Arrays

«Tipos de dados compostos»

Taxonomia de tipos de dados

Simples

- Numéricos
  - int (inteiros)
  - float (reais)
  - char (carateres)
- Apontadores
  - *
- Enumerados
  - enum

Compostos

- Definidos pelo utilizador:
  - **arrays** (cadeias e tabelas)
  - struct (registos/estruturas)
  - FILE (ficheiros)

Caraterísticas dos tipos de dados compostos

- Definidos pelo utilizador
  - Não pertencem ao léxico da linguagem
  - Requerem a utilização de
    - um mecanismo sintático,
    - uma palavra reservada (**[]**, struct, FILE, …)
- Composição
  - É um conjunto de dados
  - Exemplo (cadeia de inteiros):

    int X[5] = { -5, 4, -10, 0, 65 };
- Não têm ordem
  - Um conjunto não tem ordem ou escala
  - Exemplo (cadeias de inteiros):

    { 5, 7, 4 } < { 7, 2, 8 } (não faz sentido)

Array

Definição

- Um array é uma "variável" composta por um número fixo de elementos (que são variáveis) com as seguintes características:
  - com estrutura tipada
    - todas as variáveis são do mesmo tipo
  - contiguidade
    - as variáveis ocupam uma zona contígua da memória
  - acesso por indexação
    - cada variável é acedida por um ou vários índices, consoante a sua dimensão

Caraterísticas específicas

- Dimensão
  - 1 índice: array de 1 dimensão
  - 2 índices: array de 2 dimensões
  - 3 índices: array de 3 dimensões
  - ...
- Representação em diagrama

0	1	2
7	3	-5

array de 1 dimensão

	0	1	2
0	9	-2	0
1	-7	5	7
2	6	-7	2
3	-4	3	-6

array de 2 dimensões

Array de 1 dimensão

Definição

- Um array de 1 dimensão é uma "variável" com um número fixo de elementos, onde estes elementos são todos de um mesmo tipo,
- Um array de 1 dimensão pode ser visto como uma linha (ou coluna) com uma quantidade de valores do mesmo tipo

Analogia

- Um array de 1 dimensão é comparado a um vetor
  - por isso, é designado habitualmente por **vetor**

Declaração

- Sintaxe

```
tipo  nome[tamanho];
```

onde

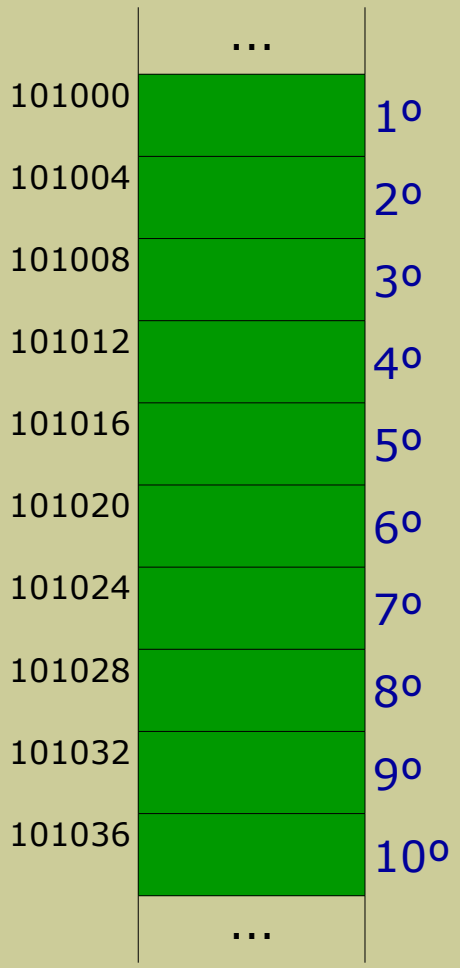
- **tipo** é o tipo de dados de todas as variáveis que formam o array
  - pode ser inteiro (int), real (float) e caráter (char)
- **nome** é a identificação das variáveis
- **tamanho** é o número de elementos (variáveis) que formam o array

Declaração

- Exemplo 1
  - declaração de um array de 1 dimensão com 10 elementos do tipo inteiro

```
...
int  V[10];
...
```

cada variável do tipo inteiro ocupa 4 bytes da memória

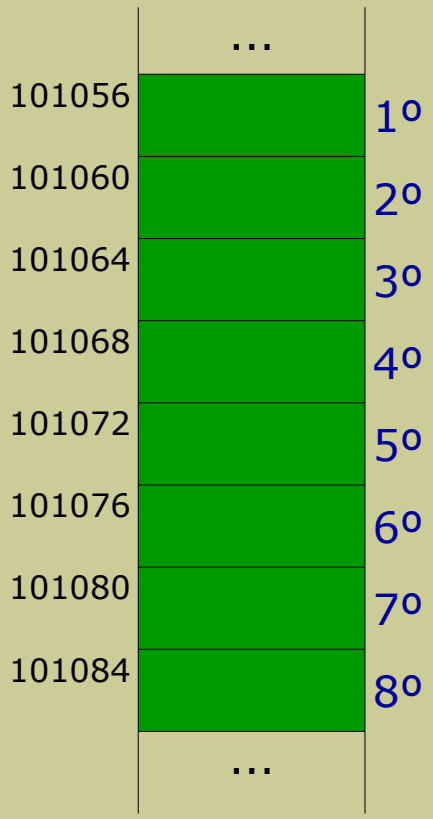


Declaração

- Exemplo 2
  - declaração de um array de 1 dimensão com 8 elementos do tipo real

```
...
float  X[8];
...
```

cada variável do tipo real ocupa 4 bytes da memória

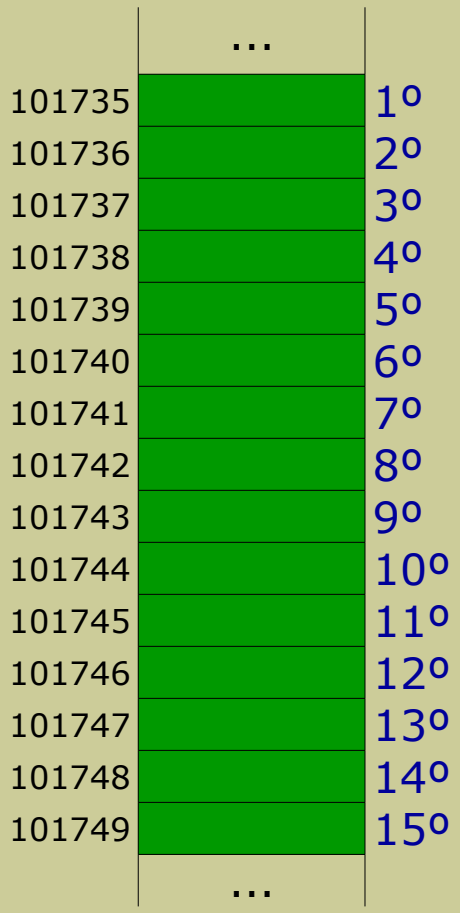


Declaração

- Exemplo 3
  - declaração de um array de 1 dimensão com 8 elementos (variáveis) do tipo caráter

```
...
char  ch[15];
...
```

cada variável do tipo caráter ocupa 1 byte da memória



Indexação

- Considere-se um array de 1 dimensão com 10 elementos do tipo inteiro

```
...
int  V[10];
...
```

  - são 10 variáveis do tipo inteiro, todas com um mesmo nome

- Então, como usar cada uma destas variáveis individualmente?

