# Estruturas de Dados I
# (Alocação dinâmica de memória)

Claunir Pavan

# Alocação dinâmica

- char c; int i; int v[10]; (alocação estática)
- Há casos em programação, onde precisamos lidar com dados dinâmicos;
  - ex.: número de clientes numa fila aumenta e diminui durante o tempo de processo.
- Alocação dinâmica permite alocar memória em tempo de execução;
- A linguagem C tem quatro funções que permitem o uso desta técnica.

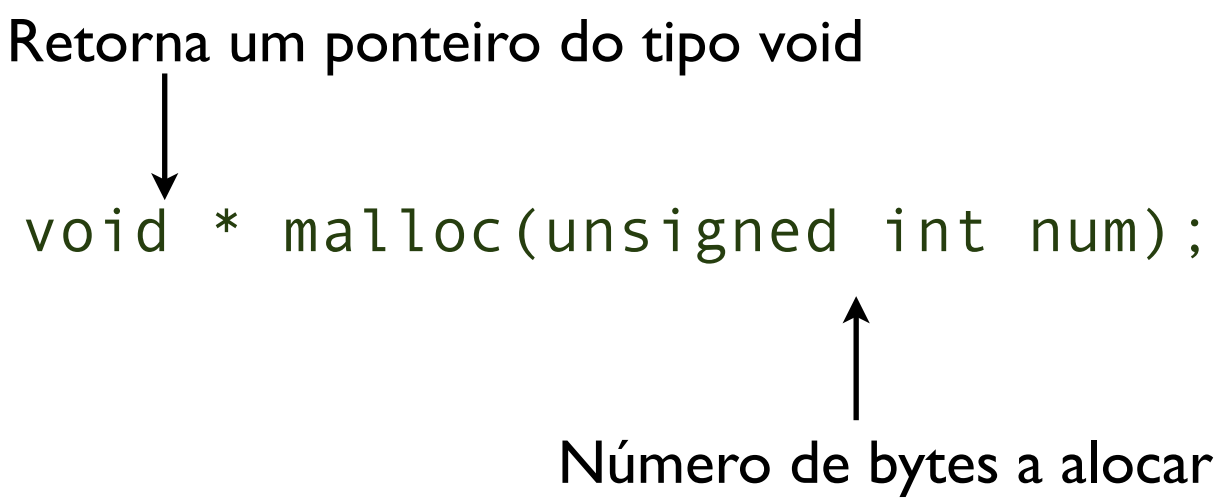
# Alocação dinâmica

- **malloc**: aloca um bloco de bytes consecutivos e retorna um *ponteiro* para o primeiro byte do espaço alocado;
- **calloc**: aloca espaços de memória em bytes, inicializa-os com zeros e retorna um ponteiro para o bloco de memória alocado;
- **free**: libera um espaço de memória previamente alocado;
- **realloc**: modifica o tamanho de um espaço de memória alocado previamente.

  - Estas funções estão disponíveis na biblioteca **stdlib.h**.

# malloc()

- Protótipo da função:




Um ponteiro nulo (NULL) é retornado se a memória não for alocada.

# malloc()

- Alocação de memória para um inteiro.

```
 //aloca espaço para 1 inteiro
 int * p = (int *)malloc(4);
//aloca espaço para 1 inteiro
int * p = (int *)malloc(sizeof(int));
//aloca espaço para um tipo struct estudante
struct estudante *e =
  (struct estudante*)(malloc(sizeof(struct estudante);
```

como alocar memória para 50 inteiros?

# malloc()

```
1  /* ex0Malloc.c */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8   int *p; //cria ponteiro para um inteiro
9   p = (int *)malloc(sizeof(int)); //aloca memória
10
11  if(p){ //testa se memória foi alocada
12    printf("Memória alocada com sucesso.\n");
13  }else{
14    printf("Não foi possivel alocar a memória.\n");
15    return 0; //finaliza o programa
16  }
17  *p = 10; //atribui valor na memória alocada
18  printf("Valor: %d\n\n", *p); //imprime o valor
19  free(p); //libera a memória
20
21  return 0;
22 }
```

# calloc()

- Protótipo da função:

Retorna um ponteiro do tipo void

```
void * calloc(unsigned int num, unsigned int size);
```

Número de elementos a alocar

Tamanho de cada objeto

Um ponteiro nulo (NULL) é retornado se a memória não for alocada.

