Filas

Outra estrutura de dados bastante usada em computação é a *fila*. Na estrutura de fila, os acessos aos elementos também seguem uma regra. O que diferencia a *fila* da *pilha* é a ordem de saída dos elementos: enquanto na pilha "o último que entra é o primeiro que sai", na fila "o primeiro que entra é o primeiro que sai" (a sigla FIFO – *first in, first out* – é usada para descrever essa estratégia). A idéia fundamental da fila é que só podemos inserir um novo elemento no final da fila e só podemos retirar o elemento do início.

A estrutura de fila é uma analogia natural com o conceito de fila que usamos no nosso dia a dia: quem primeiro entra numa fila é o primeiro a ser atendido (a sair da fila). Um exemplo de utilização em computação é a implementação de uma fila de impressão. Se uma impressora é compartilhada por várias máquinas, deve-se adotar uma estratégia para determinar que documento será impresso primeiro. A estratégia mais simples é tratar todas as requisições com a mesma prioridade e imprimir os documentos na ordem em que foram submetidos – o primeiro submetido é o primeiro a ser impresso.

De modo análogo ao que fizemos com a estrutura de pilha, neste capítulo discutiremos duas estratégias para a implementação de uma estrutura de fila: usando vetor e usando lista encadeada. Para implementar uma fila, devemos ser capazes de inserir novos elementos em uma extremidade, o *fim*, e retirar elementos da outra extremidade, o *início*.

12.1. Interface do tipo fila

Antes de discutirmos as duas estratégias de implementação, podemos definir a interface disponibilizada pela estrutura, isto é, definir quais operações serão implementadas para manipular a fila. Mais uma vez, para simplificar a exposição, consideraremos uma estrutura que armazena valores reais. Independente da estratégia de implementação, a interface do tipo abstrato que representa uma estrutura de fila pode ser composta pelas seguintes operações:

- criar uma estrutura de fila;
- inserir um elemento no fim;
- retirar o elemento do início;
- verificar se a fila está vazia;
- liberar a fila.

O arquivo `fila.h`, que representa a interface do tipo, pode conter o seguinte código:

```
typedef struct fila Fila;

Fila* cria (void);
void insere (Fila* f, float v);
float retira (Fila* f);
int vazia (Fila* f);
void libera (Fila* f);
```

A função `cria` aloca dinamicamente a estrutura da fila, inicializa seus campos e retorna seu ponteiro; a função `insere` adiciona um novo elemento no final da fila e a função

`retira` remove o elemento do início; a função `vazia` informa se a fila está ou não vazia; e a função `libera` destrói a estrutura, liberando toda a memória alocada.

12.2. Implementação de fila com vetor

Como no caso da pilha, nossa primeira implementação de fila será feita usando um vetor para armazenar os elementos. Para isso, devemos fixar o número máximo `N` de elementos na fila. Podemos observar que o processo de inserção e remoção em extremidades opostas fará com que a fila "ande" no vetor. Por exemplo, se inserirmos os elementos `1.4`, `2.2`, `3.5`, `4.0` e depois retirarmos dois elementos, a fila não estará mais nas posições iniciais do vetor. A Figura 11.1 ilustra a configuração da fila após a inserção dos primeiros quatro elementos e a Figura 11.2 após a remoção de dois elementos.

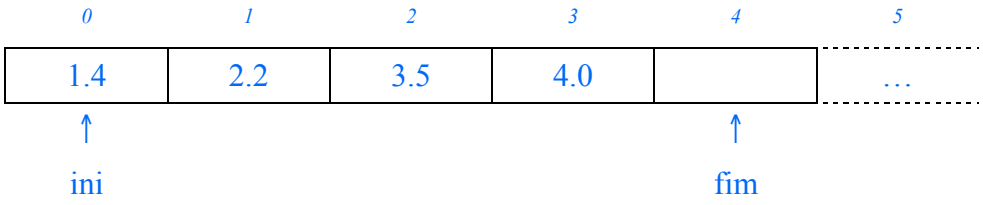

Figura 11.1: Fila após inserção de quatro novos elementos.

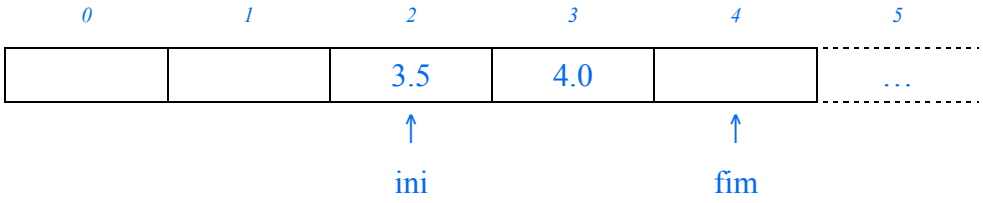

Figura 11.2: Fila após retirar dois elementos.

