Princípios de Desenvolvimento de Algoritmos MAC122

Prof. Dr. Paulo Miranda
IME-USP

Pilhas & Filas

Estruturas Lineares

- Vários problemas frequentemente envolvem a manipulação de sequências ordenadas de objetos.
- Duas representações podem ser adotadas:
  - Representação sequencial (vetores),
  - Listas ligadas.
- A escolha dependerá da distribuição das operações realizadas sobre a sequência:
  - Acessar o k-ésimo elemento;
  - Inserir novo elemento entre posições fornecidas k e k+1;
  - Remover um dado elemento;
  - Concatenar duas sequências;
  - Copiar uma sequência;
  - Buscar um elemento que satisfaz uma propriedade;
  - Reordenar uma sequência;

Pilhas & Filas

- Pilha é um caso particular de lista linear, em que as operações de inserção e de remoção podem ser feitas somente numa única extremidade da lista.

Pilhas `(ligada)`

- De uma maneira geral, diremos que a pilha é uma estrutura linear cujas extremidades serão denominadas ***fundo*** e ***topo***.

- As operações de inserção (***empilhamento***) e remoção (***desempilhamento***) podem ser realizadas somente pelo topo da pilha.

- Estarei adotando uma lista não circular, o que torna mais claro quem é o fundo e quem é o topo.

- Considerei também o uso de nó-cabeça, para evitar passagem por referência do apontador (*ponteiro duplo*).

Pilhas (ligada)

- Definição típica da estrutura utilizada.

```
/* Implementação ligada de pilhas: */

typedef struct _RegPilha{
  TipoDado           dado;
  struct _RegPilha *prox;
} RegPilha;

typedef RegPilha* Pilha;

typedef enum boolean {false,true} bool;
```

```
RegPilha *AlocaRegPilha(){
  RegPilha* q;
  q = (RegPilha*)calloc(1, sizeof(RegPilha));
  if(q==NULL) exit(-1);
  return q;
}
```

Pilhas (ligada)

- Funções: Criação, Liberação, e teste de pilha vazia.

```
Pilha    CriaPilha(){
  Pilha p;
  p = AlocaRegPilha();
  p->prox = NULL;
  return p;
}
```

```
void     LiberaPilha(Pilha p){
  RegPilha *q,*t;
  q = p;
  while(q!=NULL){
    t = q;
    q = q->prox;
    free(t);
  }
}
```

```
bool     PilhaVazia(Pilha p){
  return (p->prox==NULL);
}
```

Pilhas (ligada)

- Inserindo e removendo elementos.

```
void      Empilha(Pilha p, TipoDado x){
  RegPilha *q;

  q = AlocaRegPilha();
  q->dado = x;
  q->prox = p->prox;
  p->prox = q;
}
```

```
TipoDado Desempilha(Pilha p){
  RegPilha *q;
  TipoDado x;

  q = p->prox;
  if(q==NULL) exit(-1);
  x = q->dado;
  p->prox = q->prox;
  free(q);
  return x;
}
```

Pilhas (sequencial)

- Definição típica da estrutura utilizada.

```
/* Implementação sequencial de pilhas: */

#define TAM_MAX 1000

typedef struct _RegPilha{
  int       topo;
  TipoDado array[TAM_MAX];
} RegPilha;

typedef RegPilha* Pilha;
```

Pilhas (sequencial)

- Funções: Criação, Liberação, e teste de pilha vazia.

```
Pilha    CriaPilha(){
  Pilha p;
  p = (Pilha)calloc(1, sizeof(RegPilha));
  if(p==NULL) exit(-1);
  p->topo = 0;
  return p;
}
```

```
void     LiberaPilha(Pilha p){
  free(p);
}
```

```
bool     PilhaVazia(Pilha p){
  return (p->topo==0);
}
```

Pilhas (sequencial)

- Inserindo e removendo elementos.

```
void      Empilha(Pilha p, TipoDado x){
  if(p->topo==TAM_MAX) exit(-1);

  p->array[p->topo] = x;
  p->topo++;
}
```

```
TipoDado Desempilha(Pilha p){
  if(p->topo==0) exit(-1);

  p->topo--;
  return (p->array[p->topo]);
}
```

Filas

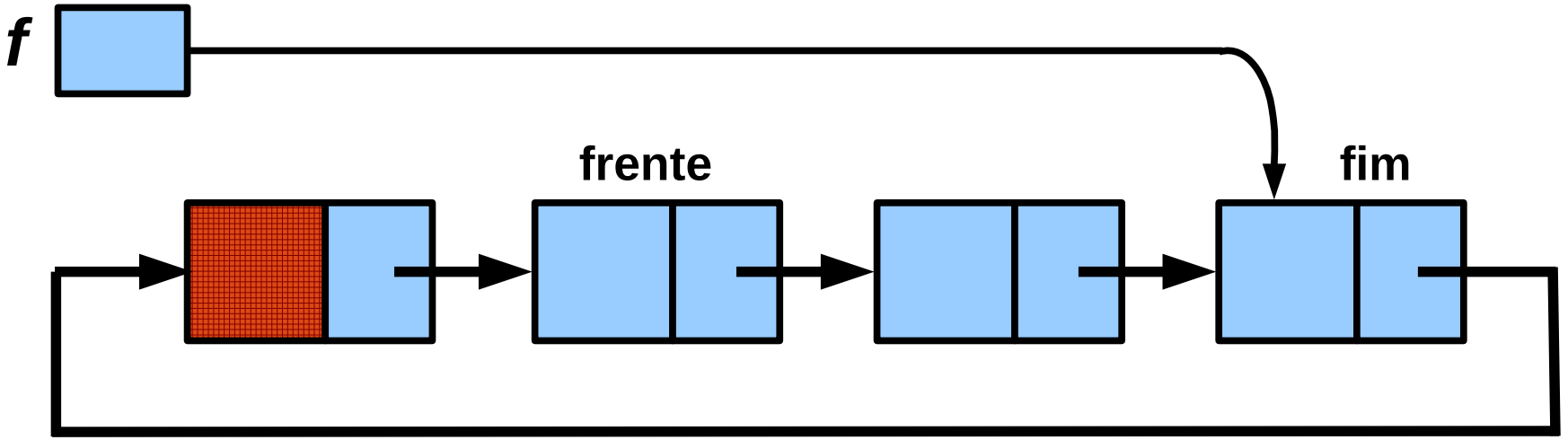
- De uma maneira geral, diremos que a fila é uma estrutura linear cujas extremidades serão denominadas ***frente*** e ***fim***.

