

Vetores - Declaração e Inicialização

- **Declaração de um vetor:**

```
int meu_vetor[10];
```

- Reserva um espaço de memória para armazenar 10 valores inteiros no vetor chamado meu_vetor.

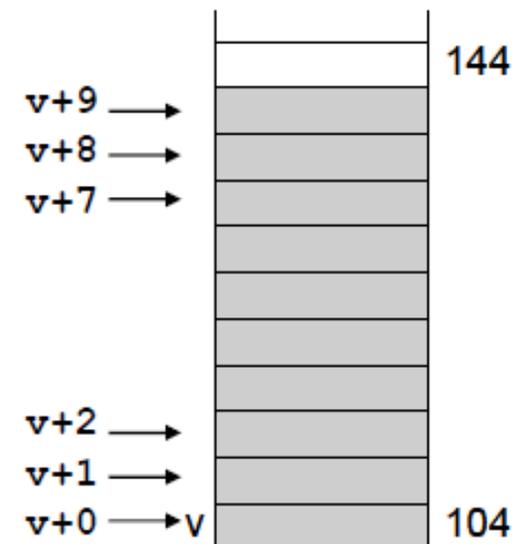
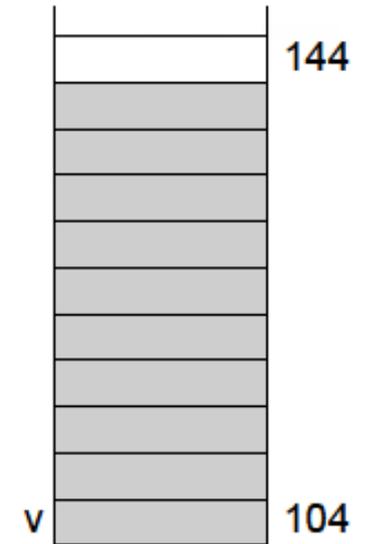
- **Inicialização de algumas posições do vetor meu_vetor:**

```
meu_vetor[0] = 5;  
meu_vetor[1] = 11;  
meu_vetor[4] = 0;  
meu_vetor[9] = 3;
```

5	11	?	?	0	?	?	?	?	3
<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>

Vetores Alocados em Memória

- Vetor é alocado em posições contíguas de memória;
 - Exemplo: `int v[10];`
- O nome do vetor aponta para endereço inicial;
- C permite aritmética de ponteiros:
 - Exemplo:
 - `v+0` : primeiro elemento de `v` ...
 - `v+9` : último elemento de `v`
 - Exemplo:
 - `&v[i]` é equivalente a `(v+i)`
 - `*(v+i)` é equivalente a `v[i]`



Aritmética de Ponteiros

- Qual é o resultado da execução desse programa?

```
int main(void)
{
    int vet[6] = {1, 2, 3, 4, 5, 6};
    printf("Valor1: %d\n", vet);
    printf("Valor2: %d\n", *vet);
    printf("Valor3: %d\n", *(vet + 2));
    return 0;
}
```

Resultado:

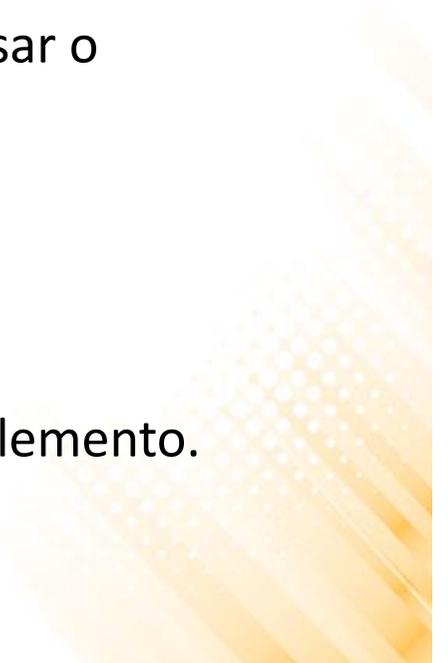
Valor1: 3997056 -> endereço de memória de vet[0]

Valor2: 1

Valor3: 3

Vetores Passados para Funções

- A linguagem C permite passar vetores como parâmetros para funções:
 - consiste em passar o endereço da primeira posição do vetor por um parâmetro do tipo ponteiro;
 - “passar um vetor para uma função” é equivalente a “passar o endereço inicial do vetor”;
 - os elementos do vetor não são copiados para a função;
 - o argumento copiado é apenas o endereço do primeiro elemento.



