Universidade do Minho

Relatório Final

Métodos de Programação I

Mestrado Integrado em Engenharia de Telecomunicações e Informática

Descodificação de uma Sequência Morse

Nome: Leandro Alves

Nº de Aluno: A82157

E-mail: leandrinos54@hotmail.com

Universidade do Minho

Índice

Introdução

Este projeto tem como objetivo solucionar um problema em que é preciso determinar o número de frases possíveis/válidas dando apenas como entrada a sequência de código morse e um dicionário de palavras, sendo ambos introduzidos pelo utilizador do programa.

Para que a resolução deste problema fosse possível, foi essencial compreender o que cada passo do programa teria de fazer recorrendo a um algoritmo bem construído.

Visto que o enunciado do problema não foi explicito à forma como o nosso programa deveria responder no caso de haver a situação de introduzir dados inválidos, isto é, de haver código morse misturado com outros caracteres, palavras em minúscula ou com dígitos/símbolos no dicionário, palavras repetidas, etc.) não foram programadas quaisquer restrições no caso de haver este tipo de dados. No entanto, fica dependente do utilizador o funcionamento correto do programa.

Universidade do Minho

Algoritmo não refinado

1- Ler sequência de código morse.

2- Ler número(n) de palavras que quer no dicionário.

3- Ler as n palavras que o dicionário irá ter.

4- Verificar quais das palavras podem dar início à sequência morse.

5- Percorrer outra palavra e verificar se esta pode ser a seguir à(s) palavra(s) já selecionada(s).

6- Se as palavras já selecionadas completarem toda a sequência morse,ou seja, todos os “.” e “-”, contar uma frase válida. Voltar ao ponto 5 se não tiver percorrido todas as palavras do dicionário.

Algoritmo refinado

1. [programa principal]

    1.1. Ler sequência de código morse (morse_code)

    1.2. Ler número de palavras que quer no dicionário(n_words)

        1.2.1. Ler palavras do dicionário

            1.2.1.1. Para i←0 até n_words fazer

                Ler dicionário(dict[i])

    1.4. Inicializar a variável que representa o número de frases válidas

        1.4.1. s_valids←0

    1.5. Chamar a função que irá comparar o dicionário introduzido com a sequência de código morse

        1.5.1. *compare (n_words, dict, morse_code)*

        1.5.2. Colocar o resultado da função *compare* na variável que representas o número de frases válidas

            1.5.2.1. s_valids ← *compare (n_words, dict, morse_code)*

    1.6. Escrever o número de frases válidas(s_valids)

2. [Comparar as palavras do dicionário com a sequência de código morse (função *compare*)]

    2.1. Inicializar a variável que irá conter o número de frases válidas

        2.1.1. valid ← 0

2.2. Percorrer cada palavra introduzida no dicionário

2.2.1. Para i ← 0 até n fazer

convert(dic[i], w_conv)

Se (strncmp(mrs_code, w_conv, strlen(w_conv)) = 0) então

Se (strlen(mrs_code) = strlen(w_conv)) então

valid ← valid +1

Senão

valid = valid + *compare(n,dic,&mrs_code[strlen(w_conv)])*

Terminar Se

Terminar Se

Terminar Para

2.3. Devolver o número de frases válidas

2.3.1. Devolver valid

3. Converter cada palavra do dicionário para codigo morse (função *convert*)

3.1. Copiar para a variável que irá conter a palavra convertida a código morse

3.1.1. strcpy(aux_w, "")

3.2. Percorrer todas as letras de cada palavra introduzida no dicionário

3.2.1. Para i←0 até strlen(word) fazer

Caso word[i] seja

'A' fazer: strcat(aux_w, ".-")

'B' fazer: strcat(aux_w, "-...")

'C' fazer: strcat(aux_w, "-.-.")

'D' fazer: strcat(aux_w,"-..")

'E' fazer: strcat(aux_w,".")

'F' fazer: strcat(aux_w,"..-.")

'G' fazer: strcat(aux_w,"--.")

