

# **Universidade da Beira Interior**

## **Departamento de Informática**



**Departamento de  
Informática**

### **Relatório do Exercício 1**

Elaborado por:

**Joel Tapia a47275**  
**Manuel Garcia a45500**  
**Tiago Ribeiro a46346**

9 de abril de 2023



## *Capítulo*

# 1

## *Introdução*

### **1.1 Motivação**

Este projeto está enquadrado na **UC! (UC!)** de Sistemas operativos, do 2<sup>a</sup> ano da Licenciatura de Engenharia Informática, da **UBI! (UBI!)**.

O conceito base passava por implementar um programa que devolve uma sequência de instruções que podem ser utilizadas para encontrar uma string passada como argumento.

### **1.2 Objetivos**

Os objetivos pretendidos para a parte B deste trabalho eram:

- Esconder uma string dada pelo utilizador num ficheiro binário dado pelo mesmo.
- Fornecer uma lista de instruções que permitam encontrar essa string.



## Capítulo

# 2

## ***Desenvolvimento e implementação***

### **2.1 Descrição do código**

A função que gera uma instrução aleatória tem 9 parâmetros importantes, os dois primeiros são feitos para o retorno da função que detém o "operador" e o "número de movimento". (\*n\_pass retorna alterando a variável principal), estas duas variáveis com "binaryl" fazem quase todos os casos. Assim, a primeira parte da função começa com o "switch" que seleciona aleatoriamente o operador, depois cada caso deve ter uma condição que verifique que após a realização da operação não excedemos o tamanho do ficheiro, pelo que todos os se devem ter o "binaril" em comparação com a posição atual "actualposition" então o programa guardará toda a informação se passar as condições e voltar ao principal.

Se o operador escolhido for "r", então realizará a comparação explicada acima, com uma comparação que verifica que o tamanho da string é maior do que o buffer. Depois, sempre que o tamanho da cadeia de caracteres e o tamanho do buffer não forem iguais, e também quando a posição atual estiver no fim do ficheiro, será criando um novo número aleatório que terá um intervalo entre um e o menor de (tamanho do ficheiro - posição atual) e (tamanho do fio - tamanho do buffer). -> (1 - o menor)

A última parte compara se a variável "sample" (que faz uma leitura de n elementos do ficheiro) e o "comparator" (que faz uma cópia de n elementos da palavra a pesquisar), são iguais, caso sejam, salvá-lo-á no buffer, caso contrário voltará ao início.

Toda a função tem um do-while necessário para repetir todo o algoritmo até encontrar um operador e um número que possa passar os condicionantes.

Na função main, no início temos comparadores que comprovam se o fi-

cheiro existe ou se foi possível abri-lo, após alocar as variáveis necessárias começamos um ciclo while que executara os operadores clamando a função generate\_instructions e dependendo do resultado as condicionais fazerem as operações, o comando "read" tem operações especiais que só fazem o guardado no buffer dos caracteres lidos, utilizando primeiro o read numa string "apender" para depois copiar toda a informação no buffer

## 2.2 Código fonte

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5
6 #define MAX_INSTR_LENGTH 6 // Tamanho maximo de uma instrucao
7 // Tamanho maximo do buffer
8
9 // Funcao que gera uma instrucao aleatoria
10 char* generate_instruction(char *instruction, int *n_pass , int ←
    tamDoBuffer, int tamDaStringARevelar ,FILE * position ,int ←
    binaryl, int actualposition, char * stringAProcurar , int *←
    posicaoDaString) {
11     char *inst = malloc(MAX_INSTR_LENGTH * sizeof(char));
12     int n = 0, sum= 0, choice;
13     // printf("Size of string: %d\n",tamDaStringARevelar);
14     //printf("Size of buffer: %d\n",tamDoBuffer);
15
16     do{
17         switch(choice=rand() % 5) {
18             case 0:
19                 strcpy(inst, "+");
20                 break;
21             case 1:
22                 strcpy(inst, "-");
23                 break;
24             case 2:
25                 strcpy(inst, "i");
26                 break;
27             case 3:
28                 strcpy(inst, "f");
29                 break;
30             case 4:
31                 strcpy(inst, "r");
32                 break;
33         }
```

```
34
35 //printf("%d Choice\n",choice);
36
37 if (inst[0] == '+' && binaryl >= actualposition + n) {
38     //puts("Debug1");
39     if(binaryl - actualposition!=0){
40         n= rand() % (binaryl - actualposition) +1;
41     }
42     else{
43         continue;
44     }
45     sprintf(instruction, "%s %d", inst, n);
46     *n_pass = n;
47     if(binaryl >= actualposition + n)
48         sum = 1;
49 }
50
51 else if (inst[0] == '-' && (actualposition-n) >= 1)
52 {
53     n= rand() % actualposition + 1 ;
54     sprintf(instruction, "%s %d", inst, n);
55     *n_pass = n;
56     if((actualposition-n) >= 1)
57         sum = 1;
58 }
59
60 else if (inst[0] == 'i' && binaryl >= n) {
61     n= rand() % binaryl +1 ;
62     sprintf(instruction, "%s %d", inst, n);
63     *n_pass = n;
64     if(binaryl >= n)
65         sum = 1;
66 }
67
68 else if (inst[0] == 'f' && n <= binaryl)
69 {
70     n= rand() % binaryl +1;
71     sprintf(instruction, "%s %d", inst, n);
72     *n_pass = n;
73     if(n <= binaryl)
74         sum = 1;
75 }
76 else if ((inst[0] == 'r') && (binaryl >= actualposition + n) && (←
tamDaStringARevelar > tamDoBuffer)) {
77     //puts("Debug2");
78     if((binaryl - actualposition !=0) && (tamDaStringARevelar ←
tamDoBuffer != 0)){
79         n= rand() % (((binaryl - actualposition)< (←
tamDaStringARevelar -tamDoBuffer)) ? (binaryl ←
```