Vetores e Funções - Exemplo

```
/* Cálculo da média e da variância de 10 números reais */
#include <stdio.h>

/* Função para cálculo da média */
float media(int n, float *v)
{
    int i;
    float s = 0.0;
    for (i=0; i<n; i++)
    {
        s += v[i];
    }
    return s/n;
}
```

Parâmetro do tipo ponteiro
para float.

É equivalente a float v[].

[continua...]

```
/* Função para cálculo da variância */
float variancia (int n, float *v, float m)
{
    int i;
    float s = 0.0;
    for (i=0; i<n; i++)
        s += (v[i] - m) * (v[i] - m);
    return s/n;
}

int main (void)
{
    float v[10];
    float med, var;
    int i;
    for (i=0; i<10; i++)
        scanf("%f", &v[i]);
    med = media(10, v);
    var = variancia(10, v, med);
    printf("Media = %f Variância = %f \n", med, var);
    return 0;
}
```

Endereço da primeira
posição do vetor passada
por parâmetro.

Vetores e Funções – Exemplo 2

```
/* Incrementa elementos de um vetor */
#include <stdio.h>

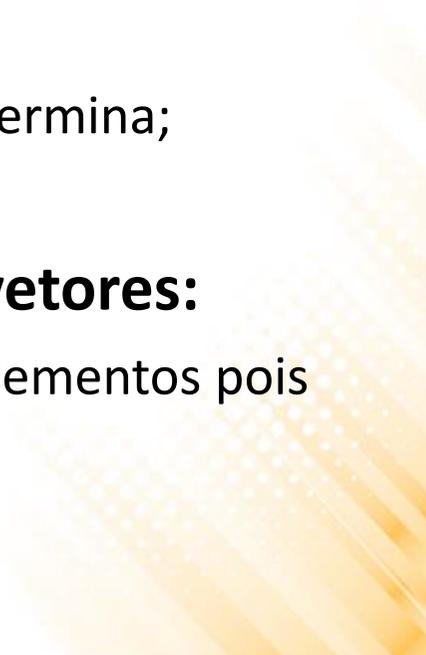
void incr_vetor(int n, int *v)
{
    int i;
    for (i=0; i<n; i++)
        v[i]++;
}

int main(void)
{
    int a[] = {1, 3, 5};
    incr_vetor(3,a);
    printf("%d %d %d \n", a[0], a[1], a[2]);
    return 0;
}
```

A função pode alterar os valores dos elementos do vetor, pois recebe o endereço do primeiro elemento do vetor (e não os elementos propriamente ditos)

Uso da Memória

- **Uso por variáveis globais (e estáticas):**
 - espaço reservado para uma variável global fica disponível enquanto o programa estiver sendo executado;
- **Uso por variáveis locais:**
 - espaço disponível apenas enquanto a função que declarou a variável está sendo executada;
 - liberado para outros usos quando a execução da função termina;
- **Variáveis globais ou locais podem ser simples ou vetores:**
 - para vetor, é necessário informar o número máximo de elementos pois o compilador precisa calcular o espaço a ser reservado;



Uso da Memória

- **Alocação dinâmica:**
 - espaço de memória é requisitado em tempo de execução;
 - espaço permanece reservado até que seja explicitamente liberado;
 - depois de liberado, o espaço ficará disponível para outros usos e não poderá mais ser acessado;
 - o espaço alocado e não liberado explicitamente será automaticamente liberado ao final da execução;



Uso da Memória

- **Memória estática:**
 - código do programa
 - variáveis globais
 - variáveis estáticas
- **Memória dinâmica:**
 - variáveis alocadas dinamicamente
 - memória livre
 - variáveis locais

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	Memória livre
	Variáveis locais (Pilha de execução)

Uso da Memória

- **Alocação dinâmica de memória:**
 - usa a memória livre;
 - se o espaço de memória livre for menor que o espaço requisitado a alocação não é feita . O programa deve tratar tal situação;
- **Pilha de execução:**
 - utilizada para alocar memória quando ocorrer a chamada de uma função:
 - sistema reserva o espaço para as variáveis locais da função;
 - quando a função termina o espaço é liberado (desempilhado);
 - se a pilha tentar crescer mais do que o espaço disponível o programa será abortado.

