

## ARRAY e POINTER em C

Na memória principal, todo byte é identificado por um endereço numérico. Por exemplo, em uma memória de 64 KB, os endereços vão de 0 a 65535. Em uma memória de 1 MB, os endereços vão de 0 a 1.048.575, de 8 MB, vão de 0 a 8.388.607. ( $8 \times 2^{20}$ ).

0	
1	
...	
1000	
1001	
...	
65535	

A memória de trabalho (RAM) é dividida em partes para o armazenamento do programa, dos dados da função principal, da pilha de execução das funções etc e a parte restante (heap) é utilizada para as alocações dinâmicas.

A declaração

```
char Pal[20];
```

faz com que seja reservada uma área de memória com 20 bytes para o armazenamento de 20 caracteres, que ocupam 1 byte cada.

O identificador do array é de um pointer para o local de índice 0.

Caso tenhamos armazenado as 5 primeiras letras do alfabeto no array Pal, teríamos, por exemplo, a alocação mostrada na tabela ao lado.

Veja que no local identificado por Pal está armazenado o endereço da primeira posição do array.

A alocação de espaço na memória para o armazenamento do array Pal pode ser:

- estática – na área de DADOS
- automática – no STACK, durante a execução de uma função
- dinâmica – na área HEAP

Uma vez que Pal foi declarada como um array de dados do tipo char, segue que Pal é um pointer para o tipo char. A passagem de um array como parâmetro de uma função na verdade é a passagem de um pointer, por isso dizemos que quando se utiliza um array como parâmetro de uma função, a passagem dos valores é feita por referência.

Por exemplo, considerando a função

```
void alterarCadeia(char A[], int n)
```

em que o parâmetro formal A é um array, ao executar a chamada da função

```
char Pal[20];
```

```
m = 5;
```

```
alterarCadeia(Pal,m);
```

colocamos apenas o identificador do array como parâmetro real e o endereço do mesmo é copiado em A.

Os parâmetros formais de alterarCadeia são A e n e os parâmetros reais na chamada de alterarCadeia são Pal e m. Por causa disso, o espaço de armazenamento do array Pal é compartilhado pela função principal e pela função que fez a chamada, fazendo com que os dados em Pal possam ser alterados por meio da função.

1000	A	Pal[0]
1001	B	Pal[1]
1002	C	Pal[2]
1003	D	Pal[3]
1004	E	Pal[4]
...	xxx	...
1019	xxx	Pal[19]
1020		
1021		m
1022		
1023		
1024		
1025	1000	Pal

1000	A	Pal[0]
1001	B	Pal[1]
1002	C	Pal[2]
1003	D	Pal[3]
1004	E	Pal[4]
...	xxx	...
1019	xxx	Pal[19]
1020	xxx	
1021	5	m
1022		
1023		
1024		
1025	1000	Pal
	1000	A
	5	n

O mesmo ocorre para matrizes. A declaração

```
int Ma[3][3];
```

faz com que seja reservada uma área com 36 bytes para o armazenamento de 9 números inteiros, que ocupam 4 bytes cada.

A tabela ao lado seguir mostra a alocação do espaço na memória.

Uma vez que o identificador da matriz é um pointer, podemos fazer a chamada

```
alterarMatriz(Ma);
```

em que a função

```
void alterarMatriz(char V[3][3]);
```

tem por finalidade, por exemplo, fazer alguma alteração nos dados definidos em Ma. Veja o exemplo:

```
#define NV 3
```

```
void mostrarMatriz(int [NV][NV]);  
void criarMatriz(int [NV][NV]);
```

```
main () {  
    int matrizM[NV][NV];  
    int P[NV][NV] = {{1,2,1},{3,4,3},{5,6,5}};  
    mostrarMatriz(P);  
    criarMatriz(P);  
    mostrarMatriz(P);  
    printf("  \n");  
    system("PAUSE");  
}  
  
void mostrarMatriz(int M[NV][NV]) {  
    int i,j;  
    for (i=0;i<NV;i++) {  
        for(j=0;j<NV;j++) printf(" %d  ",M[i][j]);  
        printf("\n");  
    }  
    printf("\n");  
}  
  
void criarMatriz(int M[NV][NV]) {  
    int i,j;  
    for(i=0;i<NV;i++)M[i][0]=i;  
    for(i=0;i<NV;i++)M[i][1]=1;  
    for(i=0;i<NV;i++)M[i][2]=i;  
}
```

1000	A	Ma[0][0]	Ma[0]
1004	B	Ma[0][1]	
1008	C	Ma[0][2]	
1012	D	Ma[1][0]	Ma[1]
1016	E	Ma[1][1]	
1020	F	Ma[1][2]	
1024	G	Ma[2][0]	Ma[2]
1028	H	Ma[2][1]	
1032	I	Ma[2][2]	
...	xxx		
1058	xxx		
1059	1000	Ma	
1060	xxx		
1061			
...	xxx		
	1000	V	
	xxx		

Se na implementação da função alterarMatriz houver alguma declaração de um array, o espaço para o mesmo é alocado na área da memória denominada STACK e após a execução da função esse espaço é devolvido para a área de bytes livres (área HEAP). As alocações que ocorrem por conta da execução de alguma função são denominadas automáticas.

Outro exemplo - com a declaração

```
char matrizCh[3][2];
```

alocamos espaço para uma matriz de caracteres, com 6 bytes, cujos locais são identificados pelas variáveis indexadas:

matrizCh[0][0],matrizCh[0][1],matrizCh[1][0],matrizCh[1][1],matrizCh[2][0],matrizCh[2][1]

Além disso, o identificador matrizCh é um pointer (são 4 bytes) contendo o endereço de memória da posição identificada por matrizCh[0][0]

Podemos imaginar o espaço de armazenamento dos caracteres como uma matriz (matemática) com 3 linhas e 2 colunas:

	0	1
matrizCh = 0		
1		
2		

A matriz matrixCh pode ser inicializada de várias formas diferentes:

```
matrizCh[0][0] = '1';  
matrizCh[0][1] = '2';  
matrizCh[1][0] = '3';  
matrizCh[1][1] = '4';  
matrizCh[2][0] = '5';  
matrizCh[2][1] = '6';
```

ou assim:

```
char matrizCh[3][2] = {'1', '2', '3', '4', '5', '6'};
```

ou assim:

```
char matrizCh[3][2] = {{49,50},{51,52},{53,54}};
```

armazenando:

	0	1
matrizCh = 0	1	2
1	3	4
2	5	6