UFMG

Alocação Dinâmica de Memória

Luiz Chaimowicz, Raquel O. Prates,
Pedro O.S. Vaz de Melo

Algoritmos e Estruturas de Dados I
DCC – UFMG

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];

- P: Qual o espaço de memória que deve ser reservado para um editor de texto?

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];

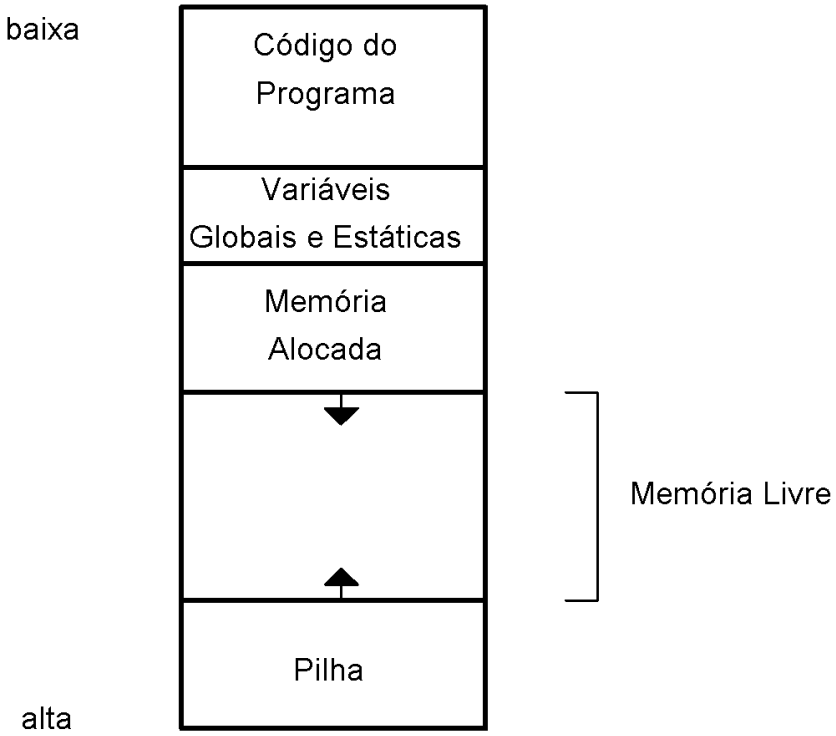
R: Na alocação dinâmica, o espaço para as variáveis pode ser alocado dinamicamente durante a execução do programa

Alocação Dinâmica

- As variáveis alocadas dinamicamente são chamadas de **Apontadores** (*pointers*) pois na verdade elas armazenam o endereço de memória de uma variável

