

Variáveis compostas

Linguagem de programação

Prof. Allan Rodrigo Leite

Variáveis primitivas

- Faça um programa que leia as 4 notas bimestrais de um aluno e calcule a média anual

Variáveis primitivas

- Faça um programa que leia as 4 notas bimestrais de um aluno e calcule a média anual

```
int main() {  
    float nota, media, soma = 0;  
  
    for (int i = 0; i < 4; i++) {  
        printf("Digite uma nota:");  
        scanf("%f", &nota);  
        soma += nota;  
    }  
  
    media = soma / 4;  
    printf("Media = %f", media);  
  
    return 0;  
}
```

Variáveis primitivas

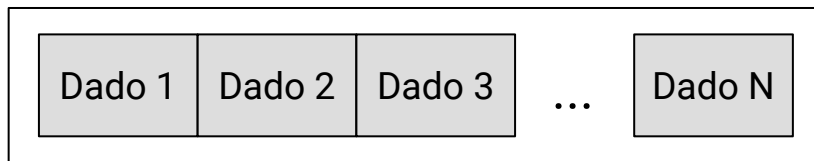
- Faça um programa que leia as 4 notas bimestrais de um aluno e calcule a média anual

```
int main() {  
    float nota, media, soma = 0;  
  
    for (int i = 0; i < 4; i++) {  
        printf("Digite uma nota:");  
        scanf("%f", &nota);  
        soma += nota;  
    }  
  
    media = soma / 4;  
    printf("Media = %f", media);  
  
    return 0;  
}
```

E se for necessário exibir quais notas estão acima da média?

Vetor

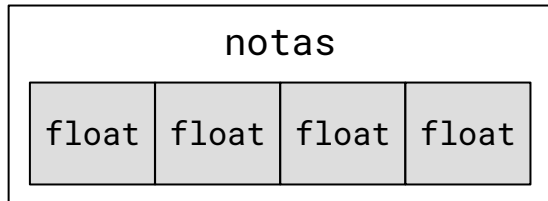
- Estrutura para armazenar uma sequência de dados do mesmo tipo
 - Também chamado de estrutura homogênea e estática
- Permite que dados sejam
 - Armazenados e estruturados de forma simples
 - São acessados diretamente
 - Mesmo que armazenados de forma sequencial



Declarando um vetor

- Para criar um vetor é necessário informar seu tamanho
 - Isto é, quantos valores este vetor permite armazenar
- Desta forma, o compilador se encarrega de reservar um espaço em memória que comporte o tamanho do vetor
 - Mesmo que armazenados de forma sequencial, os dados são acessados diretamente

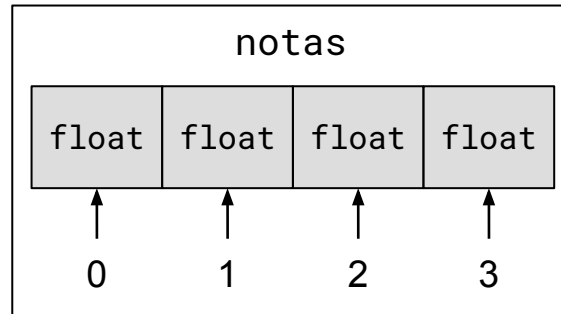
```
<tipo> nomeVetor[<tamanho>];  
float notas[4];
```



Acessando dados de um vetor

- Não é possível acessar todos os dados de uma só vez
 - Cada dado deve ser acessado individualmente
 - O acesso é realizado pelo índice do vetor
- O índice é uma sequência numerada dos dados
 - O acesso ao dado é realizado diretamente pelo índice

```
nomeVetor[<índice>];  
Notas[0] = 9.5;
```



Acessando dados do vetor

- Limites do vetor
 - O primeiro elemento do vetor recebe índice 0
 - O último elemento do vetor recebe índice $N - 1$
 - Onde N é o tamanho do vetor
- Este padrão é adotado em diversas linguagens de programação
 - Incluindo a linguagem de programação C
- Vetores são estruturas homogêneas e estáticas
 - Todos os índices são do mesmo tipo
 - O tamanho do vetor é imutável

Acessando dados do vetor

- Exemplo

```
int main() {  
    float notas[4], media, soma = 0;  
  
    for (int i = 0; i < 4; i++) {  
        printf("Digite uma nota:");  
        scanf("%f", &notas[i]);  
        soma += notas[i];  
    }  
  
    media = soma / 4;  
    printf("Media = %f", media);  
  
    return 0;  
}
```

