Trabalho Dirigido nº3

Objetivo Geral: Nesta ficha vamos explorar a noção de estruturas lineares dinâmicas (designadas de *listas ligadas*) de estruturas dinâmicas não lineares (*grafos*): como as definir, como as construir, como navegar nestas estruturas.

Para tal vamos desenvolver, de forma incremental, um jogo textual do tipo "*Choose Your Own Adventure*" em linguagem C, usando estruturas com ponteiros e representação de cenas como um grafo.

Um jogo textual desta natureza tem por elemento central um conjunto de cenas interligadas. Cada cena é um capítulo na aventura do jogo.

Por exemplo, veja a cena seguinte:

```
[5]
<<<
A sua aventura — tal como a sua sanidade — chegou a um cruzamento.
À direita, a floresta das Sombras Eternas sussurra segredos que não
deviam ser ouvidos. Em frente, o Cume das Vozes Perdidas aguarda,
com ecos estridentes que talvez sejam seus. À esquerda, o Deserto
dos Ossos Quebrados estala a sua esperança de qualquer salvação.
Inspire o medo. Expire a esperança. Faça a sua escolha.
>>><3>
***
+ 7. Seguir para a floresta das Sombras Eternas.
+ 9. seguir para o cume das Vozes Perdidas.
+ 3. seguir para o deserto dos Ossos Quebrados.
***
```

Neste exemplo, representamos uma cena comum numa história a decorrer. Esta cena tem um identificador (é a cena nº5), uma descrição textual, o número de opções que a cena oferece (aqui 3). A descrição de cada uma das 3 opções possíveis acompanhada da identificação da cena para qual a aventura segue em cada opção. A título de ilustração, se neste ponto do jogo escolher seguir para a floresta das Sombras Eternas, terá de introduzir o valor `7` na entrada standard para escolher esta opção e a aventura seguirá para a cena nº7.

Podemos igualmente ter cenas com um estatuto particular, como por exemplo cenas que descrevem o fim da história como:

```
[4]
<<<
Num último espasmo de coragem — ou loucura — lança-se para a
escuridão gotejante da gruta, fugindo no pânico cego da criatura
cujo bafo pútrido já aquece a sua nuca como um presságio do fim.
Mas é inútil. Ela conhece todos os seus recantos como se fosse a
luz negra da sua alma. A gruta é dela. Sempre foi. E agora, também
é sua. Garras surgem das trevas e, num único movimento bruto,
rasgam-no em dois, num terror estupefacto, sem um único grito. Só
vísceras.
Morreu.
>>><FAILED>
```

Ou ainda

```
[8]
<<<
A lâmina treme nas suas mãos — ou será o mundo que treme? Com um
grito que já não é de medo, mas de fúria ancestral, desce o golpe
final. A criatura contorce-se, guincha, parte-se em fumo e
silêncio. A escuridão recua. O ar volta aos pulmões.
Sobreviveu. Venceu. Mas terá paz?
>>><WON>
```

Estas duas cenas são cenas de fim de jogo. Uma cena onde o jogador perde o jogo (dita `FAILED`) e uma cena onde ganha o jogo (dita `WON`).

Repare bem no formato destas três cenas. Seguem o padrão textual preciso que utilizaremos para o input do problema por resolver.

Exercício 1 — Definição do tipo `Cena` e das suas operações básicas.

Assuma que o tamanho máximo de um texto é definido via uma macro `MAXTXT` com o valor `4096`.

Assuma igualmente que uma macro `MAXCHOICE` que define o número máximo de opções possíveis em cada cena. Assuma que este valor é `10`.

Assuma a existência do tipo que estabelece os estatutos possíveis para uma cena.

```
typedef enum {NORMAL, WON, FAILED} TipoCena;
```

Uma cena é assim constituída por:

- Um texto descritivo (de tamanho máximo `MAXTXT`, numa só linha).
- O seu identificador numérico.
- O seu estatuto.
- Um número de escolhas para a próxima cena. No exemplo dado acima, são 3.
- Um texto descritivo para cada opção possível para a próxima cena (de tamanho máximo `MAXTXT`, numa só linha).

Assim:

1. Defina o tipo `Cena` como uma estrutura `C` que arquiva a informação toda de uma cena.
2. Implemente uma função :

```
Cena *criaCena(char *descritivo, int id, TipoCena tipo, int
nopcoes, char **vopcoes)
```

que aloca dinamicamente uma `Cena` com `malloc`, preencha os seus campos textuais com recurso à função `strcpy` e a partir dos parâmetros passados (`descritivo`, `id`, `tipo`, `nopcoes`, `vopcoes`). A função deverá verificar que uma cena de tipo `NORMAL` tem pelo menos uma opção possível de continuação (`nopcoes`>0 e o vetor `vopcoes` contém exactamente `nopcoes` descrições textuais das opções possíveis). Deverá também verificar as cenas de tipo `WON` ou `FAILED` não têm nenhuma opção de continuação (`nopcoes` é 0). Em caso de erro, poderá usar os recursos da biblioteca `errno.h` e deverá devolver o apontador `NULL`.

