

# **Alocação dinâmica de memória**

**Roberto Rocha**



**E se não tenho o tamanho do  
arranjo?**

# Alocação dinâmica de memória

A alocação é estática acontece antes que o programa comece a ser executado:

```
char c;  
int i;  
int v[10];
```

Em alguns casos, a quantidade de memória a alocar só se torna conhecida durante a execução do programa.

Em C, para lidar com essa situação é preciso recorrer à alocação dinâmica de memória.

A alocação dinâmica é administrada pelas funções **malloc**, **realloc** e **free**, que estão na biblioteca `stdlib`.

Para usar essa biblioteca:

```
#include <stdlib.h>
```

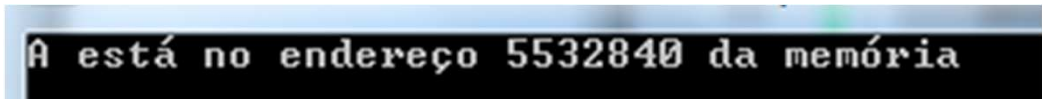
# Alocação dinâmica de memória

## A função malloc

A função malloc (o nome é uma abreviatura de *memory allocation*) aloca espaço para um bloco de bytes consecutivos na memória RAM (= *random access memory*) do computador e devolve o endereço desse bloco.

O número de bytes é especificado no argumento da função.  
No seguinte fragmento de código, malloc aloca 1 byte:

```
char *pChar; ← Ponteiro para char
pChar = malloc (1); ← Aloca 1 byte na memoria e atribui a pChar o endereço da memória
*pChar='A'; ← Atribui o caracter 'A' ao endereço apontado por pChar
printf("%c está no endereço %d da memória\n",*pChar,pChar); ← Exibe o conteúdo e endereço da memória
```



```
A está no endereço 5532840 da memória
```

# Alocação dinâmica de memória

## A função malloc

O endereço devolvido por malloc é do tipo genérico void \*.  
O programador armazena esse endereço num ponteiro de tipo apropriado.  
No exemplo, o endereço foi armazenado no ponteiro pChar, que é do tipo ponteiro-para-char.  
A transformação do ponteiro genérico em ponteiro-para-char é automática;  
não é necessário escrever pChar = (char \*) malloc (1);

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <locale.h>
4
5  int main()
6  {
7      setlocale(LC_ALL, "portuguese");
8      char *pChar;
9      pChar = malloc (1);
10     *pChar='A';
11     printf("%c está no endereço %d da memória\n", *pChar, pChar);
12
13     return 0;
14 }
```

A está no endereço 5532840 da memória

# Alocação dinâmica de memória

## Operador `sizeof` (tipo de dado)

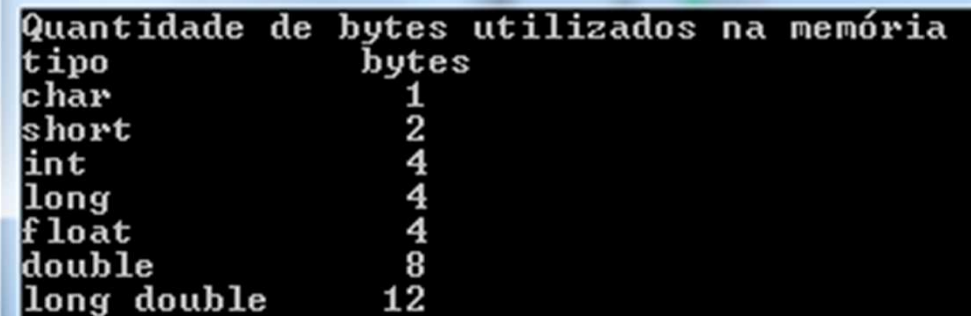
`sizeof` não é uma função mas um operador, tal como `return`, por exemplo.

Os parênteses na expressão `sizeof (data)` são necessários porque `data` é um tipo-de-dados (os parênteses são análogos aos do `casting` – o modelo de dados a ser utilizado).

O operador `sizeof` também pode ser aplicado diretamente a uma variável:

se `var` é uma variável então `sizeof var` é o número de bytes ocupado por `var`.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <locale.h>
4
5  int main()
6  {
7      setlocale(LC_ALL, "portuguese");
8      printf("Quantidade de bytes utilizados na memória\n");
9      printf("tipo          bytes\n");
10     printf("char           %2d\n", sizeof(char));
11     printf("short          %2d\n", sizeof(short));
12     printf("int            %2d\n", sizeof(int));
13     printf("long           %2d\n", sizeof(long));
14     printf("float          %2d\n", sizeof(float));
15     printf("double         %2d\n", sizeof(double));
16     printf("long double    %2d\n", sizeof(long double));
17
18     return 0;
19 }
```