Inicializando vetores

- Assim como ocorre em variáveis, vetores nem sempre são inicializados automaticamente
 - Após a criação do vetor, pode ser que exista lixo nos dados
 - Para inicializar um vetor, é possível utilizar
 - Um vetor literal
 - Uma estrutura de repetição

```
float notas[4];
```

```
for (int i = 0; i < 4; i++) {  
    notas[i] = 0;  
}
```

```
float notas[4] = { 0, 0, 0, 0 };
```

Definindo constantes

- A diretiva de compilação `#define` permite o uso de macros
 - Esta macro é usada para definir constante no código
- A utilização de constantes ajuda na legibilidade do código
 - O compilador substitui a constante pelo seu respectivo valor

```
#define NUM_NOTAS 4
```

```
float notas[NUM_NOTAS];
```

```
for (int i = 0; i < NUM_NOTAS; i++) {  
    notas[i] = 0;  
}
```

```
float notas[NUM_NOTAS] = { 0, 0, 0, 0 };
```

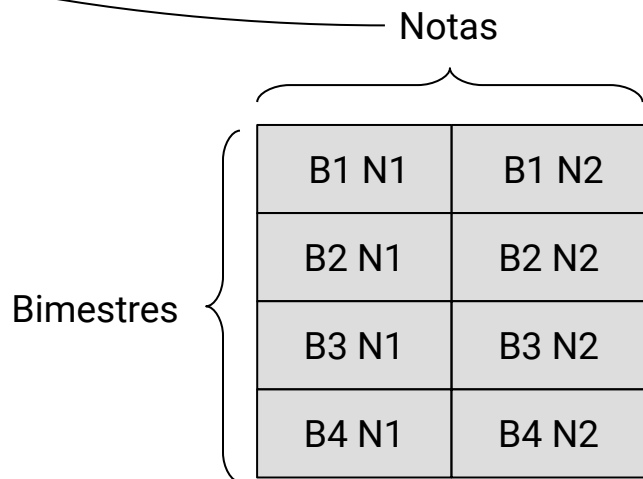
Matrizes

- Matrizes também são estruturas de dados homogêneas
 - Vetores são unidimensionais
 - Matrizes são multidimensionais
 - Análogas a estruturas tabulares, formadas por múltiplas linhas e colunas
- Matrizes permitem adicionar duas ou mais dimensões
 - Cada dimensão possui uma quantidade de elementos
 - Na grande maioria dos casos, usa-se matrizes bidimensionais

Matrizes

<tipo> nomeVetor[<tamanho dim 1>]...[<tamanho dim N>];

float notas[4][2];



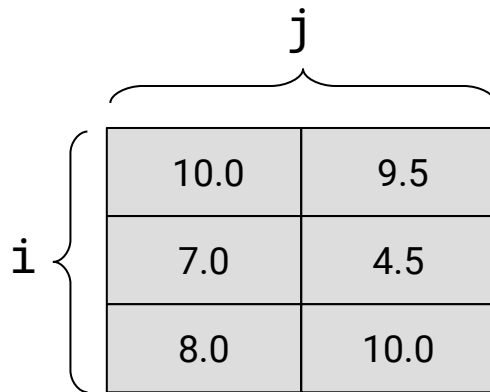
Acessando os dados da matriz

- Exemplo

```
float notas[3][2], soma = 0;
```

```
notas[0][0] = 10.0;  
notas[0][1] = 9.5;  
notas[1][0] = 7.0;  
notas[1][1] = 4.5;  
notas[2][0] = 8.0;  
notas[2][1] = 10.0;
```

```
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 2; j++) {  
        soma += notas[i][j];  
    }  
}
```



A diagram illustrating a 3x2 matrix. The rows are indexed by 'i' (0, 1, 2) and the columns by 'j' (0, 1). The matrix contains the following values:

	j=0	j=1
i=0	10.0	9.5
i=1	7.0	4.5
i=2	8.0	10.0

Exercícios

1. Faça um programa que leia um conjunto de números inteiros e armazene-os em um vetor, o qual representa um conjunto de dados. Em seguida, o programa deve ler novamente um novo conjunto de números reais com o mesmo tamanho do conjunto anterior e o armazene em um novo vetor, o qual representa um conjunto de pesos. Por fim, o programa deve retornar a média aritmética ponderada do conjunto de valores.
2. Faça um programa que leia um conjunto de números inteiros e armazene-os em um vetor que representa este conjunto de dados. Em seguida, o programa deve retornar a média, desvio padrão e variância dos valores do vetor.
3. Faça uma nova versão do programa anterior para exibir as seguintes informações a partir dos valores do vetor: i) mediana; ii) moda; iii) *outliers* (usando o método *Z score*); e iv) agrupamento dos valores em primeiro, segundo e terceiro quartil.

