

## Alocação dinâmica de memória

Prof. Leandro O. Freitas  
leandro@politecnico.ufsm.br



# Alocação dinâmica

- Muitas vezes o programador não sabe de antemão a quantidade de dados que seu programa deverá manter em memória para correto processamento.
  - Programas que devem manter na memória dados de tamanho desconhecido para seu processamento.
    - Exemplo: editor de textos.
    - Qual o número máximo de caracteres que deve suportar?



# Alocação dinâmica

- Como declarar variáveis para armazenar esses dados?
  - Uma forma de solucionar esse problema é declarar variáveis com um determinado tamanho, **razoavelmente grande**, e, caso esse tamanho seja insuficiente, o programa deve **informar o usuário sobre essa limitação e abortar a execução**.
- Isto leva a dois grandes problemas:
  - Espaço alocado demasiado, ou seja, desperdício de memória;
  - Ou espaço insuficiente para alocação de todos os dados.



# Alocação dinâmica

- Outra solução é o programa **alocar memória do sistema dinamicamente** (durante a sua execução), conforme a necessidade do usuário.
- Desta forma, o programa utilizará somente a memória necessária para o processamento que irá realizar e poderá processar **um volume de dados limitado somente pela capacidade de memória do computador**.



# Alocação dinâmica

- Definição:
  - **Alocação Dinâmica** é o processo de **solicitar e utilizar memória durante a execução de um programa**. Ela é utilizada para que um programa em C utilize apenas a memória necessária para sua execução, sem desperdícios.
- Sendo assim, a alocação dinâmica de memória **deve ser utilizada quando não se sabe**, por algum motivo, a quantidade de memória necessária para o armazenamento de algum(ns) valores.



# Alocação dinâmica

- A biblioteca ***stdlib.h*** da linguagem C possui funções para solicitação dinâmica de memória ao sistema operacional. As principais funções são:
  - Para alocação: `malloc()`
  - Para liberação: `free()`
- Considere que existem situações em que o computador não possui memória disponível para ser alocada.
  - Neste caso, um pedido de alocação será recusado. Isto deve ser tratado pelo desenvolvedor.



# Alocação dinâmica

- `malloc (<quantidade solicitada>);`
  - Pedido de alocação de memória.
- Exemplos:  
`int *v;`  
  
`v = malloc (4);` *//aloca um espaço de 4 bytes na memória;*  
`v = malloc (10 * 4);` *//aloca um espaço de 40 bytes na memória;*  
  
`v = malloc ( sizeof(int) )` *//aloca um espaço equivalente ao tamanho de um inteiro;*  
`v = malloc (10 * sizeof(int))` *//aloca um espaço para 10 valores inteiros;*  
  
`v = (int *) malloc (10 * sizeof(int)) ;` *// especifica o tipo de ponteiro de retorno;*
- Para que seja possível determinar qual o tamanho em *bytes* desejado, pode-se utilizar a função ***sizeof***.



# Alocação dinâmica

- O retorno da função **malloc()** é um ponteiro **genérico que deve ser configurado** para o tipo alocado. Por exemplo:

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

- Essa instrução solicita alocação de memória para o tamanho de um **inteiro** e recebe como retorno de malloc() um **endereço** disponível para um elemento do tipo inteiro;
- Assim, o valor do ponteiro **p** é o endereço da memória onde o espaço foi alocado. Nele, **é possível armazenar valores inteiros**;
- Caso o pedido de memória seja recusado, **p** receberá o valor **NULL** (nulo). Portanto, **deve-se testar se um pedido de memória foi aceito**, verificando se a variável que recebe o retorno de malloc() é igual a **NULL**.

```
if (p == NULL) {  
    printf("Memória insuficiente \n");  
}
```





# Alocação dinâmica

- Toda memória alocada **dinamicamente** deve ser liberada ANTES do término da execução de um programa, caso contrário, ela não estará disponível para outras aplicações ou programas.
- `free()`
  - Liberação da memória alocada **dinamicamente**.
- Sintaxe
  - `free (endereço de memória a ser liberado);`



# Alocação dinâmica

- Atenção: **você é responsável por liberar a memória que você alocou!**
- Uma vez que a única maneira de chegar a uma memória alocada dinamicamente é através de um ponteiro, a liberação desse endereço de memória deve ser realizado a partir dele.

Exemplo:

```
free(p);
```



# Alocação dinâmica

- Analise o trecho de código a seguir e interprete seu significado:

1. `int *vet = NULL;` *//declaração do ponteiro sem apontar para um endereço.*
2. `vet = (int*) malloc (25 * sizeof(int));` *//alocação de memória para vet.*
3. `vet[10] = 34;` *// ou \*(vet+10)=34. Atribui 34 para a posição 10 de vet.*
4. `free(vet);` *// liberação da memória alocada dinamicamente.*

- O código solicita a alocação dinâmica para um vetor de inteiros com 25 posições, atribui um valor para a posição 10 e depois libera a área de memória.

# Alocação dinâmica: exemplo com vetor



Sistemas para Internet

UFSM

```
2  #include <stdlib.h> //Biblioteca necessária para alocação dinâmica
3
4  int main()
5  {
6      int *vetor, i, n; //Declara um vetor sem o tamanho, através de ponteiro
7      printf("Informe o tamanho do vetor: ");
8      scanf("%d", &n); //Define tamanho
9      vetor = (int*) malloc(n * sizeof(int)); //Aloca memória para o vetor
10     if(vetor != NULL){ //Verifica se há espaço livre na memória
11         printf("Informe os valores: \n");
12         for(i=0; i<n; i++){ //Se houver espaço de memória para
13             scanf("%d", &vetor[i]); //ou (vetor + i) o vetor, é possível inserir
14         } elementos e manipulá-los conforme
15         necessidade do usuário
16         printf("Valores inseridos: \n");
17         for(i=0; i<n; i++){
18             printf("%d \n", vetor[i]); //ou *(vetor + i)
19         }
20     }
21     else { //Caso não exista espaço disponível na memória
22         printf("Memoria insuficiente. ");
23     }
24
25     free(vetor); //libera espaço de memória usada pelo vetor.
26     return 0;
27 }
```

# Exercício

- Desenvolva um algoritmo para:
  - Criar dinamicamente um vetor de  $n$  elementos na função principal;
  - Passar o vetor por parâmetro para uma função para inserir elementos;
  - A função deve apresentar os elementos ao usuário;
  - Liberar memória antes do término do programa.