Enunciado do Projeto 1 - IAED 2023/24

Data de entrega: 01 de abril 2024, às 19h59

LOG alterações

- 8mar24 - Publicação do enunciado.

1. Introdução

Pretende-se a construção de um sistema de gestão de parques de estacionamento. O seu sistema deverá permitir a definição de parques e o registo de entradas e saídas de veículos, assim como a sua consulta e facturação.

A interacção com o programa deverá ocorrer através de um conjunto de linhas compostas por uma letra (comando) e um número de argumentos dependente do comando a executar. Pode assumir que todo o *input* fornecido respeitará os tipos indicados, por exemplo onde é esperado um valor inteiro decimal nunca será introduzida uma letra. Os possíveis comandos são listados na tabela seguinte e indicam as operações a executar.

Comando	Acção
q	termina o programa
p	Cria um parque de estacionamento com o regime de facturação ou lista os parques existentes
e	Regista a entrada de um veículo
s	Regista a saída de um veículo
v	Lista as entradas e saídas de um veículo
f	Mostra a facturação de um parque de estacionamento
r	Remove um parque de estacionamento do sistema

2. Especificação do problema

O objectivo do projeto é ter um sistema de gestão de parques de estacionamento até um máximo de **20** parques de estacionamento.

Cada **parque** é caracterizado por um nome, uma capacidade máxima e um regime de facturação. No nome que descreve o parque podem ocorrer caracteres brancos (espaços ou tabulador horizontal `\t`). Neste caso, o nome é representado entre aspas. Caso não contenha caracteres brancos, o nome pode ser delimitado por aspas ou não. O nome nunca contém o carácter aspa ou o carácter `\n` na sua descrição.

O valor a facturar é definido em intervalos de 15 minutos. Dependendo da duração da permanência do veículo no parque, o valor a facturar em cada período varia. O regime de facturação de todos os parques é definido por três valores:

- *X* : o valor por cada 15 minutos na 1ª hora;
- *Y* : o valor por cada 15 minutos após a 1ª hora;
- *Z* : o valor máximo diário (24 horas);

Nos primeiros 4 períodos de 15 minutos é cobrado *X* por cada período. A partir da 1ª hora é cobrado *Y* por cada período adicional de 15 minutos. No entanto, se o período de permanência no parque for inferior a 24 horas, então o valor máximo a cobrar não pode ser superior a *Z*. Note-se que no tarifário temos sempre que *Z* > *Y* > *X*.

Os veículos podem permanecer num parque por mais de 24 horas. Nesse caso, é aplicado o valor máximo diário *Z* a cada período completo de 24 horas que permanecer no parque. O valor a cobrar pelo período remanescente é calculado de acordo com o definido para um período inferior a 24 horas como descrito no parágrafo anterior.

Os valores monetários são representados como números reais em vírgula flutuante e devem ser impressos com duas casas decimais (`%.2f`). As datas são representadas no formato DD-MM-AAAA e as horas são representadas no formato HH:MM.

Os accionistas maioritários da empresa que gere os paques de estacionamento são muito supersticiosos. No dia 29 de fevereiro o parque está sempre fechado (não há entradas ou saídas). Dessa forma, não se cobram as 24 horas correspondentes a esse dia aos veículos que permanecem nos parques.

Um **veículo** é caracterizado por uma matrícula. Uma matrícula corresponde a uma sequência de 3 pares de caracteres separados pelo carácter *'-'*, sendo que um par apenas pode conter letras maiusculas de *'A'* a *'Z'* ou apenas pode conter dígitos decimais. Um par não pode ser composto por uma letra e um dígito. Uma matrícula tem que conter sempre pelo menos um par de letras e pelo menos um par de dígitos decimais.

Sempre que um veículo entra num parque de estacionamento, o número de lugares disponíveis diminui em uma unidade. O lugar é libertado quando o veículo sai do parque.

O registo de entradas e saídas de veículos segue sempre a ordem cronológica. Ou seja, nunca pode haver um registo de entrada ou saída anterior ao último registo (entrada ou saída) no sistema. Adicionalmente, se um veículo tiver dado entrada num parque, não pode ser registada outra entrada (no mesmo ou em outro parque) sem o veículo ter saído.

3. Dados de Entrada

O programa deverá ler os dados de entrada a partir da linha de comandos do terminal. Nenhuma linha de comandos excede **BUFSIZ** *bytes* (*8192 bytes* na maioria dos sistemas).

