

Algoritmos e Programação II

Prof. Viviane Bonadia dos Santos

Aula baseada nos tópicos de aula e slides
do Prof. Tiago A. Almeida

Alocação dinâmica

- Até agora tínhamos que declarar **todas** as variáveis que íamos usar no programa, para que o computador pudesse alocar memória para elas.
- O espaço destinado à estas variáveis é denominado **memória estática**, pois seus tamanhos foram **definidos no código** fonte e foram **fixados durante a compilação**.
- **Portanto**, o tamanho dessa memória não pode ser modificado durante a execução!

Alocação dinâmica

- Mas existe um modo de definir uma variável **enquanto** o programa roda?
 - A resposta é: não exatamente
- **Não há como declarar** a variável enquanto o programa roda. O que dá para fazer é **alocar memória** para uma variável enquanto o programa roda
 - **E qual a vantagem disso?**
 - Poupa memória (em casos específicos)
 - Mas, isto tem um preço (como tudo na computação) - gasta mais processamento

Alocação dinâmica

Nesta aula estudaremos os mecanismos da linguagem C para escrever código que gerencia a memória de forma **dinâmica**, isto é, durante a execução do programa. O programa poderá **aumentar** ou **diminuir** a quantidade de memória em uso a cada instante.

Alocação dinâmica: uso

- Apesar de poder ser usada com todos os tipos de dados, alocação dinâmica é mais frequentemente empregada para **manipulação de strings, vetores e registros**
- Alocação dinâmica de registros é extremamente utilizada, uma vez que, é possível **conectar os registros** em forma de **listas, árvores ou outra estrutura de dados.**

Alocação dinâmica

Como podemos alocar memória de forma **dinâmica**??

Alocação dinâmica

Como podemos alocar memória de forma **dinâmica**??

- Tipicamente, os **ponteiros** serão o veículo através do qual as limitações da memória estática serão contornadas.

Alocação dinâmica

Como podemos alocar memória de forma **dinâmica**??

- Tipicamente, os **ponteiros** serão o veículo através do qual as limitações da memória estática serão contornadas.
- O programa precisará invocar **funções especiais** para solicitar mais espaço na memória.

Alocação dinâmica

Como podemos alocar memória de forma **dinâmica**??

- Tipicamente, os **ponteiros** serão o veículo através do qual as limitações da memória estática serão contornadas.
- O programa precisará invocar **funções especiais** para solicitar mais espaço na memória.
- Dentro de um espaço que foi alocado como memória dinâmica, o programa poderá armazenar qualquer dado (números inteiros ou fracionários, vetores, estruturas, etc)
- A qualquer instante, o programa pode também **liberar** partes da memória que requisitou dinamicamente, se não for mais usar aqueles espaços.

Alocação dinâmica

Passo a passo da alocação dinâmica

1. Solicitar memória dinâmica.

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:

???

Alocação dinâmica

Passo a passo da alocação dinâmica

1. Solicitar memória dinâmica.

2. Armazenar o endereço da memória obtida em uma variável do tipo apontador.


Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:

???



Alocação dinâmica

Passo a passo da alocação dinâmica

1. Solicitar memória dinâmica.

2. Armazenar o endereço da memória obtida em uma variável do tipo apontador.

3. Armazenar dados.

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:

xyz



Alocação dinâmica

Passo a passo da alocação dinâmica

1. Solicitar memória dinâmica.

2. Armazenar o endereço da memória obtida em uma variável do tipo apontador.

3. Armazenar dados.

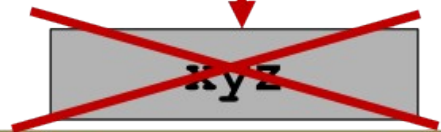
4. Liberar memória obtida.

