SIN 211 - Algoritmos e Estruturas de Dados
(Revisão 1)

Profº: Joelson Antônio dos Santos

Universidade Federal de Viçosa
Instituto de Ciências Exatas e Tecnológicas
Campus de Rio Paranaíba - MG

joelsonn.santos@gmail.com
Sala: BBT 233

13 de março de 2018

Roteiro

Créditos

O material desta aula é composto por adaptações e extensões dos originais gentilmente cedidos pelos professores **Moacir Pereira Ponti** e **Rachel Reis**.

Aula de hoje

- *Arrays*
  - *Strings*
- Estruturas

Arrays

- Podem ter múltiplas dimensões[1].
  - Unidimensional Vetor.
  - Bidimensional Matriz.

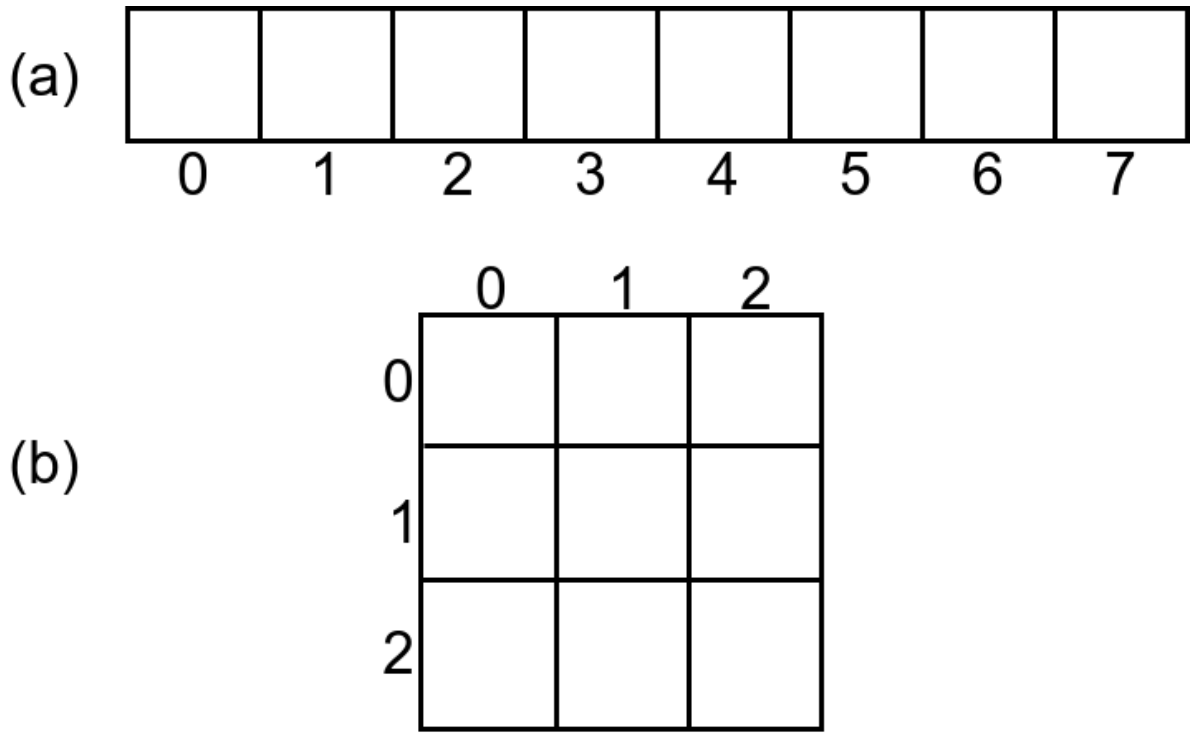


Figura: **(a)** vetor; **(b)** matriz.

[1] *Arrays* com três ou mais dimensões não serão abordados neste curso.

Arrays

- **Array (vetor ou matriz)**: Coleção de variáveis de mesmo tipo referenciada por um nome comum[2].
- Estrutura de dados estática.
- Ocupam posições contíguas de memória.
- Um elemento específico do *Array* pode ser acessado pelo seu índice.

[2]SCHILDT H. C Completo e Total. Ed. Makron Books, 3ª edição, 1997.

