Aula 23 – Matrizes (parte 2)

Norton T. Roman & Luciano A. Digiampietri

Matrizes

- A ideia de "arranjos de arranjos", em que todas as linhas têm igual tamanho, é conhecida como matriz

Matrizes

- A ideia de "arranjos de arranjos", em que todas as linhas têm igual tamanho, é conhecida como matriz
  - Útil para uma gama de problemas

Matrizes

- A ideia de "arranjos de arranjos", em que todas as linhas têm igual tamanho, é conhecida como matriz
  - Útil para uma gama de problemas
- Ex: Suponha que queiramos tabelar os preços das piscinas de 50, 100, 150 e 200 m^2, com os mais diversos materiais

Matrizes

- A ideia de "arranjos de arranjos", em que todas as linhas têm igual tamanho, é conhecida como matriz
  - Útil para uma gama de problemas
- Ex: Suponha que queiramos tabelar os preços das piscinas de 50, 100, 150 e 200 m^2, com os mais diversos materiais
- Queremos uma tabela:

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Matrizes

- Como criamos essa tabela?

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Matrizes

- Como criamos essa tabela?
  - Da mesma forma que a matriz de caracteres:

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

```
int main() {
  int i;
  double** valores = (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] = (double*) malloc(sizeof(double)*4);
  ...
```

Matrizes

- Como criamos essa tabela?
  - Da mesma forma que a matriz de caracteres:
- E como armazenamos valores nela?

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

```
int main() {
  int i;
  double** valores = (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] = (double*) malloc(sizeof(double)*4);
  ...
```

Matrizes

- Como criamos essa tabela?
  - Da mesma forma que a matriz de caracteres:
- E como armazenamos valores nela?
  - Com uma função auxiliar:

Material	*Áreas (m^2)*			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

```
/* Carrega os valores das piscinas na matriz
   de área X material
void carregaVal(double** m){
  int i,j;
  for (i=0; i<4; i++) {
    for (j=50; j<=200; j+=50) {
      m[i][j / 50 - 1] = precos[i]*j;
    }
  }
}
```

```
int main() {
  int i;
  double** valores = (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] = (double*) malloc(sizeof(double)*4);
  ...
```

Matrizes

- Note que as linhas usam o código natural dos materiais

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

```
/* Carrega os valores das piscinas na matriz
   de área X material
void carregaVal(double** m){
  int i,j;
   for (i=0; i<4; i++) {
      for (j=50; j<=200; j+=50) {
         m[i][j / 50 - 1] = precos[i] * j;
      }
   }
}
```

Matrizes

- Note que as linhas usam o código natural dos materiais
- Já colunas precisam de um cálculo

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

```
/* Carrega os valores das piscinas na matriz
   de área X material
void carregaVal(double** m){
  int i,j;
   for (i=0; i<4; i++) {
      for (j=50; j<=200; j+=50) {
         m[i][j / 50 - 1] = precos[i] * j;
      }
   }
}
```

Matrizes

- Note que as linhas usam o código natural dos materiais
- Já colunas precisam de um cálculo
- Podemos ainda usar as funções já desenvolvidas

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

```
/* Carrega os valores das piscinas na matriz
   de área X material
void carregaVal(double** m){
  int i,j;
   for (i=0; i<4; i++) {
      for (j=50; j<=200; j+=50) {
         m[i][j / 50 - 1] = valorPiscina(j,i);
      }
   }
}
```

Matrizes

- E como mostramos os valores da matriz?

```
int main() {
  int i, j;
  double** valores =
      (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
        (double*) malloc(sizeof(double)*4);
  carregaVal(valores);









  return 0;
}
```

Matrizes

- E como mostramos os valores da matriz?
- Percorrendo os índices da matriz

```
int main() {
  int i, j;
  double** valores =
      (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
        (double*) malloc(sizeof(double)*4);
  carregaVal(valores);

  for (int i=0; i<4; i++) {
    for (int j=0; j<4; j++) {
      printf("%9.2f\t",valores[i][j]);
    }
    printf("\n");
  }

  return 0;
}
```

Matrizes

```
int main() {
  int i, j;
  double** valores = (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i]=(double*) malloc(sizeof(double)*4);
  carregaVal(valores);
  for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
      printf("%9.2f\t",valores[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

```
 75000.00   150000.00   225000.00   300000.00
 55000.00   110000.00   165000.00   220000.00
 37500.00    75000.00   112500.00   150000.00
 25000.00    50000.00    75000.00   100000.00
```

Matrizes

- Vamos usar essa matriz agora para:

```
int main() {
  int i;
  double** valores = (double**)
                malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
       (double*) malloc(sizeof(double)*4);
  carregaVal(valores);









  return 0;
}
```

Matrizes

- Vamos usar essa matriz agora para:
  - Dizer o preço de uma piscina de plástico com 150m^2

```
int main() {
  int i;
  double** valores = (double**)
                malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
       (double*) malloc(sizeof(double)*4);
  carregaVal(valores);

  printf("Piscina de plastico
     de 150m2: %.2f\n",valores[PLASTICO][2]);

  return 0;
}
```

Matrizes

- Vamos usar essa matriz agora para:
  - Dizer o preço de uma piscina de plástico com 150m^2
  - Saída: `Piscina de plastico de 150m2:  75000.00`

```
int main() {
  int i;
  double** valores = (double**)
                malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
       (double*) malloc(sizeof(double)*4);
  carregaVal(valores);

  printf("Piscina de plastico
     de 150m2: %.2f\n",valores[PLASTICO][2]);









  return 0;
}
```

Matrizes

- Vamos usar essa matriz agora para:
  - Dizer o preço de uma piscina de plástico com 150m^2
  - Saída: `Piscina de plastico de 150m2:  75000.00`
  - Dizer o preço médio das piscinas de plástico

```
int main() {
  int i;
  double** valores = (double**)
                malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
       (double*) malloc(sizeof(double)*4);
  carregaVal(valores);

  printf("Piscina de plastico
     de 150m2: %.2f\n",valores[PLASTICO][2]);

  double media = 0;
  for (int i=0; i<4; i++)
                media += valores[PLASTICO][i];
  media /= 4;
  printf("Media: %.2f\n",media);

  return 0;
}
```

Matrizes

- Vamos usar essa matriz agora para:
  - Dizer o preço de uma piscina de plástico com 150m^2
  - Saída: `Piscina de plastico de 150m2:  75000.00`
  - Dizer o preço médio das piscinas de plástico
  - Saída: `Media: 62500.00`

```
int main() {
  int i;
  double** valores = (double**)
                malloc(sizeof(double*)*4);
  for (i=0;i<4;i++) valores[i] =
       (double*) malloc(sizeof(double)*4);
  carregaVal(valores);

  printf("Piscina de plastico
     de 150m2: %.2f\n",valores[PLASTICO][2]);

  double media = 0;
  for (int i=0; i<4; i++)
                media += valores[PLASTICO][i];
  media /= 4;
  printf("Media: %.2f\n",media);

  return 0;
}
```

Matrizes

- Podemos implementar uma função para o cálculo do preço médio

