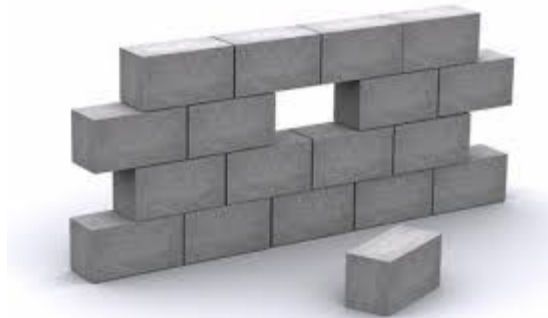




Instituto Federal
Campus Goiânia

Bacharelado em Sistemas de Informação

Estrutura de Dados I

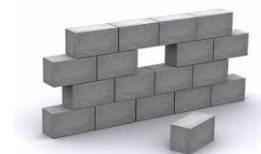


Prof. Dory Gonzaga Rodrigues



Agenda

Alocação Dinâmica de Memória





Alocação Dinâmica de Memória

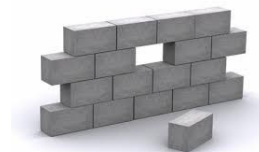
Alocação Estática x Alocação Dinâmica

- **Estática:**
 - A alocação da memória de uma variável ocorre no momento em que o programa está sendo carregado para execução;
 - O tamanho/espço da memória alocada para uma determinada variável é fixo durante a execução do programa
- **Dinâmica:**
 - A alocação da memória de uma variável ocorre em tempo de execução;
 - O tamanho/espço da memória alocada para uma determinada variável pode ser alterado durante a execução do programa;
 - A alocação é controlada pelas funções (stdlib.h)

`malloc()`

`calloc()`

`free()`





Alocação Dinâmica de Memória

Alocação Dinâmica

- Por que é interessante o uso de Alocação Dinâmica de Memória:
 - Capacidade de lidar com uma quantidade de dados cujo tamanho é imprevisível no momento em que escrevemos nosso programa;

Ex: Ao declarar um vetor cujo tamanho seja uma variável lida do usuário, na alocação estática somos obrigados a estipular um tamanho limite (máximo) e reservar um espaço (fixo) desse tamanho, mesmo que fôssemos usar menos (e nos impedindo de usar mais).





Alocação Dinâmica de Memória

Alocação Dinâmica

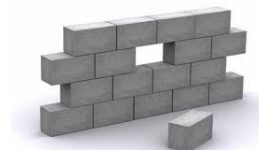
- Como é definido o espaço adequado de memória necessário para armazenar os dados de uma determinada variável ?

- Toda variável é **TIPADA**, ou seja, cada variável irá armazenar dados do **tipo** definido e declarado pelo programador, como segue:

int x, y, z;

float salário;

- Toda a alocação de memória é baseada no tamanho/espço ocupado pelos **tipos primitivos**: **INT, FLOAT, DOUBLE e CHAR**,





Alocação Dinâmica de Memória

Alocação Dinâmica

- Como é definido o espaço adequado de memória necessário para armazenar os dados de uma determinada variável ?
 - Para definir/verificar o espaço ocupado por um tipo primitivo, temos a função `sizeof()`
 - Sempre devemos usar a função `sizeof()` em vez de assumir, por exemplo, que o tamanho de um inteiro é de 2 bytes. Isso poderia causar grandes problemas quando o programa for transportado para um sistema em que o tamanho da palavra seja diferente.





Alocação Dinâmica de Memória

Alocação Dinâmica

- Como é definido o espaço adequado de memória necessário para armazenar os dados de uma determinada variável ?

- Atividade 1: Crie um programa que imprima na tela o tamanho dos tipos primitivos: **INT** **FLOAT** **DOUBLE** **CHAR**

Atenção: o tamanho é apresentado em Bytes





Alocação Dinâmica de Memória

Alocação Dinâmica

- Como é definido o espaço adequado de memória necessário para armazenar os dados de uma determinada variável ?