'H' fazer:	strcat(aux_w,"....")
'I' fazer:	strcat(aux_w,"..")
'J' fazer:	strcat(aux_w,".—")
'K' fazer:	strcat(aux_w,"-.-")
'L' fazer:	strcat(aux_w,".-..")
'M' fazer:	strcat(aux_w,"–")
'N' fazer:	strcat(aux_w,"-.")
'O' fazer:	strcat(aux_w,"—")
'P' fazer:	strcat(aux_w,".–.")
'Q' fazer:	strcat(aux_w,"–.-")
'R' fazer:	strcat(aux_w,".-.")
'S' fazer:	strcat(aux_w,"...")
'T' fazer:	strcat(aux_w,"-")
'U' fazer:	strcat(aux_w,"..-")
'V' fazer:	strcat(aux_w,"...-")
'X' fazer:	strcat(aux_w,"-..-")
'Y' fazer:	strcat(aux_w,"-.–")
'W' fazer:	strcat(aux_w,".–")
'Z' fazer:	strcat(aux_w,"–..")
Outra coisa:	Devolver 0

Terminar Para

Descrição do algoritmo

O exemplo que se segue, serve para poder descrever o algoritmo detalhamente, para se ficar a ententer como o programa em C funciona.

Assumindo que introduzimos uma sequência de código morse correspondente à palavra “CAIXA”. Tendo, assim, uma string que contém aos seguintes pontos e traços:

C | A | I | X | A

-.-. | .- | .. | ..-| .-

0 1 2 3 4 5 6 7 8 9 10 11 12

Agora iremos introduzir 5 palavras no dicionário. Sendo este uma matriz,cada palavra ficará em cada linha e em cada coluna ficará cada letra.

Como podemos repetir sempre a mesma palavra, as combinações possíveis, ou seja todas as combinações mesmo aquelas que não fazem sentido, com este dicionário é 5^5= 3 125.

I \ J	0	1	2	3	5
0	C	A	I		
1	C	O	P	O	
2	A	I			
3	M	E	S	A	
4	X	A			

Fazendo isto, acabamos a estapa 1.3.1.1 do algoritmo refinado.

Depois disto, segue-se a chamada da função *compare*, que tem como parâmetros o número das palavras do dicionário (n_words), dicionário introduzido (dict)e a sequência de código morse(morse_code).

Nesta primeira chamada, a função será feira da seguinte forma:

compare(5, dicionário introduzino, sequência de código morse)

Nesta função, será declarado:

- Uma nova string ,chamada w_conv, que irá conter uma das 7 palavras do dicionário convertida a morse.
- Uma variável do tipo inteiro(valid) que irá contar o número de frases válidas formadas pelo dicionário.
-

Na estapa 2.2.1 do algoritmo refinado, o ciclo “para” começará com a primeira palavra do dicionário, ou seja “CAI”.

De seguida, o programa chama a função *convert* que irá receber a string ”CAI” e convertê-la-á para código morseda seguinte forma:

1º- O programa “limpa” a variável aux_w pois esta é que irá conter a palavra “CAI” convertida a morse.

2º- O programa entra nos ciclos e ao encontrar o carater “C”(o primeiro caratér da palavra ”CAI”) copia para a variável aux_w o eqivalente em código morse “-.-.”.

3º- Como não chegamos ao final da palavra, o programa volta a entrar nos ciclos mas desta vez irá procurar o caratér “A” e concatenar o equivalente a morse “.-” na variavel aux_w.

4º- Repetição do passo 3 mas com o caratér ”I”.

5º- Como a palavra “CAI” chegou ao fim, o ciclo “para” acaba.

Temos a palavra “CAI” convertida a morse “-.-..-..” na variável aux_w.

Agora voltamos à função *compare* e continuamos onde tinhamos parado pois agora já podemos comparar a palavra “CAI” com o código morse.

Verificamos se o código morse e o morse resultante da palavra “CAI” são iguais até à posiçao 7,exclusive, visto que a palavra “CAI” convertida só tem 7 carateres. Nesta verificação, acabamos por concluir que os dois códigos morse são iguais, portanto “CAI” é uma palavra válida.

Sendo assim, avançamos para a próxima instrução do código.

Aqui, verificamos que o tamanho do morse da palavra “CAI” não é igual ao tamanho da sequência morse, logo o programa avança para o senão.

Nessa parte, o programa chama a primeira recursividade da funçao *compare* com os seguintes parâmetros:

compare(5,dicionário introduzido, codigo morse a partir da 7ª posição)

Na primeira recursividade da funçao compare, o programa começa a verificar palavra a palavra e a ver quais delas podem completar ou não(se não volta a chamar uma nova recursividade da função compare) esta nova parte do código morse.

Sabemos que “CAI” e “COPO” não podem ser pois um ‘C’ é “-.-.” ≠ de “..-.-” (nova parte do código morse após termos descoberto que “CAI” era uma palavra válida).

