Engenharia Informática

Programação - Trabalho Prático

RELATÓRIO

José Miguel Dias Valdeviesso Alves
21240042
P11 - Programação

Índice

Estruturas

```
typedef struct area_st{
  char id[100];
  int tipo;
  int cap;
  int pesoAct;
  //fronteiras
  int nr_front;
  char front1[100];
  char front2[100];
  char front3[100];
}Areas;
```

Estrutura das áreas

- Usada nos vectores dinamicos das areas do zoo

id » Identificador unico de cada area no zoo
tipo » tipo de area que é, 1 ou 2, jaula ou area aberta
cap » peso maximo que a area suporta
pesoAct » peso actual dos animais na área, começa a 0 quando as áreas são lidas do ficheiro ou uma nova área é adiciona;
nr_front » indica o numero de fronteiras que a area tem
front1-3 » id único da área que faz fronteira com esta area.

```
typedef struct animais_st{
  int id;
  char especie[100];
  char nome[100];
  int peso;
  char loc[100];
  //familia
  struct fam filho;
  struct fam pais;
  //next ID
  struct animais_st *prox;
}Animais;
```

Estruturas dos animais

- Usadas nas listas ligados dos animais

id » identificador único do animal, atribuído quando o animal é adicionado ao programa
especie » especie do animal
nome » nome do animal
peso » peso do animal
loc » identificador unido da localização onde o animal está
filhos » estrutura com informação relativa aos filhos do animal
pais » estrutura com informação relativa aos pais do animal
***prox** » ponteiro para o próximo elemento da lista ligada (próximo animal).

```
struct fam{
        int fnr;
        int fID;
        int fID2;
        struct animais_st *strt;
};
```

Estrutura *fam*

- Usada para guardar informações relativas à familia de determinado animal.

fnr »

- Quando usada para os filhos, indica o número de filho que o animal tem
- Quando usada para os pais, indica o número de pais que o animal tem dentro do zoo.

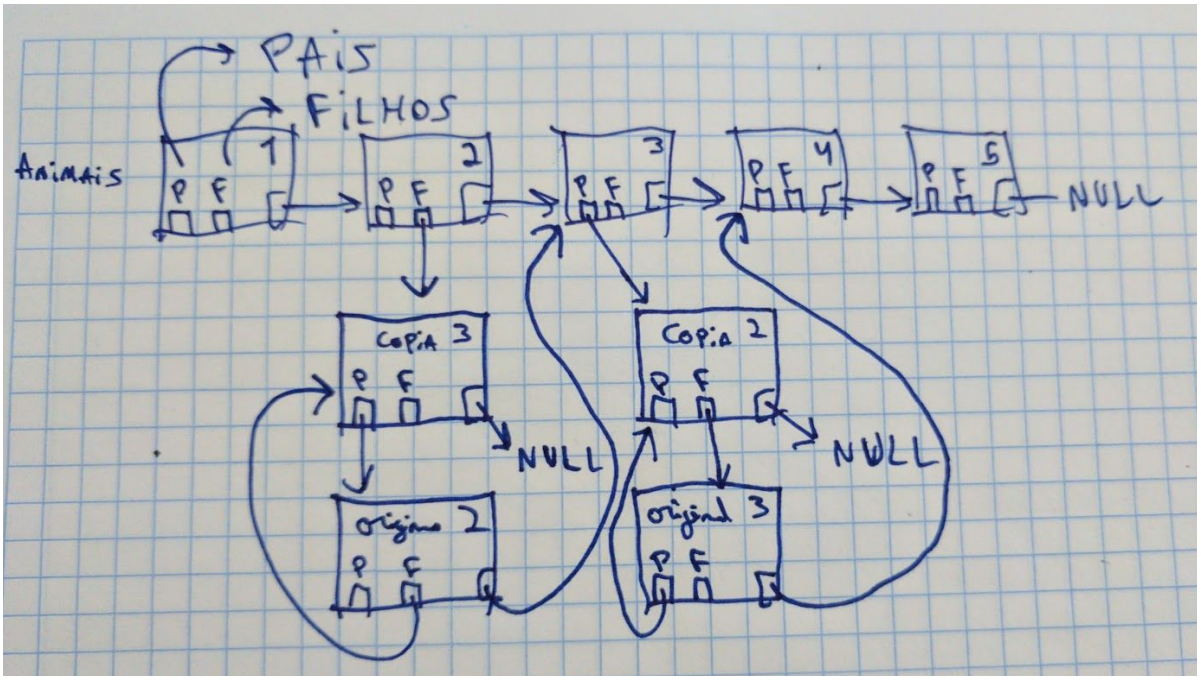
fID » Guarda o id único de um dos pais do animal
fID2 » Guarda o id único do outro pai/mae do animal
***strt** »