Arrays - Algumas operações

- Existem várias operações que podem ser feitas sobre *Arrays*, algumas delas são:
  - Declaração
  - Inicialização
  - Impressão

Arrays - Declaração

- Forma geral de declaração de um *Array* unidimensional (vetor) em C.

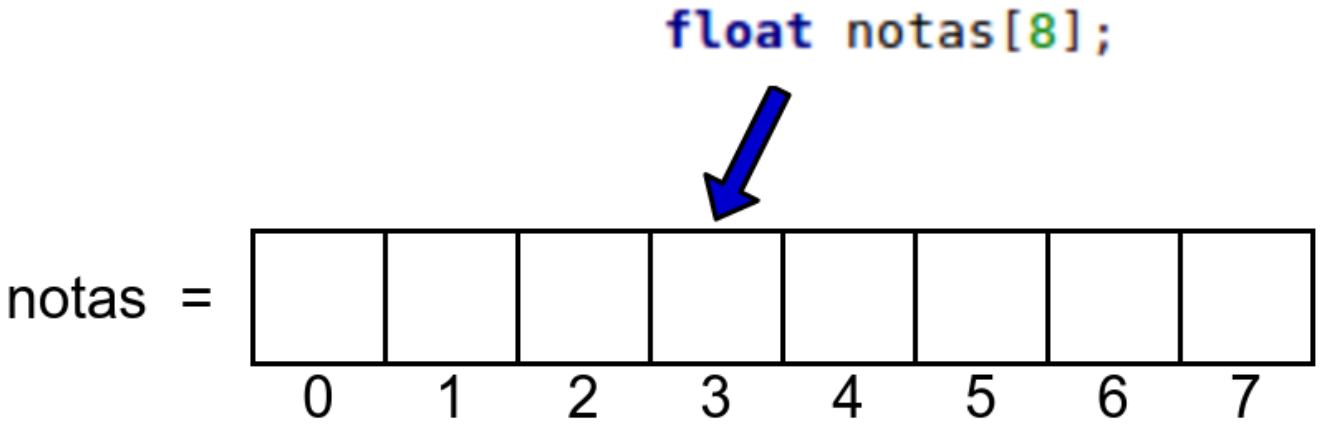
Arrays - Declaração

- Forma geral de declaração de um *Array* unidimensional (vetor) em C.
  - **tipo** nome_variavel[tamanho];

Arrays - Declaração

- Forma geral de declaração de um *Array* unidimensional (vetor) em C.
  - **tipo** nome_variavel[tamanho];

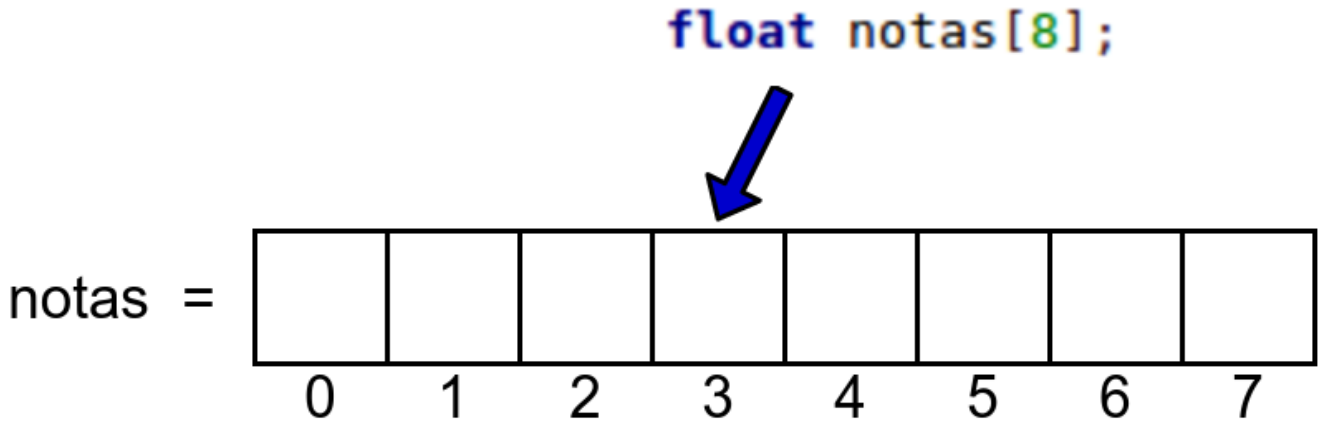
Figura: Exemplo de declaração.