Indexação

- Considere-se um array de 1 dimensão com 10 elementos do tipo inteiro

```
int  V[10];
```

  - são 10 variáveis do tipo inteiro, todas com um mesmo nome
- Então, como usar cada uma destas variáveis individualmente?
- Resposta
  - através de índices, que indicam a posição de cada um dos elementos
- Portanto, o array de 1 dimensão com 10 elementos, tem os índices de 0 a 9 e os elementos são referenciados por

```
V[0], V[1], V[2], V[3], V[4], V[5], V[6], V[7], V[8], V[9]
```

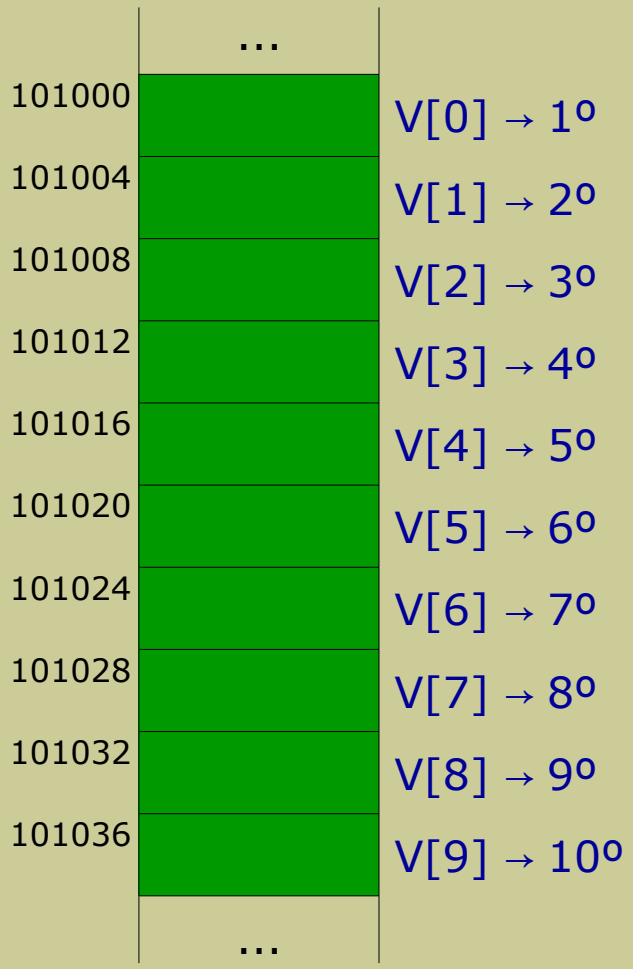
  - nome do array e o respetivo índice entre parêntesis retos
- **ATENÇÃO**
  - não confundir com a declaração (tamanho vs. índice)
  - note-se que os índices começam sempre em 0 (e não em um, como noutras linguagens)

Indexação

- Considere-se um array de 1 dimensão com 10 elementos do tipo inteiro

```
...
int  V[10];
...
```

cada variável do tipo inteiro ocupa 4 bytes da memória



Acesso aos elementos do array

- Como cada elemento é uma variável (simples), pode-se aceder a cada uma delas da mesma forma que uma variável simples (atribuir, ler e escrever), apenas temos que indicar o índice do elemento
- A não ser em situações especiais, o acesso é sempre aos elementos de um array individualmente (ou seja, utilizando os índices)

Acesso aos elementos do array

- Exemplo 1

```
...
int  V[10];
...
V[0] = 4;
V[2] = 3;
V[9] = -7;
...
```

- declaração de um array **V** de 1 dimensão com 10 elementos do tipo inteiro
- atribuir o número inteiro 4 ao elemento de índice 0 do array **V** (o primeiro)
- atribuir o número inteiro 3 ao elemento de índice 2 do array **V** (o terceiro)
- atribuir o número inteiro -7 ao elemento de índice 9 do array **V** (o último)

Acesso aos elementos do array

- Exemplo 2

```
...
int  k;
float  X[8];
...
X[4] = 2.1;
k = 2;
printf("Insira um valor real: ");
scanf("%f", &X[k]);
printf("Quinto elemento = %f\n", X[k+2]);
...
```

- declaração de um array **X** de 1 dimensão com 8 elementos do tipo real
- atribuir o número real 2.1 ao elemento de índice 4 do array X (o quinto)
- ler do teclado um número real e atribuí-lo ao elemento de índice k do array X (k = 2)
- escrever no monitor o valor do elemento de índice k+2 do array X (k+2 = 4)

Acesso aos elementos do array

- Observações
  - para a declaração

```
...
int  V[10];
...
```

é **errado**

```
...
V = 2;
scanf("%d", &V);
printf("%d\n", V);
...
```

pois faltam os índices

Acesso aos elementos do array

- Observações
  - para a declaração

```
...
int  V[10];
...
V[10] = 2;
...
```

está **errado**

pois o array V só tem 10 elementos e V[10] não existe (seria o 11º elemento)

A utilização de repetições

- De uma maneira geral, o uso de arrays de 1 dimensão envolve a aplicação de instruções de repetição (ciclos)
- Exemplo:

  preencher um array de inteiros com 500 elementos (tamanho = 500)

A utilização de repetições

- Exemplo: preencher um array de inteiros com 500 elementos
- Estratégia 1 (sem ciclos):

```
#include <stdio.h>
void main()
{
  int  V[500];
  printf("Insira um numero inteiro: ");
  scanf("%d", &V[0]);
  printf("Insira um numero inteiro: ");
  scanf("%d", &V[1]);
  ...    // nesta zona devem estar todas as instruções que faltam, que são 497
  printf("Insira um numero inteiro: ");
  scanf("%d", &V[499]);
}
```

  - o que não seria "razoável", pois o programa teria pelo menos 500 instruções (500 chamadas da função predefinida **scanf**)

A utilização de repetições

- Note-se que a inserção de um elemento tem associado 2 instruções:

```
printf("Insira um numero inteiro: ");
```

  - que é sempre a mesma (sem qualquer alteração) para todos os elementos e, portanto, pode ser incluída diretamente num ciclo

```
scanf("%d", &V[0]);
scanf("%d", &V[1]);
...   // nesta zona devem estar todas as instruções que faltam, que são 497
scanf("%d", &V[499]);
```

