Fila (implementação por meio de array e lista encadeada)

1. Fila

A fila é uma estrutura de dados que segue a seguinte regra: O primeiro elemento inserido é o primeiro elemento removido. A sigla FIFO (first in, first out) é utilizada para descrever essa estratégia. Isso significa que um novo elemento só pode ser inserido no final da fila e o primeiro elemento inserido na fila será o primeiro a ser removido.

Exemplo de aplicação: A implementação de uma fila para organizar o processo de impressão de uma impressora compartilhada por vários computadores. A estratégia mais simples é considerar todas as requisições com a mesma prioridade e imprimir os documentos na ordem em que foram submetidos.

Existem duas funções principais que são implementadas em uma fila:

- Enqueue: Adiciona um elemento ao final da fila.
- Dequeue: Remove o elemento do início da fila.

A utilização de array para a implementação é menos complexa, possibilita o acesso direto por meio de índices e ocupa um menor espaço de memória devido à alocação contígua. Entretanto, o tamanho fixo da fila em tempo de execução pode causar desperdício de memória (fila parcialmente vazia) ou não permitir a inserção de novos elementos (fila cheia).

A utilização de listas encadeadas para a implementação evita desperdício de memória pelo fato da lista possuir tamanho dinâmico, permite a inserção e remoção de elementos sem a necessidade de realocação dos demais e facilita o manuseio de elementos com tipos e tamanhos diferentes. Contudo, o acesso a um elemento exige que a fila seja percorrida sequencialmente a partir de seu início e existe a necessidade de espaço adicional em cada elemento para armazenar o ponteiro para o próximo elemento.

Exemplo 1: Implementação de uma fila por meio de array

```
#include <stdio.h>
#include <stdbool.h>

#define TAMANHO_FILA 10

typedef struct queue Queue;

struct queue {
  int itens[TAMANHO_FILA];
  int inicio;
  int fim;
};

// Inicializa a fila
void inicializarFila(Queue *fila) {
  fila->inicio = -1;
```

```
  fila->fim = -1;
}

// Verifica se a fila está vazia
bool estaVazia(Queue *fila) {
  return (fila->inicio == -1 && fila->fim == -1);
}

// Verifica se a fila está cheia
bool estaCheia(Queue *fila) {
  return (fila->fim == TAMANHO_FILA - 1);
}

// Adiciona um elemento no final da fila (enqueue)
void enqueue(Queue *fila, int item) {
  if (estaCheia(fila)) {
    printf("Não é possível adicionar mais elementos. A fila está cheia! \n");
  } else {
    if (estaVazia(fila)) {
      fila->inicio = 0;
    }
    fila->fim++;
    fila->itens[fila->fim] = item;
  }
}

// Remove um elemento do início da fila (dequeue)
int dequeue(Queue *fila) {
  int itemRemovido;
  if (estaVazia(fila)) {
    printf("Não é possível remover elementos. A fila está vazia! \n");
    return -1; // Retorna um valor sentinela para indicar erro
  } else {
    itemRemovido = fila->itens[fila->inicio];
    if (fila->inicio == fila->fim) {
      // Se há apenas um elemento na fila, ela é reinicializada após a remoção do elemento
      inicializarFila(fila);
    } else {
      fila->inicio++;
    }
    return itemRemovido;
  }
}
```

```
// Busca um elemento na fila
bool buscarElemento(Queue *fila, int elemento) {
  for (int i = fila->inicio; i <= fila->fim; i++) {
    if (fila->itens[i] == elemento) {
      return true;
    }
  }
  return false;
}

int main() {
  Queue minhaFila;
  inicializarFila(&minhaFila);

  // Inserção de 2 números
  enqueue(&minhaFila, 1);
  enqueue(&minhaFila, 2);

  // Remoção de 2 números
  int itemRemovido = dequeue(&minhaFila);
  printf("Elemento removido: %d\n", itemRemovido);
  itemRemovido = dequeue(&minhaFila);
  printf("Elemento removido: %d\n", itemRemovido);

  // Inserção de 2 números
  enqueue(&minhaFila, 3);
  enqueue(&minhaFila, 4);

  // Busca do número 3
  int elementoBuscado = 3;
  if (buscarElemento(&minhaFila, elementoBuscado)) {
    printf("Elemento %d encontrado! \n", elementoBuscado);
  } else {
    printf("Elemento %d NÃO encontrado! \n", elementoBuscado);
  }

  return 0;
}
```

