Atividade 6 - Listas Encadeadas (T1)

8 de outubro de 2025

Problema A

```c
void RetiraUltimo(TipoLista *Lista, TipoItem *Item)
{
    if(Lista == NULL || Vazia(Lista) || p->Prox == NULL){
        printf(" Erro Lista vazia\n");
        return;
    }

    TipoApontador q = Lista -> Primeiro;

    while(q->Prox != Lista-> Ultimo){
        q = q->Prox;
    }

    Retira(q, Lista, Item);
}
```

Explicação do Código: A função `RetiraUltimo` visa remover o último elemento de uma lista encadeada.

1. Primeiramente, ela verifica se a lista é nula ou vazia. Se for, exibe uma mensagem de erro e retorna.
2. Ela inicializa um ponteiro auxiliar `q` com o primeiro nó da lista (`Lista->Primeiro`).
3. Percorre a lista (`while`) até que `q` aponte para o nó **anterior** ao último elemento da lista (`q->Prox != Lista->Ultimo`).
4. Ao final do laço, o ponteiro `q` está no nó que precede o elemento que deve ser removido.
5. Finalmente, ela chama a função auxiliar `Retira(q, Lista, Item)`, que efetivamente remove o nó apontado por `q->Prox` (ou seja, o último nó).

Problema B

```
void InsereInicio(TipoItem x, TipoLista *Lista)
{
    if(Lista == NULL) return;

    TipoApontador no = (TipoApontador) malloc (sizeof(TipoCelula));
    no->Item = x;

    if(!Vazia(*Lista)){
        TipoApontador aux;
        aux = Lista->Primeiro->Prox;
        Lista->Primeiro->Prox = no;
        no->Prox = aux;
    } else {
        Lista->Primeiro->Prox = no;
        Lista->Ultimo = Lista->Primeiro->Prox;
    }
}
```

Explicação do Código: A função `InsereInicio` insere um novo elemento (`x`) no início da lista encadeada, levando em conta que a lista possui um nó cabeça.

1 **Alocação e Inicialização:** Um novo nó é alocado dinamicamente com `malloc` e o item `x` é armazenado nele.

2 **Lista Não Vazia:** Se a lista não estiver vazia:

- O ponteiro `aux` guarda o endereço do **primeiro elemento real** da lista (`Lista->Primeiro->Prox`).
- O novo nó (`no`) é inserido logo após o nó cabeça: `Lista->Primeiro->Prox = no`.
- O campo `Prox` do novo nó (`no->Prox`) passa a apontar para o antigo primeiro elemento, que estava em `aux`.

3 **Lista Vazia:** Se a lista estiver vazia:

- O novo nó (`no`) é inserido logo após o nó cabeça: `Lista->Primeiro->Prox = no`.
- O campo `Ultimo` da lista é atualizado para apontar para o novo nó, que é agora o único e último elemento.

Problema C

```c
int Tamanho(TipoLista *Lista)
{
    if(Lista == NULL || Vazia(*Lista)) return 0;

    int tamanho = 0;

    TipoApontador aux = Lista->Primeiro->Prox;
    while(aux != NULL){
        aux = aux->Prox;
        tamanho += 1;
    }

    return tamanho;
}
```

Explicação do Código : A função `Tamanho` calcula o número de elementos (nós) na lista encadeada.

1. **Casos Base:** Verifica se a lista é nula ou vazia. Em ambos os casos, retorna 0.
2. **Inicialização:** Inicializa o contador `tamanho` em 0. O ponteiro auxiliar `aux` é configurado para apontar para o primeiro elemento real da lista (`Lista->Primeiro->Prox`), ignorando o nó cabeça.
3. **Contagem:** O laço `while` itera enquanto `aux` não for `NULL` (o que marca o fim da lista).
4. **Atualização:** Em cada iteração, `aux` avança para o próximo nó (`aux = aux->Prox`), e o contador `tamanho` é incrementado em 1.
5. **Retorno:** Após percorrer toda a lista, o valor final de `tamanho` é retornado.

Problema D

```c
void ImprimeElemento(TipoLista *Lista, int i)
{
    if(Lista == NULL || Vazia(*Lista)) return;

    int j = 0;

    TipoApontador it = Lista->Primeiro;

    while(j < i && it != NULL){
        it = it->Prox;
        j++;
    }

    printf("%d\n", it->Item.Chave);

    return;
}
```

Explicação do Código: A função `ImprimeElemento` busca e imprime a chave do elemento na posição 'i' (índice) da lista encadeada. (Observação: O código assume que a posição i é um índice válido e que o primeiro elemento real tem índice $i = 1$).

1. **Verificação:** Retorna se a lista for nula ou vazia.
2. **Inicialização:** Inicializa o contador de índice `j` em 0. O ponteiro iterador `it` começa no nó cabeça (`Lista->Primeiro`).
3. **Busca por Posição:** O laço `while` avança o ponteiro `it` para o próximo nó e incrementa `j` até que o índice `j` atinja a posição desejada `i` ou o fim da lista (`it != NULL`).
4. **Impressão:** Assume-se que, ao sair do laço, `it` aponta para o nó na posição i. A chave desse item (`it->Item.Chave`) é impressa.

Problema E

```c
void ImprimeInvertido(TipoLista *Lista)
{
    if(Lista == NULL || Vazia(*Lista)) return;

    int tam = Tamanho(Lista);
    int i = tam;
    int vetor[tam];

    TipoApontador aux = Lista->Primeiro->Prox;

    while(aux != NULL){
        vetor[--i] = aux->Item.Chave;
        aux = aux->Prox;
    }

    for(int i = 0; i < tam; ++i){
        printf("%d ", vetor[i]);
    }
    printf("\n");

    return;
}
```

Explicação do Código: A função `ImprimeInvertido` imprime os elementos de uma lista encadeada na ordem inversa, utilizando um vetor auxiliar.

1 **Verificação e Tamanho:** Verifica se a lista é nula ou vazia. Obtém o tamanho da lista usando a função `Tamanho` implementada anteriormente.

2 **Estruturas Auxiliares:** Declara um vetor (`vetor`) com o tamanho exato da lista. Inicializa um índice `i` com o valor do tamanho.

3 **Armazenamento no Vetor:** O laço `while` percorre a lista do início ao fim.

- O elemento é armazenado no vetor, mas o índice `i` é decrementado antes da atribuição (`vetor[-i]`). Isso faz com que o primeiro elemento da lista seja colocado na última posição do vetor (índice $\texttt{tam} - 1$), o segundo na penúltima, e assim por diante.

4 **Impressão Invertida:** O laço `for` percorre o vetor `vetor` do índice 0 até $\texttt{tam} - 1$. Devido ao modo como os elementos foram inseridos no passo anterior, imprimir o vetor do início ao fim resulta na impressão dos elementos da lista na ordem inversa.