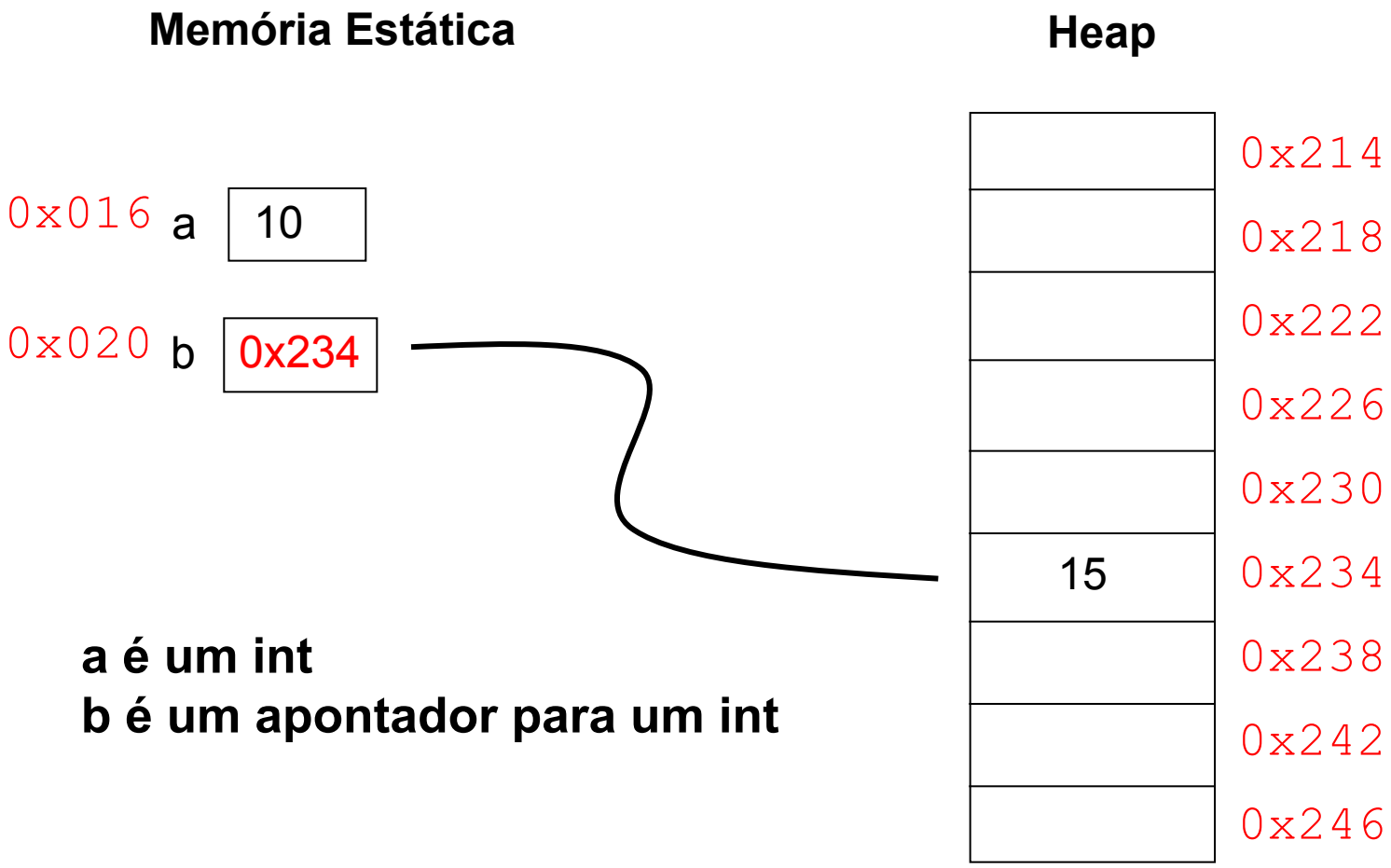
- A memória alocada dinamicamente faz parte de uma área de memória chamada ***heap***
  - Basicamente, o programa aloca e desaloca porções de memória do **heap** durante a execução

Esquema de Memória



Esquema da memória do sistema

Alocação Dinâmica



a é um int
b é um apontador para um int

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)

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);
- Conteúdo 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;
*a = 30;      // qual o valor de b?
```

Pilha

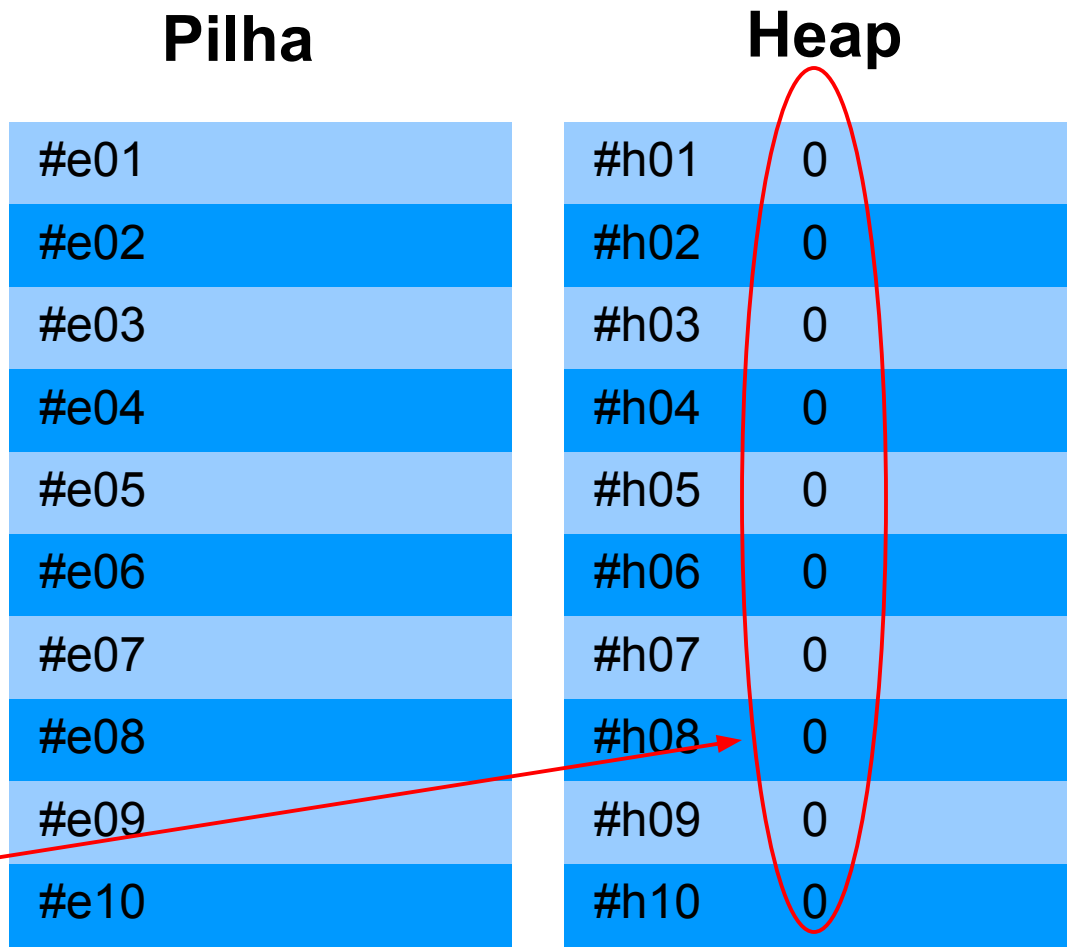
#e01	
#e02	
#e03	
#e04	
#e05	
#e06	
#e07	
#e08	
#e09	
#e10	

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?
```