- Quando usada para os filhos, é um ponteiro que aponta para uma cópia da informação relativa ao primeiro filho do animal, em que o *prox desta estrutura aponta para outro(s) filho(s) que o animal tenha dentro do zoo,

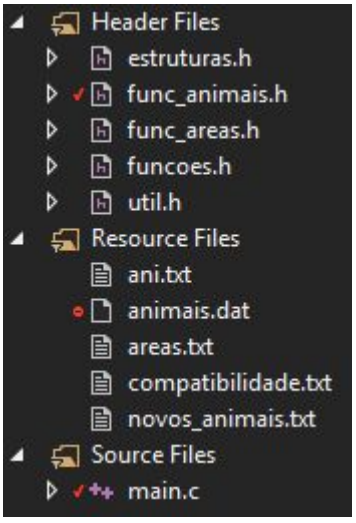
tudo usando cópias das estruturas originais, e nunca está ligado aos *blocos* de informação na lista original.

- Quando usado para os pais, é um ponteiro que aponta para uma cópia da informação de um dos pais, sendo que o *prox desta estrutura aponta para o outro pai/mãe que o animal possa ter, ou NULL caso só tenha um pai/mãe

Exemplo de uma lista ligada de animais com pais e filhos:



Ficheiros



estruturas.h » localização de todas as estruturas usadas
func_animais.h » todas as funções relativas aos animais
func_areas.h » todas as funções relativas às áreas
funções.h » funções que não são relativas às áreas ou animais
util.h » inicialização de todas as funções do programa
ani.txt » ficheiro com alguns animais para adicionar.
animais.dat » ficheiro binário com os animais presentes no zoo
areas.txt » ficheiro com as áreas pertencentes ao zoo
novos_animais.txt » ficheiro com alguns animais para adicionar.
main.c » ficheiro principal do programa

(não usado) **compatibilidade.txt** » ficheiro que indica a incompatibilidade entre especies de animais.

Código

```
int main(void) {
	int menuopt, nrAreas, last_animal_id, tType=-1, tCap, tNR=0, valid1=0, valid2=0,
valid3=0, id=0;
	char ch, t1[100], t2[100], t3[100], arID[100];
	bool compActive = false;
	Areas *zAreas = malloc(sizeof(Areas));
	Animais *zAnimais = NULL;

	zAnimais = malloc(sizeof(Animais));

	dispIntro();
```

Começo por definir algumas variáveis que serão usadas durante as múltiplas funções do programa.

Inicializo as áreas e os animais, e dou-lhes o tamanho de 1 elemento.

Depois de mostrar o intro (um logo com o nome do programa) vou preencher o vector dinâmico das áreas do zoo:

```
zAreas = readAreas(zAreas,&nrAreas);
```

A função readAreas recebe o ponteiro inicial das áreas e o numera de areas ja existentes, devolve o ponteiro para o início do vetor dinâmico e altera o valor do número de áreas.

```
Areas * readAreas(Areas *tAreas, int *nrAreas){
	int tType, tCap, tNR, i=0;
	char tID[100], t1[100], t2[100], t3[100];
	FILE *f;
	printf("A abrir ficheiro de Areas...\n");
	f = fopen("areas.txt", "r");
	if(f==NULL){
		printf("Erro a abrir ficheiro de areas!\n");
		exit(0);
	}
	printf("A preencher vector...\n");
	fscanf(f,"%s\t%d\t%d\t%d", &tID, &tType, &tCap, &tNR);
	do{
		*nrAreas = i+1;
		if(tNR == 0){
			tAreas = addAreaEnd(tAreas, tID, tType, tCap, tNR, "", "", "", *nrAreas);
		}else if(tNR == 1){
			fscanf(f,"\t%s",&t1);
			tAreas = addAreaEnd(tAreas, tID, tType, tCap, tNR, t1, "", "", *nrAreas);
		}else if(tNR == 2){
			fscanf(f,"\t%s\t%s",&t1, &t2);
			tAreas = addAreaEnd(tAreas, tID, tType, tCap, tNR, t1, t2, "", *nrAreas);
		}else if(tNR == 3){
			fscanf(f,"\t%s\t%s\t%s",&t1,&t2,&t3);
			tAreas = addAreaEnd(tAreas, tID, tType, tCap, tNR, t1, t2, t3, *nrAreas);
		}else{
			exit(0);
		}
		fscanf(f,"%s\t%d\t%d\t%d", &tID, &tType, &tCap, &tNR);
		i++;
	}while(!feof(f));
	fclose(f);
	printf("%d areas carregadas!!\n", i);
	return tAreas;
}
```

