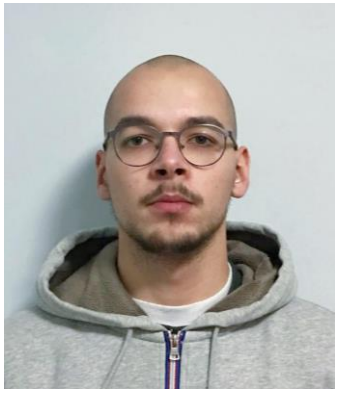
Relatório final

Métodos de programação I

Mestrado integrado em Engenharia de Telecomunicações e informática

"Desencriptador de *Vigenère*"



Hugo Machado

A80362

hudm_7@hotmail.com

Índice:

Introdução

Este projeto visa aprimorar as nossas capacidade de resolver problemas usando a linguagem C.

O problema dado consiste em determinar as varias chaves de desencriptação possíveis para uma mensagem (palavra) encriptada, segundo o cifrador de Vigenère, com um conjunto de desencriptações, geradas pelo programa, que consistem em três números juntos em inglês.

Descrição do problema/solução

Para a resolução deste problema é necessário perceber como o cifrador de *Vigenère* funciona.

Partindo de um dos exemplos que o professor nos forneceu:

```
plaintext message -  p  r  o  g   r  a  m  m   i  n  g
keyword           -  M  I  U  P   M  I  U  P   M  I  U
cipher alphabets  -  13 9  21 16  13 9  21 16  13 9  21
----------------- -  ------------------------------------
encoded message   -  C  A  J  W   E  J  H  C   V  W  B
```

Percebemos que é necessário ter em conta o numero de cada letra, e a cifração consiste na soma dos números da letra, por exemplo:

E		H		M
5	+	8	=	13

Ou seja, a desencriptação consiste na subtração, por exemplo:

H		E		C
8	-	5	=	3

Sabendo que, as letras em *ANSI C* estão ordenadas numa tabela denominada como *ASCII*:

A	65	O	79	a	97	o	111
B	66	P	80	b	98	p	112
C	67	Q	81	c	99	q	113
D	68	R	82	d	100	r	114
E	69	S	83	e	101	s	115
F	70	T	84	f	102	t	116
G	71	U	85	g	103	u	117
H	72	V	86	h	104	v	118
I	73	W	87	i	105	w	119
J	74	X	88	j	106	x	120
K	75	Y	89	k	107	y	121
L	76	Z	90	l	108	z	122
M	77	...	...	m	109	...	...
N	78	...	...	n	110	...	...

Para obter numero da letra em analise vamos ter de subtrair 64 á letra da *string* da mensagem encriptada (porque é inserida em letras maiúsculas)a e 96 á letra da *string* da desencriptação gerada (inserida em letras minúsculas).

Agora quanto as desencriptações possíveis, fiz um gerador das desencriptações que consiste num decrementador de 999 até 0, depois pega no numero e separa-o num array, 999->9,9,9, e depois vai para uma função que passa o numero para extenso, 9,9,9->*nineninenine*.

Após temos percebido como o processo de desencriptação funciona e como gerar as desencriptações, só temos que relacionar as duas *strings*.

A partir do exemplo fornecido pelo professor, do funcionamento do programa:

- **entrada**

```
PQRPQRPQR
```

- **saída**

```
WHTWHTWHT -> sixsixsix
WHTWHTVTC -> sixsixtwo
WHTWHTACM -> sixsixone
WHTVTCWHT -> sixtwosix
WHTVTCVTC -> sixtwotwo
WHTVTCACM -> sixtwoone
WHTACMWHT -> sixonesix
WHTACMVTC -> sixonetwo
WHTACMACM -> sixoneone
VTCWHTWHT -> twosixsix
VTCWHTVTC -> twosixtwo
VTCWHTACM -> twosixone
VTCVTCWHT -> twotwosix
VTCVTCVTC -> twotwotwo
VTCVTCACM -> twotwoone
VTCACMWHT -> twoonesix
VTCACMVTC -> twoonetwo
VTCACMACM -> twooneone
ACMWHTWHT -> onesixsix
ACMWHTVTC -> onesixtwo
ACMWHTACM -> onesixone
ACMVTCWHT -> onetwosix
ACMVTCVTC -> onetwotwo
ACMVTCACM -> onetwoone
ACMACMWHT -> oneonesix
ACMACMVTC -> oneonetwo
ACMACMACM -> oneoneone
```

