Programação

Flavius Gorgônio
flavius@dct.ufrn.br

Agenda

- Estruturas de dados compostas homogêneas (arrays)
  - Unidimensionais: vetores
  - Bidimensionais: matrizes
  - Multidimensionais
- Passagem de parâmetros de arrays para funções
- Vetores e ponteiros
- Aplicações

Vetores

Estruturas de armazenamento de dados homogêneos unidimensionais

Uma lista de coisas a fazer pode ser armazenada, por exemplo, em um vetor

Estruturas de dados compostas homogêneas

Uma variável do tipo *array* é uma estrutura de dados indexada que pode armazenar uma quantidade pré-determinada de valores do mesmo tipo.

Os dados armazenados em um vetor são chamados de itens do vetor. Para localizar a posição de um item em um vetor, utiliza-se um número inteiro denominado índice do vetor.

Pode ser vista como um mapeamento de um conjunto de índices para um conjunto de variáveis adjacentes. O conjunto de índices é normalmente um subintervalo de valores consecutivos.

14	**37**	**18**	**92**	**77**	**...**	**64**
0	1	2	3	4	...	n-1

Estruturas de dados compostas homogêneas

Tipos de vetores e matrizes:

Estático: o conjunto de índices é definido antes da compilação, (por exemplo, na linguagem Pascal).

Dinâmico: o conjunto de índices é definido na criação da variável, ou seja, em tempo de execução, pela avaliação de uma expressão, (por exemplo, na linguagem Ada)

Flexível: o conjunto de índices não é fixo, assim, os limites do conjunto de índices podem ser alterados sempre que um novo valor de array for associado a variável, (por exemplo, nas linguagens Java e Python)

Declaração de vetores na linguagem C

```c
int vet[10];                        // vetor para 10 valores inteiros

int i, j, k[100];                   // dois escalares e um vetor

float temp[24];                     // vetor de float

double area[6], pi = 3.1415926;     // vetor de double e escalar double

int v1[5] = {12, 23, 34, 45, 56};   // inicialização do vetor

int v2[] = {12, 23, 34, 45, 56};    // inicializa sem definir tamanho

int v3[10] = {12, 23, 34, 45, 56};  // inicializa parcialmente

float tempExterna[24] = {0.0f};     // inicializa com valor
```

Cálculo da temperatura média

Escreva um programa em linguagem C que leia um conjunto de *n* valores de temperatura, onde o valor de *n* é informado pelo usuário. Em seguida, o programa deverá calcular e exibir o valor da temperatura média, em graus Celsius.

A média de um conjunto de *n* valores x_i é definida como:

$$m = \frac{\sum_{i=1}^{n} x_i}{n}$$

Cálculo de temperatura média

```c
#include <stdio.h>
int main(void) {
  int n;
  float temp, med = 0.0f;
  printf("Programa Cálculo de Temperatura Média \n");
  printf("Quantos valores de temperatura? ");
  scanf("%d", &n);
  printf("Informe os valores coletados\n");
  for (int i = 0; i < n; i++) {
    printf("Temp %d: ", i+1);
    scanf("%f", &temp);
    med += temp;
  }
  med = med / n;
  printf("Temperatura média: %.2f °C\n", med);
  return 0;
}
```

https://repl.it/@flaviusgorgonio/mediaSemVetorc

Cálculo de temperatura média e variância

Modifique o programa anterior de forma que ele calcule a média e a variância do conjunto de *n* valores de temperatura. Ao final, o programa deverá exibir o valor da temperatura média e da variância.

A média *m* e a variância *v* de um conjunto de *n* valores x_i são definidas como:

$$m = \frac{\sum_{i=1}^{n} x_i}{n} \qquad v = \frac{\sum_{i=1}^{n} (x_i - m)^2}{n}$$

Cálculo de temperatura média e variância

```c
#include <stdio.h>
#define N 1000
int main(void)
{
  int n;
  float med, var;
  float vetTemp[N];
  med = 0.0f;
  var = 0.0f;

  printf("Programa Cálculo de Temperatura Média \n");
  printf("Quantos valores de temperatura (máx. 1000)? ");
  scanf("%d", &n);
  if (n > N) {
    printf("Valor excede o limite de %d!\n", N);
    return 1;
  }
  printf("Informe os valores coletados\n");
  for (int i = 0; i < n; i++) {
    printf("Temp %d: ", i+1);
    scanf("%f", &vetTemp[i]);
  }

  for (int i = 0; i < n; i++) {
    med += vetTemp[i];
  }
  med = med / n;
  printf("Temperatura média: %.2f\n", med);

  for (int i = 0; i < n; i++) {
    var += ((vetTemp[i]-med) * (vetTemp[i]-med));
  }
  var = var / n;
  printf("Variância de temperatura: %.2f\n", var);
  return 0;
}
```

https://repl.it/@flaviusgorgonio/mediaEVarianciaSemFuncoesc#main.c

Cálculo de temperatura média e variância

Como seria uma versão que utilizasse funções?