Começo por definir variáveis que serão preenchidas com dados que serão lidos do ficheiro onde as áreas estão guardadas.

Verifico se o ficheiro foi aberto com sucesso, caso não tenha encontrado ficheiro, ou tenha ocorrido algum erro, o programa termina imediatamente dizendo ao utilizador que o ficheiro de áreas não foi aberto com sucesso.

Depois de ter o ficheiro aberto com sucesso, vou ler os primeiros 4 elementos da primeira linha, de modo a guardar o nome/id, tipo, capacidade e número de fronteira que essa área tem, após saber o número de fronteira que a área tem vou ler 0, 1, 2, 3 mais elementos no ficheiro caso as fronteiras sejam 0, 1, 2 ou 3 respectivamente.

Por fim, vou chamar a função *addAreaEnd* que me vai adicionar os dados lidos ao vetor dinâmico e me devolve o ponteiro para o início do mesmo.

E vou repetir este processo até chegar ao final do ficheiro das áreas, fechando o ficheiro no fim, e retornando o ponteiro para o 1º elemento do vetor.

Função *addAreaEnd*:

```
Areas * addAreaEnd(Areas *tAreas, char *id, int tipo, int cap, int nr_front, char
*front1, char *front2, char *front3, int nrAreas) {
	tAreas = realloc(tAreas, (sizeof(Areas)*nrAreas));

	strcpy(tAreas[nrAreas - 1].id, id);
	tAreas[nrAreas-1].tipo = tipo;
	tAreas[nrAreas-1].cap = cap;
	tAreas[nrAreas-1].pesoAct = 0;
	tAreas[nrAreas-1].nr_front = nr_front;
	strcpy(tAreas[nrAreas-1].front1, front1);
	strcpy(tAreas[nrAreas-1].front2, front2);
	strcpy(tAreas[nrAreas-1].front3, front3);

	return tAreas;
}
```

Recebe o ponteiro para o 1º elemento do vetor e todos os dados necessários para adicionar um elemento novo ao vetor.

Realoca a memória do vetor as áreas para adicionar espaço para mais 1 elemento e copia os dados recebidos para o último elemento do vetor, retornando depois o ponteiro para o início do vetor.

Voltando à main já com com as áreas lidas e o vetor de áreas criado, vou agora preencher a lista ligada dos animais com os animais do ficheiro binário de animais caso ele exista.

```
zAnimais = readAnimais(zAnimais, zAreas, nrAreas, compActive);
```