Sabemos também que “AI” e “MESA” não podem ser uma vez que ‘A’ é “.-” e “M’ é “--” , ou seja, são diferentes de “..-.-”.

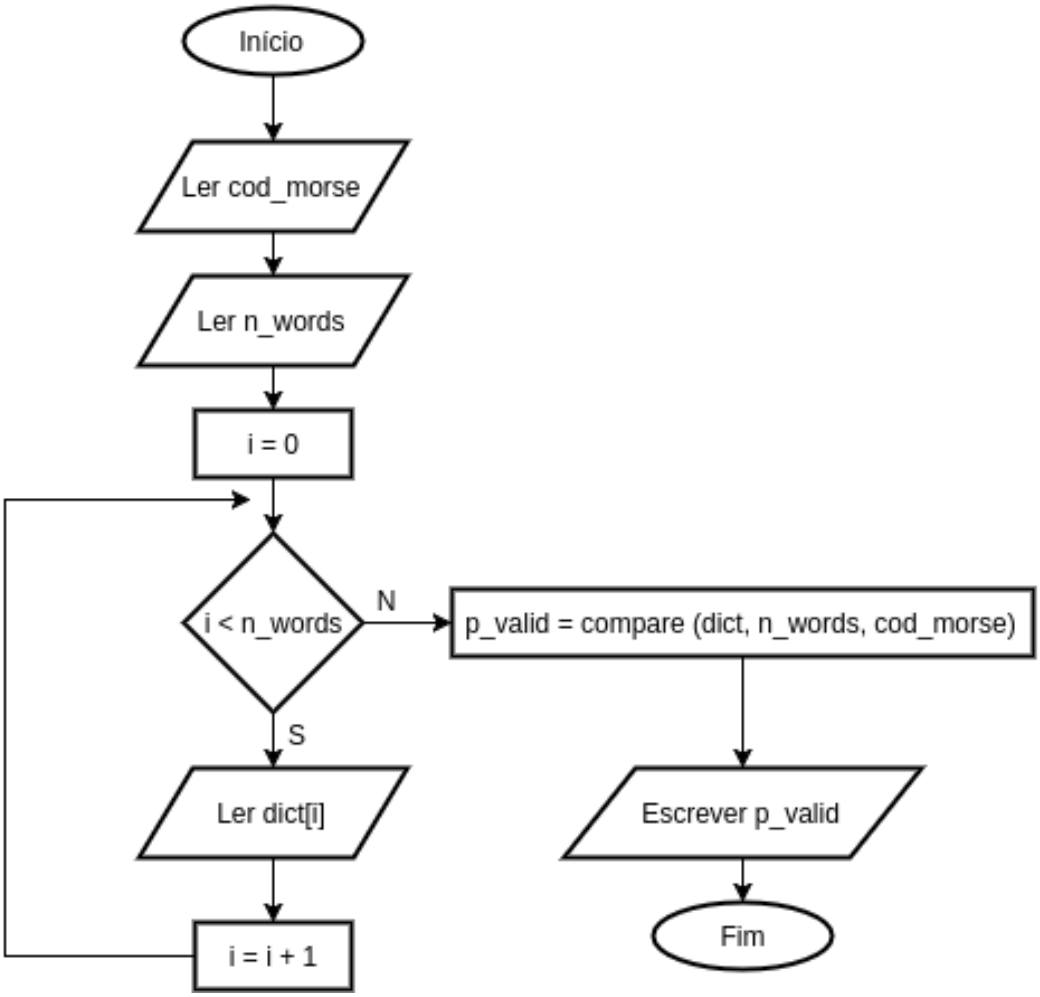
Concluimos que a única palavra que completa a nova sequência morse é “XA”, portanto o programa devolve valid = 1 para a função compare.