  - que apenas muda o índice do elemento que se pretende ler (V[0], V[1], ..., V[499]) e, portanto, não pode ser aplicada diretamente (necessita de um ajuste)

A utilização de repetições

- As instruções de leitura dos elementos do array podem ser as seguintes:

```
k = 0;
scanf("%d", &V[k]);
k = k + 1;      //  k = 1
scanf("%d", &V[k]);
...   // nesta zona devem estar todas as instruções que faltam, que são 2x497
k = k + 1;      //  k = 499
scanf("%d", &V[k]);
```

  - a instrução de leitura (scanf) é sempre a mesma e, portanto, pode ser incluida diretamente num ciclo
  - a instrução que envolve o k
    - é diferente para o primeiro elemento (k = 0) e
    - mantém-se inalterada para os restantes (k = k + 1)
  - a evolução dos valores de k retrata na perfeição a evolução dos valores da variável de controlo de um ciclo

A utilização de repetições

- O bloco que envolve a leitura dos elementos do array podem ter a seguinte disposição, o que não altera o resultado final:

```
k = 0;
scanf("%d", &V[k]);
k = k + 1;      // k = 1
scanf("%d", &V[k]);
k = k + 1;      // k = 2
...   // nesta zona devem estar todas as instruções que faltam, que são 2x496
scanf("%d", &V[k]);
k = k + 1;      // k = 499
scanf("%d", &V[k]);
k = k + 1;      // k = 500
```

  - tem-se agora 2 instruções que se mantêm inalteradas para todos os elementos

A utilização de repetições

- Resolvendo doutra forma, aplicando uma instrução de repetição
- Estratégia 2 (com ciclos):

```
#include <stdio.h>
void main()
{
  int  k, V[500];
  k = 0;
  while(k <= 499)  // ou while(k < 500)
  {
     printf("Insira um numero inteiro: ");
     scanf("%d", &V[k]);
     k = k + 1;
  }
}
```

A utilização de repetições

- Resolvendo doutra forma, aplicando uma instrução de repetição
  - por se conhecer o número de iterações do ciclo (igual ao tamanho do array), a instrução de repetição mais "adequada" é o **for**
- Estratégia 2 (com ciclos):

```
#include <stdio.h>
void main()
{
  int  k, V[500];
  for (k = 0; k <= 499; k = k + 1)
  {
     printf("Insira um numero inteiro: ");
     scanf("%d", &V[k]);
  }
}
```

Passagem de um array como argumento de um subprograma

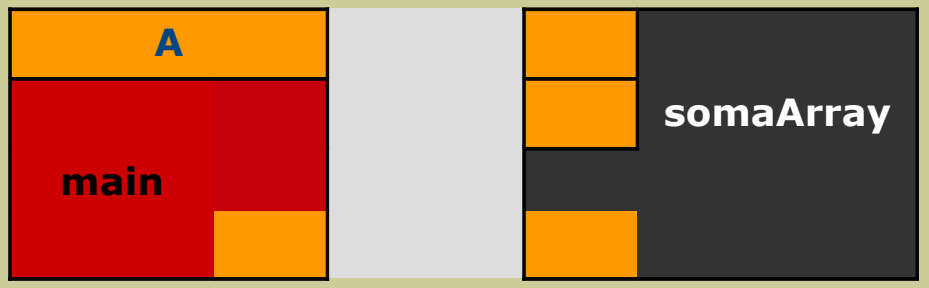
- A passagem de um array para um subprograma é feita através do seu nome
- Quando um array é passado como argumento a um subprograma, o seu tamanho é ignorado pelo compilador
- Ao compilador só interessa saber o tipo de dados dos elementos do array
- O tamanho a considerar é da inteira responsabilidade do programador, pelo que também deve ser passado como argumento, mas de forma independente

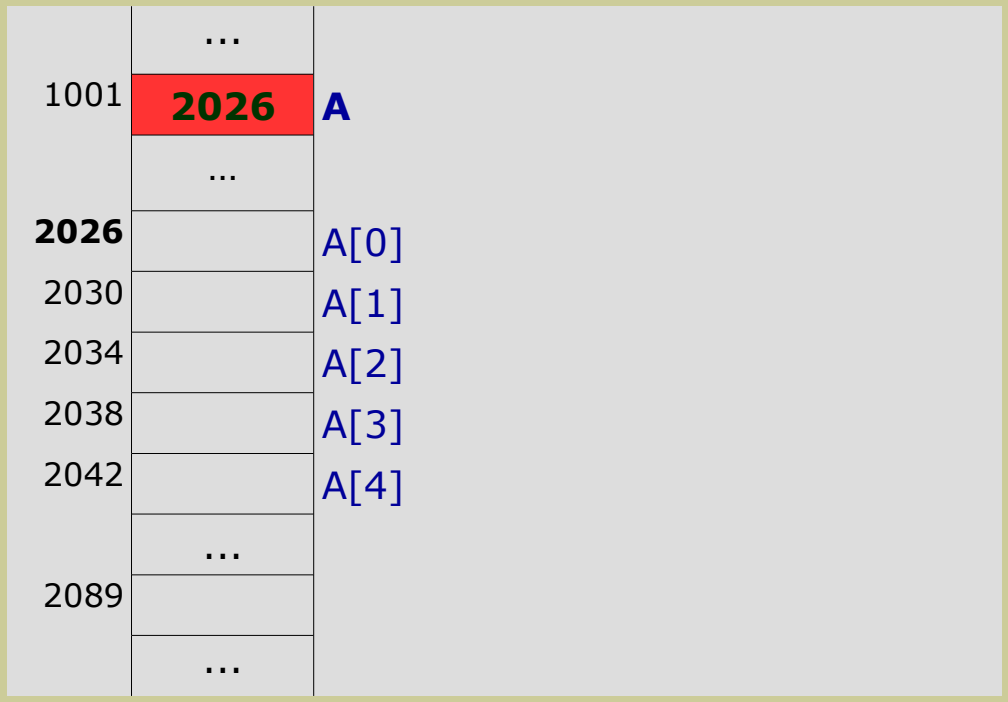
- **ATENÇÃO**
  - O nome do array funciona como endereço do primeiro elemento do array, ou seja, a sua localização em memória
- **CONSEQUÊNCIA**
  - Quando se passa um array para um subprograma, o que se está a passar é o endereço do primeiro elemento, e não o array no seu todo

Passagem de um array como argumento de um subprograma

- Exemplo (criação das variáveis e inserção dos dados no programa)

```
#include <stdio.h>
int somaArray (int V[], int N){
   int  k, soma;
   soma = 0;
   for (k = 0; k <= N-1; k = k + 1)
       soma = soma + V[k];
   return soma;
}
void main(){
   int  A[5] = { 5, 2, 9, 4, 6 };
   int  s;
   s = somaArray(A, 5);
   printf("Soma = %d\n", s);
}
```

