Introdução à Programação

Armazenamento de Grande Quantidade de Informação – Usando Vetores

Armazenando Grande Quantidade de Informação

Como armazenar tanta informação?

Vetores !

Tópicos da Aula

- Hoje, aprenderemos como armazenar grande quantidade de informação
  - Necessidade de armazenamento
  - Utilização de muitas variaveis
  - Manipulação incremental das informações
- Aprenderemos a utilizar **vetores**
  - Conceito
  - Criação
  - Inicialização
  - Acesso
  - Limites de um vetor
  - Passagem de vetores como argumentos
  - Vetores bidimensionais (matrizes)

Necessidade de Armazenamento

- Sistemas armazenam informações para que possam posteriormente acessá-las e manipulá-las
- Maioria dos sistemas requer o armazenamento de grande quantidade de valores do mesmo tipo
  - Mesmas operações para manipular estes valores
  - Ex: processamento de imagens
    - Milhões de imagens e cada imagem composta de milhões de pixels
- No desenvolvimento de um programa, devemos definir estruturas e mecanismos para armazenar e manipular esta grande quantidade de informação

Calculando a Média Aritmética

- A média aritmética de um conjunto de valores é dada pela seguinte expressão

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

Como Armazenar Grande Quantidade de Informações?

- Até agora, vimos construções de programação que permitem guardar uma única informação de cada vez
  - Variável e constante
- Neste caso, a solução seria criar uma variável para cada valor que desejamos armazenar
  - Não é uma boa solução, porém hoje veremos como podemos trabalhar com esta alternativa

Tentativa com Muitas Variáveis

2 variáveis para armazenar números

```
#include <stdio.h>
int main() {
    float numero1, numero2;
    float media;

    printf("Digite 2 numeros:\n");
    scanf("%f %f", &numero1,&numero2);
    media = (numero1 + numero2)/2;

    printf("\nA media eh %f\n",media);
    return 0;
}
```

Limitação: Só permite média de 2 números

Tentativa com Muitas Variáveis

3 variáveis para armazenar números

```
#include <stdio.h>
int main() {
    float numero1, numero2, numero3;
    float media;

    printf("Digite 3 numeros:\n");
    scanf("%f %f %f", &numero1,&numero2,&numero3);
    media = (numero1 + numero2 + numero3)/3;

    printf("\nA media eh %f\n",media);
    return 0;
}
```

Trecho de código depende da quantidade de números suportados

Limitação: Só permite média de 3 números

Problemas com Uso de Muitas Variáveis

- Dificuldade de lidar com mudança do número de informações que devem ser armazenados
  - Acrescentar ou remover variável
  - Muitas vezes requer modificações em várias linhas de código

Outra Estratégia: Manipulação Incremental da Informação

```
#include <stdio.h>
int main() {
    int qtdNumeros;
    int contador = 0;
    float numero;
    float media = 0.0;
    printf("Digite quantidade de numeros:\n");
    scanf("%d", &qtdNumeros);
    while (contador < qtdNumeros) {
      scanf("%f",&numero);
       media = media + numero;
       contador++;
    }
    media = media/qtdNumeros;
    printf("\nA media eh %f\n",media);
    return 0;
}
```

Variável que guarda a quantidade de números entrados

Cada número entrado é armazenado na variável numero

Permite média de quantidade variada de números

Outra Estratégia: Manipulação Incremental da Informação

```
#include <stdio.h>
int main() {
    int qtdNumeros;
    int contador = 0;
    float numero;
    float media = 0.0;
    printf("Digite quantidade de numeros:\n");
    scanf("%d", &qtdNumeros);
    while (contador < qtdNumeros) {
        scanf("%f",&numero);
        media = media + numero;
        contador++;
    }
    media = media/qtdNumeros;
    printf("\nA media eh %f\n", media);
    ret
}
```

Código independe da quantidade de números que serão utilizados na média

Contudo perdemos informação sobre cada número digitado!

