



**Ponteiro**  
**ou**  
**Apontador**

Variável que guarda o endereço de memória de outra variável (para onde aponta).

`<tipo> * <nome_ponteiro>` declaração de `<nome_ponteiro>` como um ponteiro que aponta para uma variável do tipo `<tipo>`  
(não reserva espaço de memória, apenas espaço para um endereço de memória)

`*<nome_ponteiro>` conteúdo da zona de memória guardada pelo ponteiro `<nome_ponteiro>`  
(valor para o qual `<nome_ponteiro>` aponta)

`&<nome_ponteiro>` endereço de memória de `<nome_ponteiro>`

`*<nome_ponteiro> = NULL` inicialização do ponteiro `<nome_ponteiro>` como `NULL`, ou seja, a apontar para nada

Exemplo

```
int n,m,*p;
```

```
n = 10;
```

```
p = &n;
```

```
m = *p;
```

```
*p = 20;
```

Endereço de n

n	
---	--

Endereço de m

m	
---	--

Endereço de p

p →

n	10
n	10
n	10
n	20

m	
m	
m	10
m	10

p	→
p	Endereço de n
p	Endereço de n
p	Endereço de n

```
int *p;  
*p = 5;
```

*p* ainda não está a apontar para um inteiro, apenas está declarado  
**segmentation fault!**

→ Só é possível porque a variável *n* já existe.

Exemplo

```
char car = 'a';
```

```
char * ptr = &car;
```

	valor	car	ptr
valor		a	101
endereço memória	100	101	105

	valor	
car	a	valor da variável car
&car	101	endereço de car
ptr	101	valor do ponteiro ptr (endereço de car)
&ptr	105	endereço de ptr
*ptr	a	valor apontado por ptr

Exemplos

```
int n,*p;  
  
p = &n;  
*p = 5;  
  
printf("n = %i",n);
```

ou

```
int *pont;  
  
pont = (int *)malloc(sizeof(int));  
*pont = 5;  
  
printf("valor = %i",*pont);  
  
free(pont);
```

*malloc*Funções da biblioteca **stdlib.h***free***void \* malloc(size\_t size)**

The **malloc()** function allocates size bytes of memory and returns a pointer to the allocated memory.

If successful malloc() returns a pointer to allocated memory.

If there is an error, it returns a NULL pointer and set errno to ENOMEM.

```
pont = (int *)malloc(sizeof(int));
```

**void free(void \*ptr)**

The **free()** function deallocates the memory allocation pointed to by ptr.

If ptr is a NULL pointer, no operation is performed.

The free() function does not return a value.

```
free(pont);
```

Usando a função `sizeof`

tipo	tamanho (bytes)	tamanho do ponteiro (bytes)
char	1	8
int	4	8
float	4	8
double	8	8

```
char c,*pc;      (int)sizeof(c)  (int)sizeof(pc)
                ...              ...
```

Exemplo

```
int a = 2, b = 3;
int *p, *q;
```

```
p = &a;
q = &b;
```

//expressão

Valor da expressão

```
p == &a
```

```
*p - *q
```

```
**&p
```

```
*p + 1
```

```
*(q - 2)
```

Uma comparação (<, <=, >, >=, ==, !=) só pode ser realizada entre ponteiros do mesmo tipo.

Exemplo

```
char car = 'a';
char * ptr = &car;
```

variável	Identificador do formato	resultado
car	%c	a
	%i ou %d	97
&car ou ptr	%i	-534504853
	%s	a
	%p (pointer)	0x7ffe0241a6b
	%u (unsigned integer)	3760462443
	%li ou %ld	140732658883179
	%x ou %X (hexadecimal)	e0241a6b

É possível realizar operações aritméticas com ponteiros (**do mesmo tipo**):

- Adicionar (`ptr++`; `ptr = ptr + 2`; `ptr +=4`; `ptr1 + ptr2`)
- Subtrair (`ptr--`; `ptr = ptr - 2`; `ptr -=4`; `ptr1 - ptr2`)
- Incrementar → avança no endereço de memória o número de bytes (`sizeof(tipo de dados)`) que o seu tipo ocupa
- Decrementar → recua no endereço de memória o número de bytes que o seu tipo ocupa

Estas operações podem ser úteis para manipular tabelas.

Exemplo

```
int v[] = {10,20,30,40,50};
int * pv;

pv = &v[0]; //atribui a pv o endereço do primeiro
            elemento da tabela
printf("%i\n",*pv);

++pv; //ponteiro aponta para segundo elemento da tabela
printf("%i\n",*pv);
```

Resultado da execução:

10  
20

Exemplo

Os elementos de uma  
tabela ocupam posições  
consecutivas de  
memória, sendo o nome  
da tabela igual ao  
endereço do primeiro  
elemento, isto é, o  
menor endereço

```
int v[] = {10,20,30,40,50};
int * pv;

pv = v; //alternativa a pv = &v[0];

for(int j = 0; j < 5; j++)
    printf("v[%i] = %i\n",j,*(v+j));
```

Resultado da execução:

```
v[0] == *v
v[1] == *(v + 1)
v[2] == *(v + 2)
v[n] == *(v + n)

v[0] = 10
v[1] = 20
v[2] = 30
v[3] = 40
v[4] = 50
```