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:



Alocação dinâmica: funções

- Em linguagem C há três funções declaradas na biblioteca **<stdlib.h>** que podem ser empregadas para alocar memória dinamicamente. Elas são:

- **malloc** – Aloca um bloco de memória mas não o inicializa (função mais empregada);

```
void *malloc(size_t size);
```

- **calloc** - Aloca um bloco de memória e o inicializa (menos eficiente que malloc);

```
void *calloc(size_t nmem, size_t size);
```

- **realloc** – Redimensiona um bloco de memória previamente alocado.

```
void *realloc(void *ptr, size_t size);
```

Alocação dinâmica: malloc

- A função **malloc** (*memory allocation*) aloca um determinado número de bytes na memória, retornando um ponteiro para o primeiro byte alocado, ou **NULL** caso não tenha conseguido alocar memória.
- A função **free**, por outro lado, libera o espaço alocado.

```
p = malloc(100000);  
if (p == NULL) {  
    /* tratamento para falha de alocação */  
}  
free(p);
```

```
if ((p = malloc(100000)) == NULL) {  
    /* tratamento para falha de alocação */  
}
```

Alocação dinâmica: malloc

- A função **malloc** (*memory allocation*) aloca um determinado número de bytes na memória, retornando um ponteiro para o primeiro byte alocado, ou **NULL** se não houver memória disponível para alocar.

- A função **free**, por outro lado,

```
p = malloc(10000);  
if (p == NULL) {  
    /* tratamento para falha  
}  
free(p);
```

Reserva 10000 bytes e guarda o endereço inicial do trecho de memória obtido no apontador **p**.

```
if ((p = malloc(10000)) == NULL) {  
    /* tratamento para falha de alocação */  
}
```


Alocação dinâmica: malloc

- O endereço retornado por malloc é totalmente genérico e **não possui um tipo especificado.**

Alocação dinâmica: malloc

- O endereço retornado por malloc é totalmente genérico e **não possui um tipo especificado**.
- Quando malloc aloca a memória, ela não faz ideia do que será posto lá: se é int, char, float ou o que for. Então, ela retorna um ponteiro genérico (**void ***). Isso é possível porque ponteiros são endereços de memória e, como tal, possuem sempre o mesmo tamanho, não importando o tipo para o qual eles apontam.

Alocação dinâmica: malloc

- O endereço retornado por malloc é totalmente genérico e **não possui um tipo especificado**.
- Quando malloc aloca a memória, ela não faz ideia do que será posto lá: se é `int`, `char`, `float` ou o que for. Então, ela retorna um ponteiro genérico (`void *`). Isso é possível porque ponteiros são endereços de memória e, como tal, possuem sempre o mesmo tamanho, não importando o tipo para o qual eles apontam.
- Contudo, C precisa que o ponteiro tenha um tipo, para poder executar operações de aritmética de ponteiros. Por isso **temos que fazer um `cast` no retorno de malloc** para o tipo de ponteiro.

Alocação dinâmica: malloc

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

int main(void)
{
    char *c; /* ponteiro para o espaço alocado */

    c = (char *) malloc(1); /* aloco um único byte na memória */

    if (c == NULL) { /* testa se conseguiu alocar.
                       Equivalente a "if (!c)" */
        printf("Não conseguiu alocar a memória\n");
        exit(1);
    }

    *c = 'd'; /* carrego um valor na região
               de memória alocada */
    printf("%c\n", *c); /* escrevo este valor */
    free(c); /* libero a memória alocada */
    return 0;
}
```

Alocação dinâmica: malloc

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

int main(void)
{
    char *c; /* ponteiro para o espaço alocado */

    c = (char *) malloc(1); /* aloco um único byte na memória */

    if (c == NULL) { /* teste se alocou
        Equivalente a: if (!c)
        printf("Não conseguiu alocar memória\n");
        exit(1);
    }

    *c = 'd'; /* carrego um valor na região
               de memória alocada */
    printf("%c\n", *c); /* escrevo este valor */
    free(c); /* libero a memória alocada */
    return 0;
}
```

“Força” a saída do malloc a ser char.

Alocação dinâmica: malloc

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

int main(void)
{
    char *c; /* ponteiro para o e
    c = (char *) malloc(1); /* alo
    if (c == NULL) { /* Equ
        printf("Não conseguiu a
        exit(1);
    }

    *c = 'd'; /* carrego um valor na região
                de memória alocada */
    printf("%c\n", *c); /* escrevo este valor */
    free(c); /* libero a memória alocada */
    return 0;
}
```

No caso de **c** ser NULL, é impresso uma mensagem de erro o programa é encerrado. */

Alocação dinâmica: malloc

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

```
int main(void)
{
```

```
    char *c; /* pont
```

```
    c = (char *) mal
```

```
    if (c == NULL) {
```

```
        printf("Não c
        exit(1);
```

```
    }
```

```
    *c = 'd'; /* carrego um valor na região
               de memória alocada */
```

