Programação Procedimental

Alocação dinâmica

Aula 08

Prof. Felipe A. Louza

Roteiro

Alocação estática de memória

As declarações abaixo alocam/reservam espaço na memória para variáveis definidas em tempo de compilação.

```
1   char c;
2   int i;
3   int v[10];
```

- A alocação é estática, acontece antes que o programa comece a ser executado.

Alocação dinâmica de memória

Em muitas aplicações, a quantidade de memória necessária só é conhecida durante a execução do programa.

```
1  ...
2  int n, v[100];
3  printf("Digite o tamanho do vetor: ");
4  scanf("%d", &n);
5  ...
6  int i;
7  for(i=0; i<n; i++)
8    scanf("%d", &v[i]);
```

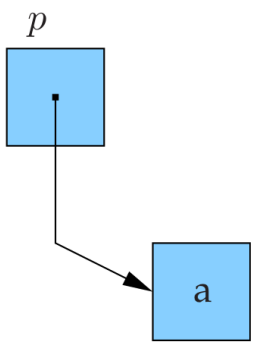
- A alocação dinâmica de memória permite resolver essa situação.

Alocação dinâmica de memória

A função `malloc`:

- Aloca um bloco consecutivos de bytes em memória e devolve o endereço desse bloco.

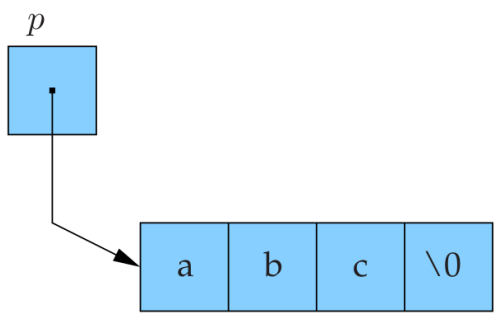
```
1   char *p;
2   p = (char*) malloc(1);
3   scanf ("%c", p);
```



Alocação dinâmica de memória

Podemos alocar vetores com a função `malloc`:

```
1  char *p;
2  p = (char*) malloc(4);
3  p[0] = 'a';
4  p[1] = 'b';
5  p[2] = 'c';
6  p[3] = '\0';
```



Alocação dinâmica de memória

A função `malloc`:

- Faz parte da biblioteca `<stdlib.h>`

```
1  #include <stdlib.h>
2  ...
3  char *p;
4  p = (char*) malloc(4);
5  ...
```

- O número de bytes é especificado no argumento da função.
- O endereço devolvido por malloc é do tipo genérico `void *`.

Alocação dinâmica de memória

A função `malloc`:

- A função devolve `NULL` caso aconteça algum erro.
- Por exemplo, se a memória do computador já estiver toda ocupada, `malloc` não consegue alocar mais espaço.

```
1   char *p;
2   p = (char*) malloc(10000000);
3   if(p == NULL) exit(EXIT_FAILURE);
4   ...
```

Alocação dinâmica de memória

Exemplo:

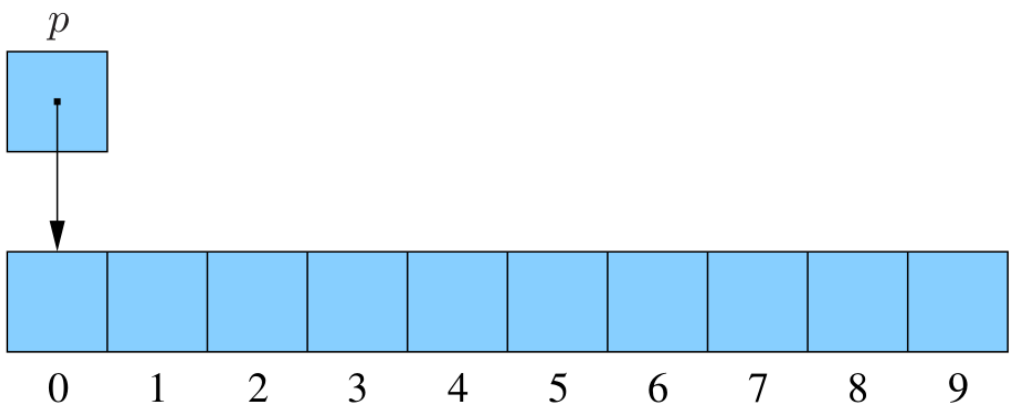
```
1  #include <stdlib.h>
2
3  int main(){
4
5    char *p;
6    p = (char*) malloc(100*1);
7
8    int *q = (int*) malloc(100*4); //cada inteiro utiliza 4 bytes
9
10 return 0;
11 }
```

