Elementos de programação em C

Ponteiros e vetores

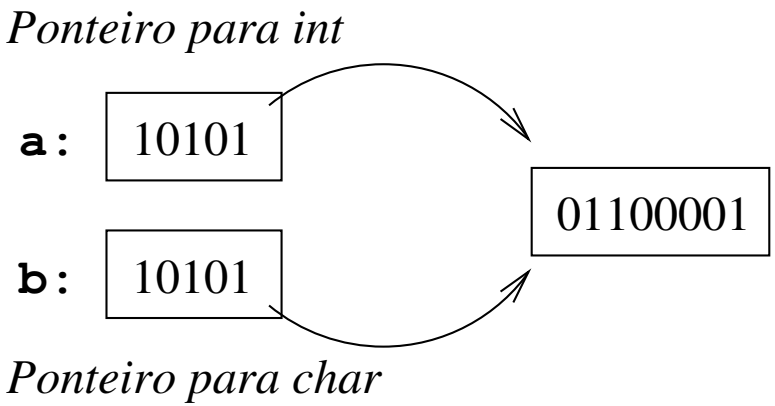
Francisco A. C. Pinheiro, *Elementos de Programação em C*, Bookman, 2012.

Visite os sítios do livro para obter material adicional: www.bookman.com.br e www.facp.pro.br/livroc

Sumário

Ponteiros

- As variáveis do tipo ponteiro para $\langle T \rangle$ armazenam endereços de memória.



- Os valores armazenados nesses endereços são interpretados como valores do tipo $\langle T \rangle$ (quando acessados por meio da variável ponteiro para $\langle T \rangle$ que designa o endereço).

Declaração de ponteiros

$$\langle DeclPonteiro \rangle ::= \ * \ [\ \langle QualifTipo \rangle\ ] \\ \quad | \quad * \ [\ \langle QualifTipo \rangle\ ]\ \langle DeclPonteiro \rangle$$

Declarações válidas:

`int *ptr_a;`	Declara `ptr_a` do tipo ponteiro para `int`.
`int *ptr_a, ptr_b;`	Declara as variáveis `ptr_a`, do tipo ponteiro para `int`, e `ptr_b`, do tipo `int`.

Declaração de ponteiros

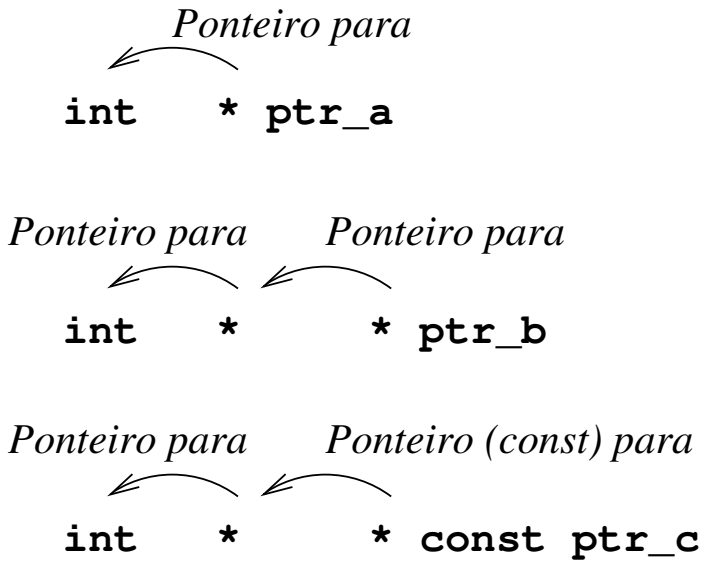
Declarações válidas:

`int **ptr_a;`	Declara `ptr_a` do tipo ponteiro para ponteiro para `int`.
`int * const ptr_a;`	Declara `ptr_a` do tipo ponteiro (constante) para `int`. O conteúdo de `ptr_a` não pode ser modificado, o conteúdo apontado por `ptr_a` pode.
`const int * ptr_a;`	Declara `ptr_a` do tipo ponteiro para `const int`. O conteúdo de `ptr_a` pode ser modificado, o conteúdo apontado por `ptr_a` não pode.