3. Implemente uma função:

   ```
   void mostrarCena(Cena *c);
   ```

   que imprime na saída standard o conteúdo da cena `c`, à semelhança dos exemplos acima descritos.

   Por exemplo `+ 7. Seguir para a floresta das Sombras Eternas.`

4. Implemente a função:

   ```
   void libertaCena(Cena *c);
   ```

   que liberta a memória apontada por `c` assim como a dos seus campos.

Exercício 2 — Definição do tipo `No` e das suas operações básicas.

Neste exercício vamos modelar a noção de trama de uma história, ou seja de como as cenas se interligam para criar uma narrativa.

Para tal vamos criar a estrutura `No` que é constituído por um campo de tipo `Cena`, e um vetor que contém os apontadores para outros nós da história.

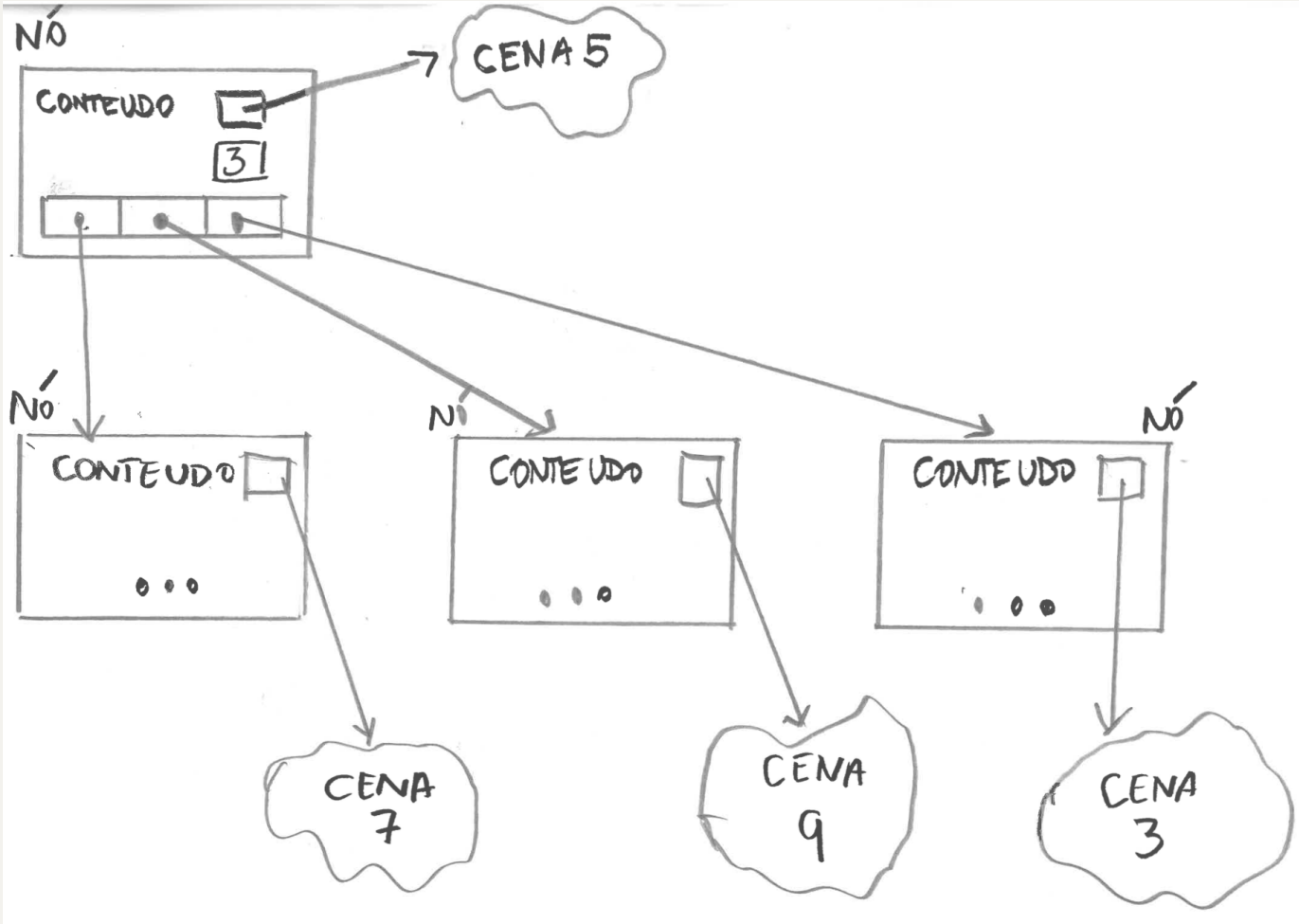
```
typedef struct no {...} No;
```

Se um valor de tipo `Cena` contém o descritivo de uma cena assim como das escolhas que a cena oferece, o conjunto de nós interliga concretamente as diferentes cenas entre si conforme a trama da história.

Um valor de tipo `No` arquiva 3 campos: um campo `conteudo` que genericamente arquiva o conteúdo do nó. Aqui neste caso concreto interessa-nos que o conteúdo seja uma cena, mas em outros usos este conteúdo poderia ser de outro tipo de dado. Para permitir uma implementação genérica (que valerá assim outros casos), preferiremos um campo de tipo `void *` que poderemos conforma a nossa vontade especializar para um tipo concreto dos dados (mediante um *cast* de tipo adequado).

