LPG0002 – Linguagem de Programação

Vetores de Ponteiros e Ponteiros para Ponteiros

Profª Luciana Rita Guedes
Departamento de Ciência da Computação
UDESC / Joinville

Material elaborado por: Prof. Rui Jorge Tramontin Junior

Tópicos abordados na aula

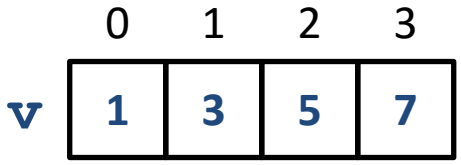
- Vetor de ponteiros;
- Parâmetros da *main()*;
- Ponteiros para ponteiros;
- Exemplos:
  - Alocação de uma matriz;
  - Alocação de um vetor de *strings* (exemplo prático);

Recapitulando os conceitos...

```
int v[4] = { 1, 3, 5, 7 };
```

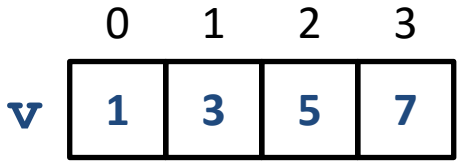
Recapitulando os conceitos...

```
// Vetor de inteiros "estático"
int v[4] = { 1, 3, 5, 7 };
```



Recapitulando os conceitos...

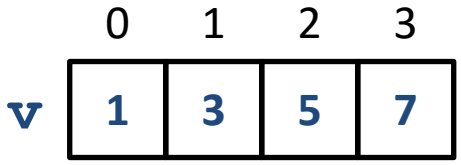
```
// Vetor de inteiros "estático"
int v[4] = { 1, 3, 5, 7 };
```



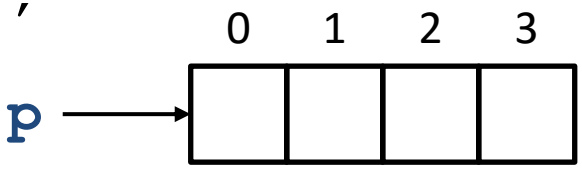

```
int *p = malloc( sizeof(int) * 4 );
```

Recapitulando os conceitos...

```
// Vetor de inteiros "estático"
int v[4] = { 1, 3, 5, 7 };
```

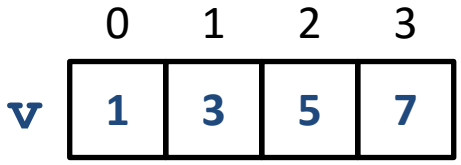



```
// Vetor de inteiros dinâmico
int *p = malloc( sizeof(int) * 4 );
```

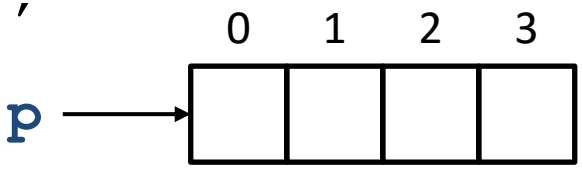


Recapitulando os conceitos...

```
// Vetor de inteiros "estático"
int v[4] = { 1, 3, 5, 7 };
```



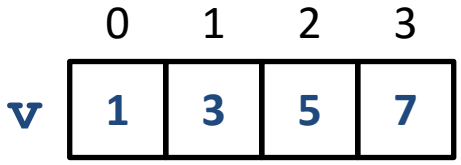

```
// Vetor de inteiros dinâmico
int *p = malloc( sizeof(int) * 4 );
```



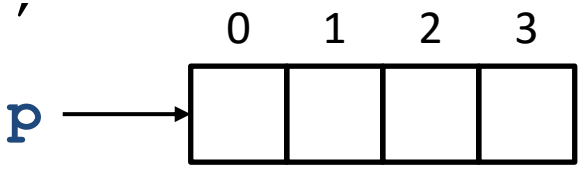

```
int *vp[4];
```

Recapitulando os conceitos...

```
// Vetor de inteiros "estático"
int v[4] = { 1, 3, 5, 7 };
```



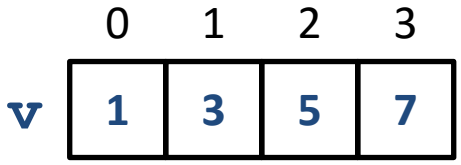

```
// Vetor de inteiros dinâmico
int *p = malloc( sizeof(int) * 4 );
```



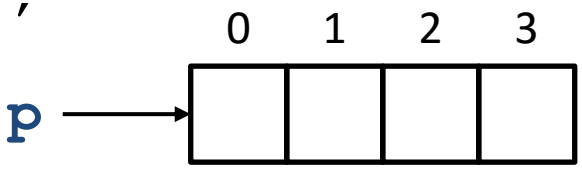

```
// O que é isso?
int *vp[4];
```

Recapitulando os conceitos...

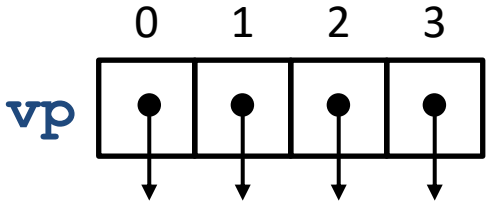
```
// Vetor de inteiros "estático"
int v[4] = { 1, 3, 5, 7 };
```




```
// Vetor de inteiros dinâmico
int *p = malloc( sizeof(int) * 4 );
```




```
// Vetor de ponteiros (int)!
int *vp[4];
```

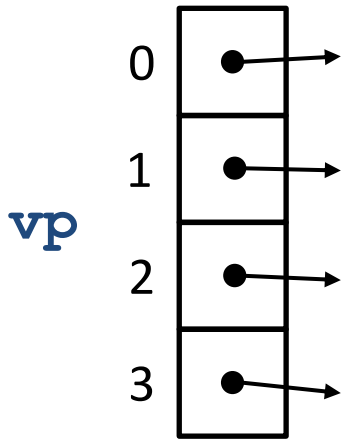


Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];
```

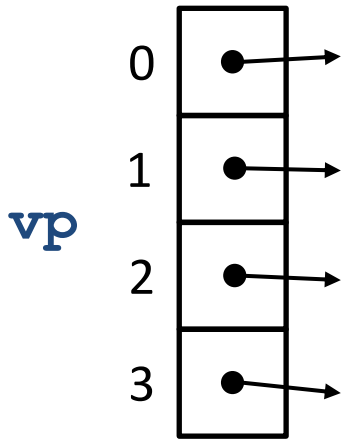
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];
```



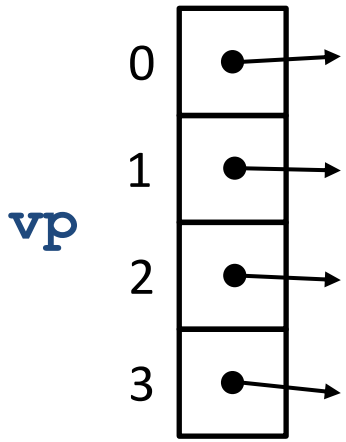
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
```



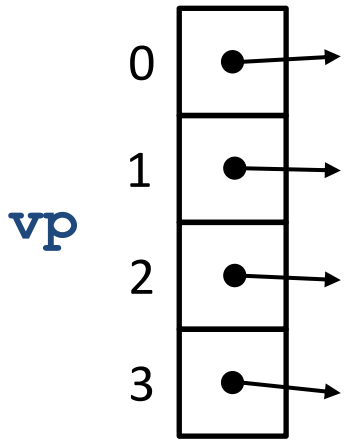
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```



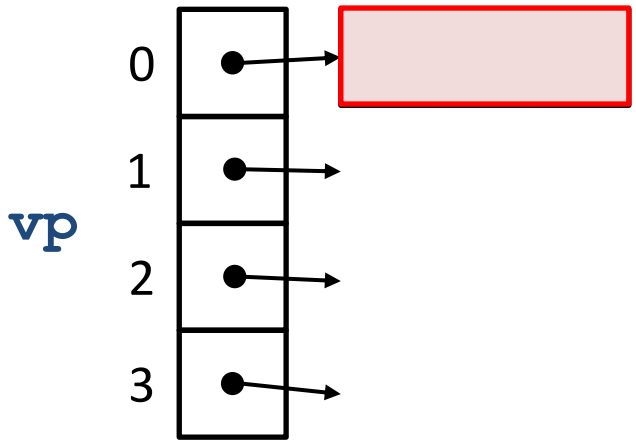
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```



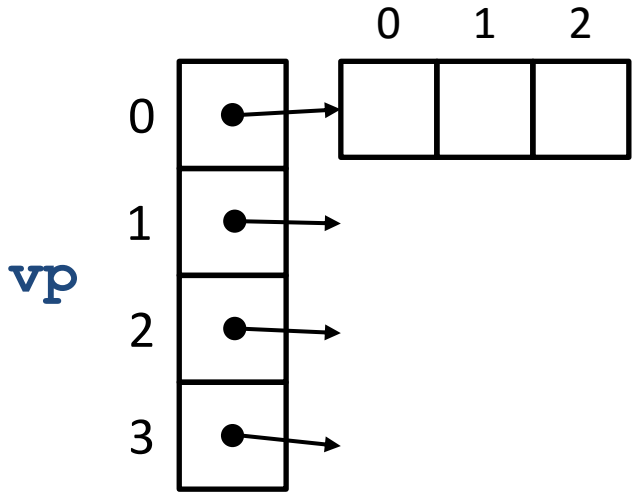
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 0
```