Exemplo

```
//...
#define DIM 20

void inicializaMatriz0(float *m)
void inicializaMatrizi(float *m)
float * soma(float *p1, float *p2)
void mostraMatriz(float *m)

int main ()
{
    float m1[DIM][DIM], m2[DIM][DIM], s[DIM][DIM];
    float *pm1 = NULL, *pm2 = NULL, *ps = NULL;

    pm1 = m1[0];
    pm2 = m2[0];

    inicializaMatriz0(pm1);
    inicializaMatrizi(pm2);

    ps = s[0];
    ps = soma(pm1, pm2);

    mostraMatriz0(ps);
    //...
```

Exemplo

```
void inicializaMatriz0(float *m)
{
    for (int i = 1; i <= DIM*DIM; i++)
    {
        *m = 0;
        m++;
    }
}

void inicializaMatrizi(float *m)
{
    for (int i = 1; i <= DIM*DIM; i++)
    {
        *m = i;
        m++;
    }
}
```

Exemplo

```
float * soma(float *p1, float *p2)
{
    float *p = NULL, *p0 = NULL;
    p0 = (float *)malloc(sizeof(float)*DIM*DIM);

    *p0 = 0;
    p = p0;

    for(int i = 1; i <= DIM*DIM; i++)
    {
        *p = *p1 + *p2;
        p++;
        p1++;
        p2++;
    }

    return p0;
}
```

Exemplo

```
void mostraMatriz (float *m)
{
    for (int i = 1; i <= DIM*DIM; i++)
    {
        printf("elemento %i = %.2f\n", i, *m);
        m++;
    }
}
```



Exemplo

```
int v[] = {10,20,30,40,50};
int * pv;
```

```
pv = &v[0];
printf("%i\n",*pv);
++pv;
printf("%i\n",*pv);
```

```
int * ptab[10]; //tabela com 10 ponteiros para inteiros
ptab[0] = pv;
printf("%i\n",*ptab[0]);
```

Exemplo

```
/*...*/
struct dados_aluno
{
    char curso[5];
    int numero;
    char nome[50];
    float media;
};
```

```
typedef struct dados_aluno aluno;
```

$(*nome\_ponteiro).campo$   
 $\Leftrightarrow$   
 $nome\_ponteiro->campo$

```
int main ()
{
```

```
    aluno a,*p;
    p = &a;
```

```
    //insercao dados aluno
    strcpy((*p).nome,"ze");
    strcpy((*p).curso,"grsi");
    (*p).numero = 123;
    (*p).media = 12.0f;
```

ou

```
    strcpy(p->nome,"ze");
    strcpy(p->curso,"grsi");
    p->numero = 123;
    p->media = 12.0f;
```

```
    //apresentacao dados aluno
    printf("nome: %s\n",(*p).nome);
    printf("curso: %s\n",(*p).curso);
    printf("numero: %i\n",(*p).numero);
    printf("media: %f\n",(*p).media);
    return 0;
}
```

ou

```
    printf("nome: %s\n",p->nome);
    printf("curso: %s\n",p->curso);
    printf("numero: %i\n",p->numero);
    printf("media: %f\n",p->media);
```

### Exemplo

```
void le_aluno1(aluno *pa)
{
    printf("curso: ");
    gets(pa->curso);
    printf("nome: ");
    gets(pa->nome);
    printf("numero: ");
    scanf("%d",&pa->numero);
    printf("media: ");
    scanf("%f",&pa->media);
}
```

```
aluno * le_aluno2()
{
    aluno *pa;
    pa = (aluno*)malloc(sizeof(aluno));

    printf("curso: ");
    gets(pa->curso);
    printf("nome: ");
    gets(pa->nome);
    printf("numero: ");
    scanf("%d",&pa->numero);
    printf("media: ");
    scanf("%f",&pa->media);
    return pa;
}
```

No main:

```
aluno a, *p;
p = &a;
le_aluno1(p);

// ou apenas

aluno a;
le_aluno1(&a);
```

```
aluno *p;
p = le_aluno2();
```

### Exemplo (anterior com tabela)

```
/*...*/
#define TAM 15
/*...*/

int main()
{
    // usando uma tabela de alunos
    aluno t_alunos[TAM];
    int i;

    for(i=0;i<TAM;i++)
        le_aluno1(&t_alunos[i]);        // void le_aluno1(aluno *pa)

    for(i=0;i<TAM;i++)
        mostra_aluno(&t_alunos[i]);    //void mostra_aluno(aluno *pa)

    return 0;
}

void mostra_aluno(aluno *pa)
{
    printf("nome: %s\n",pa->nome);
    printf("curso: %s\n",pa->curso);
    printf("media: %f\n",pa->media);
}
```

Exemplo  
(anterior com  
ponteiro para  
tabela)

```
/*...*/  
int main()  
{  
    int n, i;  
    aluno * pt_alunos;  
  
    printf("numero de alunos: ");  
    scanf("%i",&n);  
  
    pt_alunos = (aluno*)malloc(n*sizeof(aluno));  
  
    // ponteiro pt_alunos guarda endereco de memoria que ocupa  
    // [numero de alunos x numero de bytes ocupados por um aluno] bytes  
    // necessario fazer casting porque malloc devolve ponteiro genérico  
    // se pt_alunos for NULL então não foi possível reservar memória  
  
    for(i = 0; i < n; i++)  
        le_aluno1(&pt_alunos[i]);  
  
    for(i = 0; i < n; i++)  
        mostra_aluno(&pt_alunos[i]);  
  
    free(pt_alunos);  
    // torna disponível o espaço apontado pelo ponteiro pt_alunos  
    return 0;  
}
```