
Computação para Informática - Prof. Adriano Joaquim de Oliveira Cruz
Aula Prática - Funções e ponteiros

O objetivo desta aula prática é exercitar ponteiros e funções.

1 Exercícios com ponteiros

Exercício 1: Objetivo: Notação de ponteiros.
Execute o programa 1 e observe os valores impressos procure entender o que foi impresso.

Listagem 1: Programa do problema 1.

```
#include<stdio.h>

int main (void) {
    int var; // var e um inteiro
    int *varPtr; // varPtr e um ponteiro para inteiros

    var = 7;
    varPtr = &var; // varPtr aponta para onde esta var

    printf("0 endereco de var e %p\n"
           "0 valor de varPtr e %p\n", &var, varPtr);

    printf("0 valor de var e %d\n"
           "0 valor que varPtr aponta e %d\n",
           var, *varPtr);

    // Para os fortes
    printf("Mostrando que * e & sao complementares\n"
           "&*varPtr = %p\n"
           "**&varPtr = %p\n",
           &*varPtr, *&varPtr);

    return 0;
}
```

Exercício 2: Objetivo: Usar ponteiros para passar parâmetros para funções.
Complete o programa 2. Este programa usa a função `void troca (int *a, int *b)`. Esta função troca os valores apontados por a e b.

Listagem 2: Programa do problema 2.

```
#include<stdio.h>

void troca (int *a, int *b) {
    int temp;
```

```
    temp = *a;
    *a = *b;
    /* ***** Faltava um comando aqui */

} /* Fim de troca */

int main (void) {
    int x, y;

    scanf("%d %d", &x, &y);
    troca(&x, &y);
    printf("Troquei ----> %d %d\n", x, y);
    return 0;
}
```

Exercício 3: Objetivo: Manipular variáveis usando ponteiros para variáveis previamente declaradas.

Considere o programa 3. Procure entender o que será impresso. Procure entender a diferença na notação de ponteiros em `p2 = &j` e `*p2 = temp`.

Listagem 3: Programa do problema 3.

```
#include<stdio.h>
#include<stdlib.h>
int main (void) {
    int i = 10, j = 20;
    int temp;
    int *p1, *p2;

    p1 = &i;           /* p1 recebe endereço de i */
    p2 = &j;           /* p2 recebe endereço de j */
    temp = *p1;        /* conteúdo apontado por p1 para temp */
    *p1 = *p2;         /* conteúdo apontado por p2 para o apontado p1 ← */
    /*
    *p2 = temp;        /* conteúdo apontado por p1 para p2 */
    /* O que será impresso???? */
    printf("%d %d\n", i, j);
    return 0;
}
```

Exercício 4: O programa 2 foi modificado levemente e virou 4. Complete os pedaços que faltam.

Listagem 4: Programa do problema 4.

```
#include<stdio.h>

void Troca (int *a, int *b) {
    int temp;

    temp = *a; *a = *b; *b = temp;
} /* Fim de Troca */

int main (void) {
    int x, y;
    int *px, *py;

    /* Nao se usa isto normalmente. Isto é um exercício para aprendermos
       ponteiros */
    px = &x;
    py = &y;

    /*
     * Como seria o comando scanf com px e py ao inves x e y?
     * Dica: lembre-se que era scanf("%d %d", &x, &y);
     * Agora olhe os dois comandos acima e verifique quem é igual
     * a &x e &y.
     */
    scanf("%d %d",    ,    );

    /* Como seria Troca com px e py? */
    Troca(    ,    );

    /* Como seria printf com px e py? */
    printf("Troquei ----> %d %d\n",    ,    );

    return 0;
}
```

Exercício 5: O exercício 5 mostra um exemplo de função que retorna um ponteiro. A função `achaSobrenome` retorna o ponteiro `pnome`. O que você acha que o programa imprime?

Listagem 5: Programa do problema 5.

```
#include <stdio.h>

char * achaSobrenome(char nome[]) {
    char *pnome;
    int i = 0;

    while (nome[i] != ' ') {
        i++;
    }
    i++;
    pnome = &nome[i];
    return pnome;
}

int main (void) {
    char nomeCompleto[80];
    char *p;

    puts("Entre com o seu nome e um sobrenome.");
    gets(nomeCompleto);

    p = achaSobrenome(nomeCompleto);

    puts(p);

    return 0;
}
```

2 Alocando Memória

Atenção

Nos próximos exercícios usaremos vetores com tamanhos variáveis. Para isto vamos usar funções para reservar memória e quando este espaço não for mais necessário usar uma função para liberá-lo.

Estas funções são as seguintes: `malloc()`, `calloc()` e `free()`.

Protótipos

```
#include <stdlib.h>

void *calloc (size_t nmemb, size_t size);
void *malloc (size_t size);
void free(void *ptr);
```

Observe que as variáveis `nmemb` e `size` são declaradas como sendo do tipo `size_t`. De acordo com o padrão da linguagem C representa um inteiro sem sinal. Este tipo é recomendado para qualquer variável que representa um tamanho de vetor ou um índice de vetor. Este tipo é definido no cabeçalho `<stddef.h>`, que é frequentemente incluído em outros cabeçalhos, como por exemplo `<stdio.h>`. Se você tentar usar `size_t` e aparecerem erros, simplesmente inclua `<stddef.h>` em seu programa.