Com essa estratégia, é fácil observar que, em um dado instante, a parte ocupada do vetor pode chegar à última posição. Para reaproveitar as primeiras posições livres do vetor sem implementarmos uma re-arrumação trabalhosa dos elementos, podemos incrementar as posições do vetor de forma "circular": se o último elemento da fila ocupa a última posição do vetor, inserimos os novos elementos a partir do início do vetor. Desta forma, em um dado momento, poderíamos ter quatro elementos, `20.0`, `20.8`, `21.2` e `24.3`, distribuídos dois no fim do vetor e dois no início.

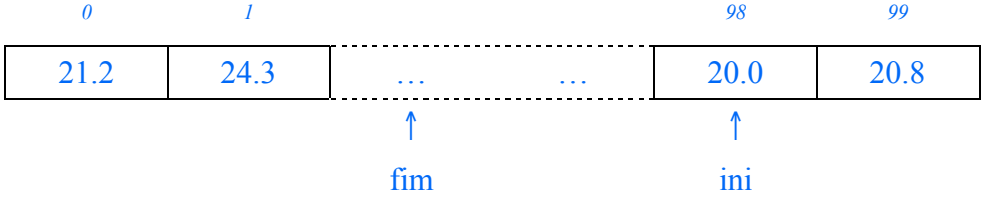

Figura 11.3: Fila com incremento circular.

Para essa implementação, os índices do vetor são incrementados de maneira que seus valores progridam "circularmente". Desta forma, se temos 100 posições no vetor, os valores dos índices assumem os seguintes valores:

```
0, 1, 2, 3, …, 98, 99, 0, 1, 2, 3, …, 98, 99, 0, 1,…
```

Podemos definir uma função auxiliar responsável por incrementar o valor de um índice. Essa função recebe o valor do índice atual e fornece com valor de retorno o índice incrementado, usando o incremento circular. Uma possível implementação dessa função é:

```
int incr (int i)
{
   if (i == N-1)
      return 0;
   else
      return i+1;
}
```

Essa mesma função pode ser implementada de uma forma mais compacta, usando o operador módulo:

```
int incr(int i)
{
   return (i+1)%N;
}
```

Com o uso do operador módulo, muitas vezes optamos inclusive por dispensar a função auxiliar e escrever diretamente o incremento circular:

```
...
i=(i+1)%N;
...
```

Podemos declarar o tipo fila como sendo uma estrutura com três componentes: um vetor `vet` de tamanho `N`, um índice `ini` para o início da fila e um índice `fim` para o fim da fila.

Conforme ilustrado nas figuras acima, usamos as seguintes convenções para a identificação da fila:

- `ini` marca a posição do próximo elemento a ser retirado da fila;
- `fim` marca a posição (vazia), onde será inserido o próximo elemento.

Desta forma, a fila vazia se caracteriza por ter `ini == fim` e a fila cheia (quando não é possível inserir mais elementos) se caracteriza por ter `fim` e `ini` em posições consecutivas (circularmente): `incr(fim) == ini`. Note que, com essas convenções, a posição indicada por `fim` permanece sempre vazia, de forma que o número máximo de elementos na fila é `N-1`. Isto é necessário porque a inserção de mais um elemento faria `ini == fim`, e haveria uma ambigüidade entre fila cheia e fila vazia. Outra estratégia possível consiste em armazenar uma informação adicional, `n`, que indicaria explicitamente o número de elementos armazenados na fila. Assim, a fila estaria vazia se `n == 0` e cheia se `n == N-1`. Nos exemplos que se seguem, optamos por não armazenar `n` explicitamente.

A estrutura de fila pode então ser dada por:

```
#define N 100

struct fila {
   int ini, fim;
   float vet[N];
};
```

A função para criar a fila aloca dinamicamente essa estrutura e inicializa a fila como sendo vazia, isto é, com os índices `ini` e `fim` iguais entre si (no caso, usamos o valor zero).

```
Fila* cria (void)
{
   Fila* f = (Fila*) malloc(sizeof(Fila));
   f->ini = f->fim = 0;   /* inicializa fila vazia */
   return f;
}
```