Problemas com Manipulação Incremental de Informação

- Perda de informação sobre os valores individuais que foram armazenados
  - No exemplo, sabemos apenas qual foi o somatório dos valores inseridos e não quais valores foram inseridos
- Limita a utilização da informação armazenada
  - No exemplo, se quiséssemos utilizar os mesmos dados para uma operação diferente (ex: multiplicação), não conseguiríamos

Necessidade de Vetores

- Precisamos de alguma estrutura de armazenamento que:
  - armazene vários valores de um determinado tipo
  - permita que os valores sejam acessados de forma simples

9.0	7.5	6.3	8.8	9.8	10.0

Vetores !

Vetores

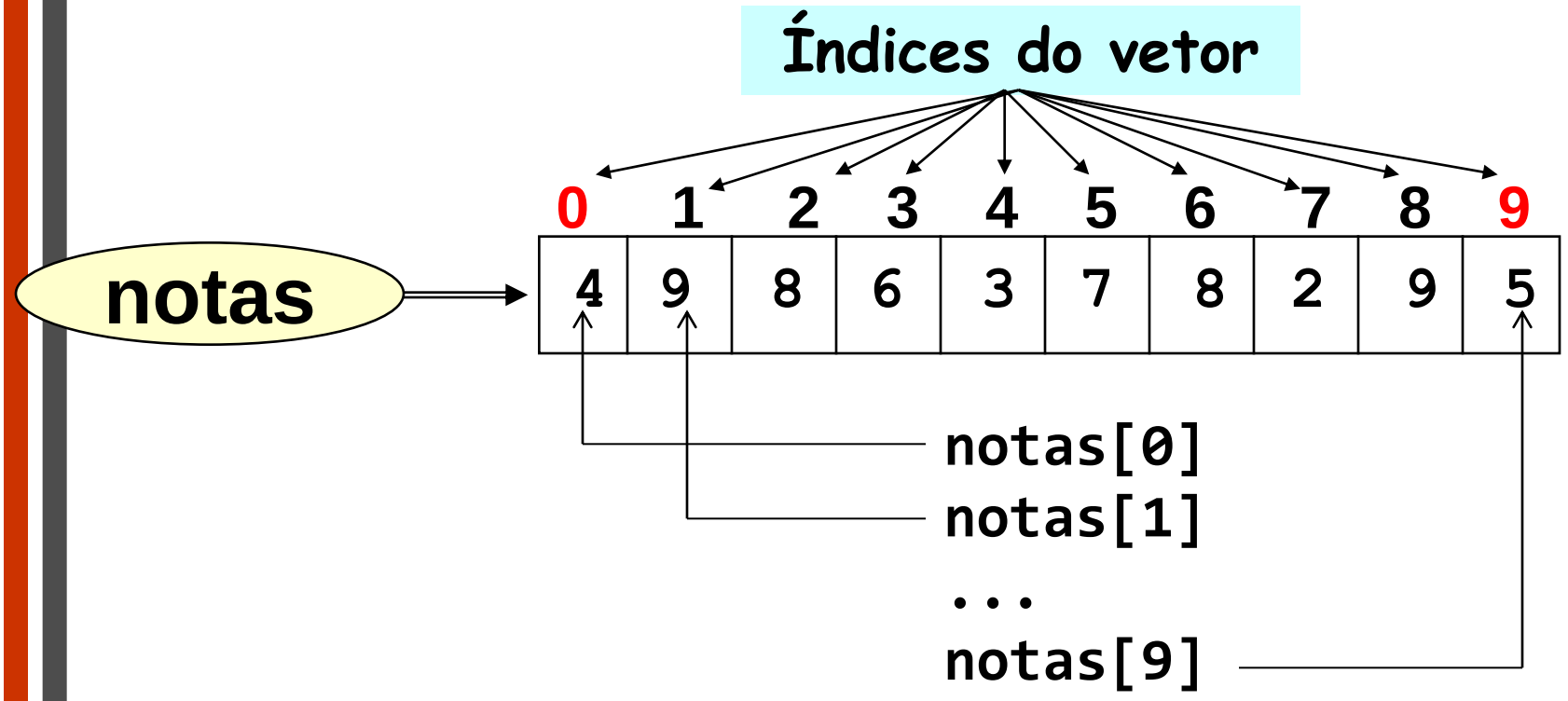
- **Vetor** ou **array** é um tipo de dado utilizado para representar um conjunto de valores homogêneos utilizando um único nome
- Define uma estrutura que armazena valores de um determinado tipo
- Um vetor é declarado usando

```
tipo nome[tamanho];
```