Declaração de ponteiros

Exemplo

As declarações
`int *ptr_a;`
`int **ptr_b;` e
`int ** const ptr_c;`
são interpretadas do seguinte modo:



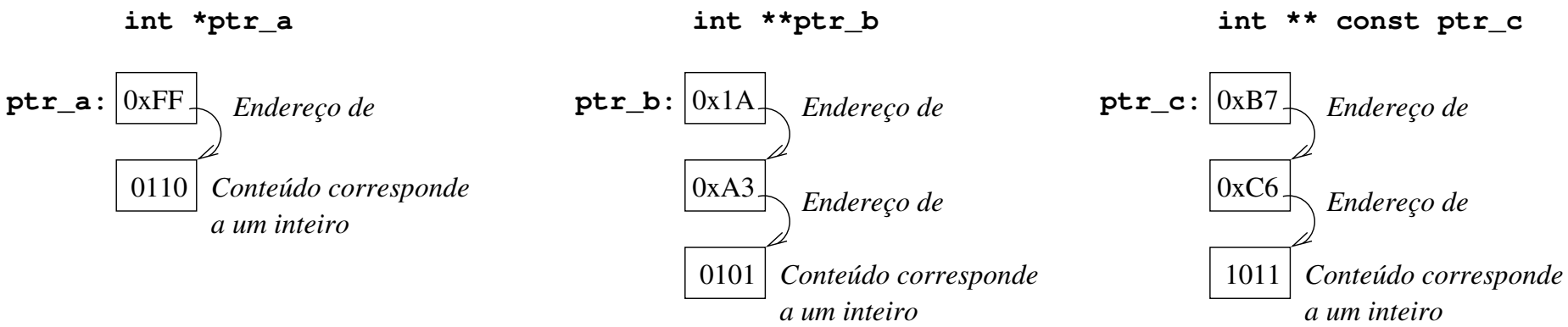
Declaração de ponteiros

Exemplo

As declarações

```
int *ptr_a;
int **ptr_b; e
int ** const ptr_c;
```

são interpretadas do seguinte modo:



Operador de endereço

O operador &, aplicado a uma expressão que designa uma localização de memória, resulta no endereço da localização designada pela expressão.

Operador de endereço

O operador `&`, aplicado a uma expressão que designa uma localização de memória, resulta no endereço da localização designada pela expressão.

Exemplo

- `&aluno` resulta no endereço da variável `aluno`
- `&(notas[i])` resulta no endereço designado pela expressão `(notas[i])`.

Operador de endereço

Exemplo

O programa ao lado imprime os endereços armazenados nas variáveis `ptr_a` e `ptr_b`.

```
#include <stdio.h>
int main(void) {
  int x = 97;
  int *ptr_a;
  char *ptr_b;
  ptr_a = &x;
  ptr_b = &x;
  printf("%p %p\n", (void *)ptr_a,
                    (void *)ptr_b);
  return 0;
}
```

Operador de acesso indireto

O operador *, quando aplicado a uma variável do tipo ponteiro, resulta na referência ao endereço apontado pelo ponteiro, que pode ser usada:

- para obter o conteúdo referido ou
- como o operando esquerdo do operador de atribuição

Operador de acesso indireto

O operador `*`, quando aplicado a uma variável do tipo ponteiro, resulta na referência ao endereço apontado pelo ponteiro, que pode ser usada:

- para obter o conteúdo referido ou
- como o operando esquerdo do operador de atribuição

Exemplo

Se `ptr_a` é uma variável do tipo ponteiro para `int`, então:

`ptr_a`	resulta no valor armazenado em `ptr_a`: um endereço.

Operador de acesso indireto

O operador `*`, quando aplicado a uma variável do tipo ponteiro, resulta na referência ao endereço apontado pelo ponteiro, que pode ser usada:

- para obter o conteúdo referido ou
- como o operando esquerdo do operador de atribuição

Exemplo

Se `ptr_a` é uma variável do tipo ponteiro para `int`, então:

`ptr_a`	resulta no valor armazenado em `ptr_a`: um endereço.
`*ptr_a`	resulta na referência ao espaço de memória apontado por `ptr_a`.

Operador de acesso indireto

O operador `*`, quando aplicado a uma variável do tipo ponteiro, resulta na referência ao endereço apontado pelo ponteiro, que pode ser usada:

- para obter o conteúdo referido ou
- como o operando esquerdo do operador de atribuição

Exemplo

Se `ptr_a` é uma variável do tipo ponteiro para `int`, então:

`ptr_a`	resulta no valor armazenado em `ptr_a`: um endereço.
`*ptr_a`	resulta na referência ao espaço de memória apontado por `ptr_a`.
`*ptr_a = 23;`	atribui o valor 23 ao espaço de memória apontado por `ptr_a`.