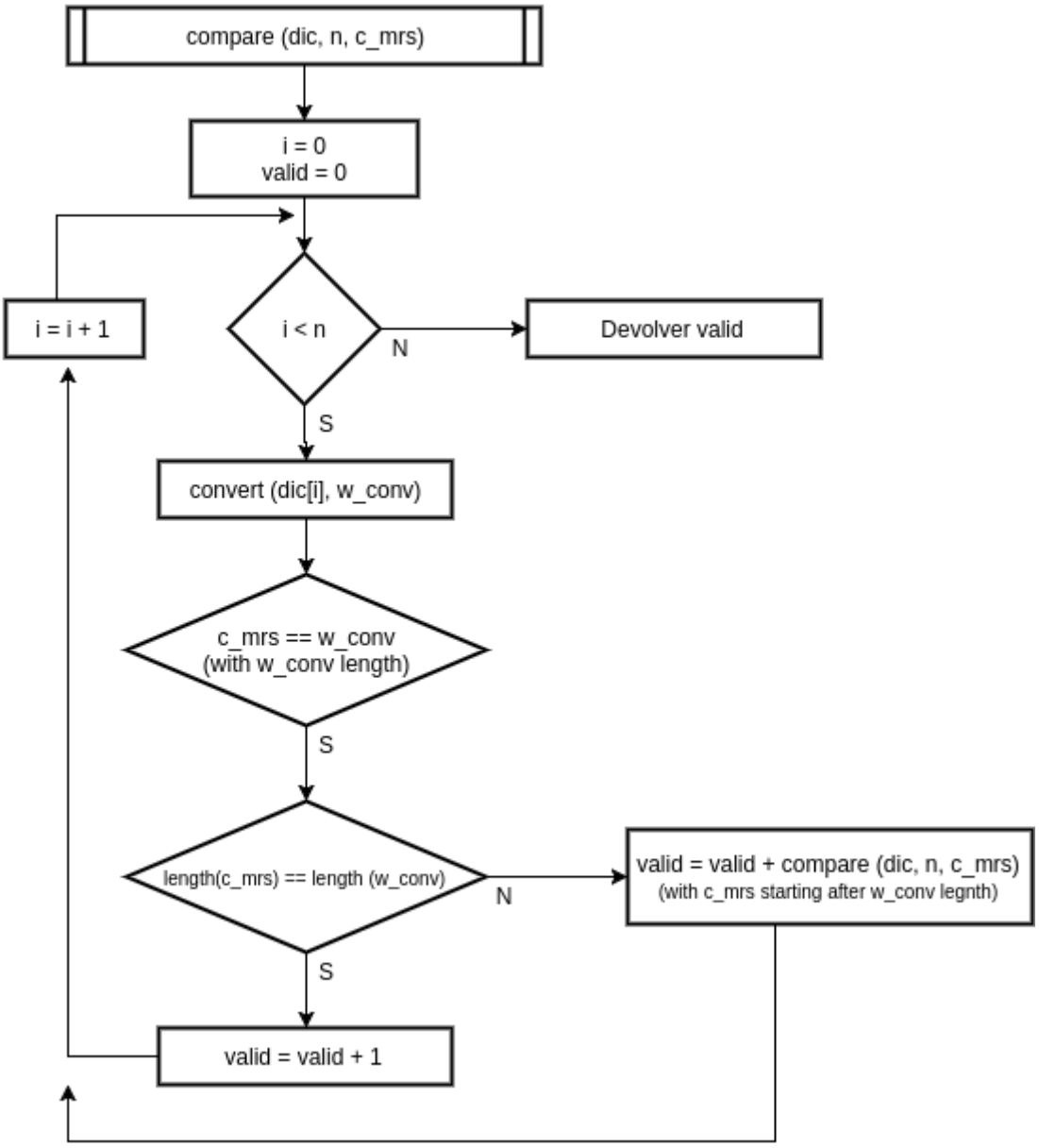
Entretando, na função compare foi-se verificando se o resto das palavras do dicionário poderiam iniciar a sequência morse inicial. Como mais nenhuma podia iniciar a sequência morse, valid ← 0(pois valid era inicialmente 0) + 1 (este valor é aquele que a primeira recursividade devolveu).