```
int main() {
  ...
  printf("Piscina de plastico de 150m2: %.2f\n",
                                        valores[PLASTICO][2]);
  printf("Media: %.2f\n",calculaMedia(valores[PLASTICO],4));
  return 0;
}
```

Matrizes

- Podemos implementar uma função para o cálculo do preço médio

```
double calculaMedia(double* arranjo, int tam){
  int i;
  double resp = 0;
  for (i=0;i<tam;i++) resp += arranjo[i];
  return resp/tam;
}


int main() {
  ...
  printf("Piscina de plastico de 150m2: %.2f\n",
                                        valores[PLASTICO][2]);
  printf("Media: %.2f\n",calculaMedia(valores[PLASTICO],4));
  return 0;
}
```

Matrizes

- Podemos implementar uma função para o cálculo do preço médio

```
double calculaMedia(double* arranjo, int tam){
  int i;
  double resp = 0;
  for (i=0;i<tam;i++) resp += arranjo[i];
  return resp/tam;
}


int main() {
  ...
  printf("Piscina de plastico de 150m2: %.2f\n",
                                        valores[PLASTICO][2]);
  printf("Media: %.2f\n",calculaMedia(valores[PLASTICO],4));
  return 0;
}
```

Matrizes

- Já temos a tabela com os valores

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Matrizes

- Já temos a tabela com os valores

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

- Suponha agora que existam descontos nos materiais:

Matrizes

- Já temos a tabela com os valores

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

- Suponha agora que existam descontos nos materiais:
  - Alvenaria está com 20% acima de 100m^2

Matrizes

- Já temos a tabela com os valores

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

- Suponha agora que existam descontos nos materiais:
  - Alvenaria está com 20% acima de 100m^2
  - Vinil tem 5% para piscinas de até 100m^2, 10% para as de 150m^2 e 15% para as acima disso

Matrizes

- Já temos a tabela com os valores

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

- Suponha agora que existam descontos nos materiais:
  - Alvenaria está com 20% acima de 100m^2
  - Vinil tem 5% para piscinas de até 100m^2, 10% para as de 150m^2 e 15% para as acima disso
  - Fibra tem descontos de 2%, 4%, 8% e 16%

Matrizes

- Já temos a tabela com os valores

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

- Suponha agora que existam descontos nos materiais:
  - Alvenaria está com 20% acima de 100m^2
  - Vinil tem 5% para piscinas de até 100m^2, 10% para as de 150m^2 e 15% para as acima disso
  - Fibra tem descontos de 2%, 4%, 8% e 16%
  - Plástico tem desconto de 5% apenas nas piscinas com 200m^2

Matrizes

- E como tabelamos os descontos?

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Matrizes

- E como tabelamos os descontos?

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Matrizes

- E como tabelamos os descontos?
- Gostaríamos agora de recalcular a tabela de valores

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Matrizes

- E como tabelamos os descontos?
- Gostaríamos agora de recalcular a tabela de valores
  - Uma tabela de preço final

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Matrizes

- E como tabelamos os descontos?
- Gostaríamos agora de recalcular a tabela de valores
  - Uma tabela de preço final
- Como?

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Matrizes

- Crie uma tabela equivalente, vazia

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Matrizes

- Crie uma tabela equivalente, vazia

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Final

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria				
Vinil				
Fibra				
Plástico				

Matrizes

- Crie uma tabela equivalente, vazia
- Nela, o valor final de cada célula será $\texttt{preco} - \texttt{preco} \times \texttt{desconto}$, ou $\texttt{preco} \times (1 - \texttt{desconto})$

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Final

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria				
Vinil				
Fibra				
Plástico				

Matrizes

- Crie uma tabela equivalente, vazia
- Nela, o valor final de cada célula será $\texttt{preco} - \texttt{preco} \times \texttt{desconto}$, ou $\texttt{preco} \times (1 - \texttt{desconto})$
- Ou seja:

```
pFinal[i][j] = preco[i][j] *
        (1 - desconto[i][j]);
```

Preços

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	75000	150000	225000	300000
Vinil	55000	110000	165000	220000
Fibra	37500	75000	112500	150000
Plástico	25000	50000	75000	100000

Descontos

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria	0	0	0.2	0.2
Vinil	0.05	0.05	0.1	0.15
Fibra	0.02	0.04	0.08	0.16
Plástico	0	0	0	0.05

Final

Material	*Áreas* (m^2)			
	50	100	150	200
Alvenaria				
Vinil				
Fibra				
Plástico				

Matrizes

- Criando as tabelas:

```
int main() {
  int i, j;
  double** valores =
              (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
   valores[i] = (double*) malloc(sizeof(double)*4);

  double descontos[][4] = {{0,0,0.2,0.2},
                           {0.05,0.05,0.1,0.15},
                          {0.02,0.04,0.08,0.16},
                                   {0,0,0,0.05}};

  double** pFinal;
  carregaVal(valores);

  pFinal = calculaFinal(valores, descontos);

  return 0;
}
```

Matrizes

- Criando as tabelas:
  - Note este outro modo de criar:

```
int main() {
  int i, j;
  double** valores =
              (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
   valores[i] = (double*) malloc(sizeof(double)*4);

  double descontos[][4] = {{0,0,0.2,0.2},
                           {0.05,0.05,0.1,0.15},
                          {0.02,0.04,0.08,0.16},
                                   {0,0,0,0.05}};

  double** pFinal;
  carregaVal(valores);

  pFinal = calculaFinal(valores, descontos);

  return 0;
}
```

Matrizes

- Criando as tabelas:
  - Note este outro modo de criar:
  - Aloca somente espaço para pFinal $\rightarrow$ um endereço

```
int main() {
  int i, j;
  double** valores =
              (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
   valores[i] = (double*) malloc(sizeof(double)*4);

  double descontos[][4] = {{0,0,0.2,0.2},
                           {0.05,0.05,0.1,0.15},
                          {0.02,0.04,0.08,0.16},
                                   {0,0,0,0.05}};

  double** pFinal;
  carregaVal(valores);

  pFinal = calculaFinal(valores, descontos);

  return 0;
}
```

Matrizes

- Criando as tabelas:
  - Note este outro modo de criar:
  - Aloca somente espaço para pFinal $\rightarrow$ um endereço

```
int main() {
  int i, j;
  double** valores =
              (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
   valores[i] = (double*) malloc(sizeof(double)*4);

  double descontos[][4] = {{0,0,0.2,0.2},
                           {0.05,0.05,0.1,0.15},
                          {0.02,0.04,0.08,0.16},
                                   {0,0,0,0.05}};

  double** pFinal;
  carregaVal(valores);

  pFinal = calculaFinal(valores, descontos);

  return 0;
}
```

- Carrega valores iniciais

Matrizes

- Criando as tabelas:
  - Note este outro modo de criar:
  - Aloca somente espaço para pFinal $\rightarrow$ um endereço

```
int main() {
  int i, j;
  double** valores =
              (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
   valores[i] = (double*) malloc(sizeof(double)*4);

  double descontos[][4] = {{0,0,0.2,0.2},
                           {0.05,0.05,0.1,0.15},
                          {0.02,0.04,0.08,0.16},
                                   {0,0,0,0.05}};

  double** pFinal;
  carregaVal(valores);

  pFinal = calculaFinal(valores, descontos);

  return 0;
}
```

- Carrega valores iniciais e calcula a nova tabela

Matrizes

- Construindo a tabela de valores finais

