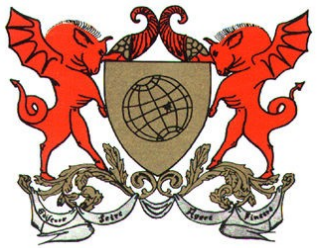
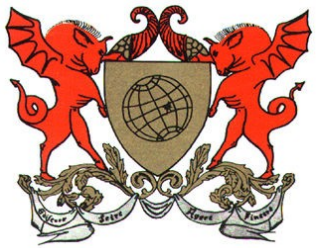


Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

SIN 110
Programação

Sistemas de Informação
Prof. Guilherme C. Pena
guilherme.pena@ufv.br



Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

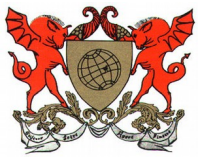
Aula de Hoje

Alocação Dinâmica

Alocação Dinâmica

Relembrando:

```
#include <stdio.h>
/* Uma variável é uma posição de memória que
armazena um dado que pode ser usado pelo
programa.
Deve ser declarada durante o desenvolvimento
do programa.*/
int main()
{
    int x, y;
    float c;
    char nome[50];
    return 0;
}
```

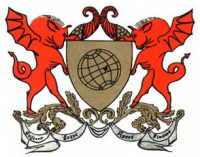


Alocação Dinâmica

Voltando àquele programa que guarda as notas dos alunos de uma sala de aula.

```
#include <stdio.h>

int main()
{
    float notas[10];
    int i;
    printf("Digite as notas dos 10 alunos\n.");
    for(i=0; i<10; i++)
        scanf("%f", &notas[i]);
    return 0;
}
```

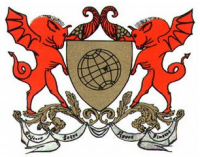


Alocação Dinâmica

Um detalhe deste programa é que o tamanho do vetor é fixo e alocado estaticamente em tempo de compilação.

```
#include <stdio.h>

int main()
{
    float notas[10];
    int i;
    printf("Digite as notas dos 10 alunos\n.");
    for(i=0; i<10; i++)
        scanf("%f", &notas[i]);
    return 0;
}
```

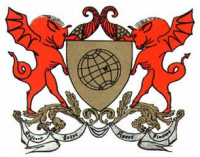


Alocação Dinâmica

Se nos precisarmos de alocar para 20 alunos, ok, só mudar para 20. E para 30, 40, 50?

```
#include <stdio.h>

int main()
{
    float notas[20];
    int i;
    printf("Digite as notas dos 20 alunos\n.");
    for(i=0; i<20; i++)
        scanf("%f", &notas[i]);
    return 0;
}
```



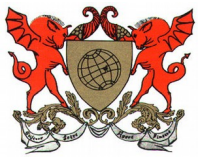
Alocação Dinâmica

Para não ter problemas vamos colocar tamanho 1000.

Mas quais são os problemas de se fazer isso?

```
#include <stdio.h>

int main()
{
    float notas[1000];
    int i;
    printf("Digite as notas dos 20 alunos\n.");
    for(i=0; i<20; i++)
        scanf("%f", notas[i]);
    return 0;
}
```



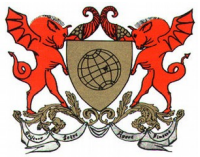
Alocação Dinâmica

Limitações da implementação com vetor

– Superdimensionar um vetor
- Alocar mais espaço do que aquele realmente necessário para armazenar a coleção.
- **Problema: desperdício de memória**

– Subdimensionar um vetor
- Alocar menos espaço do que aquele realmente necessário para armazenar a coleção.
- **Problema: erro fatal.**

Como solucionar os problemas acima?



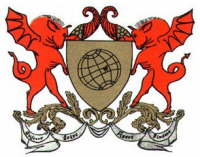
Alocação Dinâmica

- **Definição**

  Alocação dinâmica é o processo de alocar/desalocar memória em um programa durante sua **execução**.

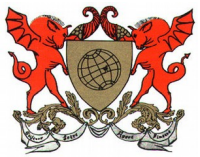
- **Como trabalhar com alocação dinâmica?**

  Por meio de ponteiros e funções que permitam **reservar** (e **liberar**) espaços da memória em tempo de **execução**.