Indicam se 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?
```

Pilha

#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?
```

Pilha

#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?
```

Pilha

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

Heap

#h01	**1**
#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?
```

Pilha

#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?
```

Pilha

#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?
```

Pilha

#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 continua alocada

Pilha

#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?
free(a);
```

Pilha

#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

Pilha

#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?
```

Pilha

#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**

Ex1: O que vai ser impresso?

```
double a;
double *p, *q;

a = 3.14;
printf("%f\n", a);
p = &a;
*p = 2.718;
printf("%f\n", a);
a = 5;
printf("%f\n", *p);
p = NULL;
p = (double *)malloc(sizeof(double));
*p = 20;
q = p;
printf("%f\n", *p);
printf("%f\n", a);
free(p);
printf("%f\n", *q);
```

Ex1: O que vai ser impresso?

```
double a;
double *p, *q;

a = 3.14;
printf("%f\n", a);
p = &a;
*p = 2.718;
printf("%f\n", a);
a = 5;
printf("%f\n", *p);
p = NULL;
p = (double *)malloc(sizeof(double));
*p = 20;
q = p;
printf("%f\n", *p);
printf("%f\n", a);
free(p);
printf("%f\n", *q);
```

3.14
2.718
5.0
20.0
5.0
?

Ex2: O que vai ser impresso?

```c
int main()
{

    int *p1, *p2, a;
    a = 10;
    printf("conteudo de a: %d\n", a);
    printf("endereco de a: %p\n\n", &a);

    p1 = &a;
    printf("conteudo de p1: %p\n", p1);
    printf("endereco de p1: %p\n", &p1);
    printf("valor apontado por p1: %d\n\n", *p1);

    p2 = (int *)malloc(sizeof(int));
    *p2 = *p1;
    printf("conteudo de p2: %p\n", p2);
    printf("endereco de p2: %p\n", &p2);
    printf("valor apontado por p2: %d\n", *p2);


    free(p2);

    return 0;
}
```

Ex2: O que vai ser impresso?

```c
int main()
{

    int *p1, *p2, a;
    a = 10;
    printf("conteudo de a: %d\n", a);
    printf("endereco de a: %p\n\n", &a);

    p1 = &a;
    printf("conteudo de p1: %p\n", p1);
    printf("endereco de p1: %p\n", &p1);
    printf("valor apontado por p1: %d\n\n", *p1);

    p2 = (int *)malloc(sizeof(int));
    *p2 = *p1;
    printf("conteudo de p2: %p\n", p2);
    printf("endereco de p2: %p\n", &p2);
    printf("valor apontado por p2: %d\n", *p2);


    free(p2);

    return 0;
}
```

```
conteudo de a: 10
endereco de a: 0028FF14

conteudo de p1: 0028FF14
endereco de p1: 0028FF1C
valor apontado por p1: 10

conteudo de p2: 003E2B50
endereco de p2: 0028FF18
valor apontado por p2: 10
```

Alocação Dinâmica de Vetores

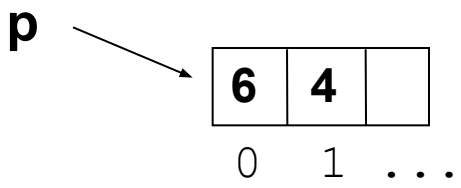
- Normalmente, a alocação dinâmica é utilizada para criar vetores em tempo de execução
- Exemplo:

```
int *p;
p = (int *)malloc(10*sizeof(int));
```

  - Aloca um vetor de inteiros com 10 posições. A manipulação pode ser feita normalmente: p[i] = ...
  - O apontador **p** guarda o endereço (aponta) da primeira posição do vetor.

