Estruturas de Dados - Struct, Ponteiro e Recursividade

Aula passada

O que vimos:

- Funções
- Protótipo e assinatura de uma função
- Sobrecarga
- Escopo de uma variável
  - variável local
  - variável global
- Passagem de parâmetros
  - por valor
  - por referência

Ponteiros

- Ponteiro:
  - tipo de variável que aponta para uma outra variável ou estrutura
  - ideia de apontar para a posição de memória da variável
  - C:
    - **int** *ponteiro = malloc(sizeof(**int**));
    - **int** *ponteiro = malloc(n*sizeof(**int**));

Estruturas/registros

- Registro:
  - variável que pode ser composta por campos de tipos diferentes
  - C:

```
struct no {
        int chave;
        struct no *proximo;
};
```

  - pseudocódigo:

```
estrutura no {
        inteiro chave
        no *proximo
}
```

Estruturas/registros

- Como acessar um registro?
  - C:
    - x = no.chave;
    - y = no.proximo;
  - pseudocódigo:
    - x = no.chave
    - y = no.proximo
- "." é um operador de acesso do registro

Vetores

- Vetor:
  - é uma estrutura composta, homogênea e unidimensional
  - representa uma sequência de variáveis do mesmo tipo de dados e com o mesmo identificador
    - utiliza-se um índice para acessar cada variável individualmente
  - suas posições são alocadas sequencialmente na memória
    - permite um rápido acesso a qualquer das posições

Vetores

- Como declarar um vetor?
  - C: tipo da variável + identificador + tamanho
    - **int** vetor[11];
  - pseudocódigo: tipo da variável + identificador + índices inicial e final
    - inteiro vetor[0..10]
    - indicar índices inicial e final em vez do tamanho pode facilitar a escrita do pseudocódigo!

Vetores

- Como acessar um vetor?
- Identificador + índice relativo à posição a ser acessada
  - C:
    - $V[3] = 10;$
    - $x = V[3];$
  - pseudocódigo:
    - $V[3] = 10$
    - $x = V[3]$

Matrizes

- Matriz:
  - é uma estrutura composta, homogênea e multidimensional
  - representa uma sequência de variáveis do mesmo tipo de dados e com o mesmo identificador
    - utiliza-se de índices (um para cada uma das dimensões da matriz) para acessar cada variável individualmente
  - assim como no caso dos vetores, tem um rápido acesso a uma dada posição

Matrizes

- Como declarar uma matriz?
  - C: tipo da variável + identificador + tamanho de cada uma das dimensões
    - **int** matriz[3][2];
  - pseudocódigo: tipo da variável + identificador + índices inicial e final de cada uma das dimensões
    - inteiro matriz[1..3,1..2]
    - inteiro matriz[1..3][1..2]

Matrizes

- Como acessar uma matriz?
- Identificador + índices relativos à posição a ser acessada
  - C:
    - $M[3][2] = 10;$
    - $x = M[3][2];$
  - pseudocódigo:
    - $M[3][2] = 10$
    - $M[3, 2] = 10$
    - $x = M[3][2]$
    - $x = M[3, 2]$

Recursividade

- Recursividade:
  - também é chamada de recursão ou recorrência
  - mecanismo de programação no qual uma definição de uma objeto refere-se ao próprio objeto que está sendo definido
  - função recursiva: função que é definida em termos de si mesma

Recursividade

- Divisão e conquista: estratégia para definir uma função recursiva
  - dividir o problema em problemas menores do mesmo tipo
  - resolver os problemas menores (!)
  - combinar as soluções dos problemas menores para obter a solução final
- A divisão sucessiva em problemas menores eventualmente leva a algum caso base
  - casos base são os casos mais simples possíveis, portanto não podem ser divididos
  - suas soluções devem ser explicitamente definidas

Recursividade

- Podemos resumir uma definição de função recursiva em duas etapas:
  - definir os casos base
    - o que fazer nas situações simples
    - a solução para esses casos deve ser dada diretamente, sem envolver chamadas recursivas
    - pode ser tratado como uma condição de parada da recursão
  - definir os casos recursivos
    - representam os casos mais gerais, mais complexos
    - faz chamadas a si mesma para casos menores
    - ao fazermos estas chamadas recursivas para casos menores, sempre consideraremos que cada chamada resolve o problema menor
- Obs: formalizar a recursão matematicamente auxilia na hora de escrever o código!

Recursividade: fatorial

$$F(n) = \begin{cases} 1, & \text{se } n = 0 \\ n \cdot F(n-1), & \text{caso contrário} \end{cases}$$

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 se n == 0 então
2 |   retorne 1
3 retorne n · Fatorial(n − 1)
```

Recursividade: número de Fibonacci

$$Fib(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ Fib(n-1) + Fib(n-2), & \text{caso contrário} \end{cases}$$

Algoritmo: Fibonacci(n)

Entrada: natural n

Saída: valor do n-ésimo número de Fibonacci

```
1 se n ≤ 1 então
2 |   retorne n
3 retorne Fibonacci(n − 1) + Fibonacci(n − 2)
```

Torre de Hanoi

Problema da Torre de Hanoi:

- n discos de tamanhos diferentes
- Os discos estão empilhados em uma torre de origem de acordo com os seus tamanhos (maiores embaixo, menores em cima)
- Três torres: origem, trabalho e destino
- Objetivo: passar todos os discos da torre de origem para a torre de destino utilizando a torre de trabalho como auxiliar, e obedecendo às seguintes regras:
  - só um disco pode ser movido por vez
  - cada movimento consiste em tirar o disco do topo de uma torre e mover pro topo de outra torre (não é permitido mover um disco no meio de uma pilha, ou mover um disco para o meio da pilha)
  - nenhum disco pode ser colocado sobre um disco de menor tamanho

Torre de Hanoi

Análise do problema:

- Casos específicos:
  - movimentos para 1 disco
  - movimentos para 2 discos
  - movimentos para 3 discos
  - movimentos para 4 discos
  - movimentos para 5 discos
  - ⋮
- Caso geral:
  - movimentos para n discos?

Torre de Hanoi

Ideia:

- Para movermos $n \geq 1$ discos de origem para destino utilizando trabalho, fazemos:
  - movemos $n-1$ discos de Origem para Trabalho usando Destino
  - movemos o n-ésimo disco de Origem para Destino
  - movemos $n-1$ discos de Trabalho para Destino usando Origem
- Caso base?
- Chamadas recursivas?

Função recursiva: Torre de Hanoi

Algoritmo: Hanoi(n, O, D, T)

Entrada: tamanho n do disco a ser movido, torres de origem O, de trabalho T, e de destino D

Saída: passos para mover n discos de O para D usando T

```
1 se n == 1 então
2 |   mover disco de O para D
3 senão
4 |   Hanoi(n − 1, O, T, D)
5 |   mover disco de O para D
6 |   Hanoi(n − 1, T, D, O)
```