Introdução à Ciência da Computação I

Alocação Dinâmica

Prof. Claudio Fabiano Motta Toledo

Sumário

- Funções para alocação de memória
- Ponteiros para ponteiros

Funções para alocação de memória

- malloc(), calloc(), realloc(), free()
- São funções utilizadas para trabalhar com alocação dinâmica (em tempo de execução) de memória.
- A memória é alocada a partir de uma área conhecida como heap.

Malloc

```
void *malloc(size_t size);
```

- size = tamanho do bloco de memória em bytes.
- size_t é um tipo pré-definido usado em stdlib.h que faz size_t ser equivalente ao tipo unsigned int.
- Retorna um ponteiro para o bloco de memória alocado.

Malloc

```
void *malloc(size_t size);
```

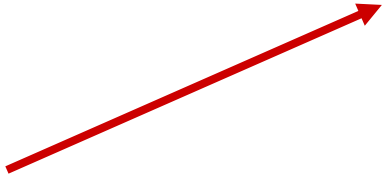
- Quando não consegue alocar memória, retorna um ponteiro nulo.
- A região alocada contém valores desconhecidos
- Sempre verifique o valor de retorno!

Malloc

```
#include <stdlib.h>

char *str;
if((str = (char *)malloc(100)) == NULL)
{
   printf("Espaco insuficiente para alocar
  buffer \n");
  exit(1);
}
printf("Espaco alocado para str\n");
```

type-casting: void para char



Malloc

```
#include <stdlib.h>

int *num;
if((num = (int *)malloc(50 * sizeof(int))) ==NULL)
{
  printf("Espaco insuficiente para alocar buffer \n");
  exit(1);
}
printf("Espaco alocado para num\n");
```

Calloc

void * calloc (size_t num, size_t size);

- A função calloc() aloca um bloco de memória para um “array” de *num* elementos, sendo cada elemento de tamanho *size.*
- A região da memória alocada é inicializada com o valor zero
- A função retorna um ponteiro para o primeiro byte
- Se não houver alocação, retorna um ponteiro nulo

Calloc

```
#include <stdlib.h>
unsigned int num;
int *ptr;
printf("Digite o numero de variaveis do tipo int: ");
scanf("%d", &num);
if((ptr = (int *)calloc(num, sizeof(int))) == NULL)
{
   printf("Espaco insuficiente para alocar \“num\” \n");
   exit(1);
}
printf("Espaco alocado com o calloc\n");
```

Malloc x Calloc

- calloc(n, sizeof(int)); ⇔ malloc(n*sizeof(int));
- A função malloc() não inicializa o espaço disponibilizado em memória.
- A função calloc() inicializa com valor zero.
- Em programas extensos, malloc() pode levar menos tempo do que calloc().
- As duas funções retornam um ponteiro do tipo void em caso de sucesso.
- Caso contrário, NULL é retornado.

Realloc

```
void * realloc (void * ptr, size_t size );
```

- A função realloc() aumenta ou reduz o tamanho de um bloco de memória previamente alocado com malloc() ou calloc()
- O argumento *ptr* aponta para o bloco original de memória.
- O argumento *size* indica o novo tamanho desejado em bytes

Realloc

- Se houver espaço para expandir, a memória adicional é alocada e prt é retornado.
- Se não houver espaço suficiente para expandir o bloco atual, um novo bloco de tamanho size é alocado em outra região da memória.
- O conteúdo do bloco original é copiado para o novo bloco.
- O espaço de memória do bloco original é liberado e a função retorna um ponteiro para o novo bloco.

Realloc

- Se o argumento *size* for zero, a memória indicada por *ptr* é liberada e a função retorna NULL.
- Se não houver memória suficiente para a realocação (nem para um novo bloco), a função retorna NULL e o bloco original permanece inalterado.
- Se o argumento *ptr* for NULL, a função atua como malloc().