```
/*
    Retorna matriz com os preços finais.
    Parâmetros:
        val - matriz de valores
        desc - matriz de descontos
*/
double** calculaFinal(double** val,
                                double desc[][4]) {
  int i, j;

  double** saida =
           (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
    saida[i] = (double*) malloc(sizeof(double)*4);

  for (i=0;i<4;i++)
    for (j=0;j<4;j++)
      saida[i][j] = val[i][j]*(1-desc[i][j]);
  return saida;
}
```

Matrizes

- Construindo a tabela de valores finais
  - Recebe matrizes como parâmetros

```
/*
    Retorna matriz com os preços finais.
    Parâmetros:
        val - matriz de valores
        desc - matriz de descontos
*/
double** calculaFinal(double** val,
                                double desc[][4]) {
  int i, j;

  double** saida =
            (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
    saida[i] = (double*) malloc(sizeof(double)*4);

  for (i=0;i<4;i++)
    for (j=0;j<4;j++)
      saida[i][j] = val[i][j]*(1-desc[i][j]);
  return saida;
}
```

Matrizes

- Construindo a tabela de valores finais
  - Recebe matrizes como parâmetros
  - Cria a matriz resposta

```
/*
    Retorna matriz com os preços finais.
    Parâmetros:
        val - matriz de valores
        desc - matriz de descontos
*/
double** calculaFinal(double** val,
                                double desc[][4]) {
  int i, j;

  double** saida =
           (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
    saida[i] = (double*) malloc(sizeof(double)*4);

  for (i=0;i<4;i++)
    for (j=0;j<4;j++)
      saida[i][j] = val[i][j]*(1-desc[i][j]);
  return saida;
}
```

Matrizes

- Construindo a tabela de valores finais
  - Recebe matrizes como parâmetros
  - Cria a matriz resposta
  - Retorna o endereço da matriz

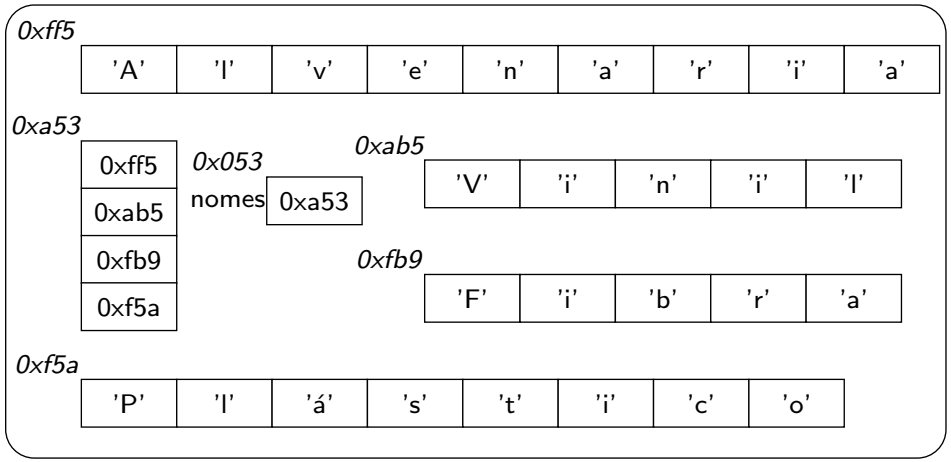
```
/*
    Retorna matriz com os preços finais.
    Parâmetros:
        val - matriz de valores
        desc - matriz de descontos
*/
double** calculaFinal(double** val,
                                double desc[][4]) {
  int i, j;

  double** saida =
            (double**) malloc(sizeof(double*)*4);
  for (i=0;i<4;i++)
    saida[i] = (double*) malloc(sizeof(double)*4);

  for (i=0;i<4;i++)
    for (j=0;j<4;j++)
      saida[i][j] = val[i][j]*(1-desc[i][j]);
  return saida;
}
```

Matrizes e Memória

- Como `nomes` está na memória?

Matrizes e Memória

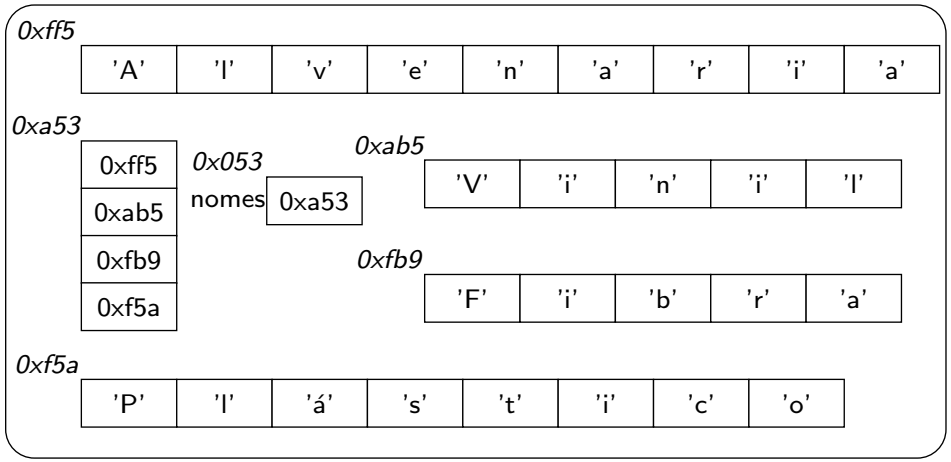
- Como `nomes` está na memória?



Matrizes e Memória

- Como `nomes` está na memória?
- Queremos fazer

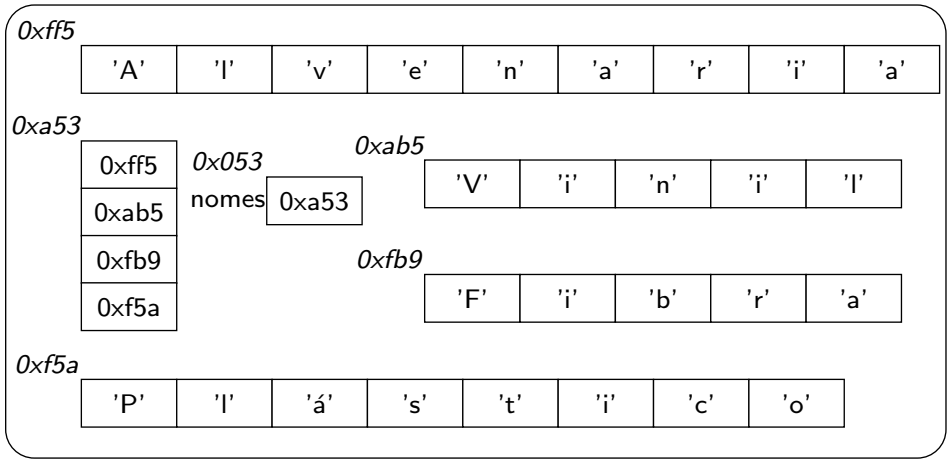
```
char** nomes2;
nomes2 = nomes;
```



Matrizes e Memória

- Como `nomes` está na memória?
- Queremos fazer

