

# Trabalho Prático



Licenciatura em Engenharia Informática

Programação  
2016/2017

## Relatório

José Hugo Sousa Silva nº21240009 – [21240009@isec.pt](mailto:21240009@isec.pt)

## Índice

1. Introdução.....	3
2. Estruturas de dados.....	3
3. Estruturas Dinâmicas.....	5
4. Ficheiros .....	7
5. Opções Tomadas .....	7
6. Estrutura Geral .....	8
7. Manual de Utilização .....	10
8. Conclusão .....	11

## 1. Introdução

O presente trabalho é sobre o programa Zoo Vale das Flores, mais concretamente um programa em Linguagem C para a gestão de o Zoo Vale das Flores. Neste relatório abordarei as soluções para os problemas submetidos no enunciado do Trabalho Prático. Demonstrarei as funcionalidades, e o código que é utilizado para essas funções funcionarem, também demonstrarei as estruturas utilizadas, descrição dos ficheiros utilizados, e justificarei as opções tomadas em termos da implementação, e farei também um pequeno manual de utilização do programa.

## 2. Estruturas de dados

Neste tópico, explicarei brevemente as estruturas de dados utilizadas na realização deste trabalho prático.

Estrutura das áreas:

```
struct areas{  
    char nome[DIM];  
    int capacidade;  
    int ocupacao;  
    int n_areas_adj;  
    char aread[3][100];  
    animas lista;  
};
```

Utilizei para representar os dados pedidos pelo enunciado, esta estrutura para representar as áreas. Com 2 variáveis tipo char, onde uma delas é simples, com uma constante DIM como tamanho. Na segunda variável char aread[3][100], utilizei esta denominação para poder guardar as áreas adjacentes que são precisas guardar, com um limite de 3, e é um array multidimensional de strings.

Em seguida utilizei, 3 variáveis do tipo int, para guardar a capacidade da área, outra para guardar a ocupação que esta possui, e por fim, outra para guardar o numero de áreas adjacentes que a área possui. Por fim, tenho um ponteiro para a estrutura dos animais, onde têm como objetivo apontar para o inicio da lista de animais que estão nesta área. A variável é do tipo struct, e possui a mesma designação que struct animais \*lista.

Estrutura das espécies:

```
struct especie{  
    char nome[50];  
    char n_serie[50];  
};
```

Utilizei esta estrutura para atribuir “automaticamente” um numero de série para cada espécie. Mas o automaticamente, é escolhido pelo utilizador, uma opção que optei, mas não sei se era mesmo necessário criar

outra que atribuí a um novo numero de série automaticamente, se fosse necessário, bastava continuar com esta estrutura e sempre que fosse criada uma nova espécie, incrementar uma variável e escrever por exemplo especie1, especie2, e por aí adiante. Em termos de variáveis, 2 variáveis do tipo char, onde a função é funcionar como string, para armazenar o nome da espécie e o seu respetivo numero de serie.

Estrutura dos animais:

```
struct animais{
    char especie[50];
    char n_serie[50];
    char nome[100];
    int peso;
    char pai[100];
    char mae[100];
    char area[100];
    animas prox;
    animas filho;
    animas filho_p;
    animas filhom;
    animas filhom_p;
};
```

Na estrutura dos animais, utilizei varias variáveis, começarei pelas mais fáceis, e depois passarei a explicar as variáveis com mais complexidade. Utilizei uma variável do tipo int para armazenar o peso do animal. Como é visível utilizei 6 variáveis do tipo char, que são todas strings, onde utilizo para armazenar o numero de serie, a espécie a que pertence o animal, o seu nome, os seus pais, e a que área pertence, sendo possível também, verificar a área que pertence a partir do ponteiro que aponta para a lista.