– Atividade 1: Crie um programa que imprima na tela o tamanho dos tipos primitivos: **INT** **FLOAT** **DOUBLE** **CHAR**

Atenção: o tamanho é apresentado em Bytes

```
#include <stdio.h>
#include <stdlib.h>
main() {
```

```
    printf("\n Tipos Primitivos e Tamanho ocupado \n");
```

```
    printf("\nO tamanho de um char:    %d bytes\n", sizeof(char) );
    printf("\nO tamanho de um int:      %d bytes\n", sizeof(int) );
    printf("\nO tamanho de um float:    %d bytes\n", sizeof(float) );
    printf("\nO tamanho de um double: %d bytes\n", sizeof(double) );
```

```
}
```





Alocação Dinâmica de Memória

Alocação Dinâmica

- Como é definido o espaço adequado de memória necessário para armazenar os dados de uma determinada variável ?
- Atividade 2: Crie um programa que imprima na tela o tamanho do **tipo estruturado** declarado abaixo:

```
typedef struct produto {  
    char    descricao[30];  
    int     quantidade;  
    double  preco_unitario;  
    double  desconto;  
    double  preco_total;  
};
```



Alocação Dinâmica de Memória

Alocação Dinâmica


- Como é definido o espaço adequado de memória necessário para armazenar os dados de uma determinada variável ?

– Atividade 2: Crie um programa que imprima na tela o tamanho do **tipo estruturado** declarado abaixo:

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

typedef struct produto {
    char    descricao[30];
    int     quantidade;
    double  preco_unitario;
    double  desconto;
    double  preco_total;
};

main() {
    printf("\n Tipos Estruturado e Tamanho ocupado \n");
    printf("\nO tamanho de um produto: %d bytes\n", sizeof(produto) );
}
```





Alocação Dinâmica de Memória

Alocação Dinâmica

- Como fazemos para alocar a memória dinamicamente ?
 - Através da função `sizeof()` temos as condições necessárias para definir o tamanho da memória e realizar a alocação dinâmica. Por exemplo, se um inteiro ocupa 4 bytes, então um vetor de n inteiros ocupará $4n$ bytes;
 - Devemos utilizar a função `malloc()` para alocar a memória de acordo com o tamanho desejado.



Alocação Dinâmica de Memória

Alocação Dinâmica

- A Função `malloc()` e `calloc()`

- O número de bytes deve ser especificado no argumento da função. Utilizamos a função `sizeof()` para definir o tamanho;
- Aloca um bloco de bytes consecutivos na memória do computador e devolve o endereço, **devendo ser convertido para o tipo do ponteiro (CAST)** que invocou a função;
- Deve ser armazenado num ponteiro do **tipo apropriado**; ou seja, o espaço alocado para inteiros deve ser armazenado num ponteiro `int *`.



Alocação Dinâmica de Memória

Alocação Dinâmica

- A Função `malloc()`
 - Atividade 3: Crie um programa que armazene vários números inteiros em um vetor de tamanho definido pelo usuário e depois liste o conteúdo do vetor.



Alocação Dinâmica de Memória

Alocação Dinâmica

- A Função `malloc()`
 - Atividade 3: Crie um programa que armazene vários números inteiros em um vetor de tamanho definido pelo usuário e depois liste o conteúdo do vetor.

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

main() {
    int n, i;
    int *v_int;

    printf("\n Digite o tamanho do vetor: ");
    scanf("%d", &n);

    v_int = (int *) malloc(n * sizeof(int) );

    printf("\n Cadastro dos valores no vetor \n");
    for (i=0; i < n; i++) {
        printf("\n Vetor[ %d ]: ", i+1);
        scanf("%d", &v_int[i]);
    }

    printf("\n Lista dos valores no vetor \n");
    for (i=0; i < n; i++) {
        printf("\n Vetor[ %d ]: %d", i, v_int[i]);
    }

    free(v_int);
    v_int = NULL;
}
```



Alocação Dinâmica de Memória

Alocação Dinâmica

- A Função `free()`
 - Quando o programa não fizer mais uso da memória alocada, o programador deve comandar a liberação deste bloco de memória;
 - A função utilizada deve ser `free()` com o ponteiro correspondente como argumento.
 - Ao liberar a memória, o sistema operacional retoma a guarda dele, permitindo que ele seja posteriormente usado por outros programas.