Descrição

`calloc` reserva memória para um vetor de `nmemb` elementos de tamanho `size` bytes cada e retorna um ponteiro para a memória reservada. A memória é inicializada com zero.

`malloc` reserva `size` bytes e retorna um ponteiro para a memória reservada. A memória não é inicializada com zero.

`free` libera a área de memória apontada por `ptr`, que foi previamente reservada por uma chamada a uma das funções `malloc()` ou `calloc()`. Comportamento indefinido ocorre se a área já foi liberada antes ou as funções não foram chamadas antes. Se o valor de `ptr` é `NULL` nada é executado.

Retorno

Em `calloc()` e `malloc()`, o valor retornado é um ponteiro para a memória alocada, que é alinhada corretamente para qualquer tipo de variável ou `NULL` se o pedido não pode ser atendido.

`free()` returns no value.

Exemplo

Como exemplo de uso destas funções considere o problema de reservar n posições para armazenar variáveis do tipo `int`. Para isto usamos o trecho de programa mostrado em 6.

Observe que após alocar o espaço foi usada a notação de vetores comuns.

Listagem 6: Programa do problema 6.

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

int main (void) {
    int i, n, *pveter;
    float media;

    /* Define um valor para n, scanf ou n = */
    scanf("%d", &n);

    /* aloca espaco na memoria */
    pveter = (int *) malloc(n * sizeof(int));
    if (!pveter) {
        puts("Sem memória.");
        return 1;
    }

    /* A PARTIR DE AGORA VOLTAMOS PARA VETORES COMUNS */

    /* aqui uso pveter, vamos ler um vetor */
    for (i = 0; i < n; i++) {
        scanf("%d", &pveter[i]);
    }

    /* faco alguma coisa */
    media = 0.0;
    for (i = 0; i < n; i++) {
        media += pveter[i];
    }
    printf("%f\n", media);

    /* aqui nao preciso mais de pveter */
    free(pveter);

    return 0;
}
```

Exercício 6: Execute o programa 6.

Exercício 7: Agora vamos usar somente ponteiros. Execute o programa 7.

Listagem 7: Programa do problema 7.

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

int main (void) {
    int i, n, *pveter;
    float media;

    /* Define um valor para n, scanf ou n = */
    scanf("%d", &n);

    /* aloca espaco na memoria */
    pveter = (int *) malloc(n * sizeof(int));
    if (!pveter) {
        puts("Sem memória.");
        return 1;
    }

    /* aqui uso pveter, vamos ler um vetor */
    for (i = 0; i < n; i++) {
        scanf("%d", pveter + i);
    }

    /* faco alguma coisa */
    media = 0.0;
    for (i = 0; i < n; i++) {
        media += *(pveter + i);
    }
    printf("%f\n", media);

    /* aqui nao preciso mais de pveter */
    free(pveter);

    return 0;
}
```

Exercício 8: Escreva um programa que descubra qual é o maior segmento de memória que é possível reservar neste computador.

Dica: Faça um laço que tente reservar espaço, se conseguir libere-o e tente reservar um espaço ainda maior. Para liberar o espaço reservado use `free()`. Faça isto até verificar que o programa não consegue mais alocar memória.

Dica da Dica: Ir aumentando de 1 byte em 1 byte vai fazer o programa demorar muito.

Exercício 9: Modifique o programa mostrado na listagem 7 para descobrir quantos números são maiores do que a média.

Exercício 10: Modifique o programa mostrado na listagem 7 para colocar o vetor em ordem crescente.

Exercício 11: Objetivo: Aritmética de ponteiros. Vamos somar um a um ponteiro. Digite o programa 8 e diga o que ele faz, caso você digite o seu primeiro e último nomes separados por um branco.

Compare este programa com o programa 5.

Listagem 8: Exercício de cadeias de caracteres

```
#include<stdio.h>

void misterio (char *n);
int main (void)
{
    char nome[41];
    gets(nome);

    misterio(nome);

    return 0;
}

void misterio (char *n)
{
    while (*n != ' ') n++;
    n++;
    puts(n);
}
```

Exercício 12:

Objetivo: Verificar que ao se mudar o conteúdo de um vetor dentro de uma função muda-se também o conteúdo do vetor no trecho de programa que chamou a função.

Complete o programa 9 que converte um número inteiro positivo da base 10 para a base 2.

A função recebe o número na variável `numeroBase10` e retorna todos os 32 bits no vetor `numeroBase2`. O número na base 2 deve ser armazenado no vetor da seguinte maneira: bit 31 na posição 31, bit 30 na posição 30 e assim sucessivamente.

Entrada

A entrada é composta de vários conjuntos de teste. Cada caso de teste é um número inteiro positivo. A entrada termina quando um número inteiro negativo for digitado.

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas. A primeira linha identifica o conjunto de teste, no formato "**Teste n**", onde *n* é numerado a partir de 1. A segunda linha deve conter o número lido. A terceira linha deve conter todos os bits do número convertido, inclusive os zeros à esquerda. A quarta linha deve ser deixada em branco. A formato mostrado no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de entrada e saída