Olhando para as variáveis do tipo animas que é o mesmo que struct \*animais e o nome da variável. Utilizei o animas prox para ligar aos animais da área, utilização simples e correta. Agora as 4 próximas variáveis do tipo animas(struct \*animais), são fruto de algum pensamento, mas um pensamento errado, porque enfrentei vários

problemas durante a tentativa de representar a família dos animais, uma das soluções que pensei, era alocar novamente, os animais só com o nome, mas como já estavam criados pensei em utilizar os animais criados para ligar. Mas descobri que utilizar esta solução possui vários problemas, que não tive tempo de corrigir, e que devia ter utilizado para manipular a informação de maneira diferente. Um dos problemas que me surgiu foi o seguinte, tentei utilizar só um filho e ponteiro para o próximo, e tecnicamente era uma coisa que devia funcionar, mas funcionava se alocasse novamente os animais. Porque, se tem 2 pais, ele não pode estar ligado aos 2, e depois também, a mãe pode ter filhos com outro animal, e, depois esse animal pode ter filhos com outro animal, e teríamos um problema que é, a família ia juntar-se e família que não era família, e acabar por ser família. Tentei resolver isto mas não foi possível, o que implementei, foi 2 ponteiros para as listas, e depois dois ponteiros para manipular a lista, para a ligar a outros animais, o problema é que depois pode-se um animal no meio, e a lista fica completamente diferente, com família que não devia ser, como expliquei em cima.

### 3. Estruturas Dinâmicas

Neste tópico, abordarei o tipo de estruturas dinâmicas utilizadas na realização deste trabalho prático.

Vetores Dinâmicos:

Utilizei 2 vetores dinâmicos, sendo um pedido obrigatoriamente pelo enunciado. Apresentarei primeiro o vetor pedido pelo enunciado, em seguida o segundo vetor dinâmico.

Na função principal(main) declarei uma variável do tipo struct areas \*array(area array), onde está apontar para o vazio(NULL). Utilizo este vetor para armazenar as áreas existentes no zoo, primeira vez preencho o vetor dinâmico é na função le\_ficheiro, onde, chamo array = le\_ficheiro(nomeFich,&tam);. A função le\_ficheiro le as áreas, e aloca memória para o vetor dinâmico. Ao longo da execução o vetor dinâmico pode sofrer alterações como, criar área ou eliminar área, e também no final onde, é guardada a informação do vetor num ficheiro de texto. E o tam é uma variável, que guarda o tamanho do vetor dinâmico.

No segundo vetor dinâmico, utilizei para guardar as espécies existentes. Com quase a mesma funcionalidade que o vetor dinâmico 1.

Listas Ligadas:

Utilizei 3 listas ligadas, sendo 2 obrigatórias, mas só aloquei memória para uma lista, as outras aproveitei já o existente para ligar umas as outras, não é a melhor implementação porque como referi em cima, tem muitas variáveis condicionante. A primeira lista ligada, tem a funcionalidade correta as outras 2 é que podiam ser melhor utilizadas.

Primeira lista ligada utilizei em várias funções, com o começo da lista na estrutura da área, e com ponteiro para o próximo na estrutura dos animais, assim, nunca perdendo o início da lista. Uma das funções que manipula, esta lista ligada é a função lista animais, que está anexada em baixo.

```
//Listagem completa de todos os animais do zoo
void listar_animais(area pv,int *tam){
    animas c;
    limpa_ecra();
    printf("Listagem de animais!\n");
    for(int i = 0;i<*tam;i++){
        //percorre todas as areas e imprime todos os animais que se encontram nelas
        if((pv+i)->lista != NULL){
            c = (pv+i)->lista;
            while(c != NULL){
                printf("Nome: %s\tN_Serie: %s\tEspecie: %s\t%s\tPeso: %dKg\n",c->nome,c->n_serie,c->especie,c->area,c->peso);
                c = c->prox;
            }
        }
    }
    system("Pause");
}
```

Utilizando o ponteiro próximo para andar na lista. E também as áreas com o ponteiro para o começo da lista, onde se pode aceder as listas a partir desse endereço, facilitando assim o trabalho.

