ALOCAÇÃO DINÂMICA

Professor Msc Paulo de Tarso F. Júnior

paulodt@gmail.com

Alocação Estática vs Alocação 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];
- P: Qual o espaço de memória que deve ser reservado para um editor de texto?

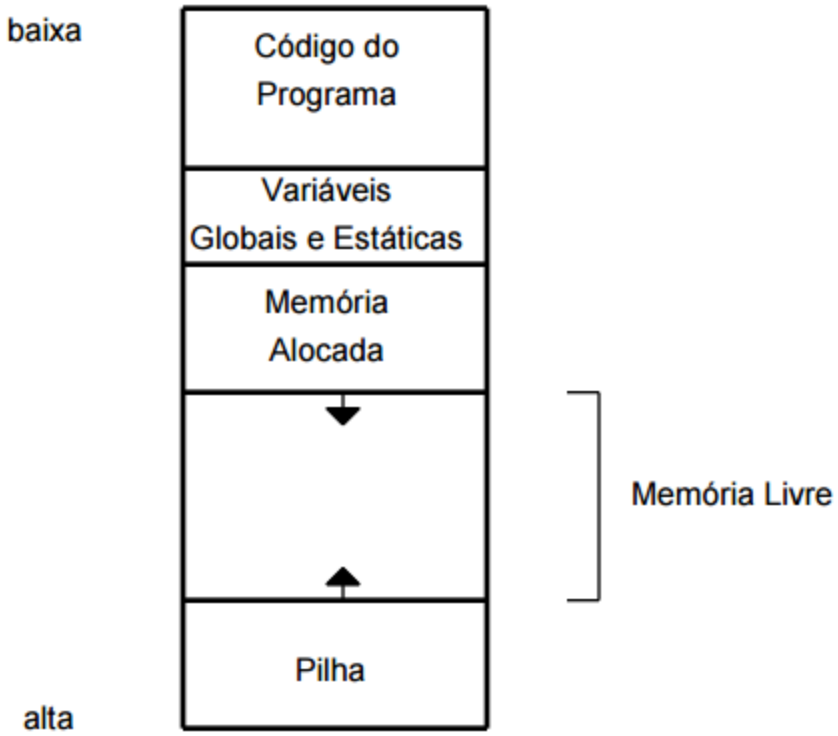
Alocação Estática vs Alocação Dinâmica

- 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];
- ***R: Na alocação dinâmica, o espaço para as variáveis pode ser alocado dinamicamente durante a execução do programa***