```
    printf("%c\n", *c); /* escrevo este valor */
```

```
    free(c); /* libero a memória alocada */
```

```
    return 0;
```

```
}
```

Se c não for NULL, então ele contém o endereço na memória onde cabe um caractere e pode ser usado normalmente.

Alocação dinâmica: malloc

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

int main(void)
{
    char *c; /* ponteiro para o espaço alocado */

    c = (char *) malloc(1); /* aloca 1 byte na memória */

    if (c == NULL) { /*
        printf("Não conseguiu alocar memória\n");
        exit(1);
    }

    *c = 'd'; /* escrevo este valor na memória */

    printf("%c\n", *c); /* escrevo este valor */
    free(c); /* libero a memória alocada */
    return 0;
}
```

Antes do encerramento do programa, é necessário liberar a memória alocada, usando a função **free**.

Alocação dinâmica: algumas observações...

- É importante solicitar o número correto de bytes com `malloc`, para **evitar desperdício!!**

Alocação dinâmica: algumas observações...

- É importante solicitar o número correto de bytes com malloc, para **evitar desperdício!!**
- O apontador retornado pela função malloc é a única forma de acessar o novo espaço de memória dinâmica. Se, durante a execução do programa, este **apontador se perder** em decorrência de um erro de programação, este espaço de memória dinâmica permanecerá alocado na memória até o final do programa, mas será **impossível recuperar sua posição**.

Alocação dinâmica: malloc

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

int main(void){
    double *n; /* ponteiro para o espaço a ser alocado */

    n = (double *) malloc(sizeof(double));

    if (!n){ /* testa a alocação */
        printf("Não conseguiu alocar a memória\n");
        exit(1);
    }

    /* usa o double */
    ...

    /* libera a memória alocada */
    free(n);

    /* o programa continua */
    ...
    return (0);
}
```

Alocação dinâmica: malloc

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

int main(void){
    double *n; /* ponteiro para o espaço a ser alocado */

    n = (double *) malloc(sizeof(double));

    if (!n){ /* testa a alocação */
        printf("Não conseguiu alocar a memória\n");
        exit(1);
    }

    /* usa o double */
    ...

    /* libera a memória alocada */
    free(n);

    /* o programa continua */
    ...
    return (0);
}
```

A função **sizeof** retorna o número de bytes necessários para armazenar dados deste tipo.

Alocação dinâmica: string

- *Strings* são armazenadas em vetores de caracteres e, é difícil prever o seu tamanho antecipadamente
- Para alocar espaço para uma *string* com *n* caracteres:

```
char *p;  
p = (char *) malloc(n + 1);
```

+1 para o caractere nulo

- *p* aponta para o primeiro caractere da *string*

Alocação dinâmica: string

- Com alocação dinâmica é possível escrever funções que retornam um ponteiro para uma “nova” *string* (uma *string* que não existia antes da chamada da função).
 - **Exemplo:** fazer uma função que concatene duas *strings* sem alterá-las

```
char *concat(const char *s1, const char *s2) {  
    char *result;  
  
    result = (char *) malloc (strlen(s1) + strlen(s2) + 1);  
    if (result == NULL) {  
        printf("Erro: malloc falhou em concat\n");  
    }  
  
    strcpy(result, s1);  
    strcpy(result, s2);  
  
    return (result);  
}
```

Alocação dinâmica

- Alocação dinâmica deve ser usada com cuidado!
Quando a variável não for mais necessária é importante liberá-la da memória usando a função **free**.
- Uma vez liberado, é impossível (ou muito perigoso) acessar novamente este espaço de memória dinâmica.
Após a execução da função free, todos os apontadores referentes a este espaço tornam-se **potencialmente inválidos** e o melhor é assumir que seria um erro tentar ler ou atribuir através deles

Alocação dinâmica: vetores

- Suponha que queremos tirar a média de n notas.
- Pedimos o valor de n e então as n notas, certo?
- E como guardaríamos?
 - Até agora, tínhamos que declarar um mega vetor e torcer para que n não fosse maior que nosso vetor.
 - Mas com **alocação dinâmica...**

