

Aula Prática 7 – Roteiro

Objetivos:

- Preenchimento de Áreas

Versão Inicial: 20/06/2023

Prazo: 27/06/2023 – 8:00

Observações:

- 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 (incluindo 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 que 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 como o valor padrão para a ferramenta de compilação e de *linkedição*.

Para a definição da ferramenta desejada, deverá ser utilizada uma macro no *FreeBSD* e um argumento com o valor desejado no *Linux*. As duas macros utilizadas deverão ser **GCC** e **CLANG** (definidas usando a opção de linha de comando **-D** do comando *make*). O argumento, identificado por *cc*, deverá ser igual a **GCC** ou a **CLANG**.

- Independente da ferramenta utilizada para a compilação, as opções de compilação poderão ser redefinidas no instante da execução do comando *make* (mantendo-se a exibição de todas as mensagens de advertência, definida pelo valor **-Wall**). O valor padrão para estas opções deverá ser **-Wall -ansi**.

Estas opções poderão ser redefinidas 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**, **C89**, **C90**, **C99** e **C11**, enquanto que no *Linux* deverá ser definido o argumento **dialeto** com um dos seguintes valores **ANSI**, **C89**, **C90**, **C99** ou **C11**.

- Os arquivos de dependências deverão incluir a macro **DIALECT** contendo o dialeto a ser utilizado na compilação do código. Esta macro será inicialmente igual a **ansi** e poderá ser alterada para **c89**, **c90**, **c99** ou **c11** de acordo com o esquema definido acima.
- Os arquivos de dependências deverão incluir também a macro **STANDARD** contendo a opção de linha de comando correspondente ao dialeto selecionado. Se, por exemplo, o dialeto selecionado for o **ANSI**, esta macro deverá ser igual a **-ansi**. Por outro lado, se o dialeto for uma das outras quatro opções, esta macro deverá ser igual a **-std=CXX**, onde **XX** deverá ser substituído pelo número correspondente (se o dialeto for igual a **C89**, **XX** deverá ser igual a **89**, se o dialeto for igual a **C90**, **XX** deverá ser igual a **90** e assim por diante).
- A *linkedição* deverá utilizar a opção **-Wall**.
- Cuidado com os nomes das macros e dos rótulos. Deverão ser exatamente os definidos neste roteiro (maiúsculas, minúsculas, caracteres especiais e extensões, se existentes).
- Todos os rótulos solicitados no roteiro são obrigatórios. Durante a correção, caso não seja possível alcançar os objetivos (binários e/ou bibliotecas e limpezas de código) solicitados, a nota correspondente ao item/aula em questão será igual a zero.
- Seguem alguns exemplos (todos devem funcionar):

- **make** - compila/*linkedita* (tanto no *FreeBSD*, quanto no *Linux*) com a ferramenta e dialeto padrões, ou seja, **gcc** e **ANSI** respectivamente.
 - **make clean-all all**
 - **make clean-all aula01**
 - **make clean aula0101**
 - **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 *Linux*).
 - **make cc=CLANG** - compila/*linkedita* usando o **clang** e o dialeto **ANSI** (somente *Linux*).
 - **make -DCLANG -DC89** - compila/*linkedita* usando o **clang** e o dialeto **C89** (somente *FreeBSD*).
 - **make -DCLANG -DC11** - compila/*linkedita* usando o **clang** e o dialeto **C11** (somente *FreeBSD*).
 - **make cc=CLANG dialero=C99** - compila/*linkedita* usando o **clang** e o dialeto **C99** (somente *Linux*).
 - **make cc=GCC dialeto=C90** - compila/*linkedita* usando o **gcc** e o dialeto **ANSI** (somente *Linux*).
- Inclua, no início de todos os arquivos solicitados (código-fonte e arquivos de dependências), os seguintes comentários (**sem caracteres especiais**):
Universidade Federal do Rio de Janeiro
Escola Politecnica
Departamento de Eletronica e de Computacao
EEL270 - Computacao II - Turma 2023/1
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, o seguinte comentário:
\$RCSfile\$

Um dispositivo de saída (monitor) pode ser visto como uma matriz bidimensional de pontos denominados *pixels*. Para simplificar, cada *pixel* que estiver funcionando corretamente poderá estar apagado ou aceso. Considerando-se um terminal do tipo texto, um *pixel* funcional (apagado ou aceso) será representado pelo caractere ' ' (espaço) com a cor definida pelo usuário para cada uma destas situações. Por outro lado, um *pixel* defeituoso será representando pelo caractere '.' (ponto) usando a cor definida para os *pixels* acesos.

Um algoritmo de preenchimento de área utilizado neste ambiente utiliza as informações atualizadas sobre todos os *pixels* do monitor através de uma matriz bidirecional. Estas informações podem incluir a definição de um polígono. Neste caso, os *pixels* correspondentes aos lados do polígono estarão acesos.