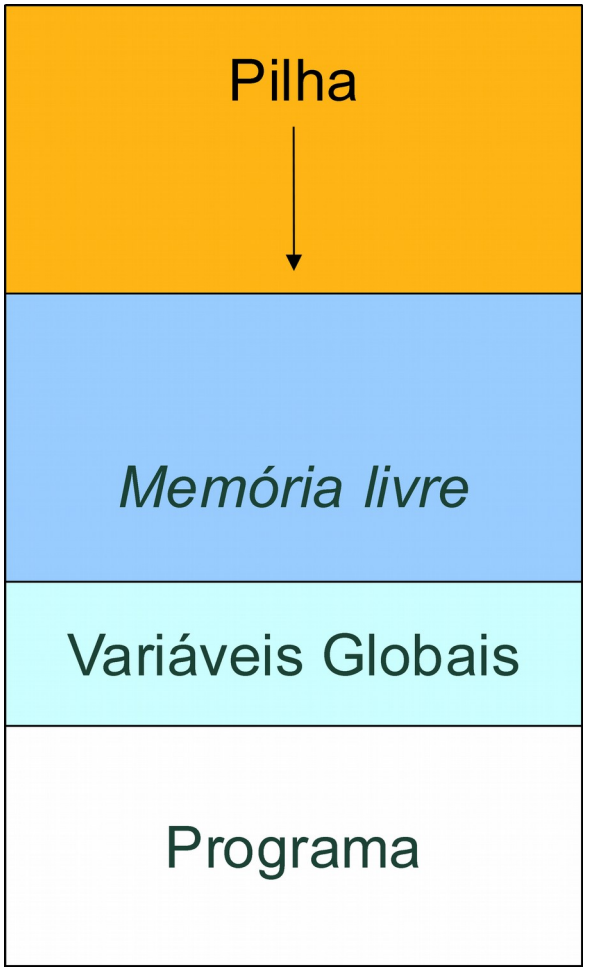


Uso da Memória Principal

- Até o momento foi usada **alocação estática**:
  - Variáveis globais: o armazenamento é fixo durante todo o tempo de execução do programa
  - Variáveis locais e vetores estáticos: o armazenamento é feito durante a execução da função, em uma estrutura de dados do tipo **pilha.**
  - Este tipo de uso da memória exige que o programador saiba, de antemão, a quantidade de armazenamento necessária para todas as situações.

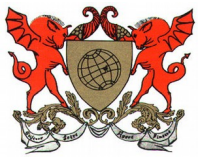


Uso da Memória Principal

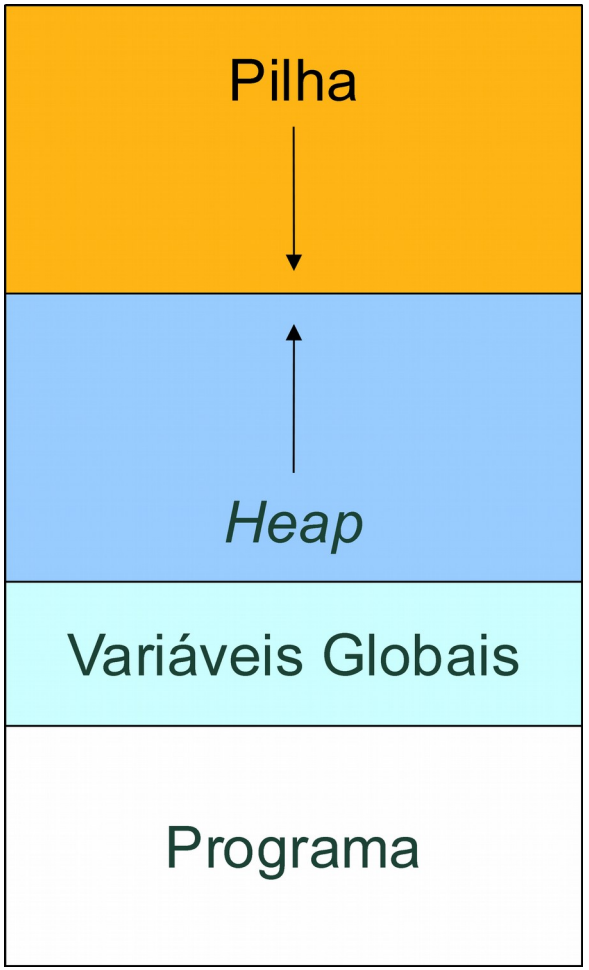


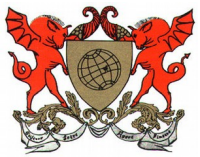
Uso da Memória Principal

- 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.