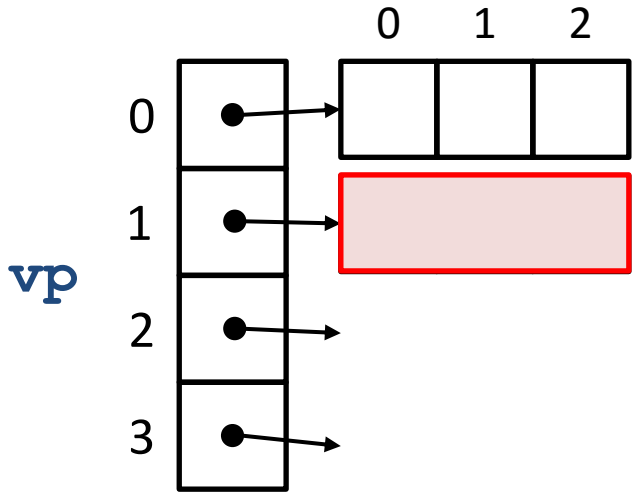
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 0
```



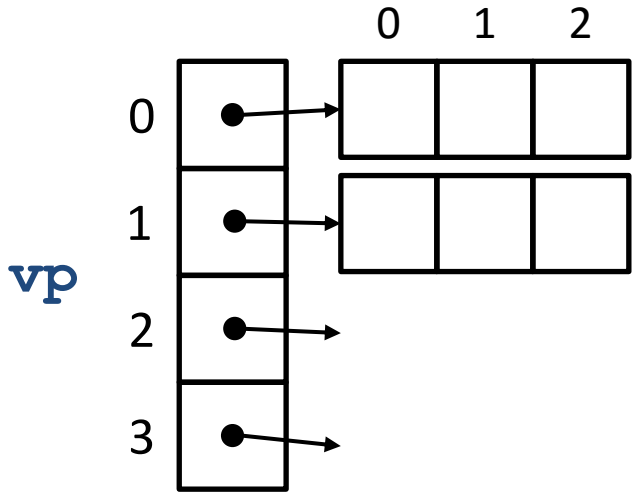
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 1
```



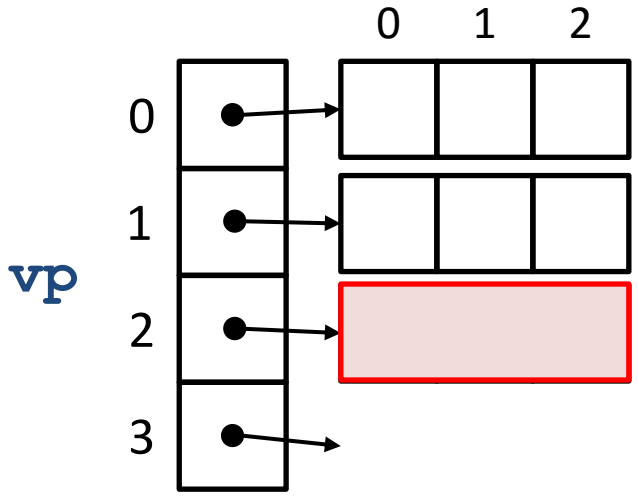
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 1
```



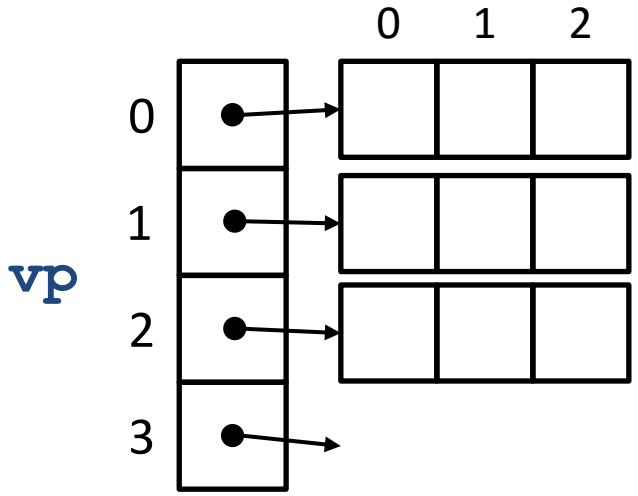
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 2
```



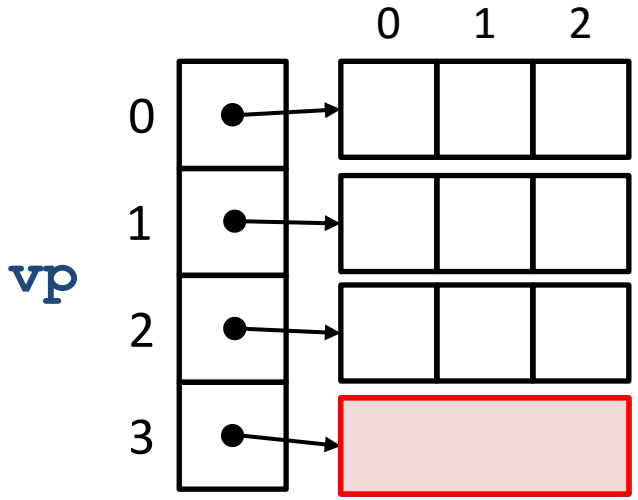
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 2
```



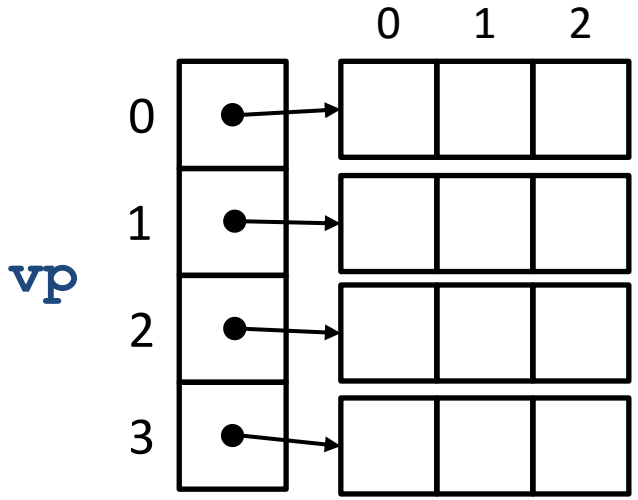
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 3
```



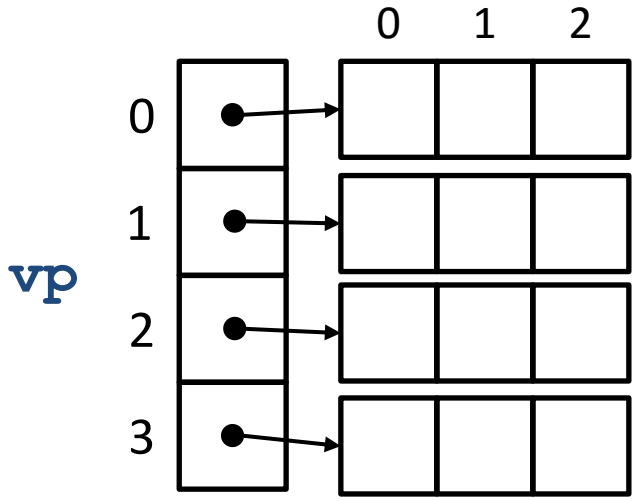
Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );  // i = 3
```



Alocando os ponteiros do vetor