- No ato da declaração, devemos especificar o tamanho (dimensão) do vetor
  - Vetores têm tamanho fixo depois de criados

Vetores

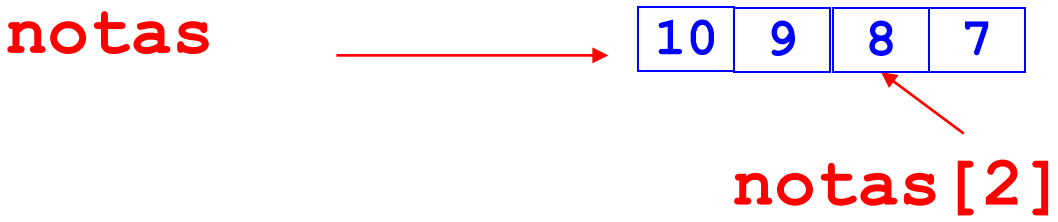
```
float notas[10];
```



- Cada elemento do vetor é referenciado através do **índice do vetor**
- Índice começa em *0* e vai até *tamanho -1*

Acessando Elementos de Vetores

- Um determinado elemento do vetor pode ser acessado usando o nome do vetor seguido do índice do elemento entre colchetes ([ ])
- A expressão **`notas[2]`** se refere ao valor **8** (3º elemento do array)



- A expressão **`notas[2]`** representa um local para armazenar um inteiro e pode ser utilizada da mesma forma que uma variável do tipo inteiro

Calculando a Média com Vetores

```
#include <stdio.h>
int main() {
   int qtdNumeros,contador = 0;
   float numeros[2000];
   float media = 0.0;
   do{
      printf("Quantidade de numeros? (<= 2000):\n");
      scanf("%d", &qtdNumeros);
   } while (qtdNumeros <= 0 || qtdNumeros > 2000);
   while (contador < qtdNumeros) {
      scanf("%f",&numeros[contador]);
      media = media + numeros[contador];
      contador++;
   }
   media = media/qtdNumeros;
   printf("\nA media eh %f\n",media)
   return 0;
}
```

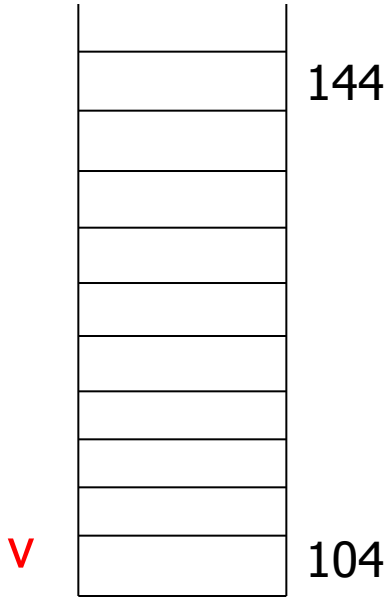
Vetor que guarda números entrados

Números podem ser acessados individualmente

Vetores na Memória

```
int v[10];
```

Aloca espaço para 10 valores inteiros, referenciados por **v**



É reservado um espaço de memória contínuo

`v[0]` → Acessa o primeiro elemento de v

⋮ ⋮

`v[9]` → Acessa o último elemento de v

Mas: `v [10]` → **Está errado !**

Checando os Limites dos Vetores

- Ao utilizar o índice para referenciar um elemento do vetor, este índice deve permitir o acesso a um elemento válido (em um endereço da memória alocado para o vetor)
  - Índice deve variar de *0* a *tamanho -1*
- C não avisa quando o limite de um vetor é excedido!
  - Se o programador transpuser o fim do vetor durante a operação de atribuição, os valores serão armazenados em outros dados ou mesmo no código do próprio programa

O programador tem a responsabilidade de verificar o limite do vetor!