```
char** nomes2;
nomes2 = nomes;
```

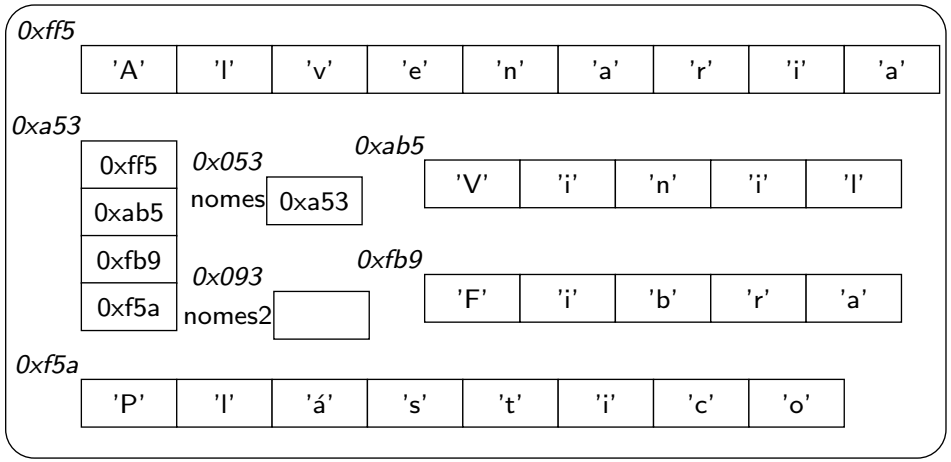


- O que acontece ao declararmos `nomes2`?

Matrizes e Memória

- Como `nomes` está na memória?
- Queremos fazer

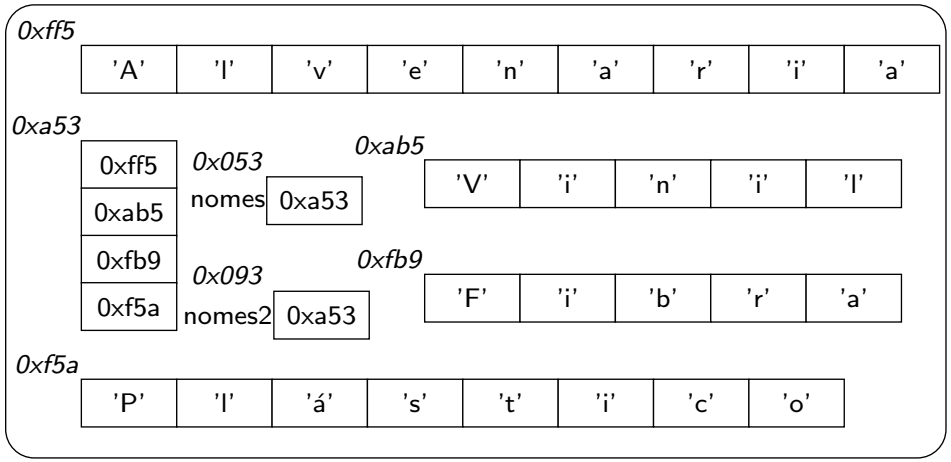
```
char** nomes2;
nomes2 = nomes;
```



- O que acontece ao declararmos `nomes2`?
  - Alocamos espaço suficiente para caber um endereço

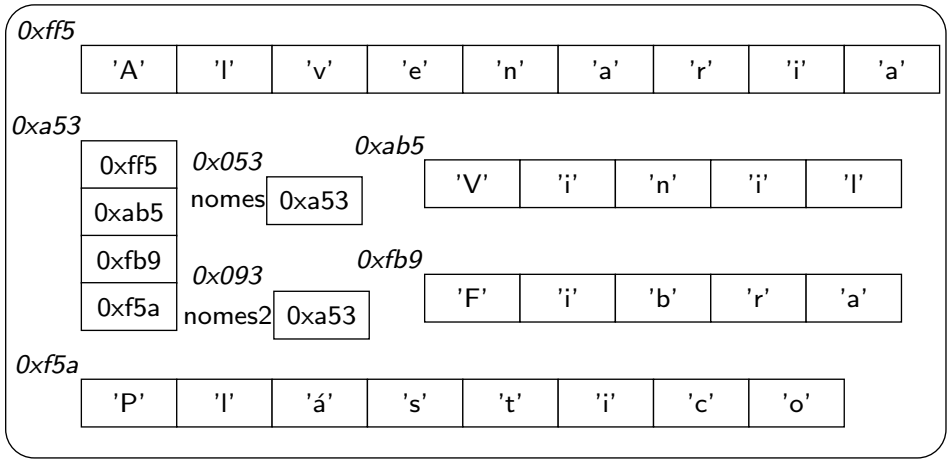
Matrizes e Memória

- Ao fazermos `nomes2 = nomes;` copiamos o conteúdo de `nomes` (ou seja, o endereço que lá está) para `nomes2`



Matrizes e Memória

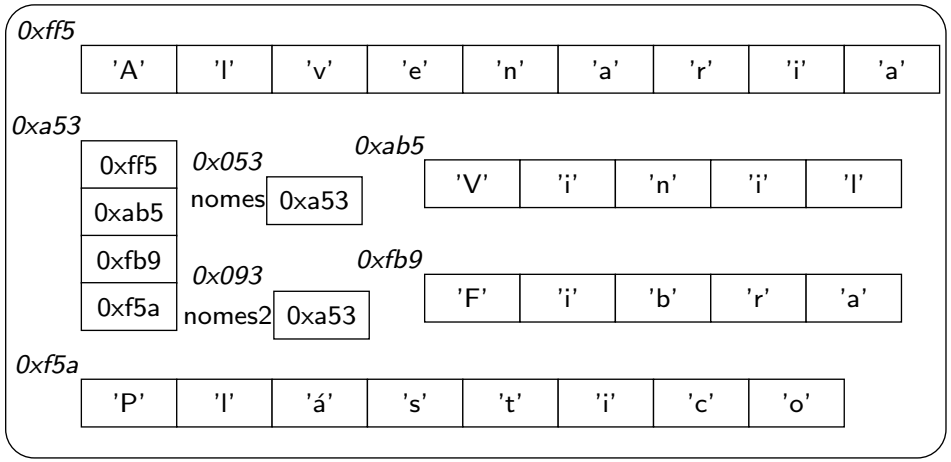
- Ao fazermos `nomes2 = nomes;` copiamos o conteúdo de `nomes` (ou seja, o endereço que lá está) para `nomes2`



- E tanto `nomes` quanto `nomes2` referenciam a mesma estrutura na memória

Matrizes e Memória

- Ao fazermos `nomes2 = nomes;` copiamos o conteúdo de `nomes` (ou seja, o endereço que lá está) para `nomes2`



- E tanto `nomes` quanto `nomes2` referenciam a mesma estrutura na memória

- Então, para copiarmos uma matriz em outra teremos que fazer elemento por elemento

Matrizes e Memória

- Considere os códigos:

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++)w=matriz[i][j];

  return 0;
}
```

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```

Matrizes e Memória

- Considere os códigos:

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++)w=matriz[i][j];

  return 0;
}
```

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```

- Média de tempo em 10 repetições:

Matrizes e Memória

- Considere os códigos:

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++)w=matriz[i][j];

  return 0;
}
```

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```

- Média de tempo em 10 repetições:

  0,60s

Matrizes e Memória

- Considere os códigos:

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++)w=matriz[i][j];

  return 0;
}
```

```c
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```

- Média de tempo em 10 repetições:

0,60s 2,14s

Matrizes e Memória

- Por que isso acontece?

