P1 - Estrutura de Dados

Pedro Mian Parra - 2207249

Exercício 1:

Para definirmos uma estrutura de dados adequada, observamos as principais características de uma ação para armazenarmos suas informações.

Nas pilhas, teremos uma estrutura de dados em que o acesso é restrito ao elemento mais recente na pilha: "Último elemento a entrar é o primeiro a sair". Em sua implementação estática, utilizamos uma variante que reservará o espaço na memória de maneira estática, tendo um tamanho máximo definido. Já na implementação dinâmica, a reserva de memória é feita através da alocação dinâmica de um ponteiro, que aponta para o próximo elemento da pilha.

Em filas, armazenamos dados de forma que quem está há mais tempo na estrutura é o primeiro a ser retirado: "Primeiro elemento a entrar é o primeiro a sair". Diferente das pilhas, nas filas manipulamos as duas extremidades do conjunto de elementos, inserindo novos no final dela e retirando no começo. Na implementação estática temos, assim como nas pilhas estáticas, um tamanho máximo definido podendo ocupar espaço desnecessário; na fila dinâmica temos a vantagem de ocupar o espaço estritamente necessário, porém tendo um tempo de execução maior para a realização da alocação.

Com listas, é necessário trabalhar com conjuntos de elementos, objetos, variáveis, tarefas, ou qualquer outra coisa que se possa enumerar e formar um conjunto ordenado. Há três tipos de implementação de listas, sendo eles:
-Single-linkage (singularmente encadeada): onde cada elemento possui o dado armazenado e a ligação com o elemento seguinte. O último elemento aponta para um valor nulo.
-Double-linkage (duplamente encadeada): nela a conexão entre os elementos é feita através de dois ponteiros, um que aponta para o elemento anterior e o outro para o elemento seguinte. Neste tipo de implementação, a lista pode ser acessada por ambos os lados.
-Circulares (nó sentinela): não possui fim, o ponteiro do último elemento aponta para o primeiro elemento da lista, em vez de null como nos outros tipos de implementação.

Exercício 2:

```
Pilha A;
Pilha B;
Pilha C;
iniciaPilha(&A);
iniciaPilha(&B);
iniciaPilha(&C);
```

