Aula Prática 7 - Roteiro

14/09/2021 - Roteiro referente à aula prática 7 - Algoritmo de

Preenchimento Versão: 14/09/2021

Prazo: 20/09/2021 - 18:00

Valor: 10,0 - Peso: 2

Leia este enunciado com **MUITA** atenção até o final antes de iniciar o trabalho. Os arquivos solicitados deverão estar disponíveis nos diretórios correspondentes (**Aulas Praticas** e ***RCS***) até o prazo estipulado acima. Cuidado com os nomes dos diretórios e dos arquivos. Deverão ser exatamente os definidos neste roteiro (maiúsculas, minúsculas, caracteres especiais e extensões, se existentes).
As tarefas deverão ser executadas na ordem solicitada neste roteiro. Os arquivos de dependências deverão possibilitar que a compilação e a linkedição sejam executadas utilizando-se tanto o *gcc*, quanto o *clang*. A seleção da ferramenta utilizada deverá ser realizada no momento da execução do comando *make*. O gcc deverá ser considerado o valor padrão para a compilação e para a *linkedição*.

Para a definição da ferramenta desejada deverá ser definida uma macro (no *FreeBSD*) ou um argumento com o valor desejado (no *CentOS*). As duas macros deverão ser *GCC* e *CLANG* (definidas usando a opção -D). O argumento, identificado por *cc,* deverá ser igual a *GCC* ou *CLANG*.
Independente da ferramenta utilizada para a compilação, o *flag* de compilação deverá ser definido no instante da execução do comando *make*. O valor padrão para este *flag* deverá ser "-Wall -ansi" (sem as aspas).

Durante a execução do comando *make* poderão ser definidos outros valores para este *flag* (mantendo a opção de exibir todas as mensagens de advertência) através de macros ou através de argumentos (de forma semelhante àquela utilizada para definir o compilador/linkeditor). No FreeBSD deverão ser definidas as macros ANSI, C99 e C11, enquanto que no *CentOS* deverá ser definido o argumento *dialeto* com os valores ANSI, C99 ou C11.
Crie uma macro, DIALETO, contendo o dialeto a ser utilizado na compilação do código. Esta macro será inicialmente igual a "ansi" e poderá ser alterada para "c99" ou "c11" de acordo com o esquema definido acima.
O *flag* de linkedição deverá ser igual a "-Wall" (sem as aspas).
Seguem alguns exemplos:

make - compila/linkedita (tanto no FreeBSD, quanto no CentOS) com a ferramenta e dialeto padrões, ou seja, gcc e ANSI respectivamente.
make -DGCC - compila/linkedita usando o *gcc* e o dialeto ANSI (somente FreeBSD). make -DCLANG - compila/linkedita usando o *clang* e o dialeto ANSI (somente FreeBSD). make cc=GCC - compila/linkedita usando o *gcc* e o dialeto ANSI (somente CentOS). make cc=CLANG - compila/linkedita usando o *clang* e o dialeto ANSI (somente CentOS). make -DCLANG -DC11 - compila/linkedita usando o *clang* e o dialeto C11 (somente FreeBSD). make cc=CLANG dialeto=C99 - compila/linkedita usando o *clang* e o dialeto C99 (somente CentOS).
Inclua, no início de todos os arquivos solicitados, os seguintes comentários:

```
Universidade Federal do Rio de Janeiro
Escola Politecnica
Departamento de Eletronica e de Computacao
EEL270 - Computacao II - Turma 2020/2
```

```
Prof. Marcelo Luiz Drumond Lanza
Autor: <nome completo>
Descricao: <descrição sucinta dos objetivos do programa>

$Author$
$Date$
$Log$
```
Inclua, no final de todos os arquivos solicitados, os seguintes comentários:

```
$RCSfile$
```

1. Um dispositivo de saída (monitor) pode ser visto como uma matriz bidimensional de pixels. Para simplificar, cada pixel que estiver funcionando corretamente poderá estar apagado ou aceso. Considerando-se um terminal do tipo texto, um pixel será representado por um caractere, ou seja, o caractere '-' (hífen) se estiver apagado ou o caractere '+' (mais) se estiver aceso. O caractere 'X' (consoante maiúscula) deverá ser utilizado para representar os pixels que estiverem com defeito.

   Um algoritmo de preenchimento de área utilizado neste ambiente recebe as informações atualizadas sobre todos os pixels do monitor através de uma matriz bidirecional. Estas informações devem incluir a definição de um polígono, ou seja, os pixels correspondentes aos lados do polígono deverão estar acesos. Além disso, o algoritmo deverá receber a quantidade de linhas e de colunas do monitor e as coordenadas de um pixel que poderá estar estar localizado dentro ou fora do polígono (não pode pertencer aos lados do polígono).