Alocação Dinâmica de Memória

Alocação Dinâmica

- A Função `free()`

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

main() {
    int n, i;
    int *v_int;

    printf("\n Digite o tamanho do vetor: ");
    scanf("%d", &n);

    v_int = (int *) malloc(n * sizeof(int) );

    printf("\n Cadastro dos valores no vetor \n");
    for (i=0; i < n; i++) {
        printf("\n Vetor[ %d ]: ", i+1);
        scanf("%d", &v_int[i]);
    }

    printf("\n Lista dos valores no vetor \n");
    for (i=0; i < n; i++) {
        printf("\n Vetor[ %d ]: %d", i, v_int[i]);
    }

    free(v_int);
    v_int = NULL;
}
```




Alocação Dinâmica de Memória

Alocação Dinâmica

- Redimensionamento e a Função `realloc()`
 - É possível realocar o tamanho do bloco alocado pela função `malloc()`
 - Neste caso utilizamos a função `realloc()` para redimensionar o bloco de bytes;
 - Exemplo: suponha que tenhamos que realocar um vetor de 10 inteiros para 20 inteiros. Veja como a função deve ser utilizada:

```
//aloca o vetor para 10 números
v_int = (int *) malloc(10 * sizeof(int) );
//realoca o vetor para n números
v_int = (int *) realloc(v_int, n * sizeof(int) );
```



Alocação Dinâmica de Memória

Alocação Dinâmica

- Redimensionamento e a Função `realloc()`

– Atividade 4: Crie uma função similar à função `realloc()` que retorne um ponteiro `p2` do tipo inteiro com tamanho `n` (recebido como parâmetro) preenchido com os valores inicialmente contidos no ponteiro `p1` (recebido como parâmetro)



Alocação Dinâmica de Memória

Alocação Dinâmica

- Redimensionamento e a Função `realloc()`
- Atividade 4: Crie uma função similar à função `realloc()` que retorne um ponteiro p2 do tipo inteiro com tamanho n (recebido como parâmetro) preenchido com os valores inicialmente contidos no ponteiro p1 (recebido como parâmetro)



```
#include <stdio.h>
#include <stdlib.h>
//prototipo
int * f_realloc(int n2, int n, int *v);

main() {
    int n, n2, i;
    int *v_int;
    printf("\n Digite o tamanho do vetor: ");
    scanf("%d", &n);
    v_int = (int *) malloc( n * sizeof(int) );
    printf("\n Cadastro dos valores no vetor \n");
    for (i=0; i < n; i++) {
        printf(" Vetor[ %d ]: ", i+1);
        scanf("%d", &v_int[i]);
    }
    printf("\n Lista dos valores no vetor \n");
    for (i=0; i < n; i++) {
        printf(" Vetor[ %d ]: %d \n", i, v_int[i]);
    }
    printf("\n Digite o NOVO tamanho do vetor: ");
    scanf("%d", &n2);
    v_int = f_realloc(n2, n, v_int);
    printf("\n Lista dos valores do vetor REDIMENSIONADO \n");
    for (i=0; i < n2; i++) {
        printf("\n Vetor[ %d ]: %d", i, v_int[i]);
    }
    free(v_int);
    v_int = NULL;
}
```



Alocação Dinâmica de Memória

Alocação Dinâmica

- Redimensionamento e a Função `realloc()`

– Atividade 4: Crie uma função similar à função `realloc()` que retorne um ponteiro `p2` do tipo inteiro com tamanho `n` (recebido como parâmetro) preenchido com os valores inicialmente contidos no ponteiro `p1` (recebido como parâmetro)

```
//função que faz a realocação da memória
//similar à função realloc()
int * f_realoc(int n2, int n, int *v){

    int i;
    int *ptemp;

    ptemp = (int *) malloc(n2 * sizeof(int));

    for (i=0; i < n2; i++) {
        ptemp[i] = v[i];
    }

    for (i=n; i < n2; i++) {
        ptemp[i] = NULL;
    }

    return ptemp;
};
```



Alocação Dinâmica de Memória

Alocação Dinâmica

- Trabalho (valor 1 ponto)
 - Crie um programa que gerencie o cadastro dos alunos com as funcionalidades apresentadas no menu. Além disso, o sistema deve ser capaz de armazenar um número determinado de alunos (opção 1) e de realocar caso seja alterado na opção 1 do menu:

Menu

- 1) Definir o número de Alunos
- 2) Cadastrar
- 3) Alterar
- 4) Listar
- 5) Sair

- Utilize o tipo aluno declarado a seguir:

```
typedef struct aluno {  
    char nome[30];  
    int idade;  
    char sexo;  
};
```