INSTITUTO SUPERIOR TÉCNICO

Introdução aos Algoritmos e Estruturas de Dados
2016/2017

Enunciado do 2º Projecto

Data de entrega: Sábado, 20 de Maio de 2017 (23h59)

1 Especificação do Programa

Neste projecto pretende-se desenvolver um programa que permita registar as entradas e saídas de produtos num armazém. Cada produto possui um *código identificativo* (que em seguida também designaremos por *chave*) e a *existência actual*, expressa em número de unidades.

O programa deverá receber como *input* um conjunto de linhas que começam com um dos caracteres listados na Tabela em baixo e que indicam as operações a executar. Dada uma chave, a opção `a` permite introduzir novas existências de um produto. A opção `l` permite listar as existências de cada produto, enquanto que a opção `m` lista o produto com maior número de unidades em stock. A opção `r` permite eliminar uma chave. Por fim, o comando `x` deverá terminar o programa, escrevendo o número de produtos (ou chaves) diferentes guardados no sistema.

Comando	Descrição
`a`	Incrementa ou reduz o número de unidades associadas ao produto com o código dado. Se o código não existir deverá ser criado um novo produto com esse código.
`l`	Lista alfabeticamente todos os produtos.
`m`	Escreve o produto com o maior número de unidades em stock.
`r`	Remove o produto com o código dado.
`x`	Termina o programa

Uma vez que as funcionalidades pretendidas são conceptualmente muito simples, pretende-se que os alunos dediquem particular esforço à implementação de soluções computacionalmente eficientes, o que será valorizado em termos da avaliação. Assim, não será apenas tida em consideração a correção do código produzido, mas também a eficiência do mesmo.

2 Dados de entrada e de saída

O programa deverá ler os dados de entrada a partir do *standard input*. As opções da Tabela anterior têm a seguinte sintaxe.

- Comando `a`:
  `a <chave> <valor>`

A `<chave>` representa o *código identificativo* ou *chave* de um produto. Corresponde a uma sequência de 8 caracteres (sem espaços) sobre o alfabeto $\Sigma = \{0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f\}$. O `<valor>` é um número inteiro representando o número de unidades adicionadas ou removidas desse produto[1]. Se `<valor>` for positivo deverá incrementar esse número às existências desse produto. Se for negativo, deverá subtrair (o módulo) desse valor às existências desse produto. Caso a subtração origine um número negativo, deverá registar um número de unidades igual a zero para esse produto, sem que este seja eliminado dos registos.
Se não existir nenhum produto com a chave introduzida, o comando deverá criar uma nova entrada com um número de unidades dadas por `<valor>`. Caso o `<valor>` do novo produto seja negativo, deverá assumir um número de unidades igual a zero para esse novo produto. O comando `a` não envolve qualquer *output*.

- Comando `l`:
  Não recebe argumentos. Deverá listar todas as chaves e unidades associadas aos produtos guardados no sistema. O comando não deverá ter output caso não haja elementos para listar. A listagem deverá ser feita por ordem alfabética das chaves dos produtos, com um par chave-valor por cada linha, com o seguinte *layout* (ver exemplos de input e output em baixo):

  ```
  Chave Unidades
  Chave Unidades
  Chave Unidades
  ...
  ```

- Comando `m`:
  Não recebe argumentos. Deverá escrever o produto com o maior número de unidades em stock usando o mesmo formato de saída do comando `l`:

  ```
  Chave Unidades
  ```

  Em caso de igualdade, deverá listar apenas o produto com maior número de unidades em stock e a *menor* chave lexicográfica (i.e., o primeiro produto segundo a ordem alfabética). O comando `m` não deverá ter output caso não haja elementos para listar.

- Comando `r`:
  `r <chave>`
  onde `<chave>` representa o código ou chave identificativa do produto a ser removido. Se a chave a ser removida não existir, os registos do programa deverão ficar inalterados. Este comando não envolve qualquer output.

- Comando `x`:
  Não tem argumentos. Escreve o número de chaves diferentes guardados no sistema antes de terminar o programa.