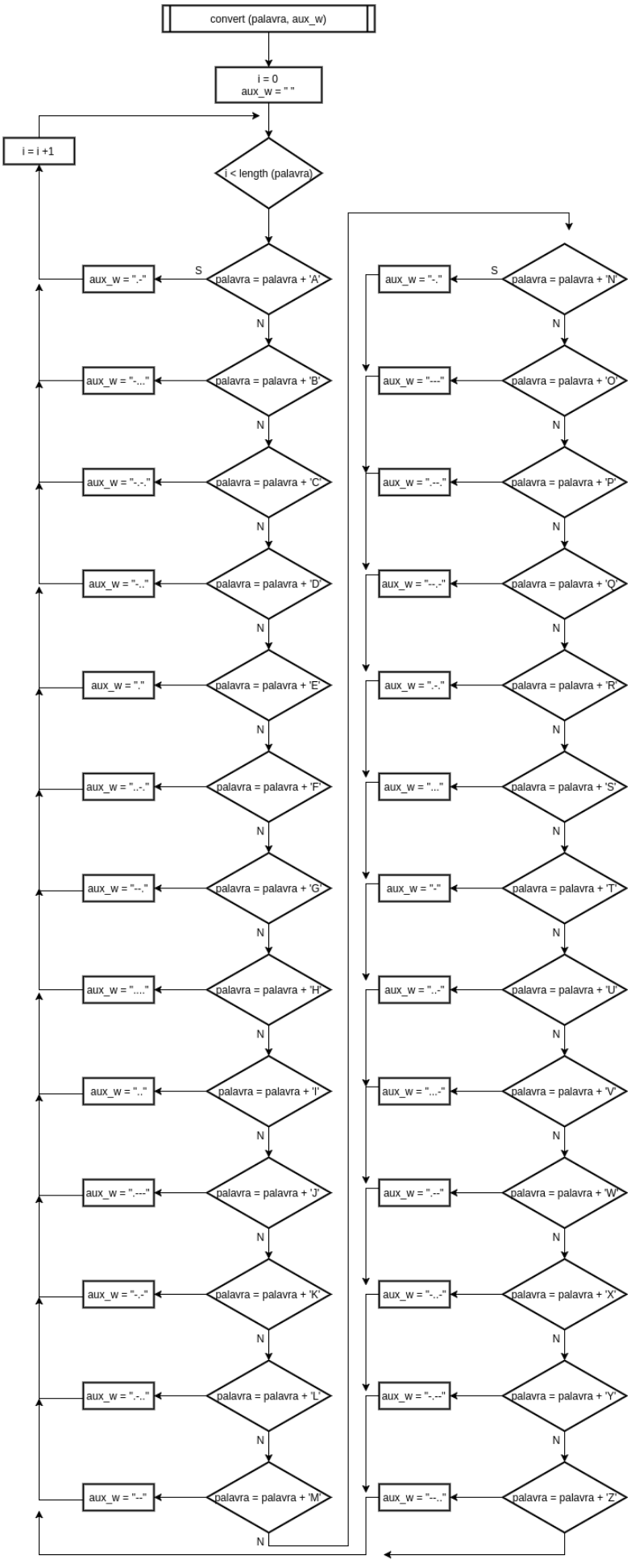
Conclui-se que o número de frases válidas é 1, ou seja, “CAI” + “XA” forma uma frase (palavra,neste caso) válida.

Universidade do Minho

Fluxograma(s)




compare (dic, n, c_mrs)
i = 0
valid = 0
i < n
N
Devolver valid
S
i = i + 1
convert (dic[i], w_conv)
c_mrs == w_conv
(with w_conv length)
S
length(c_mrs) == length (w_conv)
N
valid = valid + compare (dic, n, c_mrs)
(with c_mrs starting after w_conv legnth)
S
valid = valid + 1


convert (palavra, aux_w)
i = 0
aux_w = " "
i < length (palavra)
i = i +1
palavra = palavra + 'A'
aux_w = ".-"
palavra = palavra + 'B'
aux_w = "-..."
palavra = palavra + 'C'
aux_w = "-.-."
palavra = palavra + 'D'
aux_w = "-.."
palavra = palavra + 'E'
aux_w = "."
palavra = palavra + 'F'
aux_w = "..-."
palavra = palavra + 'G'
aux_w = "--."
palavra = palavra + 'H'
aux_w = "...."
palavra = palavra + 'I'
aux_w = ".."
palavra = palavra + 'J'
aux_w = ".---"
palavra = palavra + 'K'
aux_w = "-.-"
palavra = palavra + 'L'
aux_w = ".-.."
palavra = palavra + 'M'
aux_w = "--"
palavra = palavra + 'N'
aux_w = "-."
palavra = palavra + 'O'
aux_w = "---"
palavra = palavra + 'P'
aux_w = ".--."
palavra = palavra + 'Q'
aux_w = "--.-"
palavra = palavra + 'R'
aux_w = ".-."
palavra = palavra + 'S'
aux_w = "..."
palavra = palavra + 'T'
aux_w = "-"
palavra = palavra + 'U'
aux_w = "..-"
palavra = palavra + 'V'
aux_w = "...-"
palavra = palavra + 'W'
aux_w = ".--"
palavra = palavra + 'X'
aux_w = "-..-"
palavra = palavra + 'Y'
aux_w = "-.--"
palavra = palavra + 'Z'
aux_w = "--.."
S
N