O algoritmo deverá utilizar também as quantidades reais de linhas e de colunas do monitor (que deverão ser menores ou iguais às dimensões máximas permitidas e definidas neste trabalho através de macros) e as coordenadas de um *pixel* que será utilizado como ponto de

partida para a execução do algoritmo. Este *pixel* poderá estar localizado dentro ou fora do polígono, podendo ainda pertencer a um dos lados ou a um dos vértices do polígono.

Além disso, serão utilizadas as cores correspondentes aos *pixels* apagados e acesos.

A partir destas informações, toda a área interna ou externa ao polígono deverá ser preenchida (os *pixels* deverão ser redefinidos como acesos). A área interna será preenchida se as coordenadas do ponto inicial corresponderem a um ponto interno ao polígono. Caso correspondam a um ponto externo ao polígono a área externa será preenchida.

O primeiro passo no algoritmo é verificar se o *pixel* desejado já está aceso. Se o mesmo estiver aceso nenhuma ação será executada. Caso contrário, o *pixel* em questão deverá ser ligado (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 atividade, os maiores valores permitidos para o número de linhas e para o número de colunas são 250 e 800 respectivamente. 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.

1. Baseado no algoritmo acima, crie o arquivo ***aula0701.h*** contendo a definição das macros **APAGADO** (' '), **ACESO** (' '), **DEFEITUOSO** ('.'), **NUMERO_MAXIMO_LINHAS** (250) e **NUMERO_MAXIMO_COLUNAS** (800).

Este arquivo deverá conter também a definição do tipo ***tipoPixel*** (enumerado contendo os valores ***apagado*** - valendo 0, ***aceso*** - valendo 1 e ***defeituoso*** - valendo -1) e do tipo ***tipoErros*** (tipo enumerado que deverá conter os códigos de retorno das funções solicitadas nos próximos itens. O tipo ***tipoErros*** deverá incluir um elemento valendo 0 e correspondendo à execução com sucesso da função.

O protótipo da função ***MostrarMonitor*** também deverá ser definido neste arquivo de cabeçalhos. Esta função deverá receber o tempo de congelamento da tela (em microsegundos), uma matriz bidimensional de *pixels* (***tipoPixel***) correspondendo à configuração do monitor em um dado momento, suas dimensões reais (***numeroMaximoLinhas*** e ***numeroMaximoColunas***) e as cores representando um *pixel* apagado e um *pixel* aceso. A função deverá exibir o conteúdo desta matriz (caracteres representado pelas macros APAGADO, ACESO e DEFEITUOSO, nas cores solicitadas).

A macro referente à combinação ***ifndef*** e ***define***, ou seja, ***_AULA0701_***, deverá ser definida como uma *string* valendo: ***"@(#)aula0701.h \$Revision\$"***.

```
tipoErros
MostrarMonitor (useconds_t tempoEspera, /* E */
                tipoPixel monitor [NUMERO_MAXIMO_LINHAS][NUMERO_MAXIMO_COLUNAS ], /* E */
                unsigned numeroMaximoLinhas, /* E */
                unsigned numeroMaximoColunas, /* E */
                char *corPixelApagado, /* E */
                char *corPixelAceso /* E */);
```

Observação:

Para esta função e para as demais funções solicitadas as linhas e colunas começam em 1 (os argumentos correspondentes devem variar entre 1 e o valor máximo permitido - inclusive). Lembre-se que para a linguagem o primeiro índice de cada dimensão de uma matriz vale 0.

Além disso, considere que o ponto 1,1 corresponde ao canto superior esquerdo do monitor (elemento 0,0 da matriz).

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 ZERO ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função).

Antes de exibir o conteúdo correspondente ao dispositivo de saída em questão, a função deverá exibir uma linha N hifens (N deverá ser igual ao número de colunas do dispositivo mais quatro) seguida por uma linha contendo os caracteres delimitadores "|" (pipe) como definido abaixo.

Cada linha deverá conter, à esquerda e à direita do conteúdo, um caractere "|" seguido/precedido por um caractere de espaço.

Após a exibição do conteúdo deverá ser exibida uma linha contendo os caracteres delimitadores, seguida por uma linha com N hifens.

Estes caracteres de delimitação deverão ser exibidos sempre usando a cor preta, com fundo branco.

Entre dois caracteres representando dois *pixels* vizinhos **NÃO PODERÁ** ser exibido nenhum caractere.



3. Inclua, no arquivo **aula0701.h**, o protótipo da função *GerarDistribuicaoInicial*. Esta função deverá receber uma matriz de *pixels* (**tipoPixel**) correspondendo ao monitor, suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*), o percentual de *pixels* defeituosos, o percentual de *pixels* apagados e as cores correspondentes aos *pixels* apagados e acesos. Se todos os argumentos forem válidos, a função deverá gerar de *forma aleatória*, a distribuição de *pixels* correspondente (preenchendo a matriz com os valores correspondentes - do tipo **tipoPixel**). A função deverá retornar ZERO ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função).

