BCC202 - Estrutura de Dados I

Aula 02: Alocação Dinâmica de Memória

Reinaldo Fortes

Universidade Federal de Ouro Preto, UFOP
Departamento de Ciência da Computação, DECOM

Website: **www.decom.ufop.br/reifortes**
Email: **reifortes@iceb.ufop.br**

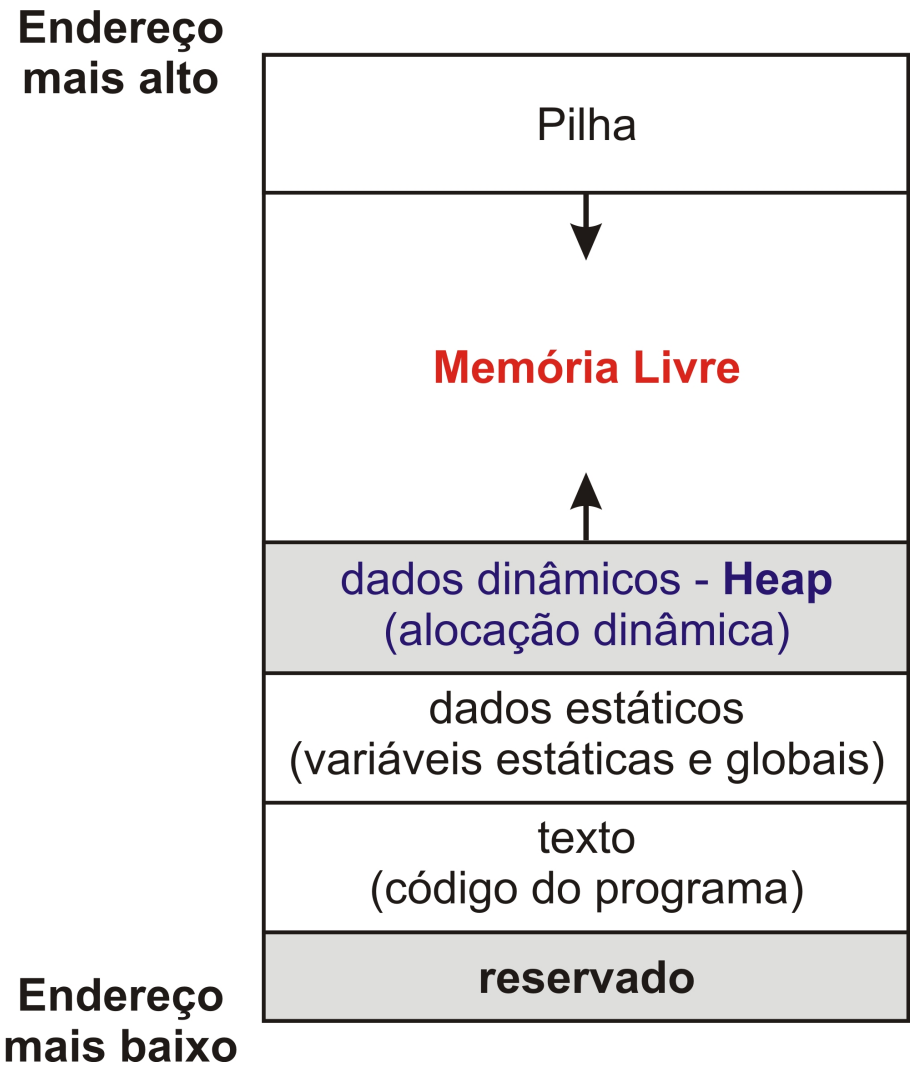
Material elaborado com base nos slides do Prof. Túlio Toffolo (curso de 2013/01).

2013/02

Conteúdo

Conteúdo

Esquema de alocação de memória do sistema



Conteúdo

Comparativo entre Alocação Dinâmica e Alocação Estática

Alocação Estática

- O espaço para as variáveis é reservado e liberado automaticamente pelo compilador.
  - Exemplo:

```
1 int a; int b[20];
```

Alocação Dinâmica

- O espaço para as variáveis é reservado e liberado dinamicamente pelo programador.
  - Exemplo:

```
1 int *a = (int*) malloc(sizeof(int));
```

Conteúdo

Exemplo 1

```
1  #include <stdio.h>
2
3  void quad(int n) {
4    n = n * n;
5    printf("n = %d\n", n);
6  }
7
8  int main() {
9    int n;
10   n = 3;
11   quad(n);
12   printf("n = %d\n", n);
13   return 0;
14 }
```

