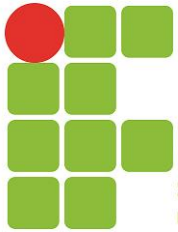


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Presidente Epitácio

Estruturas de Dados – ED1C3

Alocação Dinâmica

PROF. MARCELO ROBERTO ZORZAN

DISCIPLINA: ESTRUTURAS DE DADOS I

AULA 11

Aula de Hoje!

Alocação de Memória

- Alocação estática de memória
- Alocação dinâmica de memória
  - *Malloc, Calloc, Free, Realloc*

Alocação Estática

Definição

- As variáveis de um programa têm alocação estática se a quantidade total de memória utilizada pelos dados é previamente conhecida e definida de modo imutável, no próprio código-fonte do programa

Na Linguagem C, o espaço de memória utilizado por um programa para armazenar dados normalmente é indicado pelo programador no momento da declaração de variáveis

Exemplo

- char vetor [11]

Alocação Estática

Vantagens da Alocação Estática (Sequencial)

- Indexação eficiente facilita o acesso a uma posição qualquer de um vetor ou matriz.

Desvantagens da Alocação Estática

- É necessário saber de antemão a quantidade máxima de dados de um conjunto a ser utilizada por um programa (pode acarretar em desperdício de espaço quando a quantidade máxima não é utilizada para atender a uma dada situação).
- Inserção e remoção são custosas quando envolvem algum esforço para movimentar/deslocar elementos, de modo a abrir espaço para inserção (ex. inserção ordenada), ou de modo a ocupar espaço liberado por um elemento que foi removido.

Alocação Dinâmica

Definição

- As variáveis de um programa têm alocação dinâmica quando suas áreas de memória – não declaradas no programa – passam a existir durante execução, ou seja, o programa é capaz de criar novas variáveis enquanto executa.

Como trabalhar com alocação dinâmica?

- Para trabalhar com variáveis alocadas dinamicamente é necessário o uso de apontadores (ponteiros) e o auxílio de funções que permitam reservar (e liberar), em tempo de execução, espaços da memória para serem usados pelo programa.

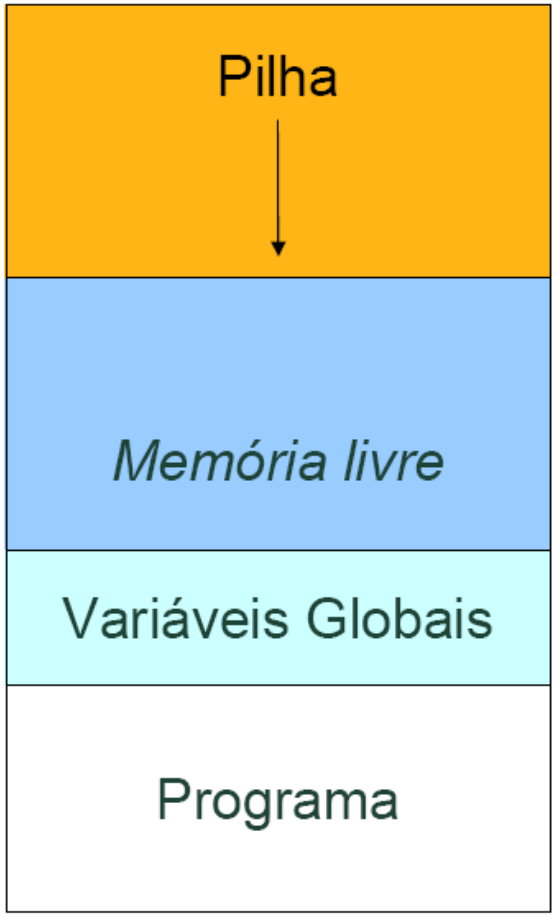
Uso da memória principal

A memória principal pode ser usada para armazenar variáveis locais e globais – incluindo *arrays* e estruturas.

