Computação eletrônica: Alocação dinâmica de memória

Gurvan Huiban
ghuiban@cin.ufpe.br

21 de janeiro de 2014

Plano de aula

Problema

Escrever um programa que armazena num vetor dados. A quantidade de dados a inserir é desconhecida, e pode ser muito grande ou bastante pequena.

Tamanho do vetor?

- Pequeno: Limita bastante o programa
- Grande: Usa muita memória, mesmo sem necessidade

Como fazer?

Alocar o vetor *durante* a execução do programa

Memória como gaveteiro (continuação)...

Alocação estática de memória

- Declarar uma variável é “colar” uma etiqueta numa(s) gaveta(s).
- A etiqueta fica colada na gaveta durante a execução inteira do programa.

Alocação dinâmica de memória

Colar e remover etiquetas das gavetas durante a execução do programa.

O comando `malloc`

```
#include<stdlib.h>
void *malloc(size_t size)
```

- Aloca *durante* a execução a quantidade de memória pedida (em bytes)
- Retorna um ponteiro genérico sobre o início do espaço alocado
- Retorna `NULL` em caso de problema

Observações

- Converter o ponteiro da forma seguinte (*cast*):

  ```
  (tipo*) malloc(tamanho);
  ```

- A memória não é inicializada
- Verificar a alocação

Alocação dinâmica de um inteiro

```c
int *n;
n = (int*) malloc(4);
if (n == NULL)
{
   printf("Erro durante alocacao de memoria\n");
}
else
{
  *n = 12;
}
```

O comando free

Liberação de memória dinamicamente alocada:

- Fim do programa
- Comando free

```
#include<stdlib.h>
void free(void *ptr)
```

- Libera a memória apontada por ptr

Observações:

- ptr continua com o mesmo endereço
- Recomendado atribuir NULL ao ponteiro

Cuidados com free

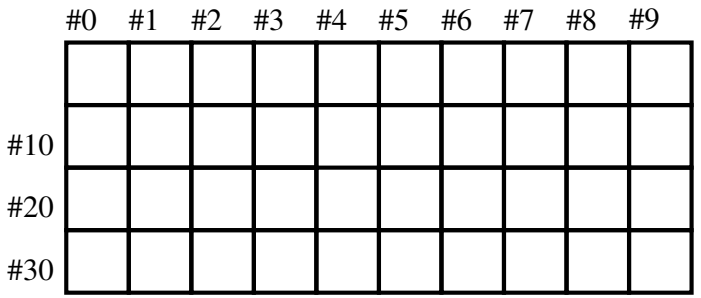
Não liberar memória alocada estaticamente

```
int n;
free(&n); // Errado: Memoria alocada estaticamente
```

Não liberar duas vezes o mesmo elemento

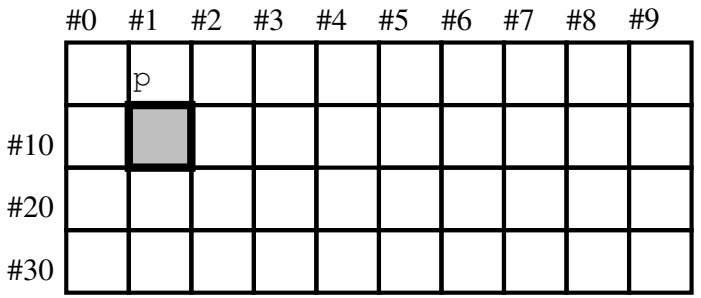
```
int *n;
n = (int*)malloc(4);
free(n);
free(n); // Errado: Memoria ja liberada
```

```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



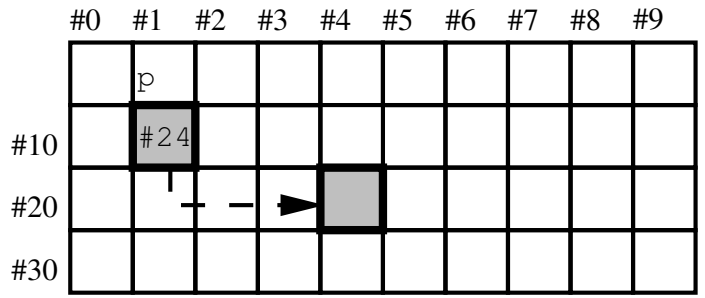
Tela