Operador de acesso indireto

O operador `*`, quando aplicado a uma variável do tipo ponteiro, resulta na referência ao endereço apontado pelo ponteiro, que pode ser usada:

- para obter o conteúdo referido ou
- como o operando esquerdo do operador de atribuição

Exemplo

Se `ptr_a` é uma variável do tipo ponteiro para `int`, então:

`ptr_a`	resulta no valor armazenado em `ptr_a`: um endereço.
`*ptr_a`	resulta na referência ao espaço de memória apontado por `ptr_a`.
`*ptr_a = 23;`	atribui o valor 23 ao espaço de memória apontado por `ptr_a`.
`printf("%d", *ptr_a);`	imprime o conteúdo do espaço de memória apontado por `ptr_a`.

Operador de acesso indireto

Exemplo

O que é impresso pelo código ao lado?

```
#include <stdio.h>
int main(void) {
  int x = 97;
  int y = 76;
  int *ptr_x = &x;
  int *ptr_y = &y;
  printf("%d %d\n", *ptr_x, *ptr_y);
  *ptr_x = 123 + *ptr_y;
  (*ptr_y)++;
  printf("%d %d\n", x, y);
  return 0;
}
```

Operador de acesso indireto

Exemplo

O que é impresso pelo código ao lado?

```
#include <stdio.h>
int main(void) {
   int x = 97;
   int y = 76;
   int *ptr_x = &x;
   int *ptr_y = &y;
   printf("%d %d\n", *ptr_x, *ptr_y);
   *ptr_x = 123 + *ptr_y;
   (*ptr_y)++;
   printf("%d %d\n", x, y);
   return 0;
}
```

Resposta:

97 76

199 77

Relação entre * e &

As seguintes equivalências são válidas:

- `&*ptr` $\equiv$ `ptr`
- `*&var_a` $\equiv$ `var_a`
- `&(vet[x])` $\equiv$ `((vet) + (x))`

Ponteiros para funções

Na declaração de ponteiros para funções deve ser observado que o operador `()` tem maior precedência que o declarador de ponteiro `*`:

Ponteiros para funções

Na declaração de ponteiros para funções deve ser observado que o operador `()` tem maior precedência que o declarador de ponteiro `*`:

`void *fun(void);`	Declara `fun` como uma função sem argumentos retornando um ponteiro para `void`.
`void (*fun)(void);`	Declara `fun` como um ponteiro para uma função sem argumentos retornando `void`.

Ponteiros para funções

Na declaração de ponteiros para funções deve ser observado que o operador `()` tem maior precedência que o declarador de ponteiro `*`:

`void *fun(void);`	Declara `fun` como uma função sem argumentos retornando um ponteiro para `void`.
`void (*fun)(void);`	Declara `fun` como um ponteiro para uma função sem argumentos retornando `void`.
`int *fun(int);`	Declara `fun` como uma função de `int` retornando um ponteiro para `int`.
`int (*fun)(int);`	Declara `fun` como um ponteiro para uma função de `int` retornando `int`.

Vetores

Sequências de elementos de um dado tipo, armazenados em posições contíguas da memória, distribuídos em um número predeterminado de dimensões.

Declaração de vetores

⟨*DeclaradorVetor*⟩ ::= ⟨*Identificador*⟩ ⟨*DeclDim*⟩ { ⟨*DeclDim*⟩ }

⟨*DeclDim*⟩ ::= **[** [⟨*ListaQualifTipo*⟩] [⟨*QtdElmDim*⟩] **]**
| **[static** [⟨*ListaQualifTipo*⟩] ⟨*QtdElmDim*⟩ **]**
| **[** ⟨*ListaQualifTipo*⟩ **static** ⟨*QtdElmDim*⟩ **]**
| **[** [⟨*ListaQualifTipo*⟩] ***** **]**

⟨*ListaQualifTipo*⟩ ::= ⟨*QualifTipo*⟩ | ⟨*ListaQualifTipo*⟩ ⟨*QualifTipo*⟩

⟨*QtdElmDim*⟩ ::= Expressão do tipo inteiro definindo o tamanho da dimensão do vetor.

Declaração de vetores

`int alunos[3];`	Vetor de `int`, de uma dimensão, com 3 elementos.
`double alunos[2][3];`	Vetor bidimensional de `double`, com 2 elementos na primeira dimensão e 3 na segunda. De fato, vetor de uma dimensão com 2 elementos do tipo vetor de uma dimensão.

Tipo dos vetores

- Quando os elementos de um vetor são de um tipo $\langle T \rangle$, diz-se que a variável que o declara é um *vetor de* $\langle T \rangle$.
- O tipo do vetor é $\langle T \rangle$`[]...[]` (incluindo a especificação de cada dimensão).
- Os vetores de $\langle T \rangle$ podem ter seu tipo:

  Completo. Quando a quantidade de elementos é definida.
  Incompleto. Quando a quantidade de elementos não é definida (a expressão da quantidade é omitida).
  Variável. Quando a quantidade de elementos não é constante.
  Variável (com quantidade não especificada). A quantidade é caracterizada por um asterisco.