- Variáveis globais: o armazenamento é fixo durante todo o tempo de execução do programa
- Variáveis locais: o armazenamento é feito na pilha do computador

Este tipo de uso da memória exige que o programador saiba, antemão, a quantidade de armazenamento necessária para todas as situações.

Uso da memória principal



Alocação Dinâmica

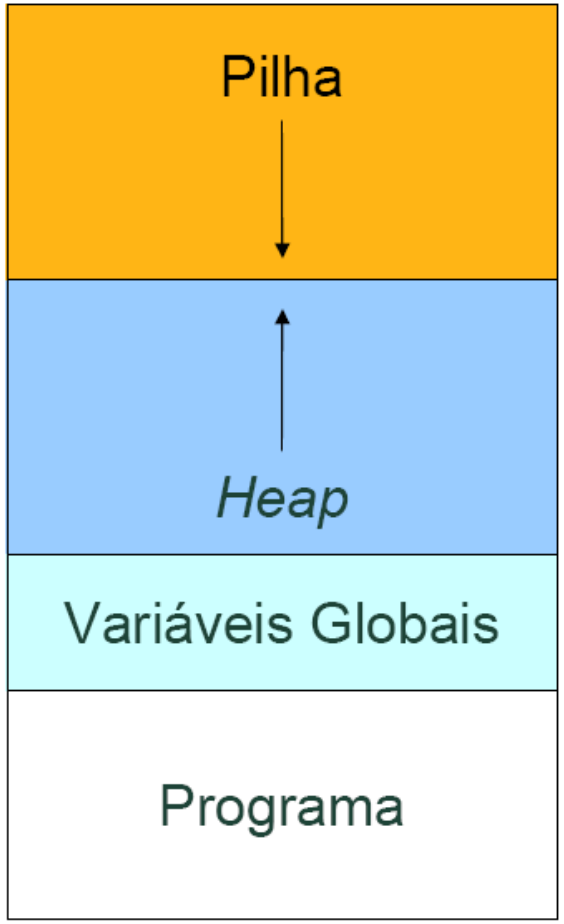
Para oferecer um meio de armazenar dados em **tempo de execução**, há um subsistema de **alocação dinâmica.**

O armazenamento de forma dinâmica é feito na região de memória livre, chamada de ***heap.***

A pilha cresce em direção inversa ao *heap.*

- Conforme o uso de variáveis na pilha e a alocação de recursos no *heap*, a memória poderá se esgotar e um novo pedido de alocação de memória falhar.

Alocação Dinâmica



Alocação Dinâmica

No núcleo do sistema de alocação dinâmica em C estão as funções:

- **malloc()**
- **free()**

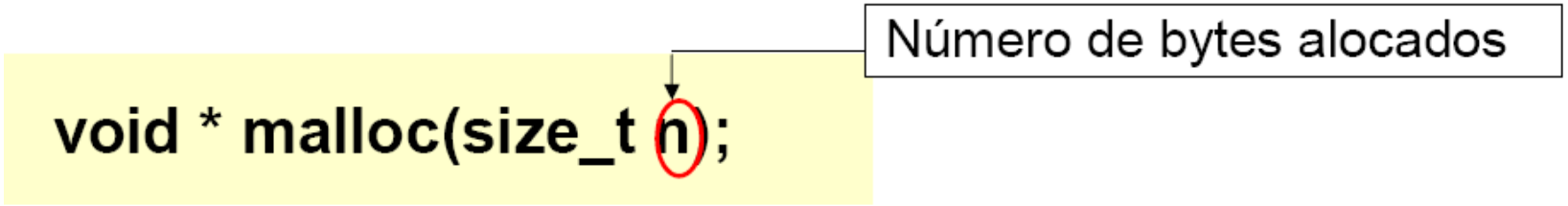
Quando **malloc()** é usada, uma porção de memória livre é alocada

Quando **free()** é usada, uma porção de memória alocada é novamente devolvida para o sistema

Os protótipos das funções de alocação dinâmica estão na biblioteca **stdlib.h**

Alocação Dinâmica - *malloc*

Função *malloc*



/*

retorna um ponteiro void para n bytes de memória não iniciados. Se não há memória disponível malloc retorna NULL

*/

Exemplo de uso:

```
int *pi;
pi= (int *) malloc (sizeof(int));
// aloca espaço para um inteiro
```

Alocação Dinâmica - *malloc*

```
ptr = malloc(size);
```

A função **malloc()** devolve um ponteiro para o primeiro byte de uma região de memória do tamanho ***size.***

Caso não haja memória suficiente, retorna nulo (**NULL**).