- As operações de inserção e remoção são sempre realizadas em extremidades opostas da fila:

  - Inserção → fim,
  - remoção → frente.

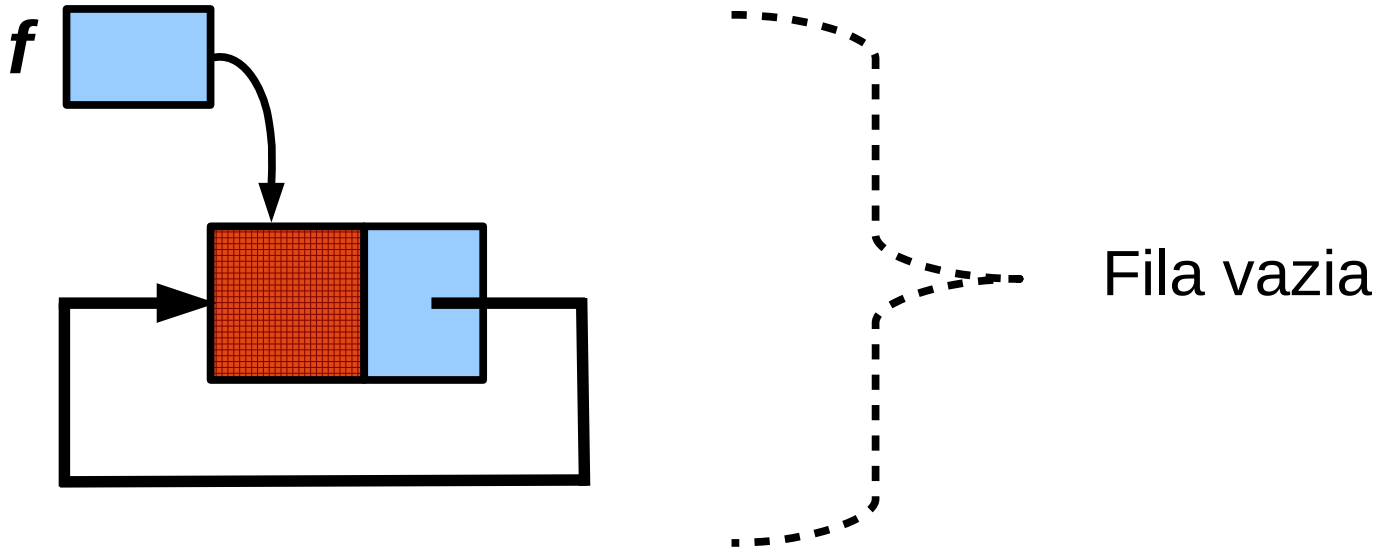
Filas (ligada)

- Estarei adotando uma lista circular com nó-cabeça, porém vou sempre conservar o apontador para a fila (ex: ***f***) apontando para o último elemento inserido, de modo a permitir o acesso eficiente a ambas extremidades com um número constante de operações.



Filas (ligada)

- Estarei adotando uma lista circular com nó-cabeça, porém vou sempre conservar o apontador para a fila (ex: ***f***) apontando para o último elemento inserido, de modo a permitir o acesso eficiente a ambas extremidades com um número constante de operações.



Filas (ligada)

- Definição típica da estrutura utilizada.

```
/* Implementação ligada de filas: */

typedef struct _RegFila{
  TipoDado          dado;
  struct _RegFila *prox;
} RegFila;

typedef RegFila* Fila;
```

```
RegFila *AlocaRegFila(){
  RegFila* q;
  q = (RegFila*)calloc(1, sizeof(RegFila));
  if(q==NULL) exit(-1);
  return q;
}
```

Filas (ligada)

- Funções: Criação, Liberação, e teste de pilha vazia.

```
Fila     CriaFila(){
  Fila p;
  p = AlocaRegFila();
  p->prox = p;
  return p;
}
```

```
void     LiberaFila(Fila p){
  RegFila *q,*t;
  q = p->prox;
  while(q!=p){
    t = q;
    q = q->prox;
    free(t);
  }
  free(p);
}
```

```
bool     FilaVazia(Fila p){
  return (p==p->prox);
}
```

Filas (ligada)

- Inserindo e removendo elementos.