Tipo dos vetores

Exemplo

Qual o tipo dos seguintes vetores?

```
int alunos[3];

double alunos[2][3];

char *alunos[];

char *alunos[2 + x];

long [*]
```

Tipo dos vetores

Exemplo

Qual o tipo dos seguintes vetores?

```
int alunos[3];
```
Vetor do tipo `int[3]`. Tipo completo.

```
double alunos[2][3];
```

```
char *alunos[];
```

```
char *alunos[2 + x];
```

```
long [*]
```

Tipo dos vetores

Exemplo

Qual o tipo dos seguintes vetores?

`int alunos[3];`	Vetor do tipo `int[3]`. Tipo completo.
`double alunos[2][3];`	Vetor do tipo `double[2][3]`. Tipo completo.
`char *alunos[];`	
`char *alunos[2 + x];`	
`long [*]`	

Tipo dos vetores

Exemplo

Qual o tipo dos seguintes vetores?

`int alunos[3];`	Vetor do tipo `int[3]`. Tipo completo.
`double alunos[2][3];`	Vetor do tipo `double[2][3]`. Tipo completo.
`char *alunos[];`	Vetor do tipo `char *[]`. Tipo incompleto.
`char *alunos[2 + x];`	
`long [*]`	

Tipo dos vetores

Exemplo

Qual o tipo dos seguintes vetores?

`int alunos[3];`	Vetor do tipo `int[3]`. Tipo completo.
`double alunos[2][3];`	Vetor do tipo `double[2][3]`. Tipo completo.
`char *alunos[];`	Vetor do tipo `char *[]`. Tipo incompleto.
`char *alunos[2 + x];`	Vetor do tipo `char *[2 + x]`. Tipo variável (porém completo).
`long [*]`	

Tipo dos vetores

Exemplo

Qual o tipo dos seguintes vetores?

`int alunos[3];`	Vetor do tipo `int[3]`. Tipo completo.
`double alunos[2][3];`	Vetor do tipo `double[2][3]`. Tipo completo.
`char *alunos[];`	Vetor do tipo `char *[]`. Tipo incompleto.
`char *alunos[2 + x];`	Vetor do tipo `char *[2 + x]`. Tipo variável (porém completo).
`long [*]`	Vetor do tipo `long [*]`. Tipo variável (de tamanho não especificado, porém completo).

Atribuição e referência

- Os elementos de um vetor são modificados apenas individualmente, em atribuições da forma `vetA[indice] = 23`.
- A referência a um elemento é feita indicando-se o índice do elemento, entre os colchetes, após o nome do vetor:
  - o primeiro elemento possui índice 0,
  - o segundo, índice 1,
  - o vigésimo, índice 19, etc.
- Os elementos de um vetor, desde que completamente determinados, podem participar de qualquer operação compatível com o seu tipo.

Atribuição e referência

Exemplo

Considerando a declaração `int vet[150];` e considerando que a variável `x` tenha o valor 17, então:

`vet[3] = 23;`	Atribui o valor 23 ao quarto elemento de `vet`.
`vet[2 * x + 1] = 3.56;`	Atribui o valor 3 (conversão do valor 3,56 do tipo `double` em um valor do tipo `int`) ao trigésimo sexto elemento de `vet`.
`vet[3]++;`	Adiciona 1 ao valor de `vet[3]`, isto é, ao quarto elemento de `vet`.
`--vet[3];`	Subtrai 1 do valor de `vet[3]`.
`(vet[3] > 2)`	Compara o valor de `vet[3]` com o inteiro 2.

Vetores unidimensionais

Exemplo

O programa ao lado lê 10 números inteiros, armazenando-os em um vetor de `int`, e imprime os números lidos, após a leitura.

```
#include <stdio.h>
int main(void) {
  int vetnum[10];
  int num;
  for (int i = 0; i < 10; i++) {
    printf("digite elm %d: ", i);
    scanf("%d", &num);
    vetnum[i] = num;
  }
  for (int i = 0; i < 10; i++) {
    printf("%d ", vetnum[i]);
  }
  return 0;
}
```

Vetores unidimensionais — tamanho variável

Exemplo

O programa ao lado lê uma quantidade de números inteiros especificada pelo usuário, armazenando-os em um vetor de `int`, e imprime os números lidos, após a leitura.

```
#include <stdio.h>
int main(void) {
  int qtd;
  printf("Digite a qtd elms: ");
  scanf("%d", &qtd);
  int vetnum[qtd];
  for (int i = 0; i < qtd; i++) {
    printf("digite elm %d: ", i);
    scanf("%d", &vetnum[i]);
  }
  for (int i = 0; i < qtd; i++) {
    printf("%d ", vetnum[i]);
  }
  return 0;
}
```

