SSC0501 - Introdução à Ciência de Computação I

Bem Vindos!

Alocação Dinâmica de Memória!

Alocação de memória

- **<u>Uso da memória:</u>**

- uso de **variáveis globais** (e **estáticas**):
  - O espaço reservado para uma variável global existe enquanto o programa estiver sendo executado.

- uso de **variáveis locais**:
  - Neste caso, o **espaço existe apenas enquanto a função que declarou a variável está sendo executada**, sendo liberado para outros usos quando a execução da função termina. Assim, a função que chama não pode fazer referência ao espaço local da função chamada.

Alocação dinâmica de memória

```
118 int main()
119 {
120     char palavra[101], invertida[101],
121          p_maiuscula[101];
122
123     scanf("%s",palavra);
124
125     if( eh_palindromo(palavra) )
126     {
127         printf("Palindromo: SIM\n");
128     }else
129     {
130         printf("Palindromo: NAO\n");
131     }
132
133     printf("Vogais: %d | Consoantes: %d\n" ,
134         contar_vogais(palavra),  contar_consoantes(palavra) );
135
136     para_maiusculas(palavra, p_maiuscula);
137
138     printf("Maiuscula: %s\n", p_maiuscula);
139
140     inverter(palavra, invertida);
141
142     printf("Invertida: %s\n", invertida);
143
144
145     return 0;
146 }
```

Alocação dinâmica de memória

```
118 int main()
119 {
120     char palavra[101], invertida[101],
121          p_maiuscula[101];
122
123     scanf("%s",palavra);
124
125     if( eh_palindromo(palavra) )
126     {
127         printf("Palindromo: SIM\n");
128     }else
129     {
130         printf("Palindromo: NAO\n");
131     }
132
133     printf("Vogais: %d | Consoantes: %d\n" ,
134         contar_vogais(palavra),  contar_consoantes(palavra) );
135
136     para_maiusculas(palavra, p_maiuscula);
137
138     printf("Maiuscula: %s\n", p_maiuscula);
139
140     inverter(palavra, invertida);
141
142     printf("Invertida: %s\n", invertida);
143
144
145     return 0;
146 }
```

Será que existe uma forma de não precisar "passar" um vetor/string vazio para ser preenchido?

Alocação dinâmica de memória

```
118 int main()
119 {
120     char palavra[101], invertida[101],
121          p_maiuscula[101];
122
123     scanf("%s",palavra);
124
125     if( eh_palindromo(palavra) )
126     {
127         printf("Palindromo: SIM\n");
128     }else
129     {
130         printf("Palindromo: NAO\n");
131     }
132
133     printf("Vogais: %d | Consoantes: %d\n" ,
134         contar_vogais(palavra),  contar_consoantes(palavra) );
135
136     para_maiusculas(palavra, p_maiuscula);
137
138     printf("Maiuscula: %s\n", p_maiuscula);
139
140     inverter(palavra, invertida);
141
142     printf("Invertida: %s\n", invertida);
143
144
145     return 0;
146 }
```

```
 97 void para_maiusculas(char palavra[], char resultado[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     for(int i = 0; i < tam; i++)
102     {
103         resultado[i] = palavra[i] - 32;
104     }
105     resultado[tam] = '\0';
106 }
```

```
14 void inverter(char palavra[], char invertida[])
15 {
16     int tamanho= tamanho_str(palavra);
17
18
19     for( int i = tamanho-1; i >= 0; i--)
20     {
21         invertida[ tamanho - i - 1 ] = palavra[ i ];
22     }
23     invertida[tamanho] = '\0';
24
25 }
```

Será que existe uma forma de não precisar “passar” um vetor/string vazio para ser preenchido?

Alocação dinâmica de memória

```
 97 void para_maiusculas(char palavra[], char resultado[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     for(int i = 0; i < tam; i++)
102     {
103         resultado[i] = palavra[i] - 32;
104     }
105     resultado[tam] = '\0';
106 }
```

```
 97 char* para_maiusculas(char palavra[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     char resultado[tam]
102
103     for(int i = 0; i < tam; i++)
104     {
105         resultado[i] = palavra[i] - 32;
106     }
107     resultado[tam] = '\0';
108
109     return resultado;
110 }
```

Será que funciona?

Alocação dinâmica de memória

```
 97 void para_maiusculas(char palavra[], char resultado[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     for(int i = 0; i < tam; i++)
102     {
103         resultado[i] = palavra[i] - 32;
104     }
105     resultado[tam] = '\0';
106 }
```

```
 97 char* para_maiusculas(char palavra[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     char resultado[tam]
102
103     for(int i = 0; i < tam; i++)
104     {
105         resultado[i] = palavra[i] - 32;
106     }
107     resultado[tam] = '\0';
108
109     return resultado;
110 }
```

Será que funciona?

Alocação dinâmica de memória

```
 97 void para_maiusculas(char palavra[], char resultado[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     for(int i = 0; i < tam; i++)
102     {
103         resultado[i] = palavra[i] - 32;
104     }
105     resultado[tam] = '\0';
106 }
```

```
 97 char* para_maiusculas(char palavra[])
 98 {
 99     int tam = tamanho_str(palavra);
100
101     char resultado[tam]
102
103     for(int i = 0; i < tam; i++)
104     {
105         resultado[i] = palavra[i] - 32;
106     }
107     resultado[tam] = '\0';
108
109     return resultado;
110 }
```

Será que funciona?
Resposta: NÃO FUNCIONA!
Porque?

Alocação dinâmica de memória

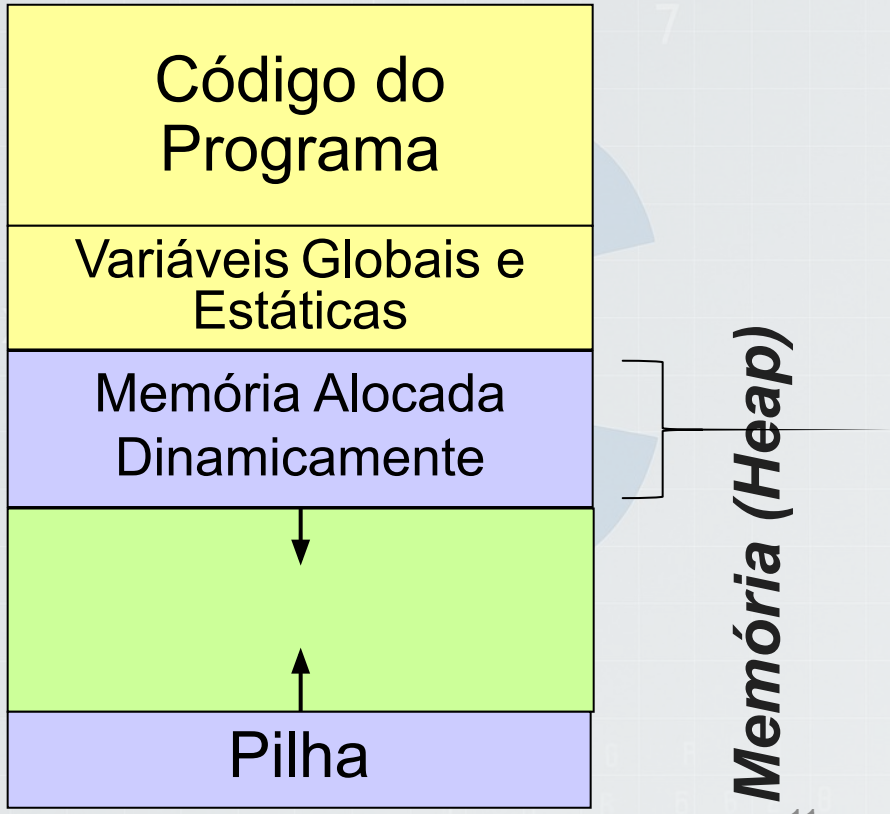
- A linguagem C oferece meios de **requisitarmos espaços de memória em tempo de execução**.

- **O espaço alocado dinamicamente permanece reservado até que explicitamente seja liberado pelo programa.**
  - Por isso, podemos alocar dinamicamente um espaço de memória numa função e acessá-lo em outra.

- A partir do momento que liberarmos o espaço, ele estará disponibilizado para outros usos e não podemos mais acessá-lo.
  - Se o programa não liberar um espaço alocado, este será automaticamente liberado quando a execução do programa terminar.

Alocação da Memória Principal

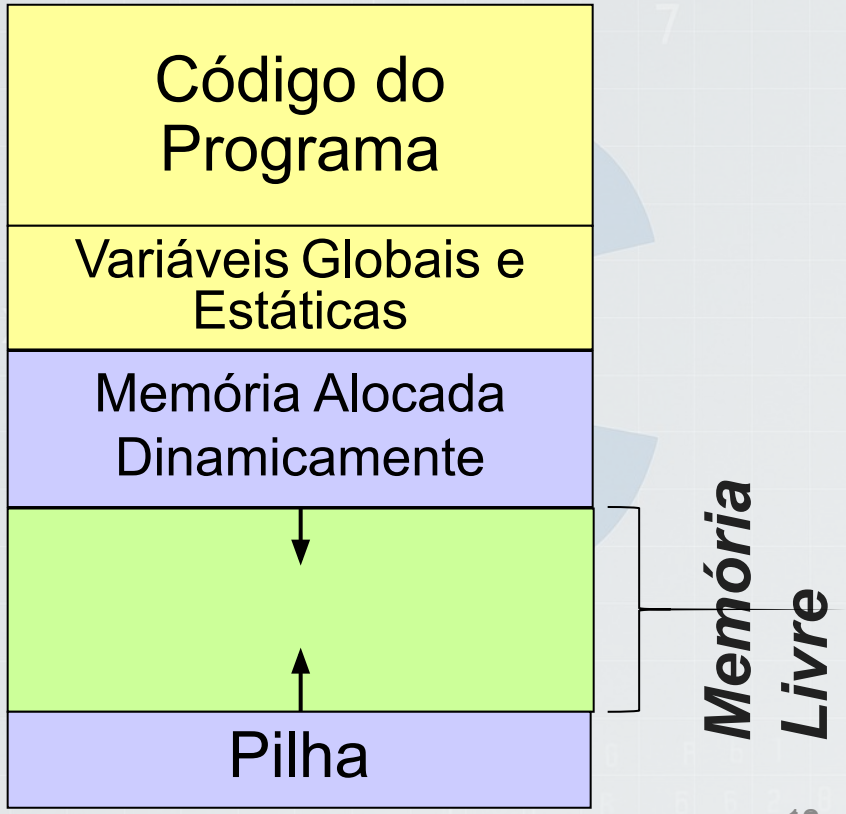
- Para executar um determinado programa, o S.O. carrega na memória o código do programa, em linguagem de máquina. Além disso, o S.O. reserva os espaços necessários para armazenar as variáveis globais (e estáticas) do programa.
- O restante da memória livre é utilizado pelas variáveis locais e pelas variáveis alocadas dinamicamente.



Alocação da Memória Principal

- Cada vez que uma função é chamada, o S.O. reserva o espaço necessário para as variáveis locais da função. Este espaço pertence à pilha de execução e, quando a função termina, é liberado.