```
// Vetor de ponteiros
int *vp[4];

int i;
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```



Equivalente a uma matriz!

```
int m[4][3];
```

Considerações

- Um ***ponteiro*** pode ser usado para alocar um **vetor** dinamicamente;

Considerações

- Um ***ponteiro*** pode ser usado para alocar um **vetor** dinamicamente;

- Um ***vetor de ponteiros*** pode ser usado para alocar uma **matriz** dinamicamente;

Considerações

- Um ***ponteiro*** pode ser usado para alocar um **vetor** dinamicamente;
- Um ***vetor de ponteiros*** pode ser usado para alocar uma **matriz** dinamicamente;
- Porém, somente o número de colunas é dinâmico;

Considerações

- Um ***ponteiro*** pode ser usado para alocar um **vetor** dinamicamente;

- Um ***vetor de ponteiros*** pode ser usado para alocar uma **matriz** dinamicamente;

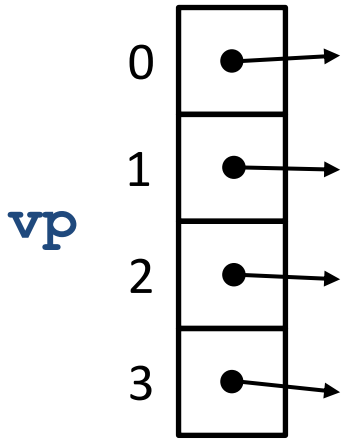
- Porém, somente o número de colunas é dinâmico;
  - O número de linhas é a capacidade do vetor!

Exemplo 1: matriz semidinâmica

```
int *vp[4]; // Vetor de ponteiros
int i, j;
```

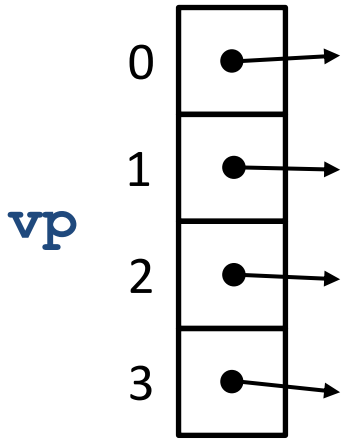
Exemplo 1: matriz semidinâmica

```
int *vp[4]; // Vetor de ponteiros
int i, j;
```



Exemplo 1: matriz semidinâmica

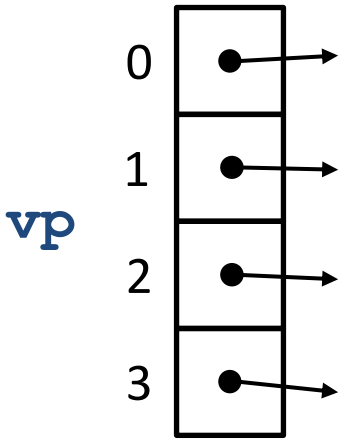
```
int *vp[4]; // Vetor de ponteiros
int i, j;

// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```



Exemplo 1: matriz semidinâmica

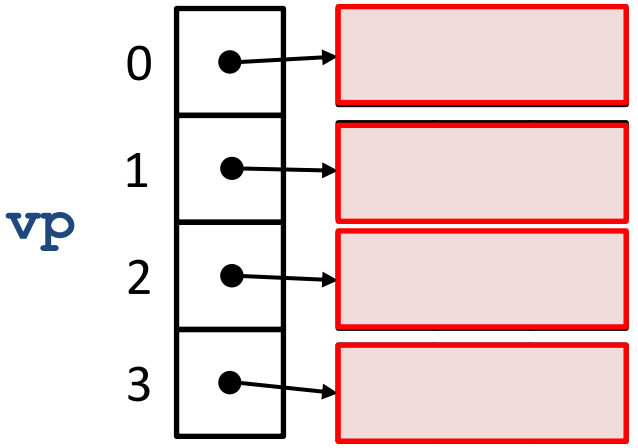
```
int *vp[4]; // Vetor de ponteiros
int i, j;
```

```
// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```



Exemplo 1: matriz semidinâmica

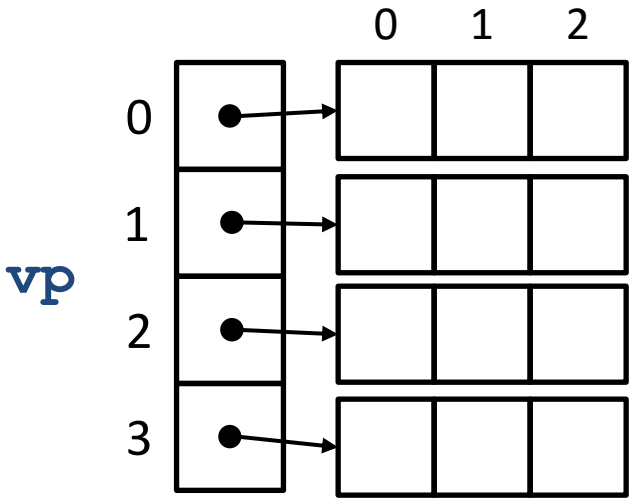
```
int *vp[4]; // Vetor de ponteiros
int i, j;

// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );

// Aloca as linhas da matriz
```



Exemplo 1: matriz semidinâmica

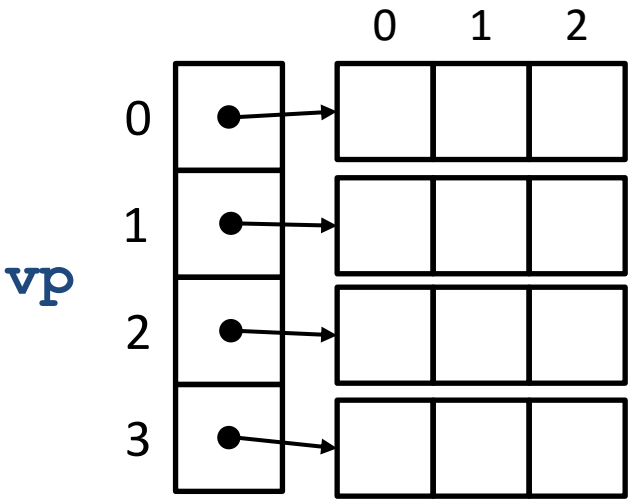
```
int *vp[4]; // Vetor de ponteiros
int i, j;
```

```
// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```



Exemplo 1: matriz semidinâmica

```
int *vp[4]; // Vetor de ponteiros
int i, j;

// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );

// Entrada de dados
for( i = 0 ; i < 4 ; i++ )
  for( j = 0 ; j < 3 ; j++ )
    scanf("%d", &vp[i][j]);
```

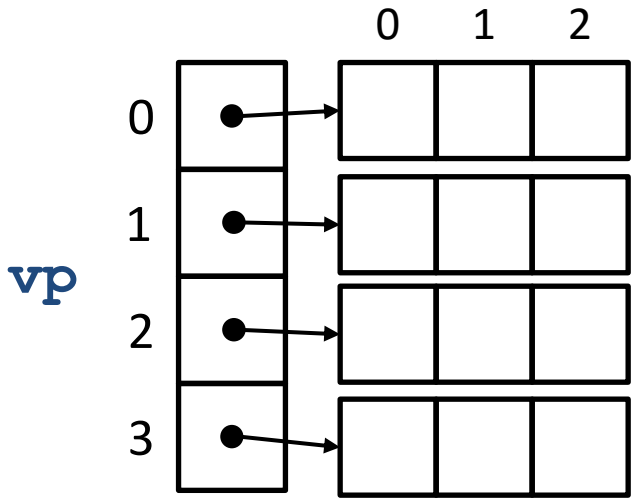


Exemplo 1: matriz semidinâmica

```
int *vp[4]; // Vetor de ponteiros
int i, j;

// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );
```

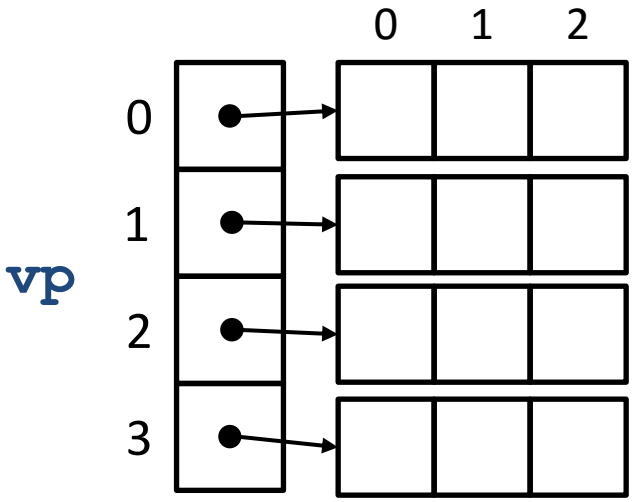
```
// Entrada de dados
for( i = 0 ; i < 4 ; i++ )
  for( j = 0 ; j < 3 ; j++ )
    scanf("%d", &vp[i][j]);
```

```
// Tal como em uma matriz!
```



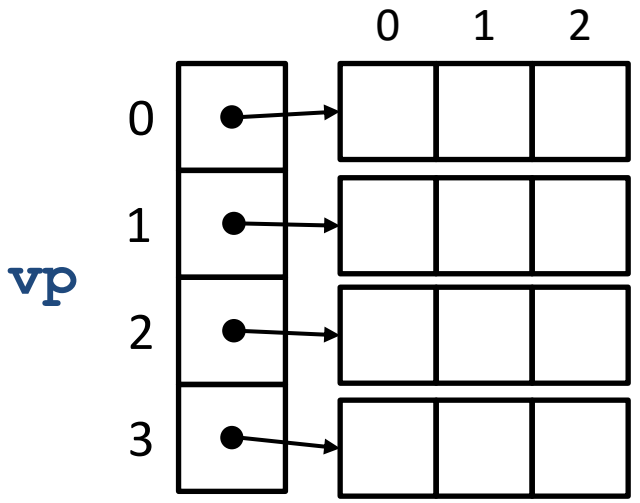
Exemplo 1: matriz semidinâmica

```
int *vp[4]; // Vetor de ponteiros
int i, j;

// Alocação
for( i = 0 ; i < 4 ; i++ )
  vp[i] = malloc( sizeof(int) * 3 );

// Entrada de dados
for( i = 0 ; i < 4 ; i++ )
  for( j = 0 ; j < 3 ; j++ )
    scanf("%d", &vp[i][j]);

// Continua...
```



Exemplo 1: matriz semidinâmica

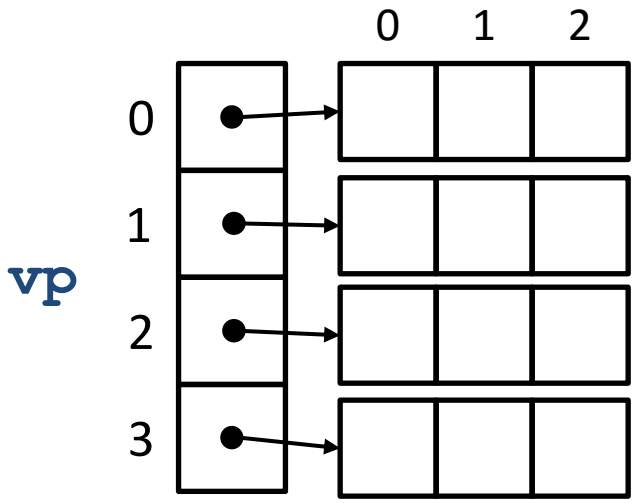
```
// Saída dos dados
for( i = 0 ; i < 4 ; i++ ){
  for( j = 0 ; j < 3 ; j++ )
    printf("%d ", vp[i][j]);
  printf("\n");
}
```



Exemplo 1: matriz semidinâmica

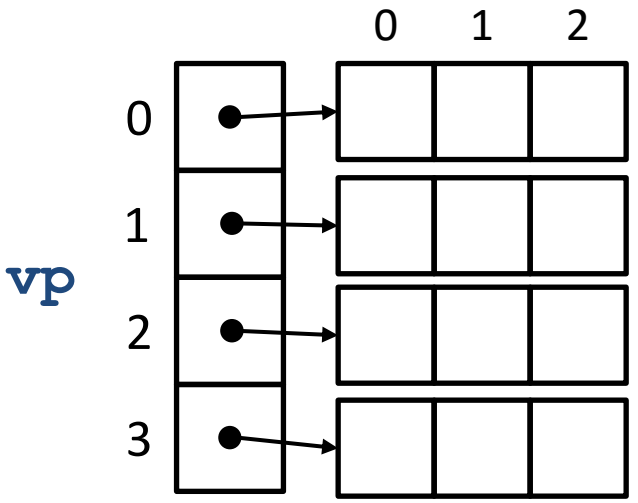
```
// Saída dos dados
for( i = 0 ; i < 4 ; i++ ){
  for( j = 0 ; j < 3 ; j++ )
    printf("%d ", vp[i][j]);
  printf("\n");
}
```

```
// Tal como em uma matriz!
```



Exemplo 1: matriz semidinâmica

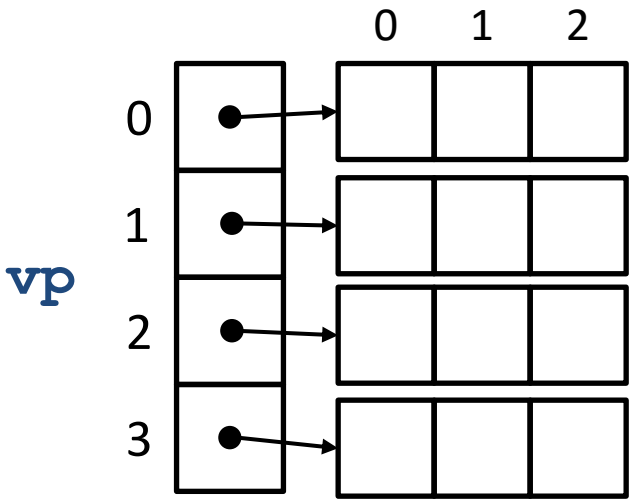
```
// Saída dos dados
for( i = 0 ; i < 4 ; i++ ){
  for( j = 0 ; j < 3 ; j++ )
    printf("%d ", vp[i][j]);
  printf("\n");
}

// Liberação da memória
for( i = 0 ; i < 4 ; i++ )
  free( vp[i] );
```



Exemplo 1: matriz semidinâmica

```
// Saída dos dados
for( i = 0 ; i < 4 ; i++ ){
  for( j = 0 ; j < 3 ; j++ )
    printf("%d ", vp[i][j]);
  printf("\n");
}
```

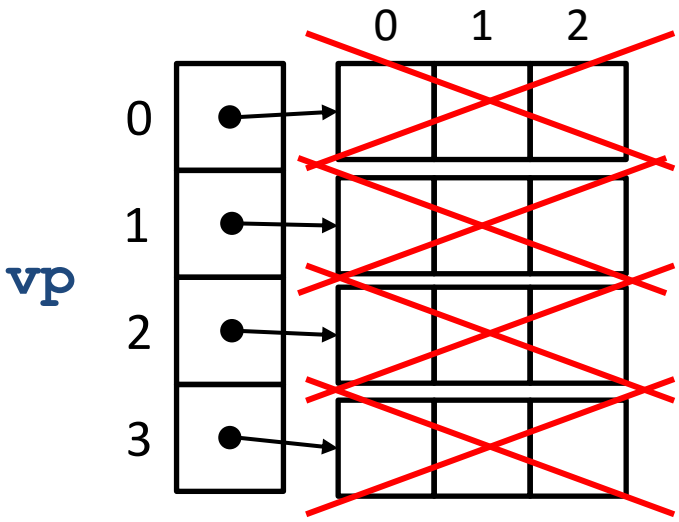
```
// Liberação da memória
for( i = 0 ; i < 4 ; i++ )
  free( vp[i] );
```



Exemplo 1: matriz semidinâmica

```
// Saída dos dados
for( i = 0 ; i < 4 ; i++ ){
  for( j = 0 ; j < 3 ; j++ )
    printf("%d ", vp[i][j]);
  printf("\n");
}
```