Alocação dinâmica de memória

O operador `sizeof`:

- Retorna quantos bytes um tipo de dados ocupa.

```
1   ...
2   int *p = (int*) malloc(10* sizeof(int) );
3   ...
```



Alocação dinâmica de memória

Exemplo:

```
1  #include <stdlib.h>
2
3  typedef struct{
4    int dia, mes, ano;
5  } t_data;
6
7  int main(){
8
9    char *p;
10   p = (char*) malloc(100*sizeof(char));
11
12   int *q = (int*) malloc(100*sizeof(int));
13
14   t_data *r = (t_data*) malloc(100*sizeof(t_data));
15
16 return 0;
17 }
```

Alocação dinâmica de memória

Importante:

- As variáveis **estáticas** "desaparecem" automaticamente assim que a execução da função termina.
- Já as variáveis **alocadas dinamicamente** continuam a existir mesmo depois que a função termina.

```c
void funcao1(){
  int a=1, b=2, c=3;
  int vetor[100];
  ...
return;
}
```

```c
void funcao2(){
  int *p;
  p = (int*) malloc(100*sizeof(int));
  ...
return;
}
```

Alocação dinâmica de memória

A função `free`:

- Desaloca/libera a porção de memória alocada por `malloc`.

```
1 void funcao2(){
2   int *p;
3   p = (int*) malloc(100*sizeof(int));
4   ...
5   free(p);
6 return;
7 }
```

- Quando fazemos alocação dinâmica, é **nossa responsabilidade** liberar a memória.

Roteiro

Alocação dinâmica de memória

Não alocar a memória antes de usar:

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

int main(){

  int *p;
  int i;
  for(i = 0; i<100; i++) p[i]=i;

  for(i = 0; i<100; i++) printf("%d\n", p[i]);

return 0;
}
```

- Erro em tempo de execução:

```
Segmentation fault (core dumped)
```

Alocação dinâmica de memória

Não desalocar a memória:

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

int main(){

  int *p = (int*) malloc(100*sizeof(int));
  int i;
  for(i = 0; i<100; i++) q[i]=i;

  for(i = 0; i<100; i++) printf("%d\n", q[i]);
  //free ??
return 0;
}
```

- Não ocorre um erro, mas está **conceitualmente errado**.

```
10
```

Alocação dinâmica de memória

Perder o endereço/referência da memória alocada:

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

int main(){

  int *p = (int*) malloc(100*sizeof(int));
  int i;
  for(i = 0; i<100; i++) p[i]=i;

  p = (int*) malloc(100*sizeof(int));
  for(i = 0; i<100; i++) p[i]=i*2;

return 0;
}
```

- Não ocorre um erro, mas está **conceitualmente errado**.

Alocação dinâmica de memória

Desalocar mais de uma vez a mesma região:

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

int main(){

  int *p = (int*) malloc(100*sizeof(int));
  ...

  free(p);
  free(p);
  ...
return 0;
}
```

- Erro em tempo de execução:

```
free(): double free detected in tcache 2
Aborted (core dumped)
```

Alocação dinâmica de memória

Outro exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5
6    int *p = (int*) malloc(100*sizeof(int));
7    int *q =  p;
8    ...
9
10   free(p);
11   ...
12   free(q); // melhor q = NULL;
13   ...
14 return 0;
15 }
```

Alocação dinâmica de memória

Utilizar uma região de memória já desalocada:

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

int main(){

  int *p = (int*) malloc(100*sizeof(int));
  int *q =  p;

  free(p);
  q[0] = 10;

  int i;
  for(i = 0; i<100; i++) printf("%d\n", q[i]);

return 0;
}
```

- Erro em tempo de execução: ??