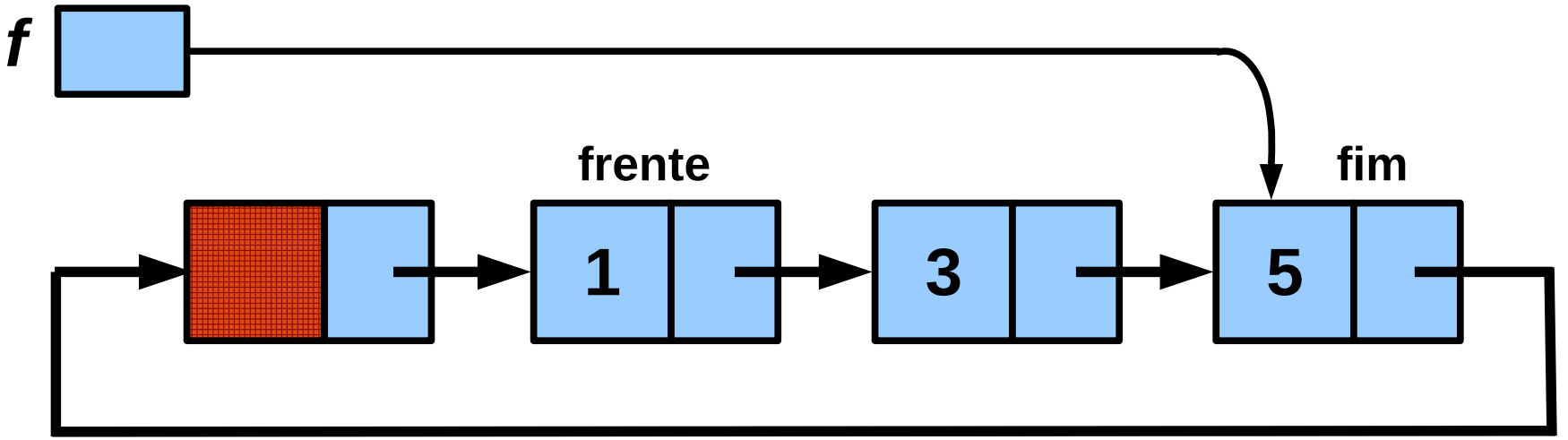
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```

```
TipoDado RemoveFila(Fila *p){
  RegFila *q,*t;
  TipoDado x;
  q = (*p)->prox;
  if(q==*p) exit(-1); /* Fila Vazia */
  t = q->prox;
  x = t->dado;
  q->prox = t->prox;
  if(t==*p) *p = q;
  free(t);
  return x;
}
```

Filas (ligada)

- Exemplo: inserindo elementos (chamando a função).

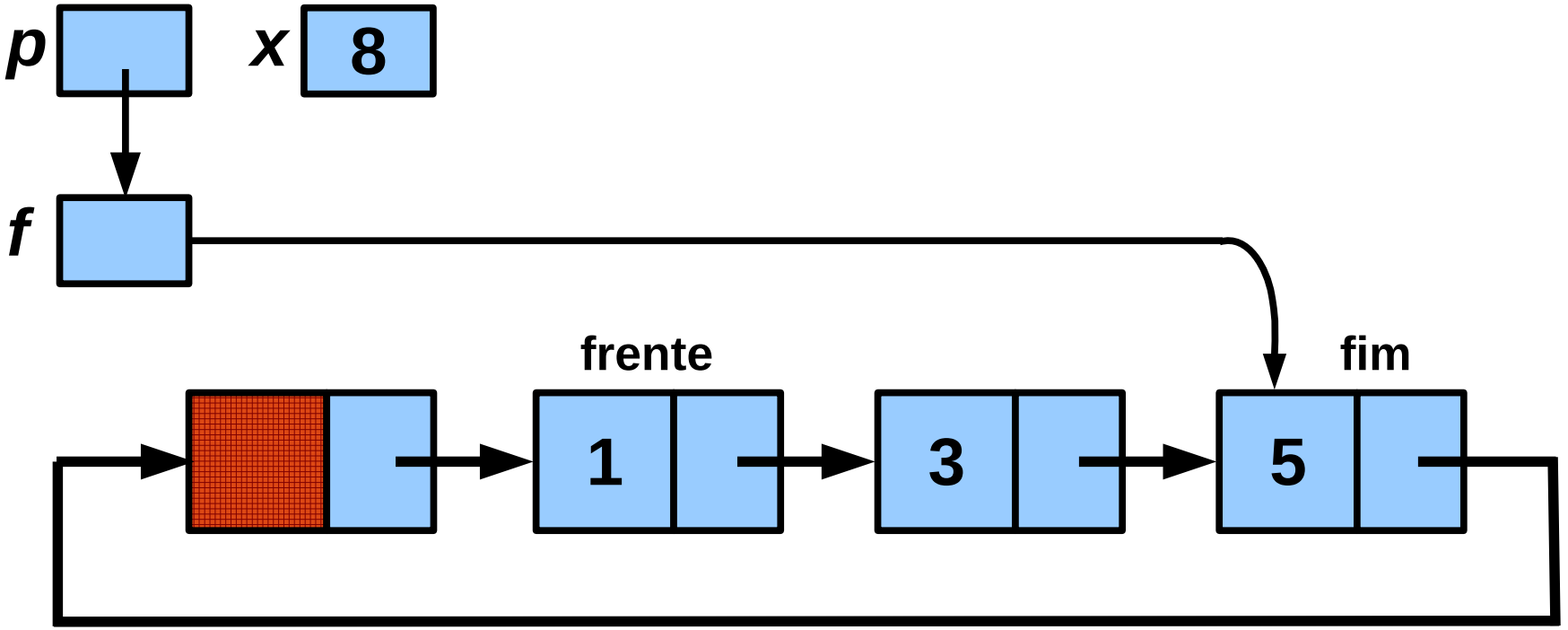
```
int main(){
  Fila f;
  f = CriaFila();
  InsereFila(&f, 1);
  InsereFila(&f, 3);
  InsereFila(&f, 5);

  InsereFila(&f, 8);
  ...
  return 0;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

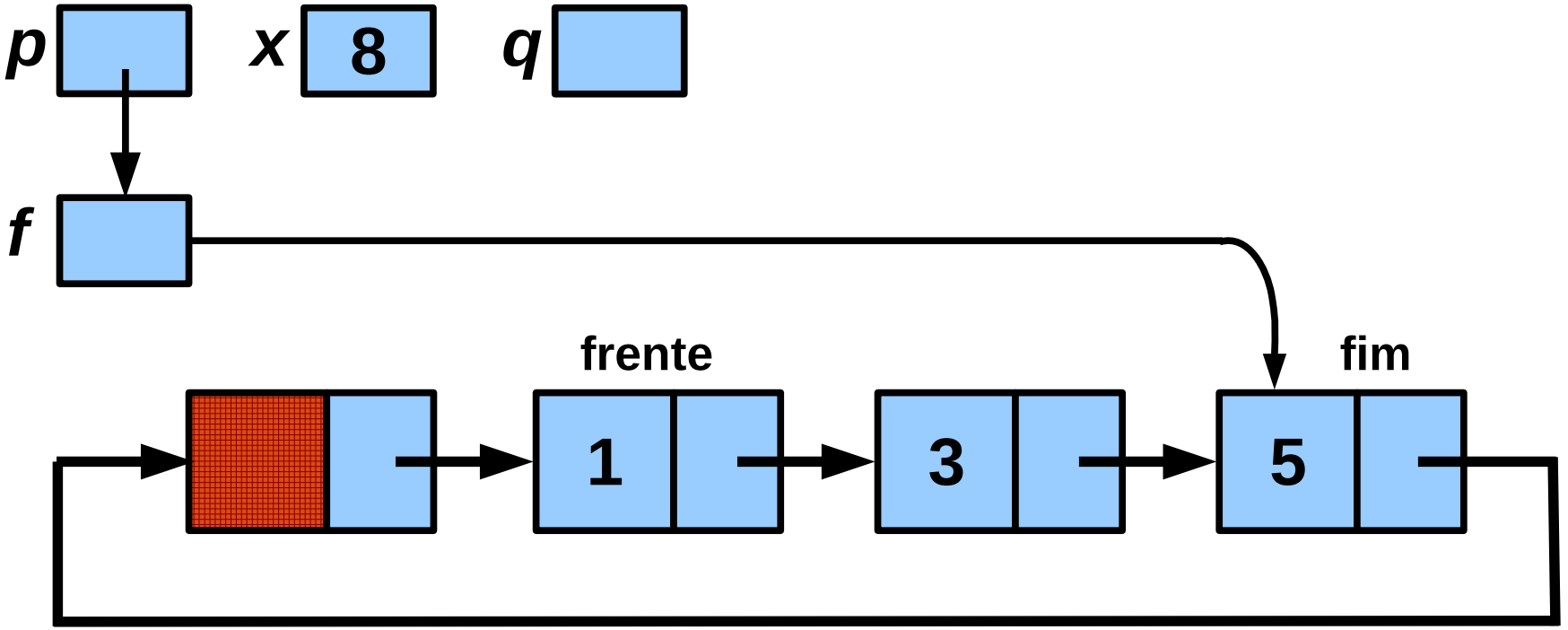
```
void     InsereFila(Fila *p, TipoDado x){
   RegFila *q;
   q = AlocaRegFila();
   q->dado = x;
   q->prox = (*p)->prox;
   (*p)->prox = q;
   *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

