SME0230 - Introdução à Programação de Computadores

PRIMEIRO SEMESTRE DE 2016

Professora: Marina Andretta (andretta@icmc.usp.br)

Monitores: Amanda Carrijo Viana Figur (amanda.figur@usp.br)
Kleber Roberto Stamboni (kleber.stamboni@usp.br)
Vinicius Volponi Ferreira (vinicius.volponi.ferreira@usp.br)

Exercícios de Laboratório 4

01/04/2016

Data máxima de entrega: 02/04/2016 até às 23h59

Forma de entrega: Os exercícios deverão ser enviados por e-mail para

`exercicios.sme0230.2016@gmail.com`

O assunto do e-mail deverá ser `IPC_Lab4`. Todos os exercícios devem estar em um único arquivo `zip` com o seguinte nome `IPC_Lab4_<`*número usp*`>`.

Formato dos arquivos: No início de cada arquivo deve haver um comentário com o nome e o número USP do aluno.

Para cada algoritmo, o nome do arquivo deverá ser

`Ex<`*i*`>_<`*número usp*`>.c`, se em linguagem C ou
`Ex<`*i*`>_<`*número usp*`>.txt`, se pseudocódigo,

em que `<`*i*`>` representa o número do exercício correspondente.

Exemplo
`Ex1_123456.c`

Observações importantes:

- Trabalhos entregues fora do prazo não serão aceitos.
- É muito importante que seu programa tenha comentários e esteja bem indentado, ou seja, digitado de maneira a ressaltar a estrutura de subordinação dos comandos do programa. A avaliação dos exercícios levará isto em conta.
- Cada programa deve ser executado tantas vezes quantas forem necessárias para testar todos os casos possíveis para as entradas.

Dica: Para criar um arquivo `zip` no Linux, basta digitar no terminal

`zip <`*arquivo de saída*`>.zip <`*arquivos de entrada*`>`

Exemplo
`zip IPC_Lab3_123456.zip Ex1_123456.txt Ex2_123456.c`

Exercício 1

Existem casos em que as variáveis não suportam o tamanho do número digitado pelo usuário. Nesses casos os números são passados para vetores, de forma que o primeiro dígito do número fique na posição 0 de um vetor, o segundo dígito na posição 1, até o último dígito ficar na última posição do vetor.

Usando esse conceito, faça um algoritmo, em linguagem C, que receba os dígitos de dois números com no mínimo 25 dígitos cada e os some. Para isso, para cada um dos números, coloque cada dígito em uma posição de um vetor.

Não esqueça de pedir ao usúario a quantidade de dígitos que tem cada número.

Exercício 2

Escreva um algoritmo, em linguagem C, que crie um vetor de 100 posições, preencha esse vetor com valores aleatórios de 0 até 100 e depois o ordene na ordem crescente.

Obs: para criar valores aleatórios em linguagem C é necessário importar as bibliotecas `stdlib.h` e `time.h`. Além disso, você deve colocar, após a declaração das suas variáveis, o código: `srand(time(NULL));` e então poderá atribuir a uma variável um valor aleatório entre 0 e 100 da forma: (supondo que `i` seja sua variável) `i = rand() % 101;`

Exercício 3

Todos os números naturais maiores que 1 podem ser decompostos num produto de dois ou mais fatores. Particularmente, cada número pode ser decomposto numa quantidade finita (e única!) de números primos. Um número é dito primo quando ele possui apenas dois divisores diferentes: 1 e o próprio número.

Escreva um algoritmo, em linguagem C, que salve os primeiros 100 números primos em um vetor e, com esse vetor, faça a decomposição em primos do valor digitado pelo usuário. Por exemplo:

Caso os números do vetor não consigam decompor o número digitado:

`nº natural: 547 (o número 547 é o 101º número primo, ou seja, os 100 números salvos no vetor não conseguem decompô-lo).`

`A resposta do seu programa será: ‘‘Não foi possível decompor o número digitado’’`

Caso os números do vetor consigam decompor o número digitado:

`nº natural: 6 (não é primo, logo, os números do vetor conseguem decompô-lo).`

`A resposta do seu programa será: 2x3`

Exercício 4

Escreva um algoritmo, em linguagem C, que receba um CPF (todos os dígitos de uma vez, sem pontos e sem hifen) e diga para o usuário se esse CPF é válido. Para isso basta fazer:

1. Pegue os nove primeiros dígitos do CPF e multiplique o primeiro por 10, o segundo por 9, até o nono dígito ser multiplicado por 2. Depois, some tudo.

   Exemplo: CPF: 12345678909, 1 * 10 + 2 * 9 + 3 * 8 + 4 * 7 + 5 * 6 + 6 * 5 + 7 * 4 + 8 * 3 + 9 * 2 = 210.

2. Pegue o resultado dessa soma, divida por 11 e pegue o resto da divisão.

   Exemplo: o resto da divisão de 210 por 11 é 1.

3. Subtraia o resto da divisão de 11.

   Exemplo: 11 - 1 = 10.

4. Caso o número obtido da subtração seja maior que 9, esse número deverá virar 0. Caso contrário, ele continuará do jeito que está.

   Após isso você deve comparar esse dígito com o décimo dígito do CPF. Se eles forem diferentes o CPF é inválido, caso contrário ele pode ser válido.

   Exemplo: Como 10 > 9, então meu número de comparação será 0. Já que o décimo dígito também é 0, o CPF pode ser válido.

5. Caso os números sejam iguais no passo 4, pode continuar. Agora faça novamente o segundo passo, porém, dessa vez deve começar as multiplicações de 11 e ir até o décimo dígito.

   Exemplo: 1 * 11 + 2 * 10 + 3 * 9 + 4 * 8 + 5 * 7 + 6 * 6 + 7 * 5 + 8 * 4 + 9 * 3 + 0 * 2 = 255.

6. Repita as operações descritas nos passos 2 e 3. Depois, compare o dígito obtido com o décimo primeiro dígito do CPF. O CPF é válido se, e somente se, os dígitos obtidos forem iguais.

   Exemplo: o resto da divisão de 255 por 11 é 2, 11 - 2 = 9, o décimo primeiro dígito também é 9, logo, o CPF é válido.

Obs: para guardar números de até 20 dígitos pode usar variaveis do tipo `long long int`. Para imprimir, use `%lli`.