# calloc()

- Alocação de memória para 10 inteiros.

```
//aloca espaço para 10 inteiros
int * p = (int *)calloc(10, sizeof(int));

if(!p){
   printf("\nEspaço insuficiente");
}
```

# realloc()

- Protótipo da função:

Retorna um ponteiro do tipo void

```
void * realloc(void *ptr, unsigned int num);
```

Quem será redimensionado

Quantidade de bytes a alocar

Um ponteiro nulo (NULL) é retornado se a memória não for redimensionada. Neste caso o bloco original é mantido.

# realloc()

- Realocação de memória: de 10 para 20 inteiros.

```
//aloca espaço para 10 inteiros
int * p = (int *)calloc(10, sizeof(int));
realloc(p, 20* sizeof(int));
int *pNew = realloc(p, 20 * sizeof(int));
```

problemas !

```
if(pNew){
   p = pNew;
}else{
   printf("\nMemória Insuficiente!\n");
   return 0;
}
```

realocar desta forma

ou tentar realocar menos memória



UFFS

# free

- Variáveis alocadas estaticamente desaparecem quando a função termina;
- Variáveis alocadas dinamicamente continuam a existir!
- free(ptr);
- ptr = NULL; (para não deixar ponteiros “soltos”)

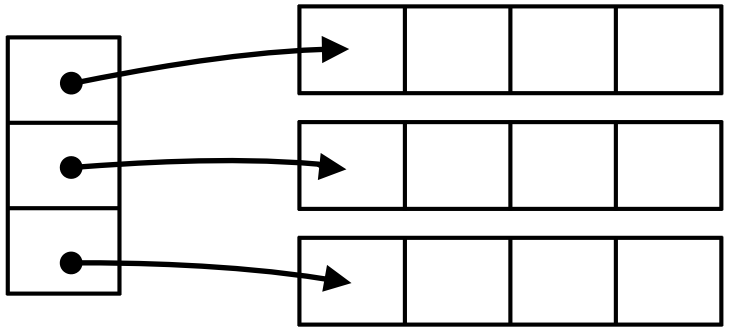
# Exercícios

```c
#include <stdio.h>

//implemente aqui a função quadrado

int main(){
  int a, b;
  printf("Informe dois valores: ");
  scanf("%d %d", &a, &b);

  quadrado(&a, &b);

  printf("O quadrado de a é: %d e de b é %d.\n", a, b);

  return 0;
}
```

# Alocação dinâmica para matrizes multidimensionais

- Uma solução é alocar uma matriz de ponteiros para ponteiros, e inicializar cada ponteiro para uma linha (dinamicamente alocada).

distancias



```
int **distancias = malloc(linhas * sizeof(int *));
for(i = 0; i < linhas; i++){
   distancias[i] = malloc(colunas * sizeof(int));
}
```

# Exercícios

1) Implemente um programa, baseado no programa `ex0Malloc.c` para alocar memória para um vetor. O número de posições do vetor será indicado via teclado. Após a atribuição de valores às posições do vetor, o programa deve imprimir (na tela) estes valores em ordem inversa à inserção.

2) Altere o programa para armazenar uma estrutura em um vetor.

```
typedef struct{
   char nome[50];
   char endereco[100];
   int matricula;
} estudante;
```

# Exercícios

3) Altere o programa do exercício 1, de forma que receba números inteiros do usuário indefinidamente. O programa finaliza quando o usuário entrar com uma letra.

a) Aloque, inicialmente, memória para 5 inteiros;

b) Caso o usuário entrar com mais inteiros, faça a realocação, alocando espaço para mais 5 inteiros e assim sucessivamente;