UC: 21045-Estruturas de Dados e Algoritmos Avançados
e-fólio A 2016-17 (cifpr)
Aluno: 1002599 - Octavio Augusto da Silva Oliveira

Relatório

Estrutura do programa

O programa foi desenvolvido com base numa *string* de 8 elementos. Os bits de um caractere (*char)* são representados nessa *string* de 8 elementos tendo em conta o seu valor na codificação ASCII. A relação entre o valor ASCII e os bits foi determinada em decimal do seguinte modo:

$$b_7 \times 2^7 + b_6 \times 2^6 + \cdots + b_0 \times 2^0 = Dec$$

Sendo *Dec* igual ao valor decimal e b_n o bit de ordem *n* (0 a 7).

Para obter uma cifra, fazem-se operações na ordem e valor dos bits, nas caixas P e S, de modo que a *string* final tenha uma ordem/valor de bits, que depende da chave k dada de 8 bits usada nas caixas **P** e k+8 nas caixas **S**. Os bits da nova *string* correspondem a um novo valor decimal, i.e., um novo *char*.

As operações efetuadas nas duas caixas P e S respeitam as regras definidas para cada uma. Inicialmente a ordem dos bits é trocada na caixa P, depois os 8 bits são divididos em 2 grupos de 4 bits, sendo os seus valores alterados nas caixas S. No final existe outra caixa P que troca novamente a ordem dos bits conforme a chave k. Convertem-se os bits da *string* num valor decimal que corresponde ao valor do novo *char* no código ASCII. Para obter a decifra fazem-se operações inversas às anteriormente descritas.

O programa foi dividido em 4 partes:

- Funções auxiliares que são usadas nas funções principais de encriptação e desencriptação;
- Funções de encriptação;
- Funções de desencriptação;
- Funções utilizadas pela função principal (*main*)

O programa foi escrito nesta ordem para facilitar a sua organização.

As imagens seguintes representam uma sequência de operações efetuadas num dispositivo PSP, utilizando a chave k=36071245. Neste exemplo, o *char* inicial é ‘1’ (decimal 49) e o final é ‘L’ (decimal 76).