Vetores e Apontadores

- Na verdade, em C todo vetor (mesmo alocados de forma estática) é um apontador para sua primeira posição.
- Pode se trabalhar usando ambas notações:
  - *p é equivalente a p[0]
  - p é equivalente a &p[0]
  - *(p + i) é equivalente a p[i]
  - considerando v um vetor alocado estaticamente, e p dinamicamente, pode-se fazer p = v, mas não v = p (v é, de certa forma, um “ponteiro constante”)



Pergunta que não quer calar...

- Qual a saída dos programas abaixo?

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

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

Pergunta que não quer calar...

- Qual a saída dos programas abaixo?

```
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

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

memória	variável	conteúdo
M1	a	
M2	b	
M3	i	
M4	v[0]	
M5	v[1]	
M6	v[2]	
M7	v[3]	
M8	v[4]	
M9	v[5]	
M10	v[6]	
M11	v[7]	
M12	v[8]	
M13	v[9]	
M14	*p	

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

memória	variável	conteúdo
M1	a	110
M2	b	10
M3	i	
M4	v[0]	
M5	v[1]	
M6	v[2]	
M7	v[3]	
M8	v[4]	
M9	v[5]	
M10	v[6]	
M11	v[7]	
M12	v[8]	
M13	v[9]	
M14	*p	M2

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

memória	variável	conteúdo
M1	a	10
M2	b	10
M3	i	
M4	v[0]	0
M5	v[1]	1
M6	v[2]	2
M7	v[3]	3
M8	v[4]	4
M9	v[5]	5
M10	v[6]	6
M11	v[7]	7
M12	v[8]	8
M13	v[9]	9
M14	*p	M4

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

memória	variável	conteúdo
M1	a	**10**
M2	b	**10**
M3	i	
M4	v[0]	**0**
M5	v[1]	**10**
M6	v[2]	**20**
M7	v[3]	**30**
M8	v[4]	**40**
M9	v[5]	**5**
M10	v[6]	**6**
M11	v[7]	**7**
M12	v[8]	**8**
M13	v[9]	**9**
M14	*p	**M4**

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

memória	variável	conteúdo
M1	a	**10**
M2	b	**10**
M3	i	
M4	v[0]	**0**
M5	v[1]	**10**
M6	v[2]	**20**
M7	v[3]	**30**
M8	v[4]	**40**
M9	v[5]	**-5**
M10	v[6]	**6**
M11	v[7]	**7**
M12	v[8]	**8**
M13	v[9]	**9**
M14	*p	**M9**

Exercício: O que vai ser impresso?

```
int a, b, i, v[10];
int *p;

b = 10;
p = &b;
a = *p + 100;
printf("%d\n", a);
a = *&b;
printf("%d\n", a);
for(i=0; i<10; i++)
  v[i] = i;
p = v;
for(i=0; i<5; i++)
  *(p+i) = 10*i;
p = p + 5;
*p = -5;
for(i=0; i<10; i++)
  printf("%d",v[i]);
```

```
110
10
0
10
20
30
40
-5
6
7
8
9
```

Apontadores para Tipos Estruturados

- Apontadores podem ser utilizados com tipos estruturados. Isso é muito comum na criação de estruturas encadeadas (listas, filas, etc...)

```
typedef struct {
     int idade;
     double salario;
} TRegistro;
...
TRegistro *a;
...
a = (TRegistro *) malloc(sizeof(TRegistro));
a->idade = 30;   // equivalente: (*a).idade
a->salario = 80;
```

Exercício: O que vai ser impresso?

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

int main()
{
    int a, *pa;
    double b, *pb;
    TRegistro *t;

    t = (TRegistro *) malloc(3*sizeof(TRegistro));
    t->idade = 30;
    t->salario = 80;

    a = 10;
    pa = &a;

    b = 3.14;
    pb = &b;

    printf("tam a: %d\n", sizeof(a)); //4
    printf("tam pa: %d\n", sizeof(pa));
    printf("tam b: %d\n", sizeof(b)); //8
    printf("tam pb: %d\n", sizeof(pb));
    printf("tam TRegistro: %d\n", sizeof(TRegistro)); //16
    printf("tam t: %d\n", sizeof(t));
```

Exercício: O que vai ser impresso?