Arrays - Declaração

- Forma geral de declaração de um *Array* unidimensional (vetor) em C.
  - **tipo** nome_variavel[tamanho];

Figura: Exemplo de declaração.

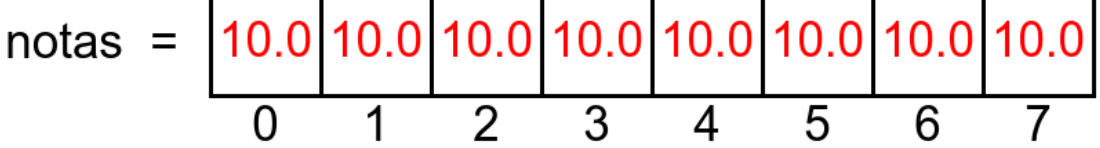


- O que está armazenado nas posições do vetor "notas" ao declará-lo?

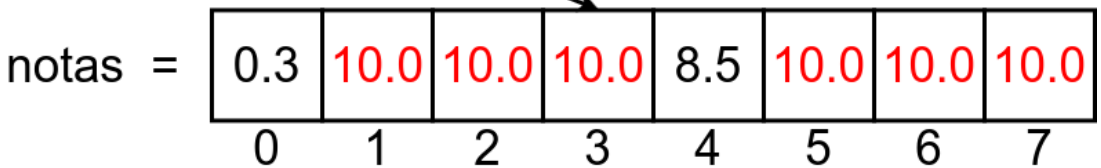
Arrays - Inicialização

- Formas de atribuição (inicialização) de um vetor.

```
for(i = 0; i < 8; i++){
    notas[i] = 10.0;
}
```




```
notas[0] = 0.3;
notas[4] = 8.5;
```



Arrays - Inicialização (Cuidados)

```
int contador[10];

for(i = 0; i < 50; i++){
   contador[i] = i;
}
```

Qual o erro neste trecho de código?

Arrays - Inicialização (Cuidados)

```
int contador[10];

for(i = 0; i < 50; i++){
   contador[i] = i;
}
```

Qual o erro neste trecho de código?

- Produz erro de compilação?

Arrays - Inicialização (Cuidados)

```
int contador[10];

for(i = 0; i < 50; i++){
   contador[i] = i;
}
```

Qual o erro neste trecho de código?

- Produz erro de compilação?
- **R:** Não, mas o limite (número de posições) do vetor "contador" é ultrapassado. Logo, essa operação está incorreta.

Arrays - Inicialização (Cuidados)