Os percentuais deverão ser maiores ou iguais a 0 e menores ou iguais a 100.

```
tipoErros
GerarDistribuicaoInicial (tipoPixel monitor [NUMERO_MAXIMO_LINHAS]NUMERO_MAXIMO_COLUNAS], /* E/S */
                        unsigned numeroMaximoLinhas /* E *//,
                        unsigned numeroMaximoColunas, /* E *//
                        float percentualDefeituosos, /* E *//
                        float percentualApagados, /* E *//
                        char *corPixelApagado, /* E *//
                        char *corPixelAceso /* E *//);
```

Observação:

Para criar uma distribuição aleatória será necessário utilizar as funções *srand*, *rand* e *time*.

A função *rand* retorna um valor inteiro pertencente a uma sequência limitada de números gerada de forma pseudo-aleatória. Esta sequência é gerada a partir de um valor chamado semente. Para a mesma semente a sequência será sempre a mesma. Uma nova semente é definida quando o dispositivo (computador) é ligado. Se um mesmo programa, implementado utilizando a função *rand* for executado várias vezes no mesmo dispositivo sem que o dispositivo seja reiniciado, a sequência gerada será a mesma. Por isso é necessário alterar o valor da semente a cada execução do programa. Isto pode ser feito utilizando-se a função *srand*. Visando garantir um valor diferente a cada execução, pode-se utilizar o valor retornado pela função *time*. Esta função retorna o número de segundos decorridos desde o instante 00:00:00 do dia 01/01/1970. Duas execuções simultâneas do mesmo programa no mesmo dispositivo, só utilizarão a mesma semente se executados dentro do mesmo segundo. A execução do código abaixo exibe 10 valores gerados de forma pseudo-aleatória. Note que uma nova semente só precisa ser definida uma vez por execução.

```
{
    ...
    unsigned semente;
    semente = time (NULL);
    srand (semente)
    for (indice = 0; indice < 10; indice++)
        printf ("%i\n", rand ());
    ...
}
```

4. Inclua, no arquivo ***aula0701.c***, a implementação da função *GerarDistribuicaoInicial*.
5. Inclua, nos arquivos de dependências, as macros ***LIBMONITOROBS*** (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.
6. Crie o arquivo ***aula0702.c*** contendo o programa de testes para as funções *GerarDistribuicaoInicial* e *MostrarMonitor*. Este programa deverá receber, através dos argumentos de linha de comando, o tempo de congelamento da exibição (em us), as dimensões reais do monitor (número de linhas e número de colunas), o percentual de *pixels* com defeito, o percentual de *pixels* apagados e as cores correspondentes aos *pixels* apagados e acesos. A partir destes valores, a função deverá executar a função *GerarDistribuicaoInicial* visando preencher a matriz com os dados correspondentes. A seguir deverá executar a função *MostrarMonitor*.

As cores deverão ser representadas pelos seguintes nomes: preto, vermelho, verde, amarelo, azul, magenta, cinza e branco.

O programa deverá permitir que vários testes sejam realizados, incluindo:

- a) todos os *pixels* apagados (sem *pixels* defeituosos).
- b) todos os *pixels* apagados (com *pixels* defeituosos).
- c) todos os *pixels* acesos (sem *pixels* defeituosos).
- d) todos os *pixels* acesos (com *pixels* defeituosos).
- e) todos os *pixels* com defeito.

f) distribuição aleatória com as três possibilidades.

Exemplo:

```
./aula0702 <tempo-congelamento> <numero-linhas> <numero-colunas> <percentual-defeituosos> <percentual-apagados> <cor-apagado> <cor-aceso>
```

```
./aula0702 5 100 200 2.5 70 azul preto
```

7. Inclua, nos arquivos de dependências, as macros **AULA0702OBS** 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.
8. Gere e teste as 20 versões do executável **aula0702** usando, na linkedição, a biblioteca **libmonitor.a**.
9. Submeta os arquivos **aula0701.h**, **aula0701.c**, **aula0702.c** e ***makefile** ao sistema de controle de versão.
10. Recupere uma cópia de leitura do arquivo **aula0702.c** e uma cópia de escrita dos demais arquivos.
11. Inclua, no arquivo **aula0701.h**, a definição do protótipo da função *LimparMonitor*. Esta função deverá receber uma matriz de *pixels* (**tipoPixel**) correspondendo à configuração atual do monitor, suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*) e as cores correspondentes aos *pixels* apagados e acesos. 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. A função deverá retornar ZERO) ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função).

```
tipoErros
LimparMonitor (tipoPixel monitor [NUMERO_MAXIMO_LINHAS ][NUMERO_MAXIMO_COLUNAS], /* E/S */
               unsigned numeroMaximoLinhas, /* E */
               unsigned numeroMaximoColunas, /* E */
               char *corPixelApagado, /* E */
               char *corPixelAceso /* E */);
```