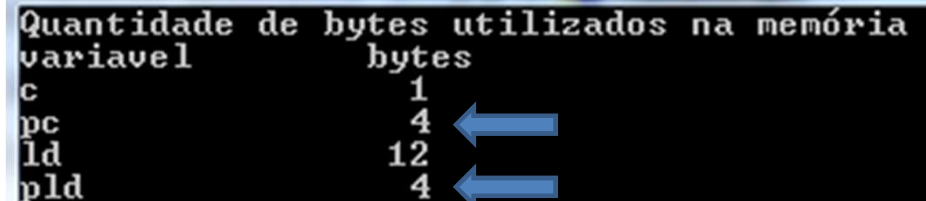


tipo	bytes
char	1
short	2
int	4
long	4
float	4
double	8
long double	12

# Alocação dinâmica de memória

## Operador `sizeof` (tipo de dado)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <locale.h>
4
5  int main()
6  {
7      setlocale(LC_ALL, "portuguese");
8      char c;
9      char *pc;
10     long double *pld;
11     long double ld;
12
13
14     printf("Quantidade de bytes utilizados na memória\n");
15     printf("variavel      bytes\n");
16     printf("c              %2d\n", sizeof(c));
17     printf("pc              %2d\n", sizeof(pc));
18     printf("ld              %2d\n", sizeof(ld));
19     printf("pld             %2d\n", sizeof(pld));
20
21     return 0;
22 }
```



variavel	bytes
c	1
pc	4
ld	12
pld	4

# Tipos de dados

Tipo	Descrição	Tamanho	intervalo
char	caracter	1	-128 a 127
signed char	Caractere com sinal	1	-128 a 127
unsigned char	Caractere sem sinal	1	0 a 255
int	Inteiro	4	-2,147,483,648 a 2,147,483,647
signed Int	Inteiro com sinal	4	-2,147,483,648 a 2,147,483,647
unsigned int	Inteiro sem sinal	4	0 a 4,234,967,295
short int	Inteiro curto	2	-32.768 a 32.767
signed short int	Inteiro curto com sinal	2	-32.768 a 32.767
unsigned short int	Inteiro curto sem sinal	2	0 a 65.535



# Tipos de dados

Tipo	Descrição	Tamanho	intervalo
long int	Inteiro long	4	-2,147,483,648 a 2,147,483,647
signed long int	Inteiro long com sinal	4	-2,147,483,648 a 2,147,483,647
unsigned longint	Inteiro longo sem sinal	4	0 a 4,234,967,295
float	Ponto flutuante com precisão simples precisão de 7 dígitos	4	$3.4 \cdot 10^{-38}$ a $3.4 \cdot 10^{+38}$
double	Ponto flutuante com precisão dupla precisão de 15 dígitos	8	$1.7 \cdot 10^{-38}$ a $1.7 \cdot 10^{+38}$
long double	Ponto flutuante com precisão dupla longo	12	$3.4 \cdot 10^{-4932}$ a $3.4 \cdot 10^{+4932}$

# Tipos de dados

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <locale.h>
4
5  int main()
6  {
7      setlocale(LC_ALL, "portuguese");
8      float m;
9      int v[10];
10     printf(" TIPO          | BYTES \n");
11     printf(" char .....: %2d bytes\n", sizeof(char));
12     printf(" signed char ...: %2d bytes\n", sizeof(signed char));
13     printf(" unsigned char.: %2d bytes\n", sizeof(unsigned char));
14
15     printf(" int.....: %2d bytes\n", sizeof(int));
16     printf(" signed int....: %2d bytes\n", sizeof(signed int));
17     printf(" unsigned int...: %2d bytes\n", sizeof(unsigned int));
18     printf(" short.....: %2d bytes\n", sizeof(short));
19     printf(" signed short...: %2d bytes\n", sizeof(signed short));
20     printf(" unsigned short: %2d bytes\n", sizeof(unsigned short));
21     printf(" long.....: %2d bytes\n", sizeof(long));
22     printf(" signed long...: %2d bytes\n", sizeof(signed long));
23     printf(" unsigned long.: %2d bytes\n", sizeof(unsigned long));
24
25     printf(" float .....: %2d bytes\n", sizeof(float));
26     printf(" double.....: %2d bytes\n", sizeof(double));
27     printf(" long double...: %2d bytes\n\n", sizeof(long double));
28     printf("\nO tamanho de m é %2d \n\n", sizeof(m));
29     printf("\nO tamanho de v é %2d \n\n", sizeof(v));
30     getch();
31     return 0;
32 }
```