Nas outras listas ligadas, é mais difícil de explicar, porque utilizei para “linkar” a família, e como descrito anteriormente não é melhor solução e mesmo assim tenho problemas com essa solução. Cria muitos problemas que não consegui encontrar solução, a solução mais viável era alocar outra vez memória para cada animal e criar uma estrutura família onde tinha o ponteiro para essa família. Função onde mostra a manipulação destas 2 listas, sendo difícil explicar detalhadamente, o que se sucede, o código está com comentários para a explicação. As 2 listas funcionam como pai e mãe, mas como os animais não era pedido o sexo, dificultou-me um bocado a tarefa, mesmo assim tentei fazer funcionar dessa forma.

```
void associa_fam(area pv,int *tam){
    //Funcao auxiliar para associar familia
    animas aux = NULL,seg = NULL;
    int i = 0,j = 0;
    for(i = 0;i<*tam;i++){ //percorrer areas
        if((pv+i)->lista != NULL){ //se tiver lista
            aux = (pv+i)->lista;
            while(aux != NULL){ // enquanto houver animal na area
                for(j = 0;j<*tam;j++){ //percorrer outra vez a lista
                    if((pv+j)->lista != NULL){
                        //encontrar os pais
                        seg = (pv+j)->lista;
                        while(seg != NULL){
                            if(strcmp(seg->nome,aux->mae) == 0){
                                aux->filhom_p = seg->filhom;
                                seg->filhom = aux;
                            }

                            //verificar se é pai ou mae
                            if(strcmp(seg->nome,aux->pai) == 0){
                                //Linkar listas
                                aux->filho_p = seg->filho;
                                seg->filho = aux;
                            }
                            //andar para o proximo na lista
                            seg = seg->prox;
                        }
                    }
                }
            }
            aux = aux->prox;
        }
    }
}
```

## 4. Ficheiros

Neste tópico abordarei os ficheiros que foram utilizados para a realização do trabalho prático.

Utilizei o ficheiro das áreas onde possui o nome de areas.txt, e é onde é armazenado as áreas que existem no zoo. Criei dois ficheiros um denominado por d.txt, e outro por ola.txt, que servem para adicionar animais ao zoo, via ficheiro. O resto dos ficheiros são ficheiros binários onde guardo os dados no final do programa. Um deles é o animais.dat que guarda a informação sobre todos os animais existentes no zoo, e no início do programa liga ao programa. O outro binário denominado por especie.dat serve para guardar a informação sobre as espécies dos animais existentes no zoo.

## 5. Opções Tomadas

Na realização deste trabalho prático deparei-me com algumas situações em que tive de optar por uma opção. E são essas opções que abordarei neste tópico.

Áreas:

Em termos de opções tomadas na implementação das áreas, optei por uma abordagem simples, onde faço um vetor dinâmico que gere as áreas, depois é o generalista, muito simples.

Animais:

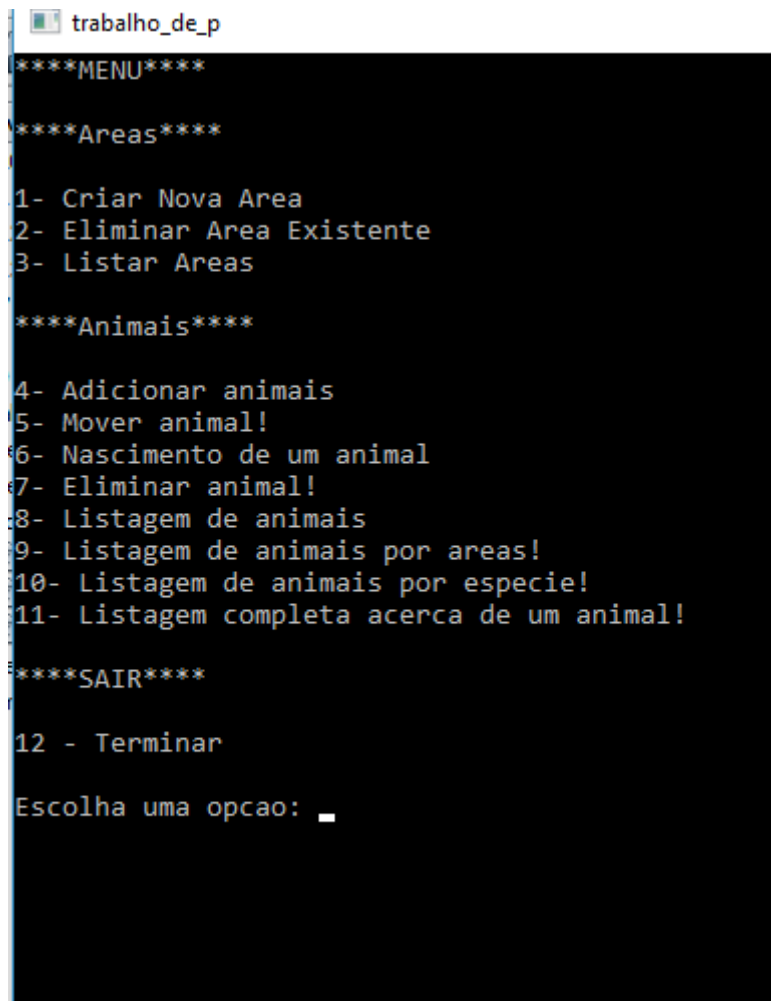
Nos animais em termos de opções tomadas é o generalista também, a única parte onde não é generalista é na abordagem a família, onde tive de tomar uma decisão que foi um bocado infeliz, porque não é nem de perto a mais correta para a solução pretendida, que é possuir 2 listas para a família, onde tem a que é pai, e onde é mãe, mais ao menos isto, que podia ser melhor pensado, mas que não consegui chegar a um consenso comigo próprio.