Chamando a função *readAnimais* que recebe o ponteiro para o 1º elemento dos animais, o ponteiro para as áreas, o número de áreas existentes e um true/false para a incompatibilidade entre os animais.

```
Animais * readAnimais(Animais *tAnimais, Areas *tAreas, int nrAreas, bool compActive) {
  Animais *temp = tAnimais;
  Animais *read;
  FILE *f;
  int i, i2;
  unsigned long end;
  printf("A procurar ficheiro de Animais...\n");
  f = fopen("animais.dat", "rb");
  if (f == NULL) {
	printf("Fichero de animais nao encontrado, ou erro a abrir!\n");
	temp->prox = NULL;
	return tAnimais;
  }
  printf("Ficheiro de animais encontrado e aberto!\n");
  if (compActive) {//caso incompatibilidade entre animais esteja activa

  }
  else {//caso nao esteja activa
```

```
        while (!feof(f)) {
          read = malloc(sizeof(Animais));
          end = fread(read, sizeof(Animais), 1, f);
          if (end != 1) break;
          for (i = 0; i < nrAreas; i++) {
               if (strcmp(tAreas[i].id, read->loc) == 0) {
                 if ((tAreas[i].pesoAct + read->peso) > tAreas[i].cap) {//area ia ficar
com peso a mais!
                       if (tAreas[i].nr_front == 1) {
                         for (i2 = 0; i2 < nrAreas; i2++) {
                               if (strcmp(tAreas[i2].id, tAreas[i].front1) == 0) {
                                 if ((tAreas[i2].pesoAct + read->peso) <= tAreas[i2].cap) {
                                       tAreas[i2].pesoAct += read->peso;
                                       strcpy(read->loc, tAreas[i2].front1);
                                       temp->prox = read;
                                       temp = temp->prox;
                                       fclose(f);
                                       return tAnimais;
                                 }
                               }
                         }
                       }
                       if (tAreas[i].nr_front == 2) {
                         for (i2 = 0; i2 < nrAreas; i2++) {
                               if (strcmp(tAreas[i2].id, tAreas[i].front2) == 0) {
                                 if ((tAreas[i2].pesoAct + read->peso) <= tAreas[i2].cap) {
                                       tAreas[i2].pesoAct += read->peso;
                                       strcpy(read->loc, tAreas[i2].front2);
                                       temp->prox = read;
                                       temp = temp->prox;
                                       fclose(f);
                                       return tAnimais;
                                 }
                               }
                         }
                       }
                       if (tAreas[i].nr_front == 3) {
                         for (i2 = 0; i2 < nrAreas; i2++) {
                               if (strcmp(tAreas[i2].id, tAreas[i].front3) == 0) {
                                 if ((tAreas[i2].pesoAct + read->peso) <= tAreas[i2].cap) {
                                       tAreas[i2].pesoAct += read->peso;
                                       strcpy(read->loc, tAreas[i2].front3);
                                       temp->prox = read;
                                       temp = temp->prox;
                                       fclose(f);
                                       return tAnimais;
                                 }
                               }
                         }
                       }
                 }
                 else {//area tem capacidade para ter o animal
                       tAreas[i].pesoAct += read->peso;
                       temp->prox = read;
                       temp = temp->prox;
                       break;
                 }
               }
          }
        }
  }

  fclose(f);
  return tAnimais;
}
```

Começo por copiar o ponteiro do início da lista para outra variável para facilitar a utilização e navegação dentro da lista ligada.

Vou também criar uma estrutura temporária para guardar os dados que vou ler ao ficheiros binário das áreas.

Caso o ficheiro não exista a função retorna uma lista de animais vazia.

Após confirmar que o ficheiro foi aberto com sucesso, vou alocar espaço para a minha estrutura temporária e usar *fread* para ler um determinado número de bytes e guardá-los na estrutura *temp*, guardando também o número de elementos que a função *fread* leu.

Caso *fread* tenho lido alguma coisa diferente de 1 elemento, deixo de ler o ficheiro pois sei que cheguei ao fim do mesmo e retorno a lista criada até ao momento.

No caso da leitura do *fread* tenha sido válida vou procurar no vetor das áreas a área a que o animal diz que pertence e verificar se a área em questão iria ou não ficar com excesso de peso ao adicionar este determinado animal. Se a área ficasse com peso a mais, vou procurar verificar se alguma fronteira da área original tem capacidade de receber esse animal e caso tenha o animal é adicionado a essa área.

Se a área tem capacidade para ficar com o animal vou então adicionar o novo animal à lista de animais e incrementar o peso actual no vetor da áreas. Repetindo isto até chegar ao final do ficheiro.

Voltando à main, temos agora a função *linkAnimais*, que trata das ligações entre pais e filhos dos animais.

Nota: os animais penas guardam informação de quem são os seus pais quando o programa escreve ou lê do ficheiro binário.