Vetores unidimensionais — tamanho variável

Exemplo

O programa ao lado lê uma quantidade de números inteiros especificada pelo usuário, armazenando-os em um vetor de `int`.
Após a leitura a função `imp_vet` é chamada para imprimir o vetor lido.

```
#include <stdio.h>
void imp_vet(int, int [*]);
int main(void) {
  int qtd;
  printf("Digite a qtd elms: ");
  scanf("%d", &qtd);
  int vetnum[qtd];
  for (int i = 0; i < qtd; i++) {
    printf("digite elm %d: ", i);
    scanf("%d", &vetnum[i]);
  }
  imp_vet(qtd, vetnum);
  return 0;
}
void imp_vet(int x, int vet[x]) {
  for (int i = 0; i < x; i++) {
    printf("%d ", vet[i]);
  }
}
```

Vetores unidimensionais — incompletos

Exemplo

prg_vetA.c

```
#include <stdio.h>
extern char estado[];
extern int  qtd;
void inicia_vet(char);
void imp_vet(char[]);
int main(void) {
  inicia_vet('a');
  imp_vet(estado);
  return 0;
}
void imp_vet(char v[]) {
  for (int i = 0; i < qtd; i++) {
    printf("%c ", v[i]);
  }
}
```

prg_vetB.c

```
#define TAM (10)
int  qtd = TAM;
char estado[TAM];
void inicia_vet(char c) {
  for (int i = 0; i < qtd; i++) {
    estado[i] = c++;
  }
}
```

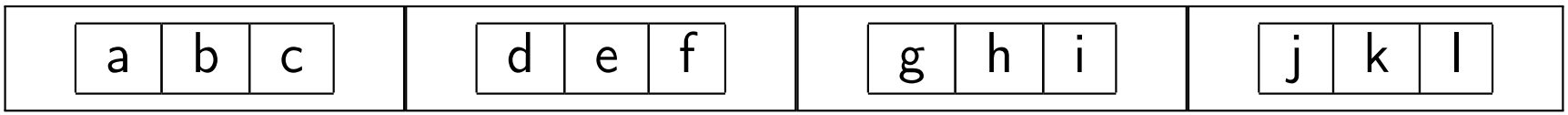
Vetores multidimensionais

Os vetores multidimensionais são implementados em C como vetores de vetores, isto é, vetores cujos elementos são vetores.

O vetor bidimensional que corresponde à matriz

a	b	c
d	e	f
g	h	i
j	k	l

é implementado da seguinte forma:



Vetores multidimensionais

A referência aos elementos dos vetores multidimensionais pode ser

parcial, com a obtenção de um vetor componente, ou

total, com a obtenção de um elemento.

Vetores multidimensionais

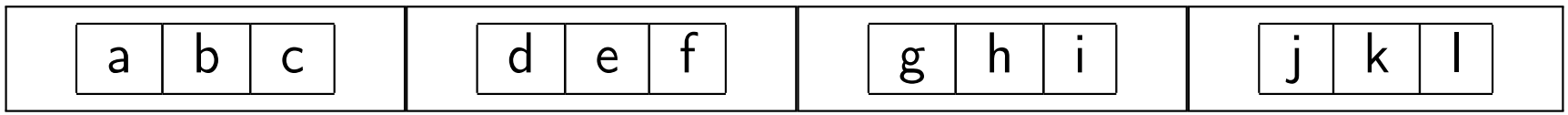
A referência aos elementos dos vetores multidimensionais pode ser

parcial, com a obtenção de um vetor componente, ou

total, com a obtenção de um elemento.

Exemplo

Para uma matriz (vetor bidimensional) `A`:



As seguintes referências são válidas:

`A[0]` = | a | b | c | `A[2]` = | g | h | i | `A[3]` = | j | k | l |

`A[0][2]` = ‘c’ `A[2][1]` = ‘h’ `A[3][0]` = ‘j’

Vetores multidimensionais — declaração

`int alunos[3][2];`	Vetor de `int[2]` (vetor bidimensional de `int`). Tipo completo.
`char *alunos[][2][4];`	Vetor de `char *[2][4]` (vetor tridimensional de `char *`). Tipo incompleto.

Vetores multidimensionais — declaração

`long [][x];`	Vetor de `long[x]` (vetor bidimensional de `long`). Tipo incompleto.
`long [][*]`	Vetor de `long[*]` (vetor bidimensional de `long`). Tipo incompleto.