Percebemos que a desencriptação precisa de ter o mesmo número de carateres que a mensagem encriptada, para poder gerar a chave.

Após estudar estes parâmetros já somos capazes de resolver o problema sem grandes dificuldades.

Algoritmo não refinado

1- Ler a mensagem encriptada.

2- Contar o numero de carateres da mensagem encriptada.

3- Se houverem carateres em letra minúscula, passa-las para maiúsculas.

4- Função para verificar se os carateres válidos.

> 4.1- Se existir algum carater invalido imprime a seguinte mensagem "`INVALID ENCRYPTED MESSAGE`" e encerra o programa.

5- Decrementar a partir de 999 até 0.

6- Passar o numero de inteiro para extenso em língua inglesa.

7- Contar o numero de carateres do numero por extenso.

8- Verificar se o numero de carateres é igual ao numero de carateres da mensagem encriptada.

> 8.1- Se sim, passa para uma função onde é a mensagem encriptada é desencriptada por um algoritmo matemático carater a carater, e é imprimida a chave juntamente da mensagem desencriptada.
>
> 8.2-Se não, volta ao passo 5.

Algoritmo refinado

1- [Inicio(`main()`)]:

1.1-Ler '`encryptedmessage[]`'.

1.2-Remover o "\n" da '`encryptedmessage[]`'.

1.3-Contar numero de elementos de '`encryptedmessage[]`' e guarda o numero em '`sizeencrytedmess`'.

1.4-Se houver letras minúscula, passa-las para letras maiúsculas.

1.5-Verificar se os elementos da mensagem encriptada são letras (função '`verification()`').

(Verificar se todos os elementos estão entre [65,90] da tabela 'ASCII' e se '`sizeencrytedmess`' está entre [9,15])

1.5.1-Se sim, passar mensagem encriptada e o respetivo tamanho para o 'Primeiro passo da Desencriptação (`decryptionprocess()`).

1.5.2-Se não, imprimir "INVALID ENCRYPTED MESSAGE".

2- [Primeiro passo da Desencriptação (`decryptionprocess()`)]:

2-Ciclo decrementador de 999->0 (atua como um decrementador, não é na realidade um decrementador)

2.1-Separar numero 999 num vetor('`keyword[]`') "{9,9,9}";

2.2-Passar para uma função ('`numerationwords()`');

2.2.1-Converte os algarismos de 0 a 9 as suas correspondentes palavras por extenso 'NINE' a 'ZERO', e concatena em '`w[]`' que fora da função ('`numerationwords()`') e declarado como '`wordformkeyword[]`'.

2.3-Contar numero de elementos de '`wordformkeyword[]`' e guarda em '`sizekeywordform`'.

2.4- Verificar se '`sizekeywordform`' é igual a '`sizeencrytedmess`'.

2.4.1-Se sim, passar para o 'Segundo passo da Desencriptação '`decryptionprint()`'.

2.4.2-Se não, voltar para o parâmetro 2.

3- [Segundo passo da Desencriptação(`decryptionprint()`)]:

(‘`sizeencrytedmess`’ passa a ‘`size`’, ‘`wordformkeyword[]`’ passa a ‘`key[]`’, e ‘`encryptedmessage[]`’ passa a ‘`message[]`’)

3.1-Ciclo para correr cada letra das palavra-passe (‘`key[]`’) e da mensagem encriptada (‘`message[]`’).