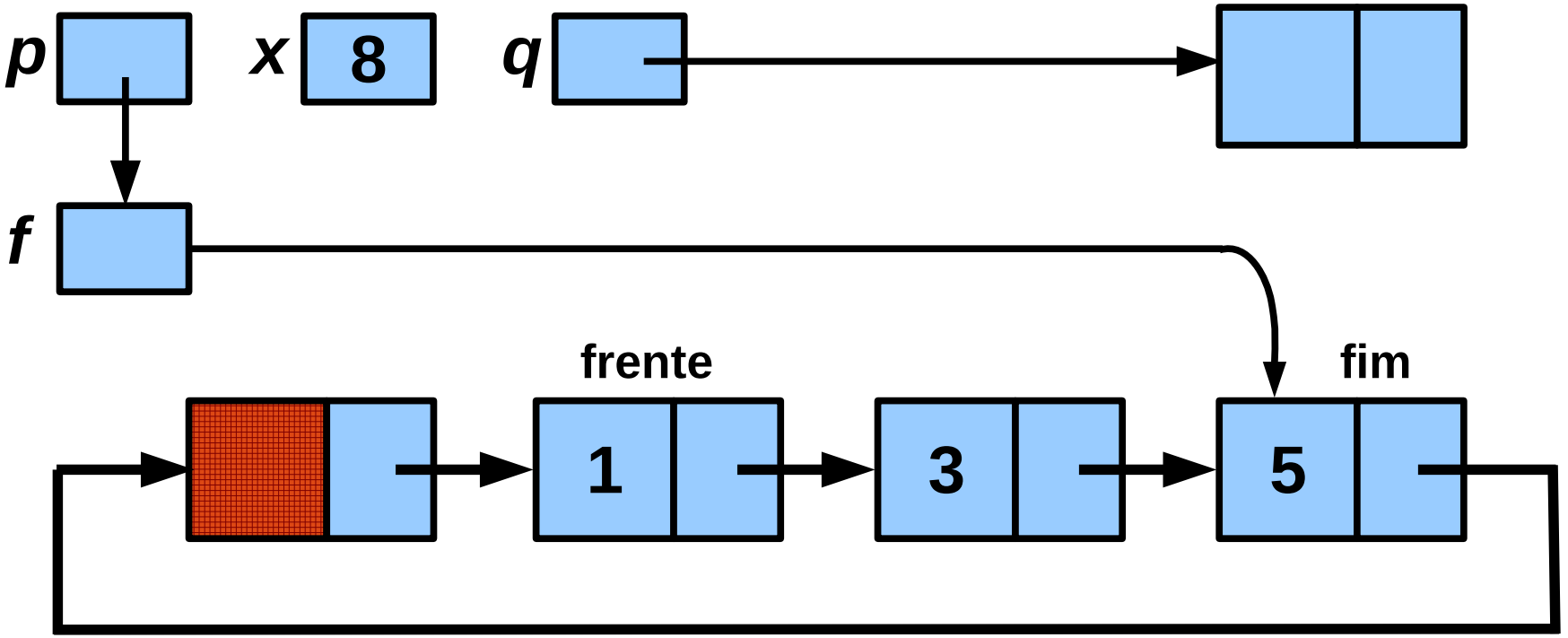
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