Na atribuição automática de espécie, optei por utilizar uma estrutura dinâmica para armazenar a informação sobre as espécies, o seu numero e nome, e guardar num ficheiro ao fim da execução e no seu começo. Optei por criar uma função que liga as famílias, pelo pai e pela mãe. O resto do programa são opções sensatas e que é o pedido pelo enunciado.

## 6. Estrutura Geral

Neste tópico abordarei a estrutura geral e apresentarei alguns algoritmos de alto nível que clarificaram as principais funcionalidades do programa.

O programa faz tudo o que é pedido, desde criar áreas, eliminar áreas, transferir animais, adicionar animais, entre outros. É um programa simples com uma estrutura geral compacta e bem definida. Segue-se o menu na figura seguinte.



```
trabalho_de_p
****MENU****

****Areas****

1- Criar Nova Area
2- Eliminar Area Existente
3- Listar Areas

****Animais****

4- Adicionar animais
5- Mover animal!
6- Nascimento de um animal
7- Eliminar animal!
8- Listagem de animais
9- Listagem de animais por areas!
10- Listagem de animais por especie!
11- Listagem completa acerca de um animal!

****SAIR****

12 - Terminar

Escolha uma opcao: _
```

Como é possível verificar é um programa simples e de fácil utilização, com apenas um menu, onde o utilizador pode manipular o sistema.



Algoritmos de alto nível:

Algoritmo para a função `le_ficheiro`:

1. Abrir ficheiro `areas.txt` para leitura.
2. Ler o número de estruturas armazenadas
3. Alocar vetor dinâmico
4. Ler as estruturas do ficheiro para o vetor
5. Fechar o ficheiro
6. Devolver o endereço do novo vetor dinâmico e o seu tamanho

Algoritmo para a função `eliminar_animal`:

1. Pede ao utilizador o animal que pretende eliminar
2. Percorre o vetor dinâmico das áreas
3. Procura o nome na lista ligada que cada área possui
4. Encontra animal e desconeta-o da lista ligada
5. Liga a lista ligada a outra metade da lista ligada
6. Retira peso de ocupação da área
7. Liberta memória

