPT_03(frase);    CRIPT_02(frase);    CRIPT_01(frase);
//-------------------------------------------------------------------------------------------//
```

Figura 07 – chamada de blocos de procedimentos

O que poderia ser apresentado no início nesse documento e por fim será apresentado agora são as bibliotecas que podem ser visualizadas na figura 08 e a constante TAMANHO também.

```
#import <stdio.h>
#import <conio.h>
#import <stdlib.h>
#import <string.h>
# define TAMANHO 100000
```

Figura 08 – bibliotecas e constante

A biblioteca stdio.h é para usar comandos de entrada e saída. A biblioteca stdlib.h é para uso do system("x"), ou seja, system(“pause”), system(“color”), system (“cls”), etc. A biblioteca string.h é para manipulação de String, ou seja, usamos um vetor de caracteres e necessitamos dessa biblioteca. A biblioteca conio.h permite modificações na interface do DOS. Exemplo mudar cor e da tela e fonte.

Por fim, a constante TAMANHO foi definida como um número grande de acordo com a figura 07 e o código. Porém, isso só é um detalhe. O código possui a condição para controlar essa constante. Essa constante só não poderia ter um valor pequeno para não acarretar erros.

CONCLUSÃO

No início do trabalho de criptografia se teve dificuldades para definir qual seria a ideia que utilizaríamos para cifrar a mensagem, pois cada um teve uma ideia diferente. A princípio, era que cada laço for do código fosse composto por uma ideia, porém a complexidade do código aumentaria e então foi abolida essa ideia. Sendo assim foi escolhido de comum da função principal chamando os procedimentos em blocos para gerar a criptografia.

Quanto à parte do código, foi escolhida uma linguagem estrutural, visto que o projeto era pequeno, e de certa forma simples de ser elaborado. Se utilizássemos Java, linguagem orientada a objetos, seria um pouco mais complicado para definirmos as classes e os objetos, além de não ser necessário no nosso projeto o reuso de código.

A implementação do código em c não gerou dificuldade,visto que no nosso grupo tem pessoas bastante experientes com a linguagem.

Bibliografia

[1]-HEAGREVES, Patrícia. Decifrando os Códigos Secretos. **Mundo Estranho.** São Paulo . Ed ABRIL. N.112, p 16-25, junho.2011.

[2]-HARDWARE.COM.BR. Criptografia. Disponível em: < http://www.hardware.com.br/termos/criptografia>. Acesso em 25/05/2012 as 16:06.