   A partir destas informações, toda a área interna (se o ponto inicial estiver localizado dentro do polígono) ou externa (se o ponto inicial estiver localizado fora do polígono) ao polígono deverá ser preenchida (os pixels deverão ser redefinidos de forma que sejam representados pelo caractere '+').

   O primeiro passo no algoritmo é verificar se o pixel desejado já está aceso. Se o mesmo estiver aceso nenhuma ação deverá ser executada.

   Caso contrário, o pixel em questão deverá ser aceso e o algoritmo deverá ser executado novamente para os quatro pixels vizinhos a este pixel (esquerda, direita, abaixo e acima do pixel em questão - nesta ordem).

   Nas questões a seguir, as macros *NUMERO_MAXIMO_LINHAS* e *NUMERO_MAXIMO_COLUNAS* representam os maiores valores permitidos pelo algoritmo. Nesta atividade, estas macros deverão ser iguais a 250 e 800. Um monitor de teste poderia ter por exemplo, 50 linhas e 100 colunas, mas não poderia ter 300 linhas e 400 colunas, já que o número máximo de linhas definido é 250.

   Baseado no algoritmo acima, crie o arquivo "*aula0701.h*" contendo a definição das macros *APAGADO* ('-'), *ACESO* ('+'), *DEFEITUOSO* ('X'), *NUMERO_MAXIMO_LINHAS* (250) e NUMERO_MAXIMO_COLUNAS (800), do tipo *tipoErros* (tipo enumerado que deverá conter os códigos de retorno das funções solicitadas nos próximos itens, por exemplo, *ok* - valendo 0, *linhaInvalida* - valendo 1, etc.), do tipo *tipoPixel* (enumerado contendo os valores apagado - valendo 0, aceso - valendo 1 e defeituoso - valendo 2) e do protótipo da função *MostrarMonitor*.

   Esta função deverá receber uma matriz de pixels (*tipoPixel*) correspondendo ao monitor (em um dado momento), suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*) e o tempo de congelamento da tela (em *microsegundos*). A função deverá exibir o conteúdo desta matriz (caracteres '-' - macro *APAGADO*, '+' - macro *ACESO* e 'X' - macro *DEFEITUOSO*).

A macro referente à combinação *ifndef* e *define*, como por exemplo *_AULA0701_*, deverá ser definida como uma *string* valendo:

"@(#)aula0701.h $Revision$"
tipoErros
MostrarMonitor (tipoPixel monitor [NUMERO_MAXIMO_LINHAS] [NUMERO_MAXIMO_COLUNAS], unsigned numeroMaximoLinhas, unsigned numeroMaximoColunas, useconds_t tempoEspera);

Observações:

A) Para esta função e para as demais funções solicitadas as linhas e colunas começam em 1. Lembre-se que para a linguagem o primeiro índice de cada dimensão vale 0. Além disso, considere que o ponto 1,1 corresponde ao canto superior esquerdo do monitor (elemento 0,0 da matriz).

B) O uso do operador ternário na chamada da função *printf* que exibe cada pixel (cada caractere correspondente) é obrigatório.

Operador Ternário:
Condição ? verdadeiro : falso

2. Crie o arquivo "*aula0701.c*" contendo o código fonte da função *MostrarMonitor*. Esta função deverá conter, antes da exibição do conteúdo, uma chamada para a função *system* executando o comando *clear*. Após a exibição do conteúdo, esta função deverá conter uma chamada para a função *usleep* (com o tempo recebido). A função deverá retornar *ok* (ZERO) ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função).
3. Inclua, nos arquivos de dependências, as macros *LIBMONITOROBJS* (correspondendo ao arquivo "*aula0701.o*") e *LIBMONITOR (*correspondendo ao arquivo "*libmonitor.a*"). O valor da macro LIBS deverá ser atualizado de forma que inclua o valor desta última macro. Inclua o objetivo correspondente, ou seja, *libmonitor.a*, com a(s) dependência(s) e comando(s) necessários para atingir este objetivo.
4. Crie o arquivo "aula0702.c" contendo o o programa de testes para a função *MostrarMonitor*. O programa deverá receber, através dos argumentos de linha de comando, as dimensões reais do monitor (número de linhas e número de colunas) e o tempo de congelamento da exibição (em us). Além disso, deverá preencher a matriz correspondente com os caracteres '-', '+' e 'X' possibilitando os seguintes testes (através de 4 execuções da função *MostrarMonitor*):

   a) todos os pixels apagados.
   b) todos os pixels acesos.
   c) todos os pixels com defeito.
   d) distribuição aleatória das três possibilidades.