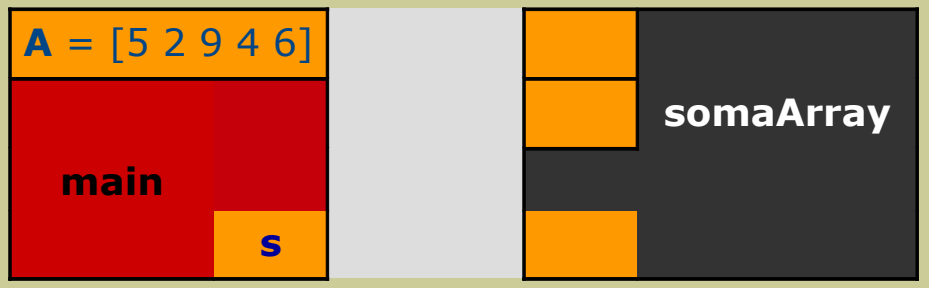




Passagem de um array como argumento de um subprograma

- Exemplo (criação das variáveis e inserção dos dados no programa)

```
#include <stdio.h>
int somaArray (int V[], int N){
   int  k, soma;
   soma = 0;
   for (k = 0; k <= N-1; k = k + 1)
       soma = soma + V[k];
   return soma;
}
void main(){
   int  A[5] = { 5, 2, 9, 4, 6 };
   int  s;
   s = somaArray(A, 5);
   printf("Soma = %d\n", s);
}
```

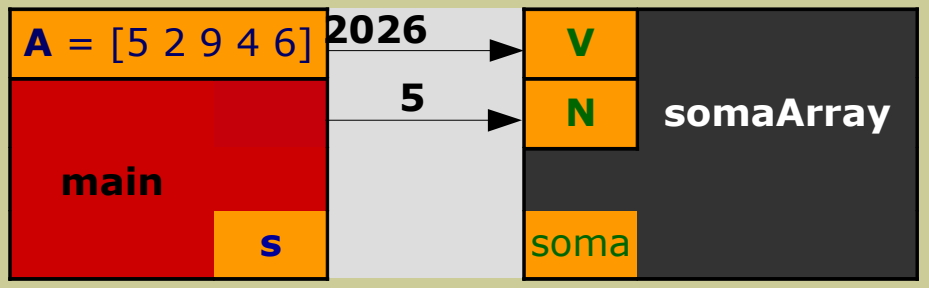


	...	
1001	2026	A
	...	
2026	5	A[0]
2030	2	A[1]
2034	9	A[2]
2038	4	A[3]
2042	6	A[4]
	...	
2089		s
	...	

Passagem de um array como argumento de um subprograma

- Exemplo (chamada do subprograma – passagem dos argumentos)

```
#include <stdio.h>
int somaArray (int V[], int N){
   int  k, soma;
   soma = 0;
   for (k = 0; k <= N-1; k = k + 1)
       soma = soma + V[k];
   return soma;
}
void main(){
   int  A[5] = { 5, 2, 9, 4, 6 };
   int  s;
   s = somaArray(A, 5);
   printf("Soma = %d\n", s);
}
```

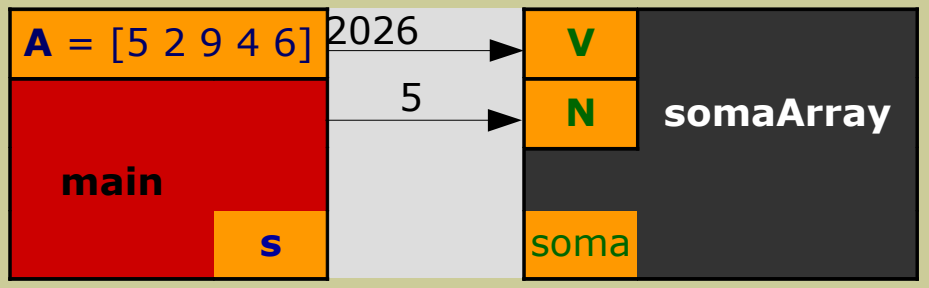


	...			...	
1001	2026	A	5065	2026	V
	...			...	
2026	5	A[0] V[0]	5089	5	N
2030	2	A[1] V[1]			
2034	9	A[2] V[2]		...	
2038	4	A[3] V[3]	5126		k
2042	6	A[4] V[4]			
	...			...	
2089		s	5249		soma
	...			...	

Passagem de um array como argumento de um subprograma

- Exemplo (execução do algoritmo do subprograma)

```
#include <stdio.h>
int somaArray (int V[], int N){
   int  k, soma;
   soma = 0;
   for (k = 0; k <= N-1; k = k + 1)
       soma = soma + V[k];
   return soma;
}
void main(){
   int  A[5] = { 5, 2, 9, 4, 6 };
   int  s;
   s = somaArray(A, 5);
   printf("Soma = %d\n", s);
}
```

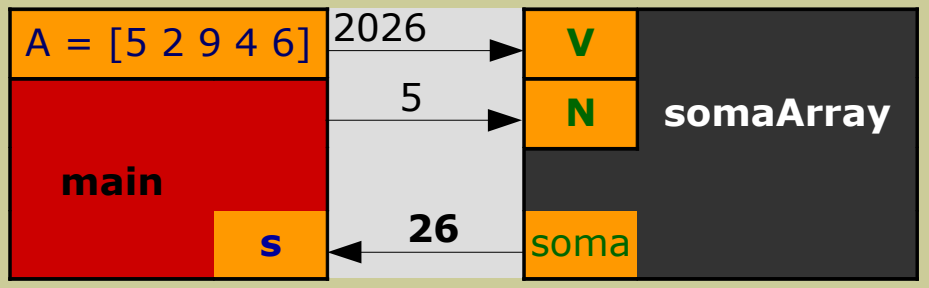


	...			...	
1001	**2026**	**A**	5065	2026	**V**
	...			...	
2026	5	A[0] V[0]	5089	5	**N**
2030	2	A[1] V[1]			
2034	9	A[2] V[2]		...	
2038	4	A[3] V[3]	5126	**5**	k
2042	6	A[4] V[4]			
	...			...	
2089		s	5249	**26**	soma
	...			...	