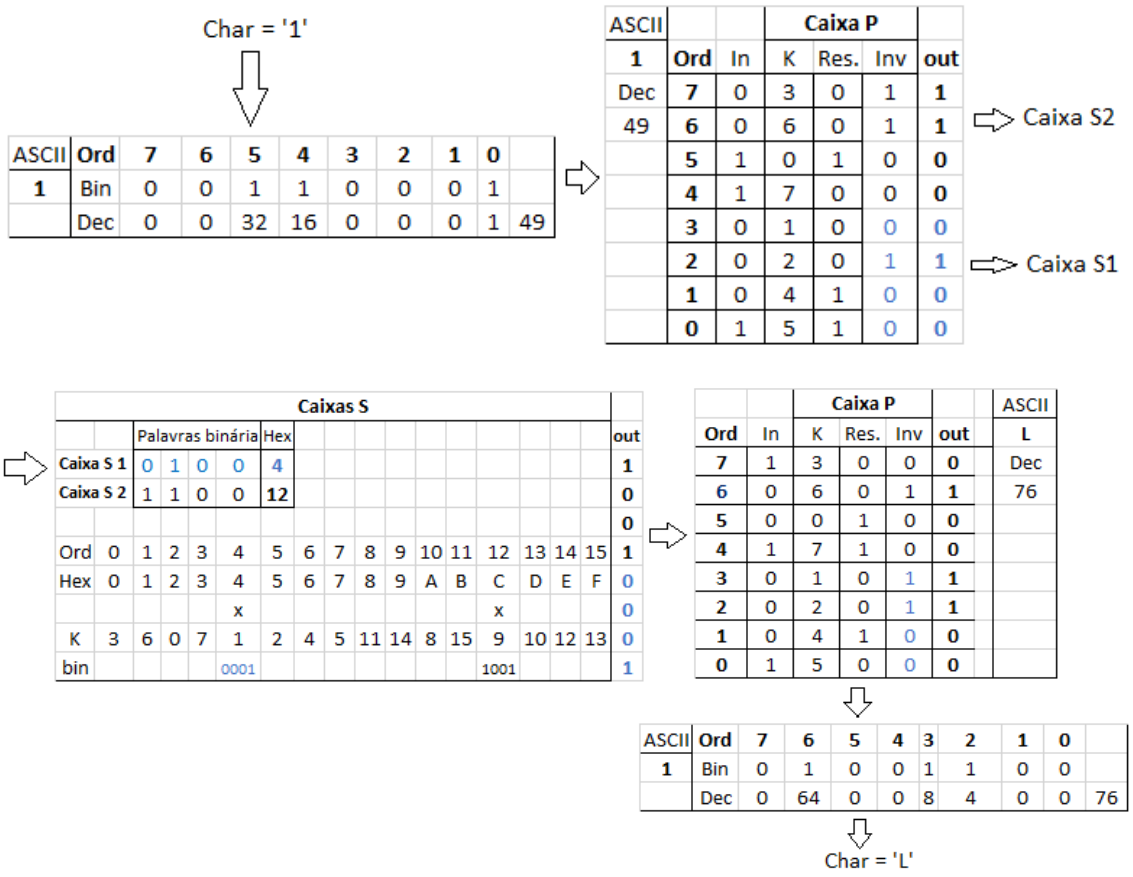

Figura 1- Sequência de operações efetuadas num dispositivo PSP

A encriptação segue o fluxograma seguinte. A desencriptação também segue este fluxograma, mas executa operações inversas à encriptação.

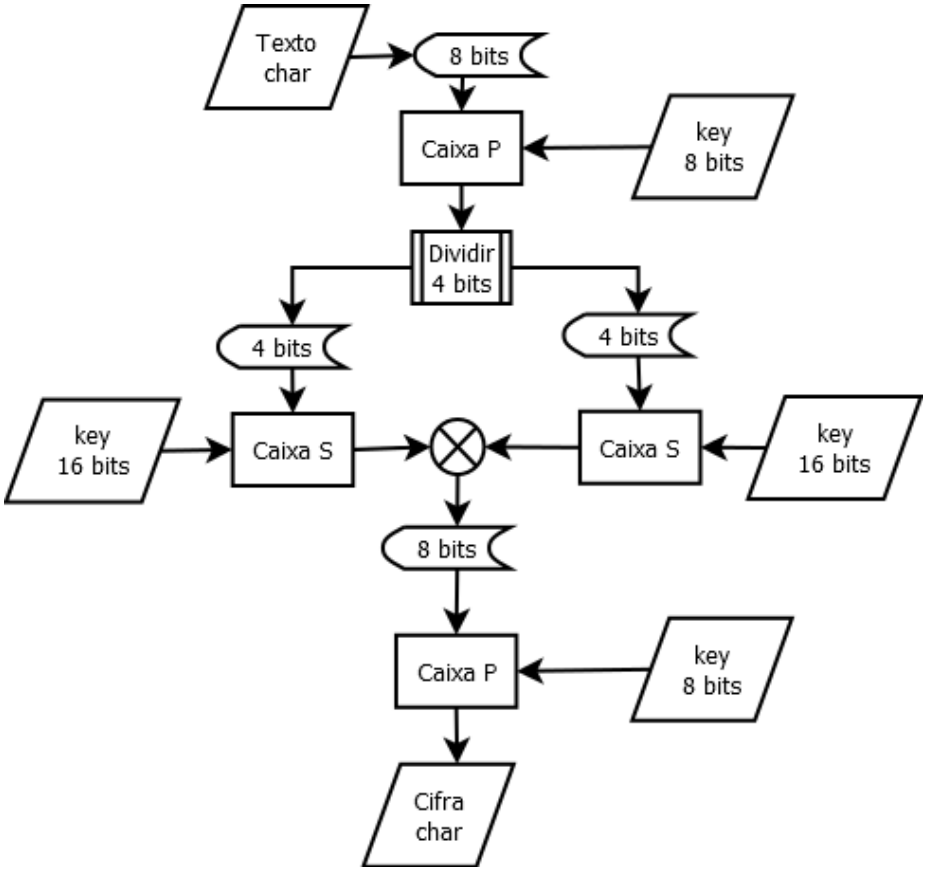

Figura 2- Fluxograma de encriptação

Leitura do ficheiro de texto

Inicialmente optou-se por ler e converter um caracter de cada vez, utilizando a função da imagem seguinte.

```
void encrypt_decrypt(char type, string key)
{
    string texto;
    char buf[1];
    while(read(0, buf, sizeof(buf))>0)
    {
        texto = buf;
        //Encriptação
        if(type == 'c')
            cout << cifraPSP(key, texto, false);
        //Desencriptação
        else if(type == 'd')
            cout << decifraPSP(key, texto);
    }
}
```

