Instituto Federal Sudeste de Minas Gerais
Campus Barbacena

Curso de Tecnologia em Sistemas para Internet

Disciplina: Estruturas de Dados I

Prof. Wender Magno Cota

Assunto:Alocação Dinâmica de Memória

Alocação Estática x Dinâmica

- C: dois tipos de alocação de memória: **Estática e Dinâmica**

- Na alocação estática, o espaço para as variáveis é reservado no início da execução, não podendo ser alterado depois
  - int a; int b[20];

- Na alocação dinâmica, o espaço para as variáveis pode ser alocado dinamicamente durante a execução do programa

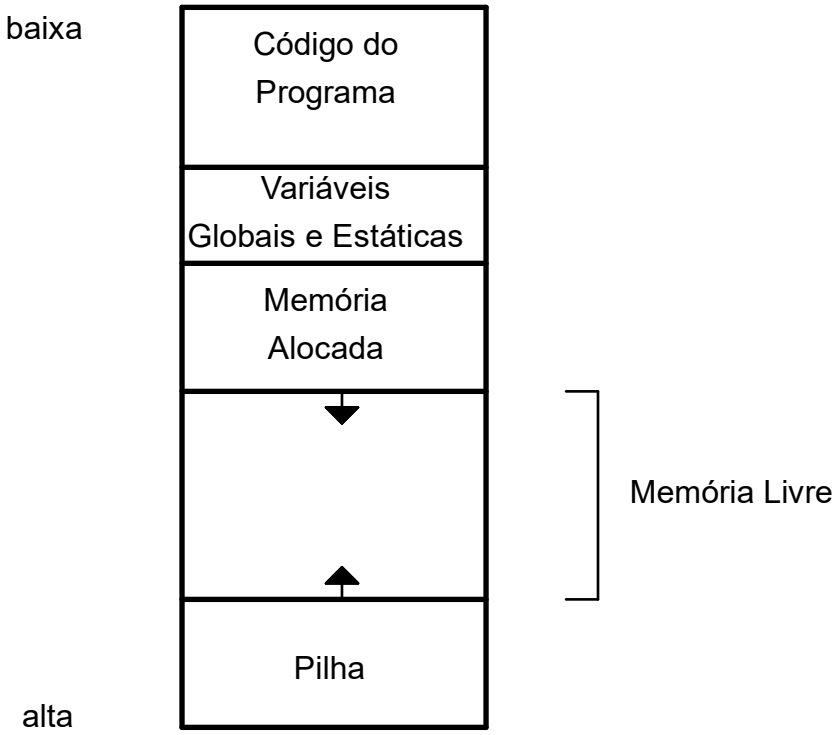
Alocação Dinâmica

- As variáveis alocadas dinamicamente são chamadas de **Apontadores** (*ponteiros*) pois na verdade elas armazenam o endereço de memória de uma variável. Seu domínio são endereços de memória.

- A memória alocada dinamicamente faz parte de uma área de memória chamada ***heap***

- Basicamente, o programa aloca e desaloca porções de memória do heap durante a execução

Esquema de Memória



Esquema da memória do sistema

Acesso a partir de Apontadores

- Acessar o valor da variável: endereço de memória armazenado
- Acessar o conteúdo que associado ao endereço de memória armazenado