Para inserir um elemento na fila, usamos a próxima posição livre do vetor, indicada por `fim`. Devemos ainda assegurar que há espaço para a inserção do novo elemento, tendo em vista que trata-se de um vetor com capacidade limitada. Consideraremos que a função auxiliar que faz o incremento circular está disponível.

```
void insere (Fila* f, float v)
{
    if (incr(f->fim) == f->ini) {    /* fila cheia: capacidade esgotada
*/
       printf("Capacidade da fila estourou.\n");
       exit(1);         /* aborta programa */
   }
   /* insere elemento na próxima posição livre */
   f->vet[f->fim] = v;
   f->fim = incr(f->fim);
}
```

A função para retirar o elemento do início da fila fornece o valor do elemento retirado como retorno. Podemos também verificar se a fila está ou não vazia.

```
float retira (Fila* f)
{
   float v;
   if (vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);         /* aborta programa */
   }
   /* retira elemento do início */
   v = f->vet[f->ini];
   f->ini = incr(f->ini);
   return v;
}
```

A função que verifica se a fila está vazia pode ser dada por:

```
int vazia (Fila* f)
{
   return (f->ini == f->fim);
}
```

Finalmente, a função para liberar a memória alocada pela fila pode ser:

```
void libera (Fila* f)
{
   free(f);
}
```

12.3. Implementação de fila com lista

Vamos agora ver como implementar uma fila através de uma lista encadeada, que será, como nos exemplos anteriores, uma lista simplesmente encadeada, em que cada nó guarda um ponteiro para o próximo nó da lista. Como teremos que inserir e retirar elementos das extremidades opostas da lista, que representarão o início e o fim da fila, teremos que usar dois ponteiros, `ini` e `fim`, que apontam respectivamente para o primeiro e para o último elemento da fila. Essa situação é ilustrada na figura abaixo:

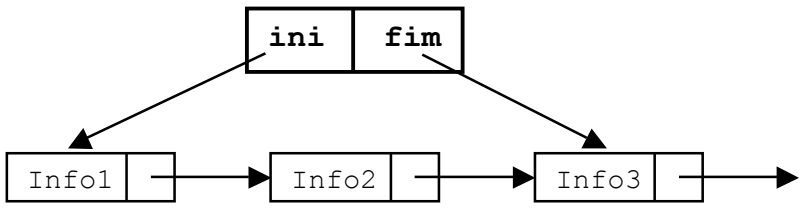

Figura 11.4: Estrutura de fila com lista encadeada.

A operação para retirar um elemento se dá no início da lista (fila) e consiste essencialmente em fazer com que, após a remoção, `ini` aponte para o sucessor do nó retirado. (Observe que seria mais complicado remover um nó do fim da lista, porque o antecessor de um nó não é encontrado com a mesma facilidade que seu sucessor.) A inserção também é simples, pois basta acrescentar à lista um sucessor para o último nó, apontado por `fim`, e fazer com que `fim` aponte para este novo nó.

O nó da lista para armazenar valores reais, como já vimos, pode ser dado por:

```
struct no {
   float info;
   struct no* prox;
};
typedef struct no No;
```

A estrutura da fila agrupa os ponteiros para o início e o fim da lista:

```
struct fila {
   No* ini;
   No* fim;
};
```

A função `cria` aloca a estrutura da fila e inicializa a lista como sendo vazia.

```
Fila* cria (void)
{
   Fila* f = (Fila*) malloc(sizeof(Fila));
   f->ini = f->fim = NULL;
   return f;
}
```

Cada novo elemento é inserido no fim da lista e, sempre que solicitado, retiramos o elemento do início da lista. Desta forma, precisamos de duas funções auxiliares de lista:

para inserir no fim e para remover do início. A função para inserir no fim ainda não foi discutida, mas é simples, uma vez que temos explicitamente armazenado o ponteiro para o último elemento. Essa função deve ter como valor de retorno o novo "fim" da lista. A função para retirar do início é idêntica à função usada na implementação de pilha.

```
/* função auxiliar: insere no fim */
No* ins_fim (No* fim, float v)
{
   No* p = (No*) malloc(sizeof(No));
   p->info = v;
   p->prox = NULL;
   if (fim != NULL)  /* verifica se lista não estava vazia */
      fim->prox = p;
   return p;
}


/* função auxiliar: retira do início */
No* ret_ini (No* ini)
{
   No* p = ini->prox;
   free(ini);
   return p;
}
```

As funções que manipulam a fila fazem uso dessas funções de lista. Devemos salientar que a função de inserção deve atualizar ambos os ponteiros, `ini` e `fim`, quando da inserção do primeiro elemento. Analogamente, a função para retirar deve atualizar ambos se a fila tornar-se vazia após a remoção do elemento:

```
void insere (Fila* f, float v)
{
   f->fim = ins_fim(f->fim,v);
   if (f->ini==NULL)  /* fila antes vazia? */
      f->ini = f->fim;
}

float retira (Fila* f)
{
   float v;
   if (vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);         /* aborta programa */
   }
   v = f->ini->info;
   f->ini = ret_ini(f->ini);
   if (f->ini == NULL)   /* fila ficou vazia? */
      f->fim = NULL;
   return v;
}
```