TIPO	:	BYTES
char .....	:	1 bytes
signed char ...	:	1 bytes
unsigned char..	:	1 bytes
int.....	:	4 bytes
signed int....	:	4 bytes
unsigned int...	:	4 bytes
short.....	:	2 bytes
signed short...	:	2 bytes
unsigned short:	:	2 bytes
long.....	:	4 bytes
signed long...	:	4 bytes
unsigned long..	:	4 bytes
float .....	:	4 bytes
double.....	:	8 bytes
long double...	:	12 bytes

O tamanho de m é 4

O tamanho de v é 40

## Arranjo de Uma Dimensão - Vetor

- Construa uma **função** que retorne um vetor real com dados informados pelo teclado de tamanho N – passe o tamanho N por valor.
- Construa um procedimento para imprimir um vetor real de tamanho N – passe o vetor e o tamanho N por valor

Escrever um programa que leia um valor e chame a função e o procedimentos criados.

## Matrizes de Uma Dimensão - Vetor

- Construa uma **função** que retorne um vetor real com dados informados pelo teclado de tamanho N – passe o tamanho N por valor.

```
funcao leVetor(N:inteiro):vetor :conjunto[] de real  
var  
    i:inteiro  
    v:conjunto[] de real  
inicio  
    para i de 0 ate N-1 passo 1 faca  
        leia(v[i])  
    fimpara  
    retorne v  
fimfuncao
```


```
40 float * leVetor (int tam)  
41 {  
42     int i;  
43     float *v;  
44     v=malloc(sizeof(float)*tam);  
45     for (i=0;i<tam; i=i+1)  
46     {  
47         printf("Digite o %d termo do vetor:", i);  
48         scanf("%f",&v[i]);  
49     }  
50  
51     return v;  
52 }
```

## Matrizes de Uma Dimensão - Vetor

- Construa um procedimento para imprimir um vetor real de tamanho N – passe o vetor e o tamanho N por valor

```
procedimento imprimeVetor [v:conjunto[] de real, N:inteiro, nomeVetor:literal)  
var  
    i:inteiro  
inicio  
    para i de 0 ate N-1 passo 1 faca  
        escreva (nomeVetor,"[",i,"]=",v[i])  
    fimpara  
fimprocedimento
```

Em C os vetores são  
sempre por  
referência



```
54 void imprimeVetor(float *v, int tam, char *nomeVetor)  
55 {  
56     int i;  
57     for (i=0;i<tam;i=i+1)  
58     {  
59         printf("%s[%d]=%.2f\n", nomeVetor, i, v[i]);  
60     }  
61 }
```

# Matrizes de Uma Dimensão - Vetor

- Construa uma **função** que retorne um vetor real com dados informados pelo teclado de tamanho N – passe o tamanho N por valor.
- Construa um procedimento para imprimir um vetor real de tamanho N – passe o vetor e o tamanho N por valor