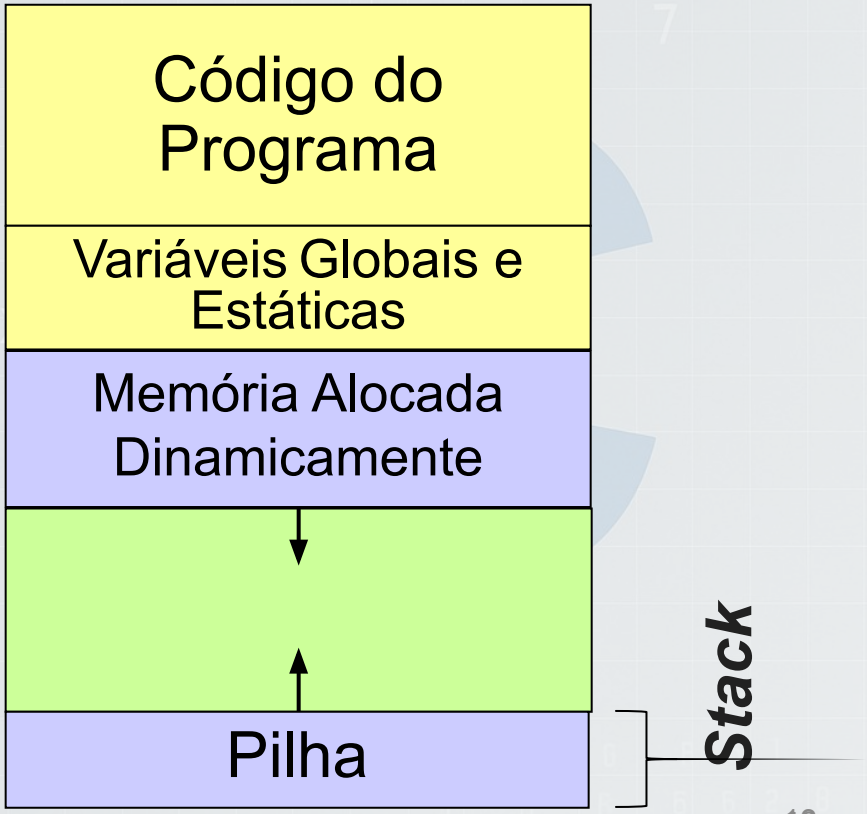
- A memória não é ocupada pela pilha de execução pode ser requisitada dinamicamente. Se a pilha tentar crescer mais do que o espaço disponível existente, dizemos que ela "estourou" e o programa é abortado com erro.



Alocação da Memória Principal

- Cada vez que uma função é chamada, o S.O. reserva o espaço necessário para as variáveis locais da função. Este espaço pertence à pilha de execução e, quando a função termina, é liberado.

- A memória não é ocupada pela pilha de execução pode ser requisitada dinamicamente. Se a pilha tentar crescer mais do que o espaço disponível existente, dizemos que ela "estourou" e o programa é abortado com erro.



Alocação Dinâmica de Memória

- As funções calloc, malloc e realloc permitem alocar blocos de memória em tempo de execução.

Protótipo da função malloc:

```
void * malloc(size_t n);   /* retorna um ponteiro void para n bytes de memória não iniciados. Se não há memória disponível malloc retorna NULL */
```

Funções para Alocar e Liberar memória

- A função *malloc* é usada para alocar espaço para armazenarmos valores de qualquer tipo. Por este motivo, *malloc* retorna um ponteiro genérico, para um tipo qualquer, representado por **void*** , que pode ser convertido automaticamente pela linguagem para o tipo apropriado na atribuição.

- No entanto, é comum fazermos a conversão explicitamente, utilizando o operador de molde de tipo (cast).

- Então:

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

Obs.: O operador *sizeof*() retorna o número de bytes de um determinado tipo de variável.

Funções para Alocar e Liberar memória

- **Se não houver espaço** livre suficiente para realizar a alocação, a função **retorna um endereço nulo** (representado pelo símbolo NULL, definido em ***stdlib.h***).

- Podemos tratar o erro na alocação do programa simplesmente verificando o valor de retorno da função *malloc*

- Ex: imprimindo mensagem e abortando o programa com a função ***exit***, também definida na ***stdlib***.

```
v = (int*) malloc(10*sizeof(int));
if (v == NULL) {
    printf("Memoria insuficiente.\n");
     exit(1);    /* aborta o programa e retorna 1 para o sist. operacional */
} …
```

Alocação da Memória Principal

Exemplos:

- Código que aloca 1000 bytes de memória livre:

```
char *p;
p = (char*) malloc(1000);
```

- Código que aloca espaço para 50 inteiros:

```
int *p;
p = (int*) malloc(50*sizeof(int));
```

Alocação Dinâmica de Memória

- As funções *calloc* e *malloc* permitem alocar blocos de memória em tempo de execução.

- Protótipo da função *calloc*:

void * calloc(**size_t** n, **size_t** size);

/ calloc retorna um ponteiro para um array com n elementos de tamanho size cada um ou NULL se não houver memória disponível. Os elementos são iniciados em zero */*

Alocação Dinâmica de Memória

- O ponteiro retornado por tanto pela função *malloc* quanto pela *calloc* devem ser convertido para o tipo de ponteiro que invoca a função

```
int *pi = (int *) malloc (n*sizeof(int));

int *ai = (int *) calloc (n, sizeof(int));
/* aloca espaço para um array de n inteiros */
```

- toda memória não mais utilizada deve ser liberada através da função *free()*:

```
free(ai); /* libera todo o array */
free(pi);
```

Funções para Alocar e Liberar memória

- A função **realloc()** serve para realocar memória. A função modifica o tamanho da memória previamente alocada apontada por ***ptr** para aquele especificado por **num**. O valor de **num** pode ser maior ou menor que o original.

- Protótipo:

```
void *realloc (void *ptr, unsigned int num);
```

Alocação Dinâmica

VETORES E MATRIZES

Vetores e alocação dinâmica

- A forma mais simples de estruturarmos um conjunto de dados é por meio de vetores.

- Definimos um vetor em C da seguinte forma:

int v[10];

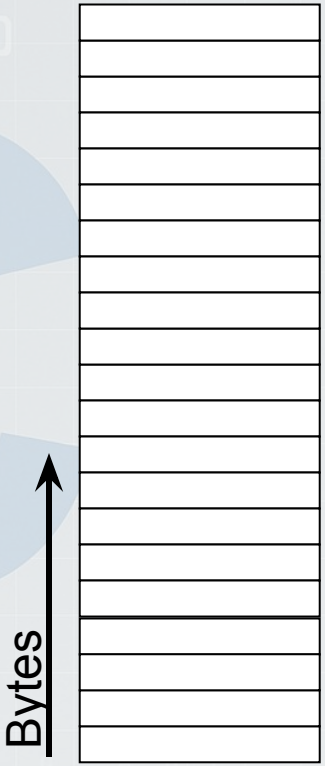
- Esta declaração diz que:
  - v é um vetor de inteiros dimensionado com 10 elementos, isto é, reservamos um espaço de memória **contínuo** para armazenar 10 valores inteiros.
  - Assim, se cada **int** ocupa 4 bytes, a declaração reserva um espaço de memória de 40 bytes

Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int)); /* aloca um vetor de n posições inteiras */
    for (i =0; i<n; i++) {
        printf(“Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);      /* armazena o valor no vetor  na posição i */
     }
     for (i =0; i<n; i++)
         printf("%d ", v[i]);
    free(v);              /* libera a memória alocada para o vetor */
    return 0;
}
```

Vetores e alocação dinâmica

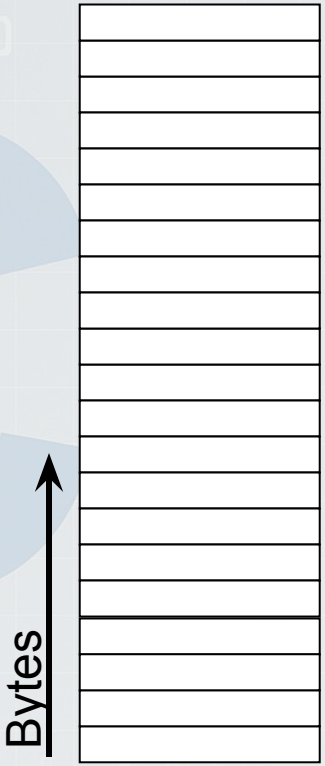
```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

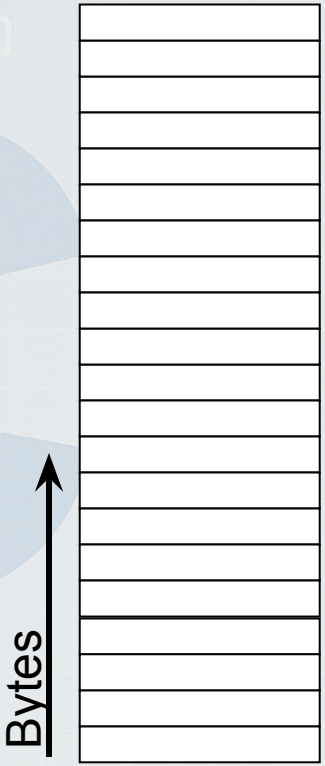
v = ??, n=??, i =??



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

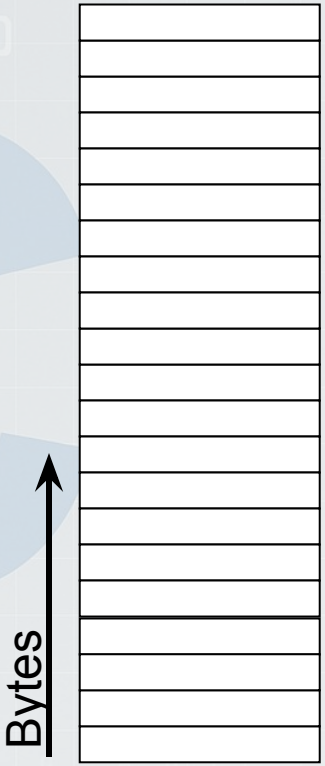
v = ??, n=??, i =??



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

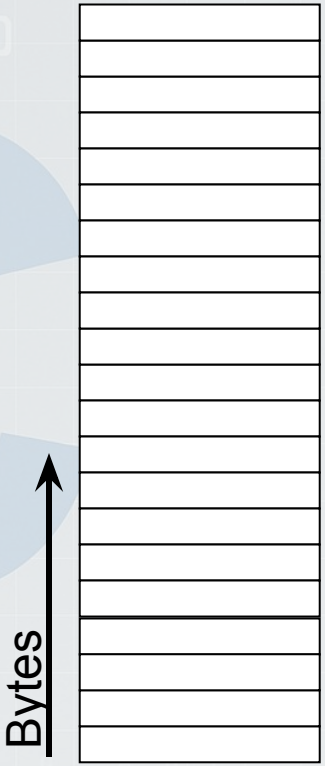
v = ??, n=??, i =??



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

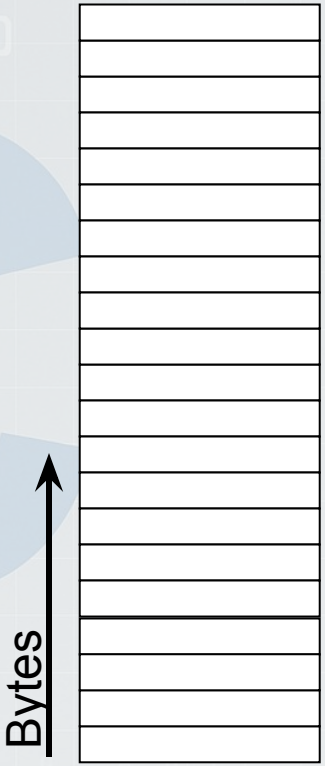
v = ??, **n=2**, i =??



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

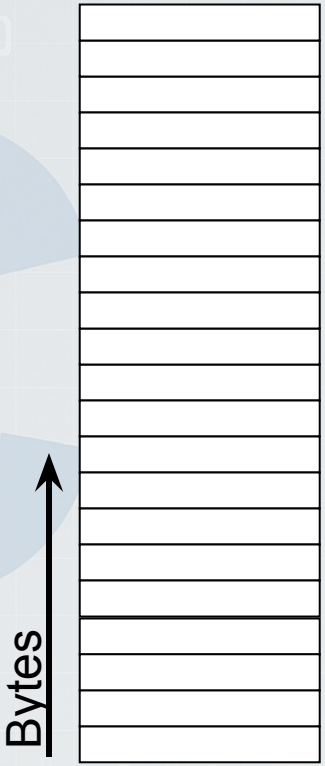
v = ??, **n=2**, i =??