   ./aula0702 100 200 5
5. Inclua, nos arquivos de dependências, as macros *AULA0702OBJS* e *AULA07*. Altere o valor da macro *EXECS,* de forma que inclua o valor da macro *AULA07*. Inclua também os objetivos *aula07* e *aula0702* com os comandos correspondentes.
6. Gere e teste as 16 versões do executável *aula0702* usando, na linkedição, a biblioteca "libmonitor.a".
7. Submeta os arquivos *"aula0701.h"*, "*aula0701.c"*, "*aula0702.c"* e "**makefile"* ao sistema de controle de versão.
8. Recupere uma cópia de leitura do arquivo "*aula0702.c"* e uma cópia de escrita dos arquivos *"aula0701.h*", "*aula0701.c*" e "**makefile"*.
9. Inclua, no arquivo "*aula0701.h*", a definição do protótipo da função *LimparMonitor*. Esta função deverá receber uma matriz de pixels (*tipoPixel*) correspondendo ao monitor e suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*). Se todos os argumentos

forem válidos, a função deverá "apagar" todos os pixels do dispositivo (na matriz correspondente). Não deverão ser apagados os pixels marcados como defeituosos.

tipoErros
LimparMonitor (tipoPixel monitor [NUMERO_MAXIMO_LINHAS][NUMERO_MAXIMO_COLUNAS], unsigned numeroMaximoLinhas, unsigned numeroMaximoColunas);

10. Inclua, no arquivo "*aula0701.c*", a implementação da função *LimparMonitor*.
11. Crie o arquivo "aula0703.c" contendo o o programa de testes para a função *LimparMonitor*. O programa deverá receber, através dos argumentos de linha de comando, as dimensões reais do monitor (número de linhas e número de colunas) e o tempo de congelamento da exibição. Além disso, deverá preencher, de forma aleatória, a matriz correspondente com os caracteres '-', '+' e 'X'. A seguir deverá executar as funções *MostrarMonitor*, *LimparMonitor* e *MostrarMonitor* novamente.

    ./aula0703 100 200 5

12. Inclua as declarações necessárias nos arquivos de dependências.
13. Gere e teste as 16 versões do executável *aula0703* usando, na linkedição, a biblioteca "*libmonitor.a*".
14. Submeta os arquivos "*aula0701.h*", "*aula0701.c*", "*aula0703.c*" e "**makefile*" ao sistema de controle de versão.
15. Recupere uma cópia de leitura do arquivo "aula0703.c" e uma cópia de escrita dos arquivos "*aula0701.h*", "*aula0701.c*" e "*makefile".
16. Inclua, no arquivo "*aula0701.h*", a definição do protótipo da função *DesenharReta*. Esta função deverá receber uma matriz de pixels (*tipoPixel*) correspondendo ao monitor, suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*) e os valores (linha e coluna) correspondentes a dois pontos (as duas extremidades da reta desejada). Se todos os argumentos forem válidos, a função deverá "acender" todos os *pixels* necessários para desenhar a reta desejada. Caso um pixel necessário para desenhar a reta esteja defeituoso, a função deverá retornar o código de erro correspondente. Antes de desenha a reta (marcar os pixels correspondentes à reta com '+', verifique todos os pontos necessários). As retas não serão necessariamente horizontais ou verticais e dada a simplicidade do dispositivo, poderão apresentar pequenas imperfeições.

    tipoErros
    DesenharReta (tipoPixel monitor [NUMERO_MAXIMO_LINHAS][NUMERO_MAXIMO_COLUNAS], unsigned numeroMaximoLinhas, unsigned numeroMaximoColunas, unsigned linhaA, unsigned colunaA, unsigned linhaB, unsigned colunaB);

17. Inclua, no arquivo "*aula0701.c*", a implementação da função *DesenharReta*. 18. Crie o arquivo "aula0704.c" contendo o o programa de testes para a função *DesenharReta*. O programa deverá receber, através dos argumentos de linha de comando, as dimensões reais do monitor (número de linhas e número de colunas), os valores correspondentes aos pontos A e B (extremidades da reta) e o tempo de congelamento da exibição. Este programa deverá executar as seguintes funções: *MostrarMonitor, LimparMonitor, DesenharReta* e *MostrarMonitor*.

    ./aula0704 100 200 5 5 70 30 5

19. Inclua as declarações necessárias nos arquivos de dependências.
20. Gere e teste as 16 versões do executável *aula0704* usando, na linkedição, a biblioteca "*libmonitor.a*".
21. Submeta os arquivos "*aula0701.h*", "*aula0701.c*", "*aula0704.c*" e "**makefile*" ao sistema de controle de versão.
22. Recupere uma cópia de leitura do arquivo "aula0704.c" e uma cópia de escrita dos arquivos "*aula0701.h*", "*aula0701.c*" e "**makefile*".