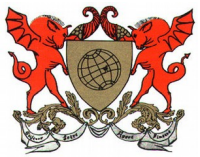
Uso da Memória Principal





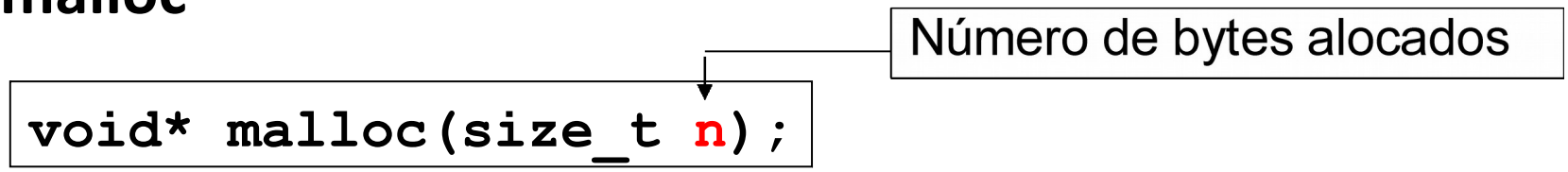
Alocação Dinâmica em C

- 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()**

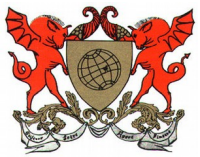
Função malloc



A função 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 (10 * sizeof(int));
// aloca espaço para um inteiro
```



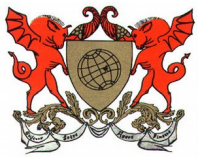
Função **sizeof()**

A função **sizeof()** retorna um valor inteiro relativo à quantidade de bytes ocupada pelo tipo que é passado como parâmetro:

```
sizeof(int)= 4;
sizeof(float)= 4;
sizeof(char)= 1;
sizeof(double)= 8;
```

```
typedef struct Sponto{
 int x, y, z;
}ponto;

sizeof(ponto)= 12;
```



Alocação Dinâmica - **malloc()**

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

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

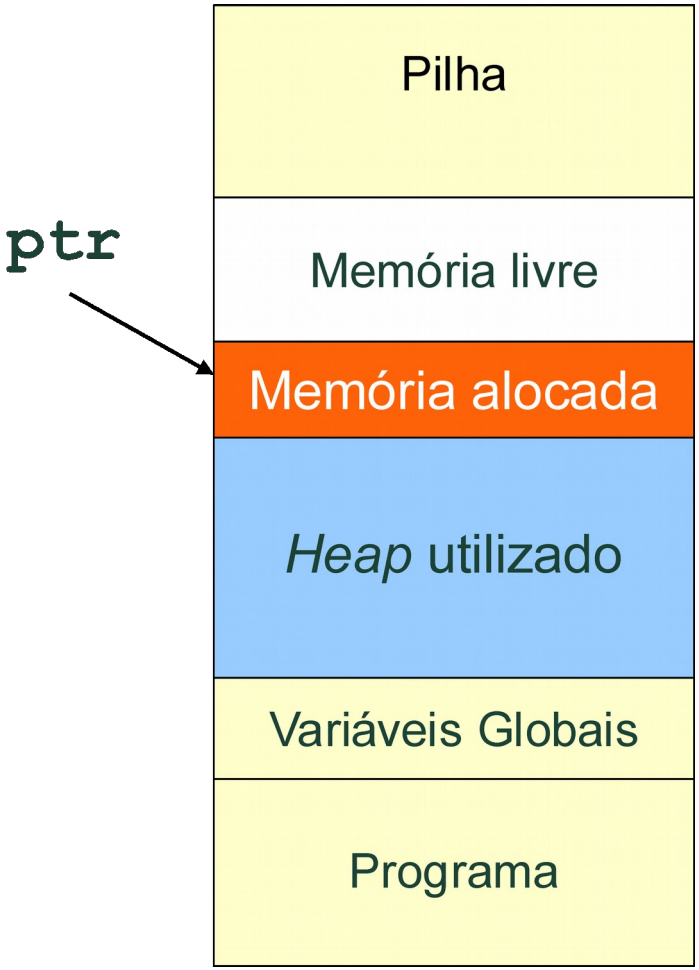
Caso não haja memória suficiente, ela retorna nulo (NULL)

Alocação Dinâmica - **malloc()**

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

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

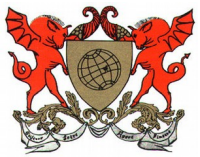
Caso não haja memória suficiente, ela retorna nulo (**NULL**)



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
```