Exemplo 2

```c
#include <stdio.h>

void quad(int n) {
  n = n * n;
  printf("n = %d\n", n);
}

int main() {
  int n;
  scanf("%d", &n);
  if(n > 10) {
    int x = 10;
    quad(x);
  } else quad(n);
  printf("n = %d\n", n);
  return 0;
}
```

Conteúdo

Conceitos de ponteiros e memória *heap*

Ponteiros

- Variáveis alocadas dinamicamente são chamadas de ponteiros ou apontadores (pointers), pois armazenam o endereço de memória de uma variável.
  - Número inteiro (32 ou 64 bits) indicando um endereço de memória.

Memória *Heap*

- A memória alocada dinamicamente faz parte de uma área da memória chamada heap.
  - Basicamente, o programa aloca e desaloca porções de memória do heap durante a execução.

Liberação de memória

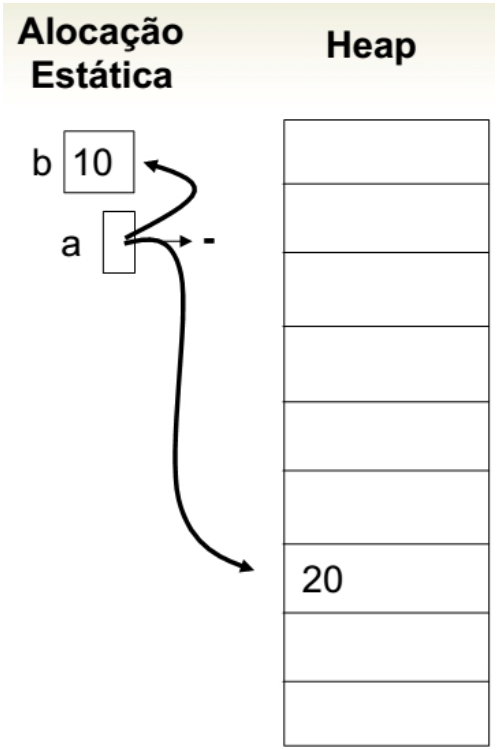
- A memória deve ser liberada após o término de seu uso.
- Este trabalho deve ser feito por quem fez a alocação:
  - **Alocação Estática**: compilador.
  - **Alocação Dinâmica**: programador.

Código de Alocação Dinâmica *vs.* Estática

- Declaração de variável:
  - Tipo *p; *vs.* Tipo p;
- Alocação de memória (apenas dinâmica):
  - p = (Tipo*) malloc(sizeof(Tipo));
- Liberação de memória (apenas dinâmica):
  - free(p)
- Conteúdo da variável:
  - *p; *vs.* p;
- Referência à variável (seu endereço de memória):
  - p; *vs.* &p;
- Valor nulo para um ponteiro (apenas dinâmica):
  - NULL

Endereçamento de Alocação Dinâmica *vs.* Estática

```
1 int *a, b;
2 :
3 :
4 b = 10;
5 a = (int*) malloc(sizeof(int));
6 a = &b;
```



Erros Comuns

- Esquecer de alocar memória e tentar acessar o conteúdo da variável.
- Copiar o valor do ponteiro ao invés do valor da variável apontada.
- Esquecer de desalocar memória.
  - A memória será desalocada apenas no encerramento do programa, o que pode ser um grande problema em loops.
  - Ocasiona "desperdício de memória", que pode causar falha do sistema.
- Tentar acessar o conteúdo da variável depois de desalocá-la.

Ponteiros *vs.* Vetores

int *a não é a declaração de um vetor de inteiros?

- Em C, todo ponteiro pode se comportar como um vetor.
- Portanto, pode-se fazer coisas do tipo:

```
1 int a[10], *b;
2 b = a;
3 b[5] = 100;
4 printf("%d\n", a[5]);
5 printf("%d\n", b[5]);
```

```
100
100
```

```
1 int a[10], *b;
2 b = (int*) malloc(
3       10*sizeof(int));
4 b[5] = 100;
5 printf("%d\n", a[5]);
6 printf("%d\n", b[5]);
```

```
42675
100
```

***OBS**: Não é permitido fazer `a = b` nestes exemplos.*

Ponteiros para Tipos Estruturados

```
1  typedef struct{
2    int idade;
3    double salario;
4  } TRegistro;
5
6  int main() {
7    TRegistro *a;
8    ...
9    a = (TRegistro*) malloc(sizeof(TRegistro))
10   a->idade = 30; // (*a).idade = 30
11   a->salario = 80;
12   ...
13 }
```

Passagem de Parâmetros

- Em C só existe passagem de parâmetro por valor, ou seja, é sempre criada uma variável como uma cópia.
- Logo, deve-se implementar a passagem por referência, utilizando-se de ponteiros.
- Cuidado com a alocação de mémória!