Função *linkAnimais*, recebe o ponteiro para o primeiro elemento dos animais.

```
void linkAnimais(Animais *tAnimais) {
  Animais *ani = tAnimais; //animal principal
  Animais *anit;//animal que é pai do ani
  Animais *ftemp; //filho do animal anit
  Animais *ptemp; //pai do animal ani
  Animais *temp; //temp geral, pode ser muita coisa

  while (ani->prox != NULL) {//reset links
       ani = ani->prox;
       ani->filho.strt = NULL;
       ani->pais.strt = NULL;
  }

  ani = tAnimais;

  while (ani->prox != NULL) {
       ani = ani->prox;
       if (ani->pais.fnr == 1) {//caso o animal tenha 1 pai
         anit = tAnimais;
         while (anit->prox != NULL) {
              anit = anit->prox;
              if (anit->id == ani->pais.fID) {
                ptemp = anit;
                ftemp = ani;
                break;
              }
         }
         ani->pais.strt = malloc(sizeof(Animais));
         memcpy(ani->pais.strt, ptemp, sizeof(Animais));
         ani->pais.strt->prox = NULL;
         ani->pais.strt->filho.strt = ani;

         if (anit->filho.strt) {
              temp = anit->filho.strt;
              while (temp->prox != NULL) {
                temp = temp->prox;
              }
              temp->prox = malloc(sizeof(Animais));
              memcpy(temp->prox, ftemp, sizeof(Animais));
              temp->prox->prox = NULL;
              temp->prox->pais.strt = anit;
         }
         else {
              anit->filho.strt = malloc(sizeof(Animais));
              memcpy(anit->filho.strt, ftemp, sizeof(Animais));
              anit->filho.strt->prox = NULL;
```

```
                anit->filho.strt->pais.strt = anit;
            }
        }
        if (ani->pais.fnr == 2) {//caso o animal tenha 2 pais
                //pai 1
            anit = tAnimais;
            while (anit->prox != NULL) {
                anit = anit->prox;
                if (anit->id == ani->pais.fID) {
                    ptemp = anit;
                    ftemp = ani;
                    break;
                }
            }
            ani->pais.strt = malloc(sizeof(Animais));
            memcpy(ani->pais.strt, ptemp, sizeof(Animais));
            ani->pais.strt->prox = NULL;
            ani->pais.strt->filho.strt = ani;

            if (anit->filho.strt) {
                temp = anit->filho.strt;
                while (temp->prox != NULL) {
                    temp = temp->prox;
                }
                temp->prox = malloc(sizeof(Animais));
                memcpy(temp->prox, ftemp, sizeof(Animais));
                temp->prox->prox = NULL;
                temp->prox->pais.strt = anit;
            }
            else {
                anit->filho.strt = malloc(sizeof(Animais));
                memcpy(anit->filho.strt, ftemp, sizeof(Animais));
                anit->filho.strt->prox = NULL;
                anit->filho.strt->pais.strt = anit;
            }

            //pai 2
            anit = tAnimais;
            while (anit->prox != NULL) {
                anit = anit->prox;
                if (anit->id == ani->pais.fID2) {
                    ptemp = anit;
                    ftemp = ani;
                    break;
                }
            }
            ani->pais.strt->prox = malloc(sizeof(Animais));
            memcpy(ani->pais.strt->prox, ptemp, sizeof(Animais));
            ani->pais.strt->prox->prox = NULL;
            ani->pais.strt->prox->filho.strt = ani;

            if (anit->filho.strt) {
                temp = anit->filho.strt;
                while (temp->prox != NULL) {
                    temp = temp->prox;
                }
                temp->prox = malloc(sizeof(Animais));
                memcpy(temp->prox, ftemp, sizeof(Animais));
                temp->prox->prox = NULL;
                temp->prox->pais.strt = anit;
            }
            else {
                anit->filho.strt = malloc(sizeof(Animais));
                memcpy(anit->filho.strt, ftemp, sizeof(Animais));
                anit->filho.strt->prox = NULL;
                anit->filho.strt->pais.strt = anit;
            }
        }
    }
}
```

Esta função está basicamente dividida em duas partes. A primeira é caso o animal só tenha 1 pai/mãe dentro do zoo, e a segunda é caso tenha ambos os seus pais no zoo.

Como já é normal, começo por definir algumas variáveis que vão ser úteis mais à frente e crio uma cópia do ponteiro inicial da lista para poder navegar pelos seus elementos mais facilmente.

Antes de começar a criar as ligações tenho de ter a certeza que não existem ligações anteriores, que poderiam ter sido lidas quando carregamos a informação do ficheiro binário, por isso vou definir todos os ponteiros pais e filhos a NULL.