Matrizes e Memória

- Por que isso acontece?
  - Por conta da cache

Matrizes e Memória

- Por que isso acontece?
  - Por conta da cache

Cache

Circuito de memória que, juntamente com a RAM (e outros) formam a memória primária do computador

Matrizes e Memória

- Por que isso acontece?
  - Por conta da cache

Cache

Circuito de memória que, juntamente com a RAM (e outros) formam a memória primária do computador

- Características:

Matrizes e Memória

- Por que isso acontece?
  - Por conta da cache

Cache

Circuito de memória que, juntamente com a RAM (e outros) formam a memória primária do computador

- Características:
  - Rapidez (por volta de 2ns) – 5 vezes mais rápido que a RAM

Matrizes e Memória

- Por que isso acontece?
  - Por conta da cache

Cache

Circuito de memória que, juntamente com a RAM (e outros) formam a memória primária do computador

- Características:
  - Rapidez (por volta de 2ns) – 5 vezes mais rápido que a RAM
  - Caro

Matrizes e Memória

Funcionamento:

Matrizes e Memória

Funcionamento:

- Quando um endereço de memória é buscado, o computador verifica primeiro se o conteúdo está na cache

Matrizes e Memória

Funcionamento:

- Quando um endereço de memória é buscado, o computador verifica primeiro se o conteúdo está na cache
- Se não estiver, ele é buscado na RAM, sendo então trazido, juntamente com alguns vizinhos, para a cache

Matrizes e Memória

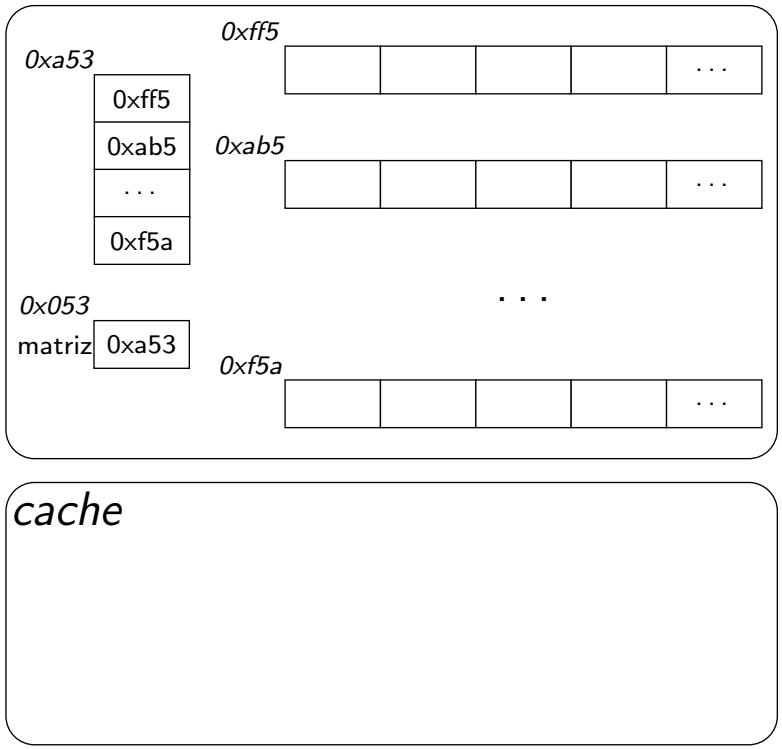
Funcionamento:

- Quando um endereço de memória é buscado, o computador verifica primeiro se o conteúdo está na cache
- Se não estiver, ele é buscado na RAM, sendo então trazido, juntamente com alguns vizinhos, para a cache
- Assim, o próximo acesso ao mesmo endereço (ou algum vizinho) será mais rápido, pois ele estará na cache