Figura 3 - Função para inserir dados (alterada no programa)

Mas esta opção no Windows tinha problemas porque este SO não tem por defeito a entrada/saída (I/O) padrão em modo binário e por isso quando se carregou um ficheiro de texto através do redirecionamento da entrada/saída padrão a nível da linha de comandos do sistema

operativo, verificou-se que os *char* com valor decimal em código ASCII igual a 26 - ***SUB*** *(substitute)* e 30 - ***RS*** *(record separator)* geravam -1 na *stream* de entrada ***cin***, o que provocava a interrupção da leitura dos restantes *char.*

Nestas condições, para utilizar o Windows, teríamos de evitar os *char* 26 e 30 no ficheiro de texto e no ficheiro da cifra. A primeira condição seria não utilizar os *char* 26 e 30 no ficheiro de texto, a segunda seria impedir a conversão destes caracteres na encriptação. Para a ultima condição foi criada a função conforme se mostra na imagem seguinte. Esta não fazia a encriptação dos *char* que geravam os *char* 26 ou 30.

```
char cifraPSP(string key, string in, bool total)
{
    //Converte o char na string correspondente ao binário
    string binary = StringToBinary(in);
    //Obtem a saída da 1° caixa P
    string out_boxP = boxP(key, binary);
    //Obtem a saída das caixas S
    string out_boxS = boxS(key, out_boxP.substr(0, 4)) +
                      boxS(key, out_boxP.substr(4, 4));
    //Obtem a saída da 2° caixa P
    out_boxP = boxP(key, out_boxS);
    //Converte o "binário" em  char
    int cif = BinaryToChar(out_boxP);
    //Quando se usa o redirecionamento da entrada/saída padrão da linha de
    //comandos, os códigos ASCII, decimal 26 e 30 geram -1 na stream de
    //entrada cin, o que impede a leitura de novos char. Para evitar este
    //problema optou-se por não codificar estes char. Assim deste modo é
    //possivel ler todos os char na desencriptação.
    //Qando se faz a desencriptação é presciso saber quais foram os char que não
    //foram encriptados, que são aqueles, quando são cifrados, são iguais a 30
    //ou 26, para deteta-los o programa usa esta função cifrando todos os char
    //com a flag "total = true).
    if ((cif == 26 || cif == 30) && !total)
        return in[0];
    else
        return cif;
}
```

Figura 4- Função de encriptação (alterada no programa)

Depois, na desencriptação o programa teria de detetar e manter os caracteres que não tinham sido encriptados, para isso foi escrita a função seguinte.

```
char decifraPSP(string key, string in)
{
    //Os char ASCII decimal 26 e 30 da cifra, não foram encriptados, por isso
    //não devem ser alterados. O "if" seguinte serve para manter esses char.
    int x_not_cod = (int)cifraPSP(key, in, true);
    if (x_not_cod == 26 || x_not_cod == 30)
        return in[0];
    //String corespondente ao binário ASCII do char dado
    string binary = StringToBinary(in);
    //Obtem a entrada da 2° caixa P
    string in_boxP = invertBoxP(key, binary);
    //Calcula o valor "hexadecimal" (entre 0 a 15) dos 2 conjuntos obtidos
    //em função da saída da caixa S, que é igual à entrada da 2° caixa P.
    int x1 = get_decimal(in_boxP.substr(0, 4));
    int x2 = get_decimal(in_boxP.substr(4, 4));
    //Junta os 2 conjuntos de 4 bits da entrada da caixa S para obter a entrda
    //da caixa S
    string inBoxS = get_binOutBoxS(key, x1) + get_binOutBoxS(key, x2);
    //Obtem a entrda da 1° caixa P à custa da sua saída
    in_boxP = invertBoxP(key, inBoxS);
    //Converte a string correspondente ao binário num char
    return BinaryToChar(in_boxP);
}
```

Figura 5- Função desencriptação (alterada no programa)

Mas esta solução não era a melhor porque não encriptava todos os caracteres e demorava o dobro do tempo na desencriptação porque precisava encriptar novamente todos os caracteres para detetar os que não tinham sido encriptados. Era uma solução pouco eficiente e por isso foi abandonada.