Escrever um programa que leia um valor e chame a função e o procedimentos criados.

```
var  
  A: conjunto[] de real  
  tam:inteiro  
inicio  
  leia(tam)  
  A <- leVetor[tam]  
  imprimeVetor[A,tam]  
fimalgoritmo
```

```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  #include <locale.h>  
4  float * leVetor (int );  
5  void imprimeVetor(float *, int,  char *);  
6  
7  int main()  
8  {  
9      setlocale(LC_ALL, "portuguese");  
10     float *A;  
11     int tam;  
12     printf("Informe o tamanho do vetor:");  
13     scanf("%d",&tam);  
14     A=leVetor(tam);  
15     printf("impressão após a leitura \n");  
16     imprimeVetor(A,tam,"A");  
17     free(A);  
18     A=NULL;  
19     return 0;  
20 }
```

```
Informe o tamanho do vetor:5  
Digite o 0 termo do vetor:0  
Digite o 1 termo do vetor:1  
Digite o 2 termo do vetor:2  
Digite o 3 termo do vetor:3  
Digite o 4 termo do vetor:4  
impressão após a leitura  
A[0]=0,00  
A[1]=1,00  
A[2]=2,00  
A[3]=3,00  
A[4]=4,00
```

→ Liberar o espaço alocado



## A função free

As variáveis declaradas localmente desaparecem assim que a execução da função termina. No entanto as variáveis alocadas dinamicamente continuam a existir mesmo depois que a execução da função termina, conforme demonstrado na função `leVetor`.

Se for necessário liberar a memória ocupada por essas variáveis, é preciso recorrer à função `free`.

A função `free` desaloca a porção de memória alocada por `malloc`.

A instrução `free (ponteiro)` avisa ao sistema que o bloco de bytes apontado pelo ponteiro está disponível para reciclagem.

A próxima invocação de `malloc` poderá tomar posse desses bytes

# Redimensionamento e a função realloc

Caso necessite alterar, durante a execução do programa, o tamanho alocado por malloc pode-se utilizar a função realloc.

A função realloc recebe o endereço previamente alocado por malloc (ou mesmo realloc) e o novo número de bytes que o bloco redimensionado deve ter. A função aloca o novo bloco, copia para ele o conteúdo do bloco original, e devolve o endereço do novo bloco

`realloc (ponteiro, tamanho);`

Se o novo bloco for uma extensão do bloco original, seu endereço é o mesmo do original (e o conteúdo do original não precisa ser copiado para o novo).

Caso contrário, realloc copia o conteúdo do bloco original para o novo e libera o bloco original (invocando free).

O tamanho do novo bloco também pode ser menor que o do bloco original.



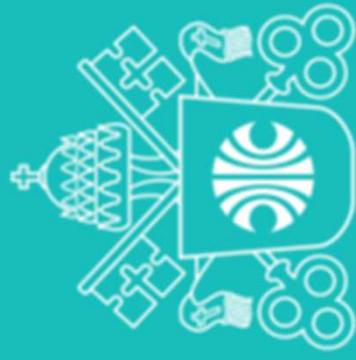
## Exemplo:

Uma sala de aula possui  $n$  alunos. Leia a matrícula de cada aluno. Em seguida suponha que seja necessário unir duas salas a anterior e mais uma com  $m$  alunos. Implemente a solução.

Uma sala de aula possui  $n$  alunos. Leia a matricula de cada aluno. Em seguida suponha que seja necessário unir duas salas a anterior e mais uma com  $m$  alunos. Implemente a solução.

```
6  int main()
7  {
8      setlocale(LC_ALL, "portuguese");
9      int *A,i,n,m;
10     printf("Numero de alunos:");
11     scanf("%d",&n);
12     A=malloc(n*sizeof(int));
13     for (i=0;i<n;i=i+1)
14     {   printf("A[%d]=", i);
15         scanf("%d",&A[i]);
16     }
17     printf("Numero de alunos da segunda sala:");
18     scanf("%d",&m);
19     A=realloc(A, (n+m)*sizeof(int));
20     for (i=n;i<(n+m);i=i+1)
21     {   printf("A[%d]=", i);
22         scanf("%d",&A[i]);
23     }
24     printf("impressão após a leitura \n");
25     imprimeVetor(A, (n+m), "A");
26     free(A);
27     A=NULL;
28     return 0;
29 }
```

```
Numero de alunos:4
A[0]=0
A[1]=1
A[2]=2
A[3]=3
Numero de alunos da segunda sala:6
A[4]=4
A[5]=5
A[6]=6
A[7]=7
A[8]=8
A[9]=9
impressão após a leitura
A[0]=0
A[1]=1
A[2]=2
A[3]=3
A[4]=4
A[5]=5
A[6]=6
A[7]=7
A[8]=8
A[9]=9
```



# PUC Minas Virtual