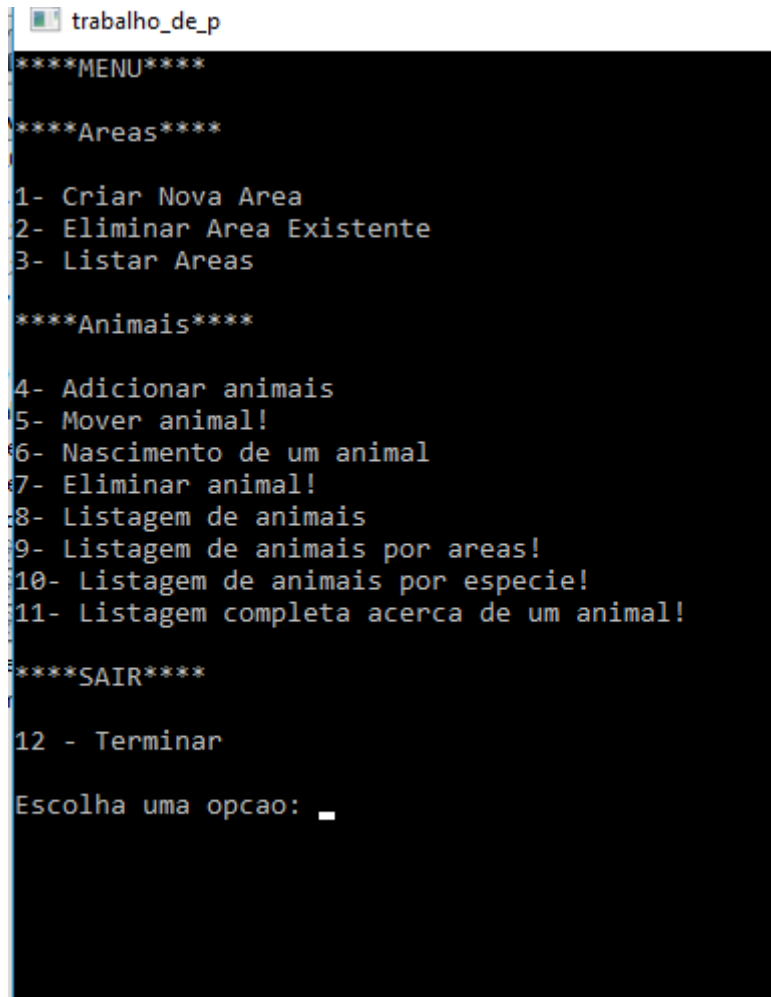
Algoritmo para a função `listar_local`:

1. Pede ao utilizador o nome da Area
2. Percorre o vetor dinâmico até encontrar a área
3. Percorre a lista ligada de animais
4. Imprime os animais

## 7. Manual de Utilização

Neste tópico abordarei brevemente como utilizar o programa.

Menu simples, como é possível verificar na figura seguinte.



```
trabalho_de_p
****MENU****

****Areas****

1- Criar Nova Area
2- Eliminar Area Existente
3- Listar Areas

****Animais****

4- Adicionar animais
5- Mover animal!
6- Nascimento de um animal
7- Eliminar animal!
8- Listagem de animais
9- Listagem de animais por areas!
10- Listagem de animais por especie!
11- Listagem completa acerca de um animal!

****SAIR****

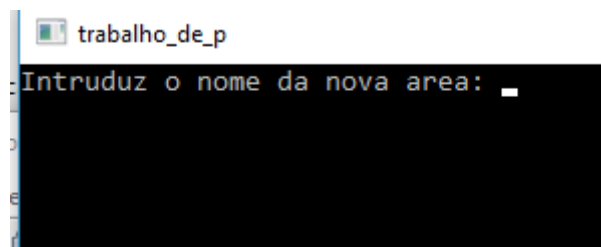
12 - Terminar

Escolha uma opcao: _
```

É pedido ao utilizador uma opção depois de escolhida, vai para a opção e o utilizador preenche como pedido.

Vou dar 1 exemplo, para o utilizador verificar a fácil utilização.

Opção 1- Criar Nova Área



```
trabalho_de_p
Intruduz o nome da nova area: _
```

Depois preenche como pedido, o resto dos passos, sempre de forma facilitada.

## **8. Conclusão**

Com este trabalho, posso concluir que é importante planejar antes de executar o código, porque surgem problemas que sem um pensamento articulado é impossível de resolver. Também pode trabalhar pela primeira vez com um projeto de dimensão grande, o que me despertou um gosto por programar.