Alocação dinâmica: vetores

- Primeiro, é preciso declarar um ponteiro para float:

```
float *v;
```

Alocação dinâmica: vetores

- Primeiro, é preciso declarar um ponteiro para float:

```
float *v;
```

- Em seguida, tudo que temos que fazer para transformá-lo em um vetor é apontá-lo para um grupo sequencial de floats na memória:

```
v = (float *)malloc(n * sizeof(float));
```

Alocação dinâmica: vetores

- Primeiro, é preciso declarar um ponteiro para float:

```
float *v;
```

- Em seguida, tudo que temos que fazer para transformá-lo em um vetor é apontá-lo para um grupo sequencial de floats na memória:

```
v = (float *)malloc(n * sizeof(float));
```

- Note que pegamos o tamanho de um float e multiplicamos pelo número de floats (n) que o vetor conterà, ou seja, calculamos o tamanho em bytes de n floats.

Alocação dinâmica: vetores

- Primeiro, é preciso declarar um ponteiro para float:

```
float *v;
```

- Em seguida, tudo que temos que fazer para transformá-lo em um vetor é apontá-lo para um grupo sequencial de floats na memória:

```
v = (float *)malloc(n * sizeof(float));
```

- Note que pegamos o tamanho de um float e multiplicamos pelo número de floats (n) que o vetor conterá, ou seja, calculamos o tamanho em bytes de n floats.
- Quando vimos ponteiros, também vimos que ao fazermos “ $v[i]$ ” estamos fazendo, na verdade, “ $*(v+i)$ ”.
- Portanto, podemos tratar nosso ponteiro como um vetor comum

Alocação dinâmica: vetores

```
int main(void) {  
  
    float *v; /* vetor de notas */  
    int i, n; /* contador e número de elementos do vetor */  
  
    printf("Qual o número de notas? ");  
    scanf("%d",&n);  
  
    /* aloco espaço suficiente para o vetor de n notas */  
    v = (float *) malloc(n * sizeof(float));  
  
    if (v == NULL){  
        printf("Não foi possível alocar o vetor\n");  
        exit(1);  
    }  
  
    for (i=0; i<n; i++) /* carrego o vetor de notas */  
        v[i] = nota;  
    for (i=0; i<n; i++) /* imprimo o vetor */  
        printf("Nota: %f\n", v[i]);  
  
    free(v); /* desaloco o vetor */  
    return 0;  
}
```

Alocação dinâmica: registros

- **O procedimento é igual!**

- Basta trocar o tipo da variável pelo registro

- **Exemplo**

- Programa para armazenar e exibir um círculo de centro (x,y) e raio r

Alocação dinâmica: registros

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

struct s_pos { int x; int y; };
struct s_circulo { struct s_pos c; /* centro do círculo */
                  float r; /* seu raio */};

int main(void){
    struct s_circulo *p; /* o ponteiro para o espaço alocado */

    /* aloco espaço para um struct s_circulo */
    p = (struct s_circulo *) malloc(sizeof(struct s_circulo));

    if (p == NULL){
        printf("Não conseguiu alocar a memória\n");
        exit(1);
    }

    p->c.x = 2; // ou ainda (*p).c.x = 2;
    p->c.y = 4; // ou ainda (*p).c.y = 4;
    p->r = 3.2; // ou ainda (*p).r = 3.2;
    printf("x = %d, y = %d\n", p->c.x, p->c.y);
    printf("r = %f\n", p->r);
    free(p);
    return 0;
}
```

Alocação dinâmica: registros

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

struct s_pos { int x; int y; };
struct s_circulo { struct s_pos c; /* centro do círculo */
                  float r; /* seu raio */};

int main(void){
    struct s_circulo *p; /* o ponteiro para o espaço alocado */

    /* aloco espaço para um struct s_circulo */
    p = (struct s_circulo *) malloc(sizeof(struct s_circulo));

    if (p == NULL){
        printf("Não conseguiu alocar a memória\n");
        exit(1);
    }

    p->c.x = 2; // ou ainda (*p).c.x = 2;
    p->c.y = 4; // ou ainda (*p).c.y = 4;
    p->r = 3.2; // ou ainda (*p).r = 3.2;
    printf("x = %d, y = %d\n", p->c.x, p->c.y);
    printf("r = %f\n", p->r);
    free(p);
    return 0;
}
```

Reserva
espaço
suficiente
para
armazenar
uma variável
do tipo
s_circulo.