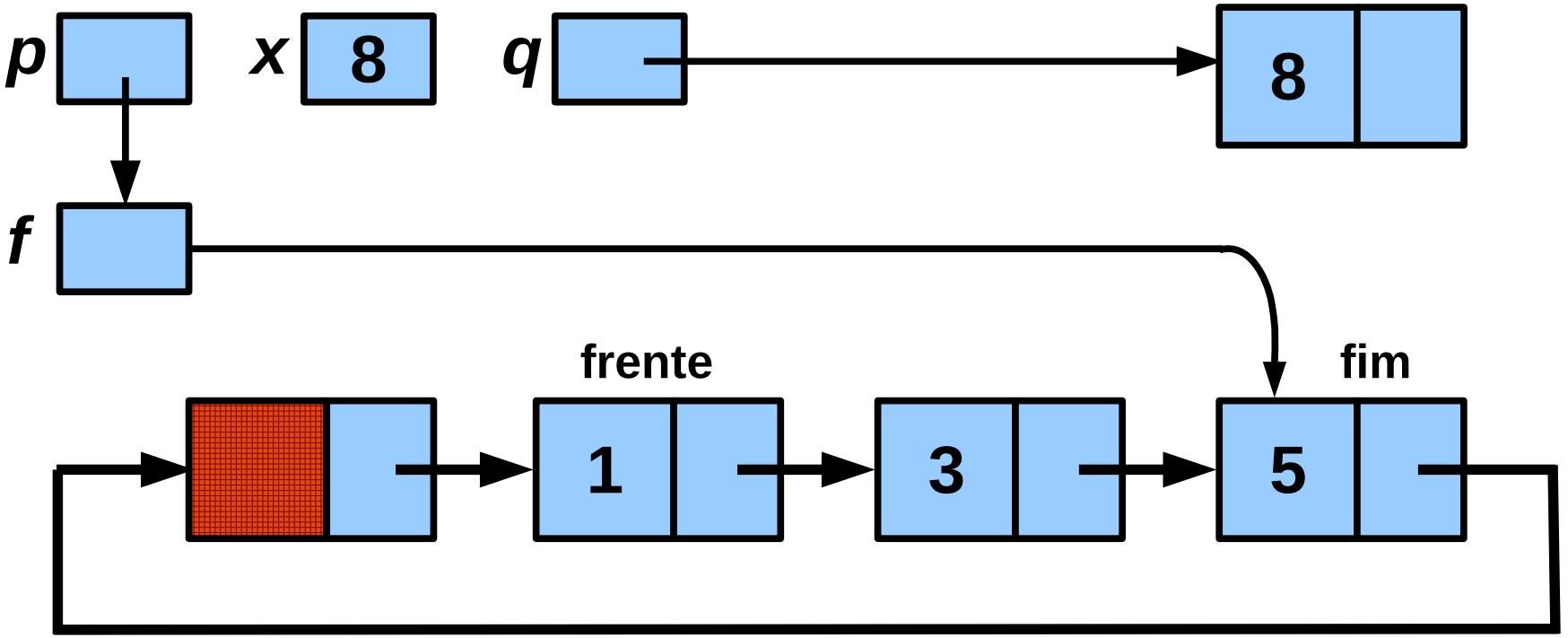
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

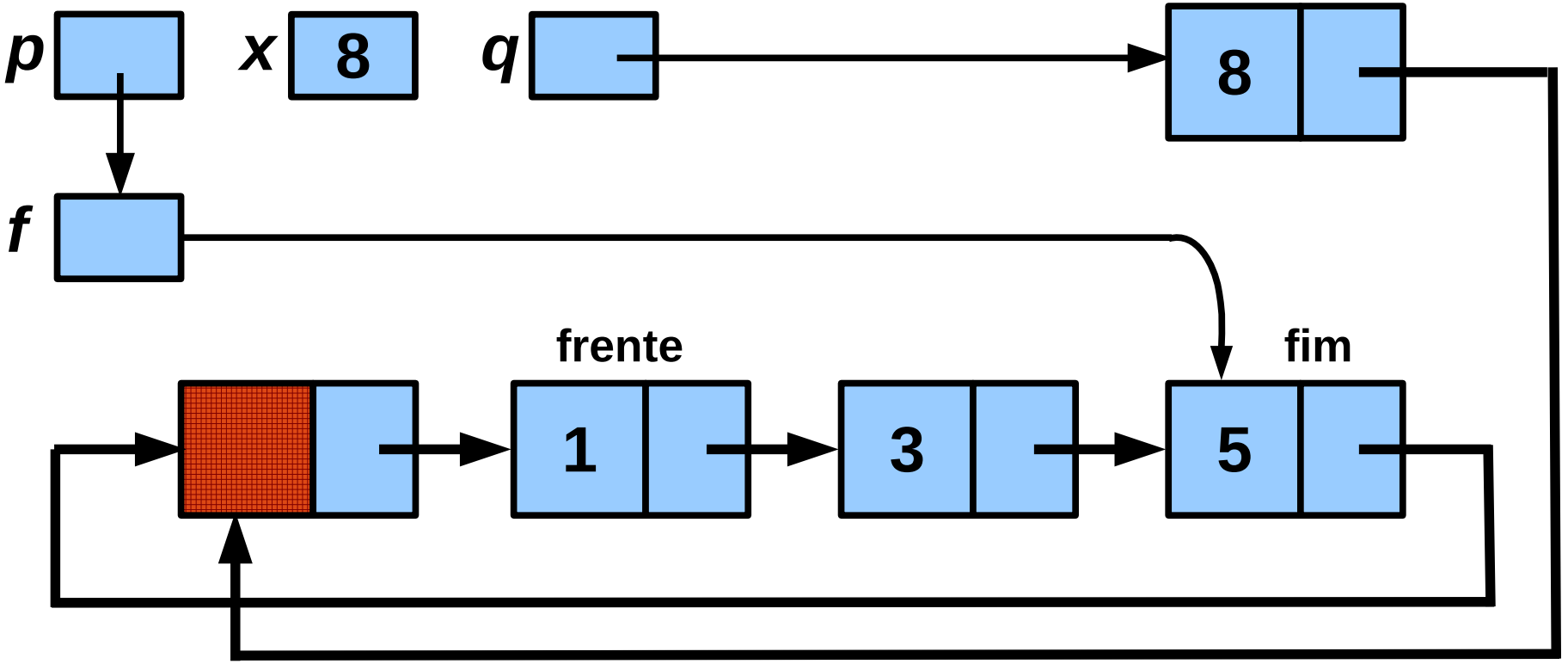
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