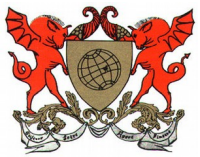
Aloca: Uma porção de memória capaz de guardar n números reais (**float**) é reservada, ficando o apontador **v** a apontar para o endereço inicial dessa porção de memória.



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
```

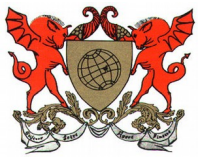
O cast da função malloc() - (float *) - garante que o apontador retornado é para o tipo especificado na declaração do apontador. Certos compiladores requerem obrigatoriamente o cast.



Alocação Dinâmica - **free()**

A função free() é usada para liberar o armazenamento de uma variável alocada dinamicamente.

```
int *pi;
pi = (int*) malloc (10 * sizeof(int));
free(pi);
```

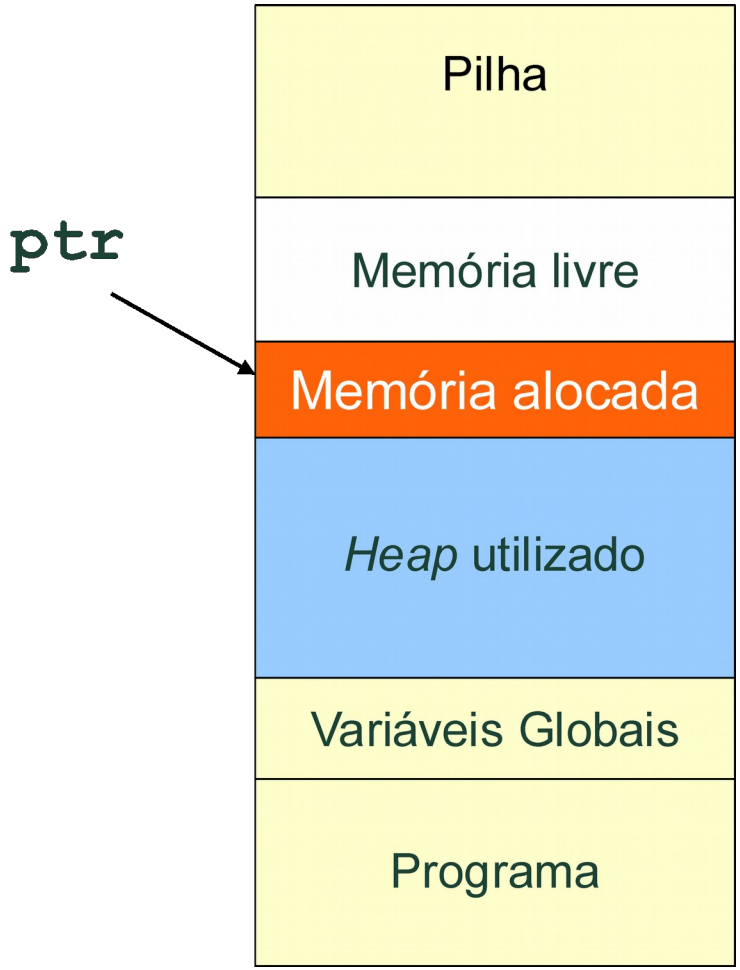



Alocação Dinâmica - **free()**

```
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 - **free()**

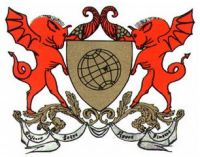
```
free(ptr);
```

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

```
#include<stdio.h>
#include<stdlib.h>
int main ()
{
   int *p;
   p = (int *) malloc( sizeof(int) );
   if ( p == NULL ){
      printf("Não foi possível alocar memória.\n");
      exit(1);
   }
   *p = 5;//p[0] = 5;
   printf("%d\n", *p/*p[0]*/);
   free(p);
   return 0;
}
```



Alocação Dinâmica - **calloc()**

Função calloc

Número de bytes alocados: n * size

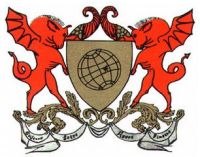
```
void* calloc(size_t n, size_t size);
```

A função 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.