Vetores multidimensionais — declaração

```
char *alunos[y][2 + x];
```

Vetor de `char *[2 + x]`
(vetor bidimensional de `char *`).
Tipo variável (porém, completo).

```
short int alunos[3][2 + x];
```

Vetor de `short int[2 + x]`
(vetor bidimensional de `short int`).
Tipo variável (porém, completo).

Vetores multidimensionais — tipo

```
long *vetA[2][3]
```

Expressão	Tipo
`vetA`	`long *[2][3]`
`vetA[1]`	`long *[3]`
`vetA[1][2]`	`long *`

Vetores multidimensionais — tipo

```
long *vetA[2][3]
```

Expressão	Tipo
`vetA`	`long *[2][3]`
`vetA[1]`	`long *[3]`
`vetA[1][2]`	`long *`

```
const char vetB[2][3][x]
```

Expressão	Tipo
`vetB`	`const char[2][3][x]`
`vetB[1]`	`const char[3][x]`
`vetB[1][2]`	`const char[x]`
`vetB[1][0][1]`	`const char`

Vetores multidimensionais

Exemplo

O programa ao lado lê dois números, L e C, ambos maiores que 0, e depois lê, coluna a coluna, os números correspondentes aos elementos de uma matriz de ordem $L \times C$. Após a leitura, a matriz lida é impressa.

Continua...

```
#include <stdio.h>
int main(void) {
  int L, C;
  do {
    printf("Digite a qtd de linhas: ");
    scanf("%d", &L);
  } while (L <= 0);
  do {
    printf("Digite a qtd de colunas: ");
    scanf("%d", &C);
  } while (C <= 0);
  int mat[L][C];
  printf("Digite, coluna a coluna, ");
  printf("os elementos de uma ");
  printf("matriz %d x %d\n", L, C);
```

Vetores multidimensionais

Exemplo

Continuação.

```
  for (int j = 0; j < C; j++) {
    for (int i = 0; i < L; i++) {
      printf("elm (%d,%d):", (i+1), (j+1));
      scanf("%d", &mat[i][j]);
    }
  }
  for (int i = 0; i < L; i++) {
    for (int j = 0; j < C; j++) {
      printf("%d ", mat[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

Vetores multidimensionais

Exemplo

A função ao lado recebe dois inteiros L e C e uma matriz de L linhas e C colunas, e imprime a matriz recebida.

```
void imp_vet(int L, int C, int m[L][C]) {
  for (int i = 0; i < L; i++) {
    for (int j = 0; j < C; j++) {
      printf("%d ", m[i][j]);
    }
    printf("\n");
  }
}
```

Iniciação de vetores

Os elementos do vetor são associados aos valores da lista de iniciação, segundo o seguinte processo:

1. Se o valor corrente pode ser atribuído ao elemento corrente, a atribuição é realizada e o processo prossegue com o próximo elemento do vetor e o próximo valor da lista, que serão os novos elemento e valor correntes.
2. Se o valor corrente é também uma lista, a atribuição de valor ao elemento corrente e a seus subelementos (se ele for um vetor) fica restrita aos valores da lista que corresponde ao valor corrente.
3. Se o elemento corrente é também um vetor, o processo prossegue recursivamente, associando o primeiro elemento desse vetor ao valor corrente, até que a atribuição seja realizada.

- Os elementos não iniciados de um vetor assumem o valor padrão que corresponde ao seu tipo.
- Os valores excedentes da lista de iniciação são ignorados.

Iniciação de vetores

Exemplo

```
char *fatorRH[8][2] =   {{"A", "+"}, {"A", "-"}, {"B", "+"}, {"B", "-"},
                         {"O", "+"}, {"AB", "-"}, {"O", "-"}};
```

1. `fatorRH[0]` e `{"A", "+"}`
   - `fatorRH[0][0]` ← `"A"`
   - `fatorRH[0][1]` ← `"+"`
2. `fatorRH[1]` e `{"A", "-"}`
   - `fatorRH[1][0]` ← `"A"`
   - `fatorRH[1][1]` ← `"-"`
3. `fatorRH[2]` e `{"B", "+"}`
   - `fatorRH[2][0]` ← `"B"`
   - `fatorRH[2][1]` ← `"+"`
4. `fatorRH[3]` e `{"B", "-"}`
5. ...

Iniciação de vetores

Exemplo