Alocação dinâmica: realloc

- A função `realloc` faz um bloco já alocado crescer ou diminuir, preservando o conteúdo já existente:

```
int *x, i;

x = (int *) malloc(4000*sizeof(int));

if (x == NULL){
    printf("Não foi possível alocar o vetor\n");
    exit(1);
}

for(i=0; i<4000; i++)
    x[i] = rand()%100;

x = (int *) realloc(x, 8000*sizeof(int));

x = (int *) realloc(x, 2000*sizeof(int));

free(x);
```



Alocação dinâmica: realloc

- Muitos erram quando utilizam a `realloc`
 - Isso acontece por que na maioria das vezes o programador esquece de “pegar” o retorno da função
 - O `realloc` tenta realocar a quantidade de memória pedida na sequência da já alocada, se não consegue, ele aloca uma nova área e retorna o ponteiro para essa área, liberando a área previamente alocada, e é aí que ocorre o erro

```
char *pointer;  
  
pointer = (char *) malloc(10 * sizeof(char));  
  
realloc(pointer, 20 * sizeof(char)); /* ERRADO */  
  
pointer = (char *)realloc(pointer, 20 * sizeof(char)); /* CERTO */
```

Alocação dinâmica: calloc

- A função calloc é parecida com a função malloc. Exceto pelo fato de que ela inicializa os elementos alocados com zeros.

```
void *calloc(size_t nmemb, size_t size);
```

- nmemb – quantidade de elementos a ser alocada
- size – tamanho de cada elemento

```
int *vetor, i;
```

```
scanf ("%d",&i); /* tamanho do vetor */
```

```
vetor = (int*) calloc(i,sizeof(int)); /*aloca e inicializa/*
```

Malloc X Calloc

Malloc

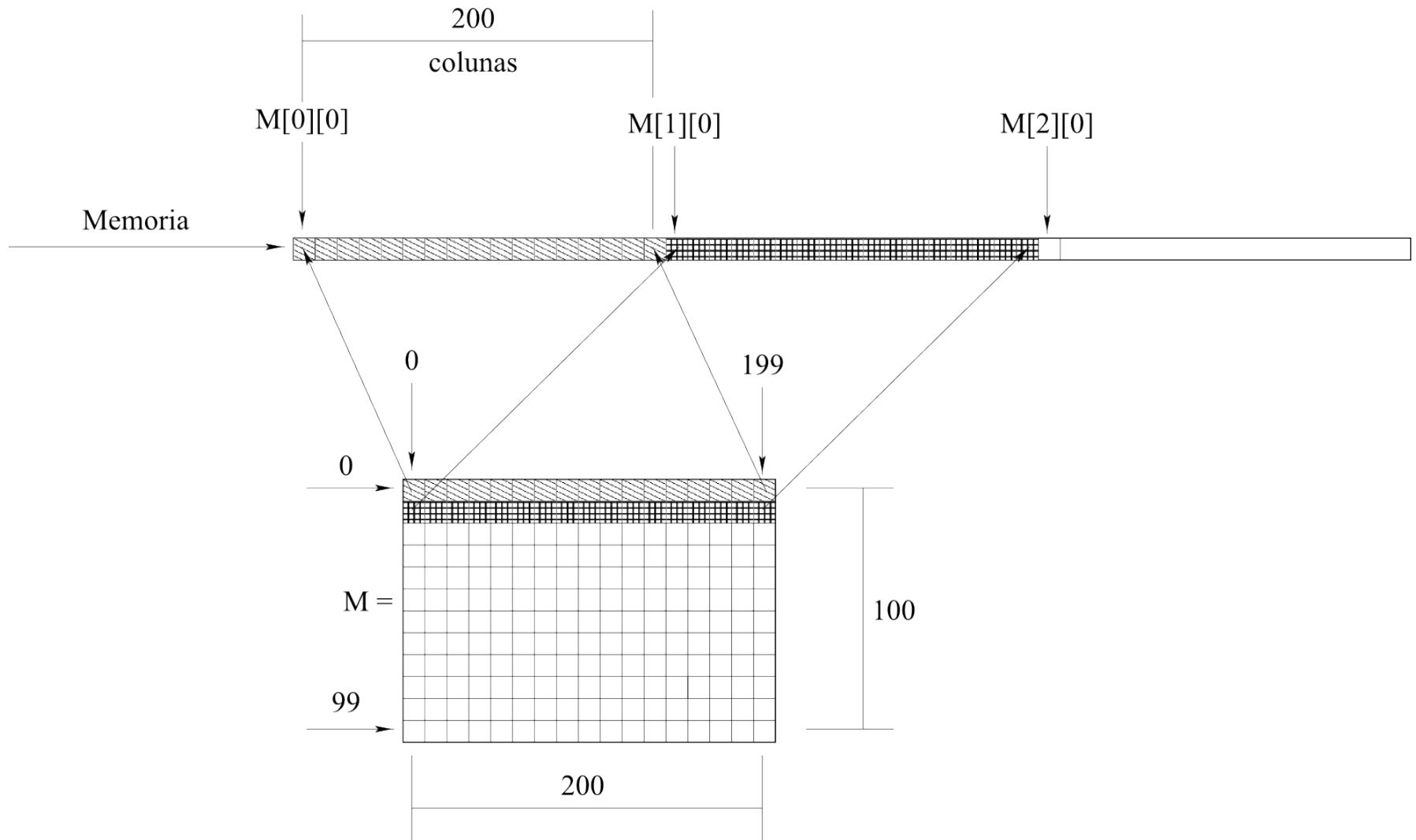
```
int *p;  
p = (int*) malloc(4);  
printf("%d", *p);  
*p = 5;  
printf("%d", *p);
```