- ➢ A solução precisa ser modularizada
- ➢ Deve-se considerar a necessidade de passagem de parâmetros entre as funções
- ➢ As funções devem, na medida do possível, poderem ser reaproveitadas em outros programas

Quais módulos poderiam ser implementados?

1. Leitura dos itens de um vetor (p.ex., para ler os valores de temperaturas)
2. Exibição dos itens de um vetor (p.ex., para exibir os valores de temperaturas armazenados)
3. Soma dos valores de um vetor
4. Cálculo da média do valores de um vetor
5. Cálculo da variância dos valores de um vetor

Função para leitura das temperaturas

```c
#include <stdio.h>
#define N 1000

void leTemperaturas(int n, float* v);

int main(void)
{
 int n;
 float med, var;
 float vetTemp[N];
 med = 0.0f;
 var = 0.0f;
 printf("Programa Cálculo de Temperatura Média \n");
 printf("Quantos valores de temperatura (máx. 1000)? ");
 scanf("%d", &n);
 if (n > N) {
   printf("Valor excede o limite de %d!\n", N);
   return 1;
 }
 leTemperaturas(n, vetTemp);
 for (int i = 0; i < n; i++) {
   med += vetTemp[i];
 }
 med = med / n;
 printf("Temperatura média: %.2f\n", med);
 for (int i = 0; i < n; i++) {
   var += ((vetTemp[i]-med) * (vetTemp[i]-med));
 }
 var = var / n;
 printf("Variância de temperatura: %.2f\n", var);
 return 0;
}

void leTemperaturas(int n, float* v) {
 printf("Informe os valores coletados\n");
 for (int i = 0; i < n; i++){
   printf("Temp %d: ", i+1);
   scanf("%f", &v[i]);
 }
}
```

https://repl.it/@flaviusgorgonio/mediaEVarianciaComFuncoes1c#main.c

Funções para média e variância

```c
#include <stdio.h>
#define N 1000

void leTemperaturas(int, float*);
float calculaMedia(int, float*);
float calculaVariancia(int, float*, float);


int main(void)
{
 int n;
 float med, var;
 float vetTemp[N];
 med = 0.0f;
 var = 0.0f;
 ...
}
```

```c
float calculaMedia(int n, float* v) {
 float media = 0.0f;

 for (int i = 0; i < n; i++) {
   media += v[i];
 }
 return (media / n);
}

float calculaVariancia(int n, float* v, float med)
{
 float var = 0.0f;

 for (int i = 0; i < n; i++) {
   var += ((v[i]-med) * (v[i]-med));
 }
 return (var / n);
}
```

https://repl.it/@flaviusgorgonio/mediaEVarianciaComFuncoes3c#main.c

Vetores e ponteiros

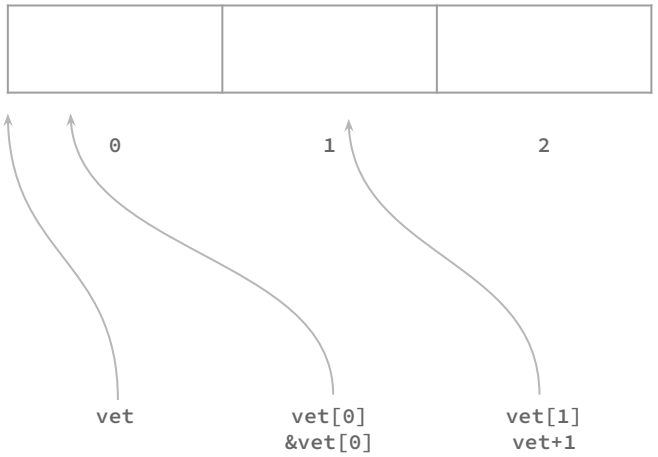
Na declaração do vetor:

```
int vet[10];
```

O símbolo **vet** (que representa o vetor) é, na verdade, uma constante que armazena o valor do endereço inicial do vetor

Assim, **vet** ou **vet[0]** representam o mesmo endereço de memória, o endereço do primeiro elemento do vetor

O tipo de uma variável vetor é, na verdade, um ponteiro para o tipo de elemento que o vetor armazena



