Algoritmos e Estruturas de dados

Listas Encadeadas - continuação

Prof. Dr. Fábio Rodrigues de la Rocha

UTFPR

Listas encadeadas com indicadores de inicio e fim

Anteriormente foram apresentadas listas encadeadas que eram assim definidas:

```
1 struct tipo_l {
2         int valor;
3         struct tipo_l *proximo;
4 };
5
6 typedef struct {
7         struct tipo_l *inicio;
8         int quantidade;
9 }Tipo_Lista;
```

Ou seja, existe um indicador de inicio da lista. Este indicador é utilizado por algumas funções para "saber" onde a lista começa.

Listas com indicador de início e fim

Uma variação do esquema apresentado é utilizar um indicador de início e fim. Utilizando este indicador pode-se conhecer quem é o último elemento da lista, sem a necessidade de percorrê-la. Isto é especialmente útil numa operação de inserção, onde devemos inserir elementos ao final da lista.

```
1  struct tipo_l {
2          int valor;
3          struct tipo_l *proximo;
4  };
5
6  typedef struct {
7           struct tipo_l *inicio;
8           struct tipo_l *fim;
9           int quantidade;
10 }Tipo_Lista;
```

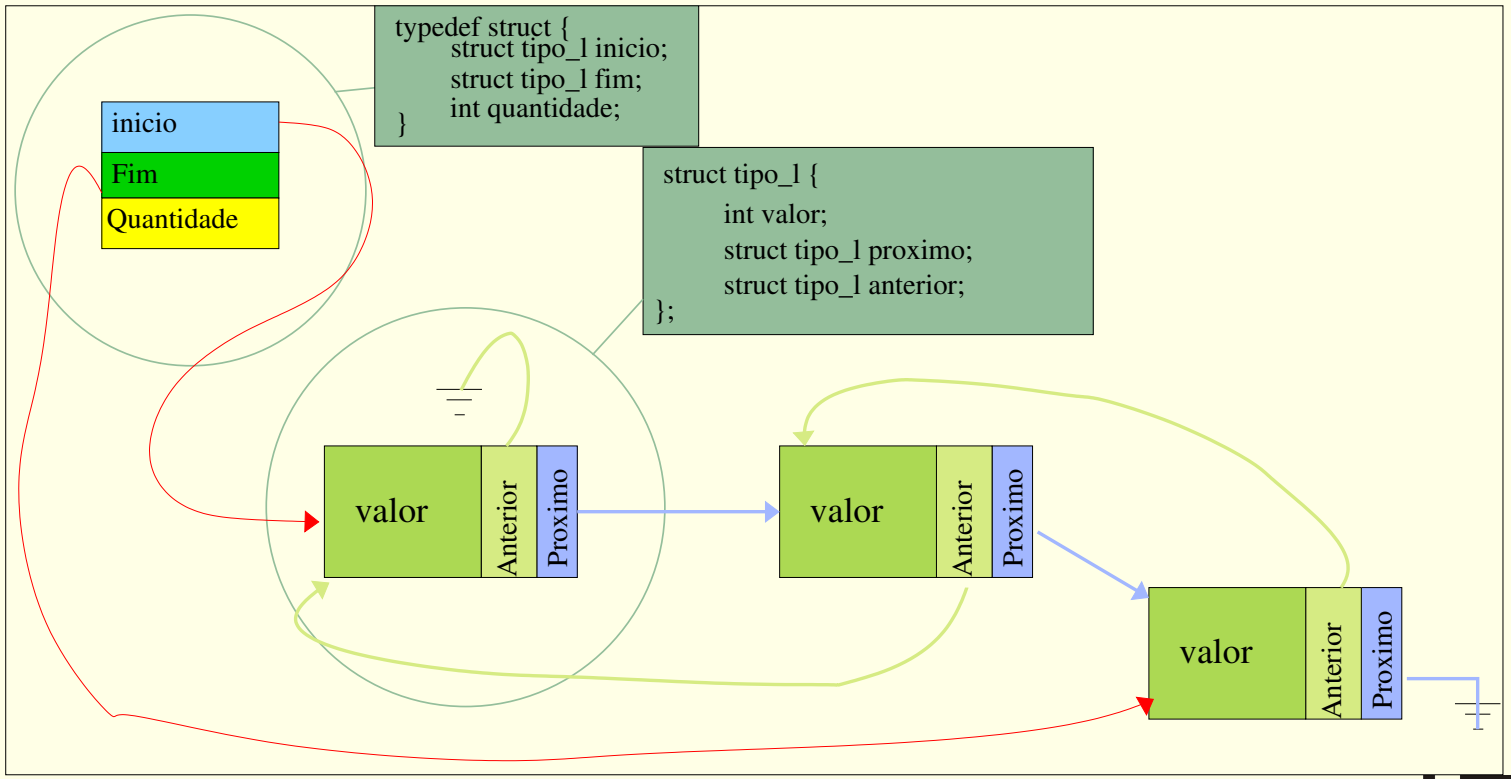
Exercício

Escreva as funções :

- `inicializa_lista(&L)` - inicializa a lista;
- `insere_elemento(&L,x)` - Insere um x elemento no final da lista;
- `mostra_lista(L)` - Mostra os elementos da lista.