Liberação de Memória

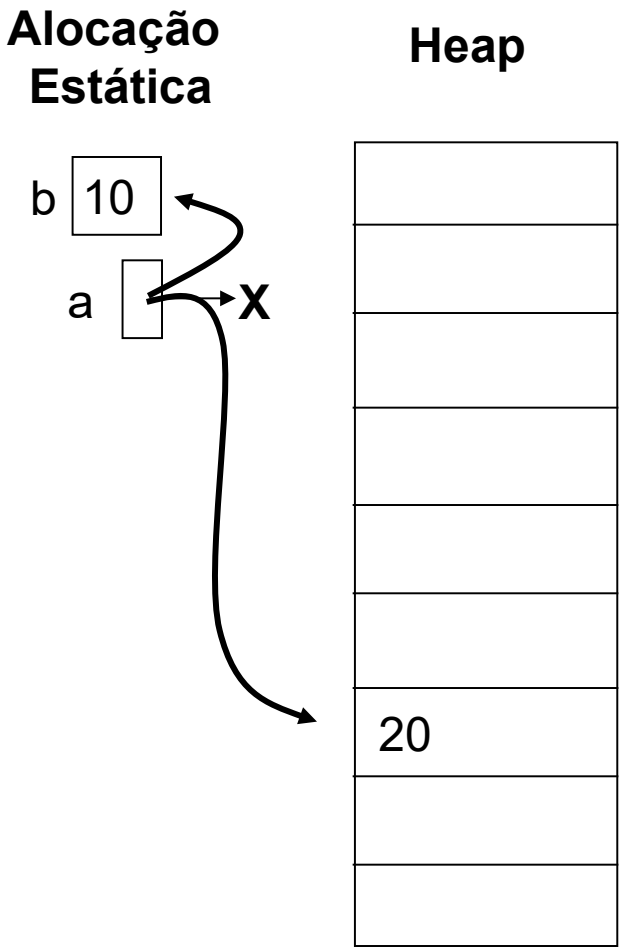
- A memória deve ser liberada após o término de seu uso
- A liberação deve ser feita por quem fez a alocação:
  - Estática: compilador
  - Dinâmica: programador

Apontadores – Notação

- definição de p como um apontador para uma variável do tipo Tipo
  - Tipo *p;
- Alocação de memória para uma variável apontada por p
  - p = (Tipo*) malloc(sizeof(Tipo));
- Liberação de memória
  - free(p);
- Conteudo da variável apontada por p
  - *p;
- Valor nulo para um apontador
  - NULL;
- Endereço de uma variável a
  - &a;

Alocação Dinâmica

```
int *a, b;
...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
```



Erros Comuns

- **Esquecer de alocar memória e tentar acessar o conteúdo da variável**
- Copiar o valor do apontador ao invés do valor da variável apontada
- Esquecer de desalocar memória
  - Ela é desalocada ao fim do programa ou procedimento função onde a variável está declarada, mas pode ser um problema em loops
- Tentar acessar o conteúdo da variável depois de desalocá-la

Alocando um bloco de memória

int *a é um ponteiro para int

- Em C, todo apontador pode se comportar como vetor:
- Desta forma pode-se fazer coisas como:

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

100
100

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

42657
100

Obs. Não se pode fazer a = b no exemplo acima

Apontadores para Tipos Estruturados

- Apontadores para estruturas

```
typedef struct {
      int idade;
      double salario;
} TRegistro;

TRegistro *a;
...
a = (TRegistro *) malloc(sizeof(TRegistro))
a->idade = 30;    /* (*a).idade = 30 */
a->salario = 80;
```

Passagem de Parâmetros

- Em pascal, parâmetros para função podem ser passados por valor ou por referência
  - **Por valor:** o parâmetro formal (recebido no procedimento) **é uma cópia** do parâmetro real (passado na chamada)
  - **Por referência:** o parâmetro formal (recebido no procedimento) **é uma referência** para o parâmetro real (passado na chamada)
    - Usa-se o termo **var** precedendo o parâmetro formal
- Em C só existe passagem por valor, logo deve-se implementar a passagem por referência utilizando-se apontadores

Passagem de Parâmetros (C)

```
void SomaUm(int x, int *y)
{
   x = x + 1;
   *y = (*y) + 1;
   printf("Funcao SomaUm: %d %d\n", x, *y);
}

int main()
{
   int a=0, b=0;
   SomaUm(a, &b);
   printf("Programa principal: %d %d\n", a, b);
}
```

```
1  1
```

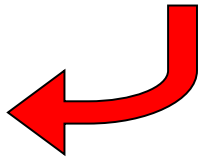
```
0  1
```

Passagem de Parâmetros

- Alocando memória dentro de uma rotina
  - Em pascal, basta passar a variável (apontador) como referência.
  - Em C também, mas como não há passagem por referência as coisas são um pouco diferente

```
void aloca(int *x, int n)
{
  x=(int *)malloc(n*sizeof(int));
  x[0] = 20;
}
int main()
{
  int *a;
  aloca(a, 10);
  a[1] = 40;
}
```

Error!
Access Violation!



```
void aloca(int **x, int n)
{
 (*x)=(int*)malloc(n*sizeof(int));
 (*x)[0] = 20;
}
int main()
{
  int *a;
  aloca(&a, 10);
  a[1] = 40;
}
```

OK