Passagem de um array como argumento de um subprograma

- Exemplo (devolução do resultado obtido pelo subprograma)

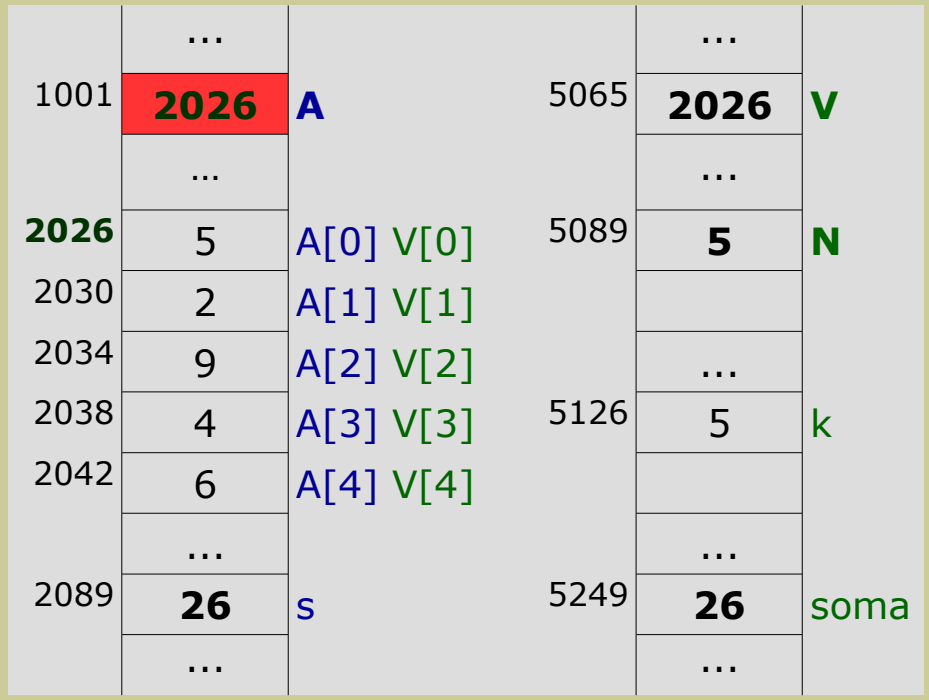
```
#include <stdio.h>
int somaArray (int V[], int N){
   int  k, soma;
   soma = 0;
   for (k = 0; k <= N-1; k = k + 1)
       soma = soma + V[k];
   return soma;
}
void main(){
   int  A[5] = { 5, 2, 9, 4, 6 };
   int  s;
   s = somaArray(A, 5);
   printf("Soma = %d\n", s);
}
```



	...			...	
1001	**2026**	**A**	5065	2026	**V**
	...			...	
2026	5	A[0] V[0]	5089	5	**N**
2030	2	A[1] V[1]			
2034	9	A[2] V[2]		...	
2038	4	A[3] V[3]	5126	5	k
2042	6	A[4] V[4]			
	...			...	
2089	**26**	s	5249	**26**	soma
	...			...	

Passagem de um array como argumento de um subprograma

- Notas
  - o espaço de memória ocupado pelo programa principal (**main**) é composto pelas variáveis **s** e **A** (A é um array de 5 elementos)
  - a declaração da variável **A** implica a reserva de memória para 6 espaços de memória: A, A[0], A[1], A[2], A[3] e A[4]
  - o espaço de memória ocupado por **somaArray** é composto pelas variáveis **N**, **k**, **soma** e **V** (V só ocupa 1 espaço)
  - os elementos do array são acedidos pelos dois módulos, através do **A** (main) e do **V** (somaArray). Porquê?

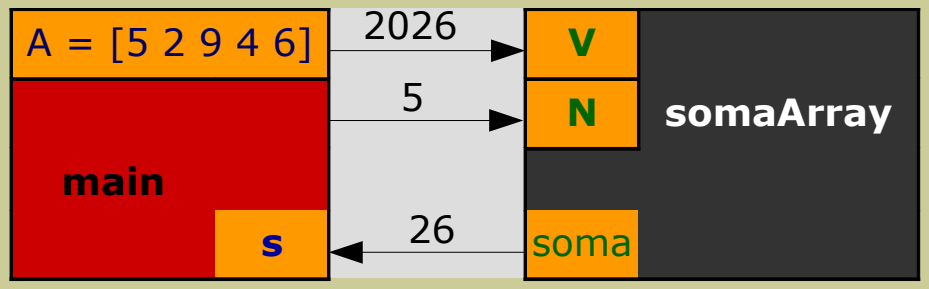


N recebe o valor do tamanho do array A

V recebe o valor de A, que é o endereço de A[0]

Passagem de um array como argumento de um subprograma

- O que se sabia até aqui é que,
  - ao passar uma variável para um subprograma, o que é passado é o seu **valor** (o seu conteúdo): o argumento formal recebe um **valor**
- O que se passa a saber a partir daqui é que,
  - se a variável representar um array, o que é passado é o **endereço** do seu primeiro elemento: o argumento formal recebe um **endereço** de memória (também é um valor)
  - neste caso, existem duas variáveis (uma no programa principal e outra no subprograma) que representam o mesmo array, o que implica que aquele array pode ser acedido e, consequentemente, modificado pelos dois módulos, mas separadamente



Exemplos

- Considerar o subprograma **lerArray** que preenche um array com N inteiros

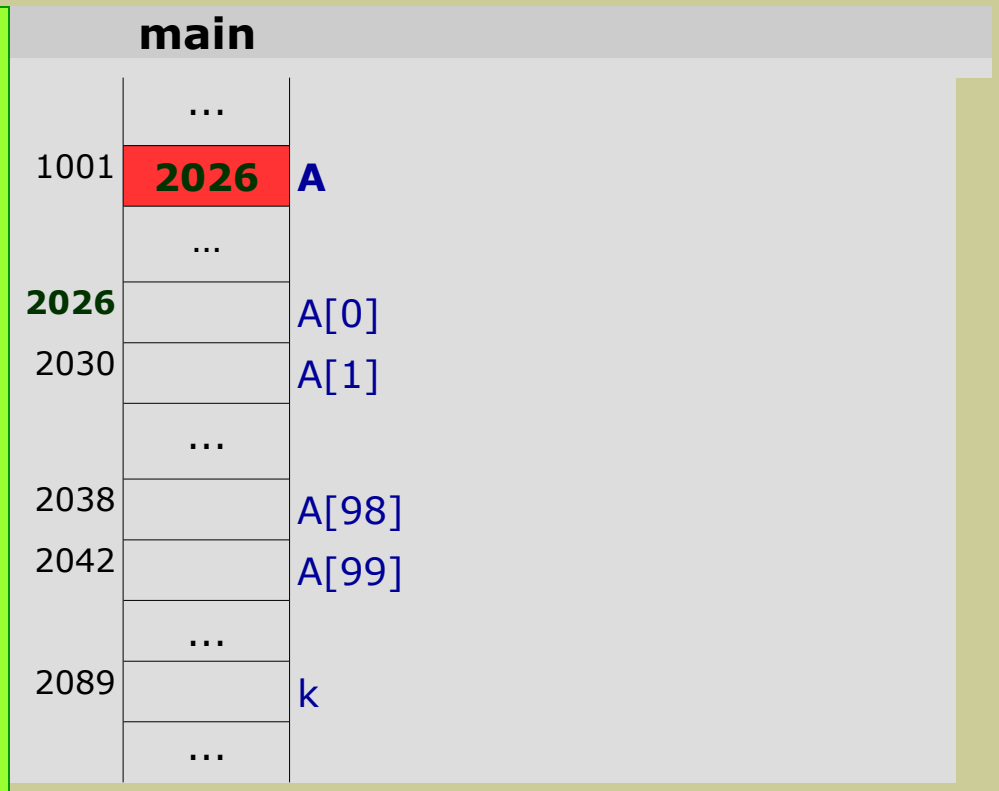
```
void lerArray (int V[], int N)
{
  int  k;
  for (k = 0; k <= N-1; k = k + 1)
  {
     printf("Inserir um numero inteiro: ");
     scanf("%d", &V[k]);
  }
}
```