A = {5, 4, 3, 2, 1} B= { } C= { }

x = desempilha(A);
empilha(B,x);
A = {5, 4, 3, 2} B= {1 } C= { }

x = desempilha(A);
empilha(C,x);
A = {5, 4, 3 } B= {1 } C= {2 }

x = desempilha(B);
empilha(C,x);
A = {5, 4, 3 } B= { } C= {2, 1 }

x = desempilha(A);
empilha(B,x);
A = {5, 4 } B= {3 } C= {2, 1 }

x = desempilha(C);
empilha(A,x);
A = {5, 4, 1} B= {3 } C= {2 }

x = desempilha(C);
empilha(B,x);
A = {5, 4, 1} B= {3, 2 } C= { }

x = desempilha(A);
empilha(B,x);
A = {5, 4 } B= {3, 2, 1 } C= { }

x = desempilha(A);
empilha(C,x);
A = {5 } B= {3, 2, 1 } C= {4 }

x = desempilha(B);
empilha(C,x);
A = {5 } B= {3, 2 } C= {4, 1 }

x = desempilha(B);
empilha(A,x);
A = {5, 2 } B= {3 } C= {4, 1 }

x = desempilha(C);
empilha(A,x);
A = {5, 2, 1 } B= {3 } C= {4 }

x = desempilha(B);
empilha(C,x);
A = {5, 2, 1 } B= { } C= {4, 3 }

x = desempilha(A);
empilha(B,x);
A = {5, 2 } B= {1 } C= {4, 3 }

x = desempilha(A);
empilha(C,x);
A = {5, } B= {1 } C= {4, 3, 2 }

x = desempilha(B);
empilha(C,x);
A = {5 } B= { } C= {4, 3, 2, 1 }

x = desempilha(A);
empilha(B,x);
A = { } B= {5 } C= {4, 3, 2, 1 }

x = desempilha(C);
empilha(A,x);
A = {1 } B= {5 } C= {4, 3, 2 }

x = desempilha(C);
empilha(B,x);
A = {1 } B= {5, 2 } C= {4, 3 }

x = desempilha(A);
empilha(B,x);
A = { } B= {5, 2, 1 } C= {4, 3 }

x = desempilha(C);
empilha(A,x);
A = {3 } B= {5, 2, 1 } C= {4 }

x = desempilha(B);
empilha(C,x);
A = {3 } B= {5, 2 } C= {4, 1 }

x = desempilha(B);
empilha(A,x);
A = {3, 2 } B= {5 } C= {4, 1 }

x = desempilha(C);
empilha(A,x);
A = {3, 2, 1 } B= {5 } C= {4 }

x = desempilha(C);
empilha(B,x);
A = {3, 2, 1 } B= {5, 4 } C= { }

x = desempilha(A);
empilha(B,x);
A = {3, 2 } B= {5, 4, 1 } C= { }

x = desempilha(A);
empilha(C,x);
A = {3 } B= {5, 4, 1 } C= {2 }

x = desempilha(B);
empilha(&C,x);
A = {3 } B= {5, 4 } C= {2, 1 }

x = desempilha(&A);
empilha(B,x);
A = { } B= {5, 4, 3 } C= {2, 1 }

x = desempilha(C);
empilha(A,x);
A = {1 } B= {5, 4, 3 } C= {2 }

x = desempilha(C);
empilha(B,x);
A = {1 } B= {5, 4, 3, 2 } C= { }

x = desempilha(A);
empilha(B,x);
A = { } B= {5, 4, 3, 2, 1 } C= { }

Exercício 3:

1: VERMELHO inserido, Fila com 1 elemento
Fila = [VERMELHO]

2: VERDE inserido, Fila com 2 elementos
Fila = [VERMELHO, VERDE]

3: AMARELO inserido, Fila com 3 elementos
Fila = [VERMELHO, VERDE, AMARELO]

4: BRANCO não inserido: Fila cheia!
Fila = [VERMELHO, VERDE, AMARELO]

5: PRETO não inserido: Fila cheia!
Fila = [VERMELHO, VERDE, AMARELO]

6: VERMELHO removido, Fila com 2 elementos
Fila = [VERDE, AMARELO]

7: VERDE removido, Fila com 1 elemento
Fila = [AMARELO]

8: ROSA inserido, Fila com 2 elementos
Fila = [AMARELO, ROSA]

9: AMARELO removido, Fila com 1 elemento
Fila = [ROSA]

10: ROSA removido, Fila vazia
Fila = [ ]

11: AZUL inserido, Fila com 1 elemento
Fila = [AZUL]

12: CINZA inserido, Fila com 2 elementos
Fila = [AZUL, CINZA]

13: AZUL removido, Fila com 1 elemento
Fila = [CINZA]

14: CINZA removido, Fila vazia
Fila = [ ]

Exercício 4:

```
void inverteFilas(FilaDinamica *F1, FilaDinamica *F2) {
  int eleF1, eleF2;
  eleF1 = tamanhoFila(F1);
  eleF2 = tamanhoFila(F2);

imprimeFila(F1);
```

```
imprimeFila(F2);
  //passando os elementos de uma fila para outra
  /* desenfileirar a fila 2 de acordo com o seu tamanho */
  for (int i = 0; i < eleF2; i++) {
    enfileira(F1, desenfileira(F2));
  }

  /* desenfileirar a fila 1 de acordo com o seu tamanho, sobrando os
elementos da fila 2 nela */
  for (int i = 0; i < eleF1; i++) {
    enfileira(F2, desenfileira(F1));
  }

imprimeFila(F1);
imprimeFila(F2);

}
```

Exercício 5:

```
Lista* constroi(int n, int* v){
  Lista* lista;
  int x;

  iniciaListaDinamica(lista);

  for (int i = 0; i < n; i++) {
    inserirListaDinamica(lista, v[i]);
  }

  imprimirListaDinamica(lista);
  return(lista);
}
```

Exercício 6:

a)

Seria necessário criar uma struct com as informações individuais de uma carta, sendo:

```
typedef struct {
  char naipe;
  char valor; // 9, 10, J, Q, K, A
  char cor;
} Carta;
```

```
typedef struct {
  Carta array[52];
} Baralho;
```

A melhor estratégia seria utilizar a implementação de pilhas dinâmicas para construirmos as 16 pilhas necessárias dentro do jogo, tanto as pilhas definitivas como as intermediárias. Desta maneira, pilhas que estiverem vazias, por exemplo, não ocupariam espaço na memória, dependendo da quantidade de cartas na pilha será ocupado apenas o espaço necessário na memória.

Para a implementação das pilhas, precisamos das seguintes estruturas:

```
typedef struct NoPilha* PtrNoPilha;

typedef struct NoPilha{
  Objeto obj;
  PtrNoPilha proximo;
} NoPilha;

typedef struct {
  PtrNoPilha topo;
  int tamanho;
} PilhaDinamica;
```

b)

EM BRANCO