Exercícios: Listas ligadas - Alocação dinâmica

Ronaldo Fumio Hashimoto

1 Lista ligada

1. Considere as seguintes declarações:

```
typedef struct celula_dupla {
  int conteudo;
  struct celula *prox;
} Celula;

Celula *lista1, *lista2;
```

   Para esta questão, considere listas ligadas com cabeça.

   - Implemente a função

     `Celula * inicializa ();`

     que inicializa uma lista ligada com cabeça.
   - Implemente a função

     `int length (Celula *ini);`

     que devolve o número de elementos da lista apontada por `ini`.

2. Implemente a função

   `int troca (int k, Celula *ini, int x);`

   que devolve (via `return`) o k-ésimo elemento da lista apontada por `ini` e modifica o conteúdo desta célula para `x`. Se `k` é maior que o número de elementos da lista, a função deve inserir `x` no final da lista e devolver `x` via `return`.

   - Implemente a função

     `void insere_esq (int k, Celula *ini, int x);`

     que insere `x` à esquerda do k-ésimo elemento da lista apontada por `ini`. Se `k` é maior que o número de elementos da lista, a função deve inserir `x` no início da lista.
   - Implemente a função

     `void insere_dir (int k, Celula *ini, int x);`

     que insere `x` à direita do k-ésimo elemento da lista apontada por `ini`. Se `k` é maior que o número de elementos da lista, a função deve inserir `x` no final da lista.
   - Implemente a função

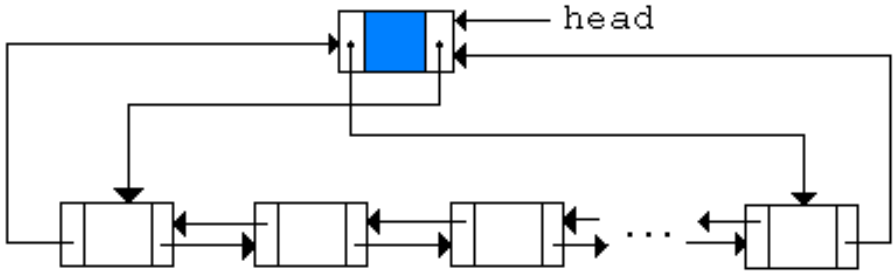
     `int remove (int k, Celula *ini);`

     que remove o k-ésimo elemento da lista apontada por `ini` e devolve, via `return`, o inteiro removido. Se `k` é maior que o número de elementos da lista e a lista não está vazia, a função deve remover o último elemento da lista e devolver o se conteúdo. No caso da lista estar vazia, a função deve devolver o inteiro 0.
   - Implemente a função

     `void banir (int x, Celula *ini);`

que elimina todas as ocorrências do elemento `x` da lista apontada por `ini`.

3. Uma **lista duplamente** encadeada possui campos que têm ligações com o sucessor e o predecessor na lista. Ainda, é usual ter um "nó head", que é um registro auxiliar que aponta para o "primeiro" e o "último" registro da lista e é apontado por eles.




```
typedef struct celula_dupla {
  int conteudo;
  struct celula *esq, *dir;
} Celula_Dupla;
```

   - Implemente a função

     `Celula_Dupla * inicializa ();`

     que inicializa uma lista duplamente ligada com cabeça.
   - Implemente a função

     `Celula_Dupla * busca (int x, Celula_Dupla * ini);`

     busca o inteiro `x` na lista duplamente ligada com cabeça apontada por `ini` e devolve o ponteiro da primeira célula que contém `x`; ou `NULL`, se `x` não está na lista.
   - Implemente a função

     `int busca_insere (int x, int y, Celula_Dupla * ini);`

     busca `y` na lista duplamente ligada com cabeça apontada por `ini` e insere `x` de forma que ele fique em uma célula imediatamente anterior a da que contém `y`; se `y` não estiver na lista, a função deverá inserir `x` na última posição da lista. Esta função devolve o inteiro 1 se `y` está na lista; 0, caso contrário.
   - Implemente a função

