MC202 — ESTRUTURAS DE DADOS

Laboratório 03 — Controle de estoque

Tarefa

Um parente distante que faz bolos caseiros está expandindo os negócios e abrindo uma loja. Por conta disso, precisa automatizar o controle de quais produtos irão expirar e quanto precisará gastar para repor o estoque após os produtos expirarem. Como o(a) consultor(a) de TI da família, sua tarefa é fazer um programa que faça esse controle.

Uma maneira de para manipular todas essas informações sobre os produtos é usando *Tipos Abstratos de Dados* (TAD). Por exemplo, um produto em estoque pode ser representado pela struct de nome Produto:

```
typedef struct Produto
{
      char nome[MAX_NOME_PRODUTO];
      float preco;
      int quantidade;
      Data validade;
} Produto;
```

Além da estrutura, existem diversas operações que podem ser realizadas sobre um produto. Neste problema, são necessárias as operações de criar um novo produto, ler um produto do teclado e verificar se o produto está vencido.

Repare que um produto contém uma data, que também pode ser representada por um tipo abstrato de dados. De maneira similar, um estoque também pode ser representado por um tipo abstrato de dados que contém uma coleção (um vetor) de produtos.

Você irá receber as interfaces dos três tipos abstratos de dados já definidas (os arquivos .h), correspondendo a uma Data, um Produto e um Estoque. Sua tarefa será ler e entender cada interface disponibilizada, implementar todas as operações listadas e utilizar os tipos abstratos de dados adequadamente no programa cliente.

Entrada

A primeira linha da entrada consiste da data em que a compra será realizada. A segunda linha contém um inteiro N ($0 \leq N \leq 100$), que será o número de produtos no estoque. Em cada uma das N linhas seguintes, estão as informações de um produto na seguinte ordem Nome, Preço, Quantidade em estoque, Data de validade.

Na próxima linha, há um inteiro M ($0 \leq M \leq N$) seguido de M linhas com a quantidade desejada em estoque de cada item. Cada linha está no seguinte formato: Nome, Quantidade desejada.

Obs: o nome de um produto é uma palavra (string) de no máximo 20 caracteres, o preço é um número de ponto flutuante, as quantidades são todas inteiras e uma data está no formato DD/MM/AA.

Saída

A saída consiste de M linhas, cada uma com o nome e o valor que será gasto com esse produto para repor a quantidade desejada na data da compra, na mesma ordem presente nas M linhas da entrada.

Um produto é comprado quando a quantidade desejada for maior do que a quantidade de produto não vencida. Se não for preciso repor um produto, o valor gasto será 0.00.

Exemplos

Entrada

```
3/3/19
3
baunilha 2.50 2 4/3/19
pimenta 3.14 3 2/3/19
farinha 2.00 5 1/3/20
3
farinha 4
baunilha 3
pimenta 10
```

Saída

```
farinha 0.00
baunilha 2.50
pimenta 31.40
```

Entrada

```
28/3/19
5
abacate 6.10 2 19/3/19
chocolate 1.00 4 20/3/19
farinha 5.55 3 20/3/20
nesquik 8.90 5 30/8/22
```

```
ovos 0.60 12 19/3/19
3
abacate 9
chocolate 25
ovos 24
```

Saída

```
abacate 54.90
chocolate 25.00
ovos 14.40
```

Critérios específicos

- Você receberá os seguintes arquivos prontos e não deverá modificá-los:
    - data.h: interface de uma data
    - produto.h: interface de um produto
    - estoque.h: interface do estoque
- Deverão ser submetidos os seguintes arquivos:
    - lab03.c: programa principal (cliente dos TADs)
    - data.c, produto.c, estoque.c: deve implementar as interfaces correspondente
- Tempo máximo de execução: 1 segundo.

Testando

Para compilar usando o Makefile fornecido e verificar se a solução está correta basta seguir o exemplo abaixo.

```
make
./lab03 < arq03.in > arq03.out
diff arq03.out arq03.res
```

onde `arq03.in` é a entrada (casos de testes disponíveis no SuSy) e `arq03.out` é a saída do seu programa. O Makefile também contém uma regra para testar todos os testes de uma vez; nesse caso, basta digitar:

```
make testar_tudo
```

Isso testará o seu programa com os casos abertos. Após o prazo, os casos de teste fechados serão liberados e podem ser baixados digitando-se:

`make baixar_fechados`

Para ver quanto tempo seu programa demora em cada teste, digite:

`make tempo`

Observações gerais

No SuSy, haverá 3 tipos de tarefas com siglas diferentes para cada laboratório de programação. Todas possuirão os mesmos casos de teste. As siglas são:

1. **DRAFT**: Esta tarefa serve para testar o programa no SuSy antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém arquivos submetidos aqui não serão corrigidos.
2. **ENTREGA**: Esta tarefa tem limite de *uma única submissão* e serve para entregar a versão final dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa: submeta aqui quando não for mais fazer alterações no seu programa.
3. **FORAPRAZO**: Esta tarefa tem limite de *uma única submissão* e serve para entregar a versão final após o prazo estabelecido para o laboratório, mas com nota reduzida (conforme a ementa). O envio nesta tarefa irá substituir a nota obtida na tarefa ENTREGA apenas se o aluno tiver realizado as correções sugeridas no feedback ou caso não tenha enviado anteriormente em ENTREGA.

Observações sobre SuSy:

- Versão do GCC: gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)
- Flags de compilação:
  `-std=c99 -Wall -Werror -Werror=vla -pedantic-errors -g -lm`

Além das observações acima, esse laboratório será avaliado pelos critérios gerais:

- Indentação de código e outras boas práticas, tais como:
  - uso de comentários (úteis e apenas quando forem relevantes);
  - código simples e fácil de entender;
  - sem duplicidade (partes que fazem a mesma coisa).
- Organização do código:
  - tipos de dados criados pelo usuário e funções bem definidas e tão independentes quanto possível.
- Corretude do programa:

- programa correto e implementado conforme solicitado no enunciado;
- inicialização de variáveis sempre que for necessário;
- dentre outros critérios.