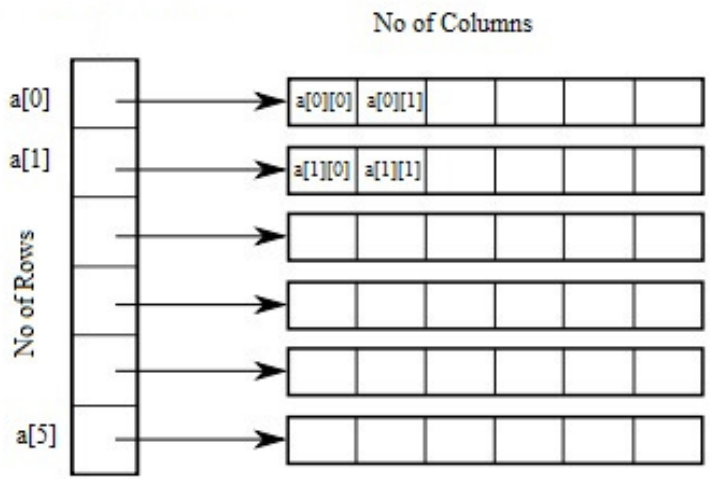
```
10
```

Roteiro

Alocação Dinâmica de Matrizes

Matrizes dinâmicas são implementadas como vetores de vetores.

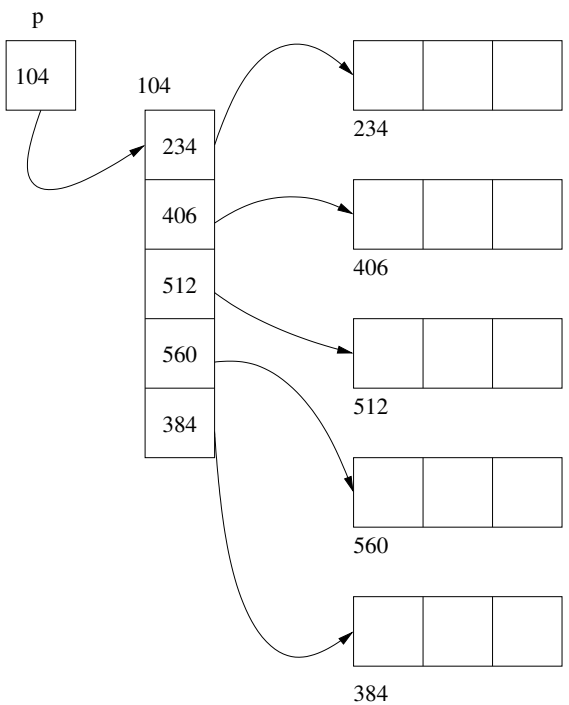
1. Criamos um vetor de ponteiros
2. Para cada ponteiro, criamos um vetor.



Alocação Dinâmica de Matrizes

Exemplo: $M_{5\times 3}$

```
1   int **p;
2   int i;
3   p = (int**) malloc(5*sizeof(int*));
4   for(i=0; i<5; i++)
5     p[i] = malloc(3 * sizeof(int));
```



Alocação Dinâmica de Matrizes

Podemos acessar posição $M_{i,j}$ com o operador `p[i][j]`:

```
1   ...
2   p[1][1] = 1;
3   p[2][1] = 2;
4   p[3][1] = 3;
```