Os outros dois campos tratam da interligação do conteúdo com os restantes elementos da história. O primeiro contém o numero de ligações. É um inteiro. O ultimo campo é um vetor que arquiva as ligações. Este é um vetor de tipo `struct no *` e de tamanho igual ao valor do campo anterior.

Numa imagem, podemos representar os nós desta forma:



1. Define o tipo `No`.
2. Implemente a função:

   ```
   No *criaNo(Cena *c, int nvizinhos)
   ```

   que cria um nó com o conteúdo a apontar para a cena que está no endereço `c` e que terá `nvizinhos` vizinhos. As células do vetor de vizinhos é inicializado, nesta função, com o valor `NULL`.
3. Implemente a função:

   ```
   void juntaVizinhoNo(No *no, int pos, No* v)
   ```

   que coloca a posição `pos` do vetor de vizinhos do nó `no` a apontar para `v`.
4. Implemente a função:

   ```
   No *proximoNo(No *no_ativo, int escolha)
   ```

   que devolve a referência do nó que corresponde à escolha `escolha` tendo em conta às opções possíveis propostas pela cena contida no `no_ativo`. Em caso de anomalia (como o de uma escolha inválida), o programa deverá reagir de forma adequada, garantindo a robustez da execução.
5. Implemente as funções:

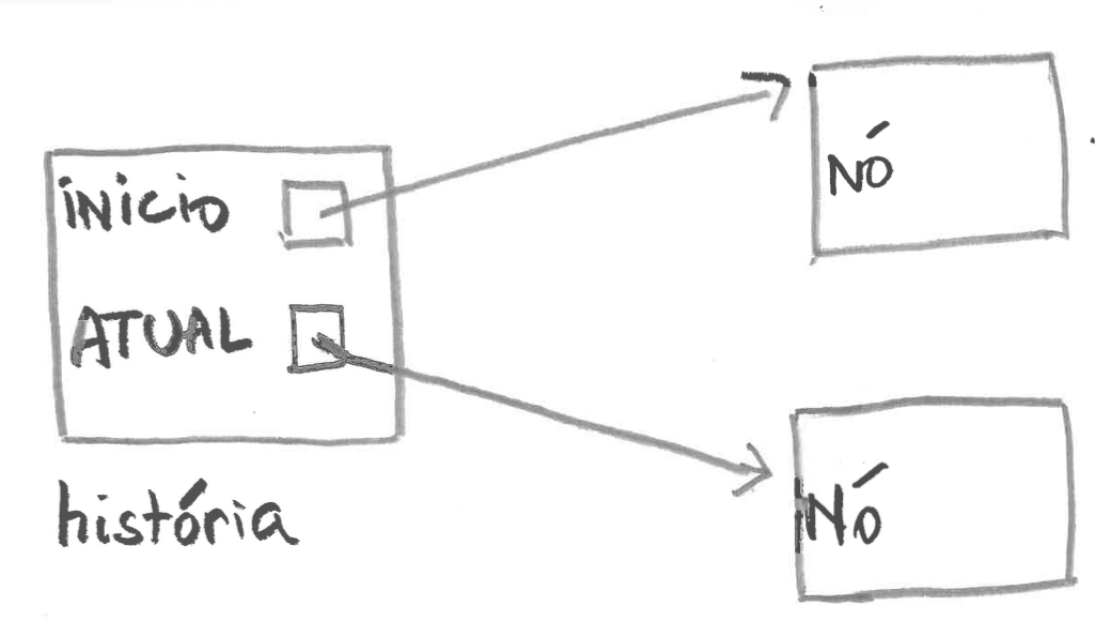
   ```
   void mostraCenaNo(No *no)
   TipoCena estadoCenaNo(No *no)
   int escolheCenaNo(No *no)
   ```

   que

   - Mostra a informação da cena (o seu descritivo, etc.) contida no nó `no`.
   - Que informa do tipo de cena contida no nó `no`.
   - Que permite ao utilizador escolher uma opção dentro das possíveis na cena contida no nó `no`.