Vetores e alocação dinâmica

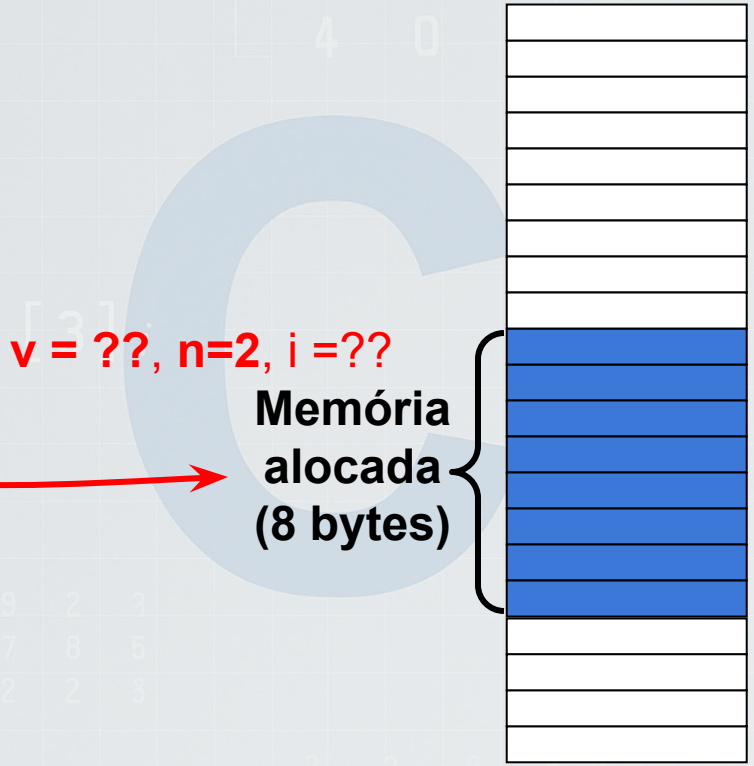
```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

v = ??, n=2, i =??



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

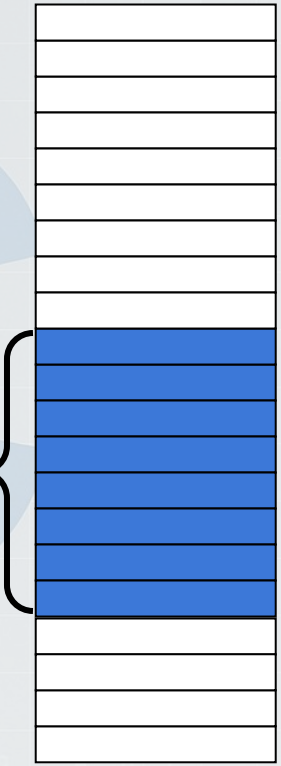


Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

Lembre-se que **1 inteiro** ocupa **4 bytes**
sizeof(int) = 4

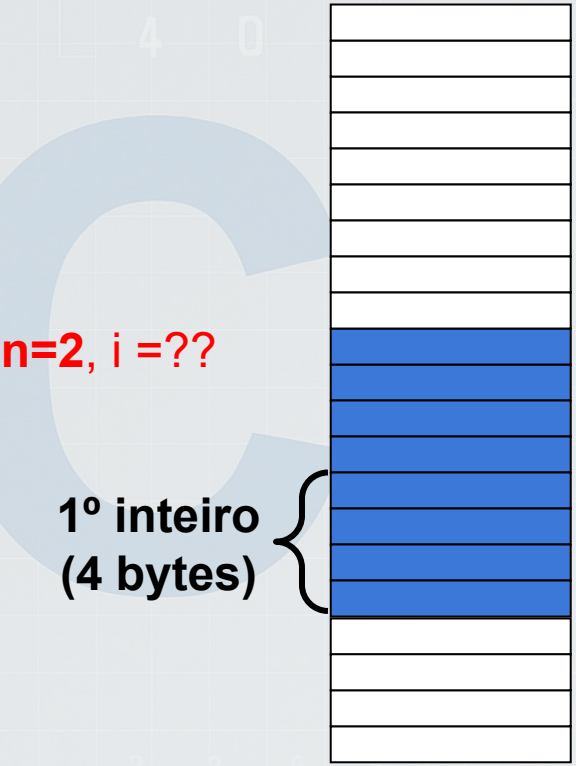
v = ??, n=2, i =??



Vetores e alocação dinâmica

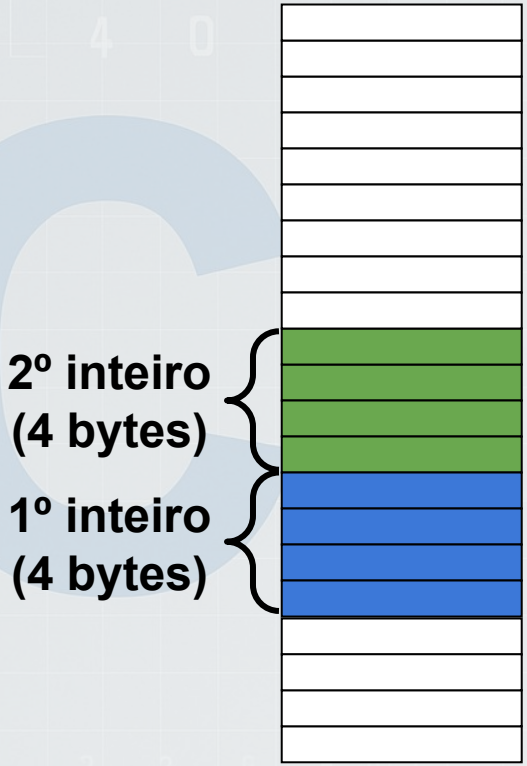
```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

v = ??, n=2, i =??



Vetores e alocação dinâmica

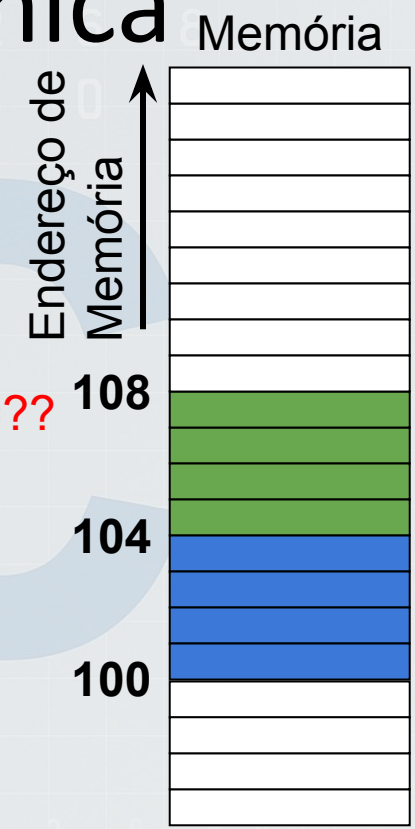
```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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



Vetores e alocação dinâmica

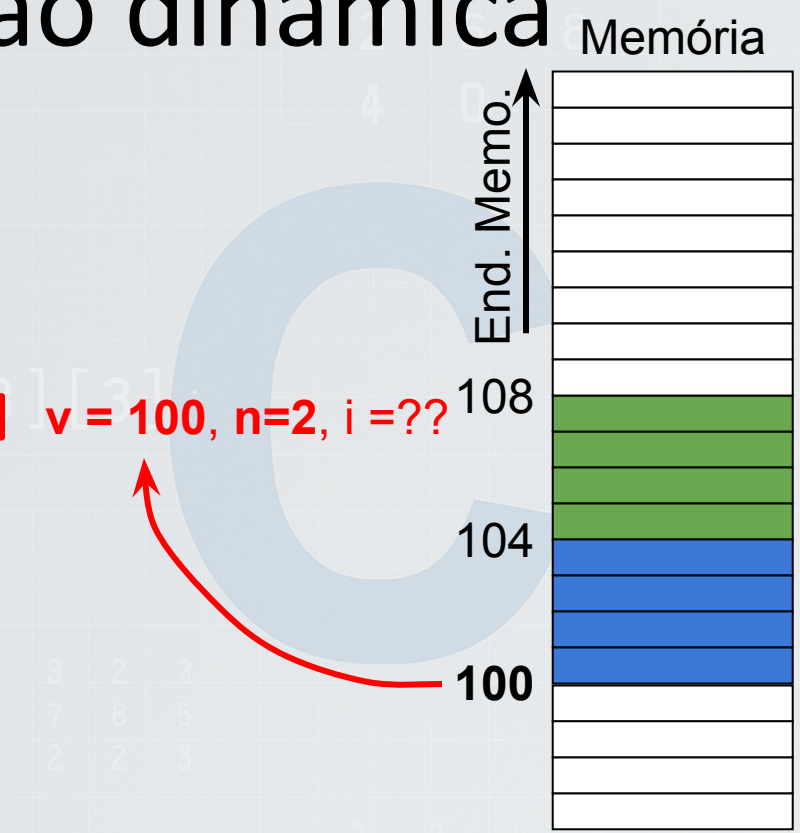
```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

v = ??, n=2, i =??



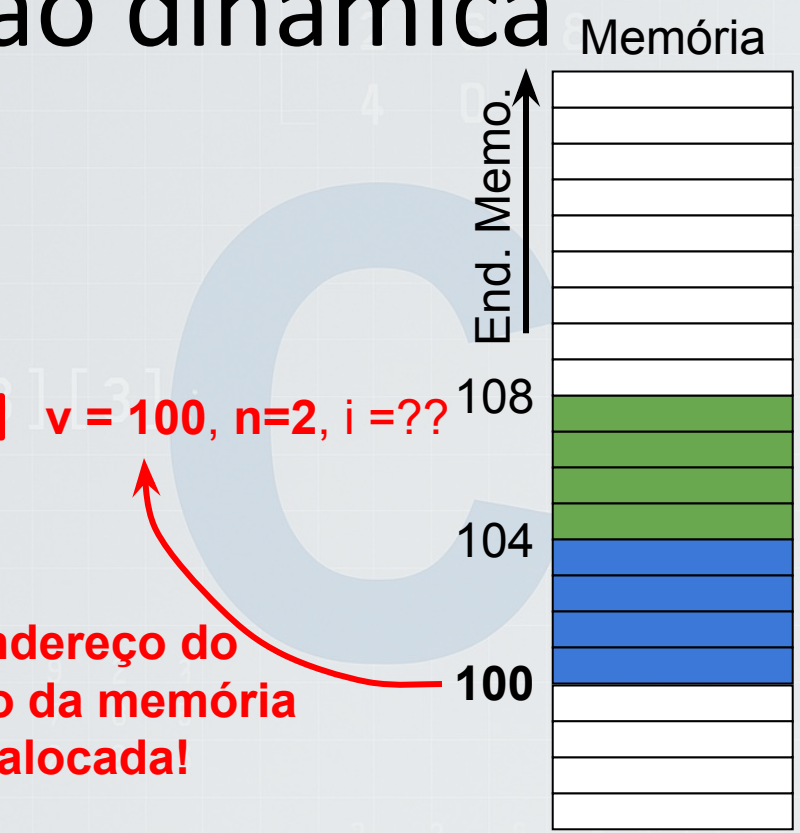
Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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