3.1.1-Guardar o valor decimal ‘ASCII’ da letra de ‘`message[]`’ em ‘`m`’ (variável do tipo inteiro) e ‘`key[]`’ para ‘`k`’.

3.1.2-Subtrair 64 a ‘`m`’ para obter o respetivo numero da letra (ex: H -> 8) (Subtraímos 64 porque a ‘`message[]`’ está em letras maiúsculas).

3.1.3-Subtrair 96 a ‘`k`’ para obter o respetivo numero da letra (ex: V -> 22) (Subtraímos 96 porque a ‘`key[]`’ está em letras minúsculas).

3.1.4-Subtrair ‘`k`’ a ‘`m`’, e guardar o valor em ‘`m`’.

3.1.5-Se ‘`m`’ for menor ou igual a 0, somamos 26.

3.1.6-Somamos ‘`m`’ com 96, e guardamos o valor em ‘`decryption[]`’.

3.2-Imprimir a mensagem desencriptada ‘`decryption[]`’ letra a letra.

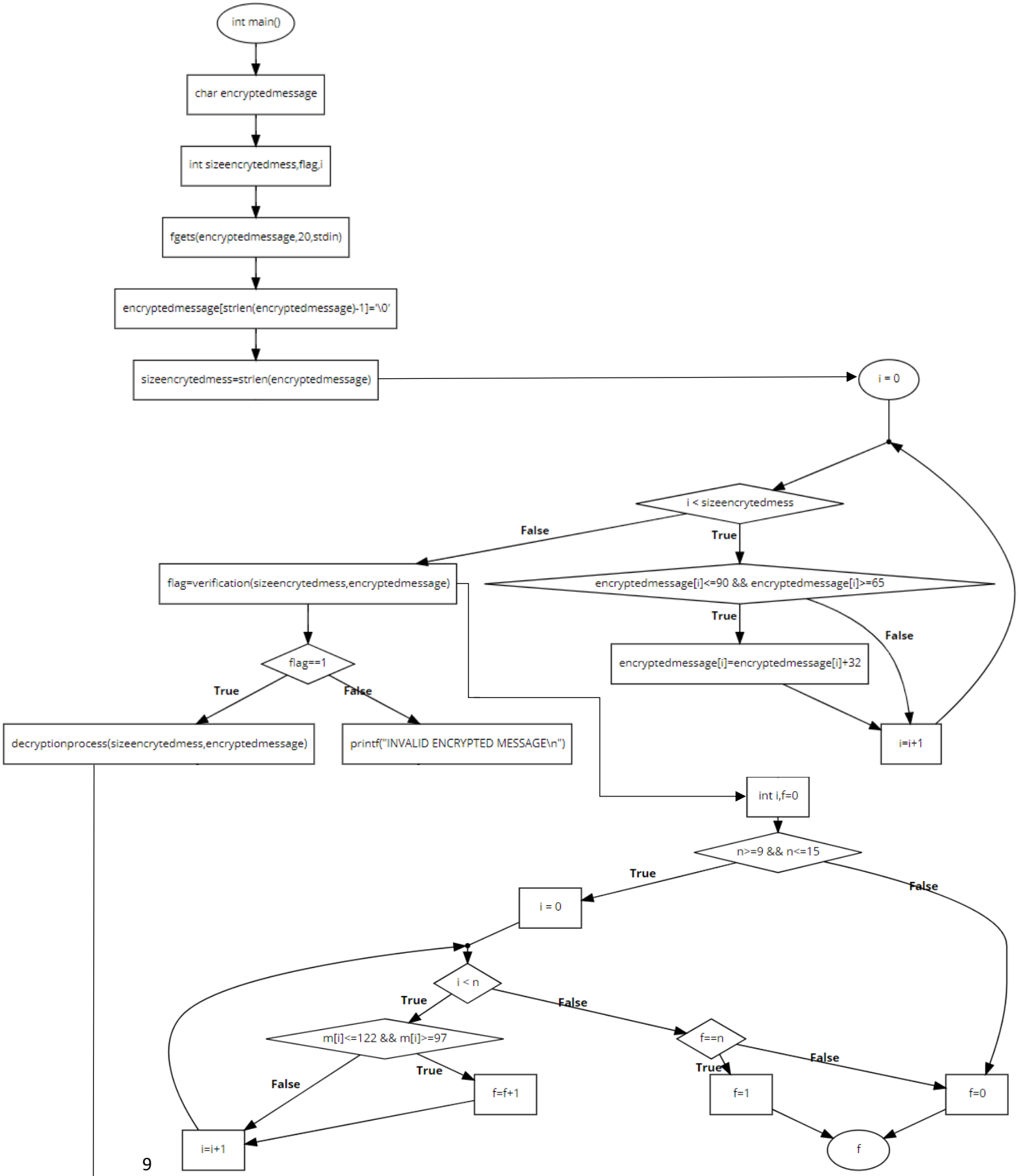
3.3-Imprimir ”->”.