Os *char* do ficheiro de texto tinham de entrar como binários e para isso tinha-se que usar a função: *_setmode(_fileno(stdin), _O_BINARY)* que colocava a entrada como se pretendia, usando a constante *_O_BINARY*. Esta era a melhor solução, lia 1 byte (caractere) de cada vez sem qualquer interrupção, depois fazia a encriptação/desencriptação também um caractere de cada vez.

As funções acima apresentadas foram alteradas de modo a implementar a melhor solução. A primeira sofreu uma ligeira alteração e as outras foram simplificadas, principalmente a última, conforme se pode ver no código do programa *cifpr.cpp*.

Validação da chave

A utilização da chave ordenada (k=01234567) não faz qualquer alteração na encriptação e por isso não foi considerada válida. As chaves com permutações de um bit, por exemplo: k=10234567, foram consideradas válidas, apesar de fazerem poucas alterações no texto.

Compilação

No Windows foi usado gcc version 4.4.1 (TDM-2 mingw32) no Linux gcc version 4.9.2 (GCC)

Comandos usados: Windows: *g++ -o cifpr.exe cifpr.cpp*; Linux: *g++ -o cifpr cifpr.cpp*

Durante a compilação os compiladores não reportaram qualquer erro.

Código do programa

```
/*
** UC: 21045-Estruturas de Dados e Algoritmos Avançados
** e-fólio A 2016-17 (cifpr)
**
** Aluno: 1002599 - Octávio Oliveira
*/

#ifdef __cplusplus
#include <cstdio>
#else
#include <stdio.h>
#endif
#include <iostream>
#include <cstdlib>//Para usar as funções atoi() e strtol()
#include <cmath>//Para usar a função pow()
#include <unistd.h>//Para detetar redirecionamento (STDIN_FILENO)

using namespace std;

#if defined(WIN32) || defined(_WIN32) || defined(_WIN32_) || defined(__WIN32) ||
defined(__WIN32__) || defined(__TOS_WIN__) || defined(__WINDOWS__)
#include <fcntl.h>
//Em Windows usa a função seguinte para colocar a entrada/saída padrão em
//modo binário (_O_BINARY), de modo a evitar bloqueio dos caracteres com código
//ASCII igual a 26 - SUB (substitute) e 30 - RS (record separator)
void set_binary_io()
{
        _setmode( _fileno( stdin  ), _O_BINARY );
        _setmode( _fileno( stdout ), _O_BINARY );
        _setmode( _fileno( stderr ), _O_BINARY );
```

```
#ifdef __cplusplus
	cin.sync_with_stdio();
#endif
}
#else
void set_binary_io(){}
#endif
//Informação obtida em msdn.microsoft.com

//Neste programa os bits de um char são representados numa string de 8 elementos
//que é usada para efetuar conversões de caracteres (encriptação/desencriptação)
//tendo em conta valor decimal em código ASCII desse char.
//A conversão binário/decimal utiliza os elementos da string b7, b6,...,b0
//b7 x 2^7 + b6 x 2^6 + ... + b0 x 2^0 = valor decimal
//O programa faz operações na ordem e valor dos bits respectivamente nas
//caixas P e S, de modo que a string final tenha uma ordem/valor de bits que
//depende da chave dada. A string resultante corresponde a um novo valor decimal
//ou seja, a um novo char.
//++++++++++++++++++++++++++++++ Funções auxiliares ++++++++++++++++++++++++++++++
//Inverte a ordem dos bits.
string invertOrder(string str)
{
	string result = "";
	for (int i = 7; i > -1; i--)
		result += str[i];
	return result;
}
//Converte uma string num char
char BinaryToChar(string str)
{
	char buffer[9];
```

```
	size_t length = str.copy(buffer, 8, 0);
	buffer[length] = '\0';
	return strtol(buffer, 0, 2);
}
//Devolve uma string que coresponde a um char do binário na ordem 7 6 ...0.
string printBinaryOrder(char c)
{
	string result = "";
	for (int i = 7; i >= 0; --i)
	{
		if(c & (1 << i))
			result += '1';