Vetores e alocação dinâmica

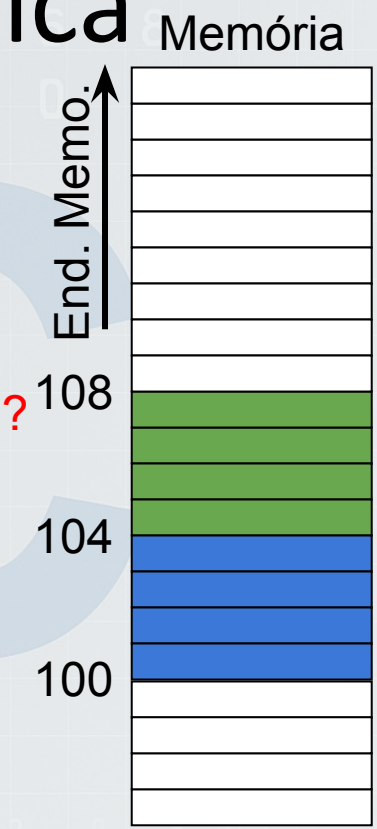
```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

v = 100, n=2, i =??

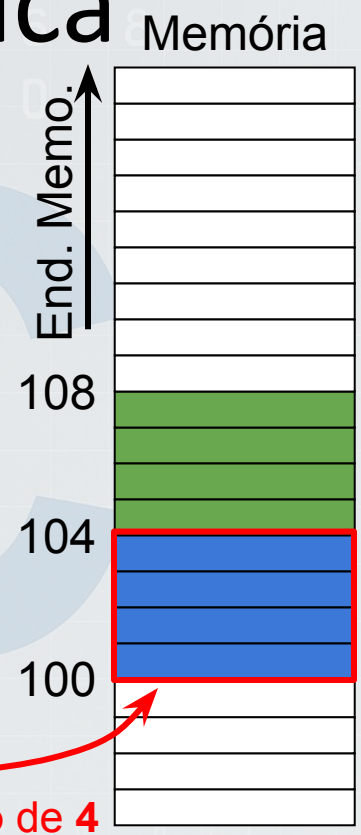


Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

v = 100, i = 0

v[0] = *(v+0) = *(100)



Acessa o **inteiro** de **4 bytes** no **endereço 100**

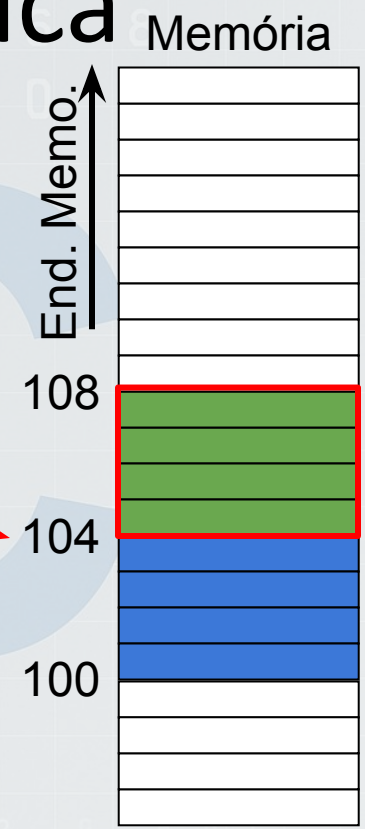
Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

v[1] = *(v+4) = *(104)

Acessa o **inteiro** de **4 bytes** no **endereço 104**

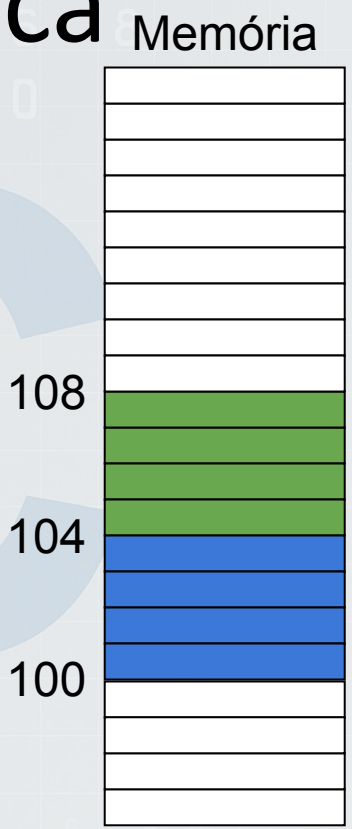
v = 100, i = 1



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

E se por acaso i=2?

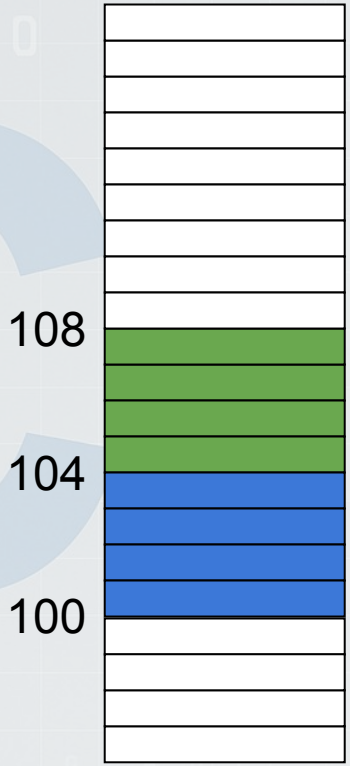


Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

E se por acaso i=2?

v[2] = *(v+8) = *(108)

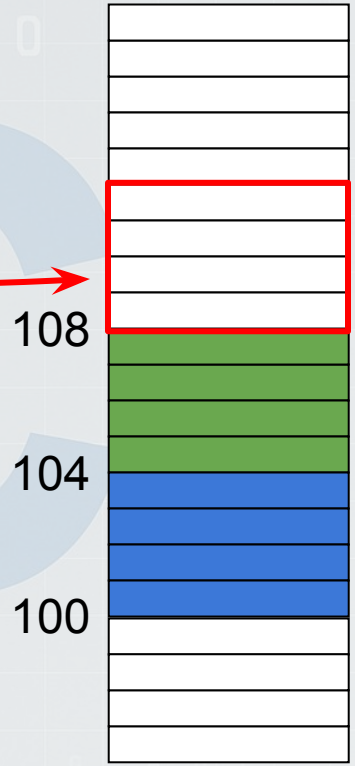


Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

E se por acaso i=2?

v[2] = *(v+8) = *(108)



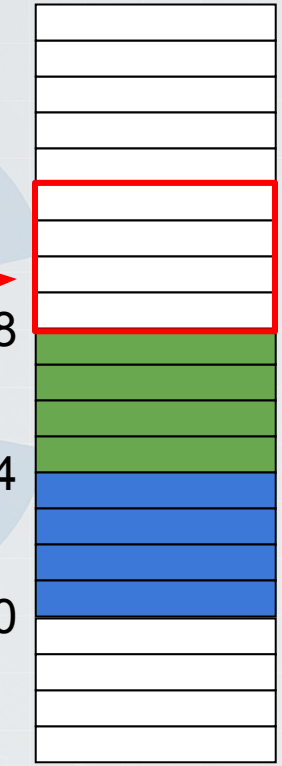
Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

E se por acaso i=2?

v[2] = *(v+8) = *(108)

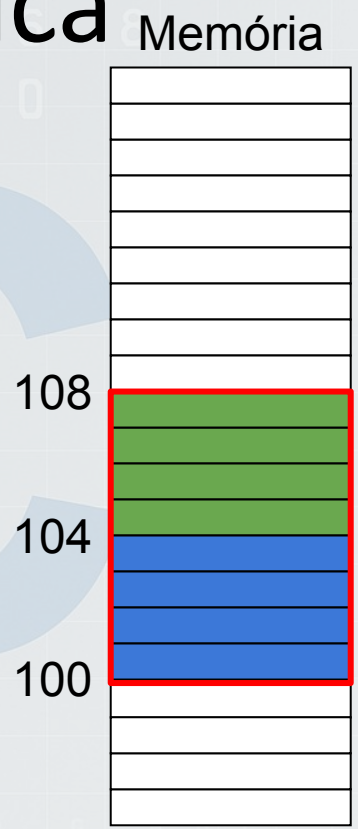
Acesso a memória não alocada!



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

Nunca esqueça de desalocar a memória!

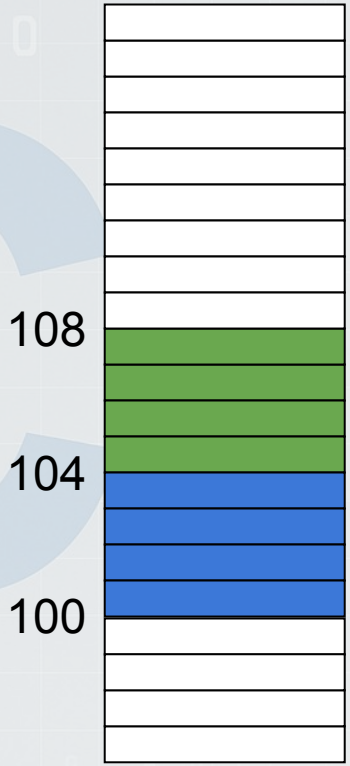


Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

free(v) = free(100)

Nunca esqueça de desalocar a memória!

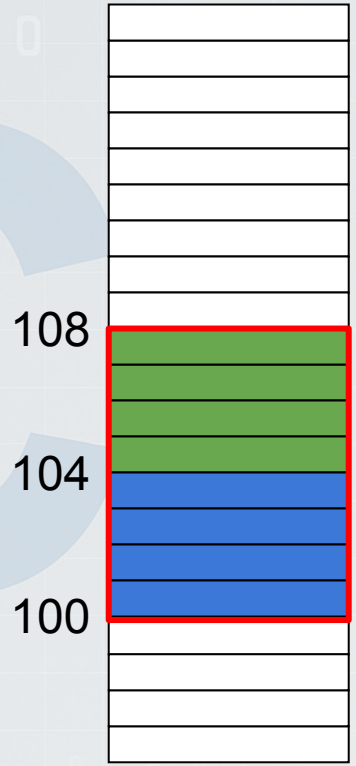


Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

Nunca esqueça de desalocar a memória!

free(v) = free(100)



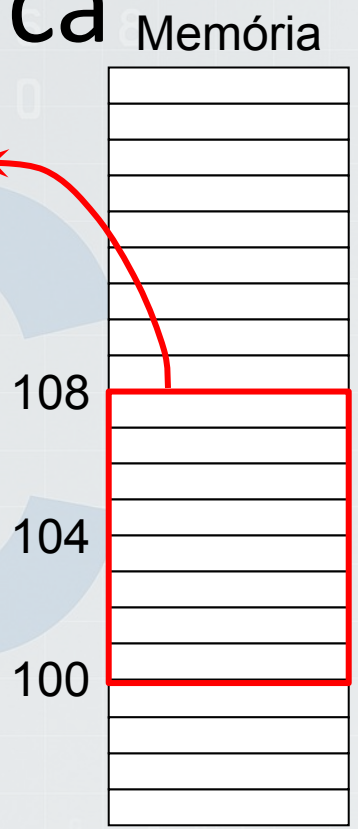
Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

Área livre para ser alocada para outros propósitos!

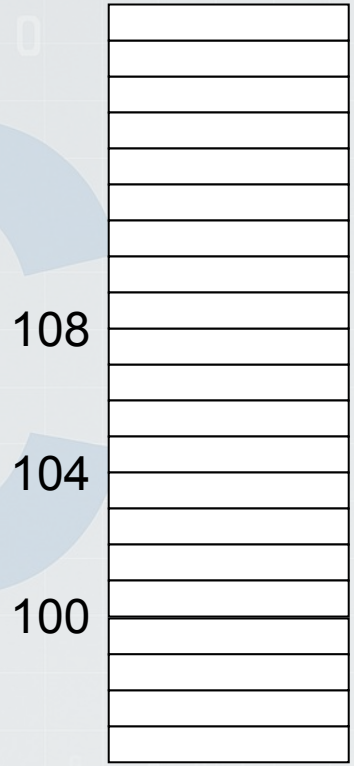
free(v) = free(100)