Checando os Limites dos Vetores

- São comuns, erros de programação no uso de vetores dentro de laços
  - Deve-se prestar atenção na parte de teste do laço

```
int main() {
      int pares[20];
      int i,somaPares = 0;
      for (i = 0; i <= 20; i++) {
        pares[i] = 2 * i ;
          somaPares = somaPares + pares[i];
      }
      ...
}
```

Teste deveria ser `i < 20`

Por causa do teste errado, esta linha gerará um erro

Inicializando Vetores

- Inicializadores de vetores são representados da seguinte forma: ***{expressões}***,onde ***expressões*** representam expressões de tipos válidos separadas por vírgulas
- Vetores podem ser inicializados na declaração

```
int  v[5] = {5,10,15,20,25} ;
```

ou simplesmente:

```
int  v[]= {5,10,15,20,25};
```

Vetores só podem ser inicializados no ato da declaração!

Opções de Inicialização de Vetores

- No caso de utilizar inicializadores de vetores, note que:
  - Quando não especificado o tamanho, o compilador aloca espaço suficiente para armazenar todos os valores contidos na inicialização
  - Quando o tamanho for especificado e houver a lista de inicialização
    - Se há menos inicializadores que o tamanho especificado, os outros serão zero
    - Mais inicializadores que o necessário implica em um warning

Quando o vetor não for inicializado, o tamanho deve ser especificado na declaração !

Passando Vetores como Argumentos de Funções

- Um vetor pode ser passado como argumento para uma função
  - Parâmetro da função deve ser do tipo `tipo[]`
- Ao passar um vetor para uma função podemos modificar o conteúdo deste vetor dentro da função
  - Passa-se na verdade o endereço do vetor na memória **(veremos em breve!)**
  - Modificar um elemento do vetor ou incluir um novo elemento
- Podemos passar também um elemento em particular de um vetor para uma função
  - Parâmetro deve ser to tipo `tipo`

Passando Vetores como Argumentos para Funções

```
#include <stdio.h>
float  media(int n, float num[]){
  int i;
  float s = 0.0;
  for(i = 0; i < n; i++)
       s  = s +  num[i] ;
  return  s/n ;
}
int main(){
     float numeros[10] ;
     float med;
     int i ;
     for(i = 0; i < 10; i++)
          scanf ("%f", &numeros[i]) ;
     med  =  media(10, numeros ) ;
     ...
}
```

Parâmetro do tipo vetor de `float`

Passando o vetor `numeros` como argumento

Passando Vetores como Argumentos para Funções

```
#include <stdio.h>
void  incrementar(int n,float num[], float valor){
  int i;
  for(i = 0; i < n; i++)
       num[i] = num[i] + valor ;
}
int main(){
     float numeros[10] ;
     int i ;
     for(i = 0; i < 10; i++)
          scanf ("%f", &numeros[i]) ;
     incrementar( 10,numeros,1.5) ;
     ...
}
```

Modificando o conteúdo do vetor passado como argumento

Passando Elementos de Vetores como Argumentos

```
#include <stdio.h>
int reprovado(float nota, float media){
  int resultado = 0;
  if ( nota < media)
      resultado = 1;
  return resultado;
}
int main(){
  float notas[] = {6.5,5.0,7.5,9.4,3.8};
  int i ;
  for(i = 0; i < 5; i++){
        if (reprovado(notas[i],7.0) == 1)
            printf("Aluno %d: REPROVADO\n",i);
        else
            printf("Aluno %d: aprovado\n",i);
   }
     ...
}
```