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	Memória livre
	Variáveis locais (Pilha de execução)

Alocação Dinâmica

- A biblioteca padrão `<stdlib.h>` possui um conjunto de funções para tratar a alocação dinâmica de memória:

```
void * malloc(int num_bytes);
```

```
int sizeof(type);
```

```
void free(void *p);
```



Alocação Dinâmica

```
void * malloc(int num_bytes);
```

- Recebe como parâmetro o número de bytes que se deseja alocar;
- Retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre:
 - o ponteiro genérico é representado por `void*`
 - ele é pode ser convertido para o tipo apropriado
 - deve-se convertê-lo explicitamente
- Retorna um endereço nulo (NULL) se não houver espaço livre.

Alocação Dinâmica

```
int sizeof(type);
```

- Retorna o número de bytes necessários para representar valores de um determinado tipo.
- Exemplos:
 - char (1 byte)
 - int (4 bytes)
 - double (8 bytes)

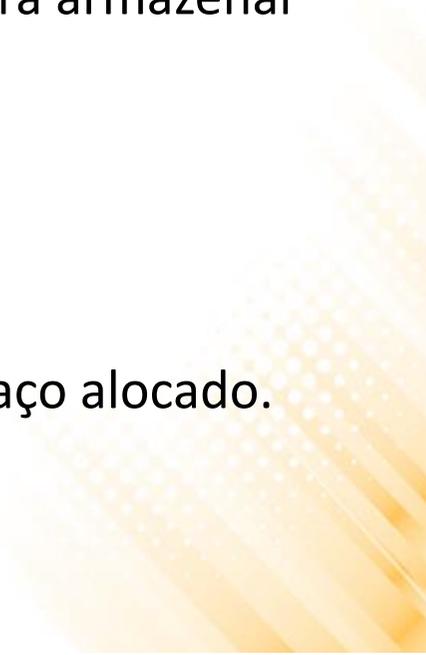
```
void free(void *p);
```

- Recebe como parâmetro o ponteiro para a área de memória a ser liberada (alocada dinamicamente em um passo anterior)

Alocação Dinâmica - Exemplo

- Alocação dinâmica de um vetor de inteiros com 10 elementos:

```
int *v;  
v = (int *) malloc(10 * sizeof(int));
```

- O malloc retorna o endereço do início da área alocada para armazenar o vetor de inteiros;
 - O ponteiro é convertido para um ponteiro de inteiro;
 - O ponteiro de inteiro *v recebe o endereço inicial do espaço alocado.
- 

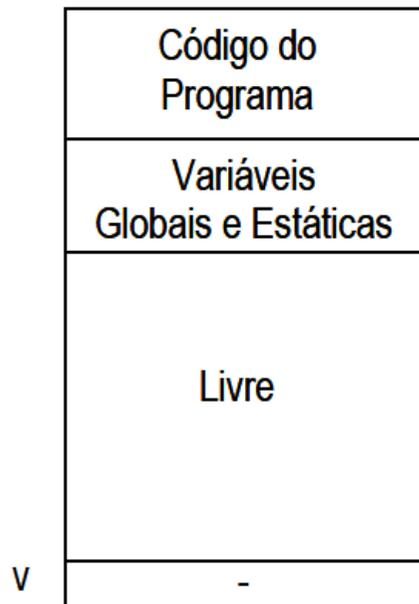
Alocação Dinâmica - Exemplo

```
int *v;  
v = (int *) malloc(10 * sizeof(int));
```