Matrizes e Memória

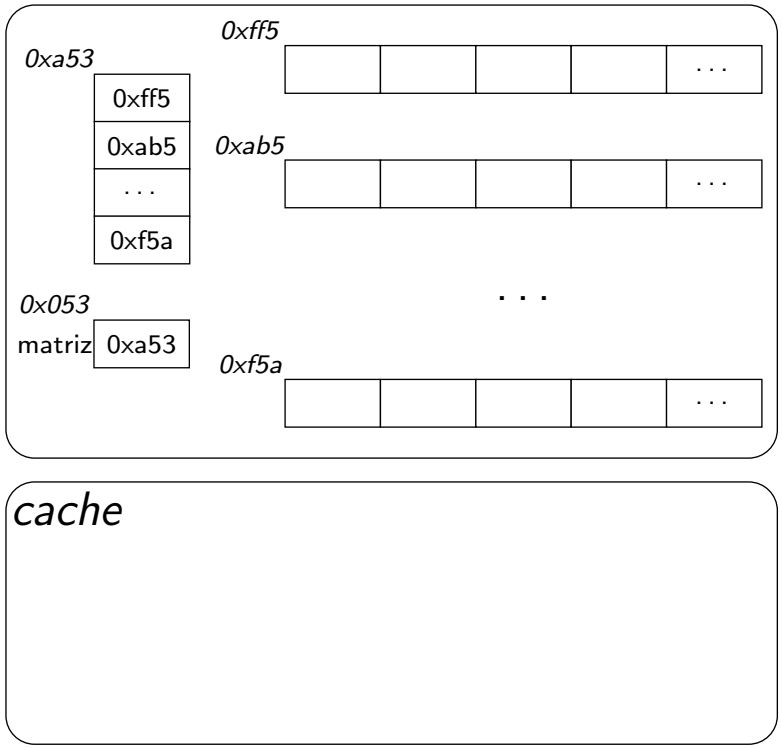
- Agora vejamos como cada arranjo é corrido na memória (supondo que o arranjo em 0xa53 está na cache):

```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



Matrizes e Memória

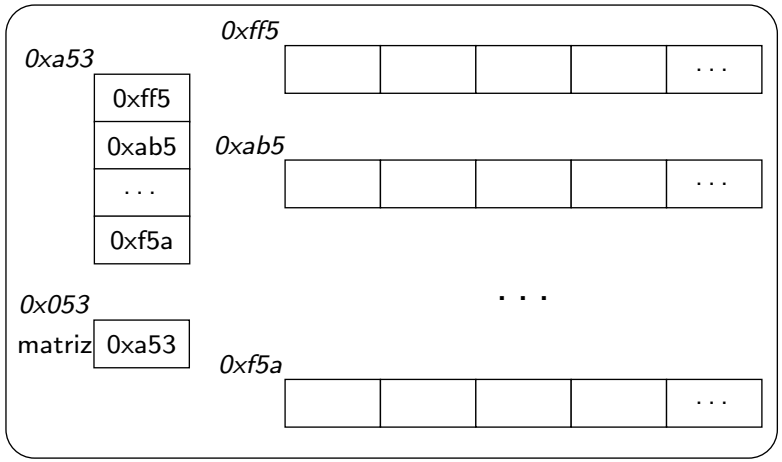
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



- O primeiro elemento (`[0,0]`) desse arranjo é buscado na cache

Matrizes e Memória

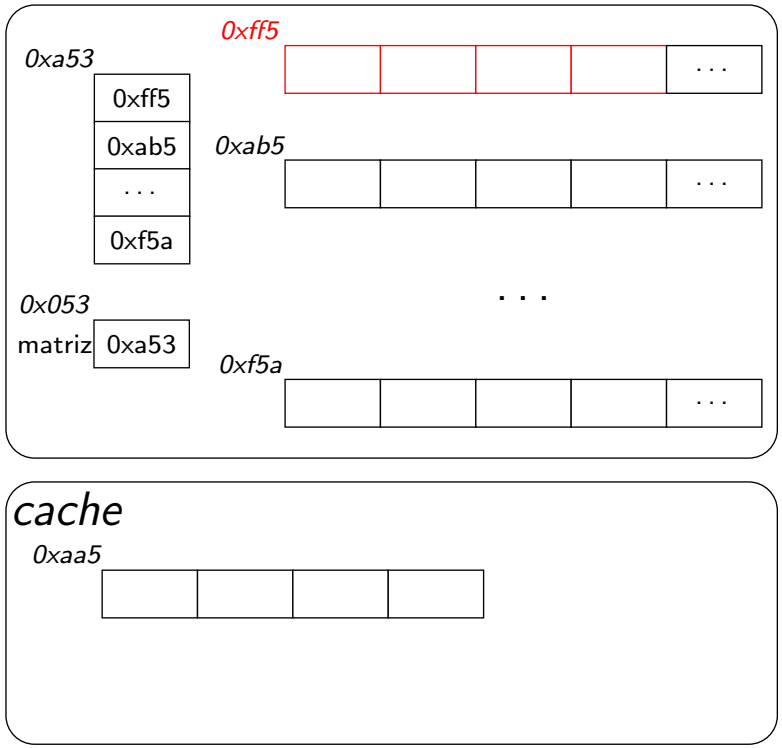
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



- O primeiro elemento (`[0,0]`) desse arranjo é buscado na cache – não é encontrado

Matrizes e Memória

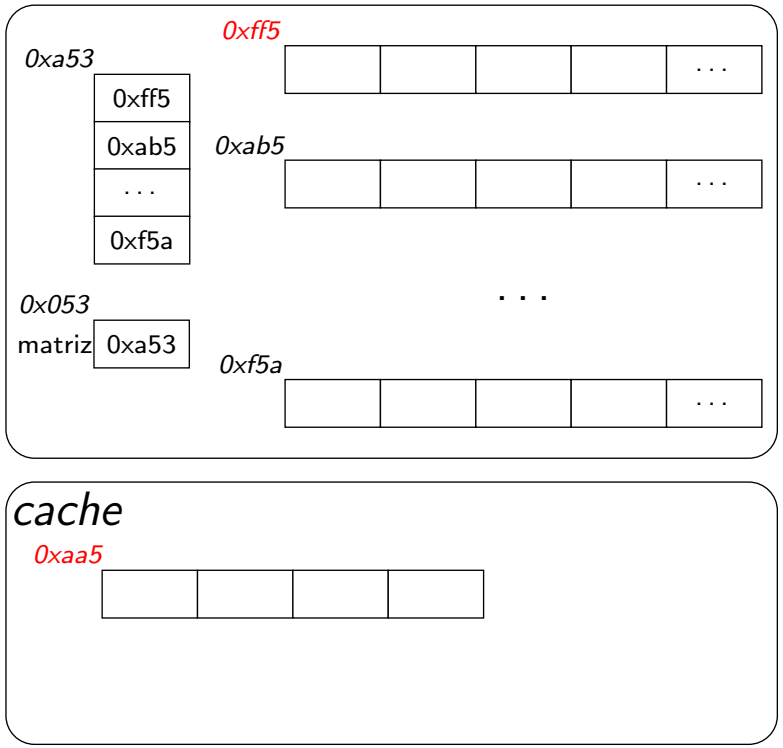
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



- O primeiro elemento (`[0,0]`) desse arranjo é buscado na cache – não é encontrado
- Então é buscado na memória, sendo alguns vizinhos trazidos para a cache

Matrizes e Memória

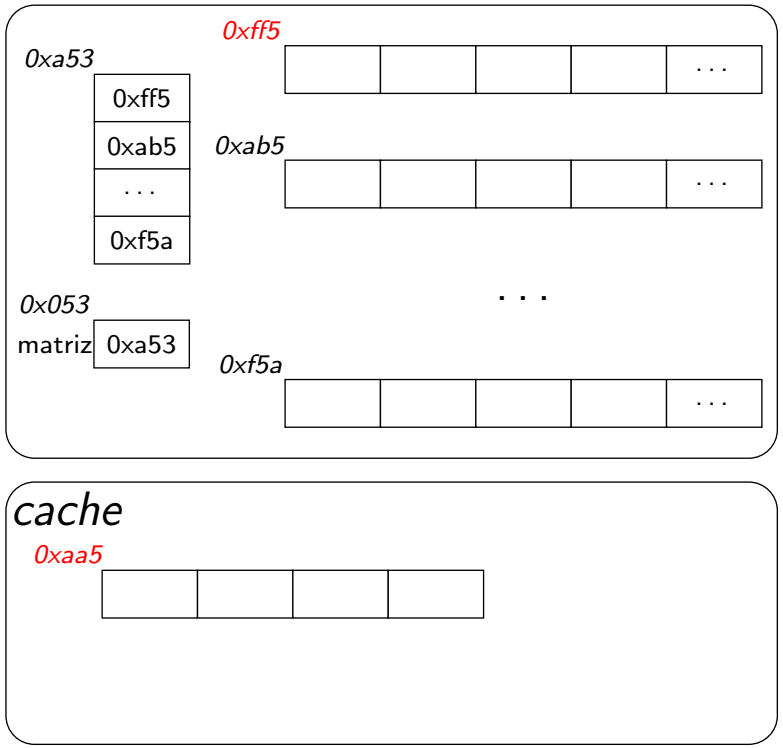
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



- O uso da cache é transparente ao programador

Matrizes e Memória

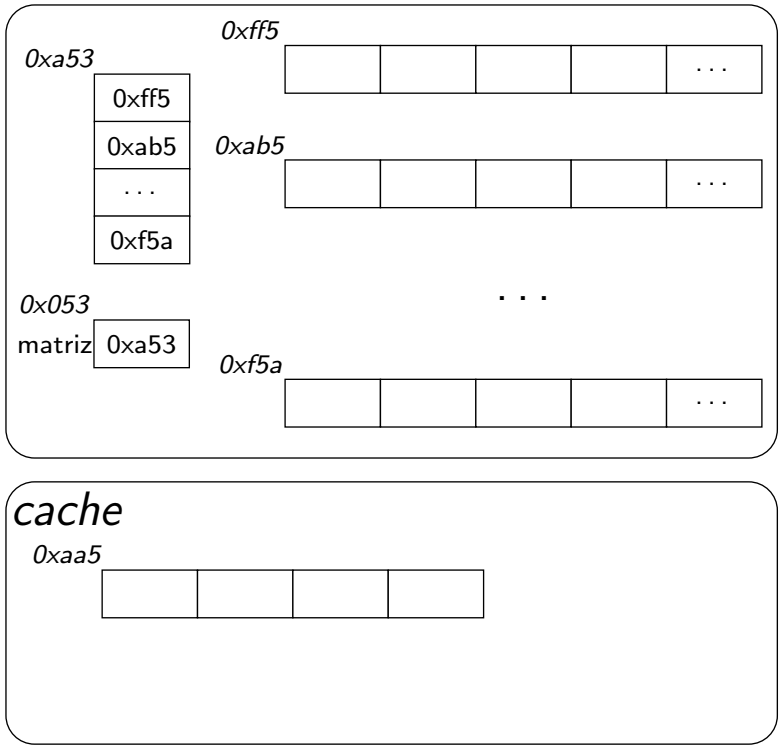
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



- O uso da cache é transparente ao programador
  - O S.O. gerencia tudo isso, com a ajuda do hardware

Matrizes e Memória

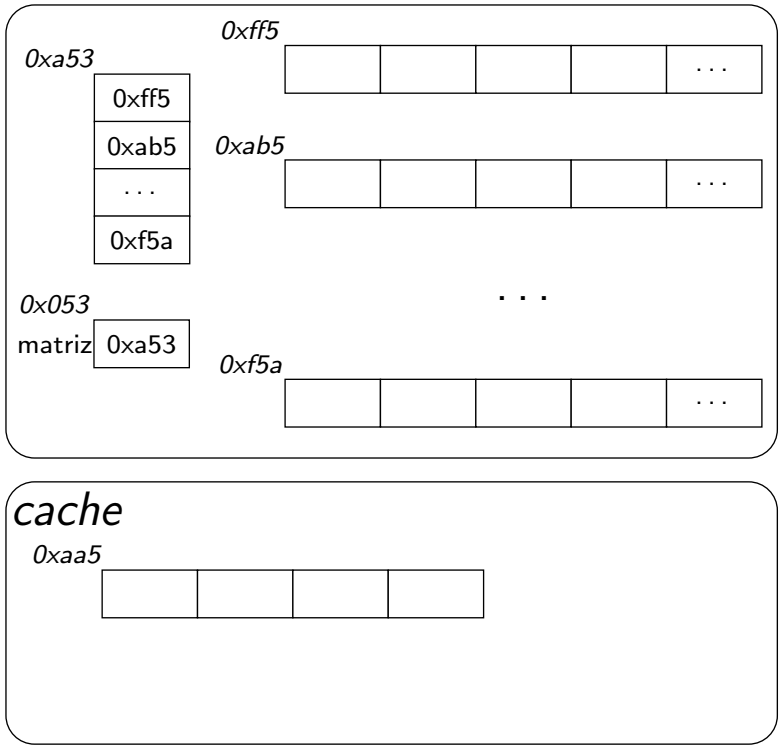
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```



- A próxima iteração do `for` buscará o segundo elemento desse primeiro arranjo

Matrizes e Memória