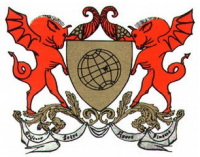
Exemplo de uso:

```
float *vf = (float *) calloc (n, sizeof(float));
/* aloca espaço para um vetor de n floats */
free(vf);
/* libera todo o vetor*/
```



Alocação Dinâmica - **calloc()**

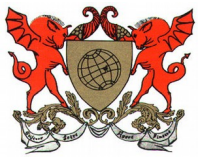
A diferença das funções **malloc()** e **calloc()** está na quantidade de parâmetros que devem ser passados na chamada das funções.



Alocação Dinâmica - **realloc()**

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

Esta função salva os valores que estão na memória até o limite do novo tamanho.



Alocação Dinâmica - **realloc()**

Exemplo de uso da função realloc():

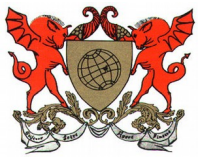
```
int *a;
a = (int *) malloc( 10 * sizeof(int) );
  ...
a = (int *) realloc( a, 23 * sizeof(int) );
...
free(a);
```

Novo tamanho do bloco de memória

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()**

Quando a função realoca o espaço de memória, ela libera o espaço antigo deixado, e no caso de uma realocação para um espaço maior, ela muda a região de memória.