```
// Liberação da memória
for( i = 0 ; i < 4 ; i++ )
  free( vp[i] );
```



Exemplo 2: parâmetros da *main()*

- Um programa em C consegue receber uma entrada de dados diretamente do *prompt* de comando;

Exemplo 2: parâmetros da *main()*

- Um programa em C consegue receber uma entrada de dados diretamente do *prompt* de comando;

- São armazenados nos parâmetros da função *main()*:

Exemplo 2: parâmetros da *main()*

- Um programa em C consegue receber uma entrada de dados diretamente do *prompt* de comando;

- São armazenados nos parâmetros da função *main()*:
  - **`int argc`** : quantidade de palavras digitadas;

Exemplo 2: parâmetros da *main()*

- Um programa em C consegue receber uma entrada de dados diretamente do *prompt* de comando;

- São armazenados nos parâmetros da função *main()*:
  - **`int argc`** : quantidade de palavras digitadas;
  - **`char *argv[]`** : vetor de *strings*, contém as palavras.

Exemplo 2: parâmetros da *main()*

- Considere que o arquivo executável do programa se chama *teste.exe*, e que o usuário digitou mais algumas palavras no *prompt* de comando (parâmetros):

Exemplo 2: parâmetros da *main()*

- Considere que o arquivo executável do programa se chama *teste.exe*, e que o usuário digitou mais algumas palavras no *prompt* de comando (parâmetros):

```
teste alô mundo 123
```

Exemplo 2: parâmetros da *main()*

- Considere que o arquivo executável do programa se chama *teste.exe*, e que o usuário digitou mais algumas palavras no *prompt* de comando (parâmetros):

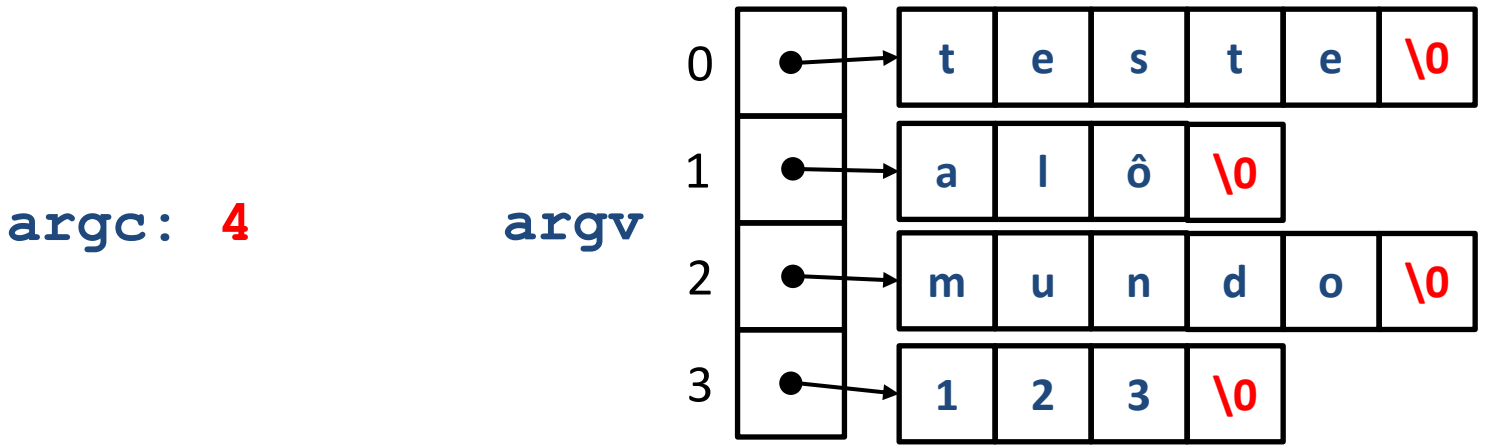
```
teste alô mundo 123
```

```
argc: 4
```

Exemplo 2: parâmetros da *main()*

- Considere que o arquivo executável do programa se chama *teste.exe*, e que o usuário digitou mais algumas palavras no *prompt* de comando (parâmetros):

```
teste alô mundo 123
```



Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){




   return 0;
}
```

Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){




   return 0;
}
```

Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){
   // Mostra o que foi digitado...



   return 0;
}
```

Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){
  // Mostra o que foi digitado...
  for( i = 0 ; i < argc ; i++ )


  return 0;
}
```

Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){
  // Mostra o que foi digitado...
  for( i = 0 ; i < argc ; i++ )
    printf("%s\n", argv[i]);

  return 0;
}
```

Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){
  // Mostra o que foi digitado...
  for( i = 0 ; i < argc ; i++ )
    printf("%s\n", argv[i]);

  return 0;
}
```

Exemplo 2: parâmetros da *main()*

```
#include <stdio.h>

int main( int argc, char *argv[] ){
   // Mostra o que foi digitado...
   for( i = 0 ; i < argc ; i++ )
     printf("%s\n", argv[i]);

   return 0;
}
```

```
teste
alô
mundo
123
```

Considerações

- O uso dos parâmetros da *main()* é feito normalmente por utilitários executados via linha de comando;

Considerações

- O uso dos parâmetros da *main()* é feito normalmente por utilitários executados via linha de comando;

- Um exemplo típico é o GCC, que recebe vários parâmetros, incluindo os arquivos a serem compilados;

Considerações

- O uso dos parâmetros da *main()* é feito normalmente por utilitários executados via linha de comando;
- Um exemplo típico é o GCC, que recebe vários parâmetros, incluindo os arquivos a serem compilados;
- Note que os valores digitados são *strings*, sendo necessário converter para outros tipos (*int* ou *float*), caso necessário;

Considerações

- O uso dos parâmetros da *main()* é feito normalmente por utilitários executados via linha de comando;
- Um exemplo típico é o GCC, que recebe vários parâmetros, incluindo os arquivos a serem compilados;
- Note que os valores digitados são *strings*, sendo necessário converter para outros tipos (*int* ou *float*), caso necessário;
  - Uma função útil para isso é **`sscanf()`**.

Mais considerações...

- O uso de vetores de ponteiros acaba sendo limitado, pois sua capacidade é fixa:

Mais considerações...

- O uso de vetores de ponteiros acaba sendo limitado, pois sua capacidade é fixa:
  - Linhas da matriz;

Mais considerações...

- O uso de vetores de ponteiros acaba sendo limitado, pois sua capacidade é fixa:
  - Linhas da matriz;
  - Quantidade de *strings*;

Mais considerações...

- O uso de vetores de ponteiros acaba sendo limitado, pois sua capacidade é fixa:
  - Linhas da matriz;
  - Quantidade de *strings*;

- É preciso, portanto, fazer a ***alocação dinâmica*** do ***vetor de ponteiros***;

Mais considerações...

- O uso de vetores de ponteiros acaba sendo limitado, pois sua capacidade é fixa:
  - Linhas da matriz;
  - Quantidade de *strings*;

- É preciso, portanto, fazer a ***alocação dinâmica*** do ***vetor de ponteiros***;

- Como isso é feito?

Mais considerações...

- O uso de vetores de ponteiros acaba sendo limitado, pois sua capacidade é fixa:
  - Linhas da matriz;
  - Quantidade de *strings*;

- É preciso, portanto, fazer a ***alocação dinâmica*** do ***vetor de ponteiros***;

- Como isso é feito? → **Ponteiros para ponteiros!**

Ponteiros para ponteiros

- Definição óbvia: *ponteiro para ponteiros* é aquele que aponta para outro ponteiro (ao invés de informação em si);

Ponteiros para ponteiros

- Definição óbvia: *ponteiro para ponteiros* é aquele que aponta para outro ponteiro (ao invés de informação em si);

- Ou seja, temos mais um nível de endereçamento;

Ponteiros para ponteiros

- Definição óbvia: *ponteiro para ponteiros* é aquele que aponta para outro ponteiro (ao invés de informação em si);

- Ou seja, temos mais um nível de endereçamento;
  - À primeira vista, parece desnecessário;

Ponteiros para ponteiros

- Definição óbvia: *ponteiro para ponteiros* é aquele que aponta para outro ponteiro (ao invés de informação em si);

- Ou seja, temos mais um nível de endereçamento;
  - À primeira vista, parece desnecessário;

- Porém, é fundamental para a alocação dinâmica de ***matrizes*** e ***vetores de strings***.

Ponteiros para ponteiros

- A declaração é feita com **, indicando que temos um ponteiro para ponteiros;

Ponteiros para ponteiros

- A declaração é feita com **, indicando que temos um ponteiro para ponteiros;

- Na prática, temos uma variável que guarda um endereço (mas agora de outro ponteiro);

Ponteiros para ponteiros

- A declaração é feita com **, indicando que temos um ponteiro para ponteiros;
- Na prática, temos uma variável que guarda um endereço (mas agora de outro ponteiro);
- O operador de indireção (*) deve ser utilizado duas vezes para se chegar à informação;

Ponteiros para ponteiros

- A declaração é feita com **, indicando que temos um ponteiro para ponteiros;
- Na prática, temos uma variável que guarda um endereço (mas agora de outro ponteiro);
- O operador de indireção (*) deve ser utilizado duas vezes para se chegar à informação;
  - Alternativamente, pode-se utilizar *dois pares de colchetes* → notação de matrizes!

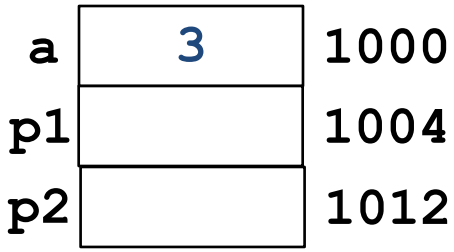
Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;
```

Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;
```

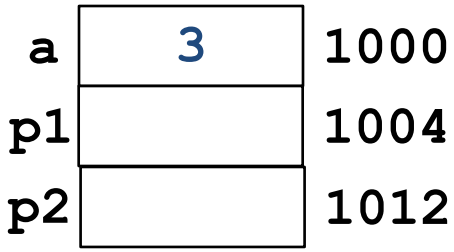
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;
```

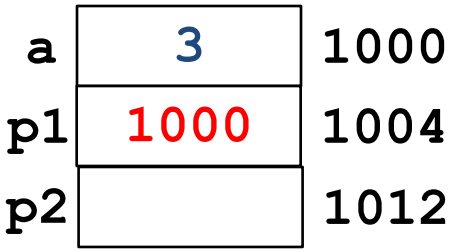
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
```

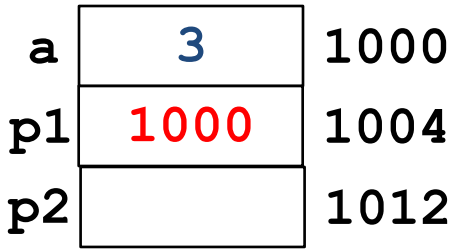
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1;
```

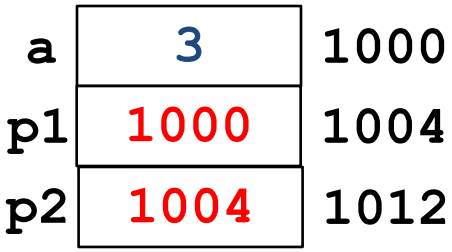
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1; // endereço de p1.
```

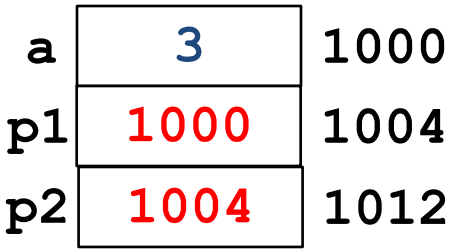
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1; // endereço de p1.

// Mostra endereço da variável p1.
printf("Endereço de P1 = %d\n", p2 );
```

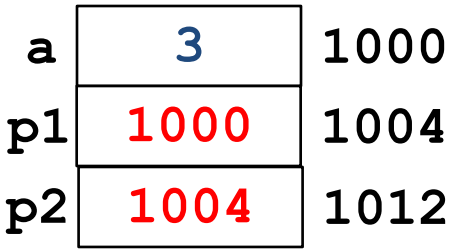
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1; // endereço de p1.

// Mostra endereço da variável p1.
printf("Endereço de P1 = %d\n", p2 );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", *p2 );
```

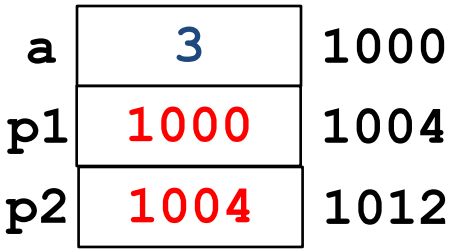
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1; // endereço de p1.

// Mostra endereço da variável p1.
printf("Endereço de P1 = %d\n", p2 );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", *p2 ); // p2[0]
```

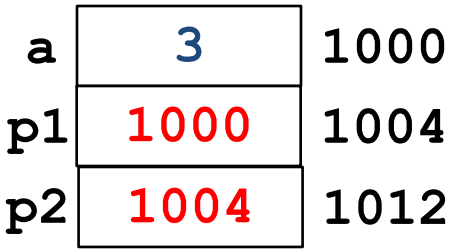
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1; // endereço de p1.

// Mostra endereço da variável p1.
printf("Endereço de P1 = %d\n", p2 );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", *p2 ); // p2[0]

// Mostra valor da variável a.
printf("A = %d\n", **p2 );
```

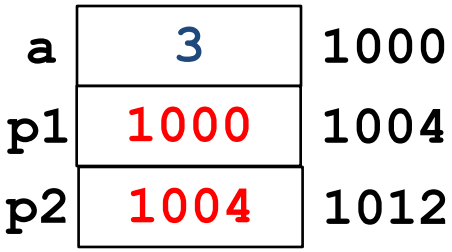
Modelo da Memória



Exemplo 3: ponteiros para ponteiros

```
int a = 3, *p1, **p2;

p1 = &a;  // endereço de a.
p2 = &p1; // endereço de p1.

// Mostra endereço da variável p1.
printf("Endereço de P1 = %d\n", p2 );

// Mostra endereço da variável a.
printf("Endereço de A = %d\n", *p2 ); // p2[0]

// Mostra valor da variável a.
printf("A = %d\n", **p2 ); // p2[0][0]
```

Modelo da Memória



Considerações

- Ponteiros para ponteiros devem apontar somente para ponteiros (e não para a informação);

Considerações

- Ponteiros para ponteiros devem apontar somente para ponteiros (e não para a informação);

- São usados para a alocação de *vetores de ponteiros*;

Considerações

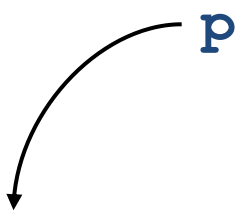
- Ponteiros para ponteiros devem apontar somente para ponteiros (e não para a informação);
- São usados para a alocação de *vetores de ponteiros*;
- Depois, cada ponteiro do vetor vai apontar para um vetor alocado dinamicamente (*linhas da matriz*, contendo os valores).

Exemplo 4: matriz dinâmica

```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
```

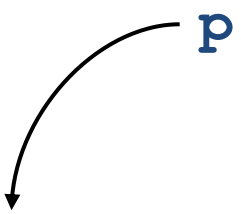
Exemplo 4: matriz dinâmica

```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
```



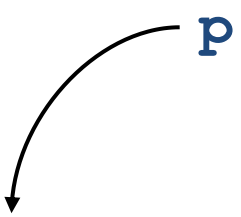
Exemplo 4: matriz dinâmica

```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
printf("Digite quantas linhas e colunas:\n");
scanf("%d%d", &lin, &col);
```



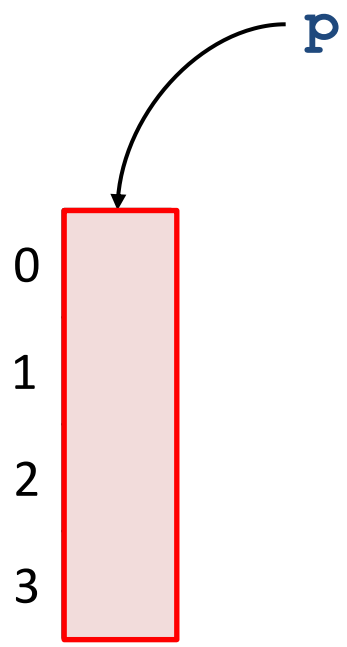
Exemplo 4: matriz dinâmica