		else result += '0';
	}
	return result;
}
//Devolve uma string que coresponde a um binário na ordem 0 1 ...7
string StringToBinary(string s)
{
	string result = "";
	for (unsigned int i = 0; i < s.size(); i += 2)
		result += printBinaryOrder(s[i]);
	return result;
}
//Obtem a chave para a caixa S
void get_key_boxS(int k[], string key)
{
	//Primeira parte é igual à chave da caixa P
	for (int i = 0; i < 8; i++)
		k[i] = atoi(key.substr(i, 1).c_str());
	//Segunda parte é igual à chave da caixa P + 8
```

```
	for (int i = 8; i < 16; i++)
		k[i] = k[i - 8] + 8;
}
//Devolve um inteiro entre 0 e 15 corespondente a 4 bits. Simula um
//hexadecimal
int get_decimal(string str)
{
	int result = 0;
	for (int i = 0; i < 4; i++)
	{
		int k = atoi(str.substr(i, 1).c_str());
		result += k * pow(2, 3 - i);
	}
	return result;
}
//Transforma um inteiro numa string que representa um binário de 4 bits.
string get_bin(int val)
{
	unsigned int mask = 1 << 3;
	string result = "";
	for(int i = 0; i < 4; i++)
	{
		if( (val & mask) == 0 )
			result += "0";
		else
			result += "1";
		mask  >>= 1;
	}
	return result;
}
//++++++++++++++++++++++++++++ FIM Funções auxiliares ++++++++++++++++++++++++++
```

```
//++++++++++++++++++++++++++++ Funções de Encriptação ++++++++++++++++++++++++++
//Executa as operações da caixa P e devolve uma string que representa um binário
//de 8 bits
string boxP(string key, string character)
{
	string result = "";
	for (int i = 0; i < 8; i++)
	{
		/*Por exemplo, o char "A" entra com a sequência de bits seguinte:
		 i =	7	6	5	4	3	2	1	0
				0	1	0	0	0	0	0	1
		 tendo em conta a chave = 36410527
		 sairá com  character[3]=0; character[6]=1; etc.
		 ou seja, 0 1 0 0 1 0 0 0. Apenas existe permutação de bits.
		*/
		int k = atoi(key.substr(i, 1).c_str());
		result += character[7 - k];
	}
	//Inverter a ordem para 7 6 ...0, o caso de "A", resulta 0 0 0 1 0 0 1 0
	return invertOrder(result);
}
//Executa as operações da caixa S
string boxS(string key, string binary)
{
	//Determina em hexadecimal o valor dos 4 bits colocados na entrada da caixa.
	//Para facilitar o calculo utilizou-se um decimal de 0 a 15, que
	//corespondente ao hexadecimal
	int k[16];
	int hex = get_decimal(binary);
	//Por exemplo o binário é: 1100 = hexadecimal C = decimal 12.
```

```
        //Carrega a chave da caixa S e procura o valor da chave na posição 12.
        get_key_boxS(k, key);
        //A função devolve esse valor, por exemplo, 9
        //A função seguinte converte o decimal numa string com o binário
        //corespondente.
        return get_bin(k[hex]);
}
//Converte um char do texto plano num char do texto cifrado.
char cifraPSP(string key, string in)
{
        //Converte o char na string correspondente ao binário
        string binary = StringToBinary(in);
        //Obtem a saída da 1º caixa P
        string out_boxP = boxP(key, binary);
        //Obtem a saída das caixas S
        string out_boxS = boxS(key, out_boxP.substr(0, 4)) +
                                        boxS(key, out_boxP.substr(4, 4));
        //Obtem a saída da 2º caixa P
        out_boxP = boxP(key, out_boxS);
        //Converte o "binário" em  char
        return BinaryToChar(out_boxP);
}
//++++++++++++++++++++++++++++ FIM Funções de Encriptação ++++++++++++++++++++

//++++++++++++++++++++++++++++ Funções de Desencriptação ++++++++++++++++++++++
//Obtem a entrada da caixa P através da sua saída (no inicio da
//desencriptação é o char cifrado, no final é a sequência de bits obtida nas
//caixas S).