Alocação Estática vs Alocação Dinâmica

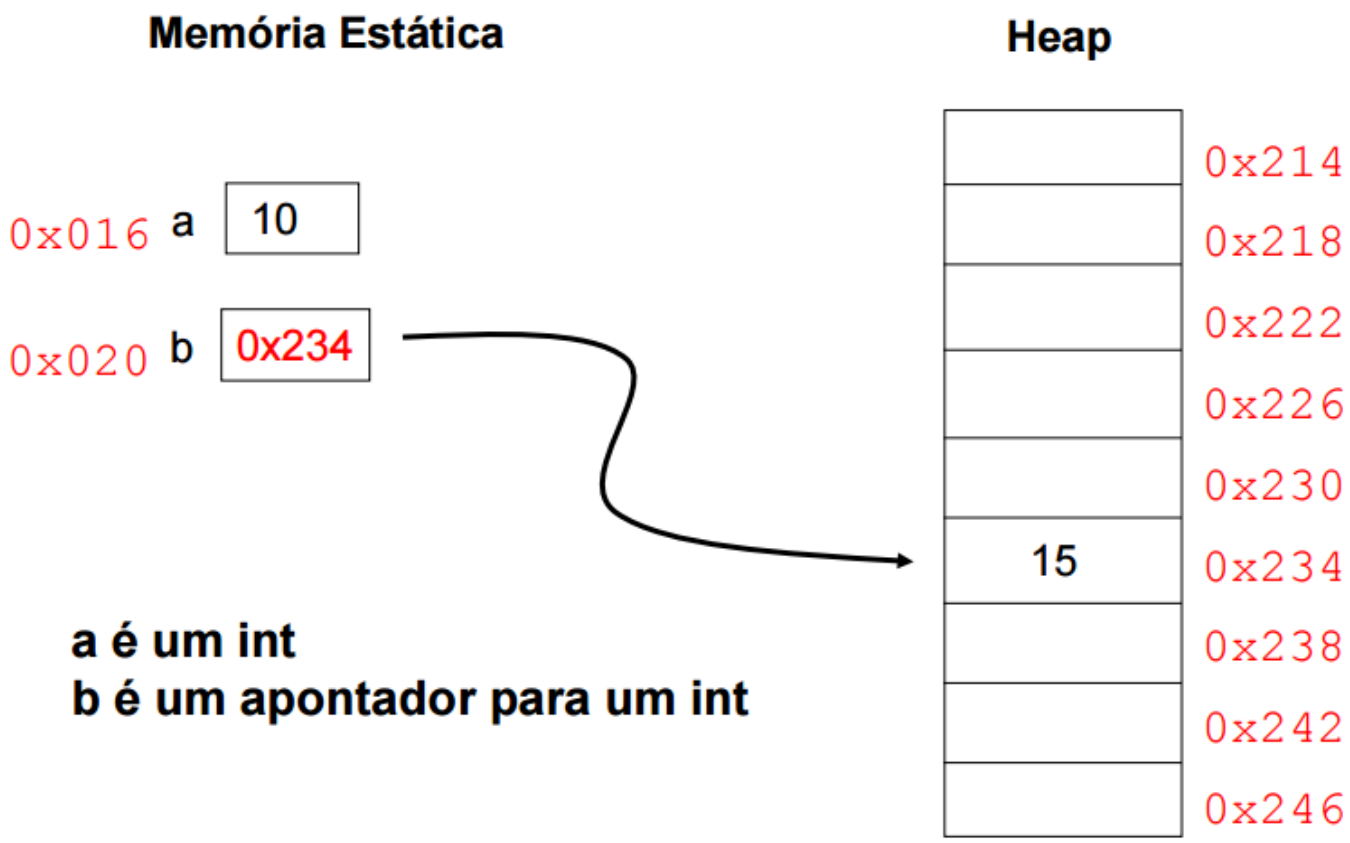
- As variáveis alocadas dinamicamente são chamadas de ***Apontadores*** (pointers) pois na verdade elas armazenam o endereço de memória de uma variável
- A memória alocada dinamicamente faz parte de uma área de memória chamada ***heap***
  - Basicamente, o programa aloca e desaloca porções de memória do ***heap*** durante a execução

Esquema de Memória



Esquema da memória do sistema

Esquema de Memória



Acesso a partir de apontadores

- int a = 10;
- int *p = &a;
- Acessar o valor da variável: endereço de memória armazenado
  - printf(“%p”, p); //imprime o endereço (ex: 0x016)
- Acessar o conteúdo que associado ao endereço de memória armazenado
  - printf(“%d”, *p); //imprime o conteúdo armazenado no endereço (ex: 10)

Acesso a partir de apontadores

- 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);
- Conteúdo da variável apontada por P
  - *p;
- Valor nulo para um apontador
  - NULL;
- Endereço de uma variável a
  - &a;

Apontadores - Notação

- Exemplos:
  - Código que aloca 1000 bytes de memória livre:
    - char *p;
    - p = malloc(1000);
  - Código que aloca espaço para 50 inteiros:
    - int *p;
    - p = malloc(50*sizeof(int));
  - Obs.: O operador sizeof() retorna o número de bytes de um inteiro.