Nunca esqueça de desalocar a memória!



Vetores e alocação dinâmica

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

int main ()
{
    int *v, n, i;
    printf("Qual o tamanho do vetor que deseja:");
    scanf("%d", &n);
    v = (int *) calloc(n, sizeof(int));

    for (i =0; i<n; i++)
    {
        printf("Informe o %dº elemento: ", i+1);
        scanf("%d", &v[i]);
    }

    for (i =0; i<n; i++)
        printf("%d ", v[i]);

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

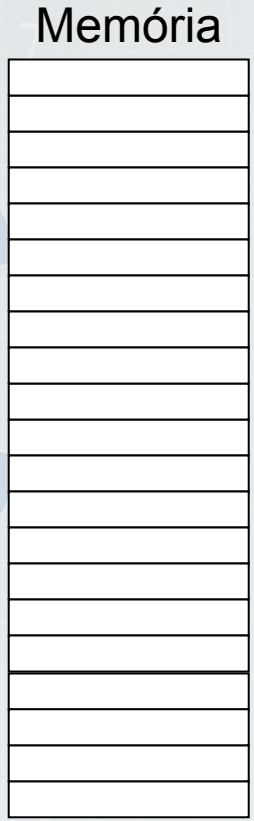


Alocação dinâmica de **matrizes**

- A alocação dinâmica de memória para **matrizes** é realizada da mesma forma que para vetores, com a diferença que teremos **um ponteiro apontando para outro ponteiro que aponta para o valor final**, o que é denominado indireção múltipla.
  - A indireção múltipla pode ser levada a qualquer dimensão desejada.

Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

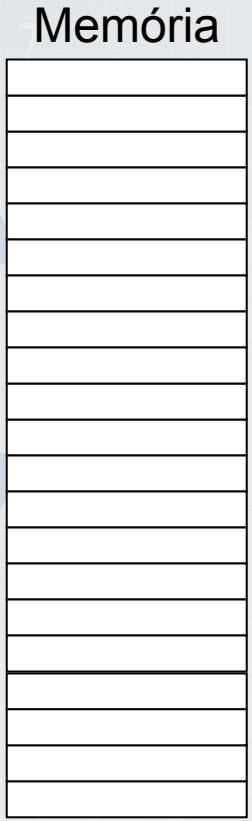
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alo
        return 1;
    }

    // Alocar cada linha (v
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

Para simplificar a visualização, considere que cada bloco representa 4 bytes de memória

Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (v
    for (i = 0; i < rows; i
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

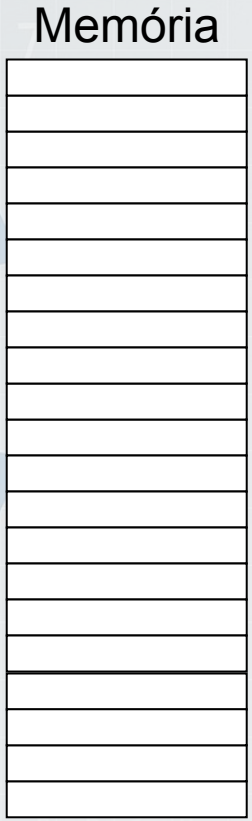
Quanto ocupa de espaço um ponteiro?



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (v
    for (i = 0; i < rows; i
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

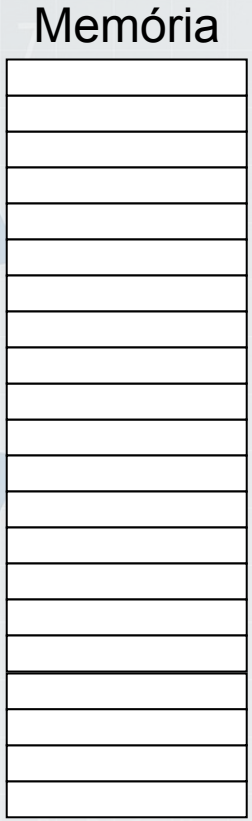
Quanto ocupa de espaço um ponteiro?
Arquitetura 32 bit -> 4 byte
Arquitetura 64 bits -> 8 bytes



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (v