string invertBoxP(string key, string binary)
{
        //Inverte a ordem dos bits
```

```
	string invert = invertOrder(binary);
	//Aplica as operações da caixa P
	char result[9];
	for (int i = 0; i < 8; i++)
	{
		int k = atoi(key.substr(i, 1).c_str());
		result[k] = invert[i];
	}
	result[8] = '\0';
	return invertOrder(result);
}
//Executa as operações inversas da caixa S. Usa a saída da caixa S em décimal
//de 0 a 15 e a chave da caixa S para fornecer os 2 conjuntos de 4 bits
//corespondentes à entrada da caixa S.
string get_binOutBoxS(string key, int x)
{
	int k[16];
	get_key_boxS(k, key);
	int i = 0, p;
	while (i < 16)
	{
		if(k[i] == x)
		{
			p = i;
			i = 16;
		}
		else i++;
	}
	return get_bin(p);
}
//Obtem o char do texto plano a partir do char do texto cifrado.
```

```
char decifraPSP(string key, string in)
{
	//String corespondente ao binário ASCII do char dado
	string binary = StringToBinary(in);
	//Obtem a entrada da 2º caixa P
	string in_boxP = invertBoxP(key, binary);
	//Calcula o valor "hexadecimal" (entre 0 a 15) dos 2 conjuntos obtidos
	//em função da saída da caixa S, que é igual à entrada da 2º caixa P.
	int x1 = get_decimal(in_boxP.substr(0, 4));
	int x2 = get_decimal(in_boxP.substr(4, 4));
	//Junta os 2 conjuntos de 4 bits da entrada da caixa S para obter a entrda
	//da caixa S
	string inBoxS = get_binOutBoxS(key, x1) + get_binOutBoxS(key, x2);
	//Obtem a entrda da 1º caixa P à custa da sua saída
	in_boxP = invertBoxP(key, inBoxS);
	//Converte a string correspondente ao binário num char
	return BinaryToChar(in_boxP);
}
//++++++++++++++++++++++++++++++ FIM Funções de Desencriptação ++++++++++++++++++

//+++++++++++++++++ Funções associadas à função principal (main) +++++++++++++++
//Função de teste de encriptação/desencriptação usada para testes e informação
//ao utilizador.
void console_encrypt_decrypt(string key)
{
	string texto = "Universidade Aberta.\nEstruturas de Dados e Algoritmos ";
	texto += "Avancados.\nAluno 1002599 - Octavio Oliveira.";
	string texto_out = "";
	//Encriptar
	for(unsigned int i = 0; i < texto.size(); i++)
		texto_out += cifraPSP(key, texto.substr(i, 1));
```

```
	cout << "Encriptar texto:\n" << texto + "\n\nCifra:\n" << texto_out << endl;
	//Desncriptar
	texto = "";
	for(unsigned int i = 0; i < texto_out.size(); i++)
		texto += decifraPSP(key, texto_out.substr(i, 1));
	cout << "\n---------------------------------------------\n" << endl;
	cout << "Desencriptar cifra:\n" << texto_out +
		"\n\ntexto:\n" << texto << endl;
	cout << "\n*********************************************\n" << endl;
	cout << "Para trabalhar com ficheiros usar:\ncifpr c 76543210 ";
	cout << "<texto.txt >texto.cif => Encriptar" << endl;
	cout << "cifpr d 76543210 <texto.cif >texto.txt => Desencriptar" << endl;
	cout << "cifpr c <texto.txt | cifpr d >texto2.txt => Encriptar";
	cout << " e Desencriptar" << endl;
}
//Executa a encriptação ou a desencriptação
void encrypt_decrypt(char type, string key)
{
	char ch;
	//Coloca a entrada/saída padrão em modo binário, lê 1 byte de cada vez.
	set_binary_io();
	while(read(STDIN_FILENO, &ch, 1) > 0)
	{
		char caracter[1];
		sprintf(caracter, "%c", ch);
		//Encriptação
		if(type == 'c')
			cout << cifraPSP(key, caracter);
		//Desencriptação
		else if(type == 'd')
			cout << decifraPSP(key, caracter);
```

```
        }
}
//Se a chave não tiver 8 algarismos diferentes de 0 a 7 devolve false.
//Também sai false se a chave estiver ordenada em modo crescente (uma chave
//deste tipo não faz encriptação.
bool check_key(string key)
