A pilha estará vazia se a lista estiver vazia:

```
int vazia (Pilha* p)
{
   return (p->prim==NULL);
}
```

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

```
void libera (Pilha* p)
{
   No* q = p->prim;
   while (q!=NULL) {
      No* t = q->prox;
      free(q);
      q = t;
   }
   free(p);
}
```

A rigor, pela definição da estrutura de pilha, só temos acesso ao elemento do topo. No entanto, para testar o código, pode ser útil implementarmos uma função que imprima os valores armazenados na pilha. Os códigos abaixo ilustram a implementação dessa função nas duas versões de pilha (vetor e lista). A ordem de impressão adotada é do topo para a base.

```
/* imprime: versão com vetor */
void imprime (Pilha* p)
{
   int i;
   for (i=p->n-1; i>=0; i--)
      printf("%f\n",p->vet[i]);
}

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

11.4. Exemplo de uso: calculadora pós-fixada

Um bom exemplo de aplicação de pilha é o funcionamento das calculadoras da HP (Hewlett-Packard). Elas trabalham com expressões pós-fixadas, então para avaliarmos uma expressão como `(1-2)*(4+5)` podemos digitar `1 2 - 4 5 + *`. O funcionamento dessas calculadoras é muito simples. Cada operando é empilhado numa pilha de valores. Quando se encontra um operador, desempilha-se o número apropriado de operandos (dois para operadores binários e um para operadores unários), realiza-se a operação devida e empilha-se o resultado. Deste modo, na expressão acima, são empilhados os valores `1` e `2`. Quando aparece o operador `-`, `1` e `2` são desempilhados e o resultado da operação, no caso `-1` (`= 1 - 2`), é colocado no topo da pilha. A seguir, `4` e `5` são empilhados. O operador seguinte, `+`, desempilha o `4` e o `5` e empilha o resultado da soma, `9`. Nesta hora, estão na pilha os dois resultados parciais, `-1` na base e `9` no topo. O operador `*`, então, desempilha os dois e coloca `-9` (`= -1 * 9`) no topo da pilha.

Como exemplo de aplicação de uma estrutura de pilha, vamos implementar uma calculadora pós-fixada. Ela deve ter uma pilha de valores reais para representar os operandos. Para enriquecer a implementação, vamos considerar que o formato com que os valores da pilha são impressos seja um dado adicional associado à calculadora. Esse formato pode, por exemplo, ser passado quando da criação da calculadora.

Para representar a interface exportada pela calculadora, podemos criar o arquivo `calc.h`:

```
/* Arquivo que define a interface da calculadora */

typedef struct calc Calc;

/* funções exportadas */
Calc* cria_calc (char* f);
void operando (Calc* c, float v);
void operador (Calc* c, char op);
void libera_calc (Calc* c);
```

Essas funções utilizam as funções mostradas acima, independente da implementação usada na pilha (vetor ou lista). O tipo que representa a calculadora pode ser dado por:

```
struct calc {
   char f[21];   /* formato para impressão */
   Pilha* p;     /* pilha de operandos */
};
```

A função `cria` recebe como parâmetro de entrada uma cadeia de caracteres com o formato que será utilizado pela calculadora para imprimir os valores. Essa função cria uma calculadora inicialmente sem operandos na pilha.

```
Calc* cria_calc (char* formato)
{
 Calc* c = (Calc*) malloc(sizeof(Calc));
 strcpy(c->f,formato);
 c->p = cria();           /* cria pilha vazia */
 return c;
}
```

A função `operando` coloca no topo da pilha o valor passado como parâmetro. A função `operador` retira os dois valores do topo da pilha (só consideraremos operadores binários), efetua a operação correspondente e coloca o resultado no topo da pilha. As operações válidas são: `'+'` para somar, `'-'` para subtrair, `'*'` para multiplicar e `'/'` para dividir. Se não existirem operandos na pilha, consideraremos que seus valores são zero. Tanto a função `operando` quanto a função `operador` imprimem, utilizando o formato especificado na função `cria`, o novo valor do topo da pilha.

```
void operando (Calc* c, float v)
{
 /* empilha operando */
 push(c->p,v);

 /* imprime topo da pilha */
 printf(c->f,v);
}
```

```
void operador (Calc* c, char op)
{
 float v1, v2, v;

 /* desempilha operandos */
 if (vazia(c->p))
    v2 = 0.0;
 else
    v2 = pop(c->p);
 if (vazia(c->p))
    v1 = 0.0;
 else
    v1 = pop(c->p);

 /* faz operação */
 switch (op) {
    case '+':  v = v1+v2;  break;
    case '-':  v = v1-v2;  break;
    case '*':  v = v1*v2;  break;
    case '/':  v = v1/v2;  break;
 }

 /* empilha resultado */
 push(c->p,v);

 /* imprime topo da pilha */
 printf(c->f,v);
}
```

Por fim, a função para liberar a memória usada pela calculadora libera a pilha de operandos e a estrutura da calculadora.

```
void libera_calc (Calc* c)
{
   libera(c->p);
   free(c);
}
```

Um programa cliente que faça uso da calculadora é mostrado abaixo:

```
/* Programa para ler expressão e chamar funções da calculadora */

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

int main (void)
{
 char c;
 float v;
 Calc* calc;

 /* cria calculadora com precisão de impressão de duas casas decimais
*/
 calc = cria_calc("%.2f\n");

 do {
  /* le proximo caractere nao branco */
  scanf(" %c",&c);
  /* verifica se e' operador valido */
  if (c=='+' || c=='-' || c=='*' || c=='/') {
     operador(calc,c);
  }
  /* devolve caractere lido e tenta ler número */
  else {
     ungetc(c,stdin);
     if (scanf("%f",&v) == 1)
        operando(calc,v);
```

```
    }
   } while (c!='q');
   libera_calc(calc);
   return 0;
  }
```

Esse programa cliente lê os dados fornecidos pelo usuário e opera a calculadora. Para tanto, o programa lê um caractere e verifica se é um operador válido. Em caso negativo, o programa "devolve" o caractere lido para o *buffer* de leitura, através da função `ungetc`, e tenta ler um operando. O usuário finaliza a execução do programa digitando `q`.

Se executado, e considerando-se as expressões digitadas pelo usuário mostradas abaixo, esse programa teria como saída:

```
3 5 8 * +              ← digitado pelo usuário
3.00
5.00
8.00
40.00
43.00
7 /                    ← digitado pelo usuário
7.00
6.14
q                      ← digitado pelo usuário
```

Exercício: Estenda a funcionalidade da calculadora incluindo novos operadores unários e binários (sugestão: `~` como menos unário, `#` como raiz quadrada, `^` como exponenciação).