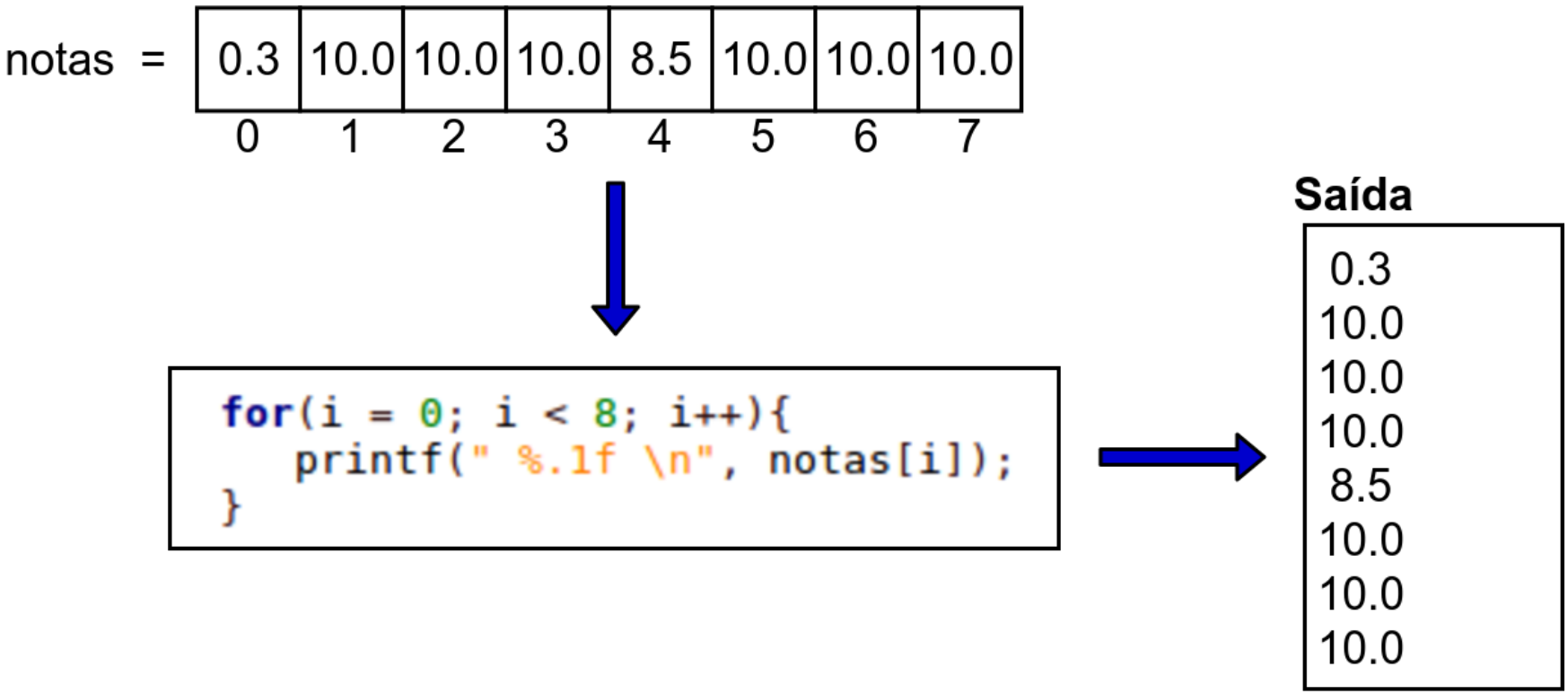
```
int contador[10];

for(i = 0; i < 50; i++){
   contador[i] = i;
}
```

Qual o erro neste trecho de código?

- Produz erro de compilação?
- **R:** Não, mas o limite (número de posições) do vetor "contador" é ultrapassado. Logo, essa operação está incorreta.
- É obrigação do programador verificar o limite de tamanho de qualquer *Array* em seus códigos.

Arrays - Impressão

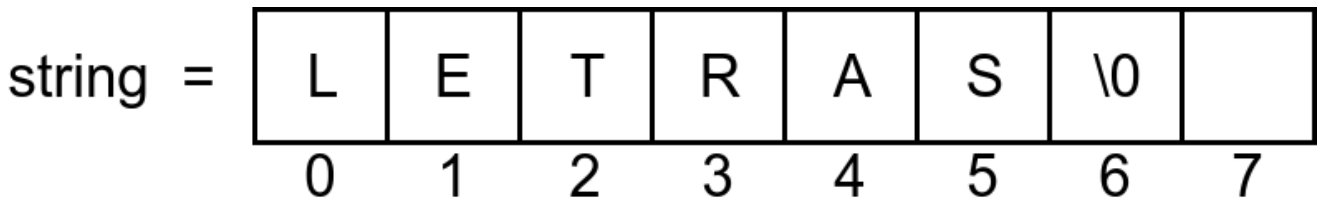
- Para imprimir os elementos de um vetor em C, faz-se:



Strings

- **String de caracteres:** Tipo mais comum de *Array* unidimensional.