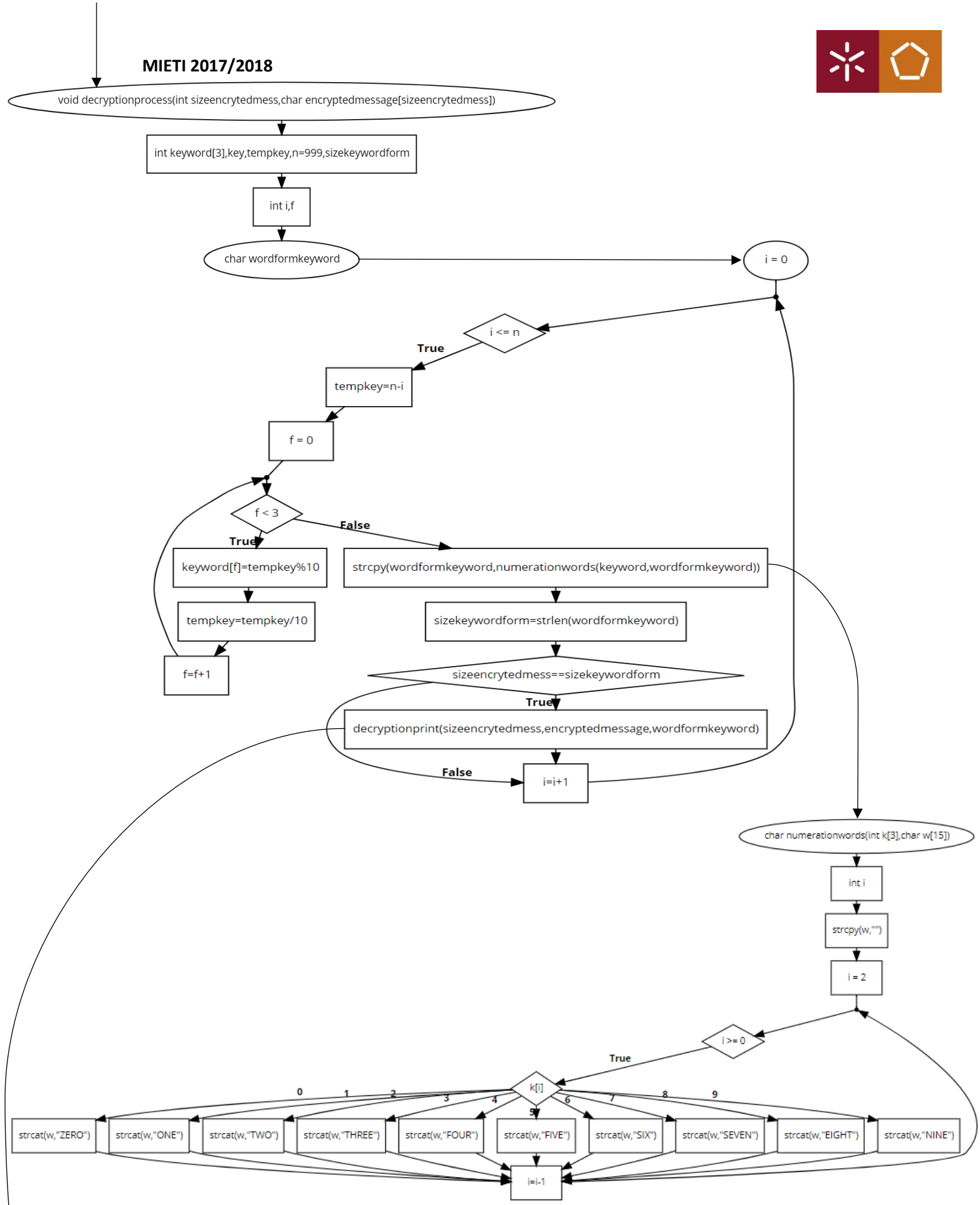
3.4-Imprimir a palavra-passe ‘`key[i]`’ usada na desencriptação, letra a letra.