A fila estará vazia se a lista estiver vazia:

```
int vazia (Fila* f)
{
   return (f->ini==NULL);
}
```

Por fim, a função que libera a fila deve antes liberar todos os elementos da lista.

```
void libera (Fila* f)
{
   No* q = f->ini;
   while (q!=NULL) {
      No* t = q->prox;
      free(q);
      q = t;
   }
   free(f);
}
```

Analogamente à pilha, para testar o código, pode ser útil implementarmos uma função que imprima os valores armazenados na fila. Os códigos abaixo ilustram a implementação dessa função nas duas versões de fila (vetor e lista). A ordem de impressão adotada é do início para o fim.

```
/* imprime: versão com vetor */
void imprime (Fila* f)
{
   int i;
   for (i=f->ini; i!=f->fim; i=incr(i))
      printf("%f\n",f->vet[i]);
}

/* imprime: versão com lista */
void imprime (Fila* f)
{
   No* q;
   for (q=f->ini; q!=NULL; q=q->prox)
      printf("%f\n",q->info);
}
```

Um exemplo simples de utilização da estrutura de fila é apresentado a seguir:

```
/* Módulo para ilustrar utilização da fila */

#include <stdio.h>
#include "fila.h"

int main (void)
{
   Fila* f = cria();
   insere(f,20.0);
   insere(f,20.8);
   insere(f,21.2);
   insere(f,24.3);
   printf("Primeiro elemento: %f\n", retira(f));
   printf("Segundo elemento: %f\n", retira(f));
   printf("Configuracao da fila:\n");
   imprime(f);
   libera(f);
   return 0;
}
```

12.4. Fila dupla

A estrutura de dados que chamamos de *fila dupla* consiste numa fila na qual é possível inserir novos elementos em ambas as extremidades, no início e no fim. Conseqüentemente, permite-se também retirar elementos de ambos os extremos. É como se, dentro de uma mesma estrutura de fila, tivéssemos duas filas, com os elementos dispostos em ordem inversa uma da outra.

A interface do tipo abstrato que representa uma fila dupla acrescenta novas funções para inserir e retirar elementos. Podemos enumerar as seguintes operações:

- criar uma estrutura de fila dupla;
- inserir um elemento no início;
- inserir um elemento no fim;
- retirar o elemento do início;
- retirar o elemento do fim;
- verificar se a fila está vazia;
- liberar a fila.

O arquivo `fila2.h`, que representa a interface do tipo, pode conter o seguinte código:

```
typedef struct fila2 Fila2;

Fila2* cria (void);
void insere_ini (Fila2* f, float v);
void insere_fim (Fila2* f, float v);
float retira_ini (Fila2* f);
float retira_fim (Fila2* f);
int vazia (Fila2* f);
void libera (Fila2* f);
```

A implementação dessa estrutura usando um vetor para armazenar os elementos não traz grandes dificuldades, pois o vetor permite acesso randômico aos elementos, e fica como exercício.

Exercício: Implementar a estrutura de fila dupla com vetor.
Obs: Note que o decremento circular não pode ser feito de maneira compacta como fizemos para incrementar. Devemos decrementar o índice de uma unidade e testar se ficou negativo, atribuindo-lhe o valor `N-1` em caso afirmativo.

12.5. Implementação de fila dupla com lista

A implementação de uma fila dupla com lista encadeada merece uma discussão mais detalhada. A dificuldade que encontramos reside na implementação da função para retirar um elemento do final da lista. Todas as outras funções já foram discutidas e poderiam ser facilmente implementadas usando uma lista simplesmente encadeada. No entanto, na lista simplesmente encadeada, a função para retirar do fim não pode ser implementada de forma eficiente, pois, dado o ponteiro para o último elemento da lista, não temos como acessar o anterior, que passaria a ser o então último elemento.

Para solucionar esse problema, temos que lançar mão da estrutura de lista duplamente encadeada (veja a seção 9.5). Nessa lista, cada nó guarda, além da referência para o próximo elemento, uma referência para o elemento anterior: dado o ponteiro de um nó, podemos acessar ambos os elementos adjacentes. Este arranjo resolve o problema de acessarmos o elemento anterior ao último. Devemos salientar que o uso de uma lista duplamente encadeada para implementar a fila é bastante simples, pois só manipulamos os elementos das extremidades da lista.