     `int busca_remove (int y, Celula_Dupla * ini);`

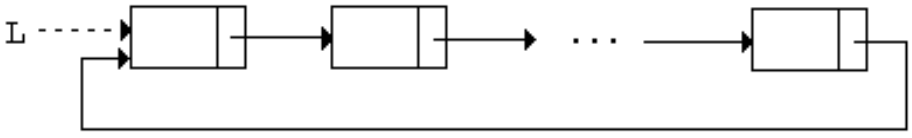
     busca `y` na lista duplamente ligada apontada por `ini` e remove a célula imediatamente anterior que contém `y`. Neste caso, a função deve devolver o conteúdo da imediatamente anterior que contém `y`. Se `y` não está na lista, temos dois casos possíveis: (*i*) a lista está vazia: neste caso, a função deve devolver o inteiro 0; (*ii*) a lista não está vazia: neste caso, a sua função deve remover a última célula da lista e devolver o seu conteúdo.

4. Escreva uma função

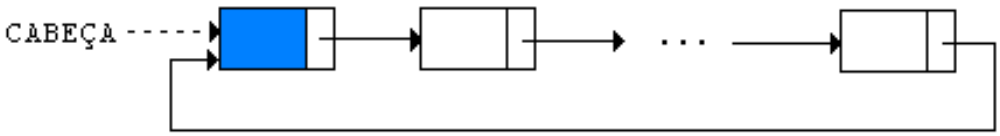
   `Celula_Dupla * concatena (Celula_Dupla * lista1, Celula_Dupla * lista2);`

   que concatena duas listas duplamente encadeadas com cabeça somente manipulando os ponteiros. A sua função deve liberar uma das cabeças.

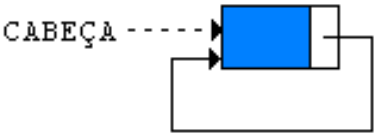
5. Uma **lista encadeada circular** é uma lista encadeada cujo último elemento aponta para o primeiro:



Vantagem: cada elemento é acessível a partir de qualquer outro. Numa lista circular, não faz mais sentido se falar em primeiro ou último elemento. Porém, devemos saber, durante um percurso na lista, se já demos uma volta completa, para evitarmos loops infinitos. Uma forma de se evitar isso, podemos assumir a existência de uma célula especial, chamado **cabeça de Lista**, cujo campo de conteúdo ou informação não pertence ao conjunto de elementos da lista (poderia até servir de sentinela numa busca):



Situação lista circular vazia está na figura abaixo:



- Implemente a função

  ```
  Celula * inicializa ();
  ```

  que inicializa uma lista ligada circular com cabeça.
- Implemente a função

  ```
  int length (Celula * ini);
  ```

  que conta o número de elementos de uma lista ligada circular com cabeça apontada por `ini`.
- Implemente a função

  ```
  Celula * insere (int x, Celula * ini);
  ```

  que insere um elemento de conteúdo `x` à esquerda da cabeça de uma lista ligada circular apontada por `ini`.
- Implemente a função

  ```
  Celula * elimina (int x, Celula * ini);
  ```

  que procura e elimina toda célula que contém o conteúdo `x` de uma lista ligada circular apontada por `ini`.
- Implemente uma função

  ```
  Celula * concatena (Celula * lista1, Celula * lista2);
  ```

  que concatena duas listas ligadas circular somente pela manipulando os ponteiros. A sua função deve liberar uma das cabeças.

6. Desejamos manipular polinômios do tipo $p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$. Tais polinômios podem ser representados por listas simplesmente encadeadas onde cada nó da lista possui três campos: um para o coeficiente que é um número real, um para o expoente que é um número inteiro e um campo que armazena um ponteiro para o próximo nó.

   - Implemente uma função

     ```
     Celula * soma (Celula * poli1, Celula * poli2);
     ```

     que contrói uma nova lista ligada (sem destruir as listas ligadas apontadas por `poli1` e `poli2`) correspondente à soma dos polinômios armazenados nas listas apontadas por `poli1` e `poli2`.

7. Implemente uma função

   ```
   Celula * multiplica (Celula * poli1, Celula * poli2);
   ```

que contrói uma nova lista ligada (sem destruir as listas ligadas apontadas por `poli1` e `poli2`) correspondente à multiplicação dos polinômios armazenados nas listas apontadas por `poli1` e `poli2`.