```
char *fatorRH[8][2] =   {"A", "+", "A", {"B", "+"}, {"B"}, {"O", "+"},
                         "O", "+", "AB", "-"};
```

1. fatorRH[0] e "A"
   - fatorRH[0][0] ← "A"
   - fatorRH[0][1] ← "+"
2. fatorRH[1] e "A"
   - fatorRH[1][0] ← "A"
   - fatorRH[1][1] ← {"B", "+"} ; fatorRH[1][1] ← "B"
3. fatorRH[2] e {"B"}
   - fatorRH[2][0] ← "B"
   - fatorRH[2][1] ← NULL
4. fatorRH[3] e {"O", "+"}
5. ...

Iniciação seletiva

Exemplo

```
char *fatorRH[8][2] =   {{"A", "+"}, [5]={"O", "-"}, {"AB", "+"},
                         [1]={"A", "-"}, {[1]= "+"}};
```

1. fatorRH[0] e {"A", "+"}
   - fatorRH[0][0] ← "A"
   - fatorRH[0][1] ← "+"
2. fatorRH[5] e {"A", "-"}
   - fatorRH[5][0] ← "O"
   - fatorRH[5][1] ← "-"
3. fatorRH[6] e {"AB", "+"}
   - fatorRH[6][0] ← "AB"
   - fatorRH[6][1] ← "+"
4. fatorRH[1] e {"A", "-"}
5. ...

Iniciação com cadeias de caracteres e literais compostos

Cadeias de caracteres

Cada caractere da cadeia, incluindo o caractere nulo ao final, inicia um elemento do vetor.

```
char vogais[] = "aeiou";
char let_ini[5] = "abcde";
char let_meio[5] = "klmnop";
char let_fim[5] = "xyz";
```

Iniciação com cadeias de caracteres e literais compostos

Cadeias de caracteres

Cada caractere da cadeia, incluindo o caractere nulo ao final, inicia um elemento do vetor.

```
char vogais[] = "aeiou";
char let_ini[5] = "abcde";
char let_meio[5] = "klmnop";
char let_fim[5] = "xyz";
```

Literais compostos

Os literais compostos podem iniciar vetores declarados como ponteiros.

```
int *primos = (int []){2, 3, 5, 7};
int *perfeitos = (int [4]){6, 28, 496, 8128};
```

Ponteiros e vetores

Na avaliação de expressões:

- Toda expressão do tipo *vetor de* $\langle T \rangle$ é convertida em uma expressão do tipo *ponteiro para* $\langle T \rangle$, cujo valor é um ponteiro apontando para o primeiro elemento do vetor.

  Exceções: operandos dos operadores `sizeof` *e* `&`*, e os literais cadeia de caracteres usados em expressões de iniciação.*

- As declarações de parâmetros de função do tipo *vetor de* $\langle T \rangle$ são ajustadas para declarações de parâmetros do tipo *ponteiro (possivelmente qualificado) para* $\langle T \rangle$.

Ponteiros e vetores

Exemplo

Considerando `vet` declarado como `char vet[3][2][4]`, então

Expressão	Tipo	Convertido em
`vet`	`char[3][2][4]`	`char (*)[2][4]`.
`vet[x]`	`char[2][4]`	`char (*)[4]`.
`vet[1][0]`	`char[4]`	`char *`.
`vet[x][y][w]`	`char`	Sem conversão a ponteiro.

Aritmética de ponteiros

Para uma variável `ptrA` do tipo *ponteiro para* $\langle T \rangle$ e um valor inteiro `qtd`:

1. A operação `ptrA + qtd` ou `qtd + ptrA` resulta no endereço que se obtém somando `qtd` $\times$ (tamanho de $\langle T \rangle$) a `ptrA`. Esse resultado é do tipo *ponteiro para* $\langle T \rangle$.

2. A operação `ptrA - qtd` resulta no endereço que se obtém subtraindo `qtd` $\times$ (tamanho de $\langle T \rangle$) de `ptrA`. Esse resultado é do tipo *ponteiro para* $\langle T \rangle$.
   - A operação `qtd - ptrA` não é definida.

Aritmética de ponteiros

Para variáveis `ptrA` e `ptrB` do tipo *ponteiro para* $\langle T \rangle$:

3. A operação `ptrA - ptrB` resulta no valor inteiro que corresponde à distância (orientada) entre os endereços `ptrA` e `ptrB` medida em termos do tamanho de $\langle T \rangle$: (endereço `ptrA` - endereço `ptrB`)/(tamanho de $\langle T \rangle$). Essa distância pode ser negativa e o valor obtido é do tipo `ptrdiff_t`.
   - Equivale à subtração dos índices que os ponteiros representam.
   - Definida para ponteiros que apontam para o mesmo tipo $\langle T \rangle$.

4. A operação `ptrA + ptrB` não é permitida.

Referenciando elementos dos vetores com ponteiros

- Para um vetor unidimensional `vet`, a referência
  - `vet[i]` corresponde a `*(vet + i)`.

Referenciando elementos dos vetores com ponteiros

- Para um vetor unidimensional `vet`, a referência
  - `vet[i]` corresponde a `*(vet + i)`.
- Para um vetor bidimensional `vet`, a referência
  - `vet[i][j]` corresponde a `*(*(vet + i) + j)`.

Referenciando elementos dos vetores com ponteiros

- Para um vetor unidimensional `vet`, a referência
  - `vet[i]` corresponde a `*(vet + i)`.
- Para um vetor bidimensional `vet`, a referência
  - `vet[i][j]` corresponde a `*(*(vet + i) + j)`.
- Para um vetor multidimensional `vet`, a referência
  - `vet`$[i_1][i_2]\ldots[i_n]$ corresponde a `*(...*(*(vet +` i_1`) +` i_2`) + ... +` i_n`)`.

Referenciando elementos dos vetores com ponteiros

Exemplo

O programa ao lado lê e imprime, na ordem inversa à que foram digitados, um vetor de N números.