```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[i][j];

  return 0;
}
```

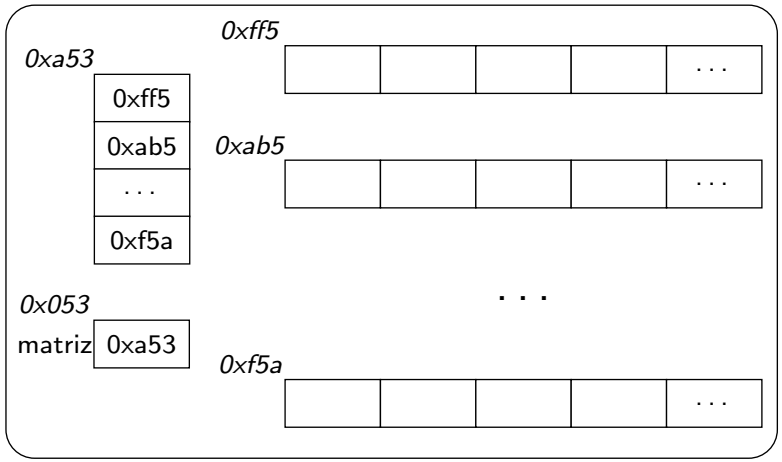


- A próxima iteração do `for` buscará o segundo elemento desse primeiro arranjo
  - Ao buscar esse elemento, ele já está na cache – é recuperado mais rapidamente

Matrizes e Memória

- E no segundo modelo?

```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```

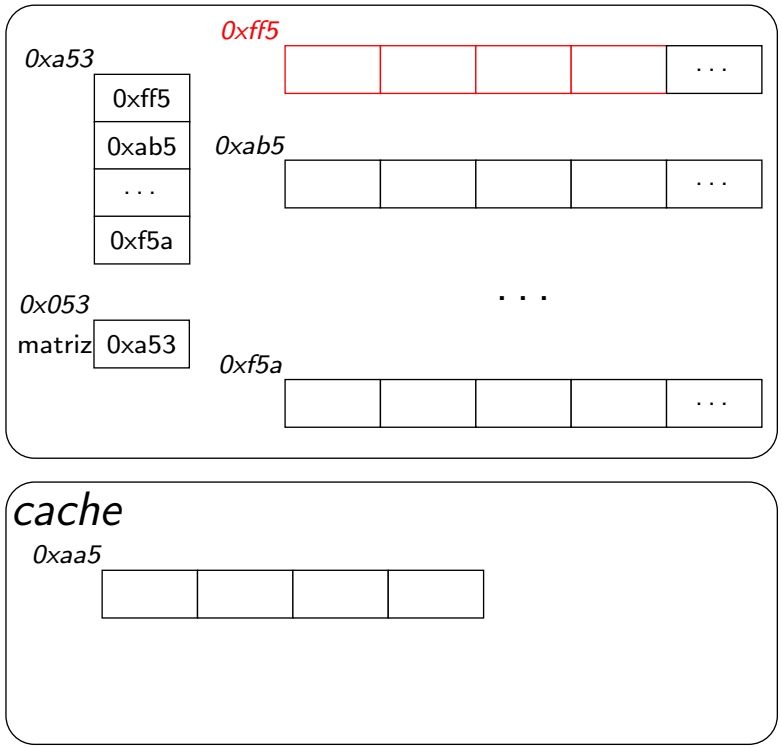


cache

Matrizes e Memória

- E no segundo modelo?