- Observações:
  - a não utilização da instrução **return**
  - subprograma do tipo **void** (cebeçalho)

Exemplo 1

- Implementar um programa que verifique que elementos de um array de 1 dimensão com 100 elementos do tipo inteiro (tamanho = 100) são nulos

```
#include <stdio.h>
void lerArray (int V[], int N);
void main()
{
  int  k, A[100];
  lerArray(A, 100);
  for (k = 0; k <= 99; k = k + 1)
  {
     if (A[k] == 0)
        printf("O de indice %d é nulo.\n");
  }
}
```



memória com a reserva de **k** e do array **A**

Exemplo 1

- Implementar um programa que verifique que elementos de um array de 1 dimensão com 100 elementos do tipo inteiro (tamanho = 100) são nulos

```
#include <stdio.h>
void lerArray (int V[], int N);
void main()
{
  int  k, A[100];
  lerArray(A, 100);
  for (k = 0; k <= 99; k = k + 1)
  {
     if (A[k] == 0)
        printf("O de indice %d é nulo.\n");
  }
}
```

memória com a chamada de **lerArray**

Exemplo 1

- Implementar um programa que verifique que elementos de um array de 1 dimensão com 100 elementos do tipo inteiro (tamanho = 100) são nulos

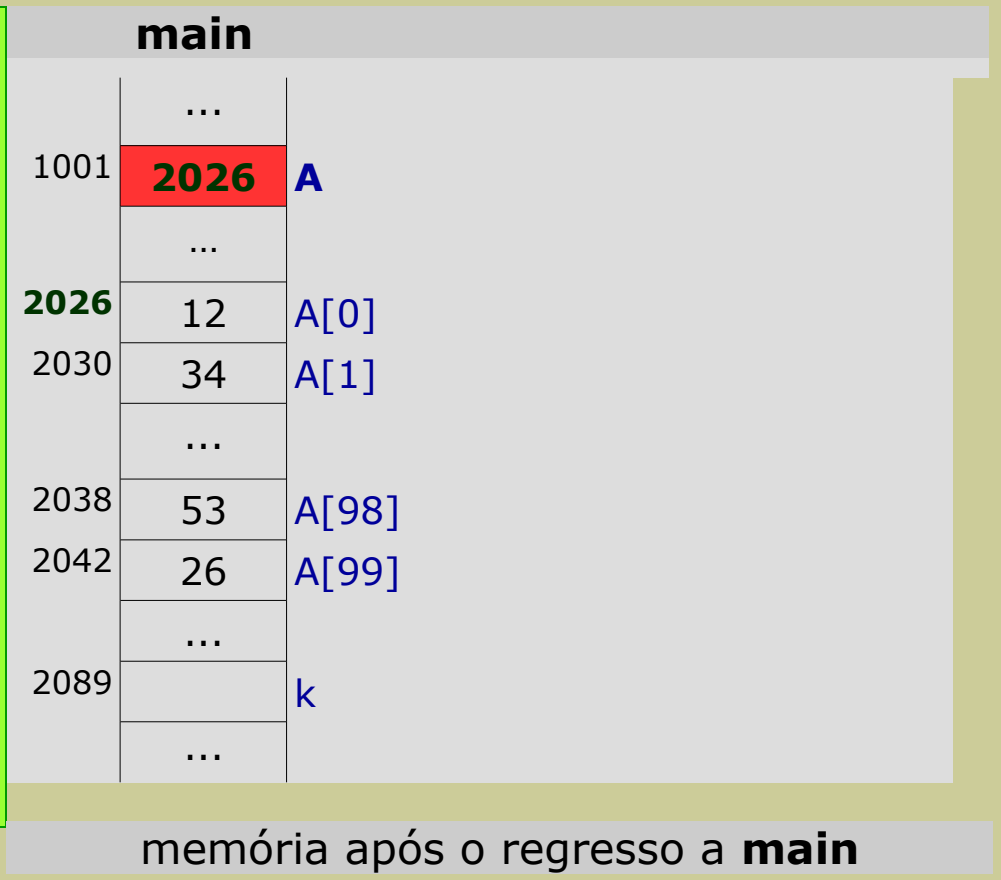
```
void lerArray (int V[], int N)
{
  int  k;
  for (k = 0; k <= N-1; k = k + 1)
  {
     printf("Inserir um numero inteiro: ");
     scanf("%d", &V[k]);
  }
}
```

memória após a execução de **lerArray**

Exemplo 2

- Implementar um programa que verifique que elementos de um array de 1 dimensão com 100 elementos do tipo inteiro (tamanho = 100) são nulos

```
#include <stdio.h>
void lerArray (int V[], int N);
void main()
{
  int  k, A[100];
  lerArray(A, 100);
  for (k = 0; k <= 99; k = k + 1)
  {
     if (A[k] == 0)
        printf("O de indice %d é nulo.\n");
  }
}
```



memória após o regresso a **main**

Exemplo 2

- Implementar um programa que faça a cópia dos elementos de um array de 1 dimensão com 100 elementos do tipo inteiro, para um segundo array

```
#include <stdio.h>
void lerArray (int V[], int N);
void main()
{
  int  k, A[100], B[100];
  lerArray(A, 100);
  for (k = 0; k <= 99; k++)
     B[k] = A[k];
}
```

- o segundo array (B) tem que ser do mesmo tipo do primeiro (A)
- o tamanho do array B tem que ser igual (ou *maior*) ao tamanho do array A
- quando se inicia a cópia, o array A já tem que ter valores valores atribuídos (lerArray)

Array de 2 dimensões

Definição

- Um array de 2 dimensões é uma "variável" com um número fixo de variáveis, onde estes elementos são todos de um mesmo tipo,
- Um array de 2 dimensões pode ser visto como uma tabela de 2 entradas (linhas e colunas) com uma quantidade de valores do mesmo tipo

Analogia

- Um array de 2 dimensões é comparado a uma matriz
  - por isso, é designado habitualmente por **matriz**