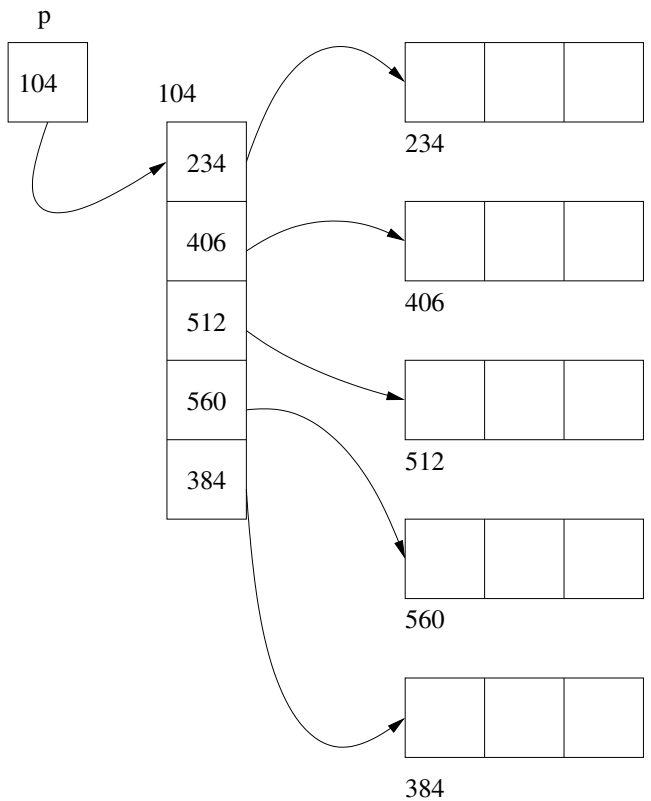
	0	1	2
0			
1		1	
2		2	
3		3	
4			

Alocação Dinâmica de Matrizes

Para desalocar uma matriz com a função `free()`:

- Primeiro desalocamos as linhas:

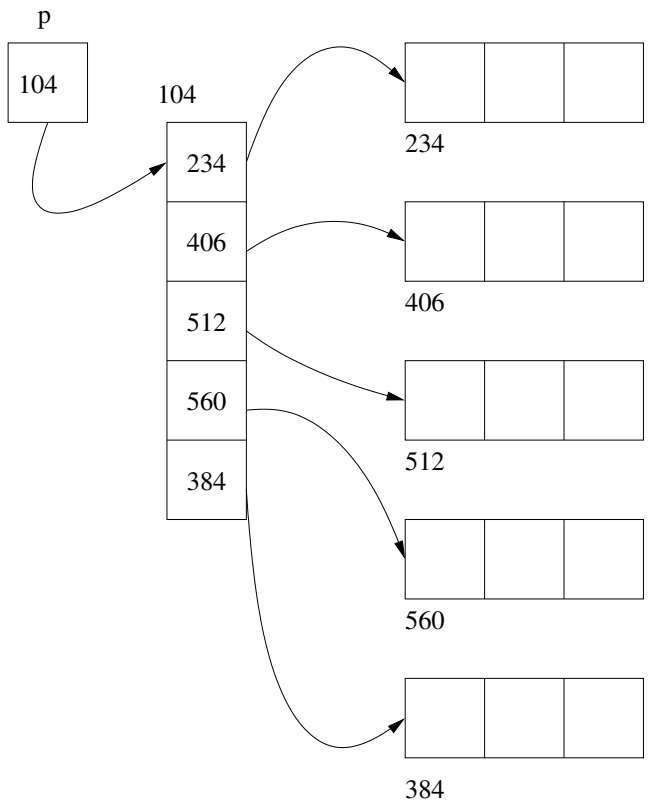
```
1 //n é o número de linhas
2 void liberaMatriz(int **M, int n){
3   int i;
4   for (int i = 0; i < n; i++)
5     free(M[i]);
6   free(M);
7 }
```



Erros Comuns ao Usar Alocação Dinâmica

Erro comum:

```
1 //n é o número de linhas
2 void liberaMatriz(int **M, int n){
3   int i;
4   free(M);
5   for (int i = 0; i < n; i++)
6     free(M[i]);
7 }
```



Roteiro

Redimensionamento de memória

Pode ser necessário alterar, durante a execução do programa, o tamanho de um bloco de bytes que foi alocado.