    for (i = 0; i < rows; i
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

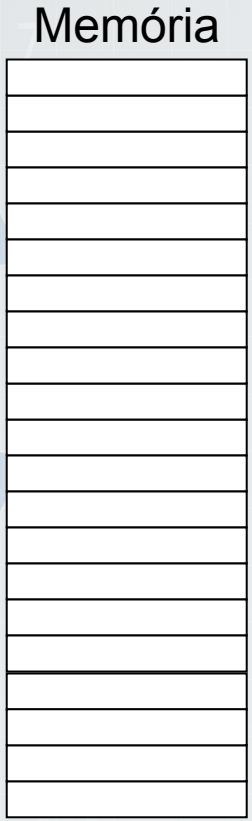
Assumindo arquitetura 64 bits (mais comum)...



Alocação dinâmica de matrizes

A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

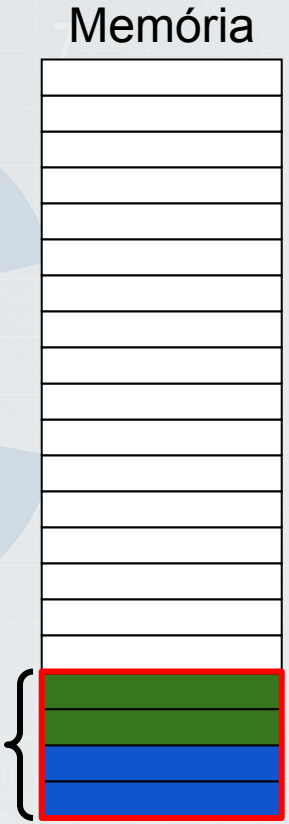
A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!

Memória para 2 ponteiros para ponteiros (16 bytes)

Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!

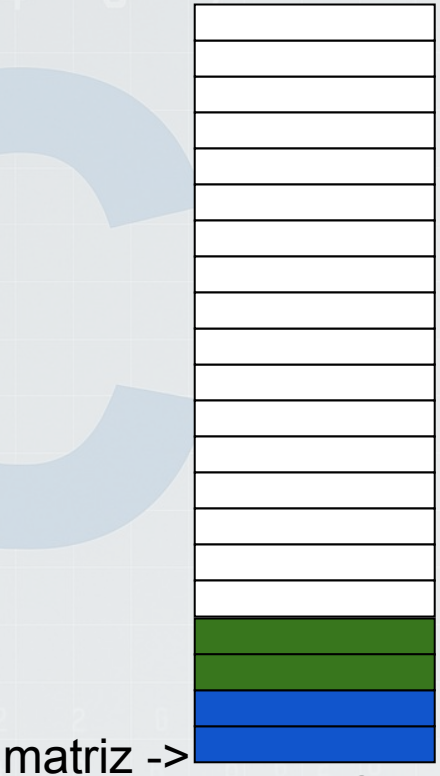


Memória para 2 ponteiros para ponteiros (16 bytes)

Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

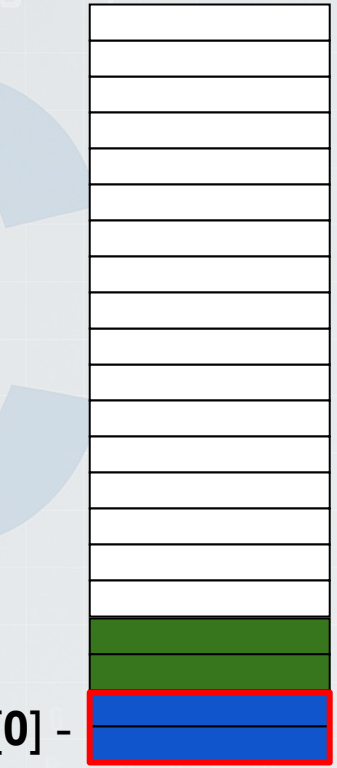
A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

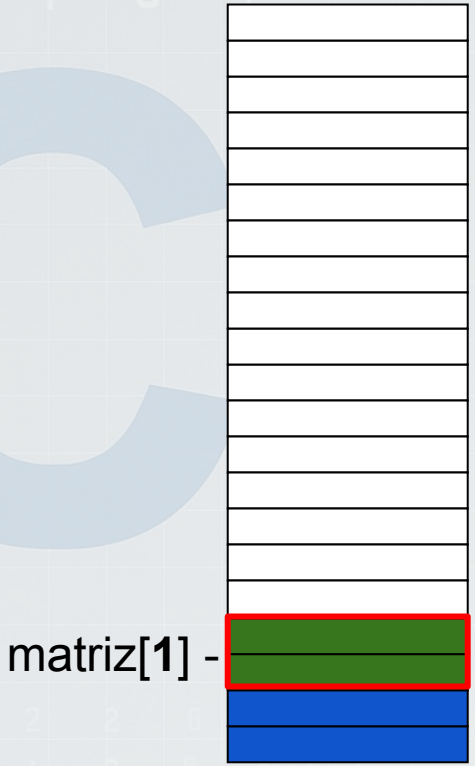
A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!



Alocação dinâmica de matrizes

A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!

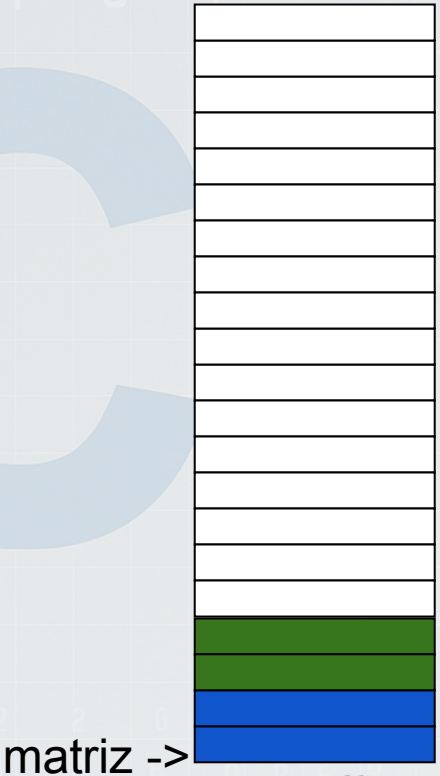
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

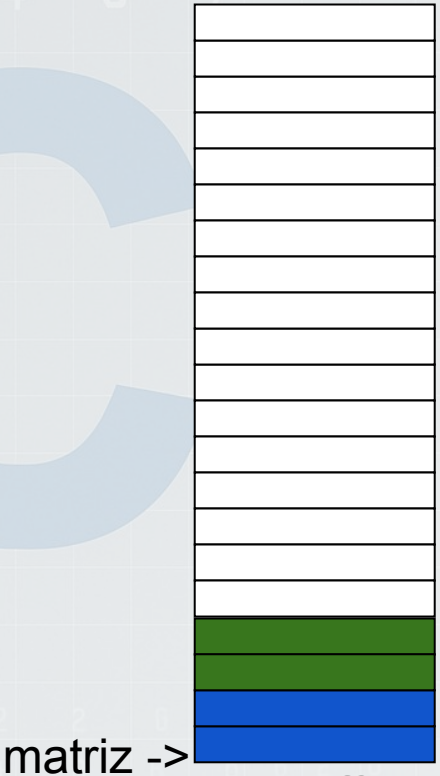
A declaração com duplo asterisco (**) indica um ponteiro para ponteiro!



Alocação dinâmica de matrizes

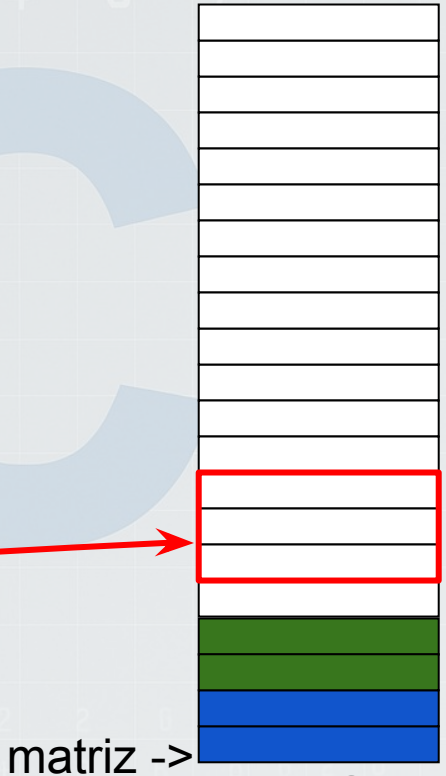
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

i=0, cols = 3



Alocação dinâmica de matrizes

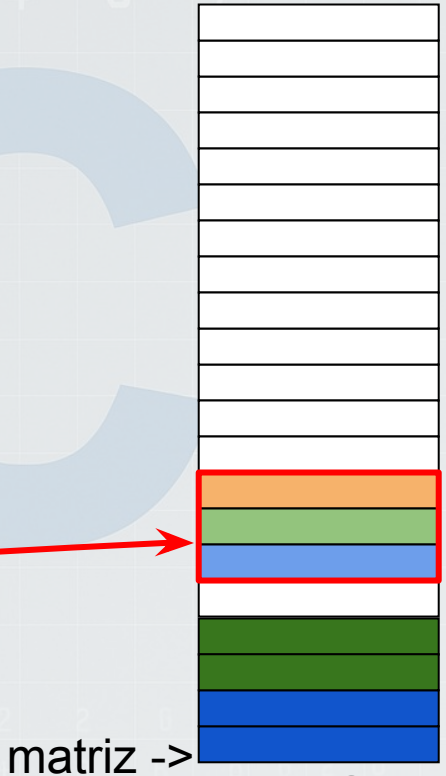
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

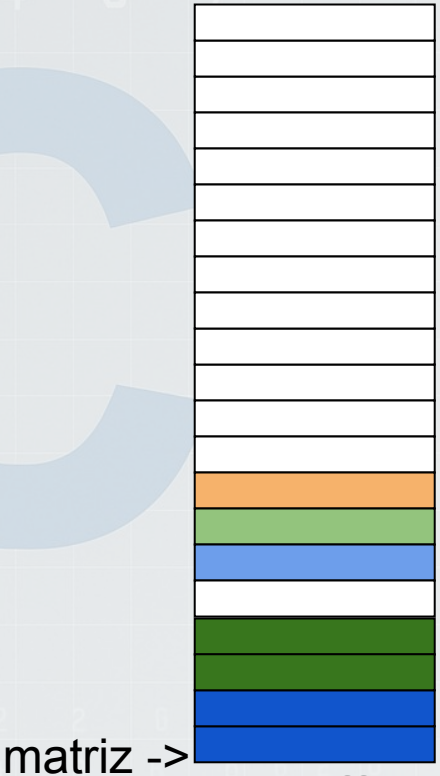
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

i=0, cols = 3



Alocação dinâmica de matrizes

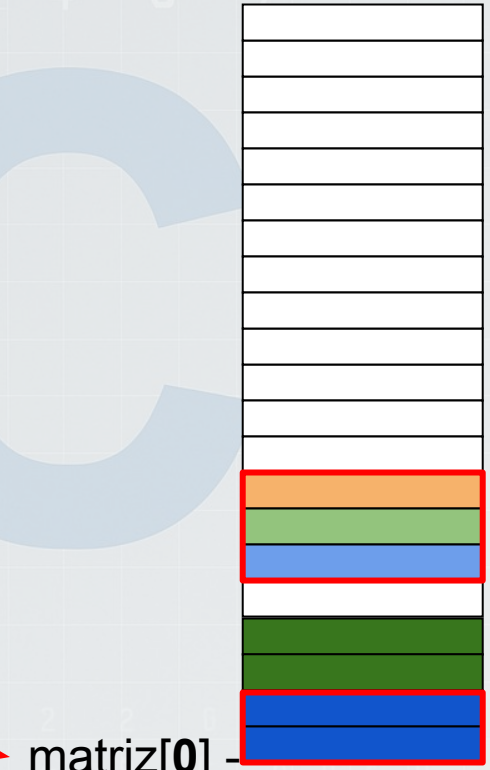
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

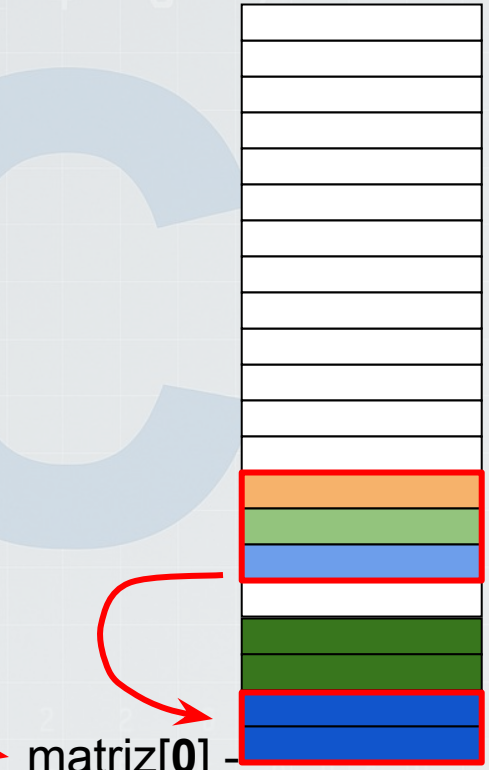
i=0, matriz[0], cols = 3



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

i=0, matriz[0], cols = 3