Vou percorrer os elementos da lista 1 a 1 e vou ler o número de pais que eles têm através da variável existente na sua estrutura. Depois de determinar se tem 1 ou 2, ou nenhum, vou correr a parte do código relativa a cada uma dessa operações.

No caso de só ter 1 pai/mãe no zoo é ligeiramente mais simples. Começo por ir percorrer a lista dos animais até encontrar o animal com o ID do pai do animal e vou guardar a estrutura do pai e do filho em variáveis temporárias.

Agora que já sei quem é o pai do animal e já tenho o pai e o filho guardados em variáveis separadas da lista principal vou alocar memória no filho para guardar as informações do pai, e vou **copiar** os dados do pai para o filho, igualando o *prox dessa estrutura a NULL para informar que o animal só tem um pai, vou ainda igualar o ponteiro do filho nessa novo estrutura ao animal original. Isto começa a criar ligações entre os animais.

Depois vou ver se o pai já tem algum filho ou se este é o seu primeiro filho. No caso de ainda não ter nenhum filho vou simplesmente alocar memória para guardar os dados do filho na estrutura do pai e vou **copiar** a informação do filho para o pai, dizendo que o *prox dessa estrutura é NULL, para indicar que este animal so tem 1 filho, e aponto o ponteiro do pai nessa estrutura ao animal na lista dos animais. Criando assim ligações como podemos ver no esquema da ***página 4***.

No caso de o animal ter ambos os seus pais no zoo, a execução é semelhante, mas vai correr o código de apenas 1 pai 2 vezes para cada um dos IDs guardados na estrutura do animal.

Esta função não retorna nada, pois não faz manipulações de alto nível na lista dos animais, penas lê e altera informações já existentes, não sendo necessário estar a devolver outra vez o ponteiro que nunca se ia alterar do primeiro elemento da lista.

E estamos de volta à main onde agora vamos encontrar o ID do último animal no zoo com a função *getLastAnimalID*

```
last_animal_id = getLastAnimalID(zAnimais);
```

```
int getLastAnimalID(Animais *tAnimais) {
  Animais *temp = tAnimais;
  if (temp == NULL) return 0;
  if (temp->prox == NULL) return 0;
  while (temp->prox != NULL) {
        temp = temp->prox;
        if (temp->prox == NULL) {
          return temp->id;
        }
  }
  return 0;
}
```

Recebe o ponteiro para o início da lista dos animais e vai percorrer a lista até ao final, devolvendo o ID do último animal na lista.

Retorna 0 caso a lista não tenha nenhum animal.

Manual

Ao abrir o programa é nos apresentado um menu simples.
O menu segue o seguinte esquema:

1. **Gerir Áreas**
    - 1.1. Adicionar Área
    - 1.2. Remover área
    - 1.3. Listar áreas
    - *1.4. Voltar ao inicio*
2. **Gerir Animais**
    - 2.1. Adicionar Animal
        - 2.1.1. Adicionar animal via ficheiro de texto
        - 2.1.2. Adicionar animal via terminal
        - *2.1.3. Voltar ao inicio*
    - 2.2. Remover animal
        - 2.2.1. Remover animal pelo ID
        - 2.2.2. Remover animal pelo Nome
        - *2.2.3. Voltar ao inicio*
    - 2.3. Listagem de animais
        - 2.3.1. Listar todos os animais
        - 2.3.2. Listar todos os animais na área X
        - 2.3.3. Listar todos os animais da espécie X
        - *2.3.4. Voltar ao inicio*
    - 2.4. Info sobre animais
        - 2.4.1. Obter info do animal com o ID x

2.4.2. Obter info sobre o animal com o nome X

2.5. Transferir animal de uma área para outra

2.6. Criar um filho

2.6.1. Criar um filho a partir de 1 animal

2.6.2. Criar um filho a partir de 2 animais

2.6.3. Voltar ao início

2.7. Voltar ao início

3. Sair do programa

O programa só volta a guardar dados para os ficheiros quando a sua execução é terminada a partir do menu, caso o programa seja terminado de outra maneira todos os dados dessa sessão são perdidos.

Todos os ficheiros a serem usados no programa têm de estar na mesma pasta/localização onde o executável do programa está.