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

int main()
{
    int a, *pa;
    double b, *pb;
    TRegistro *t;

    t = (TRegistro *) malloc(3*sizeof(TRegistro));
    t->idade = 30;
    t->salario = 80;

    a = 10;
    pa = &a;

    b = 3.14;
    pb = &b;

    printf("tam a: %d\n", sizeof(a)); //4
    printf("tam pa: %d\n", sizeof(pa));
    printf("tam b: %d\n", sizeof(b)); //8
    printf("tam pb: %d\n", sizeof(pb));
    printf("tam TRegistro: %d\n", sizeof(TRegistro)); //16
    printf("tam t: %d\n", sizeof(t));
```

```
tam a: 4
tam pa: 4
tam b: 8
tam pb: 4
tam TRegistro: 16
tam t: 4
```

Passagem de Parâmetros

- 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);
}
```

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

- E para alocar memória dentro de um procedimento?

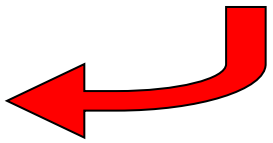
```
void aloca(int *x, int n)
{
  x=(int *)malloc(n*sizeof(int));
  x[0] = 20;
}

int main()
{
  int *a;
  aloca(a, 10);
  a[1] = 40;
}
```

Passagem de Parâmetros

- E para alocar memória dentro de um procedimento?
  - Basta passar a variável (apontador) como referência.
  - No entanto, como não há passagem por referência as coisas são um pouco mais complicadas

```
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

Passagem de Parâmetros

```
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;
}
```

ME1	a *(int *)*	Lixo

Passagem de Parâmetros

```
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;
}
```

ME1	a *(int *)*	Lixo

MH1	*x *(int)*	20
:	:	:
MH10	*(x+9) *(int)*	Lixo

MP1	x *(int *)*	**MH1**

Passagem de Parâmetros

```
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; ERRO!
}
```

ME1	a *(int *)*	Lixo

MH1	*x *(int)*	20
:	:	:
MH10	*(x+9) *(int)*	Lixo

Passagem de Parâmetros

```
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;
}
```

ME1	a *(int *)*	Lixo

Passagem de Parâmetros

```
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;
}
```

ME1	a *(int *)*	Lixo

MP1	x *(int **)*	**ME1**

Passagem de Parâmetros

```
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;
}
```

ME1	a *(int *)*	**MH1**

MH1	*x *(int)*	20
:	:	:
MH10	*(x+9) *(int)*	Lixo

MP1	x *(int **)*	ME1

Passagem de Parâmetros

```
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;
}
```

ME1	a *(int *)*	MH1

MH1	*x *(int)*	20
MH2	*x *(int)*	40
:	:	:
MH10	*(x+9) *(int)*	Lixo

Passagem de Parâmetros

- O código abaixo funciona?

```
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;
}
```

Passagem de Parâmetros

- O código abaixo funciona?
  - Não! *x é um inteiro, não é um ponteiro!

```
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;
}
```

Exercícios

1. Faça um programa que leia um valor *n,* crie dinamicamente um vetor de n elementos e passe esse vetor para uma função que vai ler os elementos desse vetor.

2. Declare um TipoRegistro, com campos *a* inteiro e *b* que é um apontador para char. No seu programa crie dinamicamente uma váriavel do TipoRegistro e atribua os valores 10 e ‘x’ aos seus campos.

Respostas (1)

```
void LeVetor(int *a, int n){
  int i;
  for(i=0; i<n; i++)
     scanf("%d",&a[i]);
}

int main(int argc, char *argv[]) {
    int *v, n, i;

    scanf("%d",&n);
    v = (int *) malloc(n*sizeof(int));
    LeVetor(v,n);

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

Apesar do conteúdo ser modificado
Não é necessário passar por referência pois todo vetor já é um apontador...

Respostas (2)

```
typedef struct {
        int a;
        char *b;
} TRegistro;
```

É necessário alocar espaço para o registro e para o campo b. *(reg->b) representa o conteúdo da variável apontada por reg->b

```
int main(int argc, char *argv[])
{
    TRegistro *reg;

    reg = (TRegistro *) malloc(sizeof(TRegistro));
    reg->a = 10;
    reg->b = (char *) malloc(sizeof(char));
    *(reg->b) = 'x';

    printf("%d %c",reg->a, *(reg->b));
}
```