Exercícios

4. Faça um programa que, dado um vetor de números inteiros v de tamanho n e um número k , retorne verdadeiro se a soma de qualquer par de números em v for igual a k .
 - Exemplo: dado $v = [10, 15, 3, 7]$ e $k = 17$, a saída deve ser `true`, pois $10 + 7$ é 17.
5. Faça um programa que, dado um vetor de números inteiros v , retorne um novo vetor de forma que cada elemento no índice i seja o produto de todos os números no vetor original, com exceção de i .
 - Exemplo 1: dado $v = [1, 2, 3, 4, 5]$, a saída esperada é $[120, 60, 40, 30, 24]$.
 - Exemplo 2: dado $v = [3, 2, 1]$, a saída esperada é $[2, 3, 6]$.
6. Faça um programa que, dado um vetor de números inteiros v , retorne o máximo divisor comum entre os números do vetor.
 - Exemplo: dado $v = [46, 56, 14]$, a saída esperada é 14.

Variáveis homogêneas

- As variáveis estudadas até então são chamadas de homogêneas
 - São primitivas pois suportam tipos de dados básicos
 - Inteiro, real, caractere, etc.
 - São homogêneas pois tem um único tipo de dado em sua composição
- Como armazenar um conjunto de informações relacionadas?
 - Endereço
 - Dados de um cliente
 - Dados de um produto
 - ...

Variáveis compostas

- Conjunto heterogêneo de dados
 - Estrutura de dados composta por diversos tipos diferentes
 - Primitivos ou não
- Este conjunto é chamado de variável heterogênea composta
 - São variáveis que requerem uma definição estrutural
 - Na linguagem C, são chamadas de estruturas ou `structs`

Variáveis compostas

- Uma estrutura pode ser definida como
 - Uma coleção de uma ou mais variáveis relacionadas
 - Variáveis relacionadas são referidas como atributos ou campos
 - Cada variável pode ser de um tipo de dado distinto
- Exemplo
 - Representação de um funcionário
 - Composto por 4 campos

Funcionário

nome	char [20]
sobrenome	char [20]
salario	float
cargo	char [20]

Definição de estruturas

- Possibilita a criação de estruturas complexas
 - São formadas por um conjunto de atributos
 - Utiliza-se a instrução `struct` para definição de tipos complexos
- Exemplo

```
struct <nome da estrutura> {  
    //definição da estrutura  
}
```

```
struct funcionario {  
    char nome[20];  
    char sobrenome[20];  
    double salario;  
    char cargo[20];  
}
```

Declaração de estruturas

- Variáveis do tipo estrutura
 - Requerem a palavra reservada `struct` antes do nome da variável
- Exemplo

```
struct funcionario f1, f2, f3;  
struct funcionario outro;
```

Definição de novos tipos de dados

- Definição de novos tipos baseados em estruturas
 - Utiliza-se a instrução typedef para definição de novos tipos
- Exemplo

```
typedef struct {  
    //definição da estrutura  
} <nome do tipo de dados>
```

```
typedef struct {  
    char nome[20];  
    char sobrenome[20];  
    double salario;  
    char cargo[20];  
} Funcionario;
```

```
Funcionario f1, f2, f3;
```

Definição e declaração de estruturas

- A definição de estruturas deve ficar fora da função principal `main` e de qualquer outra função
 - É possível definir mais de uma estrutura em um mesmo código fonte
 - A definição do novo tipo (`typedef`) também segue a mesma regra
- A declaração de uma variável do tipo estrutura deve ficar dentro de alguma função ou na função principal `main`
 - Do contrário será uma variável global

Manipulando estruturas

- Cada campo ou atributo pode armazenar um valor
 - Como qualquer variável, este valor pode ser acessado, modificado, etc.
- Operador .
 - Campos são acessados com . entre o nome da variável declarada como estrutura e o nome do campo
- Exemplo

```
Funcionario f;  
scanf("%s",&(f.nome));  
printf("Nome do funcionário: %s\n",f.nome);
```


Definição de estruturas aninhadas

- Estrutura que contém atributos referentes a outras estruturas
 - Contudo, o compilador do C segue a abordagem top-down
 - Significa que uma estrutura, para ser referenciada em outra estrutura, deve ter sido previamente declarada

```
typedef struct {  
    int dia; int mes; int ano;  
} Data;
```

```
typedef struct {  
    char nome[20];  
    double salario;  
    Data nascimento;  
} Funcionario;
```

```
Funcionario f;  
f.nascimento.dia = 10;
```

Vetores de estruturas

- É possível criar vetores de estruturas
 - Assim como utilizados em vetores de tipos primitivos
- Os exemplos até então utilizaram uma única instância da estrutura
 - Requer a definição da estrutura antes de declarar um vetor de estrutura
- Exemplo

```
typedef struct {  
    char nome[20];  
    double salario;  
} Funcionario;
```

```
Funcionario f[50];  
f[0].nome = "Paulo";
```