```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
printf("Digite quantas linhas e colunas:\n");
scanf("%d%d", &lin, &col);
// Alocação do vetor de ponteiros
p = malloc( sizeof(int*) * lin );
```



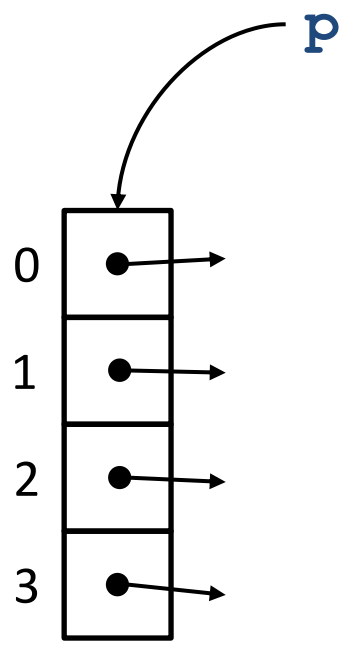
Exemplo 4: matriz dinâmica

```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
printf("Digite quantas linhas e colunas:\n");
scanf("%d%d", &lin, &col);
// Alocação do vetor de ponteiros
p = malloc( sizeof(int*) * lin );
```



Exemplo 4: matriz dinâmica

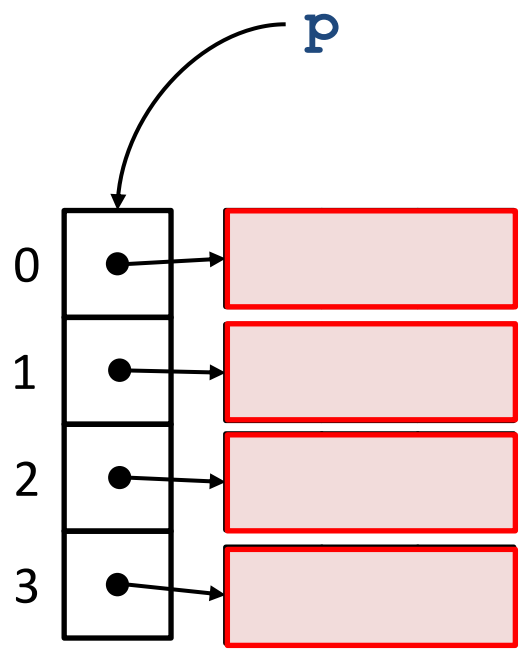
```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
printf("Digite quantas linhas e colunas:\n");
scanf("%d%d", &lin, &col);
// Alocação do vetor de ponteiros
p = malloc( sizeof(int*) * lin );

// Alocação das linhas
for( i = 0 ; i < lin ; i++ )
  p[i] = malloc( sizeof(int) * col );
```



Exemplo 4: matriz dinâmica

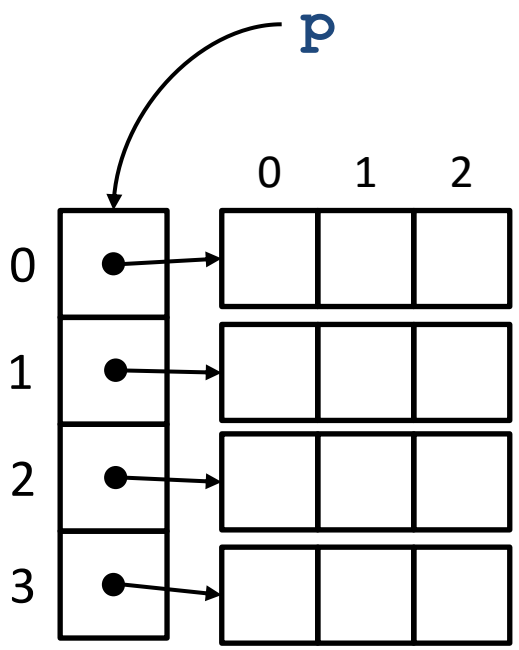
```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
printf("Digite quantas linhas e colunas:\n");
scanf("%d%d", &lin, &col);
// Alocação do vetor de ponteiros
p = malloc( sizeof(int*) * lin );
```

```
// Alocação das linhas
for( i = 0 ; i < lin ; i++ )
  p[i] = malloc( sizeof(int) * col );
```



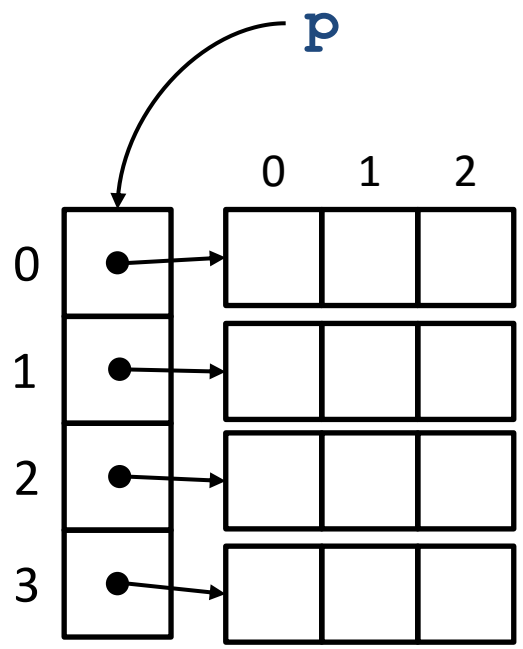
Exemplo 4: matriz dinâmica

```
int **p; // Ponteiro de ponteiro (será a matriz)
int lin, col, i, j;
printf("Digite quantas linhas e colunas:\n");
scanf("%d%d", &lin, &col);
// Alocação do vetor de ponteiros
p = malloc( sizeof(int*) * lin );

// Alocação das linhas
for( i = 0 ; i < lin ; i++ )
  p[i] = malloc( sizeof(int) * col );

// Continua...
```



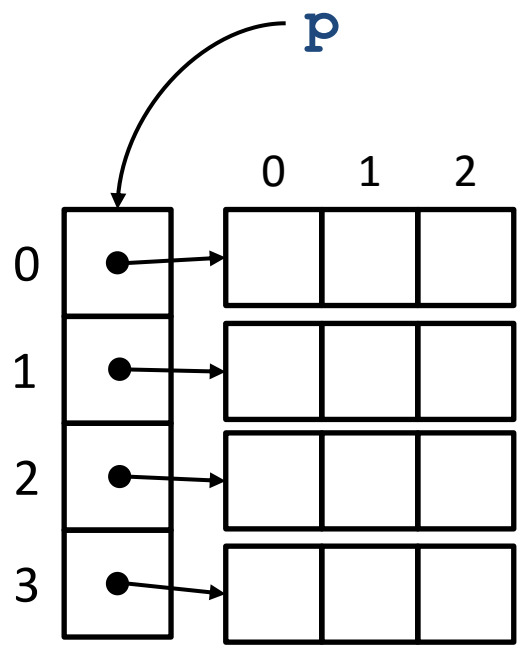
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
```



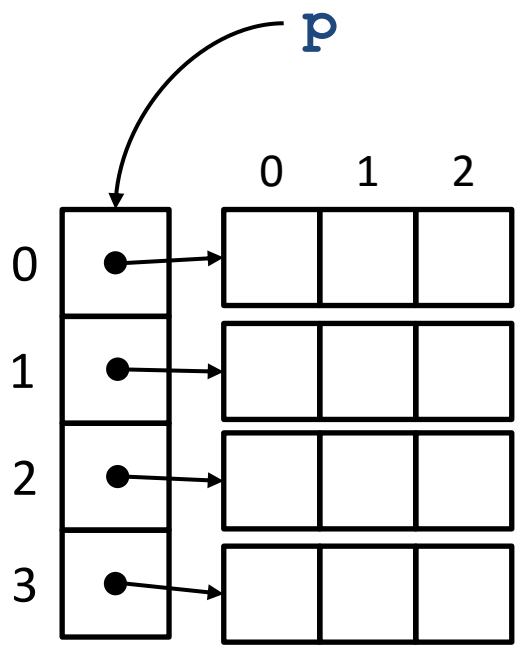
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
```



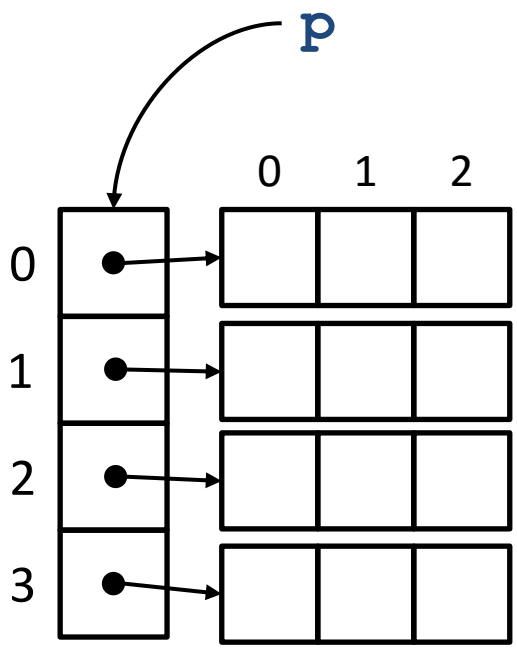
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
```



