Computação para Informática - Prof. Adriano Joaquim de Oliveira Cruz
Oitava Aula Prática - 29 de outubro de 2010

O objetivo desta aula prática é exercitar ponteiros e funções.

1 Exercícios com ponteiros

Exercício 1: Objetivo: Usar ponteiros para passar parâmetros para funções.
Complete o programa 1. Este programa usa a função **void troca (int *a, int *b)**. Esta função função troca os valores apontados por `a` e `b`.

Listing 1: Programa do problema 1.

```
#include<stdio.h>

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

      temp = *a;
      *a = *b;
      /* ******* Falta 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 2: Objetivo: Manipular variáveis usando ponteiros para variáveis previamente declaradas.
Considere o programa 2. Procure entender o que será impresso. Procure entender a diferença na notação de ponteiros em `p2 = &j` e `*p2 = temp`.

Listing 2: Programa do problema 2.

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

      p1 = &i;             /* p1 recebe endereco de i */
```

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

Exercício 3: O programa 1 foi modificado levemente e virou 3. Complete os pedaços que faltam.

Listing 3: Programa do problema 3.

```
#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 e 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 4: O exercício 4 mostra um exemplo de função que retorna um ponteiro. A função `achaSobrenome` retorna o ponteiro `pnome`. O que você acha que o programa imprime?

Listing 4: Programa do problema 4.

```
#include<stdio.h>

char * acheSobrenome(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 = acheSobrenome(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);
```

Pode considerar que `size_t` é equivalente a **int**.

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 5.

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

Listing 5: Programa do problema 5.

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

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

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

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

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

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

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

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

     return 0;
}
```

Exercício 5: Execute o programa 5.

Exercício 6: Agora vamos usar somente ponteiros. Execute o programa 6.

Listing 6: Programa do problema 6.

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

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

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

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

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

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

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

      return 0;
}
```

Exercício 7: Modifique o programa anterior para descobrir quantos números são maiores do que a média.

Exercício 8: Modifique o programa anterior para colocar o vetor em ordem crescente.

Exercício 9: 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 (usar `free()`) este espaço e tente reservar um espaço ainda maior. Vai assim até verificar que o programa não consegue. Ir aumentando de 1 byte em 1 byte vai fazer o programa demorar muito.

Exercício 10: Objetivo: Aritmética de ponteiros. Vamos somar um a um ponteiro. Digite o programa 7 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 4.

Listing 7: 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 11:

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 8 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

Entrada:	**Saída para o Exemplo da Entrada**
5 9 13 -1	Teste 1 5 00000000000000000000000000000101 Teste 2 9 00000000000000000000000000001001 Teste 3 13 00000000000000000000000000001101

Listing 8: Exercício de uso de vetor em funcoes

```
#include <stdio.h>

void Converter (int numeroBase10, int numeroBase2[32]);

int main(int argc, char *argv[])
{
      int nb10, nb2[32], i, teste = 1;

      while (1)
      {
            scanf("%d", &nb10);
            if (nb10 < 0 ) break;

            /*
             * Como a funcao e void nao ha valor retornado
             * e portanto nao posso atribuir a nenhuma variavel
             */

            Converter(nb10, nb2);

            printf("Teste %d\n", teste++);
            printf("%d\n", nb10);

            /* Aqui imprimo o vetor */

            /* Aqui falta codigo */

            printf("\n\n");
      }
      return 0;
}
void Converter (int numeroBase10, int numeroBase2[32])
{
      /* aqui falta codigo */

      /* Nao tem return
}
```