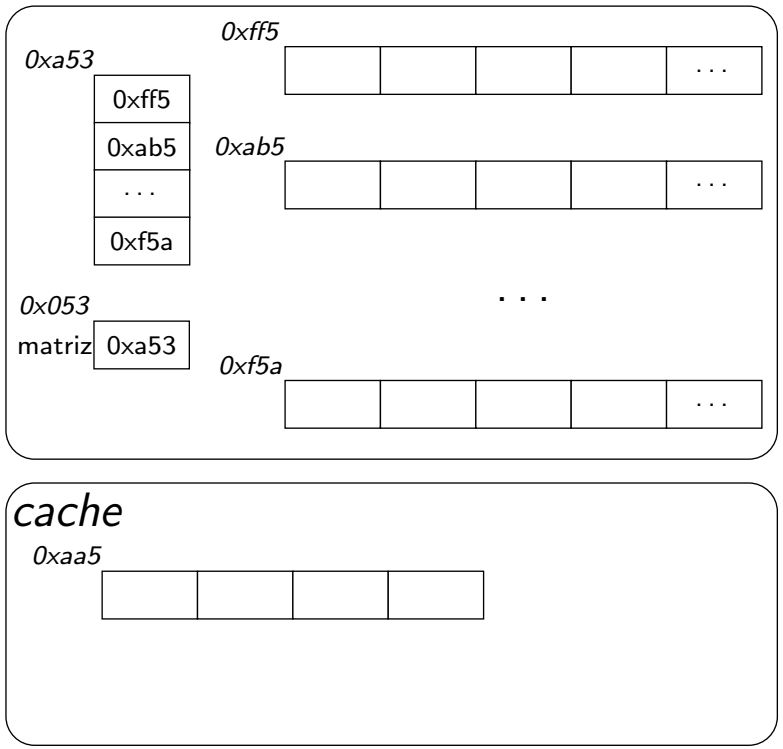
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```



- Primeiro elemento do primeiro arranjo: idêntico ao esquema anterior

Matrizes e Memória

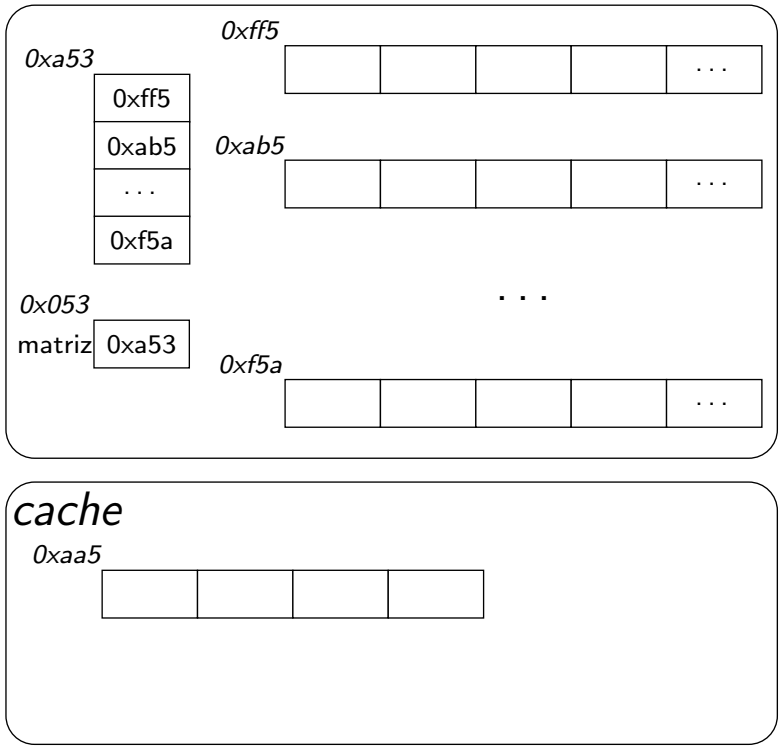
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```



- Próxima iteração do `for`: primeiro elemento do segundo arranjo

Matrizes e Memória

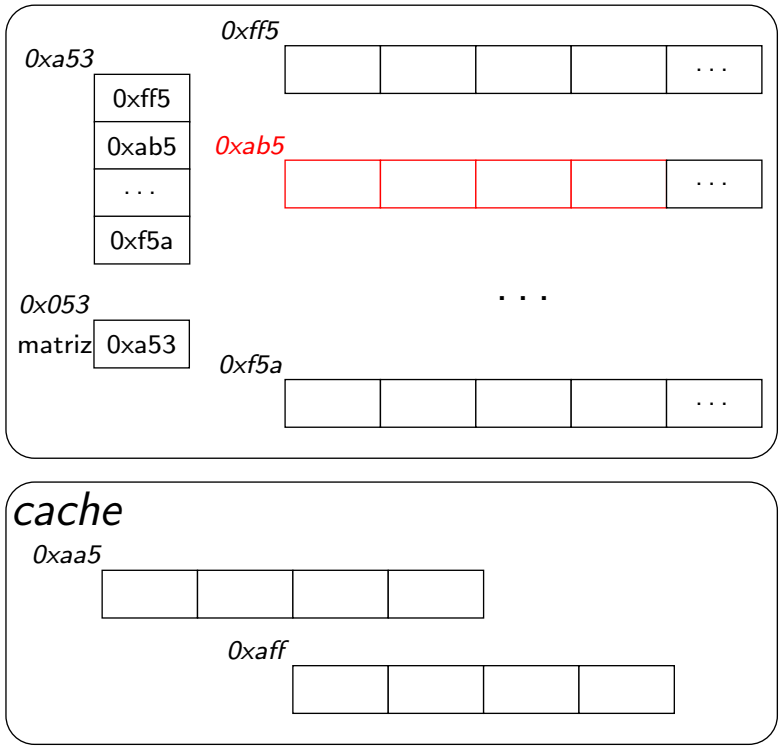
```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```



- Ao buscar esse elemento, ele não estará na cache

Matrizes e Memória

```
#include <stdlib.h>
#define ORDEM 10000
int main() {
  int i, j, w;
  double** matriz = (double**)
         malloc(sizeof(double*)*ORDEM);
  for (i=0;i<ORDEM;i++)
    matriz[i] = (double*)
          malloc(sizeof(double)*ORDEM);

  for (i=0;i<ORDEM;i++)
    for (j=0;j<ORDEM;j++) w=matriz[j][i];

  return 0;
}
```



- Ao buscar esse elemento, ele não estará na cache
- Deve ser trazido da RAM → mais lento

Matrizes Multidimensionais

- Finalmente, matrizes não existem apenas em duas dimensões

Matrizes Multidimensionais

- Finalmente, matrizes não existem apenas em duas dimensões
- Podem ter mais:

Matrizes Multidimensionais

- Finalmente, matrizes não existem apenas em duas dimensões
- Podem ter mais:
- Ex: `double**** matriz1; //quatro dimensões`

Matrizes Multidimensionais

- Finalmente, matrizes não existem apenas em duas dimensões
- Podem ter mais:
- Ex: `double**** matriz1; //quatro dimensões`
  - Este comando apenas cria a variável matriz1

Matrizes Multidimensionais

- Finalmente, matrizes não existem apenas em duas dimensões
- Podem ter mais:
- Ex: `double**** matriz1; //quatro dimensões`
  - Este comando apenas cria a variável matriz1
- Ex: `double matriz2[3][2][4][5];`

Matrizes Multidimensionais

- Finalmente, matrizes não existem apenas em duas dimensões
- Podem ter mais:
- Ex: `double**** matriz1; //quatro dimensões`
  - Este comando apenas cria a variável matriz1
- Ex: `double matriz2[3][2][4][5];`
- Acessadas com `matriz[2][1][0][1]`

Aula 23 – Matrizes (parte 2)

Norton T. Roman & Luciano A. Digiampietri