```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



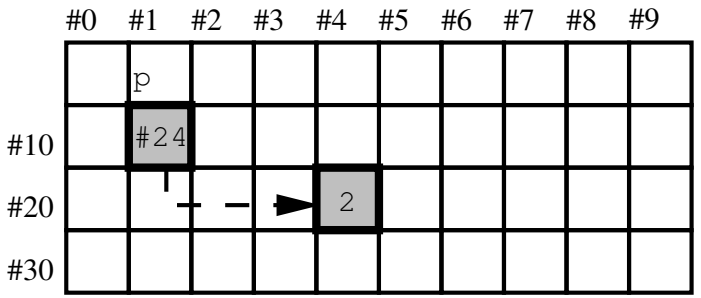
Tela

```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



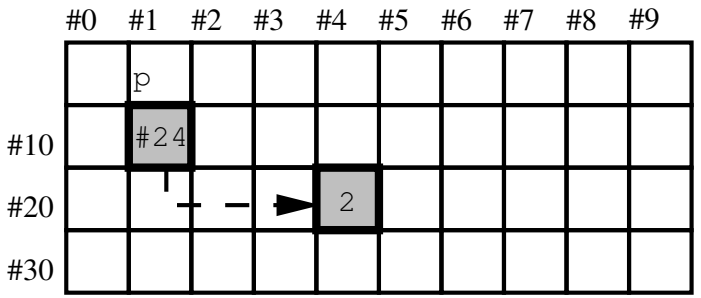
Tela

```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



Tela

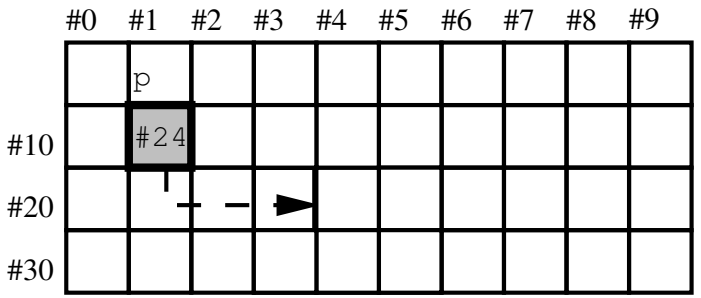
```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



Tela

```
2
```

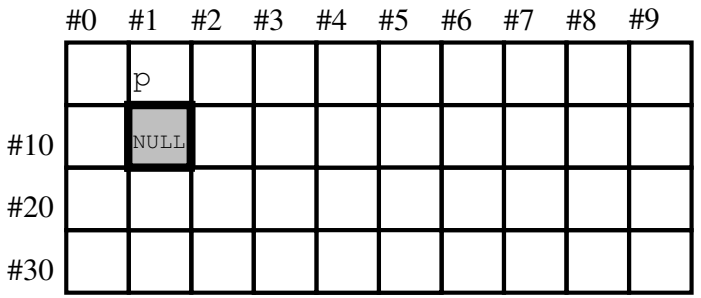
```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



Tela

2

```
int *p;
p = (int*) malloc(4);
*p = 2;
printf("%d\n", *p);
free(p);
p = NULL; // Recomendado
```



Tela

2

O comando `sizeof`

Tamanho das variáveis/dos tipos

- Potencialmente difíceis de calcular (estrutura)
- Dependência da arquitetura do computador

```
size_t sizeof nome;
size_t sizeof(int);
```

Retorna o tamanho (em bytes) da variável/do tipo no computador

Exemplo de uso do `sizeof`

```
typedef struct {
    char nome[250];
    int mat;
    char turma[64];
} TAluno;
int main(void) {
    TAluno *a;
    printf("Tamanho de um int=%ld\n", sizeof(int));
    printf("Tamanho de um TAluno=%ld\n", sizeof(TAluno));
    a = (TAluno*) malloc(sizeof(TAluno));
    a->mat = 123;
    ...
}
```

Tela

```
Tamanho de um int=4
Tamanho de um TAluno=320
```

Alocação dinâmica de vetor unidimensional

Vetor unidimensional

- Sequência de `n` elementos do mesmo tipo
- Um ao lado do outro na memória

Alocação de um vetor

Alocar um espaço de `n` vezes o tamanho do tipo:

```
tipo *p;
p = (tipo*)malloc(n*sizeof(tipo));
```

Exemplo de alocação dinâmica de vetor

```
int i,n, *p;
printf("Tamanho do vetor? ");
scanf("%d", &n);
p = (int*) malloc(n*sizeof(int));
if (p == NULL)
{
   printf("Erro na alocacao de memoria");
}
else
{
   for (i=0; i<n; i++)
   {
      p[i] = i;
   }
   ...
}
```