Enunciado do Projecto 2 - IAED 2020/21

Data de entrega: 24 de Maio de 2021, às 19h59m

LOG alterações

- 7mai21 - Publicação do enunciado.

1. Introdução

O objectivo deste projeto é o desenvolvimento, em linguagem C, de um sistema de armazenamento hierárquico semelhante a um sistema de ficheiros. Cada valor a armazenar, uma cadeia de carateres, é identificado por um caminho. A interacção com o programa deverá ocorrer através de um conjunto de linhas compostas por uma palavra (comando) e um número de argumentos dependente do comando a executar. Os possíveis comandos são listados na tabela seguinte e indicam as operações a executar.

Comando	Acção
help	Imprime os comandos disponíveis.
quit	Termina o programa.
set	Adiciona ou modifica o valor a armazenar.
print	Imprime todos os caminhos e valores.
find	Imprime o valor armazenado.
list	Lista todos os componentes de um caminho.
search	Procura o caminho dado um valor.
delete	Apaga um caminho e todos os subcaminhos.

2. Especificação do problema

O objetivo do projeto é ter um sistema de armazenamento hierárquico em memória. O sistema de armazenamento associa a um caminho um valor. Tanto o caminho como o valor são cadeias de carateres que não podem conter o caráter *NULL* (`'\0'`) e o caminho também não pode conter carateres brancos (`' '`, `'\t'`, `'\n'`). O caminho pode ser decomposto em componentes separados pelo caráter `'/'`. Um ou mais separadores `'/'` consecutivos correspondem a um só separador. Os primeiros componentes de um caminho formam um sub-caminho ou um prefixo de um caminho. Por exemplo, o caminho */usr/local/bin/lizard* tem 4 componentes e 3 sub-caminhos (*/usr*, */usr/local* e */usr/local/bin*). A existência de separador no início ou no fim do caminho não modifica os componentes, logo */usr/local//* é igual a *usr/local*.

Cada caminho pode ser associado a um valor. Por exemplo, o caminho */usr/local/bin/lizard* pode ser associado ao valor *analisador de complexidade*. Notar que não existe o conceito de diretoria pelo que é possível associar valores a */usr* ou */usr/local*. No entanto, os sub-caminhos de um caminho podem ser manipulados, como se tratasse de uma diretoria. Por exemplo, os caminhos */usr/local/bin/lizard* e */usr/local/bin/ncinfo* possuem prefixo comum, pelo que ao listar os caminhos com sub-caminho */usr/local/bin* deverá listar ambos os caminhos.

Não existem limites no número nem na dimensão dos caminhos ou dos valores, logo deve procurar utilizar a memória estritamente necessária. Para facilitar a introdução dos dados, pode assumir que cada instrução não excede 65535 carateres. Se a memória se esgotar, o programa deve terminar de forma controlada, imprimindo a mensagem `No memory.` Antes de terminar, o programa deve libertar toda a memória reservada.

3. Dados de Entrada

Durante a execução do programa as instruções devem ser lidas do standard input na forma de um conjunto de linhas iniciadas por uma palavra, que se passa a designar por *comando*, seguido de um número de informações dependente do comando a executar. Os comandos e os argumentos são separados por espaços ou tabuladores. No entanto, o último argumento pode conter espaços ou tabuladores se for um `<valor>`, sendo que um `<valor>` não tem espaços ou tabuladores no início ou no fim.

Os comandos disponíveis são descritos de seguida. Cada comando indica uma determinada ação que se passa a caracterizar em termos de formato de entrada, formato de saída, e erros. No caso de múltiplos erros para o mesmo comando deverá retornar apenas o primeiro desses erros.