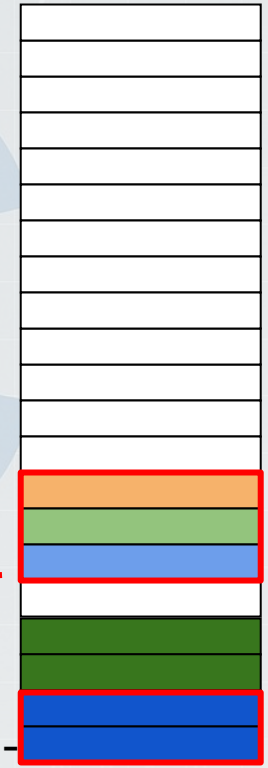


Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

i=0, matriz[0], cols = 3

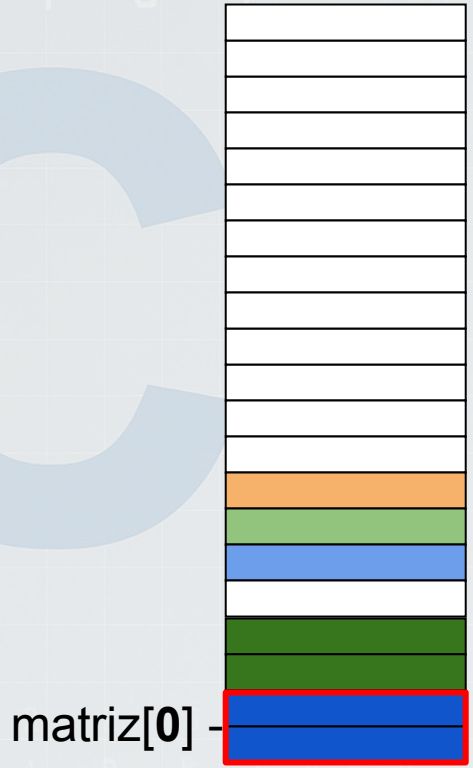
Atribui endereço de memória



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

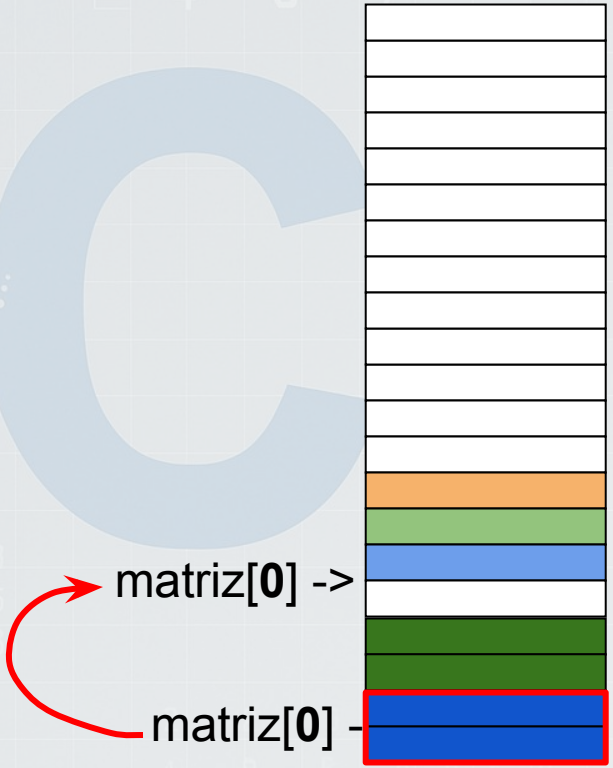
i=0, matriz[0], cols = 3



Alocação dinâmica de matrizes

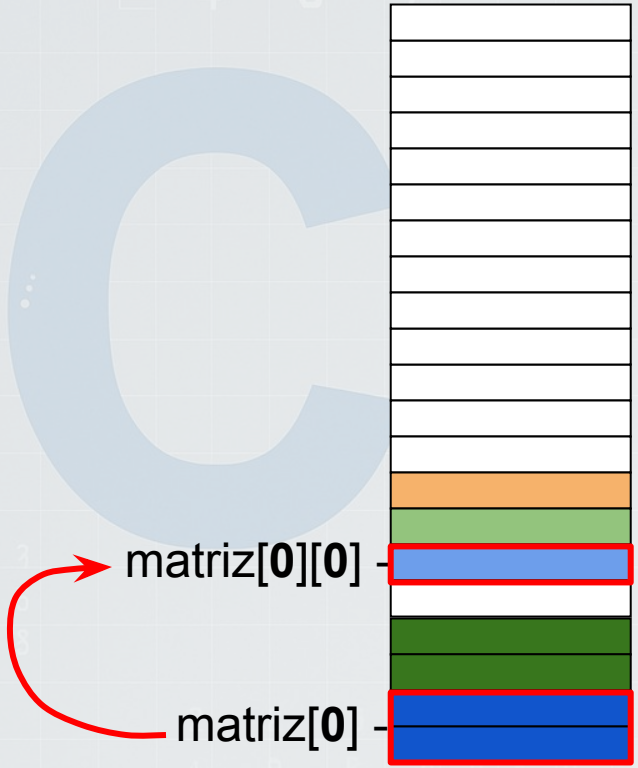
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

i=0, matriz[0], cols = 3



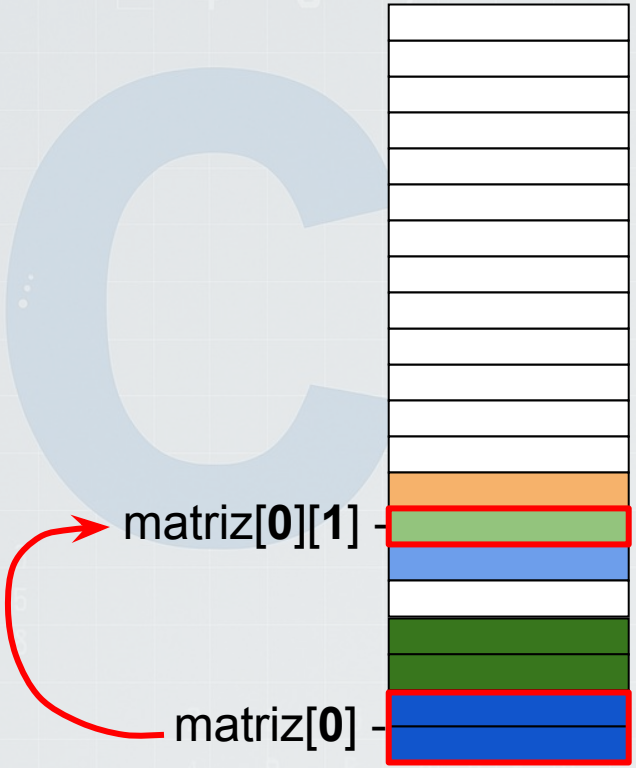
Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



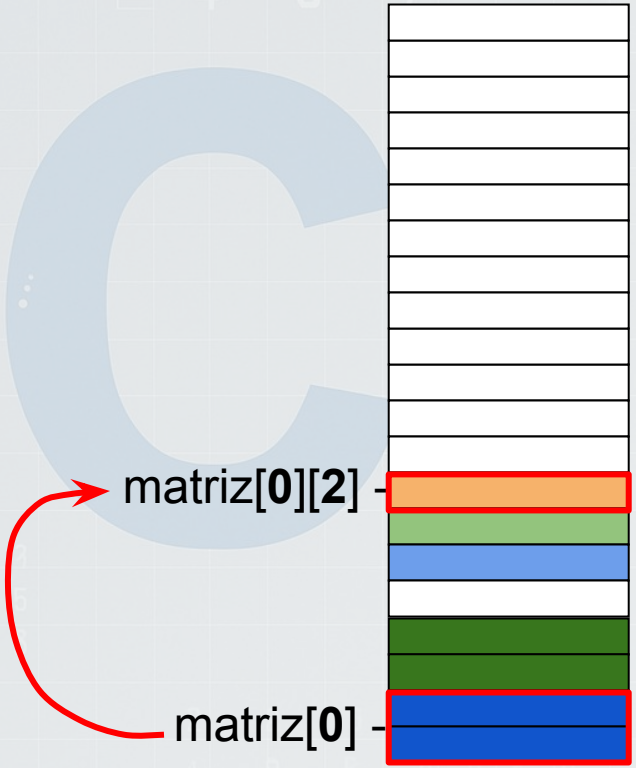
Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



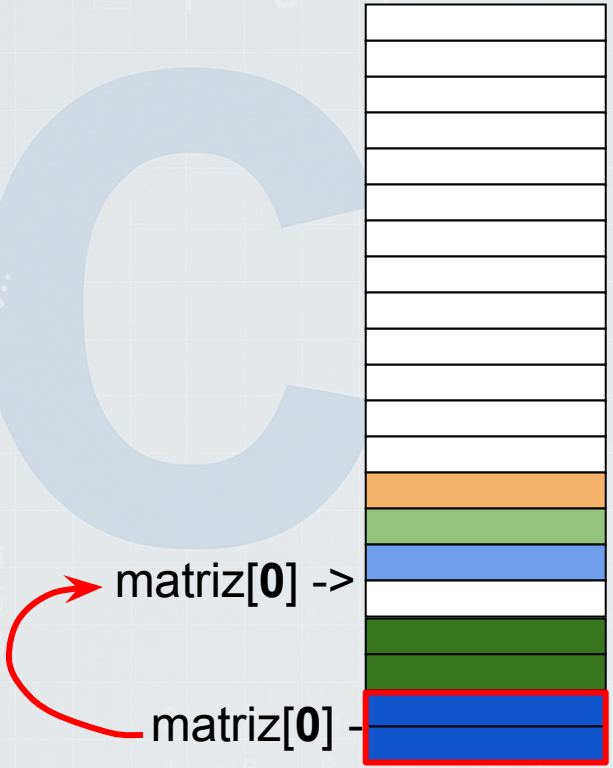
Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

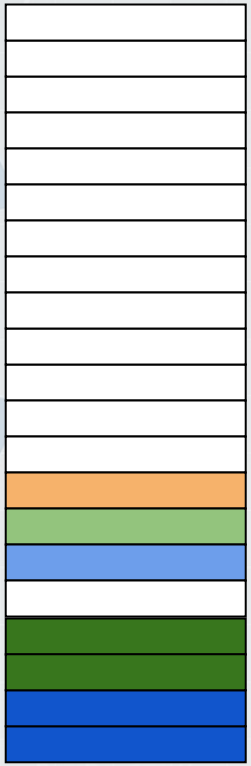
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

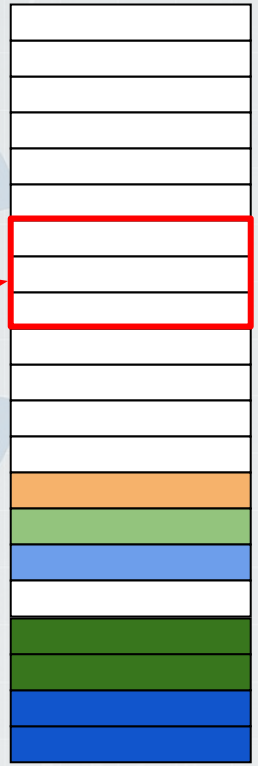
i=1, cols = 3



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

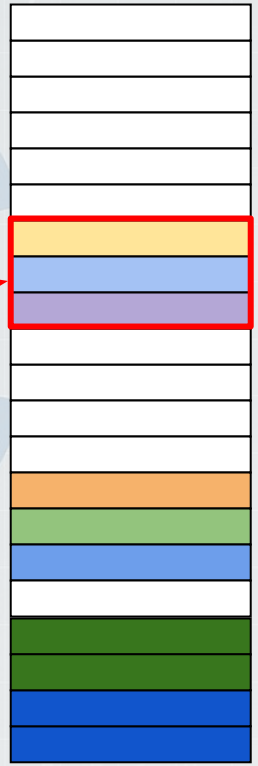
i=1, cols = 3



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

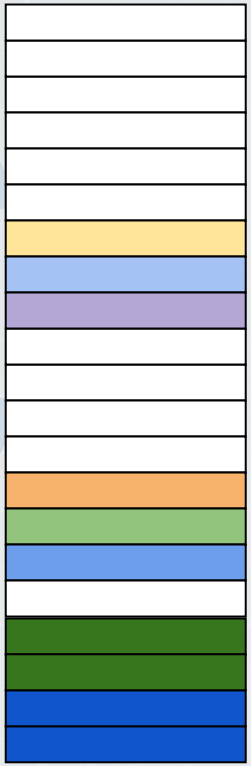
i=1, cols = 3



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

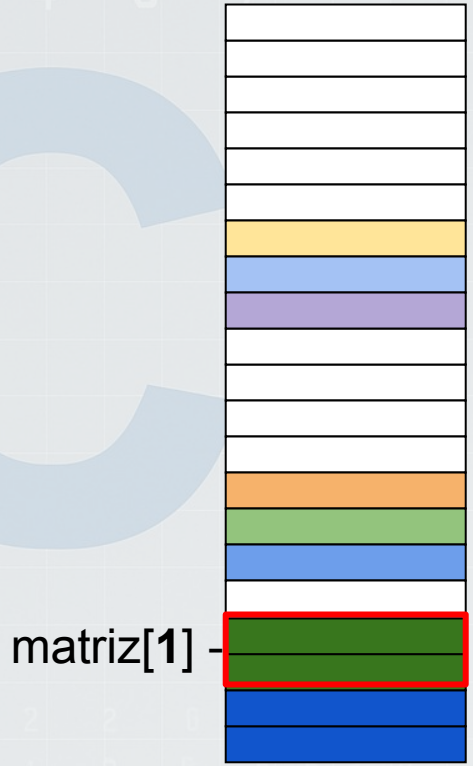
i=1, cols = 3, matriz[1]



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

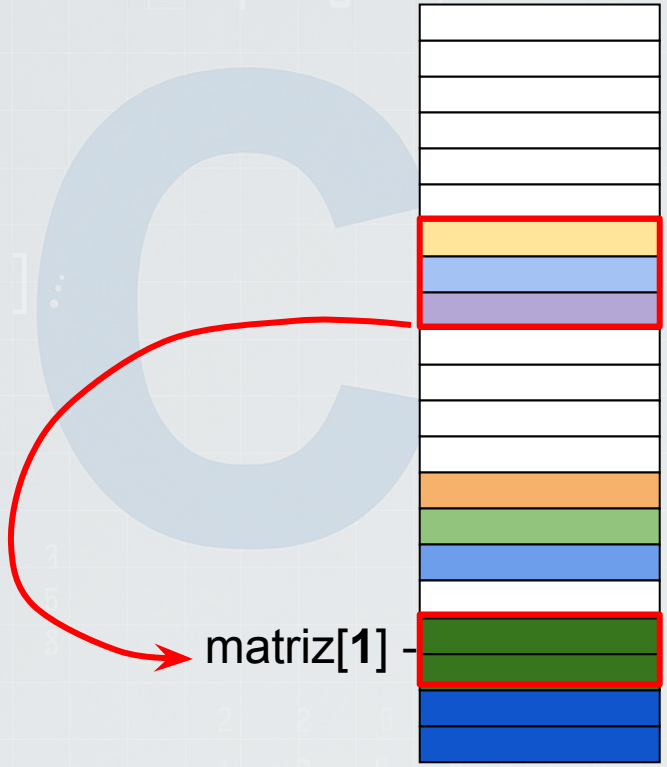
i=1, cols = 3, matriz[1]



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

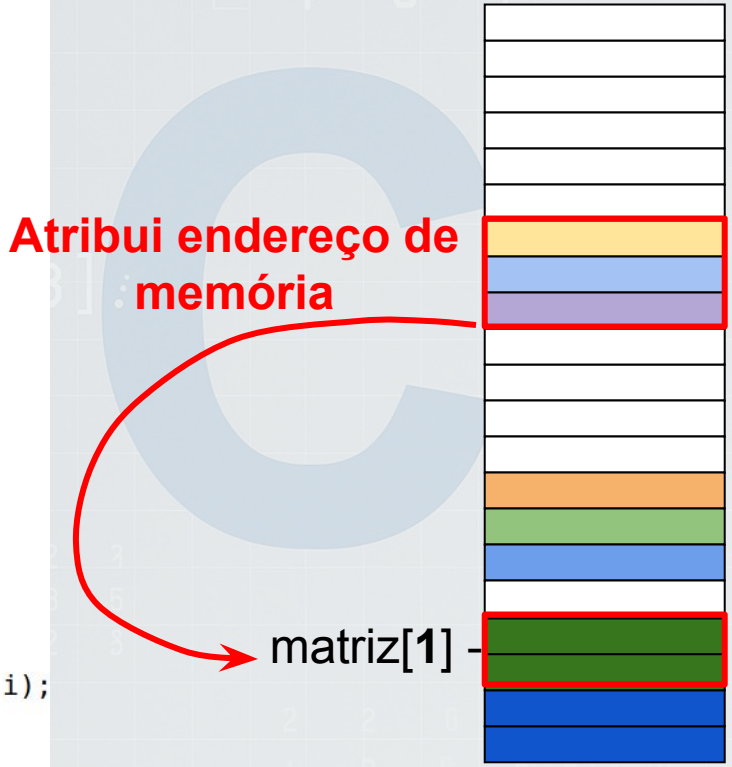
i=1, cols = 3, matriz[1]



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

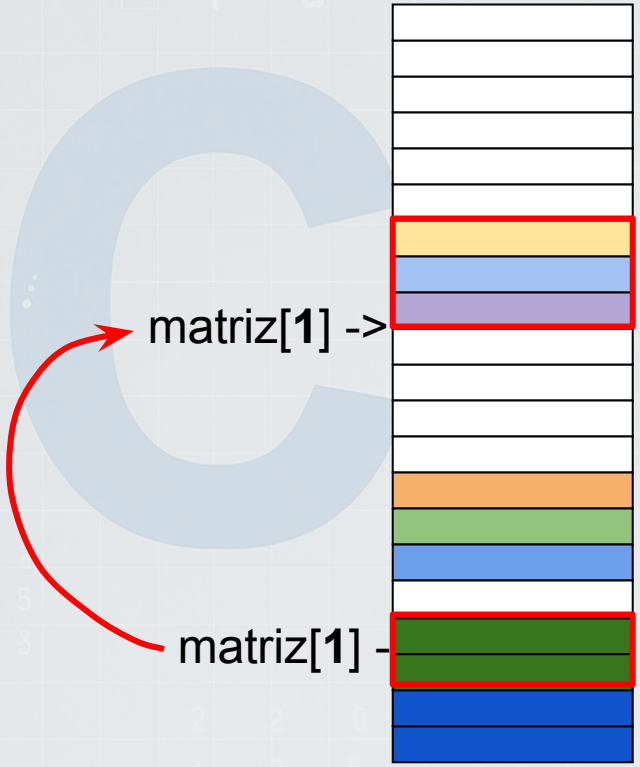
i=1, cols = 3, matriz[1]



Alocação dinâmica de matrizes

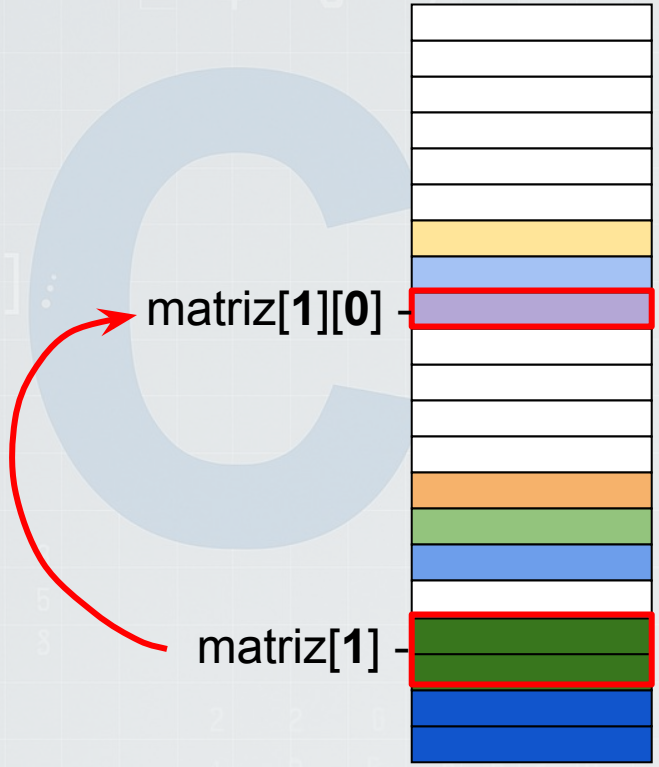
```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```