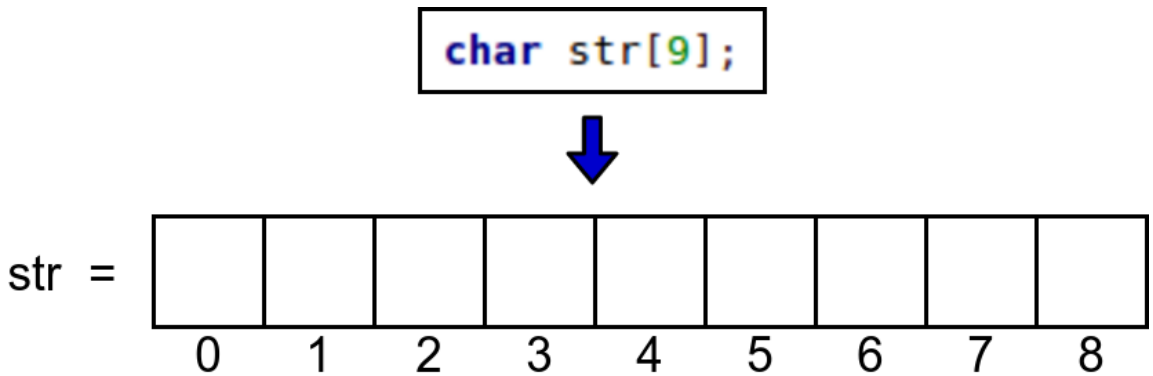
- String de caracteres é definida como um vetor de caracteres terminado com um valor nulo definido por "\0".



Strings - Declaração

- A **declaração** de uma *string* em C deve considerar o tamanho do vetor mais a posição para armazenar o valor "\0" no final do mesmo.

- **Exemplo** - Se desejarmos declarar um vetor de caracteres chamada "str" de tamanho 8. Então devemos declarar da seguinte maneira:



Strings - Constantes

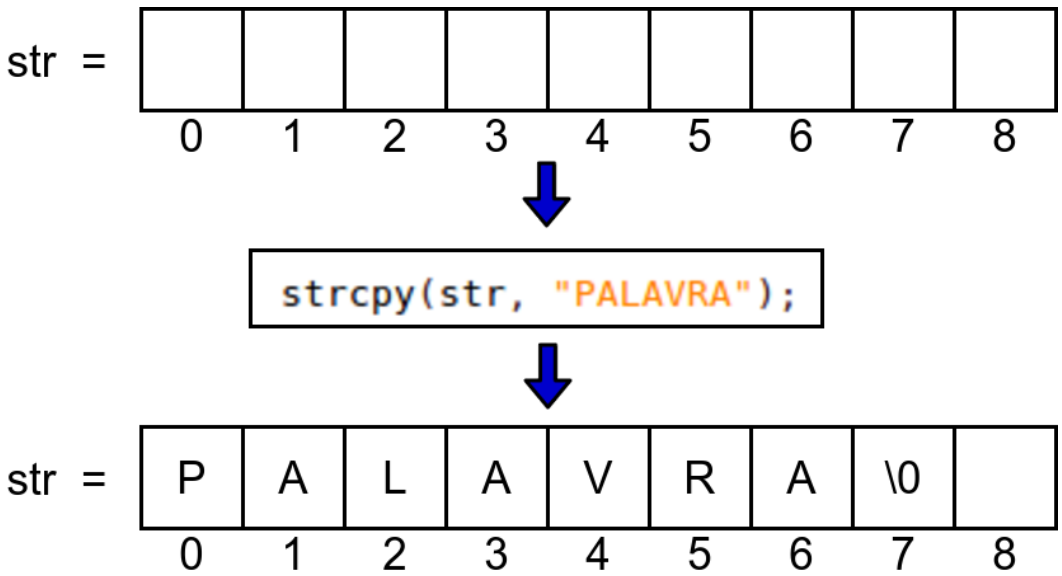
- Uma *string* também pode ser representada por uma lista de caracteres fixa chamada de *string constante* (exemplo: “letras”).

- A adição do valor nulo \0 no final da *string constante* não é necessária, pois o compilador faz isso automaticamente.