Listas duplamente encadeadas

Uma outra "variação" possível para listas encadeadas são as chamadas listas duplamente encadeadas. Uma lista duplamente encadeada é uma lista encadeada que além de possuir um "ponteiro" para o próximo elemento possuí um ponteiro para o elemento anterior. Veja na figura:



Listas duplamente encadeadas

Vantagens das listas duplamente encadeadas:

- É possível percorrer a lista em qualquer direção. Se a lista estiver ordenada em ordem crescente, podemos decidir percorrê-la partindo do fim e sempre consultar o indicador `anterior`, até chegar no primeiro elemento da lista.
- Se possuirmos uma operação `pesquisa` (que retorna o "ponteiro" do elemento encontrado ou NULL se não existe elemento) podemos facilmente criar uma operação elimina. Por que ? Pois se soubermos o endereço do elemento, podemos acessar o anterior e o próximo e atualizar os ponteiros destes antes de liberar a memória.

```
1    // procura o valor 100;
2    x = pesquisa (L,100);
3    if (x != NULL ) elimina (&L,x);
```

Listas duplamente encadeadas

Desvantagens das listas duplamente encadeadas:

- Gasto de memória com ponteiros;
- Mais variáveis para atualizar quando se realiza operações de inserção/eliminação.

UTFPR

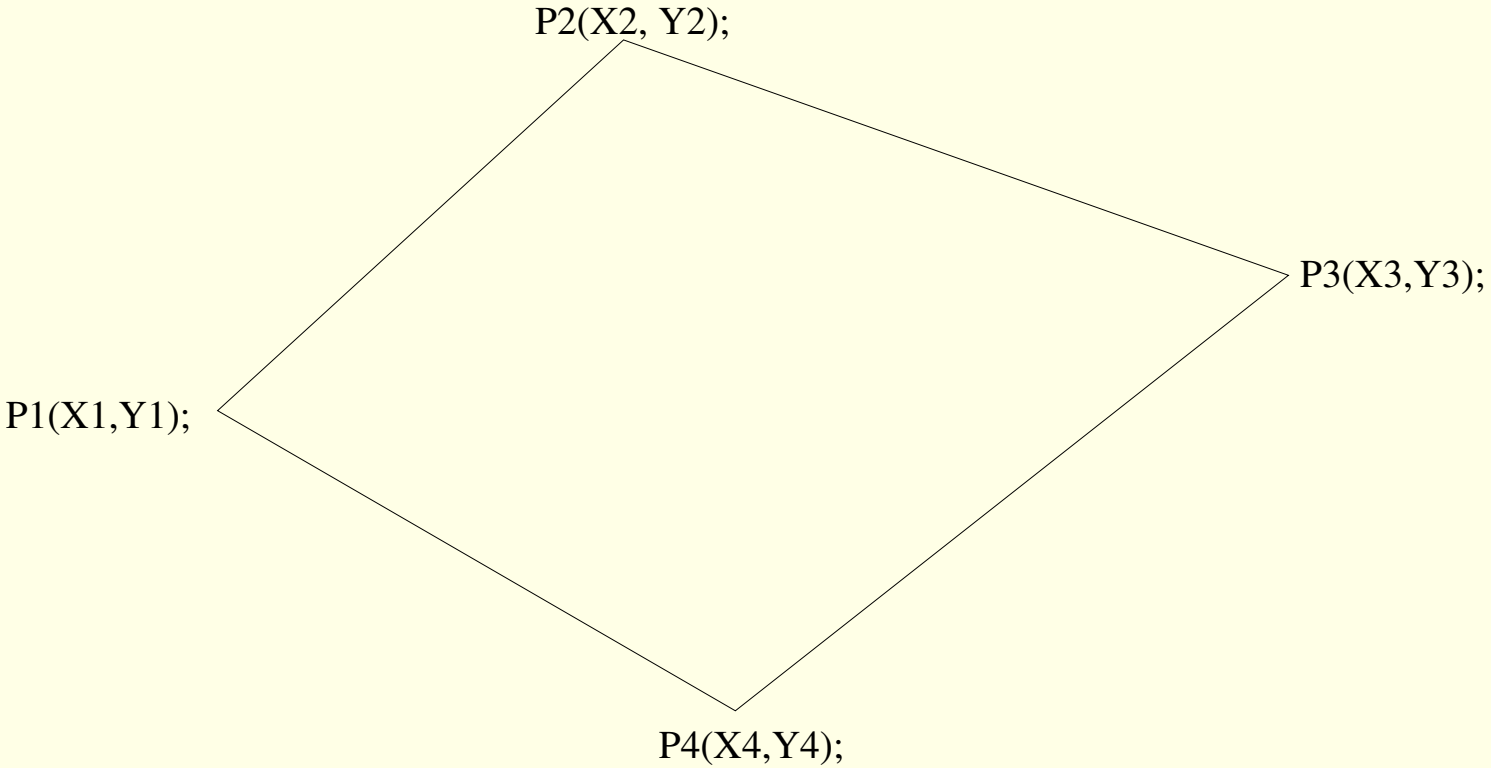
Exercício

Implemente em C as seguintes funções para trabalhar com listas duplamente encadeadas:

- `inicializa(&L);`
- `insere(&L,x);`
- `end=pesquisa (L, x);`
- `elimina (&L, end);`
- `elimina_todos_seguintes (&L,end)` Elimina todos os elementos seguintes ao elemento informado.

Exercício

Crie uma lista simplesmente encadeada para armazenar os pontos de um polígono. Crie as funções: `inicializa_poligono(&P)`, `insere_pontos_poligono(&P,x,y)` e `calcula_perímetro_polígono(P)`;



Exercício

Crie uma lista simplesmente encadeada para armazenar os pontos de uma função. Crie as funções: `inicializa_funcao(&P)`, `insere_pontos_funcao(&P,x,y)` e `calcula_integral(P, x_inicial, x_final);`