RELATÓRIO - ESTRUTURA DE DADOS.
Caio Marcos Moita Mourão || Semestre 2

Estrutura do Programa

O arquivo `necio_trabalho.c` implementa operações estatísticas sobre um vetor de inteiros criado dinamicamente, utilizando ponteiros para manipulação eficiente dos dados.

1. Criação e Inicialização do Vetor

- Alocação dinâmica:
  - Função `numbers_create(int size)` aloca um vetor de inteiros com tamanho definido em tempo de execução, usando `malloc`. Retorna um ponteiro para o início do vetor, permitindo acesso e manipulação dos elementos via ponteiro.
- Preenchimento:
  - Função `numbers_read(int *numbers, int size)` preenche o vetor com valores aleatórios de 0 a 99, usando um laço. O ponteiro permite modificar diretamente o vetor alocado na memória.

2. Exibição dos Valores

- Função `numbers_show`:
  - Imprime todos os valores do vetor em formato de lista, facilitando a visualização dos dados gerados.

3. Operações

- Média: `numbers_average(int *numbers, int size)` percorre o vetor, soma os elementos e retorna a média.
- Maior valor: `maior_num(int *numbers, int size)` retorna o maior elemento do vetor.
- Menor valor: `menor_num(int *numbers, int size)` retorna o menor elemento do vetor.
- Números pares: `num_pares(int *numbers, int size)` conta e exibe os pares.
- Números ímpares: `num_impares(int *numbers, int size)` conta e exibe os ímpares.
- Múltiplos de cinco: `multiplos_de_cinco(int *numbers, int size)` conta e exibe os múltiplos de cinco.

Todas essas funções recebem o vetor por ponteiro, acessam seus elementos e retornam estatísticas ou imprimem resultados.

4. Liberação de Memória

- Função `numbers_destroy(int *numbers)`:
    - Libera o espaço do vetor criado, evitando desperdício de memória.

Uso dos Ponteiros

Todas as operações principais usam ponteiros para acessar e modificar o vetor. Ao passar `numbers` (do tipo `int *`) para uma função, ela acessa e altera o vetor original alocado, sem criar cópias desnecessárias. Isso torna o programa eficiente e permite manipulação direta dos dados na memória.

Trecho das Funções e Explicações

1. `int *numbers_create(int size)`

- Objetivo: Alocar dinamicamente um vetor de inteiros com tamanho definido em tempo de execução.
- Como funciona:
    - `int *num = NULL;` — Declara um ponteiro para inteiro, inicializado como `NULL`.
    - `num = (int *)malloc(size * sizeof(int));` — Usa `malloc` para reservar espaço suficiente para `size` inteiros. O resultado é convertido para ponteiro de inteiro. Se, por exemplo, `size` for 10, serão reservados 10 × 4 bytes (em sistemas onde `int` tem 4 bytes).
    - `if (num == NULL) return NULL;` — Verifica se a alocação falhou (sem memória suficiente). Se sim, retorna `NULL`.
    - `return num;` — Retorna o ponteiro para o início do vetor alocado.
- Por que usar? Permite criar vetores de tamanho variável, economizando memória e evitando desperdício. O vetor existe enquanto for necessário e pode ser manipulado por ponteiros em outras funções.

---

2. `void numbers_read(int *numbers, int size)`

- Objetivo: Preencher o vetor com valores aleatórios entre 0 e 99.
- Como funciona:
  - `if (numbers != NULL) { ... }` — Só executa se o vetor foi alocado corretamente.
  - `for (int i = 0; i < size; i++) numbers[i] = rand() % 100;` — Percorre cada posição do vetor e atribui um valor aleatório.
- Por que usar ponteiro? Permite modificar diretamente o vetor original, sem cópias. O ponteiro aponta para o bloco de memória alocado, e cada posição pode ser acessada por índice ou aritmética de ponteiros.

---

3. `float numbers_average(int *numbers, int size)`

- Objetivo: Calcular a média dos valores do vetor.
- Como funciona:
  - `float sum = 0.0;` — Inicializa a soma dos elementos.
  - `if (numbers != NULL) for (int i = 0; i < size; i++) sum += numbers[i];` — Soma todos os valores do vetor.
  - `if (sum != 0) return sum / (float)size;` — Se a soma não for zero, retorna a média.
  - `return 0;` — Se a soma for zero, retorna zero.
- Por que usar ponteiro? Permite acessar todos os elementos do vetor criado dinamicamente, sem precisar copiar dados.