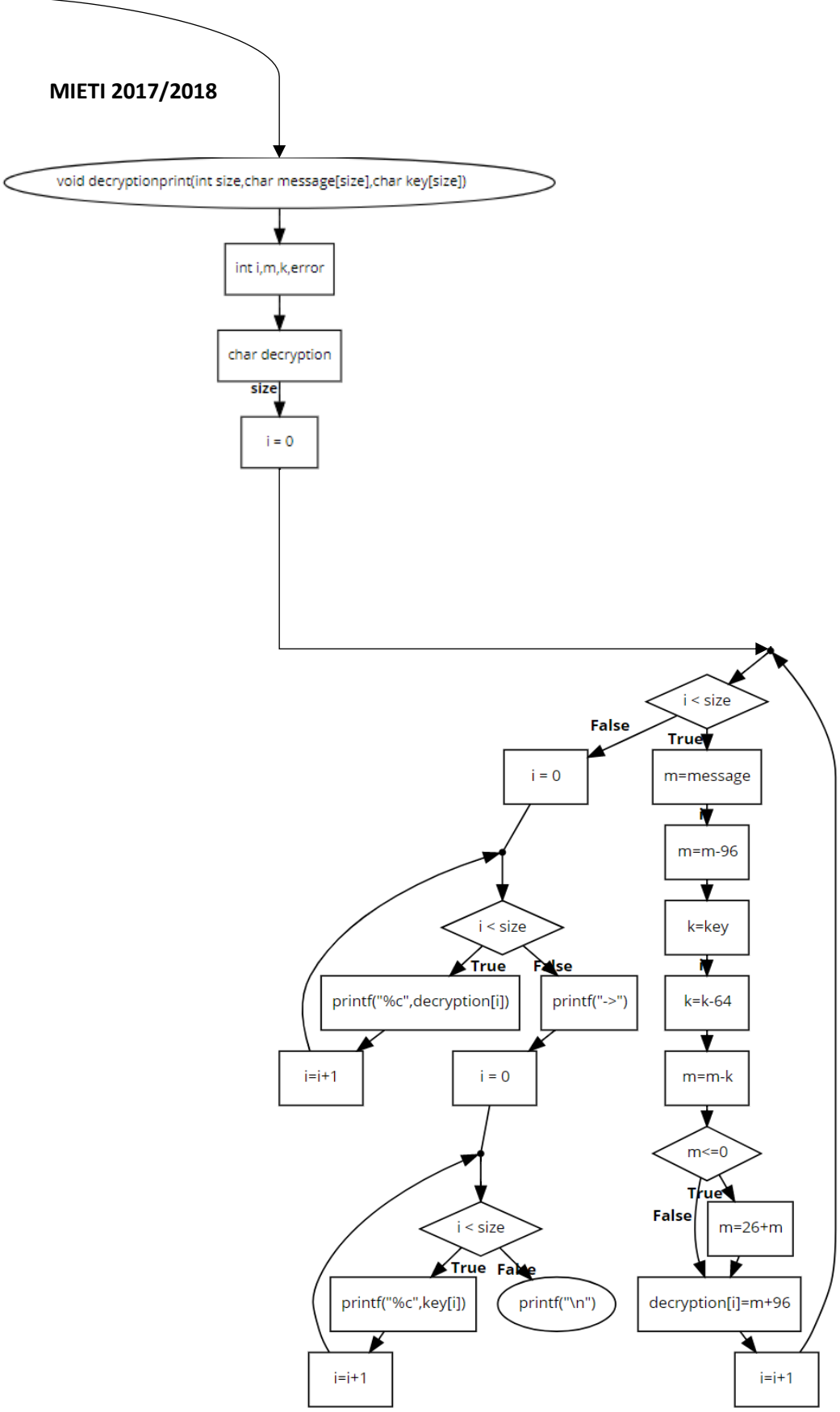
3.5-Imprimir “\n”.