Calloc

```
int *p;  
p = (int*) calloc(1,4);  
printf("%d", *p);  
*p = 5;  
printf("%d", *p);
```

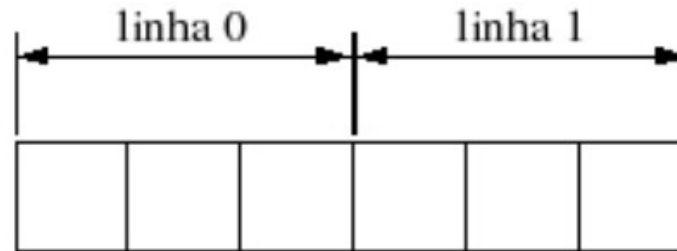
Alocação dinâmica: matrizes

- Como uma matriz fica armazenada na memória?

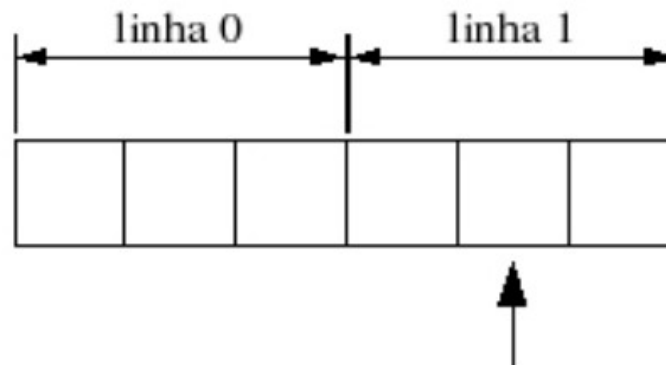


Alocação dinâmica: matrizes

- Quando fazemos `int m[2][3];` o C aloca espaço suficiente para 6 inteiros, um ao lado do outro:



- Uma atribuição do tipo `m[1][1] = 3;` só funciona se o compilador conhecer o comprimento da linha (ou seja, o número de colunas da matriz), aí ele sabe que tem que andar, no caso acima, uma linha inteira mais 2 casas, ou seja, cairá em:



Alocação dinâmica: matrizes

- Lembra que quando fazemos `v[i]`, estamos na verdade, fazendo `*(v+i)`?
- No caso de matrizes, quando fazemos `matriz[i][j]` estamos, na verdade, fazendo
`*(matriz + (i * numerocolunas) + j)`
- Então, vamos agora, trabalhar com alocação dinâmica e aritmética de ponteiros.

