

UNIVERSIDADE DE BRASÍLIA

Introdução à Ciência da Computação

Disciplina: 113913



Prof: Luiz Augusto F. Laranjeira
luiz.laranjeira@gmail.com

Universidade de Brasília – UnB
Campus Gama



14. PONTEIROS E ALOCAÇÃO DE MEMÓRIA



PONTEIROS




4. Ponteiros

- Ponteiros, como o próprio nome diz, é um tipo de variável que aponta para outra (*de um tipo qualquer*).
- Na verdade um ponteiro guarda o endereço de memória (*local onde se encontra na memória*) de uma variável.





4. Ponteiros



```
int teste=20;  
int *p;  
  
p=&teste;
```



```
int teste=20;  
int *p;  
  
p=&teste;  
printf("%d\n",*p);
```

- ✓ p irá armazenar o endereço de memória da variável teste. Ou seja, ***p não armazena o valor 20***, mas sim o endereço da variável teste que, esta sim, armazena o valor 20.
- ✓ como chegar ao valor 20 usando a variável p? Resposta: *p



4. Ponteiros

▪ Outro exemplo:

```
char algo[5] = { '5', '4', '3', '2', '1' };  
char *p_char;
```

```
p_char=&algo[2];
```

- ✓ Colocamos em p_char o endereço do terceiro elemento de algo:

```
p_char[0] == '3',  p_char[1] == '2',  p_char[2] == '1'
```

- ✓ Se tivéssemos feito p_char=&algo[3], então:

```
p_char[0] == '2'   e   p_char[1] == '1'
```





4. Ponteiros

```
int vet_notas[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
int *pont_notas = NULL;
```

```
pont_notas=vet_notas;      // pont_notas = &vet_notas[0]
```

- ✓ Para imprimir a primeira e a décima nota de nosso vetor, temos duas opções:

```
print ("A primeira nota é: %d", vet_notas[0]);
```

```
print ("A primeira nota é: %d", *pont_notas);
```

```
print ("A primeira nota é: %d", pont_notas[0]);
```

```
print ("A décima nota é: %d", vet_notas[9]);
```

```
print ("A décima nota é: %d", *(pont_notas+9));
```

```
print ("A décima nota é: %d", pont_notas[9]);
```





4. Ponteiros

- Equivalência entre ponteiros

```
vet_notas[0] ==* (pont_notas);  
vet_notas[1] == *(pont_notas+1);  
vet_notas[2] == *(pont_notas+2);
```





Ponteiros e Strings – Exercício 1

Escreva um programa que leia uma string de até 80 caracteres, provida pelo usuário, e escreva na tela a string invertida.

Observação: não é necessário inverter a string lida.



Exercício 1 - Solução

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

int main()
{
    // Declarações de variáveis
    char mystring[200], *p1, *p2, c;
    int len;

    // Ler a string
    do {
        printf("\n\n");
        printf("Entre uma string nao nula: ");
        printf("\n\n ");
        gets(mystring); len = strlen(mystring);
        fflush(stdin);
    } while (len ==0 || len > 80);

    // Posicionando p1 no início da string
    p1 = mystring;

    // Posicionando p2 no fim da string
    p2 = mystring;
    while (*(p2 + 1) != '\0') p2++;
```

```
// Pulando linhas para clareza na tela
printf("\n\n");

// Escrevendo a string de forma invertida
printf("String invertida: \n\n ");
while (p2 >= p1) {
    printf("%c", *p2);
    p2--;
};

// Pulando linhas para clareza na tela
printf("\n\n");

// Colocando uma pausa na tela
system("pause");
return(0);

} // end of main
```



Ponteiros e Strings – Exercício 2

Escreva um programa que leia uma string de até 80 caracteres, provida pelo usuário, inverta os caracteres desta string e escreva na tela a string invertida.

Observação: só é permitido utilizar uma variável string para esta operação.



Exercício 2 - Solução

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

int main()
{
    // Declarações de variáveis
    char mystring[200], *p1, *p2, c;
    int len;

    // Ler a string
    do {
        printf("\n\n");
        printf("String nao nula a ser invertida: ");
        printf("\n\n ");
        gets(mystring); len = strlen(mystring);
        fflush(stdin);
    } while (len ==0 || len > 80);

    // Posicionando p1 no início da string
    p1 = mystring;

    // Posicionando p2 no fim da string
    p2 = mystring;
    while (*(p2 + 1) != '\0') p2++;
```

```
// Invertendo a string in place
while (p1 < p2) {
    c = *p1;
    *p1 = *p2;
    *p2 = c;
    p1++;
    p2--;
}

// Pulando linhas para clareza na tela
printf("\n\n");

// Escrevendo a string invertida
printf("String invertida: \n\n ");
puts(mystring);

// Pulando linhas para clareza na tela
printf("\n\n");

// Colocando uma pausa na tela
system("pause");
return(0);

} // end of main
```



