Ponteiros

1) Defina ponteiro, qual a sua importância? Cite um exemplo da sua aplicação.

Variável que armazena o endereço de memória.

- Permite manipulação indireta de dados.
- Facilita alocação dinâmica de memória.
- Otimiza passagem de parâmetros em funções (evita cópias).
- Essencial para estruturas dinâmicas (listas, árvores, grafos).

Casos de uso: Troca de valores entre variáveis, manipulação direta de endereços de memória em funções.

2) Diferencie memória alocada estaticamente de memória alocada dinamicamente.

Memória Estática

- Alocada em tempo de compilação.
- Tamanho fixo e predefinido (ex: arrays estáticos).
- Liberada automaticamente (variáveis locais: ao sair do escopo; globais: no fim do programa).
- Armazenada na stack.

Memória Dinâmica

- Alocada em tempo de execução (via malloc, calloc, etc.).
- Tamanho flexível e definido durante a execução.
- Liberação manual obrigatória (com free), sob risco de vazamentos.
- Armazenada no heap.

3) Comente sobre as funções MALLOC e FREE.

malloc:

- Aloca memória no heap durante a execução (dinamicamente).
- void* malloc(size_t tamanho); (tamanho em bytes).

Características:

- Retorna um ponteiro genérico (void*), exigindo cast para o tipo desejado.
- Não inicializa a memória (valores "lixo" presentes).
- Retorna NULL se falhar (ex: memória insuficiente).

free:

- Libera memória previamente alocada com malloc, calloc ou realloc.
- void free(void* ptr); (recebe o ponteiro alocado).

Regras críticas:

- Só use com ponteiros alocados dinamicamente.
- Nunca libere memória já liberada ou estática (comportamento indefinido).
- Após o free, o ponteiro vira "dangling" (defina como NULL para segurança).

```c
int *vetor = (int*) malloc(5 * sizeof(int)); // Aloca 5 inteiros
if (vetor == NULL) {
    // Tratar falha
}
// ... usa o vetor ...
free(vetor);
vetor = NULL; // Boa prática
```

Funções

→ todas em código

Structs

→ todas em código

Listas Encadeadas

1) Referente ao trecho de código a seguir:

```c
Elemento *NameFuncao(Lista *lt){
    Elemento *aux = lt->primeiro;
    if(aux == NULL){
        return aux;
    }else{
        if(aux->proximo == NULL){
            lt->primeiro = NULL;
            lt->ultimo = NULL;
        }else{
            lt->primeiro = lt->primeiro->proximo;
            lt->primeiro->anterior = NULL;
        }
        aux->anterior = aux->proximo = NULL;
        lt->num--;
        return aux;
    }
}
```

a) A função implementa qual funcionalidade de Lista? Dê um nome para a função.

Ela remove o primeiro elemento da lista. nome adequado: removePrimeiroElemento

b) O Código refere-se a uma função de lista simplesmente ou duplamente encadeada?

Duplamente encadeada (possui próximo e anterior)

c) Comente sobre os testes que ocorrem nas linhas 3,6,9.

Linha 3: Verifica se a lista está vazia (aux == NULL).
Linha 6: Checa se há apenas um elemento na lista (aux->proximo == NULL).
Linha 9: Atualiza o anterior do novo primeiro elemento para NULL, garantindo a integridade da lista.

d) Comente sobre a linha de código 10.

Passa a referência do primeiro elemento da lista para o próximo elemento.
E em seguida remove a referência do elemento anterior do novo primeiro elemento.

e) As linhas 10 e 11 podem ter sua ordem trocada? Comente.

Não, pois primeiro é necessário trocar o primeiro elemento da lista para o seu próximo elemento, e então remover a referência anterior desse novo primeiro elemento. Caso contrário, primerio→anterior já é null, e a referência do anterior do próximo elemento não seria anulada.

2) Referente ao trecho de código a seguir:

```
1   void nomefunção(Lista lt){
2       elemento *aux = lt.ultimo;
3       if(aux == NULL)
4           printf("\n LISTA VAZIA!");
5       else{
6           printf("\n Início da Lista!");
7           while(aux != NULL){
8               mostraElemento(*aux);
9               //insira o comando ausente
10          }
11          printf("\n Fim da Lista DE!");
12      }
13  }
```

a) A função implementa qual funcionalidade de Lista? Dê um nome para a função.

A função imprime a lista em ordem inversa (do último ao primeiro elemento).
Nome adequado: mostraListaReversa.

b) O Código refere-se a uma função de lista simplesmente ou duplamente encadeada? Comente.

Duplamente encadeada, pois a função percorre a lista a partir do último elemento (lt.ultimo) e avança para o nó anterior (aux->anterior), o que só é possível se cada nó tiver um ponteiro para o elemento anterior.

c) Defina a linha de código ausente (linha 9), para o código processar.

aux = aux->anterior;
Move a referência de aux para o elemento anterior.