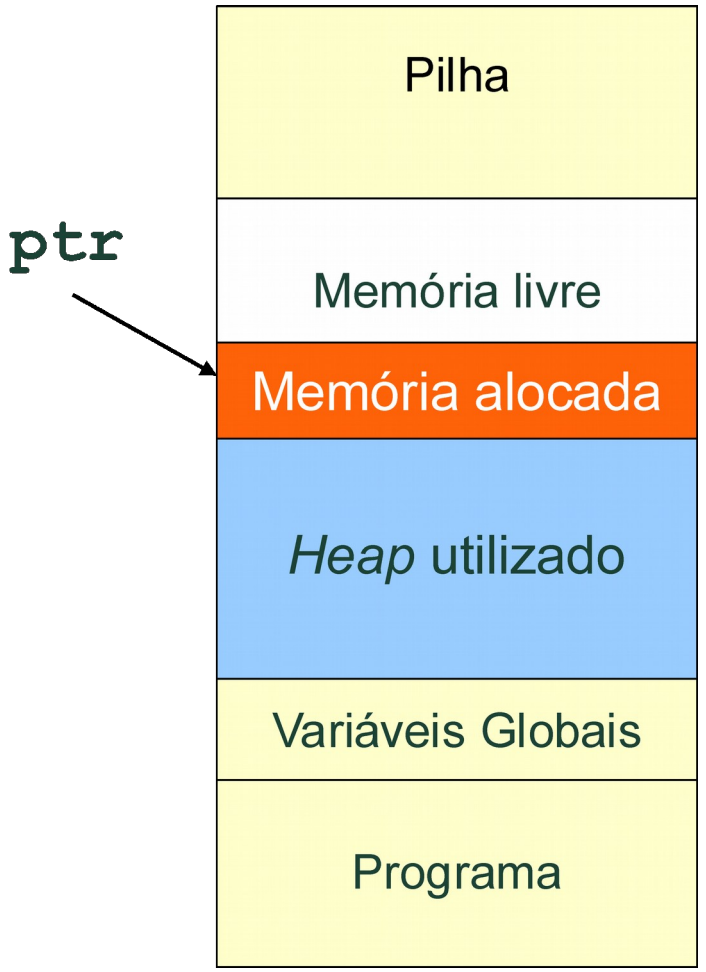


Alocação Dinâmica - **realloc()**

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

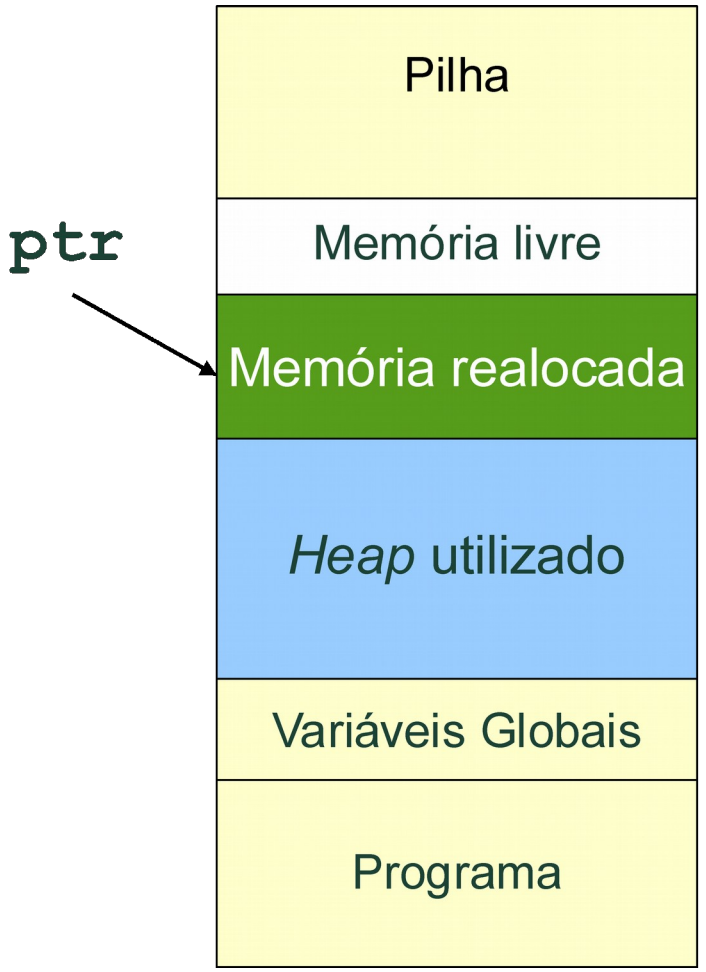
A função **realloc()** modifica o tamanho da memória alocada previamente e apontada pelo ponteiro ***ptr***

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



Alocação Dinâmica - **realloc()**

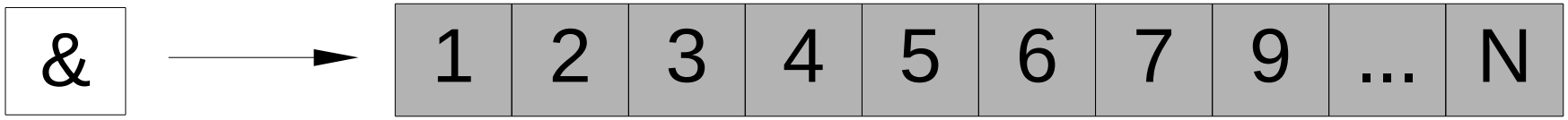
```
ptr = realloc(ptr, size);
```



Obs: Se o parâmetro size for **0 (zero)**, a memória apontada por *ptr* é liberada. Como se tivesse chamado o comando free(ptr).

Alocação Dinâmica - Matriz

Na alocação dinâmica de um vetor de uma dimensão, **cria-se um ponteiro** que apontará para a **região de memória de tamanho N** que conterá os elementos do vetor.



Ponteiro

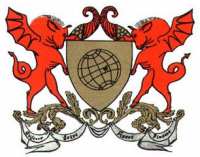
Região de N elementos retornada pelo *malloc()*.

Alocação Dinâmica - Matriz

Na alocação dinâmica de um vetor de duas dimensões (**Matriz**), deve-se **criar vários ponteiros distintos (vetor de ponteiros).**

Cada um apontará para uma **região distinta da memória cujo tamanho é definido pelo usuário**

Essas regiões que conterão os elementos da matriz.



Alocação Dinâmica - Matriz

```
#include<stdio.h>
#include<stdlib.h>
int main ()
{
   int **matriz, i;
   //Alocando vetor de ponteiros de tamanho L elementos
   matriz = (int **) malloc( num_linhas*sizeof(int*) );
   if ( matriz == NULL ){
      printf("Não foi possível alocar memória.\n");
      exit(1);
   }
   for(i=0; i<num_linhas, i++){ //Alocando LxC Elementos
      matriz[i] = (int*) malloc( num_col*sizeof(int) );
   }
   return 0;
}
```

Alocação Dinâmica - Matriz

Vetor de Ponteiros

Regiões de N elementos cada retornadas pelo *malloc()*.