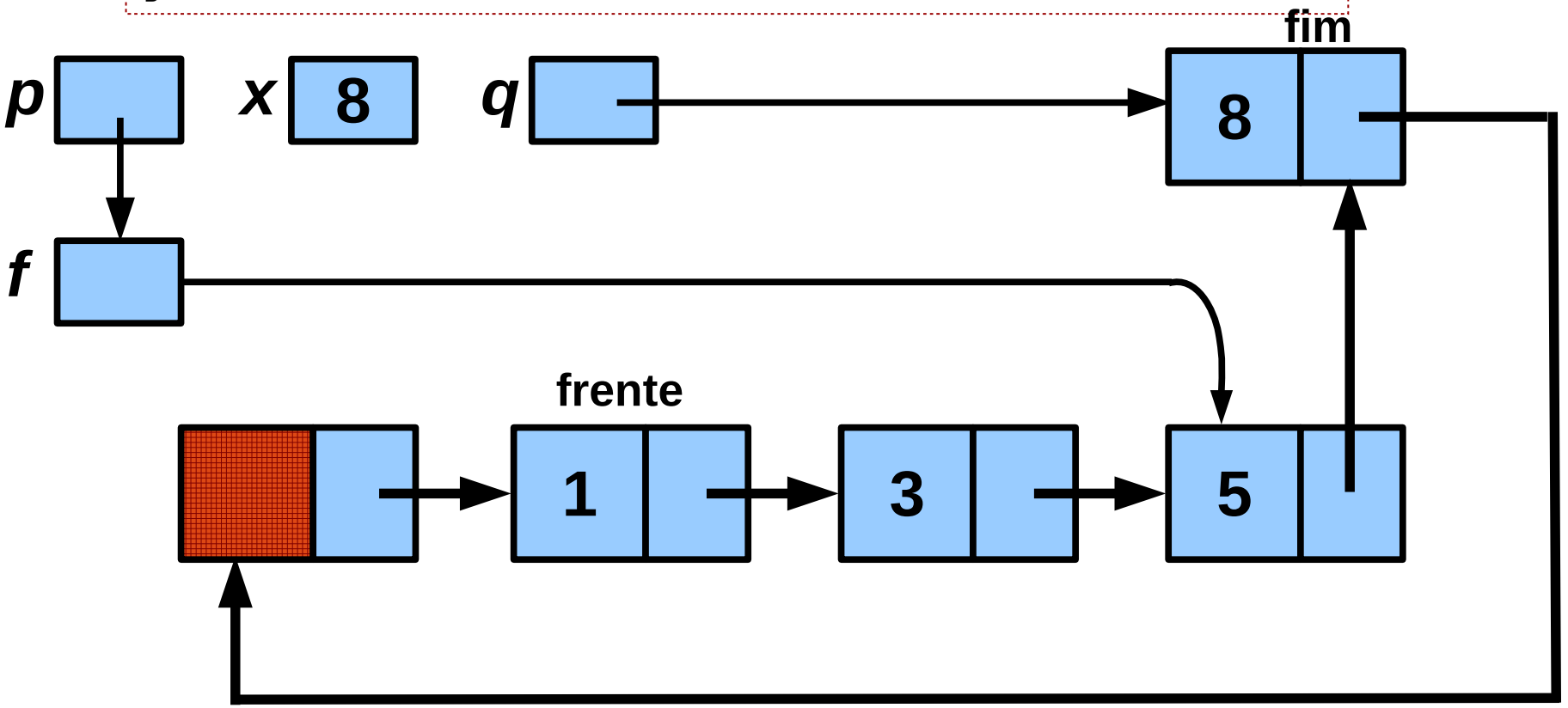
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

