Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Escola Politécnica – Curso de Bacharelado em Engenharia de *Software*
Disciplina: Programação de *Software* Básico – Professor: Roland Teodorowitsch
20 de março de 2025

Lista de Exercícios 2 (2025/1)

Este trabalho consiste em resolver a lista de exercícios das páginas a seguir, em C.
Para a entrega devem ser seguidas as seguintes regras:

- criar um arquivo compactado no formato ZIP com o nome do aluno no formato *camelHump* (por exemplo, para João Pedro da Silva, usar `JoaoPedroDaSilva.zip`), **SEM SUBDIRETÓRIOS** e **APENAS COM OS ARQUIVOS DE CÓDIGO-FONTE** (NÃO envie quaisquer outros arquivos, como, por exemplo, arquivos .class, .ctxt, README.txt, .o ou executáveis);
- o código-fonte deve ser **ADEQUADAMENTE INDENTADO**;
- o arquivo compactado deve conter programas em C para resolver cada um dos exercícios, salvando o código-fonte em um arquivo com o nome `Exercicio` seguido do número do exercício com **TRÊS dígitos** (por exemplo, `Exercicio001.c`, `Exercicio002.c`, ..., `Exercicio050.c`, ..., `Exercicio101.c`, ...);
- **ATENÇÃO:** os exercícios NÃO seguem necessariamente uma sequência contínua, então tome cuidado de **USAR O NÚMERO CORRETO DO EXERCÍCIO NO RESPECTIVO ARQUIVO DE CÓDIGO-FONTE**;
- **NÃO USAR ACENTOS NO NOME DE ARQUIVOS E DE FUNÇÕES** ;
- no início de cada arquivo em C, incluir um comentário informando o nome do arquivo, o nome do autor, a finalidade do programa e a versão (ou data) de criação (ou atualização);
- quando houver dados a serem lidos, **LER OS DADOS SEMPRE NA MESMA ORDEM EM QUE ELES SÃO CITADOS NO ENUNCIADO**, escolhendo os tipos numéricos adequadamente (se houver dúvida entre usar um tipo inteiro ou ou tipo real, use os exemplos de entradas e saídas que aparecem após cada exercício);
- **ESCREVER OS RESULTADOS SEMPRE NA MESMA ORDEM EM QUE ELES SÃO CITADOS NO ENUNCIADO**, escolhendo os tipos numéricos adequadamente (**NÚMEROS REAIS DEVEM SER IMPRESSOS SEMPRE COM 4 CASAS DECIMAIS**, a não ser que seja explicitamente indicado de outra forma);
- na versão final, tomar o cuidado de **NÃO IMPRIMIR NADA DIFERENTE DA SAÍDA ESPERADA** (não devem aparecer, por exemplo, mensagens pedindo que o usuário forneça ou digite determinado valor no terminal);
- a entrega deverá ser feita no dia e horário informado pelo professor em sala de aula e/ou definida na opção de entrega da plataforma moodle da PUCRS.

1. Considere a seguinte *struct*, que define os campos para numerador e denominador de frações:

```
typedef struct {
  int numerador;
  int denominador;
} fracao_t;
```

Escreva as seguintes funções para calcular e retornar, respectivamente, a soma e a multiplicação de duas frações:

```
fracao_t soma(fracao_t a, fracao_t b);
fracao_t multiplica(fracao_t a, fracao_t b);
```

Essas funções devem ser inseridas, e devem funcionar corretamente, no seguinte programa:

```
#include <stdio.h>

typedef struct {
  int numerador;
  int denominador;
} fracao_t;

int mmc(int a, int b);
fracao_t soma(fracao_t a, fracao_t b);
fracao_t multiplica(fracao_t a, fracao_t b);

int mmc(int a, int b) {
  int resto, n1, n2;
  n1 = a;
  n2 = b;
  do {
    resto = n1 % n2;
    n1 = n2;
    n2 = resto;
  } while (resto != 0);
  return (a*b)/n1;
}

int main() {
  fracao_t f1, f2, res;

  scanf("%d%d", &f1.numerador, &f1.denominador);
  scanf("%d%d", &f2.numerador, &f2.denominador);
  res = soma(f1, f2);
  printf("%d␣%d\n", res.numerador, res.denominador);
  res = multiplica(f1, f2);
  printf("%d␣%d\n", res.numerador, res.denominador);
  return 0;
}
```

Observe que o código acima apresenta os protótipos das funções, um exemplo de código para ler duas frações e testar a implementação das funções, e também a implementação de uma função para calcular o MMC (Mínimo Múltiplo Comum) – que poderá ser útil na implementação da soma de frações.

Exemplo(s):

Teste	Entrada	Saída
1	3 4 1 5	19 20 3 20
2	7 8 3 5	59 40 21 40
3	1 2 1 3	5 6 1 6
4	1 6 5 7	37 42 5 42
5	5 9 2 9	7 9 10 81
6	1 5 2 3	13 15 2 15
7	1 3 1 2	5 6 1 6
8	5 6 2 5	37 30 10 30
9	6 7 -1 3	11 21 -6 21
10	3 8 2 -8	1 8 6 -64

2. Escreva uma função chamada `eh_substring()`, que recebe duas *strings* como parâmetro e verifica se a segunda é *substring* da primeira, retornando a posição onde ela começa na primeira. Caso não seja encontrada, retorne -1.

   Essa função deve ser inserida, e deve funcionar corretamente, no seguinte programa:

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

#define TAM 100

int eh_substring(char s1[], char s2[]);

int main() {
  int t;
  char str1[TAM+1], str2[TAM+2];

  fgets(str1, TAM+1, stdin);
  t = strlen(str1);
  if ( str1[t-1] == '\n' ) str1[t-1]='\0';
  fgets(str2, TAM+1, stdin);
  t = strlen(str2);
  if ( str2[t-1] == '\n' ) str2[t-1]='\0';
  printf("%d\n", eh_substring(str1, str2) );
  return 0;
}
```

   Observe que o código acima apresenta o protótipo da função. Este código lê duas linhas da entrada padrão usando `fgets()`, removendo o final de linha (`'\n'`) de cada *string* obtida.

   Exemplo(s):

Teste	Entrada	Saída
1	0101110 111	3
2	ABCABCDABCDEABCDEGABCDEFABCDEH ABCDEF	18
3	11101010101111 1111	10
4	11101010101111 111	0
5	ABCDABCDEABCDABCAB ABCDEF	-1
6	abcdeABCD_EABCD E ABCDE	-1
7	12345678987654321 8765	9
8	A B C D E F G C D	-1
9	012345678987654321 99	-1
10	ALGORITMO RITMO	4