i=1, cols = 3, matriz[1]



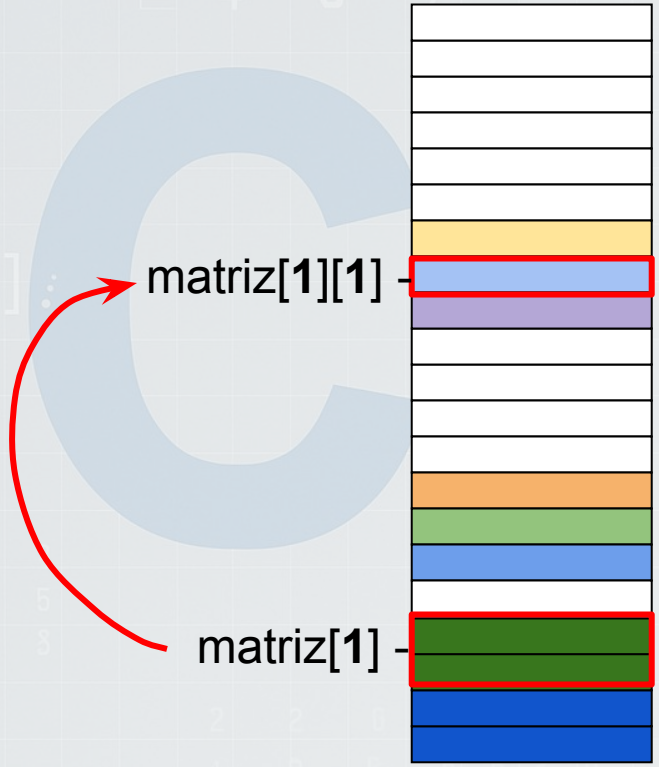
Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



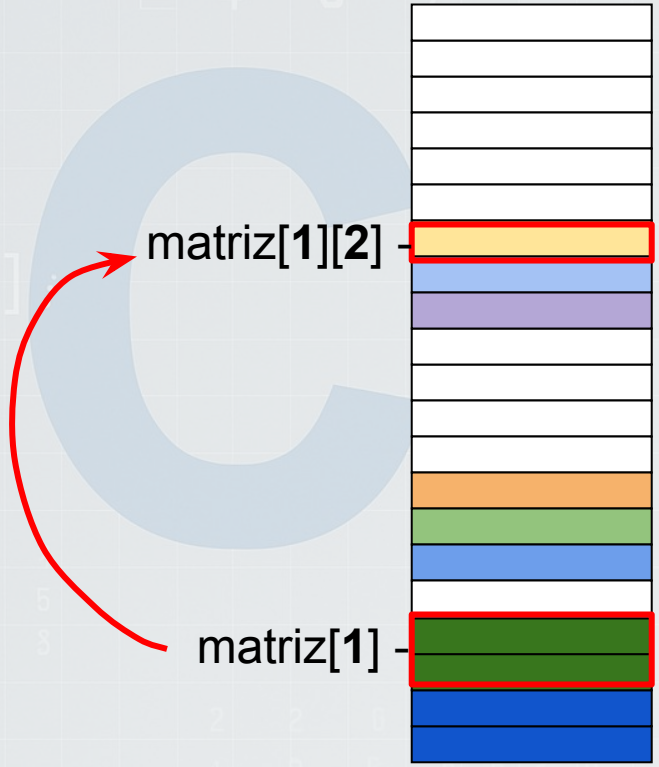
Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

```c
int main()
{
    int i, j;
    int rows = 2;
    int cols = 3;

    // Alocação da matriz: ponteiro para ponteiros
    int **matriz = (int **)malloc(rows * sizeof(int *));

    if (matriz == NULL)
    {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Alocar cada linha (vetor de colunas)
    for (i = 0; i < rows; i++)
    {
        matriz[i] = (int *)malloc(cols * sizeof(int));
        if (matriz[i] == NULL)
        {
            printf("Erro na alocação de memória da linha %d.\n", i);
            return 1;
        }
    }
```



Alocação dinâmica de matrizes

Implementar funções:

- alocar_matriz_real
- liberar_matriz_real

Alocação dinâmica de matrizes

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

float **alocar_matriz_real(int m, int n) {
    float **v;              /* ponteiro para a matriz */
    int i;                  /* variável auxiliar */

    if (m < 1 || n < 1) {       /* verifica parâmetros */
      printf ("** Erro: Parametro invalido **\n");
      return NULL;
    }
```

Alocação dinâmica de matrizes

```
    v = (float **) calloc(m, sizeof(float *));     /* aloca as linhas da matriz */
    if (v == NULL) {
       printf ("** Erro: Memoria Insuficiente **");
       return NULL;
    }
    for ( i = 0; i < m; i++ ) {                    /* aloca as colunas da matriz */
       v[i] = (float*) calloc(n, sizeof(float));
       if (v[i] == NULL) {
          printf ("** Erro: Memoria Insuficiente **");
          return NULL;
       }
    }
    return v;                          /* retorna o ponteiro para a matriz */
}
```

Alocação dinâmica de matrizes

```
float ** liberar_matriz_real (int m, int n, float **v) {
    int i;                          /* variável auxiliar */

    if (v == NULL) return NULL;

    if (m < 1 || n < 1) {           /* verifica parâmetros recebidos */
       printf ("** Erro: Parametro invalido **\n");
       return v;
    }
    for (i = 0; i < m; i++) free (v[i]); /* libera as linhas da matriz */

    free (v);                       /* libera a matriz */
    return NULL;                    /* retorna um ponteiro nulo */
}
```

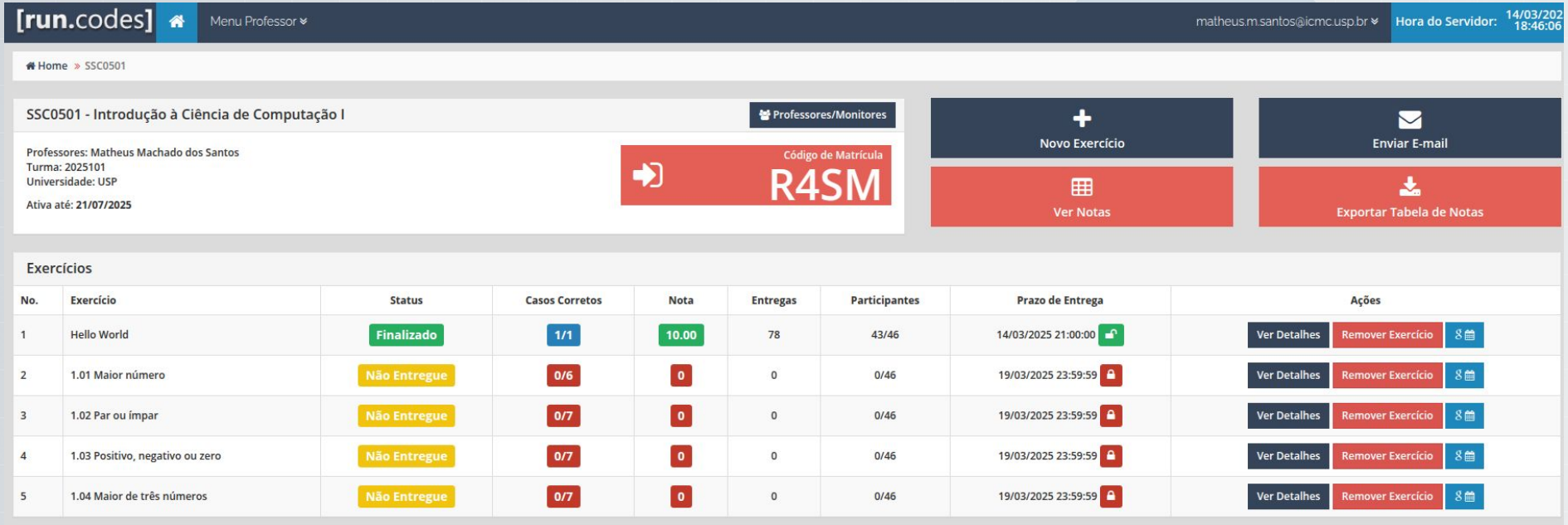
Alocação dinâmica de matrizes

```
void main () {
    float **mat;    /* matriz a ser alocada */
    int l, c;       /* numero de linhas e colunas */
    ...
    /* outros comandos, inclusive inicialização de l e c */
    mat = Alocar_matriz_real (l, c);
     /* outros comandos utilizando mat[][] normalmente */
    if (Liberar_matriz_real (l, c, mat) != NULL) {
       printf("Erro ao desalocar a matriz!\n");
       return ;
    }
    ...
}
```

Prática!

Código Matricula: R4SM

https://runcodes.icmc.usp.br/offerings/view/83



SSC0501 - Introdução à Ciência de Computação I

Obrigado pela atenção!!