Etapa 3 de Laboratórios de Informática II

Licenciatura de Engenharia Informática

Ano Lectivo 09/10

Versão 1.1

Conteúdo

1 Critérios de Aprendizagem

No final desta etapa os alunos deverão ser capazes de:

1. Efectuar gestão de memória, incluíndo alocação e libertação;
2. Utilizar listas ligadas;
3. Utilizar um depurador de código (*debugger*) para corrigir os eventuais erros que possam surgir.

2 Enunciado

Implementar os seguintes comandos **sem qualquer tipo de variação na sintaxe**:

`ab` Liga ou desliga o automático branco. Automáticamente pinta de preto todas as células na mesma linha e coluna que uma célula que acabou de ser pintada de branco;

`ap`	Liga ou desliga o automático preto. Automáticamente pinta de branco as quatro vizinhas ortogonais de uma célula que acabou de ser pintada de preto;
`anc`	Anular cor. Anula a última atribuição de cor efectuada;
`anm`	Anula o último movimento efectuado;
`it`	Inicia tentativa. Cria uma marca que serve para saber o estado para onde passar utilizando o comando `rb`;
`rb`	Roll back. Volta ao estado do tabuleiro aquando da invocação do último comando `it`;
`st`	O estado do tabuleiro. Imprime a mensagem `E_WRONG_SOLUTION` no `STDERR` (usando `mensagem_de_erro`) caso o estado do tabuleiro seja inválido.

3 Automáticos

Os comandos `ab` e `ap` activam atribuições automáticas de cor. Isso quer dizer que ao invocar *qualquer* comando que mude a cor de uma célula (e.g. `p`, `b`, `trp`, `snd`, etc) se o automático correspondente estiver ligado então as células que esse automático indicar são pintadas. Caso o outro automático estiver ligado, o acto de pintar as células da cor contrária activa esse automático pela sua vez e assim sucessivamente.

Imaginemos que ambos os automáticos estão activados. Quando um comando pintar uma célula de branco então as outras células com a mesma letra existentes na mesma linha e coluna são pintadas de preto. Seguidamente, as células vizinhas de cada célula que foi pintada de preto serão pintadas de branco. Para cada uma das células pintadas de branco, procuram-se as letras iguais que se pintam de preto e assim sucessivamente.

4 Anular

Os comandos de anular repõe o estado antes de uma operação (e.g., mudança de cor, movimento). Se se aplicar os comandos várias vezes vamos recriando estados anteriores do tabuleiro até chegar ao estado inicial. Se um dos comandos for aplicado quando o tabuleiro está no seu estado inicial (i.e., se ainda não ocorreu um movimento do género esperado) os comandos deverão emitir a mensagem do erro `E_NO_MOVES`.

O comando `anc` anula uma atribuição de cor. Repare que um comando pode provocar várias mudanças de cor (e.g. `trp`). Já o comand `anm` anula um movimento. Repare que só interessam os movimentos que implicam mudanças de cor. Portanto, não interessam comandos como por exemplo `gr` que não mudam a cor a células. Considera-se que o comando `cr` é especial: não há anulações depois de carregar um tabuleiro.

5 Avaliação

Esta etapa vale no máximo 4 valores e deve ser entregue até ao fim do dia 2 de Maio no site da disciplina. Para não reprovar à disciplina os alunos precisam de tirar um mínimo de 50% da classificação desta etapa (2 valores). A entrega atrasada incorre numa penalização de 10% ao dia.

A avaliação é feita durante as aulas da semana seguinte. Caso um aluno não apareça à avaliação sem apresentar motivos de força maior devidamente justificados por documentos este reprova automaticamente à disciplina por faltar à avaliação.

5.1 Material a entregar para cada uma das etapas

O grupo deverá entregar em cada etapa um arquivo criado com o comando `tar` e comprimido com o algoritmo de compactação `bzip2` contendo a seguinte informação:

code Uma pasta com todo o código fonte do trabalho que deverá incluir a `makefile` utilizada para compilar todo o código e gerar a documentação;

doc Uma pasta com a documentação em formato `html` gerada a partir do código utilizando a ferramenta `Doxygen`;

relatorio.pdf Um ficheiro gerado utilizando o comando LaTeX no formato `pdf` com o relatório;

doc.pdf Um ficheiro gerado utilizando o comando LaTeX a partir dos ficheiros gerados pela ferramenta `Doxygen`.