Alocação Dinâmica - *malloc*

```
float *v;
int n;
printf("Quantos valores? ");
scanf("%d", &n);
v = (float *) malloc(n * sizeof(float) );

if(v!=NULL)
   // manipula a região alocada
```

Uma porção de memória capaz de guardar **n** números reais (float) é reservada, ficando o apontador **v** a apontar para o endereço inicial dessa porção de memória

O *cast* da função malloc() - (float *) - garante que o apontador retornado é para o tipo especificado na declaração do apontador. Certos compiladores requerem obrigatoriamente o *cast*

Alocação Dinâmica - *free*

A função *free* é usada para liberar o armazenamento de uma variável alocada dinamicamente

```
int *pi;
pi= (int *) malloc (sizeof(int));

free(pi);
```

Alocação Dinâmica - *free*

```
free(ptr);
```

A função **free()** devolve ao *heap* a memória apontada pelo ponteiro **ptr**, tornando a memória livre.

Deve ser chamada apenas com ponteiro previamente alocado.

Alocação Dinâmica – Exemplo 1

```
int main ()
{
    int *p;
    p = (int *) malloc( sizeof(int) );
    if ( p == NULL )
    {
        printf("Não foi possível alocar memória.\n");
        exit(1);
     }
   *p = 5;
    printf("%d\n", *p);
    free(p);
    return 0;
}
```

Alocação Dinâmica – Exemplo 2

(tenta alocar uma estrutura)

```
struct sEndereco
{
  char rua[20];
  int numero;
};
```

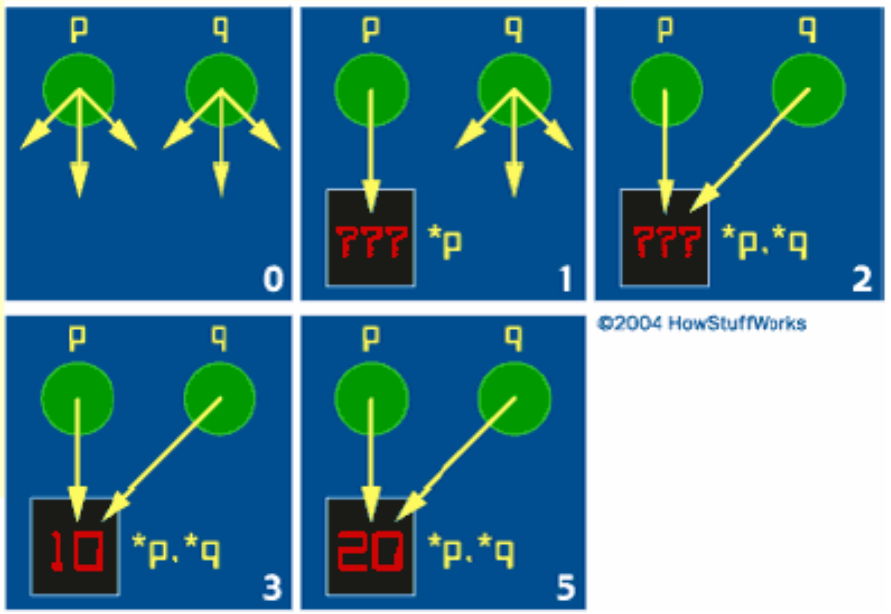
```
int main ()
{
  struct sEndereco *pend;
  pend = (struct sEndereco *)malloc( sizeof(struct sEndereco) );
  if ( pend == NULL )
  {
       printf("Não foi possível alocar memória.\n");
       exit(1);
  }
  return 0;
}
```