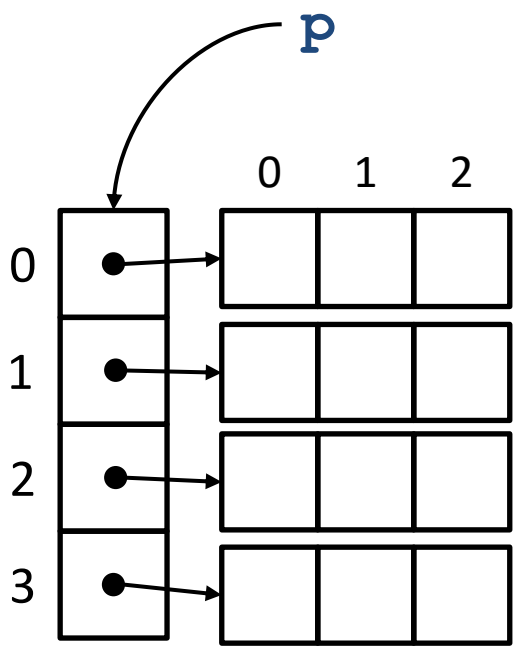
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
```



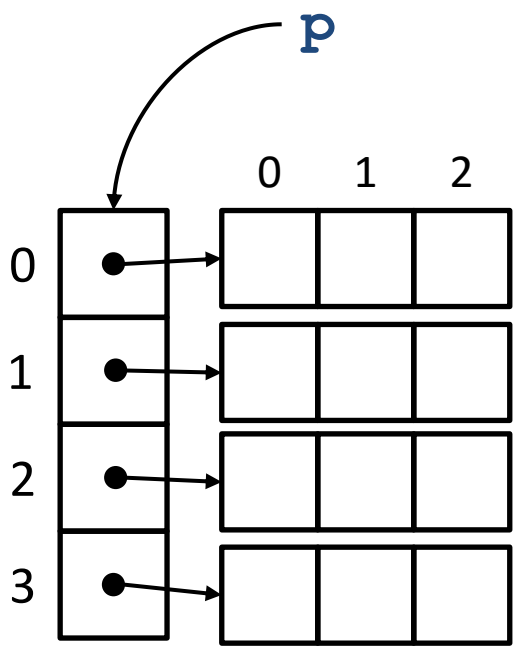
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
// Liberação da memória
for( i = 0 ; i < lin ; i++ )
  free( p[i] );
free( p );
```



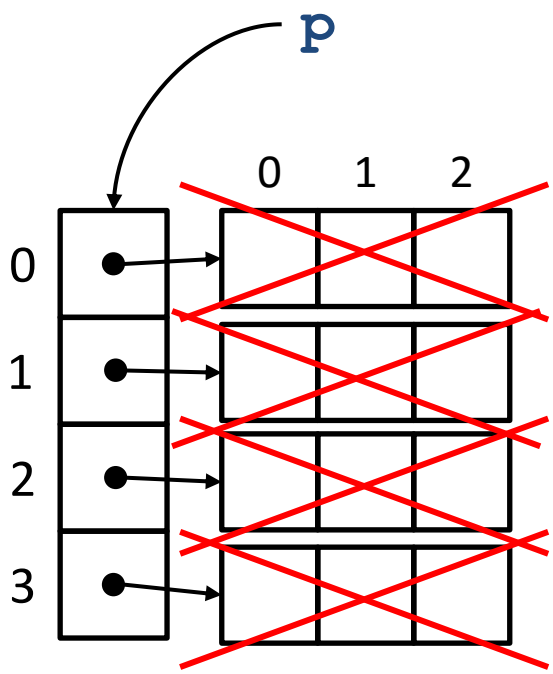
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
// Liberação da memória
for( i = 0 ; i < lin ; i++ )
  free( p[i] );
free( p );
```



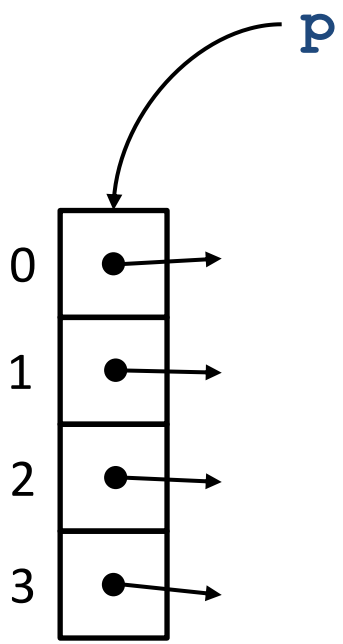
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
// Liberação da memória
for( i = 0 ; i < lin ; i++ )
  free( p[i] );
free( p );
```



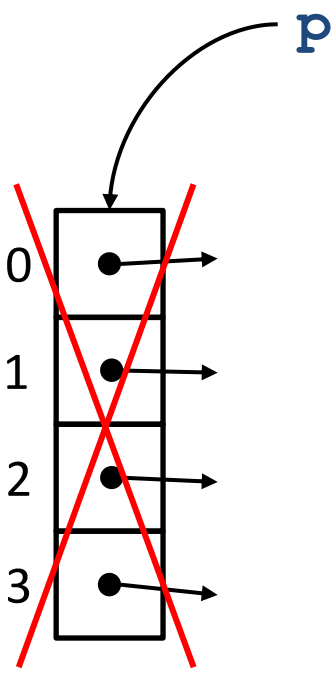
Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
// Liberação da memória
for( i = 0 ; i < lin ; i++ )
  free( p[i] );
free( p );
```



Exemplo 4: matriz dinâmica

```
// Entrada de dados
for( i = 0 ; i < lin ; i++ )
  for( j = 0 ; j < col ; j++ )
    scanf("%d", &p[i][j]);
// Saída dos dados
for( i = 0 ; i < lin ; i++ ){
  for( j = 0 ; j < col ; j++ )
    printf("%d ", p[i][j]);
  printf("\n");
}
// Liberação da memória
for( i = 0 ; i < lin ; i++ )
  free( p[i] );
free( p );
```



Considerações

- Os **operadores** sobre ponteiros e vetores continuam valendo;

Considerações

- Os **operadores** sobre ponteiros e vetores continuam valendo;
  - Neste caso, é preciso utilizar 2 vezes (2 dimensões);

Considerações

- Os **operadores** sobre ponteiros e vetores continuam valendo;
  - Neste caso, é preciso utilizar 2 vezes (2 dimensões);

- Tal como em ponteiros *simples*, a **realocação** também funciona com *ponteiros para ponteiros*;

Considerações

- Os **operadores** sobre ponteiros e vetores continuam valendo;
  - Neste caso, é preciso utilizar 2 vezes (2 dimensões);
- Tal como em ponteiros *simples*, a **realocação** também funciona com *ponteiros para ponteiros*;
- Para cada dimensão a mais na matriz, devemos aumentar o nível de endereçamento;

Considerações

- Os **operadores** sobre ponteiros e vetores continuam valendo;
  - Neste caso, é preciso utilizar 2 vezes (2 dimensões);
- Tal como em ponteiros *simples*, a **realocação** também funciona com *ponteiros para ponteiros*;
- Para cada dimensão a mais na matriz, devemos aumentar o nível de endereçamento;
  - Para 3 dimensões, teríamos `int ***p;`

Exemplo prático

- Alocação dinâmica de um **vetor de *strings***;
  - Strings são alocadas dinamicamente e adicionadas ao vetor de ponteiros;
  - À medida que o usuário digita novas palavras, o vetor de ponteiros vai sendo realocado;