Strings - Funções de Manipulação

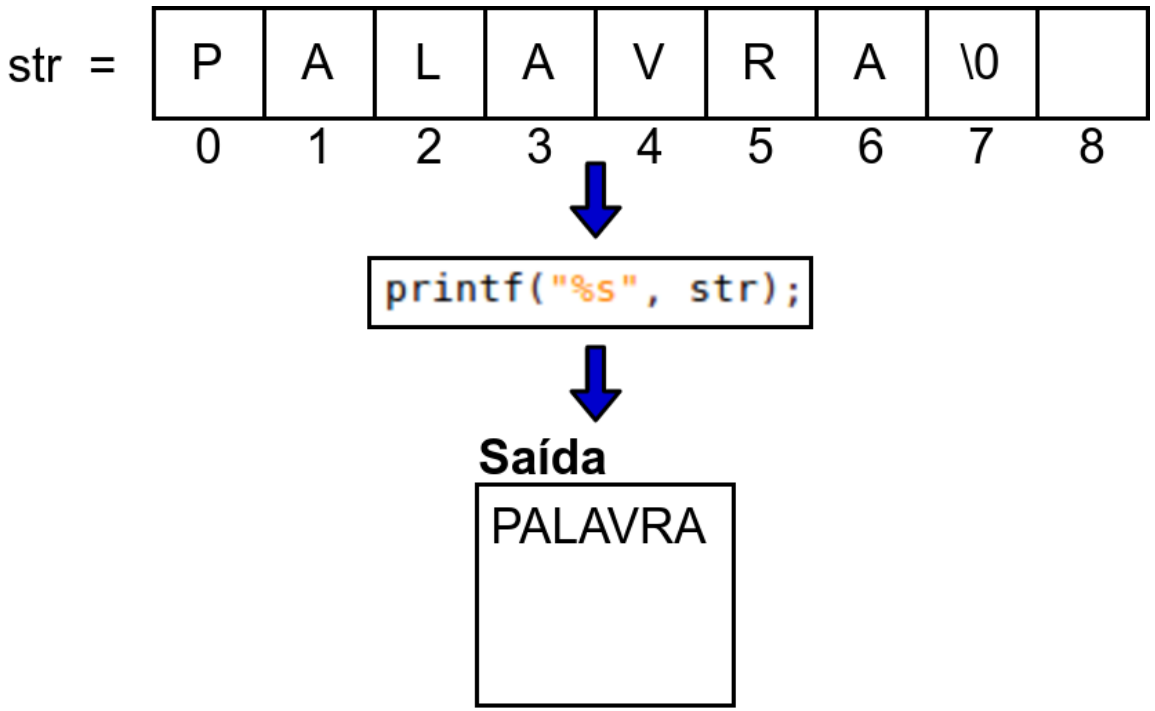
- Existem diversas funções de manipulação de *strings* em C, uma delas é:
  - *strcpy(string1, string2)*: Copia *string2* em *string1*.

Exemplo:



Strings - Impressão

- Imprimindo uma *string*.



Estruturas

Características de estruturas:

- Agrupamento de variáveis referenciadas por um nome (identificador).
- As variáveis que compreendem a estrutura são chamadas de *elementos* ou *campos*.
- São estruturas estáticas.

Estruturas

Características de estruturas:

- Ocupam posições consecutivas de memória.
- Geralmente, os elementos de uma estrutura possuem relacionamento lógico entre eles.

Exemplo: Estrutura *endereço* = {*rua*, *bairro*, *cidade*, *estado*, *número*, *CEP*}.

Estruturas - Algumas Operações

- Definição
- Declaração
- Manipulação
- Impressão

Estruturas - Definição

- Em linguagem C, uma estrutura é representada pela palavra chave **struct**.

Escopo de definição de uma **struct**:

```
struct identificacao_estrutura {

    tipo1   nome1;
    tipo2   nome2;
       ⋮
    tipoN   nomeN;
};
```

Estruturas - Definição

Exemplo de definição de uma **struct** Endereço:

```
struct sEndereco{
   char rua[40];
   int numero;
   char bairro[40];
   char cidade[20];
   char estado[3];
   long int CEP;
};
```

- **Observação:** O trecho de código acima sempre deve ser definido no início do programa logo após os *includes*.

Estruturas - Declaração

- Como declarar a **struct** *sEndereco* na função principal:

```
int main(){

    // Cria variável endereco como struct do tipo sEndereco.
    struct sEndereco endereco;
```

Estruturas - Manipulação

- Elementos individuais de uma estrutura são acessados pelo *operador ponto* (“.”).