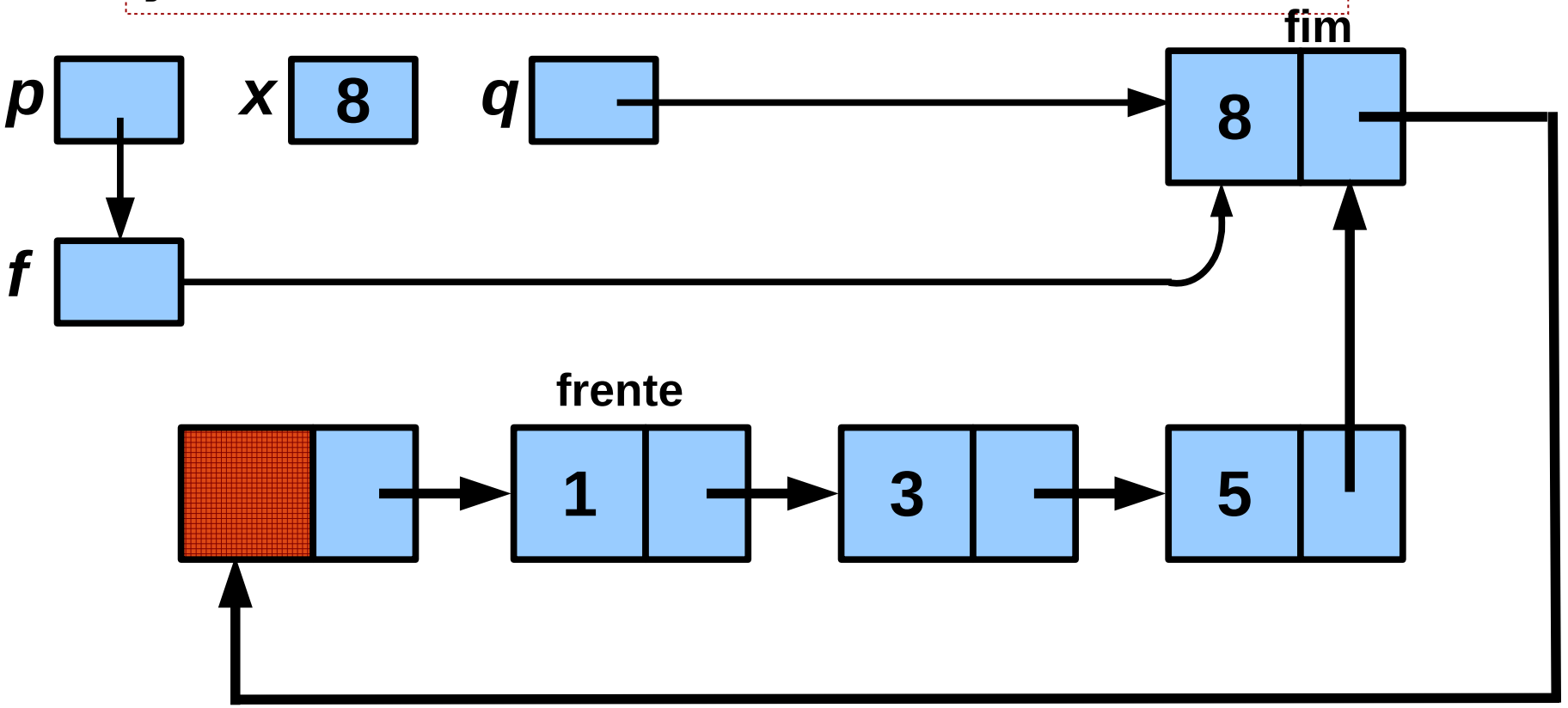
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

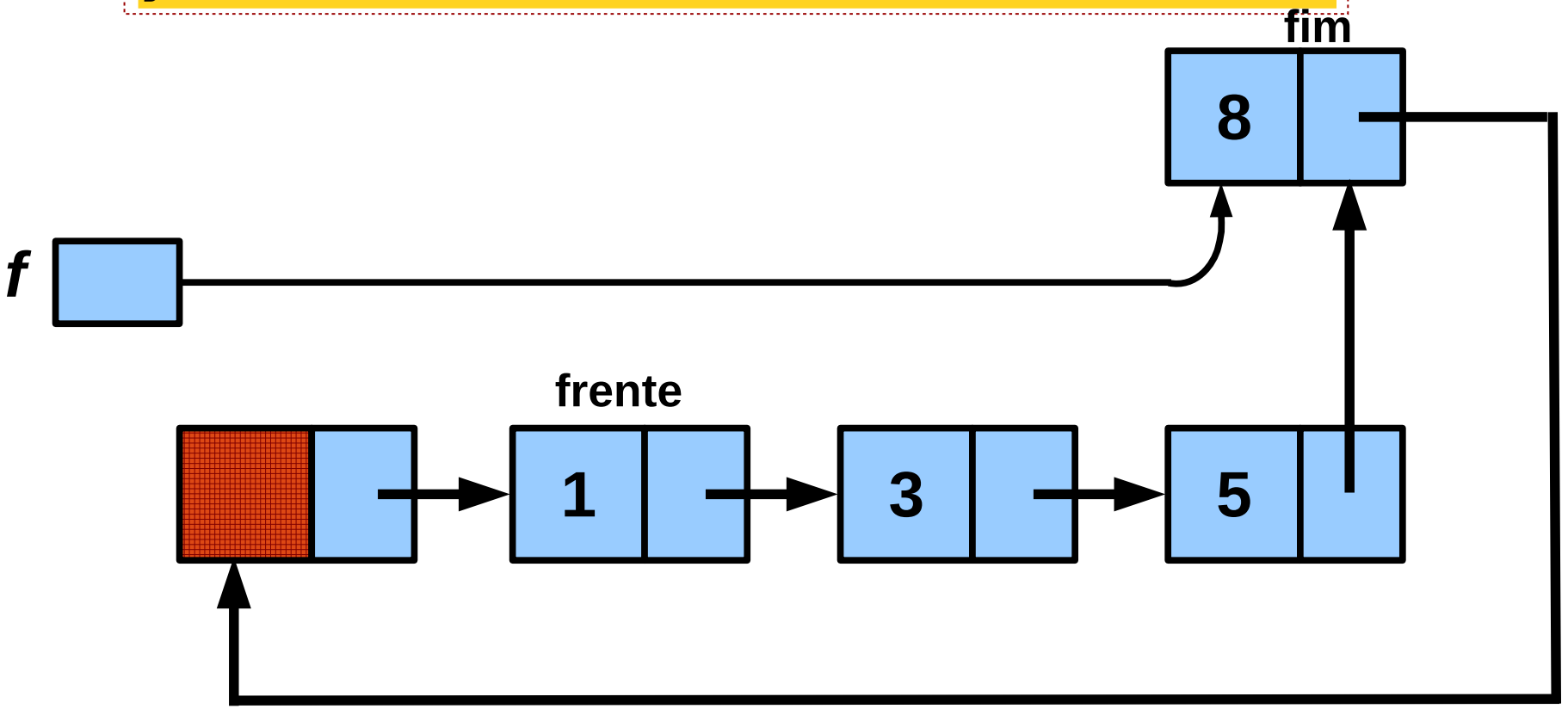
```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (ligada)

- Exemplo: inserindo elementos.