Durante a execução do programa as instruções devem ser lidas do terminal (*standard input*) na forma de um conjunto de linhas iniciadas por um carácter, que se passa a designar por comando, seguido de um número de informações dependente do comando a executar; o comando e cada uma das informações são separados por pelo menos um carácter branco.

Os comandos disponíveis são descritos de seguida. Os caracteres `<` e `>` são utilizados apenas na descrição dos comandos para indicar os parâmetros. Os parâmetros opcionais estão indicados entre caracteres `[` e `]`. As repetições estão indicadas entre caracteres `{` e `}`. Cada comando tem sempre todos os parâmetros necessários à sua correcta execução. Os caracteres `...` indicam possíveis repetições de um parâmetro.

Cada comando indica uma determinada acção que se passa a caracterizar em termos de formato de entrada, formato de saída e erros a retornar.

Se o comando gerar mais de um erro, deverá ser indicado apenas o primeiro.

- **q** - termina o programa:
  - Formato de entrada: `q`
  - Formato de saída: NADA
- **p** - Cria um parque de estacionamento com o regime de facturação ou lista os parques existentes:
  - Formato de entrada: `p [ <nome-parque> <capacidade> <valor-15> <valor-15-apos-1hora> <valor-max-diario> ]`

    - Formato de saída sem argumentos: `<nome-parque> <capacidade> <lugares-disponiveis>`, pela ordem de criação dos parques.
    - Formato de saída com argumentos: NADA
    - Erros:
        - `<nome-parque>: parking already exists.` no caso do nome do parque já existir.
        - `<capacidade>: invalid capacity.` no caso da capacidade ser menor ou igual a 0.
        - `invalid cost.` no caso de um dos custos ser menor ou igual a 0 ou se os valores do tarifário não forem crescentes.
        - `too many parks.` no caso do número de parques criados estar no limite.
- **e** - Regista a entrada de um veículo:
    - Formato de entrada: `e <nome-parque> <matrícula> <data> <hora>`
    - Formato de saída: `<nome-parque> <lugares-disponiveis>`.
    - Erros:
        - `<nome-parque>: no such parking.` no caso do nome do parque não existir.
        - `<nome-parque>: parking is full.` no caso do parque estar cheio.
        - `<matrícula>: invalid licence plate.` no caso da matrícula ser inválida.
        - `<matrícula>: invalid vehicle entry.` no caso do carro estar dentro de um parque.
        - `invalid date.` no caso da data ou hora ser inválida ou anterior ao último registo de entrada ou saída no sistema.
- **s** - Regista a saída de um veículo:
    - Formato de entrada: `s <nome-parque> <matrícula> <data> <hora>`
    - Formato de saída: `<matrícula> <data-entrada> <hora-entrada> <data-saída> <hora-saída> <valor-pago>`.
    - Erros:
        - `<nome-parque>: no such parking.` no caso do nome do parque não existir.
        - `<matrícula>: invalid licence plate.` no caso da matrícula ser inválida.
        - `<matrícula>: invalid vehicle exit.` no caso do carro não estar dentro do parque indicado.
        - `invalid date.` no caso da data ou hora ser inválida ou anterior ao último registo de entrada ou saída no sistema.
- **v** - Lista as entradas e saídas de um veículo:
    - Formato de entrada: `v <matrícula>`
    - Formato de saída: `<nome-parque> <data-entrada> <hora-entrada> <data-saída> <hora-saída>`, ordenados primeiro pelo nome do parque e depois pela data e hora de entrada. Se o veículo estiver dentro de um parque, não mostra a data e hora de saída associada a essa entrada.
    - Erros:
        - `<matrícula>: invalid licence plate.` no caso da matrícula ser inválida.
        - `<matrícula>: no entries found in any parking.` no caso da matrícula ser válida, mas não tem registos de entradas em parques.
- **f** - Mostra a facturação de um parque de estacionamento:
    - Formato de entrada: `f <nome-parque> [ <data> ]`
    - Formato de saída com um argumento: `<data> <valor-facturado>`, ordenados pela data. Nesta opção, é mostrado no output o resumo da facturação diária do parque de estacionamento.
    - Formato de saída com dois argumentos: `<matrícula> <hora-saída> <valor-pago>`, ordenados pela data de saída. Nesta opção, é mostrado no output a lista dos valores facturados num determinado dia.
    - Erros:
        - `<nome-parque>: no such parking.` no caso do nome do parque não existir.