- Como acessar os elementos *rua* e *numero* da **struct** *sEndereco* e inicializar/manipular seus valores individualmente (dentro da função *main*):

```
strcpy(endereco.rua, "Rua 16 de agosto");
endereco.numero = 1000;
```

Estruturas - Impressão

- Impressão dos elementos *rua* e *numero* da **struct** *sEndereco* dentro da função principal:

```
printf("%s\n", endereco.rua);
printf("%d\n", endereco.numero);
```

Estruturas - Exemplo Completo

```c
#include<stdio.h>


struct sPonto{
  int x, y;
};


int main(){

   // Cria variável ponto como struct do tipo sPonto.
   struct sPonto ponto;

   // Inicializa elementos da variável ponto.
   ponto.x = 10;
   printf("Digite o valor de y: ");
   scanf("%d", &ponto.y);

   // Imprime valores dos elementos x e y.
   printf("x: %d e y: %d\n", ponto.x, ponto.y);

 return 0;
}
```

Estruturas - Redefinição

- A linguagem C permite definir novos nomes para tipos de dados.

  Notação: **typedef** *tipo* novo_nome;

- **typedef** no contexto de **struct**.

```
typedef struct identificacao_estrutura {

    tipo1   nome1;
    tipo2   nome2;
       ⋮
    tipoN   nomeN;
}id_novo_nome;
```

Estruturas - Definição, Redefinição, Manipulação e Impressão

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

typedef struct sPonto{
  int x, y;
}Ponto; // Cria struct sPonto e define o novo nome como Ponto.


int main(){

   Ponto ponto; // declara variável ponto do tipo Ponto.

   ponto.x = 10;
   printf("Digite o valor de y: ");
   scanf("%d", &ponto.y);

   printf("x: %d e y: %d\n", ponto.x, ponto.y);

 return 0;
}
```

Estruturas Aninhadas

- São estruturas em que um ou mais elementos que também são estruturas.

Exemplo:

```
typedef struct identificacao_estrutura1 {
    tipo1   nome1;
    tipo2   nome2;
            ⋮
    tipoN   nomeN;
}id_novo_nome1;

typedef struct identificacao_estrutura2 {
    id_novo_nome1 nome1;
    tipo2   nome2;
            ⋮
    tipoN   nomeN;
}id_novo_nome2;
```

Estruturas Aninhadas

Exemplo2:

```
typedef struct sHora{
  int hora;
  int minuto;
  int segundo;
}Hora;

typedef struct sRelogio{
    Hora H;
    char modelo[10];
}Relogio;
```

Estruturas Aninhadas - Definição, Redefinição, Manipulação e Impressão

```c
#include<stdio.h>
#include<string.h>


typedef struct sHora{
  int hora;
  int minuto;
  int segundo;
}Hora;

typedef struct sRelogio{
    Hora H;
    char modelo[10];
}Relogio;
```

```c
int main(){

 // declaração
 Relogio r1;

 // manipulação
 r1.H.hora = 10;
 r1.H.minuto = 15;
 r1.H.segundo = 30;
 strcpy(r1.modelo, "Cassio");

 // Impressão
 printf("Modelo: %s\n", r1.modelo);
 printf("%d:%d:%d",
        r1.H.hora,
        r1.H.minuto,
        r1.H.segundo);

 return 0;
}
```

Array e Estruturas

- É possível combinar *arrays* e estruturas para a criação de diferentes estruturas de dados.
- Podemos ter uma estrutura contendo um ou mais membros do tipo *array*, **ou**;
- Criar um *array* cujo os elementos sejam estruturas.

Exercícios

- Leia os capítulos 4 e 7 do livro *C completo e Total*[3].
- Crie uma estrutura para disciplina. Ela possui: *nome*, *codigo*, *numeroCreditos*.
- Crie uma estrutura semestre. Ela possui: *numeroPeriodo* e um vetor de disciplinas.
- O número máximo de disciplinas permitidas por um aluno é de 8 por semestre.
- Inicialize diretamente uma estrutura do tipo semestre com:
  - O período que está cursando.
  - As disciplinas que está cursando.

[3] SCHILDT H. C Completo e Total. Ed. Makron Books, 3ª edição, 1997.

Bibliografia Básica

- SCHILDT H. C Completo e Total. Ed. Makron Books, 3ª edição, 1997

Dúvidas?