Exemplo: calloc seguido de realloc

```
unsigned int num; int *ptr;
printf("Digite o numero de variaveis do tipo int: ");
scanf("%d", &num);
if((ptr = (int *)calloc(num, sizeof(int))) == NULL){
      printf("Espaco insuficiente para alocar \"num\" \n");
      exit(1);
  }
//duplica o tamanho da região alocada para ptr
if((ptr = (int *)realloc(ptr, 2*num*sizeof(int))) == NULL){
      printf("Espaco insuficiente para alocar \"num\" \n");
        exit(1);
 }
 printf("Novo espaço \“realocado\“ com sucesso\n");
```

Free

```
void free ( void * ptr );
```

- O espaço alocado dinamicamente com calloc() ou malloc() não retorna ao sistema quando o fluxo de execução deixa uma função.
- A função free() “desaloca”/libera um espaço de memória previamente alocado usando *malloc*, *calloc* ou *realloc.*
- O espaço de memória fica disponível para uso futuro.

Free

void free (void * ptr);

- A função deixa o valor de *ptr* inalterado, porém apontando para uma região inválida.
- O ponteiro não se torna NULL.
- Se for passado um ponteiro nulo, nenhuma ação será realizada.
- Ex: free(ptr);

Ponteiros para ponteiros

```
#include <stdio.h>
#include <stdlib.h>
double *alocandoVetor(int *);
double *liberandoVetor(int , float *);

void main (void)
{
  double *vetor;
  int tam;
  vetor = alocandoVetor (&tam);
  printf("tam=%d",tam);
  vetor = liberandoVetor(tam, vetor);
}
```

Ponteiros para ponteiros

```
double *alocandoVetor(int *tam)
{
  double *vet;
  do{
     printf ("\nTamanho do vetor:");
     scanf("%d",tam);
     }while(*tam<1);
  vet = (double *) calloc (*tam+1, sizeof(double));
  if (!vet) {
     printf ("\nEspaço em Memoria Insuficiente\n");
     return (NULL);
     }
  return (vet);
}
```

Ponteiros para ponteiros

```
double *liberandoVetor(int tam, float *vet)
{
  if (!vet) return (NULL);
  free(vet);
  return (NULL);
}
```

Ponteiros para ponteiros

```
#include <stdio.h>
#include <stdlib.h>

double **alocandoMatriz (int *, int *);
float **liberandoMatriz(int, int, float **);

void main (void)
{
  float **matriz;  /* matriz a ser alocada */
  int   ln, cl;   /* numero de linhas e colunas da matriz */
  matriz = alocandoMatriz(&ln, &cl);
  matriz = liberandoMatriz(ln, cl, matriz);
  }
```

Ponteiros para ponteiros

```
double **alocandoMatriz (int* linhas, int* colunas)
{
  int   i;
  double **mat;
 //Recebendo num. de linhas e colunas
  do{
     printf("Num. linhas=");
     scanf("%d", linhas);
  }while(*linhas<1);

  do{
     printf("Num. colunas=");
     scanf("%d",colunas);
  }while(*colunas<1);
```

```
  //Alocação das linhas
   mat = (double **) calloc (*linhas, sizeof(double *));
   if (!mat) {
      printf ("\nEspaço em Memoria Insuficiente\n");
      return (NULL);
   }

//Alocação das colunas
   for ( i = 0; i < *linhas; i++ ) {
       mat[i] = (double*) calloc (*colunas, sizeof(double));
       if (!mat[i]) {
          printf ("Espaço em Memoria Insuficiente");
          return (NULL);
        }
   }
   return (mat);
}
```

```
float **liberandoMatriz(int linhas, int colunas,
float **mat)
{
  int  i;

  if (!mat)
     return (NULL);

  //Liberando as linhas da matriz
  for (i=0; i<linhas; i++) free (mat[i]);

  //Liberando a matriz
  free (mat);

  return (NULL);
}
```

Exercício

- Implemente um código capaz de realizar a soma de duas matrizes, sendo os tamanhos e os conteúdos das mesmas informado pelo usuário