ALOCAÇÃO DINÂMICA DE MEMÓRIA



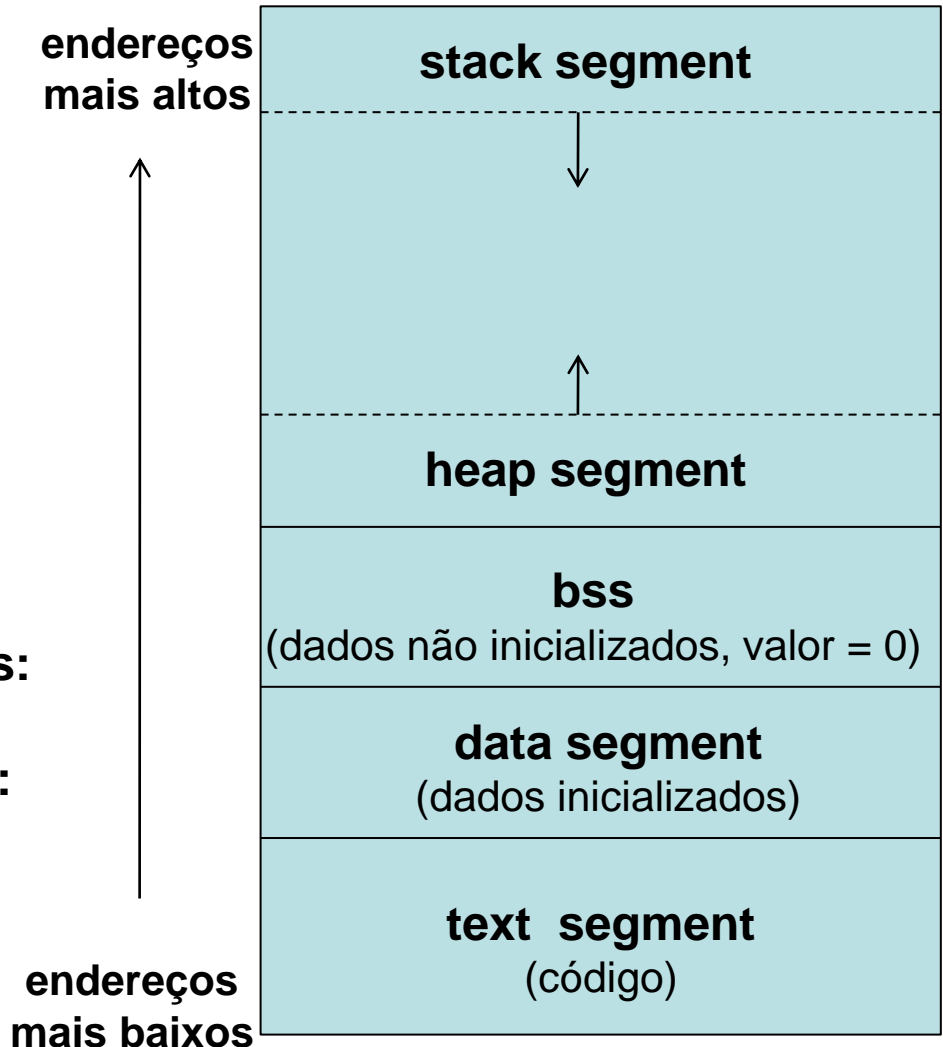
4. Segmentos de memória

ALOCAÇÃO DE MEMÓRIA

1. Variáveis globais
 - Inicializadas: **data**
 - Não-inicializadas: **bss**
2. Variáveis estáticas: **bss**
3. Variáveis locais: **stack**
4. Argumentos e endereços de retorno de chamadas de funções: **stack**
5. Alocação dinâmica (malloc/free): **heap**
6. Programas: **text**

bss = **b**lock **s**tarted by **s**ymbol

bss: nome de uma operação existente no [assembler](#) do [IBM 704](#) (mid 1950's).