Declaração

- Sintaxe

```
tipo nome[linhas][colunas];
```

onde

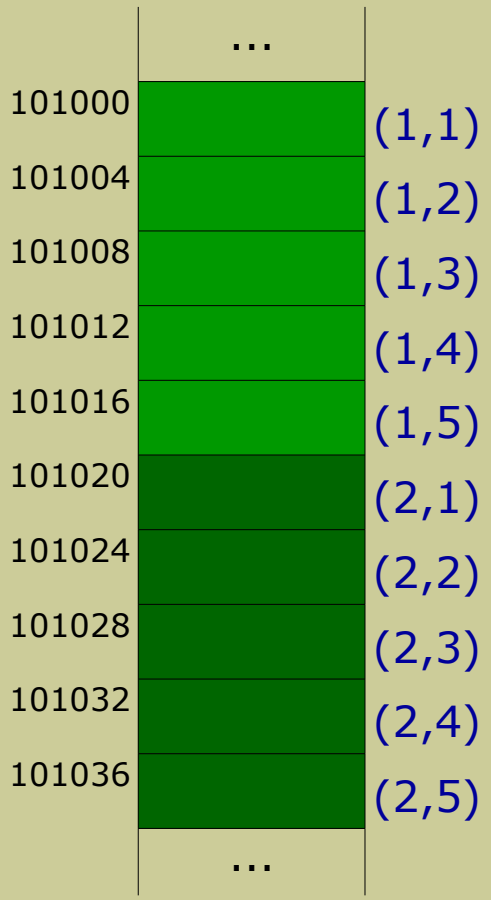
- **tipo** é o tipo de dados de todas as variáveis que formam o array
  - pode ser inteiro (int), real (float) e caráter (char)
- **nome** é a identificação das variáveis
- **linhas** é o número de linhas do array (“altura” do array)
- **colunas** é o número de colunas do array (“largura” do array)

Declaração

- Exemplo 1
  - um array de 2 dimensões com 10 elementos do tipo inteiro, distribuídos numa tabela de 2 linhas e 5 colunas (2x5 = 10)

```
...
int  M[2][5];
...
```

cada variável do tipo inteiro ocupa 4 bytes da memória

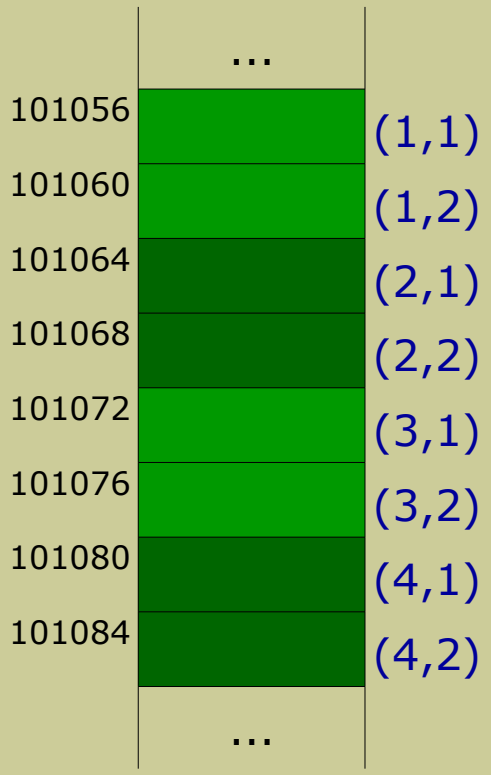


Declaração

- Exemplo 2
  - declaração de um array de 1 dimensão com 8 elementos do tipo real, distribuídos numa tabela com 4 linhas e 2 colunas (4x2 = 8)

```
...
float X[4][2];
...
```

cada variável do tipo real ocupa 4 bytes da memória



Indexação

- A utilização de um array de 2 dimensões é similar ao de um array de 1 dimensão, mas agora é necessário incluir mais um índice (ou seja, 2 índices), para indicar a posição do elemento na "tabela"
- Cada elemento do array é identificado pela posição que ocupa na "tabela", pelo que precisa de 2 índices
  - um índice para identificar a linha
  - um índice para identificar a coluna

Indexação

- Considere-se um array de 2 dimensões com 8 elementos do tipo inteiro, distribuídos por 2 linhas e 4 colunas

```
int  M[2][4];
```

  - são 8 variáveis do tipo inteiro, todas com um mesmo nome
- Então, como usar cada uma destas variáveis individualmente?
- Através de índices, que indicam a posição de cada um dos elementos:
  - para as 2 linhas, usar os índices 0 e 1
  - para as 4 colunas, usar os índices 0, 1, 2 e 3
- Portanto, os elementos do array com 2 linhas e 4 colunas, são referenciados por:

```
M[0][0], M[0][1], M[0][2], M[0][3], M[1][0], M[1][1], M[1][2], M[1][3]
```

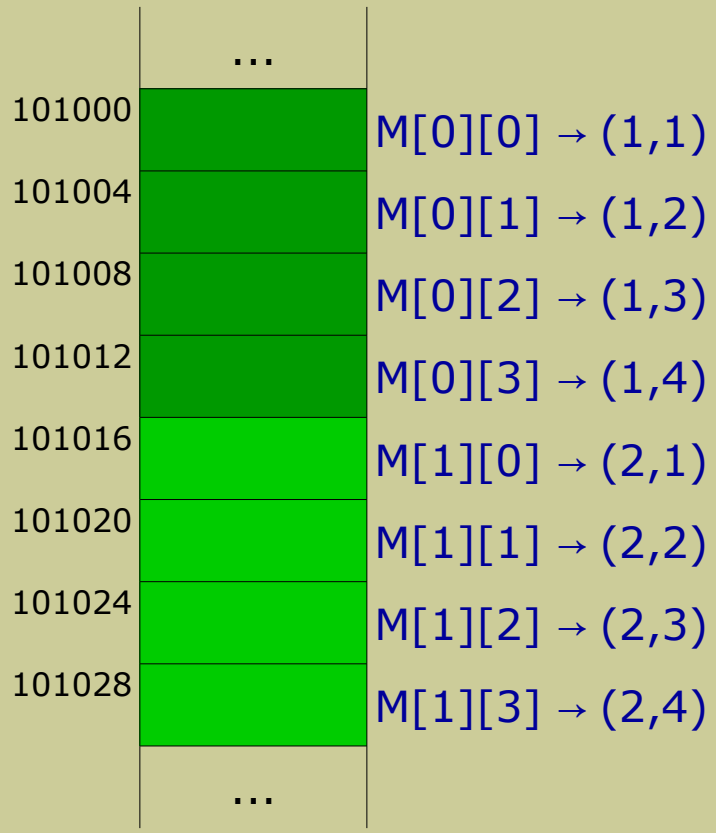
  - nome do array e o respetivo par de índices, cada índice entre parêntesis retos
  - M[L][0], M[L][1], M[L][2] e M[L][3] são os elementos da linha L (L = 0, 1)
  - M[0][C] e M[1][C] são os elementos da coluna C (C = 0, 1, 2, 3)