3.6-Voltar para o parâmetro 2.

Fluxograma


int main()
char encryptedmessage
int sizeencrytedmess,flag,i
fgets(encryptedmessage,20,stdin)
encryptedmessage[strlen(encryptedmessage)-1]='\0'
sizeencrytedmess=strlen(encryptedmessage)
i = 0
i < sizeencrytedmess
False
True
flag=verification(sizeencrytedmess,encryptedmessage)
encryptedmessage[i]<=90 && encryptedmessage[i]>=65
True
False
encryptedmessage[i]=encryptedmessage[i]+32
flag==1
True
False
decryptionprocess(sizeencrytedmess,encryptedmessage)
printf("INVALID ENCRYPTED MESSAGE\n")
i=i+1
int i,f=0
n>=9 && n<=15
True
False
i = 0
i < n
True
False
m[i]<=122 && m[i]>=97
True
False
f=f+1
f==n
True
False
f=1
f=0
i=i+1
f


void decryptionprocess(int sizeencrytedmess,char encryptedmessage[sizeencrytedmess])
int keyword[3],key,tempkey,n=999,sizekeywordform
int i,f
char wordformkeyword
i = 0
i <= n
True
tempkey=n-i
f = 0
f < 3
False
True
keyword[f]=tempkey%10
tempkey=tempkey/10
f=f+1
strcpy(wordformkeyword,numerationwords(keyword,wordformkeyword))
sizekeywordform=strlen(wordformkeyword)
sizeencrytedmess==sizekeywordform
True
decryptionprint(sizeencrytedmess,encryptedmessage,wordformkeyword)
False
i=i+1
char numerationwords(int k[3],char w[15])
int i
strcpy(w,"")
i = 2
i >= 0
True
k[i]
0
1
2
3
4
5
6
7
8
9
strcat(w,"ZERO")
strcat(w,"ONE")
strcat(w,"TWO")
strcat(w,"THREE")
strcat(w,"FOUR")
strcat(w,"FIVE")
strcat(w,"SIX")
strcat(w,"SEVEN")
strcat(w,"EIGHT")
strcat(w,"NINE")
i=i-1


void decryptionprint(int size,char message[size],char key[size])
int i,m,k,error
char decryption
size
i = 0
i < size
False
True
i = 0
m=message
m=m-96
k=key
k=k-64
m=m-k
m<=0
True
False
m=26+m
decryption[i]=m+96
i=i+1
i < size
True
False
printf("%c",decryption[i])
printf("->")
i=i+1
i = 0
i < size
True
False
printf("%c",key[i])
printf("\n")
i=i+1