Alocação Dinâmica - Matriz

Liberação de memória para a matriz faz o processo inverso:

- Primeiro libera a memória apontada pelo vetor de ponteiros.
- Depois libera o vetor de ponteiros.

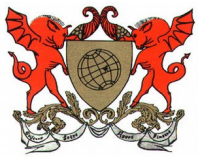
```
    ...
    for(i=0; i<num_linhas, i++){
       free(matriz[i]);
    }
    free(matriz);

    return 0;
}
```

Alocação Dinâmica
ERROS COMUNS

Quando se programa com alocação dinâmica, dois erros são comumente cometidos por programadores desatentos:

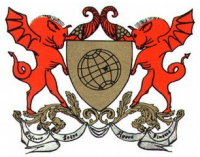
- ***Geração de Lixo***: quando, após a saída de um trecho do programa onde a memória alocada não é mais necessária, esta não é liberada com free().
- ***Referência Danosa***: quando um ponteiro referencia uma região da memória que não está alocada para o programa (ou foi alocada mas já foi liberada com **free()**).



Alocação Dinâmica
ERROS COMUNS

Exemplo 1: Geração de Lixo: Programador esquece de desalocar a memória alocada:

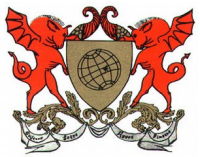
```
int main(){
    int *v = (int *) malloc (sizeof(int));
    *v = 100;
    ...
    //Suponha que não houve: free(v);
}
```



Alocação Dinâmica
ERROS COMUNS

Exemplo 2: Geração de Lixo: Outra forma de deixar resíduos na memória:

```
int main(){
    int *v = (int *) malloc (sizeof(int));
    *v = 100;
    ...
    v = (int *) malloc (sizeof(int));
/*Perde a referência da primeira região(não é
mais possível desalocá-la). O ponteiro v passou
a referenciar a segunda região alocada.*/
}
```



Alocação Dinâmica
ERROS COMUNS

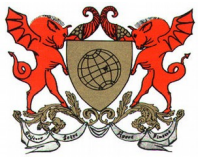
Exemplo 3: Referência Danosa: Outra forma de deixar resíduos na memória:

```
int main(){
    int *x, *y;
    x = (int *) malloc (sizeof(int));
    *x = 100;
    y = x;//Os dois ponteiros apontam pra mesma região.
    free(x);//A área é desalocada.
    *y = 200;
    /*ERRO! Pois y aponta para a mesma região (já
   desalocada) de x;*/
}
```

Exercícios

1) Faça um programa que leia do usuário o tamanho de um vetor a ser lido e faça a alocação dinâmica de memória. Em seguida, leia do usuário seus valores, imprima o vetor lido e por fim desaloque o vetor.

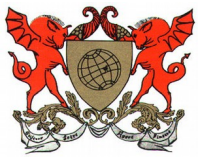
2) Faça um programa que receba do usuário o tamanho de uma string e aloque dinamicamente essa string. Em seguida, o usuário deverá informar o conteúdo dessa string. O programa deve por fim imprimir a string sem suas vogais.



Exercícios

3) Faça um programa que leia do usuário o tamanho de um vetor a ser lido e faça a alocação dinâmica de memória. Em seguida, preencha com valores aleatórios (rand()) e mostre quantos dos números são pares e quantos são ímpares.

4) Faça um programa que leia um número N e:
- Crie dinamicamente e leia um vetor de inteiro de N posições;
- Leia um número inteiro X e conte e mostre os múltiplos desse número que existem no vetor.

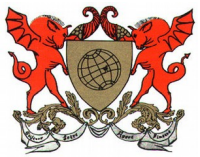


Exercício

5) Crie um programa para alocar dinâmicamente um vetor de alunos (nome, matricula, nota e endereço) de tamanho n, sendo que n é um valor digitado pelo usuário.

6) Modifique o programa anterior, construindo funções para:
- Cadastrar alunos
- Imprimir um aluno (de acordo com seu índice no vetor)
- Retornar o índice do aluno com a maior nota.

LEMBRE-SE: Libere a memória alocada antes do fim do programa.



Exercícios

7) Crie um programa que aloque dinamicamente uma matriz com tamanhos de colunas variados. Um detalhe, o número de linhas e o número de valores em cada coluna deverá ser digitado pelo usuário.

Inicialize os elementos de cada linha com o índice de sua coluna e depois imprima toda a matriz.

Libere todo o espaço de memória.