{
        if(key.length() != 8)
                return false;
        int i = -1;
        bool bit[8], up_order = true;
        for(int i = 0; i < 8; i++)
                bit[i] = false;
        while(i++ < 7)
        {
                //Só aceita ordem de bits de 0 a 7
                if(key[i] < 48 || key[i] > 56)
                {
                        i = 8;
                        return false;
                }
                //Verifica se a ordem do bit é repeitada e para isso utiliza o vetor
                //bool bit[]. Coloca bit[0]=true...bit[7]=true para todos, se ocorrer
                //bit[i]=true, quando bit[i] já é true,significa que existe um algarismo
                //repetido e por isso a chave não é respeitada. Neste caso sai false.
                else
                {
                        if (bit[key[i] - 48])
                                return false;
                        else bit[key[i] - 48] = true;
                }
```

```
		//Verifica se a chave tem a ordem 01234567. Inicialmente considera que
		//tem, se ocorrer um caso que não seja crescente coloca up_order=false,
		//termina a verificação e considera a chave válida.
		if(i < 7 && up_order)
		{
			if(key[i] > key[i + 1])
				up_order = false;
		}
	}
	if (up_order)
		return false;
	else
		return true;
}
//+++++++++++++++++ FIM Funções associadas à função principal (main) ++++++++++

int main(int argc, char* argv[])
{
	//Chave por defeito.
	string key = "36410527";
	//Se o programa não tem argumento, se o argumento é superior a 3
	//ou o primeiro argumento não é 'c' ou 'd', apresenta os dados da função
	//de teste: console_encrypt_decrypt() e informa acerca da utilização
	//do programa. Esta informação pode ser enviada para ficheiro ou para
	//a consola, conforme tenha ou não redirecionamento.
	if(argc < 2 || argc > 3)
	{
		console_encrypt_decrypt(key);
		return 0;
	}
	//Considera apenas o primeiro char
```

```
if(argv[1][0] != 'c' && argv[1][0] != 'd')
{
	console_encrypt_decrypt(key);
	return 0;
}
//Se tem 2 argmentos verifica se key é válida
if(argc > 2)
{
	if (check_key(argv[2]))
		key = argv[2];
	else
	{
		//Se a chave não é válida informa o utilizador
		cout << "A chave fornecida (" << argv[2] << ") nao e valida!" << endl;
		cout << "Deve ter 7 algarismos diferentes entre 0 e 7." << endl;
		return 0;
	}
}
//Apenas considera o primeiro e segundo argumentos, ignora os restantes.
//Nesta fase já validou o primeiro e segundo argumentos. Agora verifica se
//existe o ficheiro de entrada no redirecionamento.
if (isatty(STDIN_FILENO))
{
	//Neste caso não existe, mas pode existir redirecionamento para o
	//ficheiro de saída e existindo a informação do cout (teste e exceções,
	//porque não tem nada para cifrar) segue para o ficheiro de saída.
	//Se não existir o ficheiro de saída, o programa envia os dados do
	//cout para a consola.
	cout << "Nao foi indicado o ficheiro de entrada." << endl;
}
else
```

```
        {
                //Neste caso existe o ficheiro de entrada no redirecionamento. Processa
                //normalmente a encriptação ou desencriptação se os casos excepcionais
                //acima tratados não ocorrerem, se ocorrer algum, não processa a
                //a encriptação ou desencriptação e envia a informação das exceções
                //dtetadas para ficheiro de saída.
                encrypt_decrypt(argv[1][0], key);
        }

        return 0;
}
//--------------------------- FIM do programa ---------------------------------
```