1 - Declaração:

```
int *v
```

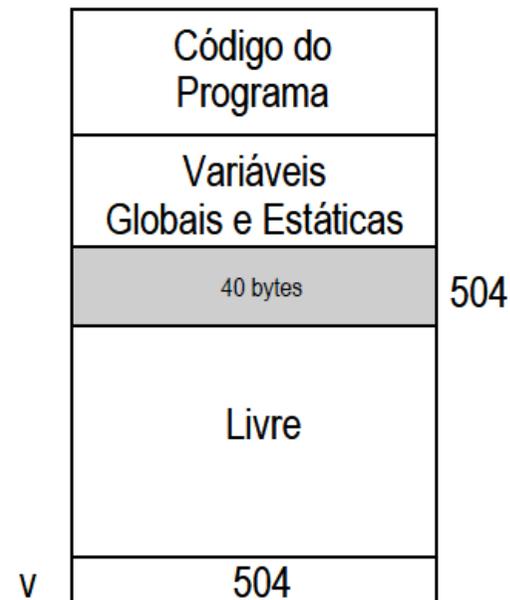
Abre-se espaço na pilha para o ponteiro (variável local)



2 - Comando:

```
v = (int *) malloc(10 * sizeof(int))
```

Reserva espaço de memória da área livre e atribui endereço à variável



Alocação Dinâmica - Exemplo

```
int *v;  
v = (int *) malloc(10 * sizeof(int));
```

- `v` armazena endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros;
- `v` pode ser tratado como um vetor declarado estaticamente
 - `v` aponta para o início da área alocada;
 - `v[0]` acessa o espaço para o primeiro elemento;
 - `v[1]` acessa o segundo;
 - até `v[9]`



Alocação Dinâmica - Exemplo

- Tratamento de **erro** após chamada de `malloc`
 - imprime mensagem de erro;
 - aborta o programa (com a função `exit`);

```
int *v;
v = (int*) malloc(10 * sizeof(int));

if (v == NULL)
{
    printf("Memoria insuficiente.\n");
    exit(1); /* aborta o programa - retorna 1 */
}

...
free(v);
```

Alocação Dinâmica - Exemplo

```
#include <stdlib.h>

int main (void)
{
    float *v;
    float med, var;
    int i, n;
    printf("Entre n e depois os valores\n");
    scanf("%d",&n);
    v = (float *) malloc(n * sizeof(float));
    if (v == NULL)
    {
        printf("Falta memoria\n");
        exit(1);
    }
}
```

[continua...]

Alocação Dinâmica - Exemplo

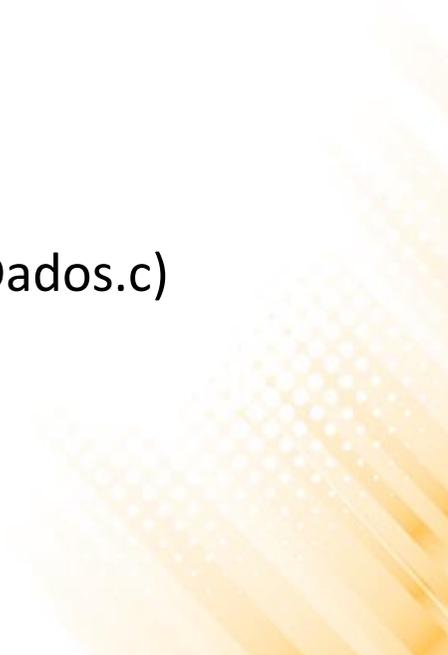
```
for (i = 0; i < n; i++)
{
    scanf("%f", &v[i]);
}

med = media(n, v);
var = variancia(n, v, med);
printf ("Media = %f Variancia = %f \n", med, var);

free(v);
return 0;
}
```

Alocação Dinâmica – Exemplo 2

- **Exemplo:** armazenar as notas dos alunos de uma turma em um vetor e em seguida calcular a média da turma.
- **Podemos dividir o programa em funções:**
 - Uma função para ler os valores e armazená-los em um vetor;
 - Uma função para calcular a média;
- **Podemos dividir o programa em módulos:**
 - Módulo ManipulaDados (ManipulaDados.h e ManipulaDados.c)
 - Principal (Principal.c)