- Implemente uma função

  ```
  Celula * derivada (Celula * poli);
  ```

  que contrói uma nova lista ligada (sem destruir a lista ligada apontada por `poli`) correspondente à derivada do polinômio armazenado na lista apontada por `poli`.

8. Considere conjuntos sendo implementados através de listas encadeadas.

   - Implemente uma função

     ```
     Celula * uniao (Celula * conj1, Celula * conj2);
     ```

     que contrói uma nova lista ligada (sem destruir as listas ligadas apontadas por `conj1` e `conj2`) correspondente à união dos conjuntos armazenados nas listas apontadas por `conj1` e `conj2`. Observe que na representação de conjuntos, não pode existir elementos repetivos nas listas ligadas.
   - Implemente uma função

     ```
     Celula * interseccao (Celula * conj1, Celula * conj2);
     ```

     que contrói uma nova lista ligada (sem destruir as listas ligadas apontadas por `conj1` e `conj2`) correspondente à intersecção dos conjuntos armazenados nas listas apontadas por `conj1` e `conj2`. Observe que na representação de conjuntos, não pode existir elementos repetivos nas listas ligadas.
   - Implemente uma função

     ```
     Celula * diferenca (Celula * conj1, Celula * conj2);
     ```

     que contrói uma nova lista ligada (sem destruir as listas ligadas apontadas por `conj1` e `conj2`) correspondente à diferença dos conjuntos armazenados nas listas apontadas por `conj1` e `conj2` (`conj1` menos `conj2`). Observe que na representação de conjuntos, não pode existir elementos repetivos nas listas ligadas.
   - Implemente uma função

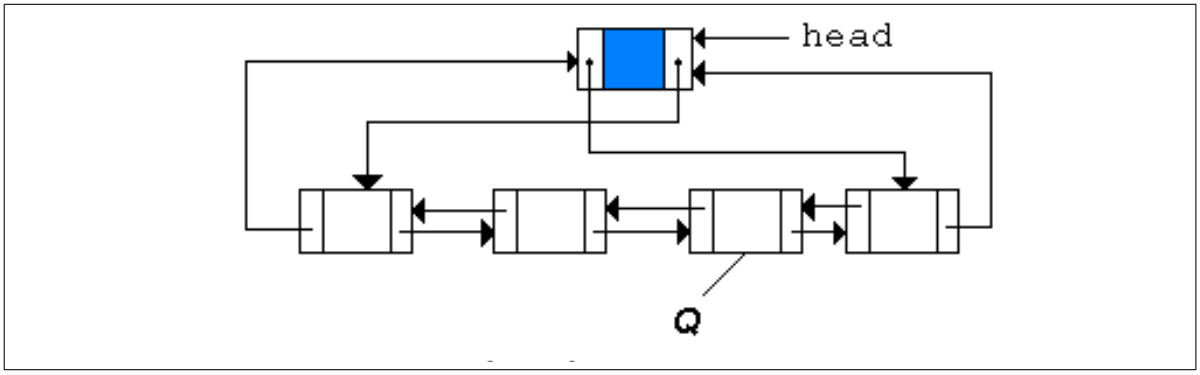
     ```
     int pertence (int x, Celula * conj);
     ```

     que verifica se um inteiro `x` pertence ao conjunto armazenado na lista apontada por `conj` devolvendo 1, em caso afirmativo; ou 0, caso contrário.

9. Considere a seguinte situação: um congresso é formado por 400 deputados. Cada deputado está filiado a, no máximo, um partido. Mudanças de partido são permitidas e frequuentes. O presidente do congresso deseja fornecer periodicamente à imprensa listagens com a bancada de cada partido.

   - Considere um vetor de listas ligadas como estrutura de dados para o problema;
   - Faça uma função de nome `troca` que recebe o nome de um deputado e altera o partido do deputado (dados o partido antigo e o novo);

10. Uma **lista duplamente encadeada circular** é ilustrada na figura abaixo:



- Escreva uma função

```
Celula_Dupla quebra (Celula_Dupla * ini, Celula_Dupla * Q);
```

que recebe um ponteiro para uma célula da lista duplamente encadeada circular apontada por `ini` e a divide em uma lista duplamente encadeada circular com os elementos a partir de Q, como mostrado na figura abaixo.