Exemplo 2: Implementação de uma fila por meio de lista encadeada

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <stdbool.h>

typedef struct lista Lista;
typedef struct queue Queue;

// Definição da estrutura de um nó da lista encadeada
struct lista {
  int nro;
  struct lista *prox;
};

// Definição do Tipo Abstrato de Dados (TAD) Fila (Queue)
struct queue {
  Lista *inicio;
  Lista *fim;
};

// Inicializa a fila
void inicializarFila(Queue *fila) {
  fila->inicio = NULL;
  fila->fim = NULL;
}

// Verifica se a fila está vazia
bool estaVazia(Queue *fila) {
  return (fila->inicio == NULL);
}

// Adiciona um elemento à fila (enqueue)
void enqueue(Queue *fila, int item) {
  Lista *novoNo = malloc(sizeof(Lista));
  novoNo->nro = item;
  novoNo->prox = NULL;

  if (estaVazia(fila)) {
    fila->inicio = novoNo;
    fila->fim = novoNo;
  } else {
    fila->fim->prox = novoNo;
    fila->fim = novoNo;
  }
}
```

```c
// Remove um elemento da fila (dequeue)
int dequeue(Queue *fila) {
  if (estaVazia(fila)) {
    printf("Não é possível remover elementos. Fila vazia! \n");
    return -1; // Retorna um valor sentinela para indicar erro
  } else {
    Lista *nodoRemovido = fila->inicio;
    int itemRemovido = nodoRemovido->nro;

    fila->inicio = fila->inicio->prox;
    free(nodoRemovido);

    return itemRemovido;
  }
}

// Busca um elemento na fila
bool buscarElemento(Queue *fila, int elemento) {
  Lista *noAtual = fila->inicio;
  while (noAtual != NULL) {
    if (noAtual->nro == elemento) {
      return true;
    }
      noAtual = noAtual->prox;
  }
  return false;
}

// Libera a memória alocada para a fila
void liberarFila(Queue * fila) {
  while (!estaVazia(fila)) {
    dequeue(fila);
  }
}

int main() {
  Queue minhaFila;
  inicializarFila(&minhaFila);

  // Inserção de 4 números
  enqueue(&minhaFila, 1);
  enqueue(&minhaFila, 2);
  enqueue(&minhaFila, 3);
```

```
    enqueue(&minhaFila, 4);

    // Remoção de 1 número
    int itemRemovido = dequeue(&minhaFila);
    printf("Elemento removido: %d\n", itemRemovido);

    // Busca do número 3
    int elementoBuscado = 3;
    if (buscarElemento(&minhaFila, elementoBuscado)) {
      printf("Elemento %d encontrado na fila.\n", elementoBuscado);
    } else {
      printf("Elemento %d NÃO encontrado na fila.\n", elementoBuscado);
    }

    // Libera a memória alocada para a fila
    liberarFila(&minhaFila);

    return 0;
}
```

2. Exercícios

2.1. Inclua uma função para imprimir os elementos dos Exemplos 1 e 2.

2.2. Crie um programa para organizar o acesso a um caixa eletrônico. O usuário deve aguardar na fila até o momento de seu atendimento. Ao ser atendido, ele deve informar o valor do saque. O nome do usuário e o valor definido para saque devem ser impressos.

Crie uma versão implementada com array e outra com lista encadeada

2.3. Crie um programa para o gerenciamento das atrações em um parque de diversões. Cada atração do parque possui uma fila para organizar o acesso dos visitantes. Ao escolher uma atração, o visitante é incluído na fila. Ao entrar na atração, ele é removido da fila.

Crie uma versão implementada com array e outra com lista encadeada

2.4. Elabore um programa para gerenciar a fila de impressão em um escritório. Implemente funções para a adição e remoção de documentos da fila. Ao retirar um documento da fila, o usuário associado e a descrição do documento devem ser impressos.

Crie uma versão implementada com array e outra com lista encadeada

2.5. O que é uma fila de prioridade? Em quais situações devemos utilizá-la? Crie um programa que implemente o conceito de fila de prioridade.

A utilização de array ou lista encadeada permite a implementação? Se sim, qual das opções é menos complexa?