Código em *ANSI C*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <ctype.h>

int verification(int n,char m[n])
{
  int i,f=0;
  if (n>=9 && n<=15) {
    for (i = 0; i < n; i=i+1) {
      if (m[i]<=90 && m[i]>=65) {
        f=f+1;}}
    if (f==n) {
      f=1;
    }else{
      f=0;}
  }else{
    f=0;}
  return f;
}

char *numerationwords(int k[3],char w[15])
{
  int i;
  strcpy(w,"");
  for (i = 2; i >= 0; i=i-1) {
    switch (k[i]) {
      case 0:strcat(w,"zero");break;
      case 1:strcat(w,"one");break;
      case 2:strcat(w,"two");break;
      case 3:strcat(w,"three");break;
      case 4:strcat(w,"four");break;
      case 5:strcat(w,"five");break;
      case 6:strcat(w,"six");break;
      case 7:strcat(w,"seven");break;
      case 8:strcat(w,"eight");break;
      case 9:strcat(w,"nine");break;
    }
  }
  return w;
}

void decryptionprint(int size,char message[size],char key[size])
{
  int i,m,k;
  char decryption[size];
  for (i = 0; i < size; i=i+1) {
      m=message[i]-64;
      k=key[i]-96;
      m=m-k;
      if (m<=0) {
        m=26+m;
      }
```

```
        decryption[i]=m+64;
    }
    for (i = 0; i < size; i=i+1) {
      printf("%c",decryption[i]);}
    printf(" -> ");
    for (i = 0; i < size; i=i+1) {
      printf("%c",key[i]);}
    printf("\n");
  }

  void decryptionprocess(int sizeencrytedmess,char
encryptedmessage[sizeencrytedmess])
  {
    int keyword[3],tempkey,n=999,sizekeywordform;
    int i,f;
    char wordformkeyword[15];
    for (i = 0; i <= n; i=i+1) {
      tempkey=n-i;
      for (f = 0; f < 3; f=f+1) {
        keyword[f]=tempkey%10;
        tempkey=tempkey/10;
      }

strcpy(wordformkeyword,numerationwords(keyword,wordformkeyword));
      sizekeywordform=strlen(wordformkeyword);
      if (sizeencrytedmess==sizekeywordform) {

decryptionprint(sizeencrytedmess,encryptedmessage,wordformkeyword);
      }
    }
  }

  int main() {
    char encryptedmessage[20];
    int sizeencrytedmess,flag,i;

    fgets(encryptedmessage,20,stdin);
    encryptedmessage[strlen(encryptedmessage)-1]='\0';
    sizeencrytedmess=strlen(encryptedmessage);

    for (i = 0; i < sizeencrytedmess; i=i+1) {
      if (encryptedmessage[i]<=122 && encryptedmessage[i]>=97 ) {
        encryptedmessage[i]=encryptedmessage[i]-32;
      }
    }

    flag=verification(sizeencrytedmess,encryptedmessage);

    if (flag==1) {
      decryptionprocess(sizeencrytedmess,encryptedmessage);
    }else{
      printf("INVALID ENCRYPTED MESSAGE\n");
    }
    return 0;
  }
```

Discussão critica da solução

Quanto ás vantagens no meu código, tenho uma função que de verificação [‘`verification()`’] , que verifica se os carateres da mensagem encriptada são todos validos, ou seja, imprime uma mensagem de uma “mensagem invalida” se forem encontrados, carateres diferentes de letras.

```
int verification(int n,char m[n])
{
  int i,f=0;
  if (n>=9 && n<=15) {
    for (i = 0; i < n; i=i+1) {
      if (m[i]<=122 && m[i]>=97) {
        f=f+1;}}
    if (f==n) {
      f=1;
    }else{
      f=0;}
  }else{
    f=0;}
  return f;
}
```

Quanto ás desvantagens, o meu código não esta o mais eficaz possível.

Conclusão

Para concluir, o problema foi resolvido com sucesso, o programa através da mensagem encriptada e da possível desencriptação, descobre a chave de encriptação.

O código em linguagem C, não foi difícil, apenas tive que aplicar o algoritmo da resolução do problema em C.