- **help** - Imprime os comandos disponíveis:
    - Formato de entrada: `help`
    - Formato de saída: Imprime a lista de comandos disponíveis, um por linha, com `<comando>: <descrição>` pela ordem e texto apresentados na tabela acima.
    - Erros: Não aplicável.
- **quit** - Termina o programa:
    - Formato de entrada: `quit`
    - Formato de saída: NADA
    - Erros: Não aplicável.
- **set** - Adiciona ou modifica o valor a armazenar:
    - Formato de entrada: `set <caminho> <valor>`
    - Formato de saída: NADA
    - Erros: Não aplicável.
- **print** - Imprime todos os caminhos e valores:
    - Formato de entrada: `print`
    - Formato de saída: Imprime todos os caminhos e valores (um caminho e valor por linha), em profundidade, pela ordem de criação dos componentes. Apenas os caminhos com valor associado devem ser impressos. Os caminhos deve ser iniciados pelo separador '/'e separados do valor por um espaço.
    - Erros: Não aplicável.
- **find** - Imprime o valor armazenado de um caminho:
    - Formato de entrada: `find <caminho>`
    - Formato de saída: Imprime o valor associado ao `<caminho>`.
    - Erros:
        - `not found` no caso de não existir o caminho.
        - `no data` no caso de o caminho não ter valor associado.
- **list** - Lista todos os componentes imediatos de um sub-caminho:
    - Formato de entrada: `list <caminho>`
    - Formato de saída: Imprime todos os componentes imediatos do `<caminho>` por ordem alfabética (ordem *ASCII*, maiúsculas primeiro), ou seja o seu diretório. Se o comando for invocado sem argumentos, lista os componentes da raiz.
    - Erros:

        - `not found` no caso de não existir o caminho.

- **search** - Procura o caminho dado um valor:
    - Formato de entrada: `search <valor>`
    - Formato de saída: Imprime o primeiro caminho encontrado que contém exatamente o `<valor>` indicado. O caminho inicia-se com o separador '/' e tem apenas um separador '/' entre cada componente. Cada componente deve ser pesquisado pela ordem de criação.
    - Erros:
        - `not found` no caso de não existir nenhum caminho com o valor indicado.

- **delete** - Apaga todos os caminhos de um sub-caminho:
    - Formato de entrada: `delete <caminho>`
    - Formato de saída: Apaga o `<caminho>` indicado e todos os outros caminhos para o qual `<caminho>` é um sub-caminho. Se for invocado sem argumentos apaga todos os caminhos armazenados.
    - Erros:
        - `not found` no caso de não existir o caminho.

4. Dados de Saída

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 o número de espaços entre elementos do seu output, assim como os espaços no final de cada linha. Procure respeitar escrupulosamente as indicações dadas.

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

```
$ gcc -Wall -Wextra -Werror -ansi -pedantic -o proj2 *.c
```

Este comando 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 deverá escrever mensagens de aviso (warnings).

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

5. Execução do Programa

O programa deve ser executado da forma seguinte:

```
$ ./proj2 < 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 `tests/`. na pasta `tests/`. Para executar todos os testes com o *valgrind* poderá executar `make valgrind` na pasta `tests/`.

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 todos os ficheiros fonte que constituem o programa.
- Se o seu código tiver apenas um ficheiro o zip conterá apenas esse ficheiro.
- Se o seu código estiver estruturado em vários ficheiros (`.c` e `.h`) não se esqueça de os juntar também ao pacote.
- 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` (se for o caso), 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 aluno. Tenha especial atenção a este facto na altura da submissão final.
- Data limite de entrega do projecto: **24 de Maio de 2021, às 19h59m**. Até à data limite poderá efectuar o número de submissões que desejar, sendo utilizada para efeitos de avaliação a última submissão efectuada. Deverá portanto verificar cuidadosamente que a última submissão corresponde à versão do projecto que pretende que seja avaliada. Não existirão excepções a esta regra.

7. Avaliação do Projecto

Na avaliação do projecto serão consideradas as seguintes componentes:

1. A primeira componente será feita automaticamente e 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, alocação dinâmica de memória, estruturação, modularidade e divisão em ficheiros, abstracção de dados, 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 aqui.
3. Na segunda componente serão utilizadas as ferramentas *lizard*, *valgrind*, e a opção *fsanitize* por forma a detectar a complexidade de código, fugas de memória (“memory leaks”) ou outras incorrecções no código, que serão penalizadas. Aconselha-se que

utilizem estas ferramentas para fazer debugging do código e corrigir eventuais incorrecções, antes da submissão do projecto. Algumas dicas para degugging podem ser encontradas aqui.

4. A classificação da primeira componente da avaliação do projecto 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 descritos 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 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.

5. 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.