4. malloc e free



■ Alocação dinâmica de memória (heap)

```
#include <stdio.h>
main()
{
    int *notas, numero, i;
    printf("Entre com o número total de alunos\n");
    scanf("%d", &numero);

    notas=(int *)malloc(numero * sizeof(int));

    for (i=0; i < numero; i++) {
        printf("Digite a nota do aluno %d", i+1);
        scanf("%d", &notas[i]);
        printf("\n A nota do aluno %d é :%d: , i+1, notas[i]);
    }

    free(notas);
}
```



Ponteiros e Alocação Dinâmica de Memória – Exercício 3



Escreva um programa que leia o número de nomes que o usuário deve entrar. Em seguida o programa deve ler os nomes dados pelo usuário. Finalmente o programa deve escrever os nomes digitados pelo usuário.

Use um vetor de ponteiros para os nomes. Cada ponteiro deste vetor deve ser inicialmente inicializado com o valor NULL. O programa deverá alocar memória (30 caracteres) para cada nome antes de lê-lo. Após ter escrito os nomes o programa deverá liberar toda a memória alocada.

Exercício 3 - Solução

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

main()
{
    // Declarações de variáveis
    int num, i;
    char *p_nomes[50];

    // Inicializando o vetor de ponteiros
    for (i=0; i < 50; i++)
    {
        p_nomes[i] = NULL;
    }

    // Pulando linhas para clareza na tela
    printf("\n\n");

    // Lendo o número de nomes
    printf("Entre o numero de nomes a serem digitados: ");
    scanf("%d", &num);
    fflush(stdin);
```

```
    // Pulando linhas para clareza na tela
    printf("\n\n");

    // Lendo os nomes
    for (i=0; i < num; i++)
    {
        p_nomes[i] = (char *) malloc(30);
        printf("Escreva o nome [%d]: ", i);
        gets(p_nomes[i]);
        fflush(stdin);
    }

    // Liberando a memória alocada
    for (i=0; i < num; i++)
    {
        free(p_nomes[i]);
    }

    // Pulando linhas para clareza na tela
    printf("\n\n");

    system("pause");
    return(0);
} // main
```



Ponteiros e Alocação Dinâmica de Memória – Exercício 4



Escreva um programa que leia o número de nomes que o usuário deve entrar. Em seguida o programa deve ler os nomes dados pelo usuário. Finalmente o programa deve escrever os nomes digitados pelo usuário.

Use um vetor de ponteiros para os nomes. Cada ponteiro deste vetor deve ser inicialmente inicializado com o valor NULL. O programa deverá alocar memória (30 caracteres) para cada nome antes de lê-lo. Após ter escrito os nomes O programa deverá liberar toda a memória alocada.

A alocação de memória e leitura de um nome deverá ser feita por meio de uma função que retorne um ponteiro para o nome lido (char *).

Exercício 4 – Solução (parcial)

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

main()
{
    // Declarações de variáveis
    int num, i;
    char *p_nomes[50];

    // Inicializando o vetor de ponteiros
    for (i=0; i < 50; i++)
    {
        p_nomes[i] = NULL;
    }

    // Pulando linhas para clareza na tela
    printf("\n\n");

    // Lendo o número de nomes
    printf("Entre o numero de nomes a serem digitados: ");
    scanf("%d", &num);
    fflush(stdin);
```

```
    // Pulando linhas para clareza na tela
    printf("\n\n");

    // Lendo os nomes
    for (i=0; i < num; i++)
    {
        p_nomes[i] = ler_nome();
    }

    // Liberando a memória alocada
    for (i=0; i < num; i++)
    {
        free(p_nomes[i]);
    }

    // Pulando linhas para clareza na tela
    printf("\n\n");

    system("pause");
    return(0);
} // main
```

Exercício 4 – Solução (função)

```
char* ler_nome()
{
    char *p_string = (char *) malloc(30);
    printf("Escreva um nome: ");
    gets(p_string);
    fflush(stdin);
    return(p_string);
}
```