Alocação Dinâmica – Exemplo 3

```
int main ()
{
    int *p, *q;
    p = (int *) malloc(sizeof(int));
    q = p;
    *p = 10;
    *q = 20;
    free(p);
    q = NULL;
    return 0;
}
```

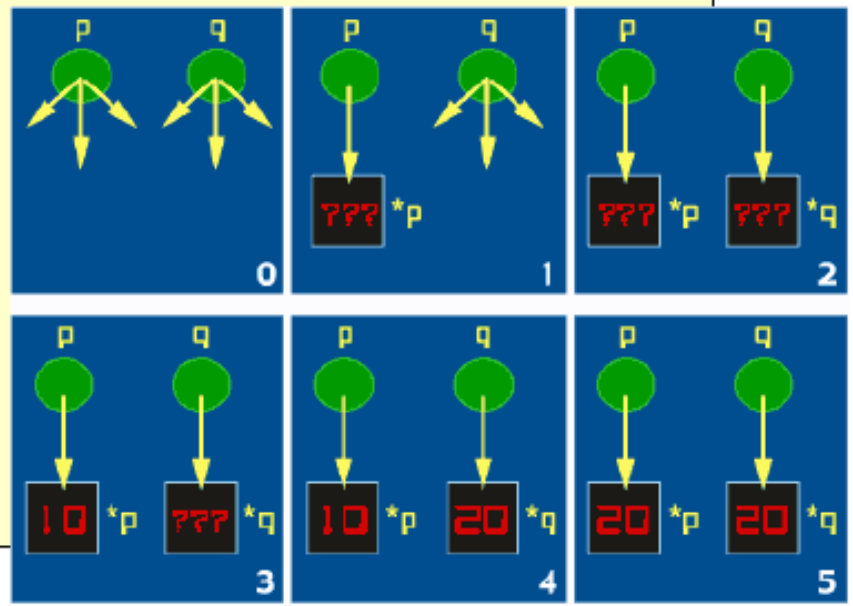
Alocação Dinâmica – Exemplo 3

```
int main (void) {
   int *p, *q;
   p = (int *) malloc(sizeof(int));
   q = p;
   *p = 10;
  *q = 20;
   free(p);
   q = NULL;
   return 0;
}
```



Alocação Dinâmica – Exemplo 4

```
int main (void) {
  int *p, *q;
  p = (int *) malloc(sizeof(int));
  q = (int *) malloc(sizeof(int));
  *p = 10;
 *q = 20;
  *p = *q;
  free(p);
  free(q);
  return 0;
}
```



Alocação Dinâmica – Exemplo 5

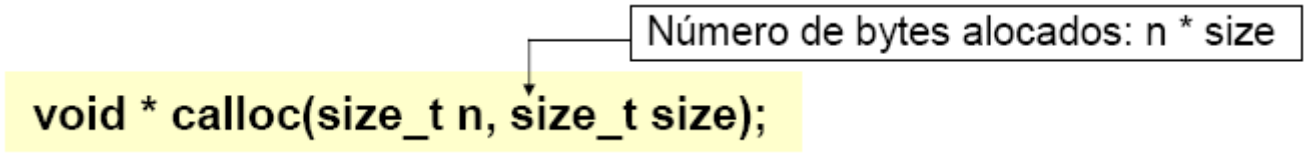
```
int main ()
{
   int *array1;
   array1 = malloc(100 * sizeof(int));
   if (array1 == NULL )
   {
      printf("Não foi possível alocar memória.\n");
      exit(1);
   }
   array1[99] = 301;
   printf("%d\n", array1[99]);
   free(array1);
   return 0;
}
```

