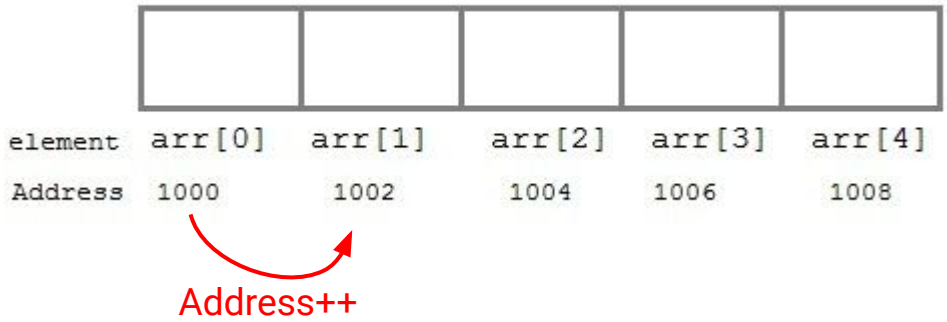
Estruturas de Dados Homogêneos: Vetores e Matrizes - parte 2

Prof. Gabriel Barbosa da Fonseca
Email: gbfonseca@sga.pucminas.br

Aritmética de ponteiros

- Quando fazemos uma operação de soma em um ponteiro (que guarda um endereço de memória), temos como resultado o **próximo endereço após o término do elemento atual.**



Aritmética de Ponteiros

```c
int main(){
    char vetor[5];
    char *p;
    p = vetor;
    for(int i = 0;i<=5;i++){
      p++;
      printf("Adress = %p\n",p);
    }
    return 0;
}
```

```
./main

Adress = 0x7ffd7d2b0dc4

Adress = 0x7ffd7d2b0dc5

Adress = 0x7ffd7d2b0dc6

Adress = 0x7ffd7d2b0dc7

Adress = 0x7ffd7d2b0dc8

Adress = 0x7ffd7d2b0dc9
```

Aritmética de Ponteiros

```c
int main(){
    float vetor[5];
    float *p;
    p = vetor;
    for(int i = 0;i<=5;i++){
      p++;
      printf("Adress = %p\n",p);
    }
    return 0;
}
```

```
./main

Adress = 0x7ffdb7e94604

Adress = 0x7ffdb7e94608

Adress = 0x7ffdb7e9460c

Adress = 0x7ffdb7e94610

Adress = 0x7ffdb7e94614

Adress = 0x7ffdb7e94618
```

Aritmética de ponteiros

- Vetor1 (o nome da variável) funciona como um ponteiro que aponta para o primeiro elemento do vetor.
- Vetor1 = &Vetor1[0]

vetor1;
vetor1+1;
vetor1+2;

Memória RAM		
Endereço	Variável	Valor
7ffe5367e000	vetor1[0]	15
7ffe5367e001		
7ffe5367e002	vetor1[1]	14
7ffe5367e003		
7ffe5367e004	vetor1[2]	13
7ffe5367e005		

Exercício

Como acessar todos os elementos de um vetor, **sem utilizar os colchetes e índices**???

```
int main(){
    float vetor[5] = {3,2,1,0,1};
    float atual;
    for(int i = 0;i<5;i++){
      printf("vetor[%d] = %f\n",i,vetor[i]);
    }
    return 0;
}
```

```
./main
vetor[0] = 3.000000
vetor[1] = 2.000000
vetor[2] = 1.000000
vetor[3] = 0.000000
vetor[4] = 1.000000
```

Exercício

Como acessar todos os elementos de um vetor, **sem utilizar os colchetes e índices**???

```
int main() {
  float vetor[5] = {3, 2, 1, 0, 1};
  float *vetorPtr;
  vetorPtr = vetor;
  for (int i = 0; i < 5; i++) {
    printf("vetor[%d] = %f\n", i, *vetorPtr);
    vetorPtr++;
  }
  return 0;
}
```

```
./main
vetor[0] = 3.000000
vetor[1] = 2.000000
vetor[2] = 1.000000
vetor[3] = 0.000000
vetor[4] = 1.000000
```

Passagem de parâmetros - Vetores

- int Vetor1[40];
- Vetor1 (o nome da variável) funciona como um ponteiro que aponta para o primeiro elemento do vetor.
- Vetor1 = &Vetor1[0]
- Passamos o vetor por parâmetro, com uma **passagem por referência:**

Opção 1:

```
void printVetor(int vetor[])
```

Opção 2:

```
void printVetor(int *vetor)
```

Chamada da função:

```
int vec[90];
printVetor(vec);
```

Alocação dinâmica

- Podemos declarar um vetor de tamanho pré-definido:
  - int vetor1[50]
- Podemos também declarar um vetor usando alocação dinâmica, com tamanho variável, decidido em tempo de execução:

```
int n;
printf(“Digite o tamanho do vetor”);
scanf(“%d”,&n);
int *pv = (int*) malloc(n * sizeof(int));
    for(int i=0; i<n; i++) {
        pv[i] = i+1;
    }
```