Exercício 3 — Definição do tipo história e das suas funções

Considere o tipo estrutural `Historia` que tem dois campos. O primeiro campo é um apontador para o nó que arquiva a cena inicial da história. O segundo campo é um apontador para o nó que contem a cena atualmente ativa da história a decorrer.



1. Define o tipo `Historia`.
2. Define a função:

   ```
   Historia *criaHistoria(No *no)
   ```

   que cria uma história a partir do nó inicial `no`. Nesta situação ambos os campos apontam para o `no`.

Exercício 4 — Carregar uma história completa

Considere que o formato de input é o seguinte:

Na primeira linha, encontramos o inteiro n que indica quantas cenas a história tem.

As cenas são numeradas de 0 até $n-1$ e são dadas textualmente nesta ordem seguindo o formato dos exemplos acima descritos. A cena que inicia a história é sempre a cena 0.

Segue, depois, linhas com um inteiro (de 0 a n-1)

1. Leia todas as cenas e guarde-as num array ou dynvec de ponteiros para valores de tipo `No`.
2. Após a leitura dos nós e das respectivas cenas no vetor, crie o grafo da história (cada nó aponta adequadamente para os restantes nós vizinhos) e a estrutura de tipo `Historia`.
3. Desenrole a história com base nas opções escolhidas dadas no input.

   Vamos nesta fase ignorar a impressão do texto das cenas e das escolhas.

   Cada execução tem 3 desfechos possíveis.

   a. O jogo termina com a vitória. Deverá escrever numa linha `WON`.

   b. O jogo termina com uma derrota. Deverá escrever numa linha `FAILED`.

   c. Não deu opções em quantidade suficiente para terminar o jogo ou apareceu uma opção inválida no decorrer do jogo. Deverá escrever numa linha `WAITING`.

Não se esqueça de libertar toda a memória alocada no fim do jogo.

Exemplo de Input

```
4
[0]
<<<
A sua aventura — tal como a sua sanidade — chegou a um cruzamento.
À direita, a floresta das Sombras Eternas sussurra segredos que não
deviam ser ouvidos. À esquerda, o Deserto dos Ossos Quebrados
estala a sua esperança de qualquer salvação. Inspire o medo. Expire
a esperança. Faça a sua escolha.
```

```
>>><2>
***
+ 1. Seguir para a floresta das Sombras Eternas.
+ 2. Seguir para o deserto dos Ossos Quebrados.
***
[1]
<<<
Atravessa o limiar da floresta, onde a luz hesita e os sussurros não têm dono. As árvores parecem mover-se, como se respirassem com a noite. Algo observa. Algo sempre observou. Cada passo é engolido pelo musgo e pela memória de quem nunca regressou. Mas agora é tarde para recuar — ou será que não?
>>><3>
***
+ 0. Voltar para trás.
+ 2. Seguir para o deserto dos Ossos Quebrados.
+ 3. Seguir em frente.
***
[2]
<<<
Num último espasmo de coragem — ou loucura — lança-se para a escuridão gotejante da gruta, fugindo no pânico cego da criatura cujo bafo pútrido já aquece a sua nuca como um presságio do fim. Mas é inútil. Ela conhece todos os seus recantos como se fosse a luz negra da sua alma. A gruta é dela. Sempre foi. E agora, também é sua. Garras surgem das trevas e, num único movimento bruto, rasgam-no em dois, num terror estupefacto, sem um único grito. Só vísceras.
Morreu.
>>><FAILED>
[3]
<<<
A lâmina treme nas suas mãos — ou será o mundo que treme? Com um grito que já não é de medo, mas de fúria ancestral, desce o golpe final. A criatura contorce-se, guincha, parte-se em fumo e silêncio. A escuridão recua. O ar volta aos pulmões.
Sobreviveu. Venceu. Mas terá paz?
>>><WON>
1
```

```
0
1
0
2
```

Neste caso particular o jogador perde, e a mensagem `FAILED` aparece.

No caso de vitória, a mensagem `WON` é mostrada. Nos casos restantes, a mensagem `WAITING` é apresentada.