Apontadores - Notação

- Função básica para alocar memória é malloc()
  - int *vet; vet = malloc(10*4);
  - Após estes comandos, se a alocação for bem sucedida, ***vet*** armazenará o endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros.
  - Desta forma, consideramos que um inteiro ocupa 4 bytes. Para ficarmos independentes de compiladores e máquinas, usamos o operador sizeof( )
    - v = malloc(10*sizeof(int));

Apontadores - Notação

- 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. oper.*/
} …
```

Apontadores - Notação

- As funções calloc e malloc 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 */

Apontadores - Notação

- 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 */

Apontadores - Notação

- 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)); /* aloca espaço para um inteiro */
  - 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); /* libera o inteiro alocado */

Exemplo de Uso

```
#include
/* Incrementa elementos de um vetor */
void incr_vetor (int *v , int tam) {
	int i;
	for (i = 0; i < tam; i++)
	v[i]++;
}

int main () {
	int a[ ] = {1, 3, 5};
	incr_vetor(a, 3);
	printf("%d %d %d\n", a[0], a[1], a[2]);
	return 0;
}
```

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática
#e01
#e02
#e03
#e04
#e05
#e06
#e07
#e08
#e09
#e10

Heap	
#h01	0
#h02	0
#h03	0
#h04	0
#h05	0
#h06	0
#h07	0
#h08	0
#h09	0
#h10	0

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática	Heap	
#e01	#h01	0
#e02	#h02	0
#e03	#h03	0
#e04	#h04	0
#e05	#h05	0
#e06	#h06	0
#e07	#h07	0
#e08	#h08	0
#e09	#h09	0
#e10	#h10	0

Indicam que a memória já foi alocada (0 = N, 1 = S)

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática	
#e01	a
#e02	b
#e03	
#e04	
#e05	
#e06	
#e07	
#e08	
#e09	
#e10	

Heap	
#h01	0
#h02	0
#h03	0
#h04	0
#h05	0
#h06	0
#h07	0
#h08	0
#h09	0
#h10	0

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática

#e01	a	
#e02	b	10
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	0
#h02	0
#h03	0
#h04	0
#h05	0
#h06	0
#h07	0
#h08	0
#h09	0
#h10	0

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática

#e01	a	#h01
#e02	b	10
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	1	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática

#e01	a	#e02
#e02	b	10
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	1	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;    // qual o valor de b?
```

Memória Estática

#e01	a	#e02
#e02	b	30
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	1	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática

#e01	a	#e02
#e02	b	30
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	1	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

Memória continua alocada

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
free(a);
```

Memória Estática

#e01	a	#e02
#e02	b	30
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	?	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
a = &b;
*a = 30;     // qual o valor de b?
free(a);
```

a aponta para memória estática

Memória Estática

#e01	a	#e02
#e02	b	30
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	1	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

Alocação Dinâmica

```
int *a, b;
//...
b = 10;
a = (int *) malloc(sizeof(int));
*a = 20;
free(a);
a = &b;
*a = 30;     // qual o valor de b?
```

Memória Estática

#e01	a	#e02
#e02	b	30
#e03		
#e04		
#e05		
#e06		
#e07		
#e08		
#e09		
#e10		

Heap

#h01	0	20
#h02	0	
#h03	0	
#h04	0	
#h05	0	
#h06	0	
#h07	0	
#h08	0	
#h09	0	
#h10	0	

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, mas pode ser um problema em loops
- Tentar acessar o conteúdo da variável depois de desalocá-la

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, mas pode ser um problema em loops
- Tentar acessar o conteúdo da variável depois de desalocá-la

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, mas pode ser um problema em loops**
- Tentar acessar o conteúdo da variável depois de desalocá-la

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, mas pode ser um problema em loops
- **Tentar acessar o conteúdo da variável depois de desalocá-la**

Exercícios

- Escreva um programa em linguagem C que solicita ao usuário a quantidade de alunos de uma turma e aloca um vetor de notas (números reais). Depois de ler as notas, imprime a média aritmética, a maior e a menor nota.
- Obs1: não deve ocorrer desperdício de memória; e após ser utilizada a memória deve ser devolvida.
- Obs2: usar apenas as variáveis a seguir:
  - Int qtdAlunos, i;
  - float *notas, *maior, *menor, *media;

Referências

- Material baseado na aula “Alocação Dinâmica de Memória” dos professores Luiz Chaimowicz, Raquel O. Prates, Pedro O.S. Vaz de Melo Algoritmos e Estruturas de Dados II DCC – UFMG

ALOCAÇÃO DINÂMICA

Professor Msc Paulo de Tarso F. Júnior

paulodt@gmail.com