```
#include <stdio.h>
int main(void) {
  int N;
  do {
    scanf("%d", &N);
  } while ((N <= 0) || (N > 1000));
  int lval[N];
  for (int i = 0; i < N; i++) {
    scanf("%d", (lval + i));
  }
  for (int i = N - 1; i >= 0; i--) {
    if ((*(lval + i) % 2) == 0) {
      printf("%d ", *(lval + i));
    }
  }
  return 0;
}
```

Referenciando elementos dos vetores com ponteiros

Dois modos de usar ponteiros para referenciar os elementos de uma matriz:

Apenas ponteiros

```
void imp_vet(int L, int C, int m[L][C]) {
  for (int i = 0; i < L; i++) {
    for (int j = 0; j < C; j++) {
      printf("%d ", *(*(m + i) + j));
    }
    printf("\n");
  }
}
```

Ponteiros e índices

```
void imp_vet(int L, int C, int m[L][C]) {
  for (int i = 0; i < L; i++) {
    for (int j = 0; j < C; j++) {
      printf("%d ", (*(m + i))[j]);
    }
    printf("\n");
  }
}
```

Definindo tipos vetores

Na definição de tipos vetores com `typedef` usa-se a declaração do novo tipo como se fosse uma variável do tipo vetor que se desejar substituir:

`typedef int vA_t[10]`	Declara o tipo `vA_t` como sinônimo de `int [10]`.
`typedef char vB_t[3][6]`	Declara o tipo `vB_t` como sinônimo de `char [3][6]`.
`typedef float vC_t[][34]`	Declara o tipo `vC_t` como sinônimo de `float [][34]`.

Qualificando as variáveis do tipo vetor

`int const vet[x]`	A variável `vet` é declarada como um vetor de `int const`.
`int vet[const x]`	A variável `vet` é constante, declarada como um vetor de `int`.
`int vet[static 23]`	A variável `vet` é um vetor de `int`, com (*a expectativa de*) pelo menos 23 elementos.

Qualificando as variáveis do tipo vetor

`int const vet[x]`	A variável `vet` é declarada como um vetor de `int const`.
`int vet[const x]`	A variável `vet` é constante, declarada como um vetor de `int`.
`int vet[static 23]`	A variável `vet` é um vetor de `int`, com (*a expectativa de*) pelo menos 23 elementos.

Observação

A qualificação de variáveis do tipo vetor só pode ocorrer em protótipos e declaração de parâmetros.

Compatibilidade de vetores e ponteiros

Dois vetores são compatíveis se

- o tipo dos seus elementos são compatíveis,
- possuem as mesmas dimensões e
- os especificadores de tamanho, se existirem e forem constantes, têm o mesmo valor.

Compatibilidade de vetores e ponteiros

Dois vetores são compatíveis se

- o tipo dos seus elementos são compatíveis,
- possuem as mesmas dimensões e
- os especificadores de tamanho, se existirem e forem constantes, têm o mesmo valor.

Dois ponteiros são compatíveis se

- possuem os mesmos qualificadores e
- apontam para tipos compatíveis.

Bibliografia

ISO/IEC

C Programming Language Standard

ISO/IEC 9899:2011, International Organization for Standardization; International Electrotechnical Commission, 3rd edition, WG14/N1570 Committee final draft, abril de 2011.

Francisco A. C. Pinheiro

Elementos de programação em C

Bookman, Porto Alegre, 2012.

www.bookman.com.br, www.facp.pro.br/livroc