Alocação Dinâmica – Exemplo 6

```
int main () {
  CLIENTE *pc;
  pc = (CLIENTE *) malloc( 50 * sizeof(CLIENTE) );

  gets( pc[0].nome );
  scanf("%d", &pc[0].idade );

  printf("%s", pc[0].nome);
  printf("%d", pc[0].idade);

  free(pc);
  return 0;
}
```

```
typedef struct cli
{
   char nome[30];
   int idade;
} CLIENTE;
```

Alocação Dinâmica - *calloc*




```
/*calloc retorna um ponteiro para um vetor com n elementos de tamanho
   size cada um ou NULL se não houver memória disponível. Os elementos
   são iniciados em zero
 */
```

Exemplo de uso

```
float *ai = (float *) calloc (n, sizeof(float));
/* aloca espaço para um vetor de n float */
```

Toda memória não mais utilizada deve ser liberada pela função free:

```
free(ai);  /* libera todo o vetor*/
```

Alocação Dinâmica - *realloc*

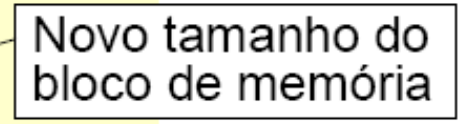
É possível alterar o tamanho do bloco de memória reservado, utilizando a função ***realloc().***

Esta função salva os valores anteriormente digitados em memória, até ao limite do novo tamanho (especialmente importante quando se reduz o tamanho do bloco de memória).

Alocação Dinâmica - *realloc*

Exemplo de uso da função realloc:

```
int *a;
a = (int *) malloc( 10 * sizeof(int) );
 ...
a = (int *) realloc( a, 23 * sizeof(int) );
...
free(a);
```



A chamada da função *realloc*() recebe como argumentos um apontador para o bloco de memória previamente reservado pela função *malloc*() ou *calloc*() de forma a identificar qual a porção de memória será redimensionada, e o novo tamanho absoluto para o bloco de memória.

Alocação Dinâmica - *realloc*

```
ptr = realloc(ptr, size);
```

A função **realloc()** modifica o tamanho da memória alocada previamente e apontada pelo ponteiro ***ptr***

Caso não haja memória suficiente, retorna nulo (**NULL**)

Exercícios

1- Refaça o exercício abaixo supondo agora que o usuário irá fornecer quantos produtos ele deseja armazenar

→ Considere um cadastro de produtos de um estoque, com as seguintes informações para cada produto:

- Código de identificação do produto: representado por um valor inteiro
- Nome do produto: com ate 50 caracteres
- Quantidade disponível no estoque: representado por um número inteiro
- Preço de venda: representado por um valor real

(a) Defina uma estrutura em C, denominada produto, que tenha os campos apropriados para guardar as informações de um produto.

(b) Crie um conjunto de 10 produtos e peça ao usuário para entrar com as informações de cada produto.

(c) Encontre o produto com o maior preço de venda.

(d) Encontre o produto com a maior quantidade disponível no estoque.

Exercícios

2. Crie um vetor de inteiros com X posições, onde X é um valor fornecido pelo usuário. Preencha as posições desse vetor com números consecutivos.

Exercícios

3. Elaborar um programa em C que auxilie no controle de uma fazenda de gado que possui um total de 2000 cabeças de gado. A base de dados é formada por um conjunto de estruturas (registros) contendo os seguintes campos referente a cada cabeça de gado:
   - código: código da cabeça de gado
   - leite: número de litros de leite produzido por semana
   - alim: quantidade de alimento ingerida por semana - em quilos
   - nasc: data de nascimento - mês e ano
     - struct data que por sua vez, possui dois campos: mês e ano
   - abate: ‘N" (não) ou ‘S’ (sim).

   (a) Leia a base de dados armazenada em um vetor de estruturas.

   (b) Preencha o campo abate, considerando que a cabeça de gado irá para o abate caso
     - tenha mais de 5 anos, ou
     - produza menos de 40 litros de leite por semana, ou
     - produza entre 50 e 70 litros de leite por semana e ingira mais de 50 quilos de alimento por dia