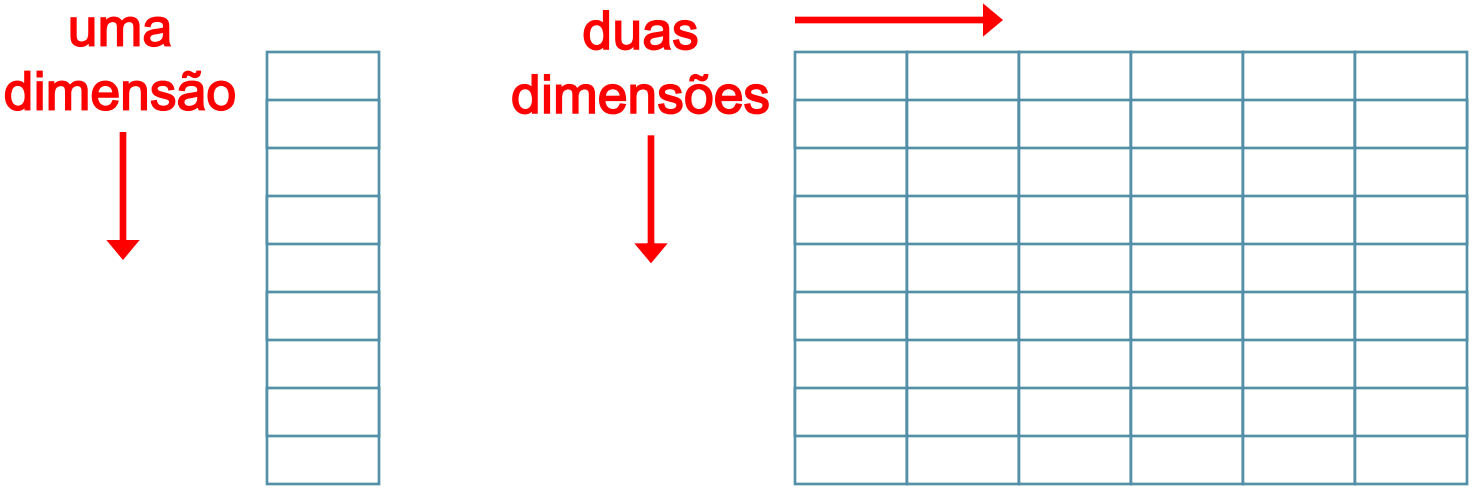
Passando um único elemento do vetor para uma função

Vetores Bidimensionais

- Um vetor unidimensional armazena uma lista de elementos
- Um vetor bidimensional pode ser visto como uma **matriz** com linhas e colunas

```
int matriz[9][6];
```

- Em C, um vetor bidimensional é um vetor de vetores



Inicializando Matrizes

- A inicialização de uma matriz também pode ser feita das seguintes formas:

```
float mat[4][3]={{5.0,10.0,15.0},{20.0,25.0,30.0} ,
                 {35.0,40.0,45.0},{50.0,55.0,60.0}}
```

```
float mat[4][3] = {5.0, 10.0, 15.0, 20.0, 25.0, 30.0,
                   35.0, 40.0,45.0, 50.0,55.0,60.0}
```

```
float mat [][3]= {5.0, 10.0, 15.0, 20.0, 25.0, 30.0,
                  35.0,40.0,45.0,50.0,55.0,60.0}
```

Deve ser passada a segunda dimensão

Percorrendo Matrizes

Saída: 1 2
5 6

```
int main() {
  int i,j;
  int matriz[2][2]={{1,2},{5,6}};
  for (i=0;i<2;i++){
     for (j=0;j<2;j++){
          printf("%d ",matriz[i][j]);
     }
     printf("\n");
  }
  return 0;
}
```

Laços aninhados para percorrer matriz

Passando Matrizes como Argumentos de Funções

- Uma matriz pode ser passada como argumento para uma função
  - Parâmetro da função deve ser do tipo `tipo[][tamanho colunas]`
- A linguagem C **exige** que a segunda dimensão seja especificada

Passando Matrizes como Argumentos

```
void imprimeMatriz(int linhas, int mat[][2]) {
  int i,j;
  for (i=0; i < linhas; i++){
     for (j=0;j < 2; j++){
          printf("%d ",mat[i][j]);
     }
     printf("\n");
  }
}
```

Deve ser especificada pelo menos a segunda dimensão

Resumindo ...

- Armazenamento de grande quantidade de informação
  - Necessidade de armazenamento
  - Problemas
    - Utilização de muitas variaveis
    - Manipulação incremental das informações
- **Vetores**
  - Conceito
  - Declaração
  - Inicialização
  - Acesso
  - Limites de um vetor
  - Passagem de vetores como parâmetros
  - Vetores bidimensionais (matrizes)