        - `invalid date.` no caso da data ser inválida ou posterior ao dia do último registo de entrada ou saída no sistema.
- **r** - Remove um parque do sistema e todas as entradas e saídas de veículos desse parque:
    - Formato de entrada: `r <nome-parque>`
    - Formato de saída: `<nome-parque>`, ordenados pelo nome do parque.
    - Erros:
        - `<nome-parque>: no such parking.` no caso do nome do parque não existir.

Só poderá usar as funções de biblioteca definidas em `stdio.h`, `stdlib.h`, `ctype.h` e `string.h`

Nota importante: não é permitida a utilização da instrução `goto`, da declaração `extern`, nem da função `qsort` nativa do C e nenhum destes *nomes* deve aparecer no vosso código.

Exemplos de utilização dos comandos

Comando `p`

O comando `p` sem argumentos permite listar todos os parques existentes no sistema.

```
p
```

O comando `p` com argumentos permite adicionar um parque ao sistema. Neste caso, se não houver erros no input, não há nada a mostrar no output.

```
p Saldanha 200 0.20 0.30 12.00
p "CC Colombo" 400 0.25 0.40 20.00
```

Comando `e`

O comando `e` permite registar a entrada de um veículo num parque. A data de entrada não pode ser anterior ao último registo de entrada ou saída de um outro veículo e o veículo não pode estar registado como estando dentro de um parque de estacionamento.

```
e Saldanha AA-00-AA 01-03-2024 08:34
```

Comando `s`

O comando `s` permite registar a saída de um veículo. O veículo tem que estar registado como estando dentro do parque indicado e a data de saída não pode ser anterior ao último registo de entrada ou saída de um outro veículo.

```
s Saldanha AA-00-AA 01-03-2024 10:59
```

Comando `v`

O comando `v` permite listar a utilização dos parques por um determinado veículo.

```
v AA-00-AA
```

Comando `f`

O comando `f` permite listar a facturação de um parque de estacionamento. O valor cobrado a um veículo é facturado na data de saída do veículo, independentemente da data de entrada.

O exemplo seguinte mostra o resumo da facturação diária do parque de estacionamento.

```
f Saldanha
```

O exemplo seguinte mostra todos os valores facturados no parque de estacionamento num determinado dia.

```
f Saldanha 01-03-2024
```

Comando `r`

O comando `r` permite remover um parque de estacionamento do sistema. Todas as entradas e saídas de veículos do parque removido são também removidas do sistema.

```
r "CC Colombo"
```

4. Compilação e teste

O compilador a utilizar é o `gcc` com as seguintes opções de compilação: `-O3 -Wall -Wextra -Werror -Wno-unused-result`. Para compilar o programa deve executar o seguinte comando:

```
  $ gcc -O3 -Wall -Wextra -Werror -Wno-unused-result -o proj1 *.c
```

O programa deverá escrever no *standard output* as respostas aos comandos apresentados no *standard input*. As respostas são igualmente linhas de texto formatadas conforme definido anteriormente neste enunciado. Tenha em atenção ao número de espaços entre elementos do seu output, assim como a ausência de espaços no final de cada linha. Procure respeitar escrupulosamente as indicações dadas.

Ver os exemplos de input e respectivos output na pasta `public-tests/`.

O programa deve ser executado da forma seguinte:

```
  $ ./proj1 < test.in > test.myout
```

Posteriormente poderá comparar o seu output (`*.myout`) com o output previsto (`*.out`) usando o comando `diff`,

```
  $ diff test.out test.myout
```

Para testar o seu programa poderá executar os passos indicados acima ou usar o comando `make` na pasta `public-tests/`.

5. Entrega do Projeto

Será criado um repositório `git` para cada aluno desenvolver e submeter o projeto. Este repositório será criado no GitLab da RNL e será activado quando da publicação deste enunciado.

