Aula Prática 3

Para os exercícios abaixo, use quantos arquivos .c e .h você quiser. Evite copiar e colar os códigos deste documento, pois alguns caracteres podem ser convertidos de forma errônea. Além disso, é recomendável, quando estamos aprendendo, a iniciar cada programa do zero.

Forma de entrega: compacte todos os arquivos implementados e os envie pelo Moodle.

Prazo de entrega: 1 semana (conferir no Moodle)

1) Conversão de tipo

a) Escreva uma função que recebe um número ponto flutuante como parâmetro e retorna um ponto flutuante equivalente à sua parte inteira. Exemplo: se a sua função receber -3.141592 como parâmetro, ela deve retornar -3.0. Protótipo da função:

```
float parteInteira(float x);
```

b) Escreva uma função que recebe um número ponto flutuante como parâmetro e retorna um ponto flutuante equivalente à sua parte fracionária. Exemplo: se a sua função receber -3.141592 como parâmetro, ela deve retornar -0.141592. Protótipo da função:

```
float parteFracionaria(float x);
```

c) Escreva uma função que recebe dois inteiros `x` e `y` como parâmetros e retorna um ponto flutuante correspondente à divisão do primeiro pelo segundo (`x` dividido por `y`). Protótipo da função:

```
float divInts(int x, int y);
```

d) Faça um programa para testar as funções das letras **a**, **b** e **c**. Para cada função, teste pelo menos três entradas diferentes que, idealmente, dão saídas diferentes.

2) Introdução à ponteiros

a) Escreva um programa que cria três variáveis, uma do tipo inteiro, outra do tipo ponto flutuante e outra do tipo caractere. Depois imprima na tela os endereços de memória que essas variáveis fazem acesso.

b) Altere o programa abaixo de forma a fazer com que o mesmo imprima 3 ao invés de 0. Você não pode executar nenhuma função de atribuição usando a variável `x` e nem alterar o comando `printf` da última linha do corpo da função `main`.

```
#include <stdio.h>
void main() {
      int x = 0;
      //coloque seu código aqui:

      printf("%d", x);
}
```

c) Escreva uma função de nome `soma1` que recebe como parâmetro um endereço de memória de inteiros (ponteiro para inteiro) e que soma 1 ao valor do seu conteúdo. Essa função deve ser do tipo `void`. Parte do seu protótipo é:

```
void soma1(_________________);
```

d) Resolva o exercício **2b** usando a função do `soma1` do exercício anterior.

3) Atribuição dinâmica de valores

A função `scanf` permite que o usuário entre com valores para serem armazenados em endereços da memória. Caso você tenha variáveis no seu programa, a função `scanf` pode atribuir os valores que o usuário entrar diretamente nos endereços das suas variáveis, ou seja, atribui valores às próprias variáveis.

Considere o programa abaixo, em que o usuário entra com dois valores ponto flutuante, que são atribuídos aos endereços das variáveis `x` e `y`. Altere este programa de forma a trocar os valores das duas variáveis. Você não pode alterar o último `printf`.

```
#include <stdio.h>
void main() {
      float x,y;
      scanf("%f %f", &x, &y);
      //coloque seu código aqui:

      printf("\nx = %f\ny = %f", x, y);
}
```

4) Introdução à passagem de parâmetro por referência

a) Escreva uma função de nome `troca` que recebe como parâmetros duas variáveis capazes de armazenar endereços de memória de pontos flutuantes (que tipo de variável é capaz de fazer isso?), `end_valor1` e `end_valor2`. Essa função deve trocar o conteúdo dos endereços armazenados nessas variáveis, ou seja, o conteúdo armazenado no primeiro endereço (`end_valor1`) deve ser armazenado no endereço do segundo parâmetro (`end_valor2`), e vice-versa.

b) Resolva o Exercício **3** usando a função `troca`.

5) Fast Pow

a) Neste exercício você deve usar operações de deslocamento de bits `<<` e/ou `>>`. Primeiro, faça uma função de nome `fast_pow_2` que recebe um inteiro `expoente` e retorna um `unsigned long long` contendo a potência de dois correspondente. Por exemplo, se o `expoente` passado como parâmetro for 7, a sua função deve retornar `2` elevado a `7`. Um número do tipo `unsigned long long` é um inteiro sem sinal de 64 bits. Você o trata exatamente como um `unsigned int`, mas os seu alcance é muito maior. Protótipo da função:

```
unsigned long long fast_pow_2(int expoente);
```

b) Depois, faça um programa para responder a seguinte pergunta: qual é a maior potência de 2 que um `unsigned long long` é capaz de armazenar? Faça um programa que use a função `fast_pow_2` para imprimir esse limite. Para imprimir um `unsigned long long` você deve usar o especificador de formato `%llu`.

6) Funções simples sem operadores condicionais

Implemente funções para realizar as operações abaixo sobre parâmetros recebidos como números inteiros sem sinal (`unsigned int`). Suas funções **não devem** usar qualquer operador condicional (exemplo: `if`). Dica: algumas delas podem requerer operações bit-a-bit. Abaixo um exemplo de uma função que retorna o negativo do parâmetro:

```
int neg(unsigned int number) {
   return -number;
}
```

a) DDD. Extrair código de área de números de telefone com 8 dígitos (e.g., para o telefone 3134095858 a sua função deve retornar 31). Protótipo:

```
int ddd(unsigned int number);
```

b) Soma 1 se for par. Transformar um número par no próximo número ímpar e manter um número ímpar inalterado. Exemplo: para o número 4 a sua função deve retornar 5 e para o número 5 a sua função deve retornar 5. Protótipo:

```
int soma1SePar(unsigned int number);
```

c) Par ou ímpar. Retornar `verdadeiro` se o número for par ou `falso` caso contrário. Dica: lembre dos conceitos de verdadeiro e falso para a linguagem C. Protótipo:

```
int parOuImpar(unsigned int number);
```

d) Programa principal. Faça um programa para testar as funções `DDD`, `soma1SePar` e `parOuImpar`. Para cada função, teste pelo menos três entradas diferentes que, idealmente, dão saídas diferentes.