Alocação dinâmica

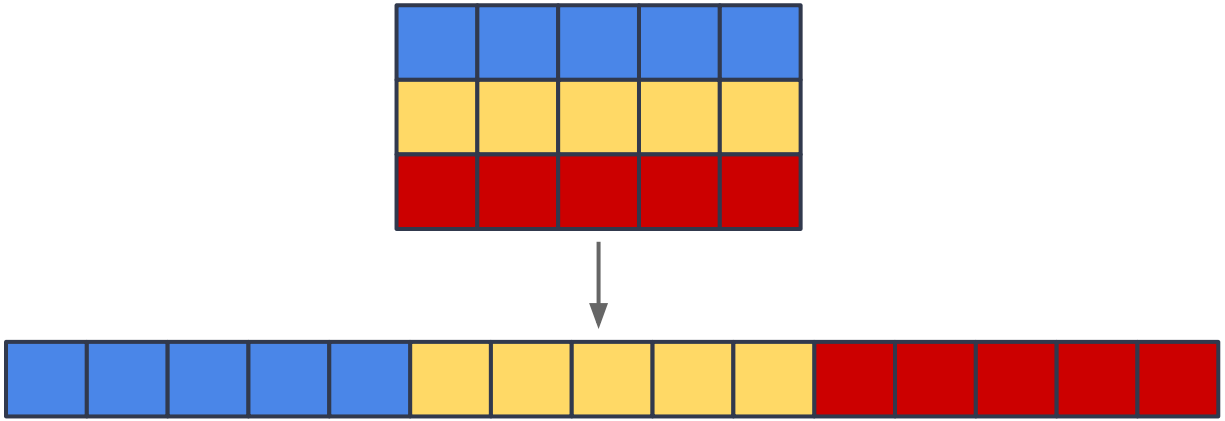
- Podemos declarar uma matriz de tamanho pré-definido:
  - int mat1[5][3]
- Podemos também declarar uma matriz usando alocação dinâmica, com tamanho variável, decidido em tempo de execução:

```
int n,m;
printf(“Digite o tamanho da matriz:”);
scanf(“%d %d”,&n,&m);
int *matriz = (int *)malloc(maxLn * maxCol * sizeof(int));
for (int i = 0; i < maxLn; i++) {
        for (int j = 0; j < maxCol; j++) {
            *(matriz + (i*maxCol) + j) = i*j;
        }
    }
```

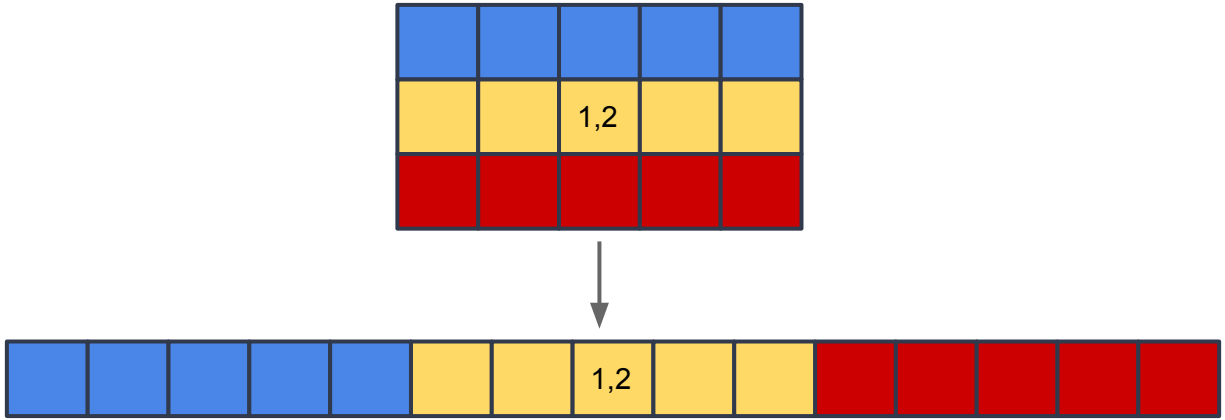
Explicação

Quando declaramos a matriz com:
`int *matriz = (int *)malloc(maxLn * maxCol * sizeof(int))`,

Estamos na verdade declarando um vetor, que representa uma matriz de forma “esticada”, concatenando suas linhas.



Explicação



Para acessar a posição 1,2 da matriz original, multiplicamos o índice da linha que queremos acessar (1) pelo total de colunas (5) e somamos o índice da coluna (2):

*(matriz + (i*maxCol) + j)

Opção 2 - Alocação dinâmica de matriz

```
//Vetor de ponteiros que aponta para as
//linhas da matriz
int linhas=3,colunas=5;
int *mat[linhas];
  for (i = 0; i < linhas; i++){
    mat[i] = (int *)malloc(colunas * sizeof(int));
  }
//Acesso usando indices normalmente
mat[i][j] = 5;
```