Alocação dinâmica: matrizes

```
int *m, nlin, ncol, i;

scanf("%d %d", &nlin, &ncol);

m = (int *) malloc(nlin * ncol * sizeof(int));

if (m == NULL){
    printf("Memoria nao alocada");
    exit(1);
}

for (i=0; i<nlin; i++)
    for (j=0; j<ncol; j++){
        printf("Entre m[%d][%d]: ", i, j);
        scanf("%d", (m + (i * ncol) + j)); // Igual à m[i][j]
    }

free(m);
```


Alocação dinâmica: matrizes

- Note que, como passamos o elemento da matriz que queríamos carregar ao scanf: $(m + (i * c) + j)$. Não precisamos usar o &.
- **Por que?**
 - Porque $(m + (i * c) + j)$ é um ponteiro para o elemento $[i][j]$ da matriz.
- A matriz é então alocada como um mega vetor, onde as linhas vêm uma em seguida da outra.
- O programa seguinte faz exatamente a mesma coisa que o anterior, só que sem usar
$$\text{matriz}[i][j] \rightarrow *(matriz + (i * c) + j)$$

Alocação dinâmica: matrizes

```
int *m, *p; /* p eh ponteiro auxiliar */
int nlin, ncol, i;

scanf("%d %d", &nlin, &ncol);

m = (int *) malloc(nlin * ncol * sizeof(int));

if (m == NULL){
    printf("Memoria nao alocada");
    exit(1);
}

p = m; /* p aponta para o primeiro elemento da matriz */

for (i=0; i<nlin; i++)
    for (j=0; j<ncol; j++){
        printf("Entre m[%d][%d]: ", i, j);
        scanf("%d", p); /* p, ou seja, um endereço */
        p++;
    }

free(m);
```

Alocação dinâmica: matrizes

- Essa última versão, apesar de ocupar mais memória (ponteiro auxiliar), é mais rápida, pois poupa as multiplicações ($i * c$), além de não precisar sempre buscar i , c e j na memória.
- Note também que poderíamos ter substituído

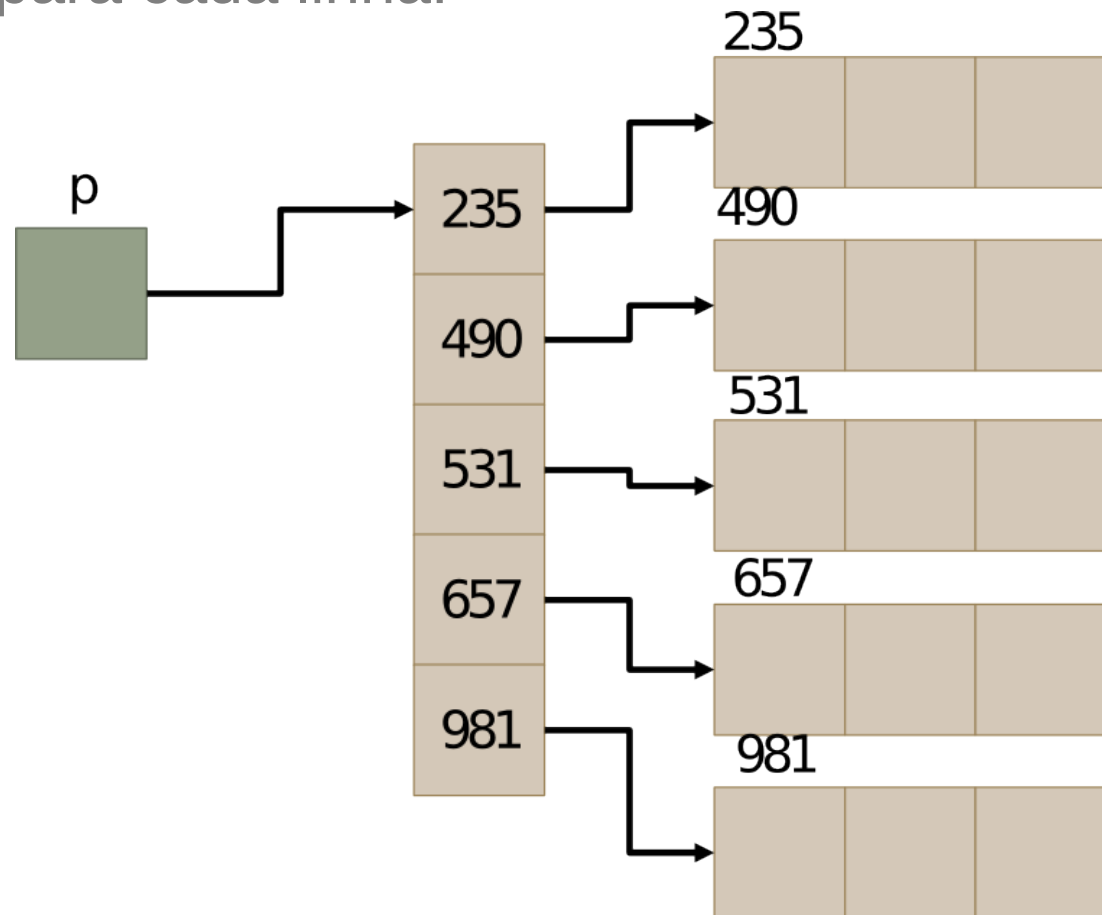
```
for (i=0; i<nlin; i++)  
    for (j=0; j<ncol; j++)
```