```

        actualposition) : (tamDaStringARevelar - tamDoBuffer)) ←
            +1;
80     }
81     else {
82         continue;
83     }
84     sprintf(instruction, "%s %d", inst, n);
85     *n_pass = n;
86
87     char *sample = malloc( sizeof(char) * (n+1));
88     fread(sample,1,n,position);
89     sample[n] = '\0';
90     char *comparator = malloc(sizeof(char) * (n+1));
91     strncpy(comparator,stringAProcurar,n);
92     comparator[n] = '\0';
93     // printf("Sample:%s\n",sample);
94     // printf("Comparator:%s\n",comparator);
95     if (n + actualposition <= binaryl && (n <= (tamDaStringARevelar - ←
        tamDoBuffer)) && strcmp(comparator , sample) == 0){
96         sum = 1;
97         *posicaoDaString += n;
98     }
99
100    if(strcmp(comparator, sample) != 0){
101        fseek(position,-n,SEEK_CUR);
102    }
103
104    free(sample);
105    free(comparator);
106
107    }
108    } while (sum != 1 );
109    if(n!=0)
110        printf("%s\n",instruction);
111    return instruction;
112 }
113
114 int main(int argc, char* argv[]) {
115     if (argc != 3) {
116         printf("Binary file and message required\n");
117         return 1;
118     }
119
120     FILE* binary_file = fopen(argv[1], "rb");
121     if (binary_file == NULL) {
122         printf("Error opening file\n");
123         return 1;
124     }
125

```



```
126     char* string = argv[2];
127     strtok(string, "\n");
128     int string_len = strlen(string);
129     fseek(binary_file, 0, SEEK_END);
130     int binarylength = ftell(binary_file);
131     rewind(binary_file);
132
133     srand(time(NULL)); // Inicializa o gerador de numeros aleatorios
134
135     char instruction[MAX_INSTR_LENGTH];
136     int param;
137     int maxBufferSize = strlen(argv[2]) + 1 ;
138     char * buffer = malloc(maxBufferSize);
139     buffer[maxBufferSize] = '\0';
140     int buffer_pos = 0;
141     int posicaoNaString = 0;
142
143     while (strlen(buffer) < string_len) {
144
145         generate_instruction(instruction, &param, strlen(buffer) , ←
            strlen(string), binary_file, binarylength, ftell(←
            binary_file), &(string[posicaoNaString]), &posicaoNaString←
            );
146
147         if (instruction[0] == '+' && param != 0) {
148             fseek(binary_file, param, SEEK_CUR);
149         }
150         else if (instruction[0] == '-' && param != 0) {
151             fseek(binary_file, -param, SEEK_CUR);
152         }
153         else if (instruction[0] == 'i' && param != 0) {
154             fseek(binary_file, param, SEEK_SET);
155         }
156         else if (instruction[0] == 'f' && param != 0) {
157             fseek(binary_file, -param, SEEK_END);
158         }
159         else if (instruction[0] == 'r' && param != 0) {
160             /*fread(buffer + buffer_pos, 1, param, binary_file);
161             buffer_pos += param;*/
162
163             char * apender = malloc(sizeof(char) * (param + 1));
164
165             fread (apender, 1, param ,binary_file );
166             apender[param] = '\0';
167
168             strcat(buffer, apender);
169
170             free(apender);
171         }
```

```
172     else {continue;}
173
174     if (buffer_pos >= maxBufferSize -1) {
175         printf("s 0\n");
176         memset(buffer, 0,maxBufferSize);
177         buffer_pos = 0;
178     }
179 }
180
181 free(buffer);
182 fclose(binary_file);
183 printf("s 0\n");
184 //printf("%s\n",buffer);
185 return 0;
186 }
```

Excerto de Código 2.1: Código fonte exercicio 1

## Capítulo

# 3

## Exemplos de execução

```
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$ gcc -std=c99 -Wall -o ProgA -lm main.c
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$ ./ProgA test.bin < input1.txt
password
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$ gcc -std=c99 -Wall -o ProgB -lm mainTiago.c
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$ ./ProgB test.bin password > inputA.txt
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$ ./ProgB test.bin password
- 2
i 10
- 4
f 5
i 8
- 5
+ 2
r 1
f 10
f 18
i 3
+ 5
- 4
i 4
f 17
f 4
+ 4
- 10
f 9
- 7
- 1
r 3
s 0
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$ ./ProgA test.bin < inputA.txt
password
caesar@LeNovoDasCouves:~/S0/trabalho1ParteB$
```