Vetores e ponteiros

```c
#include <stdio.h>

int main(void) {
  int vet[3];
  int *p1 = vet;
  int *p2 = &vet[0];
  int *p3 = &vet[1];
  int *p4 = &vet[2];
  printf("Vetores e endereços\n");
  printf("Endereço de vet: %p\n", vet);
  printf("Valor de p1: %p\n", p1);
  printf("Valor de p2: %p\n", p2);
  printf("Valor de p3: %p\n", p3);
  printf("Valor de p4: %p\n", p4);
  vet[0] = 100;
  vet[1] = 200;
  vet[2] = 300;
  printf("Vetor = [%4d %4d %4d]\n", vet[0], vet[1], vet[2]);
  p1[0] = 200;
  printf("Vetor = [%4d %4d %4d]\n", vet[0], vet[1], vet[2]);
  p2[0] = 300;
  printf("Vetor = [%4d %4d %4d]\n", vet[0], vet[1], vet[2]);
  p3[0] = 400;
  printf("Vetor = [%4d %4d %4d]\n", vet[0], vet[1], vet[2]);
  p4[0] = 500;
  printf("Vetor = [%4d %4d %4d]\n", vet[0], vet[1], vet[2]);
  return 0;
}
```

https://repl.it/@flaviusgorgonio/vetoresEPonteirosc#main.c

Possíveis problemas com alocação de memória

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void preencheVetor(int, int*);
void exibeVetor(int, int*);


void preencheVetor(int n, int* v) {
 for (int i = 0; i < n; i++) {
   v[i] = rand() % 100;
 }
}

void exibeVetor(int n, int* v) {
 for (int i = 0; i < n; i++) {
   printf("%02d\t", v[i]);
 }
}
```

```c
int main(void) {
 int v[10];
 int tam;

 srand(time(NULL));
 printf("Problemas com uso de memória em C\n");
 printf("\n");
 printf("Preenchendo o vetor\n");
 printf("Quantos elementos deseja preencher(max. 10)? ");
 scanf("%d", &tam);
 preencheVetor(tam, v);
 printf("Vetor criado com sucesso!\n");
 printf("\n");
 printf("Exibindo o vetor\n");
 printf("Quantos elementos deseja exibir(max. 10)? ");
 scanf("%d", &tam);
 exibeVetor(tam, v);
 printf("\n");
 return 0;
}
```

Teste com valores maiores que 10

Teste com valores maiores que 10

https://repl.it/@flaviusgorgonio/problemasComAlocacaoDeMemoriac#main.c

Uso de vetores para representação de nadadeiras

Baleias e golfinhos possuem marcas naturais na nadadeira dorsal que podem ser comparadas com impressões digitais

É possível fazer o reconhecimento de diferentes animais a partir dessas imagens (método não invasivo), assim o animal não precisa ser capturado ou marcado fisicamente

Após a captura da imagem, mapeia-se a nadadeira através de uma função matemática e é possível montar um catálogo de indivíduos identificados

Fonte:

http://brydesdobrasil.com.br/catalogo

https://g1.globo.com/sp/campinas-regiao/terra-da-gente/noticia/2019/06/08/pesquisadores-criam-guia-ilustrado-para-a-identificacao-de-mamiferos-marinhos.ghtml

Aplicação: Representação de Polinômios

Algumas sugestões de funções para operar sobre polinômios:

```
void preenche(int n, float *p);
float avalia(int n, float *p, float x);
int igualdade(int n, float *p1, float *p2);
void soma(int n, float *p1, float *p2, float *p3);
void produto(int n, float *p1, float *p2, float *p3);
int derivada(int n, float *p, float *d);
```

Aplicação: Representação de Polinômios

Um polinômio é uma função matemática definida da seguinte forma:

$$a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_gx^g$$

onde:

$a_0, a_1, a_2, a_3, \ldots, a_p$ são números reais (coeficientes do polinômio)

g é um número inteiro (grau do polinômio)

Portanto, um polinômio pode ser representado apenas por seus coeficientes e pelo seu grau, na forma de um vetor de coeficientes, com $g+1$ elementos.

Aplicação: Representação de Polinômios

```c
void preenche(int n, float *p) {
 printf("Entre com os coeficientes do polinômio: \n");
 for (int i=0; i<n; i++) {
   printf("a(%d): ", i);
   scanf("%f", &p[i]);
 }
}

void exibe(int n, float *p) {
printf("Polinômio: \n");
 for (int i=n-1; i>0; i--) {
   printf("%.1f x^%d + ", p[i], i);
 }
 printf("%.1f\n", p[0]);
}
```

https://repl.it/@flaviusgorgonio/AplicacaoDeVetoresPolinomios#main.c

Matrizes

Estruturas de armazenamento de dados homogêneos multidimensionais

Uma agenda de compromissos, distribuídos em dias, semanas e meses ao longo de um ano, pode ser armazenada, por exemplo, em uma matriz

Estruturas bidimensionais: Matrizes