[1] Para a leitura do código e do valor, poderá utilizar simplesmente o `scanf`.
[2] Para uma breve introdução ao Valgrind, veja (por exemplo): http://www.cprogramming.com/debugging/valgrind.html

3 Exemplos

Nos exemplos seguintes, os comandos de *input* que devolvem algum *output* estão assinalados a *bold*, e com a cor correspondente ao *output* respectivo.

- **#exemplo1**

Input:

```
a 60fdba63 4
a c614e44d 149
a ff4095a1 38
a c614e44d 36
a 5d5c3b04 28
a 60fdba63 247
a 47cd69e4 223
a 376f7f4d 43
a 376f7f4d -14
l
x
```

Output:

```
376f7f4d 29
47cd69e4 223
5d5c3b04 28
60fdba63 251
c614e44d 185
ff4095a1 38
6
```

- **#exemplo2**

Input:

```
a 60fdba63 4
a c614e44d 149
a ff4095a1 38
a 60fdba63 -5
l
r c614e44d
a 90adba63 4
l
x
```

Output:

```
60fdba63 0
c614e44d 149
ff4095a1 38
60fdba63 0
90adba63 4
ff4095a1 38
3
```

- **#exemplo3**

Input:

```
a 60fdba63 24
a c614e44d 2
a ff4095a1 38
a c614e44d 36
a 5d5c3b04 28
a 47cd69e4 33
a 376f7f4d 43
a 376f7f4d -10
l
m
a 5d5c3b04 10
l
m
r 5d5c3b04
m
x
```

Output:

```
376f7f4d 33
47cd69e4 33
5d5c3b04 28
60fdba63 24
c614e44d 38
ff4095a1 38
c614e44d 38
376f7f4d 33
47cd69e4 33
5d5c3b04 38
60fdba63 24
c614e44d 38
ff4095a1 38
5d5c3b04 38
c614e44d 38
5
```

4 Compilação do Programa

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

```
$ gcc -Wall -o proj2 *.c
```

o qual deve ter como resultado a geração do ficheiro executável `proj2`, caso não haja erros de compilação. A execução deste comando não deverá escrever qualquer resultado no terminal. Caso a execução deste comando escreva algum resultado no terminal, considera-se que o programa não compilou com sucesso. Por exemplo, durante a compilação do programa, o compilador não deve escrever mensagens de aviso (*warnings*).

5 Execução do Programa

O programa deve ser executado da seguinte forma:

```
$ ./proj2 < test01.in > test01.myout
```

Posteriormente poderá comparar o seu output com o output previsto usando o comando **`diff`**

```
$ diff test01.out test01.myout
```

No *output* do comando `diff`, as linhas diferentes em ambos os ficheiros serão listadas, sendo as linhas esperadas precedidas por < e as linhas erradas no seu output precedidas por >.

Em alternativa, para testar o seu programa poderá usar os scripts **`run.sh`** e **`runall.sh`** distribuídos no ficheiro **`p2-exemplos.zip`**. Se quiserem executar apenas (por exemplo) o **`teste01.in`** deverão executar

```
$ ./run.sh teste01.in
```

Para executarem todos os testes deverão executar

```
$ ./runall.sh
```

Estes scripts compilam o código (usando o comando da secção 4) e comparam o resultado obtido com o resultado esperado. Se apenas indicar o tempo de execução é porque o comando **`diff`** não encontrou nenhuma diferença. Caso indique mais informação, então é porque o resultado obtido e o resultado esperado diferem. Para obter a informação detalhada das diferenças poderá remover a opção **`-q`** da linha 10 do ficheiro **`run.sh`**.

6 Entrega do Projecto

A entrega do projecto deverá respeitar o procedimento seguinte:

- Na página da disciplina aceda ao sistema para entrega de projectos. O sistema será activado uma semana antes da data limite de entrega. Instruções acerca da forma de acesso ao sistema serão oportunamente fornecidas.
- Efectue o upload de um ficheiro arquivo com extensão `.zip` que inclua os ficheiros fonte (`.c`) e cabeçalho (`.h`) que constituem o programa.
- Para criar um ficheiro arquivo com a extensão `.zip` deve executar o seguinte comando na directoria onde se encontram os ficheiros com extensão `.c` e `.h`, criados durante o desenvolvimento do projecto:
  ```
  $ zip proj2.zip *.c *.h
  ```
- Como resultado do processo de upload será informado se a resolução entregue apresenta a resposta esperada num conjunto de casos de teste.
- O sistema não permite submissões com menos de 10 minutos de intervalo para o mesmo grupo. Exemplos de casos de teste serão oportunamente fornecidos.
- Data limite de entrega do projecto: Sábado, **20 de Maio de 2017 (23h59m)**. Até à data limite poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efectuada. Deverá portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretende que seja avaliada. Não serão abertas excepções.