por

```
for (i=0; i<(nlin*ncol); i++)
```

Alocação dinâmica: matrizes

- Uma outra forma de alocar matrizes dinamicamente é alocar um vetor de apontadores, ou seja, um apontador por linha, e depois um vetor de elementos para cada linha:



Alocação dinâmica: matrizes

- Uma outra forma de alocar matrizes dinamicamente é alocar um vetor de apontadores, ou seja, um apontador por linha, e depois um vetor de elementos para cada linha:

```
int main(void){
    int **m;
    int nlin, ncol, i;

    scanf("%d %d", &nlin, &ncol);

    m = (int **) malloc(nlin*sizeof(int *));
    if(m != NULL){
        for (i = 0; i < nlin; i++)
            m[i] = (int *)malloc(ncol*sizeof(int));
    }

    // o programa continua...
    for (i = 0; i < nlin; i++)
        free(m[i]);
    free(m);
    return 0;
}
```

Alocação dinâmica: matrizes

- Uma outra forma de alocar matrizes dinamicamente é alocar um vetor de apontadores, ou seja, um apontador por linha, e depois um vetor de elementos para cada linha:

```
int main(void){
    int **m;
    int nlin, ncol, i;

    scanf("%d %d", &nlin, &ncol);

    m = (int **) malloc(nlin*sizeof(int *));
    if(m != NULL){
        for (i = 0; i < nlin; i++)
            m[i] = (int *)malloc(ncol*sizeof(int))
    }

    // o programa continua...
    for (i = 0; i < nlin; i++)
        free(m[i]);
    free(m);
    return 0;
}
```

Aloca um vetor de apontadores.

Alocação dinâmica: matrizes

- Uma outra forma de alocar matrizes dinamicamente é alocar um vetor de apontadores, ou seja, um apontador por linha, e depois um vetor de elementos para cada linha:

```
int main(void){
    int **m;
    int nlin, ncol, i;

    scanf("%d %d", &nlin, &ncol);

    m = (int **) malloc(nlin*sizeof(int *));
    if(m != NULL){
        for (i = 0; i < nlin; i++)
            m[i] = (int *)malloc(ncol*sizeof(int));
    }

    // o programa continua...
    for (i = 0; i < nlin; i++)
        free(m[i]);
    free(m);
    return 0;
}
```

Para cada linha aloca um vetor de elementos.

Alocação dinâmica: retorno de ponteiros

- Vimos que funções não podem retornar um vetor ou matriz (a não ser por passagem de parâmetros). Mas é possível retornar um ponteiro para um vetor ou matriz!

```
int **criaMatriz(int nlin, int ncol) {  
    int **m, i, j;  
  
    m = (int **) malloc(nlin*sizeof(int *));  
  
    if(m != NULL){  
        for (i = 0; i < nlin; i++)  
            m[i] = (int *)malloc(ncol*sizeof(int));  
    }  
  
    for (i=0; i<nlin; i++)  
        for (j=0; j<ncol; j++)  
            scanf("%d", &m[i][j]);  
  
    return(m);  
}
```


Alocação dinâmica: retorno de ponteiros

- Note que a função retorna um ponteiro duplo para inteiro, ou seja, seu tipo de retorno é “`int **`”.

```
int main(void) {  
    int **m, nlin, ncol;  
  
    printf("Entre o número de linhas: ");  
    scanf("%d %d", &nlin, &ncol);  
    m = criaMatriz(nlin, ncol);  
  
    // o programa continua ...  
    free(m);  
  
    return 0;  
}
```