- Por exemplo, durante a leitura de um arquivo que se revela maior que o esperado.

Redimensionamento de memória

A função `realloc`:

- Recebe o endereço de um bloco previamente alocado e o número de bytes que o bloco redimensionado deve ter.

```
1    int *p = (int*) malloc (10*sizeof(int));
2    ...
3    p = realloc(p, 20*sizeof(int));
4    ...
```

  - Se `p==NULL`, um novo bloco é alocado.

Redimensionamento de memória

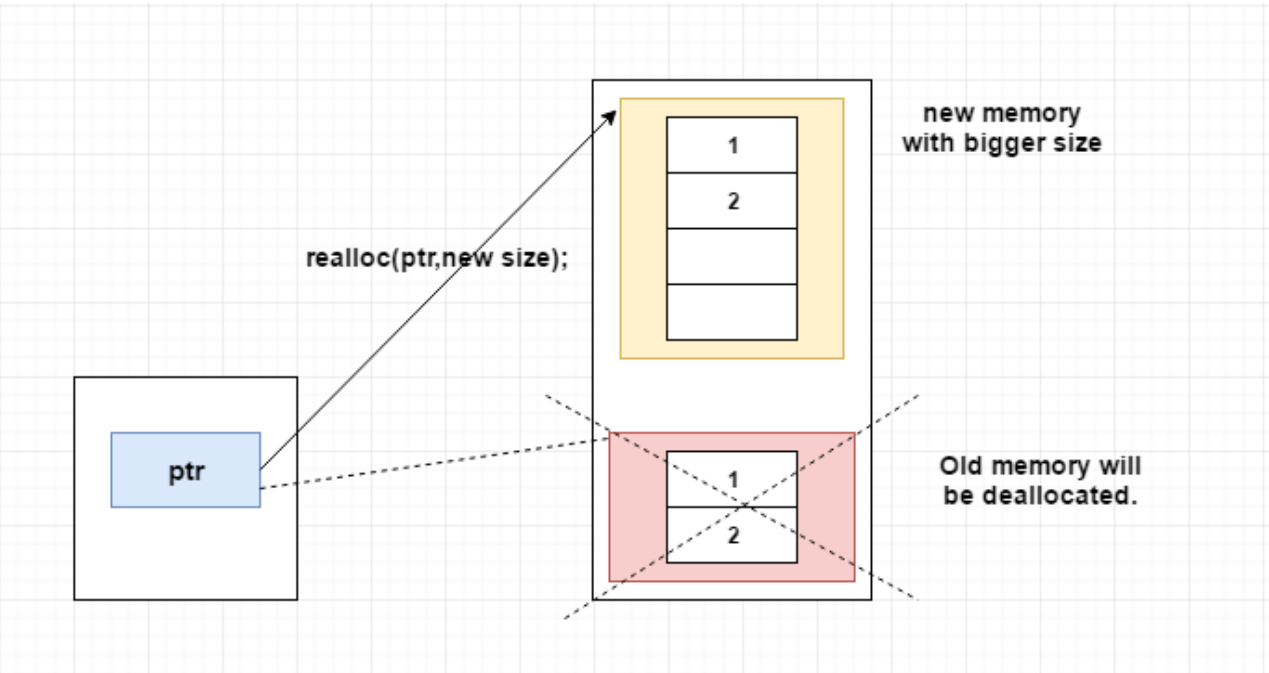
Exemplo:

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

int main(){

  int *p = (int*) malloc(2*sizeof(int));
  p[0] = 10;
  p[1] = 20;

  int *q = (int*) realloc(ptr, 3*sizeof(int));
  q[2] = 30;

  int i;
  for(i = 0; i < 3; i++)
    printf("%d ", q[i]);

return 0;
}
```

Redimensionamento de memória

Alguns detalhes:

- Se o novo bloco for uma extensão do original, seu endereço é o mesmo (e o conteúdo não precisa ser copiado).
- Caso contrário, `realloc` copia o conteúdo do bloco original para o novo e libera o bloco original (invocando `free`).



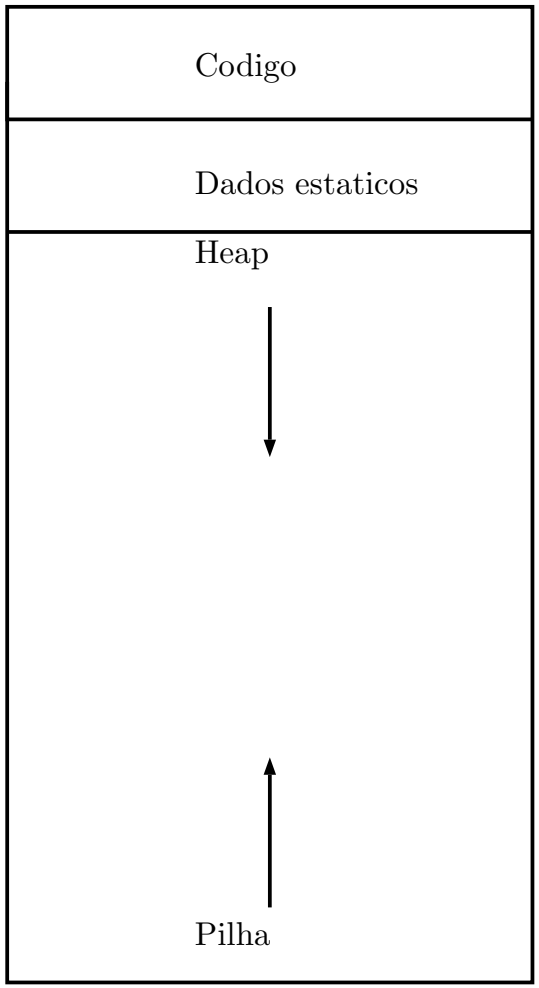
Roteiro

Organização da Memória do Computador

A memória do computador durante a **execução de um programa** é organizada em quatro segmentos:

- **Código executável:** Contém o código binário do programa.
- **Dados estáticos:** Contém variáveis globais e estáticas que existem durante toda a execução do programa.
- **Pilha:** Contém as variáveis locais que são criadas na execução de uma função e depois são removidas da pilha ao término da função.
- **Heap:** Contém as variáveis criadas por alocação dinâmica.

Organização da Memória do Computador



Organização da Memória do Computador

Em C99 podemos declarar vetores de tamanho variável em tempo de execução, com o valor de uma variável.

- Abaixo declaramos o vetor `v[n]`, com `n` lido do teclado.

```c
#include <stdio.h>

int main(){
  int n, i;
  printf("Digite o tamanho do vetor:");
  scanf("%d", &n);
  int v[n]; //Vetor alocado com tamanho n não pré-estabelecido

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

return 0;
}
```

Organização da Memória do Computador

Porém a criação de vetores desta forma faz a alocação de memória na área Pilha que possui um limite máximo.

- Execute o programa digitando 1000000 e depois 4000000.

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

int main(){
  int n, i;
  printf("Digite o tamanho do vetor:");
  scanf("%d", &n);
  int v[n]; //Vetor alocado com tamanho n não pré-estabelecido

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

return 0;
}
```

Organização da Memória do Computador

O programa anterior será encerrado **(segmentation fault)** se for usado um valor grande o suficiente para n.

- Isto se deve ao fato de que o **SO limita o que pode ser alocado** na **Pilha** durante a execução de uma função.
- Este limite não existe para o **Heap** (com exceção do limite de memória do computador).

Organização da Memória do Computador

Utilizando alocação dinâmica não temos o problema de erro do programa anterior.

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

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

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

  free(v);

return 0;
}
```

Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados da apostila de Programação de Computadores do Prof. Martinez, da UFMS.