Iniciando estruturas

- Pode-se inicializar um vetor de estruturas durante sua criação
 - Similar à inicialização de vetores de tipos primitivos
 - O C99 permite determinar os atributos e os respectivos valores
- Exemplo

```
typedef struct {  
    char nome[20];  
    double salario;  
} Funcionario;
```

```
Funcionario f1 = {"Paulo", 2450.00};  
Funcionario f2 = {.nome = "Paulo", .salario = 2450.00};  
Funcionario f3 = {"Paulo"};
```

Inicializando vetores de estruturas

- Pode-se inicializar um vetor de estruturas durante sua criação
 - Similar à inicialização de vetores de tipos primitivos
- Exemplo

```
typedef struct {  
    char nome[20];  
    double salario;  
} Funcionario;
```

```
Funcionario f[3] = {{ "Paulo", 2450.00}  
                    { "Henrique"  
                    { "Joaquim", 1400.00}}}
```

Enumeração

- Enumeração (enum) é um tipo de dados definido pelo desenvolvedor
 - Se trata de um conjunto finito de enumeradores
 - Usado principalmente para atribuir nomes a constantes integrais
 - Isto torna mais legível o código um programa
- Exemplos de uso
 - Dias da semana
 - Meses do ano
 - Situação de um pedido em um e-commerce
 - Aguardando confirmação de pagamento
 - Pedido confirmado
 - Em transporte
 - Entregue

Definindo enumeradores

- Usa-se a instrução `enum` para definir enumeradores
 - Os valores das constantes são inteiros
 - Por padrão incrementam em uma unidade e iniciam em 0
 - `<Constante 1> = 0`
 - `<Constante 2> = 1`
 - `<Constante N> = N - 1`

`enum` <nome do enumerador> {<constante 1>, <constante 2>, ..., <constante N>;

● Exemplo

```
enum semana { Segunda, Terca, Quarta, Quinta, Sexta };  
enum semana { Segunda = 1, Terca = 2, Quarta = 3, Quinta = 4, Sexta = 5 };
```

Declarando enumeradores

- Usa-se a instrução enum para declarar variáveis do tipo enum
 - As constantes podem ser utilizadas dentro do escopo do enum
 - Portanto, as constantes devem ser únicas em um mesmo escopo

```
enum <nome do enumerador> <identificador>;
```

- Exemplo

```
enum semana { Segunda, Terca, Quarta, Quinta, Sexta }; //declaração do enum
enum semana dia; //definição da variável do tipo enum
//ou
enum semana { Segunda, Terca, Quarta, Quinta, Sexta } dia; //declaração e definição

dia = Segunda;

printf("%d\n", dia);
```

Exercícios

1. Faça um programa que leia o nome de um produto, o preço e a quantidade comprada e armazene estas informações em uma estrutura. Em seguida, o programa deve escrever o nome do produto, quantidade, preço unitário e o valor total a ser pago, considerando que são oferecidos diferentes descontos com base no número de unidades compradas. Os descontos concedidos devem respeitar tabela a seguir:
 - a. Até 5 unidades: sem desconto
 - b. De 6 a 15 unidades: 10% de desconto
 - c. Acima de 16 unidades: 20% de desconto
2. Faça um programa para cadastrar veículos. O programa deve conter um menu inicial com opções para incluir um novo veículo, listar todos os veículos cadastrados e finalizar o programa. A estrutura do veículo deve conter a placa, categoria (carro, moto, ônibus, caminhão, etc.), modelo e o ano. Não pode haver veículos com placas idênticas. Use enumeradores para descrever as categorias dos veículos.

Exercícios

3. Faça um programa para manter as informações de uma agenda de contatos utilizando estruturas e vetores. O programa deve conter um menu inicial com 5 opções: i) incluir um novo contato; ii) excluir um contato existente; iii) alterar um contato existente; iv) listar todos os contatos cadastrados; e v) finalizar o programa. A estrutura do contato deve conter um código de identificação, nome, e-mail e celular.

4. Faça um programa que, dadas as posições e dimensões de dois retângulos em um plano bidimensional, retorne a área de intersecção entre eles. Se não houver intersecção, deve retornar 0. Use estruturas para representar os retângulos em 2D.
 - Exemplo: dado os retângulos abaixo, a área de intersecção é 6.
 - `r1 = { .y = 1, .x = 4, .largura = 3, .altura = 3 }`
 - `r2 = { .y = 0, .x = 5, .largura = 4, .altura = 3 }`

Variáveis compostas

Linguagem de programação

Prof. Allan Rodrigo Leite