ManipulaDados.h

```
void ler_dados(float *vet, int num);  
  
float calcula_media(float *vet, int num);
```

ManipulaDados.c

```
#include <stdio.h>  
#include "ManipulaDados.h"  
  
void ler_dados(float *vet, int num)  
{  
    int i;  
    for(i=0;i<num;i++)  
    {  
        printf("Entre com o valor %d: ", i+1);  
        scanf("%f", &vet[i]);  
    }  
}  
  
float calcula_media(float *vet, int num)  
{  
    float soma = 0.0;  
    int i;  
    for(i=0;i<num;i++)  
        soma = soma + vet[i];  
    return soma/num;  
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "ManipulaDados.h"

int main (void)
{
    float *notas;
    int alunos;
    printf("Digite o total de alunos da turma: ");
    scanf("%d", &alunos);

    notas = (float *) malloc(alunos * sizeof(float));
    if (v == NULL){
        printf("Memoria insuficiente.\n");
        exit(1);
    }

    ler_dados(notas, alunos);
    printf("Media: %.2f\n.", calcula_media(notas, alunos));

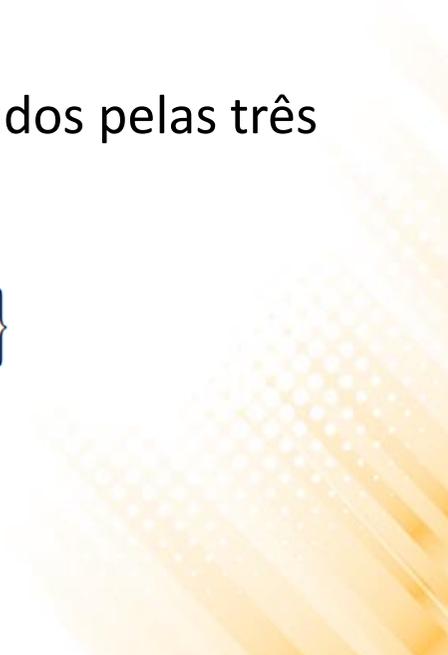
    free(notas);

    return 0;
}
```

Vetores Locais e Funções

- **Área de memória de uma variável local:**
 - só existe enquanto a função que declara a variável estiver sendo executada;
 - requer cuidado quando da utilização de vetores locais dentro de funções.
- **Exemplo:**
 - produto vetorial de dois vetores \mathbf{u} e \mathbf{v} em 3D, representados pelas três componentes x , y , e z :

$$\mathbf{u} \times \mathbf{v} = \left\{ u_y v_z - v_y u_z, \quad u_z v_x - v_z u_x, \quad u_x v_y - v_x u_y \right\}$$



Vetor Declarado Localmente

```
float* prod_vetorial(float* u, float* v)
{
    float p[3];
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p;
}
```

- Variável `p` declarada localmente:
 - área de memória que a variável `p` ocupa deixa de ser válida quando a função `prod_vetorial` termina;
 - função que chama `prod_vetorial` não pode acessar a área apontada pelo valor retornado.

Vetor Alocado Dinamicamente

```
float* prod_vetorial(float* u, float* v)
{
    float *p = (float*) malloc(3 * sizeof(float));
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p;
}
```

- A variável `p` recebe endereço inicial da área alocada dinamicamente:
 - área de memória para a qual a variável `p` aponta permanece válida mesmo após o término da função `prod_vetorial`;
 - a variável `p` será destruída após o término da função `prod_vetorial`;

Vetor Alocado Dinamicamente

```
float* prod_vetorial(float* u, float* v)
{
    float *p = (float*) malloc(3 * sizeof(float));
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p;
}
```

- A variável `p` recebe endereço inicial da área alocada dinamicamente:
 - a função que chama `prod_vetorial` pode acessar a área apontada pelo valor retornado;
 - **problema:** alocação dinâmica para cada chamada da função:
 - ineficiente do ponto de vista computacional;
 - requer que a função que chama seja responsável pela liberação do espaço alocado.

