3ª Lista de Exercícios – Linguagem de Programação

Turma	Data de Entrega
LP- Noturno	02/12/2018

Natureza do Trabalho: em dupla (mesma dupla da lista2).

- ➔ Entregar a lista através do e-mail palomar.cris@gmail.com até às 23:59 do dia 02/12/2018 (domingo). Enviar apenas os programas fonte (.c). Os nomes dos arquivos fontes deverão ter no máximo 8 caracteres (além da extensão) e não poderão conter caracteres especiais ou acentuação.
- ➔ Cada programa deverá conter um comentário com o Ra e nome do(s) aluno(s).
- ➔ Não serão aceitas listas entregues fora do prazo determinado
- ➔ A lista merecerá uma nota de 0 a 10, sendo o critério de avaliação definido pelo professor.
- ➔ Exercícios que sejam considerados como fruto de algum tipo de fraude por parte da equipe (como cópia total ou parcial, por exemplo) atribuirão nota zero na atividade para TODOS os envolvidos.
- ➔ Os programas serão compilados e executados no ambiente Cygwin ou CodeBlocks.
- ➔ IDENTAR todos os programas.

EXERCÍCIOS

1. (8,0) Crie um programa que leia um arquivo texto contendo informações de contas a pagar de uma empresa. O programa deverá validar o arquivo, segundo critérios pré-determinados, gerar uma listagem dos erros e mostrar em tela os pagamentos que estão com os dados corretos. Ao final, o programa deverá permitir pesquisa de pagamentos por fornecedor, além de totalizar a soma dos valores a pagar.

   Layout do arquivo

Campo	Posição	Descrição	Observação
Código	1 a 5	Código do fornecedor	5 dígitos
Nome	6 a 40	Nome do Fornecedor	35 dígitos
Valor a Pagar	41 a 49	Valor a Pagar	9 dígitos sendo os centavos os dois últimos.
Banco	50 a 52	Código do Banco	3 dígitos
Agência	53 a 56	Código da Agência	4 dígitos
Conta Corrente	57 a 62	Número da conta bancária	6 dígitos
CNPJ	63 a 76	CNPJ do fornecedor	14 dígitos

a) Leitura e validação: ler o arquivo pagamento.txt (de acordo com o layout acima). Para cada registro lido validar se a conta a pagar está válida. São consideradas contas inválidas aquelas cujo banco seja diferente de "001" e o CNPJ não esteja informado (zerado).
b) Geração de Arquivo de Erros: cada conta inválida deverá ser gravada em um arquivo de erros (erro.txt) que deverá ser recriado a cada execução do programa.
c) As contas válidas deverão ser inseridas em um array de struct e exibidas em tela de acordo com o layout abaixo:

```
Conta:10010          Eletropaulo 5000.50  001  1234  023120  11.111.111/1111-11
Conta:10011                 SAAE 500.00  001  1234  023120  11.111.111/1111-12
Conta:10012                  GVT 400.00  001  1234  023120  11.111.111/1111-13
Conta:10013                 VIVO 10.10  341  1234  023120  11.111.111/1111-14
Conta:10014               Soroca 1000.00  341  1234  023120  11.111.111/1111-15
Conta:10015        Ze da Esquina 2000.00  341  1234  023120  11.111.111/1111-16
Conta:10010          Eletropaulo 3000.00  341  1234  023120  11.111.111/1111-11
Conta:10016                  TIM 4000.00  341  1234  023120  11.111.111/1111-19
```

Total: xxxxx,xx

Ao final, deverá ser exibido o valor total das contas válidas.

d) Finalmente, o usuário poderá pesquisar uma conta por nome de fornecedor (para isto utilize algum método de ordenação, como bolha, inserção ou seleção) e implemente a pesquisa binária.

⇨ Deverá ser utilizada a função SUBSTRING criada na lista 2 para "quebrar" a linha lida do arquivo e separá-la de acordo com o layout do arquivo. Esta função poderá ser colocada em um outro programa.c.
⇨ Deverão ser retirados os espaços ao final do nome do fornecedor antes de armazená-lo na estrutura.

2. (2,0) Escreva uma função recursiva que converta uma string em um número.
Ex.:

2	4	6	\0		

Número: 246

➔ Não utilizar função ATOI nem qualquer outra, desenvolver sua própria função de conversão.
➔ Finalizar o programa quando a string "FIM" for informada.
➔ Utilizar notação ponteiro