O arquivo deverá utilizar o seguinte arquétipo para o nome:
`g<número do grupo><turno>-et<número da etapa>.tar.bz2`
Por exemplo, o ficheiro entregue na terceira etapa pelo grupo número 3 do turno PL1 teria como nome `g3PL1-et3.tar.bz2`.

5.2 Relatório

Cada etapa deve vir acompanhada de um relatório escrito em LaTeX e do código documentado. O relatório deve necessariamente ter as seguintes secções:

Resumo Onde se apresenta um breve resumo do relatório que não deverá ultrapassar as 250 palavras;

Introdução Onde se apresenta a introdução do relatório, tipicamente, deverá ter subsecções como motivação, objectivos e estrutura do relatório;

Desenvolvimento O desenvolvimento poderá ser mais do que uma secção, no caso deste relatório deverá explicar as opções tomadas e apresentar breves algoritmos que ajudem a compreender as partes críticas do código;

Conclusão Onde se apresentam as conclusões objectivas do trabalho efectuado;

Bibliografia Onde se citam as referências bibliográficas utilizadas no trabalho.

Lembre-se que o relatório pretende explicar quais foram as dificuldades encontradas e como estas foram resolvidas. A apresentação deverá ser técnica de forma a ajudar o avaliador a perceber o que foi feito e como foi feito.

É obrigatório apresentar a documentação gerada automáticamente através do código comentado utilizando o `Doxygen` `http://www.stack.nl/~dimitri/doxygen`. Todas as funções deverão ser documentadas com uma pequena descrição que ilustre o seu funcionamento. Assim, não caia na tentação de descrever as várias funções no relatório já que o resultado do `Doxygen` produzirá um documento com essa informação. Deverá configurar o `Doxygen` para gerar documentação para todas as entidades para garantir que não se esquece de nenhuma.

5.3 Critérios

Esta etapa será avaliada de acordo com os seguintes critérios:

1. Requisitos básicos (2 pontos dos 4 possíveis):
   (a) Deve existir um ficheiro chamado `makefile` que possibilite a utilização do comando `make` para compilar todo o projecto e gerar um executável chamado `letrorium`;
   (b) A `makefile` deve compilar os ficheiros com as opções `-Wall -Wextra -pedantic -ansi -O2`;
   (c) A compilação daí resultante não pode ter erros;
   (d) Os comandos devem funcionar todos correctamente em situações normais (correndo os testes que estão em `htto://omega.di.uminho.pt/cgi/verifica.pl`).
   (e) O relatório deve ser claro e documentar todas as opções de codificação tanto de estruturas de dados como da implementação das funções (não devendo repetir informação que está na documentação das funções).
   (f) O código deve estar correctamente comentado e documentado (i.e., a documentação gerada pelo `Doxygen` deve ser completa, fidedigna e clara);

2. Requisitos médios (2 pontos dos 4 possíveis):

   (a) O programa compilado com `-Wall -Wextra -pedantic -ansi -O2` não pode ter warnings;

   (b) As estruturas de dados deve ser a mais eficiente possível tanto em termos de armazenamento como de facilidade de acesso;

   (c) Deve passar nos testes de esforço (na mesma script);

   (d) A implementação dos comandos deve ser eficiente;

   (e) O código deve estar bem estruturado;

   (f) As funções devem ser curtas[1] e usar funções auxiliares que implementem tarefas comuns;

   (g) As linhas devem ser fácilmente compreensíveis: não deve ser necessário perder mais do que meio segundo por linha para perceber o que ela faz;

   (h) Todas as funções devem ter nomes que ajudem a perceber o que elas fazem;

   (i) Todas as variáveis devem ter nomes que ajudem a perceber a sua tarefa;

   (j) Não devem existir variáveis globais.

[1] As linhas não devem ultrapassar os 60 caracteres e a função não deve ter mais do que 20 linhas