Vetor Alocado Previamente

```
void prod_vetorial(float* u, float* v, float* p)
{
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
}
```

- Espaço de memória para o resultado passado pela função que chama:
 - função `prod_vetorial` recebe três vetores:
 - dois vetores com dados de entrada;
 - um vetor para armazenar o resultado;
 - **solução mais adequada**, pois não envolve alocação dinâmica

Exercício 1

- a) Crie uma função para separar as notas baixas de uma turma em um novo vetor. Uma nota é considerada baixa se ela estiver abaixo de 5.0.
- A função deve receber como parâmetro um vetor contendo as notas da turma, o número total de notas e um ponteiro de inteiro onde deverá ser especificado o número de notas baixas encontradas;
 - A função deve retorna um ponteiro para um vetor alocado dinamicamente contendo todas as notas baixas da turma;

```
float *notas_baixas(float *notas, int n, int *totalbaixas);
```

Exercício 1a - Solução

```
float *notas_baixas(float *vnotas, int n, int *totalbaixas)
{
    float *notasbaixas;
    int i, j;

    *totalbaixas = 0;
    for (i = 0; i < n; i++)
    {
        if (vnotas[i] < 5)
            (*totalbaixas)++;
    }

    if (*totalbaixas == 0)
        return NULL;
}
```

ponteiro que guardará o endereço do vetor das notas baixas

conta quantas notas abaixo de 5 existem

se não há notas abaixo de 5, retorna NULL sem criar um vetor

```
notasbaixas = (float*) malloc(*totalbaixas * sizeof(float));
```

```
if (notasbaixas == NULL)  
    return NULL;
```

```
j = 0;  
for (i = 0; i < n; i++)  
{  
    if (vnotas[i] < 5)  
    {  
        notasbaixas[j] = vnotas[i];  
        j++;  
    }  
}
```

```
return notasbaixas;
```

```
}
```

aloca memória para o novo vetor

verifica se foi possível alocar a memória

preenche o novo vetor com as notas baixas

retorna o endereço do novo vetor

Exercício 1

- b) Crie um programa que utilize a função `notas_baixas` para encontrar e listar as notas baixas de uma turma cujo número de alunos e as notas de cada aluno devem ser informados pelo usuário.



Exercício 1b - Solução

```
int main(void)
{
    float *vet_notas;
    float *vet_baixas;
    int i, n, total_baixas;
```

ponteiros para guardar os
endereços dos vetores de
notas e notas baixas

```
printf("Digite o numero de notas: ");
scanf("%d", &n);
vet_notas = (float*) malloc(n * sizeof(float));
```

aloca memória para o
vetor de notas

```
if (vet_notas == NULL)
{
    printf("Erro: Falta de memoria!\n");
    exit(1);
}
```

verifica se foi possível
alocar a memória

```
for (i = 0; i < n; i++)  
{  
    printf("Digite a nota: ");  
    scanf("%f", &vet_notas[i]);  
}
```

**lê as notas e as armazena
no vetor**

```
vet_baixas = notas_baixas(vet_notas, n, &total_baixas);
```

**obtêm o vetor de
notas baixas**

```
if (vet_baixas == NULL)  
    printf("Não existem notas baixas!\n");
```

```
else
```

```
{
```

```
    printf("Notas baixas: ");
```

```
    for (i = 0; i < total_baixas; i++)
```

```
        printf("%f ", vet_baixas[i]);
```

**imprime as notas
baixas encontradas**

```
}
```

```
if (vet_baixas != NULL)
```

```
    free(vet_baixas);
```

```
free(vet_notas);
```

**libera a memória alocada
dinamicamente**

```
return 0;
```

```
}
```

Resumo

- **Comandos e funções para gerência de memória:**

- `sizeof` retorna o número de bytes ocupado por um tipo;

- `malloc` recebe o número de bytes que se deseja alocar:

- retorna um ponteiro para o endereço inicial, ou
 - retorna um endereço nulo (NULL)

- `free` recebe o ponteiro da memória a ser liberada.