23. Inclua, no arquivo "*aula0701.h*", a definição do protótipo da função *DesenharPoligono*. Esta função deverá receber uma matriz de pixels (*tipoPixel*) correspondendo ao monitor, suas dimensões reais (numeroMaximoLinhas e numeroMaximoColunas), o número de vértices do polígono desejado, as informações (linha, coluna) sobre estes vértices e o tempo de congelamento da tela (em microsegundos). Se todos os argumentos recebidos forem válidos, a função deverá "acender" todos os pixels que correspondam aos lados do polígono em questão (usando a função *DesenharReta*). A função deverá retornar *ok* ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função). Se um pixel defeituoso for detectado durante o procedimento, a execução da função deverá ser interrompida, retornando o código de erro correspondente.

    tipoErros
    DesenharPoligono (tipoPixel monitor [NUMERO_MAXIMO_LINHAS][NUMERO_MAXIMO_COLUNAS], unsigned numeroMaximoLinhas, unsigned numeroMaximoColunas, unsigned numeroVertices, unsigned linhasVertices [NUMERO_MAXIMO_LINHAS], unsigned colunasVertices [NUMERO_MAXIMO_COLUNAS]);

24. Inclua, no arquivo de "*aula0701.c*", a implementação da função *DesenharPoligono*. 25. Crie o arquivo "*aula0705.c*" contendo o o programa de testes para a função *DesenharPoligono*. O programa deverá receber, através dos argumentos de linha de comando, as dimensões reais do monitor (número de linhas e número de colunas), o tempo de congelamento da exibição, o número de vértices do polígono e os valores correspondentes aos vértices deste polígono. Este programa de testes deverá executar as seguintes funções: *MostrarMonitor*, *DesenharPoligono* e *MostrarMonitor*.

    ./aula0705 100 200 5 3 5 70 30 105 30 35
26. Inclua as declarações necessárias nos arquivos de dependências.
27. Gere e teste as 16 versões do executável aula0705 usando, na linkedição, a biblioteca "*libmonitor.a*".
28. Submeta os arquivos "*aula0701.h*", "*aula0701.c*", "*aula0705.c*" e "**makefile*" ao sistema de controle de versão.
29. Recupere uma cópia de leitura do arquivo "*aula0705*.c" e uma cópia de escrita dos arquivos "*aula0701.h*", "*aula0701.c*" e "**makefile*".
30. Inclua, no arquivo "*aula0701.h*", a definição do protótipo da função *PreencherPoligono*. Esta função deverá receber uma matriz de pixels (*tipoPixel*) correspondendo ao monitor (incluindo a definição de um polígono - pixels correspondentes aos lados do polígono deverão estar acesos), suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*), as informações sobre um pixel que esteja localizado dentro ou fora do polígono (linha e coluna) e o tempo de congelamento da tela (em *microsegundos*). Se todos os argumentos forem válidos, a área interna ou externa ao polígono deverá ser preenchido de acordo com o algoritmo definido acima. Se, durante o procedimento de preenchimento, um pixel defeituoso foi detectado, a execução da função deverá ser interrompida , retornando o código de erro correspondente. A cada pixel alterado com sucesso, a função *MostrarMonitor* deverá ser executada.

    tipoErros
    PreencherPoligono (tipoPixel monitor [NUMERO_MAXIMO_LINHAS][NUMERO_MAXIMO_COLUNAS], unsigned numeroMaximoLinhas, unsigned numeroMaximoColunas, unsigned linha, unsigned coluna, useconds_t tempoEspera);

31. Inclua, no arquivo de "*aula0701.c*", a implementação da função *PreencherPoligono*.

32. Crie o arquivo "*aula0706.c*" contendo a implementação do programa de testes para a função *PreencherPoligono*. Este programa deverá receber o número de linhas e o número de colunas do dispositivo desejado, o número de vértices do polígono desejado, as informações (linha, coluna) sobre estes vértices, as informações (linha, coluna) sobre um ponto qualquer do monitor (interno ou

externo ao polígono) e o tempo de congelamento da tela (em microsegundos). Todos estes argumentos deverão ser recebidos via argumentos da CLI (argv's) e na ordem definida anteriormente. Se todos os argumentos recebidos forem válidos, o programa deverá preencher a matriz correspondente ao monitor de forma aleatória e então deverá executar as funções *LimparMonitor*, *DesenharPoligono, MostrarMonitor* e *PreencherPoligono*. 33. Inclua as declarações necessárias nos arquivos de dependências.

34. Gere e teste as 16 versões do executável aula0706 usando, na linkedição, a biblioteca "*libmonitor.a*".
35. Submeta os arquivos "*aula0701.h*", "*aula0701.c*", "*aula0706.c*" e "**makefile*" ao sistema de controle de versão.
36. Recupere uma cópia de leitura dos arquivos "*aula0701.h*", "*aula0701.c*" e"*aula0706.c*" e uma cópia de escrita dos arquivos "**makefile*".
37. Limpe o diretório (*make clean-all*).