```c
#include <stdio.h>

  int main(void) {
  int mat[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
  printf("Hello Matrizes\n");
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++)
      printf("%d\t", mat[i][j]);
    printf("\n");
  }
  return 0;
}
```

https://repl.it/@flaviusgorgonio/matrizes1c#main.c

Parâmetros e o uso de matrizes em funções

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void preenche(int m[3][3]);
void exibe(int m[3][3]);

int main(void) {
  int mat[3][3];
  srand(time(NULL));
  printf("Hello Matrizes\n");
  preenche(mat);
  exibe(mat);
  return 0;
}
```

```c
void preenche(int m[3][3]) {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      m[i][j] = rand() % 100;
    }
  }
}

void exibe(int m[3][3]) {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      printf("%02d\t", m[i][j]);
    }
    printf("\n");
  }
}
```

https://repl.it/@flaviusgorgonio/matrizes2c#main.c

Aplicação: O Jogo da Velha

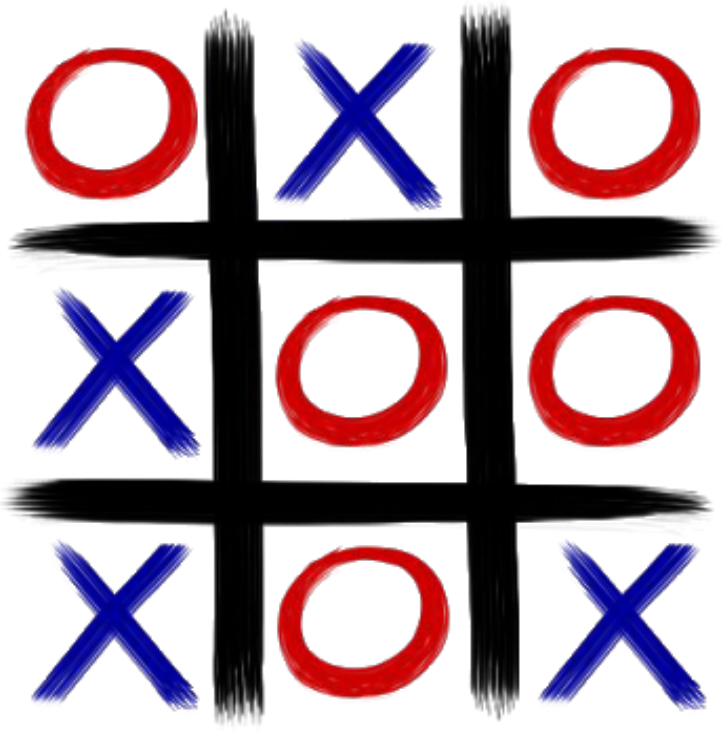
O tabuleiro é uma matriz de três linhas por três colunas

Cada jogador escolhe uma marcação, geralmente um círculo (O) e um xis (X).

Os jogadores jogam alternadamente, uma marcação por vez, numa lacuna que esteja vazia

O objetivo é conseguir três círculos ou três xis em linha, quer horizontal, vertical ou diagonal , e ao mesmo tempo, quando possível, impedir o adversário de ganhar na próxima jogada

Se os dois jogadores jogarem sempre da melhor forma, o jogo terminará sempre em empate.



Algumas funções úteis

```
// Recebe uma matriz tab com tamanho dim e preenche-a com o caracter '_'
void limpaTabuleiro(int tab[][dim], int dim);

// Exibe na tela a matriz tab com tamanho dim
void exibe(int tab[][dim], int dim);

// Lê e retorna as coordenadas correspondentes a uma jogada em um tabuleiro com tamanho dim
void jogada(int dim, int* lin, int* col);

// Recebe uma matriz tab com tamanho dim e as coordenadas de uma posição. Retorna 1 se a
posição tab[lin][col] existe e está disponível (vazia) ou 0 se não estiver
int vazio(int tab[][dim], int dim, int lin, int col);

// Recebe uma matriz tab com tamanho dim e insere o caractere 'X' na posição tab[lin][col]
void jogaX(int tab[][dim], int dim, int lin, int col);

// Recebe uma matriz tab com tamanho dim e insere o caractere '0' na posição tab[lin][col]
void joga0(int tab[][dim], int dim, int lin, int col);
```

Bibliografia Recomendada

Celes, W. *et al*. Introdução a Estruturas de Dados com Técnicas de Programação em C, 2ª ed. Rio de Janeiro: Elsevier, 2016.

Sebesta, R. W., Conceitos de Linguagens de Programação, 9ª ed. Porto Alegre: Bookman, 2016.

Varejão, F., Linguagens de Programação: Conceitos e Técnicas, Rio de Janeiro: Campus, 2004.