Passagem de Parâmetros - Alocação de Memória

```
1  typedef struct{
2    int idade;
3    double salario;
4  } TRegistro;
5
6  void novoTRegistro(TRegistro *t) {
7    t = (TRegistro *) malloc(sizeof(TRegistro));
8  }
9
10 int main() {
11   TRegistro *a;
12   ...
13   novoTRegistro(a);
14   a->idade = 30;
15   a->salario = 80;
16   ...
17 }
```

Onde está o ERRO? **Alocando memória para a cópia de `*t`.**

Passagem de Parâmetros - Alocação de Memória

```
1  typedef struct{
2    int idade;
3    double salario;
4  } TRegistro;
5
6  TRegistro* novoTRegistro() {
7    TRegistro *t = (TRegistro *) malloc(sizeof(TRegistro));
8    return t;
9  }
10
11 int main() {
12   TRegistro *a;
13   ...
14   a = novoTRegistro();
15   a->idade = 30;
16   a->salario = 80;
17   ...
18 }
```

Agora sim! Qual a diferença? **Retornando o ponteiro** `t`.

Passagem de Parâmetros - Valor *vs.* Referência

```
1  void SomaUm(int x, int *y) {
2    x = x + 1;
3    *y = (*y) + 1;
4    printf("Funcao SomaUm: %d %d\n", x, *y);
5  }
6
7  int main() {
8    int a = 0, b = 0;
9    SomaUm(a, &b);
10   printf("Programa principal: %d %d\n", a, b);
11 }
```

Funcao SomaUm: 1 1
Programa Principal: 0 1

Passagem de Parâmetros - Valor *vs.* Referência

- E para alocar memória dentro de um procedimento?
  - Em pascal ou C++ existe a passagem por referência.
  - Em C não existe: usa-se ponteiros.

```
1  void aloca(int *x, int n) {
2    x = (int*)malloc(
3          n*sizeof(int));
4    x[0] = 20;
5  }
6
7  int main() {
8    int *a;
9    aloca(a, 10);
10   a[1] = 40;
11 }
```

ERRO: Access Violation!

```
1  void aloca(int **x, int n) {
2    (*x) = (int*)malloc(
3             n*sizeof(int));
4    (*x)[0] = 20;
5  }
6
7  int main() {
8    int *a;
9    aloca(&a, 10);
10   a[1] = 40;
11 }
```

OK: (*x) é um ponteiro!

Conteúdo

Conclusão

- Nesta aula foram apresentados conceitos e exemplos de alocação dinâmica de mémoria.
- Pontos de maior atenção para alocação e liberação de memória e passagem de parâmetros por valor e referência.
- Em seus programas você deverá estar atento ao momento exato de liberação da memória.
  - Os enunciados das tarefas da disciplina geralmente solicitam que TODA a memória alocada seja liberada.
- *Próxima aula*: Tipos Abstratos de Dados (TADs).
- Dúvidas?

Conteúdo

Exercício 01

Faça um programa que leia um valor `n`, crie dinamicamente um vetor de `n` elementos inteiros e passe esse vetor para uma função que deverá preenchê-lo com valores digitados pelo usuário.

A leitura do vetor deve ser feita via `scanf`.

Importante: não se esqueça de liberar a memória alocada para o vetor.

Exercício 02

Faça um programa semelhante ao anterior, mas agora os elementos são do tipo `TRegistro*`, definido no Slide 17.

A leitura dos valores das identidades e do salário devem ser feitas via `scanf`.

Importante: não se esqueça de liberar a memória alocada. Note que agora, além de alocar memória para o vetor, você também deverá alocar memória para cada um de seus elementos. Desta forma, será necessário liberar toda a memória alocada para os elementos do vetor, para em seguida, desalocar a memória do vetor.