7 Avaliação do Projecto

7.1 Componentes da Avaliação

Na avaliação do projecto serão consideradas as seguintes componentes:

1. A primeira componente avalia o desempenho e 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 na discussão final do projecto. Nesta componente será também utilizado o sistema ***valgrind***[2] de forma a detectar fugas de memória ("memory leaks") ou outras incorrecções no código, que serão penalizadas. Aconselha-se por isso que os alunos utilizem este sistema para fazer debugging do código e corrigir eventuais incorrecções, antes da submissão do projecto. Para utilizar o valgrind, comece por compilar o seu código com o gcc mas com a *flag* adicional `-g` :

   ```
   $ gcc -g -Wall -o proj2 *.c
   ```

   Esta flag adiciona informação para *debugging* que será posteriormente utilizada pelo *valgrind*. Depois disso, basta escrever

   ```
   $ valgrind --tool=memcheck --leak-check=yes ./proj2 < teste01.in
   ```

[2] Para uma breve introdução ao Valgrind, veja (por exemplo): http://www.cprogramming.com/debugging/valgrind.html

Neste exemplo, estaria a usar o `teste01.in` como *input*. Deverá estar particularmente atento/a a mensagens como `Invalid read` e `Invalid write` que indicam está a tentar ler ou escrever fora da área de memória reservada por si. O *valgrind* também detecta a utilização de variáveis não inicializadas dentro expressões condicionais. Nesse caso receberá a mensagem `Conditional jump or move depends on uninitialised value(s)`. Por fim, o *valgrind* oferece a preciosa informação sobre a quantidade de memória alocada e libertada na *heap*, indicando quando essas duas quantidades não são iguais. Nesse caso terá *memory leaks*. Aqui fica um exemplo de output do *valgrind* quando tudo corre bem:

```
HEAP SUMMARY:
in use at exit: 0 bytes in 0 blocks
total heap usage: 1,024,038 allocs, 1,024,038 frees, 28,682,096 bytes allocated
All heap blocks were freed -- no leaks are possible
```

3. Durante a discussão final do projecto será averiguada a participação de cada elemento do grupo na realização do projecto, bem como a sua compreensão do trabalho realizado, sendo a respectiva classificação ponderada em conformidade, isto é, elementos do mesmo grupo podem ter classificações diferentes. Elementos do grupo que se verifique não terem participado na realização do respectivo projecto terão a classificação de 0 (zero) valores.

4. Avaliação dos projectos inclui a utilização de um sistema para detecção de situações de cópia entre projectos. A submissão de um projecto pressupõe o compromisso de honra que o trabalho incluso foi realizado única e exclusivamente pelos alunos referenciados nos ficheiros submetidos para avaliação. A quebra deste compromisso, tem como consequência a reprovação de todos os alunos envolvidos (incluindo quem possibilitar a ocorrência de cópias) à disciplina de IAED, assim como a comunicação da ocorrência ao respectivo Coordenador de curso e ao Conselho Pedagógico do IST.

7.2 Atribuição Automática da Classificação

- A classificação da primeira componente da avaliação do projecto é obtida automaticamente através da execução de um conjunto de testes executados 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. Projectos 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 podem 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, até um máximo de 64 Mb, e no tempo total disponível para execução, sendo o tempo limite distinto para cada teste.

- Note-se que o facto de um projecto passar com sucesso o conjunto de testes disponibilizado na página da disciplina não implica que esse projecto 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.

- Em caso algum será disponibilizada qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. A totalidade de ficheiros de teste usados na avaliação do projecto serão disponibilizados na página da disciplina após a data de entrega.