O arranjo de memória para implementarmos a fila dupla com lista é ilustrado na figura abaixo:

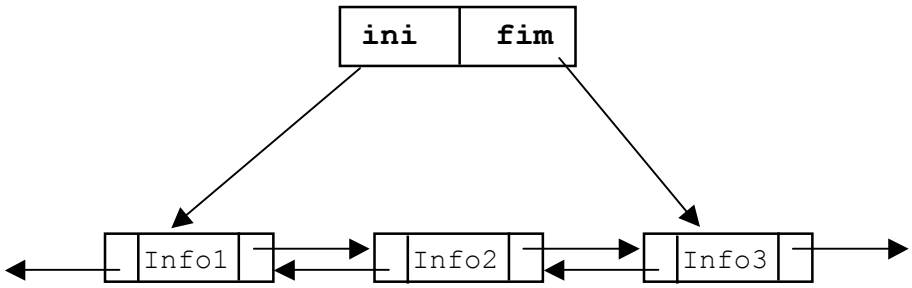

Figura 11.5: Arranjo da estrutura de fila dupla com lista.

O nó da lista duplamente encadeada para armazenar valores reais pode ser dado por:

```
struct no2 {
   float info;
   struct no2* ant;
   struct no2* prox;
};
typedef struct no2 No2;
```

A estrutura da fila dupla agrupa os ponteiros para o início e o fim da lista:

```
struct fila2 {
   No2* ini;
   No2* fim;
};
```

Interessa-nos discutir as funções para inserir e retirar elementos. As demais são praticamente idênticas às de fila simples. Podemos inserir um novo elemento em qualquer extremidade da fila. Portanto, precisamos de duas funções auxiliares de lista: para inserir no início e para inserir no fim. Ambas as funções são simples e já foram exaustivamente discutidas para o caso da lista simples. No caso da lista duplamente encadeada, a diferença consiste em termos que atualizar também o encadeamento para o elemento anterior. Uma possível implementação dessas funções é mostrada a seguir. Essas funções retornam, respectivamente, o novo nó inicial e final.

```
/* função auxiliar: insere no início */
No2* ins2_ini (No2* ini, float v) {
   No2* p = (No2*) malloc(sizeof(No2));
   p->info = v;
   p->prox = ini;
   p->ant = NULL;
   if (ini != NULL)   /* verifica se lista não estava vazia */
      ini->ant = p;
   return p;
}

/* função auxiliar: insere no fim */
No2* ins2_fim (No2* fim, float v) {
   No2* p = (No2*) malloc(sizeof(No2));
   p->info = v;
   p->prox = NULL;
   p->ant = fim;
   if (fim != NULL)   /* verifica se lista não estava vazia */
      fim->prox = p;
   return p;
}
```

Uma possível implementação das funções para remover o elemento do início ou do fim é mostrada a seguir. Essas funções também retornam, respectivamente, o novo nó inicial e final.

```
/* função auxiliar: retira do início */
No2* ret2_ini (No2* ini) {
   No2* p = ini->prox;
   if (p != NULL)   /* verifica se lista não ficou vazia */
      p->ant = NULL;
   free(ini);
   return p;
}

/* função auxiliar: retira do fim */
No2* ret2_fim (No2* fim) {
   No2* p = fim->ant;
   if (p != NULL)   /* verifica se lista não ficou vazia */
      p->prox = NULL;
   free(fim);
   return p;
}
```

As funções que manipulam a fila fazem uso dessas funções de lista, atualizando os ponteiros `ini` e `fim` quando necessário.

```
void insere_ini (Fila2* f, float v) {
   f->ini = ins2_ini(f->ini,v);
   if (f->fim==NULL)  /* fila antes vazia? */
      f->fim = f->ini;
}

void insere_fim (Fila2* f, float v) {
   f->fim = ins2_fim(f->fim,v);
   if (f->ini==NULL)  /* fila antes vazia? */
      f->ini = f->fim;
}

float retira_ini (Fila2* f) {
   float v;
   if (vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);         /* aborta programa */
   }
   v = f->ini->info;
   f->ini = ret2_ini(f->ini);
   if (f->ini == NULL)   /* fila ficou vazia? */
      f->fim = NULL;
   return v;
}

float retira_fim (Fila2* f) {
   float v;
   if (vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);         /* aborta programa */
   }
   v = f->fim->info;
   f->fim = ret2_fim(f->fim);
   if (f->fim == NULL)   /* fila ficou vazia? */
      f->ini = NULL;
   return v;
}
```