---

4. `int maior_num(int *numbers, int size)`

- Objetivo: Encontrar o maior valor do vetor.
- Como funciona:
  - `int maior = numbers;` — Inicializa o maior valor com o primeiro elemento.
  - `for (int i = 1; i < size; i++) { if (numbers[i] > maior) maior = numbers[i]; }` — Percorre o vetor, atualizando o maior valor encontrado.
  - `return maior;` — Retorna o maior valor.
- Por que usar ponteiro? Permite acessar cada elemento do vetor alocado.

---

5. `int menor_num(int *numbers, int size)`

- Objetivo: Encontrar o menor valor do vetor.
- Como funciona:
  - `int menor = numbers;` — Inicializa o menor valor com o primeiro elemento.
  - `for (int i = 1; i < size; i++) { if (numbers[i] < menor) menor = numbers[i]; }` — Percorre o vetor, atualizando o menor valor encontrado.
  - `return menor;` — Retorna o menor valor.
- Por que usar ponteiro? Permite acessar cada elemento do vetor alocado.

---

6. `int num_pares(int *numbers, int size)`

- Objetivo: Contar e exibir os números pares do vetor.
- Como funciona:
  - `int pares = 0;` — Inicializa o contador de pares.
  - `printf("Numeros Pares: [");` — Inicia a impressão dos pares.
  - `for(int i = 0; i < size; i++){ if(numbers[i] % 2 == 0){ printf("%d ", numbers[i]); pares++; } }` — Percorre o vetor, imprime e conta os pares.
  - `printf("]\n");` — Fecha a lista impressa.
  - `return pares;` — Retorna a quantidade de pares.
- Por que usar ponteiro? Permite acessar e modificar o vetor original.

---

7. `int num_impares(int *numbers, int size)`

- Objetivo: Contar e exibir os números ímpares do vetor.
- Como funciona:
  - `int impares = 0;` — Inicializa o contador de ímpares.
  - `printf("Numeros Impares: [");` — Inicia a impressão dos ímpares.
  - `for(int i = 0; i < size; i++){ if(numbers[i] % 2 != 0){ printf("%d ", numbers[i]); impares++; } }` — Percorre o vetor, imprime e conta os ímpares.
  - `printf("]\n");` — Fecha a lista impressa.
  - `return impares;` — Retorna a quantidade de ímpares.
- Por que usar ponteiro? Permite acessar e modificar o vetor original.

---

8. `int multiplos_de_cinco(int *numbers, int size)`

- Objetivo: Contar e exibir os múltiplos de cinco do vetor.
- Como funciona:
  - `int multiplos = 0;` — Inicializa o contador de múltiplos de cinco.
  - `printf("Multiplos de 5: [");` — Inicia a impressão dos múltiplos.
  - `for (int i = 0; i < size; i++) { if (numbers[i] % 5 == 0) { printf("%d ", numbers[i]); multiplos++; } }` — Percorre o vetor, imprime e conta os múltiplos de cinco.
  - `printf("]\n");` — Fecha a lista impressa.
  - `return multiplos;` — Retorna a quantidade de múltiplos de cinco.
- Por que usar ponteiro? Permite acessar e modificar o vetor original.

---

9. `void numbers_show(int *numbers, int size)`

- Objetivo: Exibir todos os valores do vetor em formato de lista.
- Como funciona:
  - `printf("Todos os numeros: [");` — Inicia a impressão do vetor.
  - `if (numbers != NULL) { for (int i = 0; i < size; i++) if (i == (size - 1)) printf("%d", numbers[i]); else printf("%d, ", numbers[i]); }` — Percorre o vetor, imprime cada valor, separando por vírgula.
  - `printf("]\n");` — Fecha a lista impressa.
- Por que usar ponteiro? Permite acessar todos os elementos do vetor alocado.

---

10. `void numbers_destroy(int *numbers)`

- Objetivo: Liberar a memória alocada para o vetor.
- Como funciona:
  - `free(numbers);` — Libera o espaço reservado por `malloc`, evitando desperdício de memória.
- Por que usar ponteiro? O ponteiro aponta para o bloco de memória que deve ser liberado.

Fluxo no `main`

1. Gera tamanho aleatório para o vetor.
2. Cria o vetor dinamicamente usando ponteiro.
3. Preenche o vetor com valores aleatórios.
4. Exibe o vetor.
5. Calcula média, maior, menor, pares, ímpares e múltiplos de 5.
6. Exibe todos os resultados.
7. Libera a memória usada pelo vetor.

Endereço do Repositório

RockyzFX/Ponteiros-AlocacaoDinamica

Arquivo analisado: `necio_trabalho.c`