Na sua submissão do projeto deve considerar os seguinte pontos:

- Considera-se que os seus ficheiros de desenvolvimento do projeto (`.c` e `.h`) estão na raiz do repositório e não numa directoria. *Qualquer ficheiro fora da raíz não será considerado como pertencendo ao seu projeto*.
- A última versão que estiver no repositório da RNL será considerada a submissão para avaliação do projeto. Qualquer versão anterior ou que não esteja no repositório não será considerada na avaliação.
- Quando actualizar os ficheiros `.c` e `.h` na directoria `src` no seu repositório na RNL, esta versão será avaliada e será informado se essa versão apresenta a resposta esperada num conjunto de casos de teste. Tal como no repositório dos laboratórios, o resultado da avaliação automática será colocado no *repositório de feedback* do aluno.
- Para que o sistema de avaliação seja executado, tem que esperar pelo menos 10 minutos. Sempre que fizer uma actualização no repositório, começa um novo período de espera de 10 minutos. Exemplos de casos de teste serão oportunamente fornecidos.
- Data limite de entrega do projeto: **01 de abril de 2024, às 19h59m**. Até à data limite poderá efectuar o número de submissões que desejar, sendo utilizada para efeitos de avaliação a última versão. Deverá portanto verificar cuidadosamente que a última versão no repositório GitLab da RNL corresponde à versão do projeto que pretende que seja avaliada. Não existirão excepções a esta regra.
- Será dado um valor de bónus a quem entregar o projeto até dia **27 de março de 2024, às 19h59**. Caso haja qualquer alteração ao repositório após esta data e hora, o valor de bónus não será considerado.

6. Avaliação do Projeto

Na avaliação do projeto serão consideradas as seguintes componentes:

1. A primeira componente avalia o desempenho da funcionalidade do programa realizado. Esta componente é avaliada entre 0 e 16 valores.
2. A segunda componente avalia a qualidade do código entregue, nomeadamente os seguintes aspectos: comentários, indentação, estruturação, modularidade, abstracção, entre outros. Esta componente poderá variar entre -4 valores e +4 valores relativamente à classificação calculada no item anterior e será atribuída posteriormente. Algumas *guidelines* sobre este tópico podem ser encontradas guidelines.
3. A classificação da primeira componente da avaliação do projeto é obtida através da execução automática de um conjunto de testes num computador com o sistema operativo GNU/Linux. Torna-se portanto essencial que o código compile correctamente e que respeite o formato de entrada e saída dos dados descrito anteriormente. Projetos que não obedeçam ao formato indicado no enunciado serão penalizados na avaliação automática, podendo, no limite, ter 0 (zero) valores se falharem todos os testes. Os testes considerados para efeitos de avaliação poderão incluir (ou não) os disponibilizados na página da disciplina, além de um conjunto de testes adicionais. A execução de cada programa em cada teste é limitada na quantidade de memória que pode utilizar, e no tempo total disponível para execução, sendo o tempo limite distinto para cada teste.
4. Note-se que o facto de um projeto passar com sucesso o conjunto de testes disponibilizado na página da disciplina não implica que esse projeto esteja totalmente correcto. Apenas indica que passou alguns testes com sucesso, mas este conjunto de

testes não é exaustivo. É da responsabilidade dos alunos garantir que o código produzido está correcto.

5. Em caso algum será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. A totalidade dos ficheiros de teste usados na avaliação do projeto serão disponibilizados na página da disciplina após a data de entrega.
6. Os projectos que não usarem alocação dinâmica de memória nas estruturas de dados de tamanho variável serão penalizadas na segunda componente de avaliação até um máximo de 4 valores. Quem optar por alocação de memória estática pode considerar os seguintes limites nas estruturas de dados:
    - Comprimento do nome dos parques de estacionamento: 50 *bytes*. Notar que um carácter acentuado em *utf-8* utiliza mais de um *byte*. Por exemplo `praça` tem 5 letras mas ocupa 6 *bytes* (`char` em **C**).
    - Número máximo de veículos simultaneamente no sistema: 10000
    - Número máximo de entradas de um veículo nos parques de estacionamento: 10000

7. Dicas para Desenvolvimento do Projeto

Abaixo podem encontrar algumas dicas simples que facilitam o despiste de erros comuns no desenvolvimento do projeto. Sugerimos que **desenvolvam os vossos projetos de forma incremental e que testem as vossas soluções localmente antes de actualizarem o repositório remoto**.

Sugerimos que sigam os seguintes passos:

1. Desenvolva e corrija o código de forma incremental garantindo que compila sem erros nem *warnings*. Não acumule uma série de erros pois o *debug* é tanto mais complexo quanto a dimensão da base de código a analisar.
2. Garanta que está a ler o *input* e a escrever o *output* correctamente, em particular garanta que as *strings* não ficam com espaços extra, `\n` no final, que a formatação está correcta e de acordo com o que está no enunciado, *etc*.
3. Procure desenvolver os comandos pela ordem apresentada.
4. Teste isoladamente cada comando e verifique que funciona correctamente.