Código em C

```
/* PROJECT MIETI 2016/2017 Métodos de Programação I*/
/* Made by: Leandro Alves nº A82157    */
/* Last edit: 14/01/2017     */
/* FINAL VERSION   */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int convert(char *word, char *aux_w){
 //Scroll through all the letters of the word and convert the whole word to morse
    int i;
    strcpy(aux_w,"");              // "Clean" the variable before using it
    for (i=0;i<strlen(word);i++){
 //Enter each letter of the word to be converted and as we move from letter
        switch(word[i]){
 //to letter until we reach the end of the word we are converting and joining them
            case 'A': strcat(aux_w,".-"); break;
            case 'B': strcat(aux_w,"-..."); break;
            case 'C': strcat(aux_w,"-.-."); break;
            case 'D': strcat(aux_w,"-.."); break;
            case 'E': strcat(aux_w,"."); break;
            case 'F': strcat(aux_w,"..-."); break;
            case 'G': strcat(aux_w,"--."); break;
            case 'H': strcat(aux_w,"...."); break;
            case 'I': strcat(aux_w,".."); break;
            case 'J': strcat(aux_w,".---"); break;
            case 'K': strcat(aux_w,"-.-"); break;
            case 'L': strcat(aux_w,".-.."); break;
```

```
            case 'M': strcat(aux_w,"--"); break;
            case 'N': strcat(aux_w,"-."); break;
            case 'O': strcat(aux_w,"---"); break;
            case 'P': strcat(aux_w,".--."); break;
            case 'Q': strcat(aux_w,"--.-"); break;
            case 'R': strcat(aux_w,".-."); break;
            case 'S': strcat(aux_w,"..."); break;
            case 'T': strcat(aux_w,"-"); break;
            case 'U': strcat(aux_w,"..-"); break;
            case 'V': strcat(aux_w,"...-"); break;
            case 'X': strcat(aux_w,"-..-"); break;
            case 'Y': strcat(aux_w,"-.--"); break;
            case 'W': strcat(aux_w,".--"); break;
            case 'Z': strcat(aux_w,"--.."); break;
            default: return 0; //Return 0 if a character isn't found
        }
    }
}

int compare(int n, char dic[][101], char *mrs_code){
    char w_conv[101];
    int i,valid=0;

    for(i=0; i<n; i++){
        convert(dic[i],w_conv);  //Converts each word of the dictionary to morse

        if(strncmp(mrs_code,w_conv,strlen(w_conv))==0){
//If the translated word is the same as the morse code to the indicated position
//advances to the following line
            if (strlen(mrs_code)==strlen(w_conv)){
 //If the whole morse code is the same as the converted word
```

```
            valid++;          //The valid variable takes the value of valid + 1
        }else{
            valid=valid + compare(n,dic,&mrs_code[strlen(w_conv)]);
 //Calls the function again and does the same but from the end of the word
//found
        }
    }
   //If there is no correspondence continues to the next word
   }//end of the loop
   return valid; //Returns the number of valid sentences found
}


void main(){
   int n_words=0;          //number of words.
   int i;
   int s_valids=0;        //number of possible sentences
   char morse_code[1001];
   char dict[10001][101];
   scanf("%s",morse_code); // Read morse sequence.
   scanf("%d",&n_words); // Reads number of words to put in the dictionary
   for(i=0;i<n_words;i++)
   scanf("%s",dict[i]);
//Reads all words to put in the dictionary until you reach the limit(n_words)
   s_valids=compare(n_words, dict, morse_code);
 //Calls the function that will compare the dictionary and the morse sequence
  //and places the result in s_valids.
   printf("%d",s_valids);  // Print number of possible sentences
}
```

Conclusão

O problema foi solucionado com sucesso através do uso da recursividade, o que permitiu testar as frases válidas ignorando um largo número de combinações que não dariam resultado.

A implementação em C deste problema, ao contrário do algoritmo, foi o processo mais trabalhoso e demorado, nomeadamente na implementação da recursividade da função que compara as palavras com o código morse. Tive dificuldades a implementar esta funçao recursiva pois requereu mais trabalho e dedicação do que as restantes funções que, pelo contrário, foram fáceis de construir.