Indexação

- Considere-se um array de 2 dimensões de 2 linhas e 4 colunas, com 8 elementos do tipo inteiro

```
...
int M[2][4];
...
```

cada variável do tipo inteiro ocupa 4 bytes da memória



Acesso aos elementos do array

- Como cada elemento é uma variável (simples), pode-se aceder a cada uma delas da mesma forma que uma variável simples (atribuir, ler e escrever), apenas temos que indicar os índices do elemento (linha e coluna)
- A não ser em situações especiais, o acesso é sempre aos elementos de um array individualmente (ou seja, utilizando os índices)

Acesso aos elementos do array

- Exemplo 1

```
...
int  M[10][10];
...
M[0][0] = 4;
M[3][7] = 3;
M[6][2] = 23;
M[9][9] = -7;
...
```

- declaração de um array **M** de 2 dimensões de inteiros com 10 linhas e 10 colunas
- atribuir o número inteiro 4 ao elemento da linha 0 e coluna 0 do array M (o "primeiro")
- atribuir o número inteiro 3 ao elemento da linha 3 e coluna 7 do array M
- atribuir o número inteiro 23 ao elemento da linha 6 e coluna 2 do array M
- atribuir o número inteiro -7 ao elemento da linha 9 e coluna 9 do array M (o "último")

Acesso aos elementos do array

- Exemplo 2

```
...
int  L, C;
float X[5][10];
...
X[4][8] = 2.1;
L = 2; C = 8;
printf("Insira um valor real: ");
scanf("%f", &X[L][C]);
printf("Elemento da linha %d e coluna %d = %f\n", L+2, C, X[L+2][C]);
...
```

- declaração de um array **X** de 2 dimensões, com 5 linhas e 10 colunas, de números reais
- atribuir o número real 2.1 ao elemento da linha 4 e coluna 8 do array M
- ler do teclado um número real e atribuí-lo ao elemento da linha L e coluna C do array M
- escrever no monitor o valor do elemento da linha L+2 e coluna C do array M

Acesso aos elementos do array

- Observações
  - para a declaração

```
...
int  M[10][10];
...
```

é **errado**

```
...
M = 2;
scanf("%d", &M[1]);
printf("%d\n", M);
...
```

pois faltam os índices

Acesso aos elementos do array

- Observações
  - para a declaração

```
...
int  M[10][10];
...
M[10][0] = 21;
M[1][10] = 32;
...
```

estão **errados**

- o array M só tem 10 linhas e M[10][0] referencia a linha 11, que não existe
- o array M só tem 10 colunas e M[1][10] referencia a coluna 11, que não existe

A utilização de repetições

- De uma maneira geral, o uso de arrays de 2 dimensões envolve a aplicação de instruções de repetição (ciclos),
  - se pretender-se percorrer o array por linhas (linha a linha), é necessário dois ciclos
    - um exterior para percorrer as linhas
    - um interior para percorrer as colunas
  - se pretender-se percorrer o array por colunas (coluna a coluna), é necessário dois ciclos
    - um exterior para percorrer as colunas
    - um interior para percorrer as linhas
- Exemplo:

  preencher um array de elementos do tipo inteiro, com 4 linhas e 5 colunas

A utilização de repetições

- Exemplo:

preencher um array de elementos do tipo inteiro, com 4 linhas e 5 colunas

```
#include <stdio.h>
void main() {
  int  L, C, M[4][5];
  L = 0;
  while(L <= 3) {    // ou while(L < 4)
     C = 0;
     while(C <= 4) {  // ou while(C < 5)
        printf("Insira um numero inteiro: ");
        scanf("%d", &M[L][C]);
        C = C + 1;
     }
     L = L + 1;
  }
}
```

A utilização de repetições

- Exemplo:

  preencher um array de elementos do tipo inteiro, com 4 linhas e 5 colunas

  - por se conhecer o número de iterações dos ciclos (iguais ao número de linhas e de colunas do array), a instrução de repetição mais "adequada" para ambos é o **for**

```
#include <stdio.h>
void main()
{
  int  L, C, M[4][5];
  for (L = 0; L <= 3; L = L + 1)
     for (C = 0; C <= 4; C = C + 1)
     {
        printf("Insira um numero inteiro: ");
        scanf("%d", &M[L][C]);
     }
}
```

Passagem de um array como argumento de um subprograma

- A passagem de um array para um subprograma é feita através do seu nome
- Quando um array é passado como argumento a um subprograma, o compilador **tem que ser informado**
  - da quantidade de colunas, ou
  - da quantidade de linhas **e** da quantidade de colunas
- As quantidades de linhas e de colunas a considerar é da inteira responsabilidade do programador, pelo que também devem ser passadas como argumentos, mas de forma independente
- **ATENÇÃO**
  - O nome do array funciona como endereço do "primeiro" elemento do array (?[0][0]), ou seja, a sua localização em memória
- **CONSEQUÊNCIA**
  - Quando se passa um array para um subprograma, o que se está a passar é o endereço do "primeiro" elemento, e não o array no seu todo

Passagem de um array como argumento de um subprograma

- Exemplo 1
  - subprograma para ler um array de 2 dimensões com 5 colunas

```
void lerArray2D (int M[][5], int LIN, int COL)
{
  int  L, C;
  for (L = 0; L <= LIN-1; L = L + 1)     // ou L < LIN
     for (C = 0; C <= COL-1; C = C + 1) // ou C < COL
     {
        printf("Insira um numero inteiro M[%d][%d]: ", L, C);
        scanf("%d", &M[L][C]);
     }
}
```

Passagem de um array como argumento de um subprograma

- Exemplo 1
  - subprograma para escrever um array de 2 dimensões com 4 linhas e 5 colunas

```
void escreverArray2D (int M[4][5], int LIN, int COL)
{
  int  L, C;
  for (L = 0; L <= LIN-1; L = L + 1)
  {
     printf("Linha %d:\n", L);
     for (C = 0; C <= COL-1; C = C + 1)
        printf("%d  ", M[L][C]);
     printf("\n");
  }
}
```

Passagem de um array como argumento de um subprograma

- Exemplo 1
  - programa para ler um array de 2 dimensões com 4 linhas e 5 colunas, de inteiros, e depois mostrá-lo no monitor

```
#include <stdio.h>
void lerArray2D (int M[][5], int LIN, int COL);
void escreverArray2D (int M[4][5], int LIN, int COL);
void main()
{
  int  X[4][5];
  lerArray2D(X, 4, 5);
  printf("Mostrar array:\n");
  escreverArray2D(X, 4, 5);
}
```

Passagem de um array como argumento de um subprograma

- Exemplo 1
  - programa para ler um array de 2 dimensões com M linhas e N colunas, de inteiros, e depois mostrá-lo no monitor

```
#include <stdio.h>
void lerArray2D (int M[][5], int LIN, int COL);
void escreverArray2D (int M[4][5], int LIN, int COL);
void main()
{
  int  M, N, X[4][5];
  M = 3;
  N = 2;
  lerArray2D(X, M, N);
  printf("Mostrar array:\n");
  escreverArray2D(X, M, N);
}
```