```
void     InsereFila(Fila *p, TipoDado x){
  RegFila *q;
  q = AlocaRegFila();
  q->dado = x;
  q->prox = (*p)->prox;
  (*p)->prox = q;
  *p = q;
}
```



Filas (sequencial)

- Por convenção, o índice "*frente*" aponta para a posição do primeiro elemento da fila, enquanto que o índice "*fim*" aponta para a posição que sucede o último elemento.

- Dificuldade para distinguir entre uma fila cheia e uma fila vazia:

  - Teste **`frente==fim`** dá verdadeiro nos dois casos.
  - Solução: sacrificar uma posição do vetor e usar como condição de fila cheia: **`frente==(fim+1)%n`**.

Filas (sequencial)

- Definição típica da estrutura utilizada.

```
#define TAM_MAX 1000


typedef struct _RegFila{
  int frente;
  int fim;
  TipoDado array[TAM_MAX];
} RegFila;


typedef RegFila* Fila;
```

Filas (sequencial)

- Funções: Criação, Liberação, e teste de pilha vazia.

```
Fila     CriaFila(){
  Fila p;
  p = (Fila)calloc(1, sizeof(RegFila));
  if(p==NULL) exit(-1);
  p->frente = 0;
  p->fim = 0;
  return p;
}
```

```
void     LiberaFila(Fila p){
  free(p);
}
```

```
bool     FilaVazia(Fila p){
  return (p->frente==p->fim);
}
```

Filas (sequencial)

- Inserindo e removendo elementos.

```
void      InsereFila(Fila *p, TipoDado x){
  Fila q = *p;

  if(q->frente==(q->fim+1)%TAM_MAX) /*Fila cheia */
    exit(-1);
  q->array[q->fim] = x;
  q->fim = (q->fim + 1)%TAM_MAX;
}
```

```
TipoDado RemoveFila(Fila *p){
  Fila q = *p;
  TipoDado x;

  if(FilaVazia(q)) exit(-1);

  x = q->array[q->frente];
  q->frente = (q->frente + 1)%TAM_MAX;

  return x;
}
```

Exemplos

- Problema de balanceamento de parênteses.

```
int main(){
  char cadeia[] = "([](([[[]]]))[()])";

  if( ParentesesBalanceados(cadeia) )
    printf("Correto\n");
  else
    printf("Incorreto\n");

  return 0;
}
```

Exemplos

- Problema de balanceamento de parênteses.

```
char OpeningPair(char c){
  switch(c){
  case ']': return '[';
  case ')': return '(';
  }
  exit(-1);
}
```

```
char ClosingPair(char c){
  switch(c){
  case '[': return ']';
  case '(': return ')';
  }
  exit(-1);
}
```

Exemplos

```
typedef char TipoDado;
bool ParentesesBal(char *cadeia){
  Pilha p;
  char t;
  int i;
  bool balanceada = false;
  bool continua = true;

  p = CriaPilha();
  i = 0;

  while(continua){
    switch(cadeia[i]){
    case '\0':
      balanceada = PilhaVazia(p);
      continua = false;
      break;

    case '(':
    case '[':
      Empilha(p, cadeia[i]);
      break;

    case ')':
    case ']':
      if(PilhaVazia(p))
        continua = false;
      else{
        t = (char)Desempilha(p);
        if(t!=OpeningPair(cadeia[i]))
          continua = false;
      }
      break;
    }
    i++;
  }

  LiberaPilha(p);
  return balanceada;
}
```

Exemplos

- Transformação da notação infixa para pós-fixa.
- Na notação pós-fixa:
  - Não é necessário parênteses.
  - A ordem dos operadores na expressão diz a ordem em que eles vão ser executados (da esquerda para a direita).
  - Nota-se que os nomes das variáveis são copiados da entrada infixa para a saída pós-fixa na mesma ordem.
  - Os operadores esperam até que apareça na entrada um de prioridade menor ou igual.

Exemplos

- Transformação da notação infixa para pós-fixa.

```
int main(){
  char expr[]="a*(b+c)*(d-g)*h";

  In2Pos(expr);

  return 0;
}
```

Exemplos

```
typedef char TipoDado;
void In2Pos(char *expr){
  Pilha p = CriaPilha();
  char c,t;
  int i;
  bool fim;

  Empilha(p, '(');
  i = 0;
  do{
    c = expr[i]; i++;
    switch(c){

    case ')': case '\0':
      do{
        t = (char)Desempilha(p);
        if(t!='(')
          printf("%c",t);
      }while(t!='(');
      break;

    case '+': case '-': case '*':
    case '/': case '^':
      fim = false;
      do{
        t = (char)Desempilha(p);
        if(Prioridade(c,t)){
          Empilha(p, t);
          Empilha(p, c);
          fim = true;
        }
        else
          printf("%c",t);
      }while(!fim);
      break;

    case '(':
      Empilha(p, '(');
      break;

    default:
      if(c>='a' && c<='z')
        printf("%c",c);
    }
  }while(c!='\0');
  LiberaPilha(p);
  printf("\n");
}
```