12. Inclua, no arquivo **aula0701.c**, a implementação da função *LimparMonitor*.
13. Crie o arquivo **aula0703.c** contendo o programa de testes para a função *LimparMonitor*. Este programa deverá receber, através dos argumentos de linha de comando, o tempo de congelamento da exibição, as dimensões reais do monitor (número de linhas e número de colunas), o percentual de *pixels* com defeito, o percentual de *pixels* apagados e as cores correspondentes aos *pixels* apagados e acesos. A partir destes valores, o programa deverá executar a função *GerarDistribuicaoInicial* visando preencher a matriz com os dados correspondentes. A seguir deverá executar as funções *MostrarMonitor*, *LimparMonitor* e *MostrarMonitor* novamente.

Exemplo:

```
./aula0703 <tempo-congelamento> <numero-linhas> <numero-colunas> <percentual-defeituosos> <percentual-apagados> <cor-apagado> <cor-aceso>
```

```
./aula0703 5 100 200 15 20 amarelo vermelho
```

14. Inclua as declarações necessárias nos arquivos de dependências.

15. Gere e teste as 20 versões do executável **aula0703** usando, na linkedição, a biblioteca **libmonitor.a**.
16. Submeta os arquivos **aula0701.h**, **aula0701.c**, **aula0703.c** e ***makefile** ao sistema de controle de versão.
17. Recupere uma cópia de leitura do arquivo **aula0703.c** e uma cópia de escrita dos arquivos demais arquivos.
18. Inclua, no arquivo **aula0701.h**, a definição do protótipo da função *DesenharReta*. Esta função deverá receber uma matriz de *pixels* (*tipoPixel*) correspondendo à configuração atual do monitor, suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*), os valores (*linha* e *coluna*) correspondentes a dois pontos (os dois vértices da reta desejada) e as cores correspondentes aos *pixels* apagados e acesos. Se todos os argumentos forem válidos, a função deverá acender todos os *pixels* necessários para desenhar a reta desejada. A função deverá retornar ZERO ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função).

Caso um pixel necessário para desenhar a reta esteja defeituoso, a função deverá retornar o código de erro correspondente. Antes de desenhar a reta (marcar os *pixels* correspondentes à reta com o valor correspondente a aceso (*tipoPixel*), verifique todos os pontos necessários). As retas não serão necessariamente horizontais ou verticais e dada a simplicidade do dispositivo, poderão apresentar imperfeições.

```

tipoErros
DesenharReta (tipoPixel monitor [NUMERO_MAXIMO_LINHAS ][NUMERO_MAXIMO_COLUNAS], /* E/S */
    unsigned numeroMaximoLinhas, /* E */
    unsigned numeroMaximoColunas, /* E */
    unsigned linhaA, /* E */
    unsigned colunaA, /* E */
    unsigned linhaB, /* E */
    unsigned colunaB, /* E */
    char *corPixelApagado, /* E */
    char *corPixelAceso /* E */);

```

19. Inclua, no arquivo **aula0701.c**, a implementação da função *DesenharReta*.
20. Crie o arquivo **aula0704.c** contendo o programa de testes para a função *DesenharReta*. Este programa deverá receber, através dos argumentos de linha de comando, o tempo de congelamento da exibição (us), as dimensões reais do monitor (número de linhas e número de colunas), o percentual de *pixels* com defeito, o percentual de *pixels* apagados, os valores correspondentes aos pontos A e B (vértices da reta) e as cores correspondentes aos *pixels* apagados e acesos. A partir destes valores, o programa deverá executar a função *GerarDistribuicaoInicial* visando preencher a matriz com os dados correspondentes. A seguir deverá executar as funções *MostrarMonitor*, *LimparMonitor*, *DesenharReta* e *MostrarMonitor*.

Exemplo:

```

./aula0704 <tempo-congelamento> <numero-linhas> <numero-colunas> <percentual-defeituosos> <percentual-apagados> <linha-ponto-1> <coluna-ponto-1> <linha-ponto-2> <coluna-ponto-2> <cor-apagado> <cor-aceso>

./aula0704 5 100 200 10 70 30 5 10 70 verde cinza

```

21. Inclua as declarações necessárias nos arquivos de dependências.
22. Gere e teste as 20 versões do executável **aula0704** usando, na linkedição, a biblioteca **libmonitor.a**.

23. Submeta os arquivos ***aula0701.h***, ***aula0701.c***, ***aula0704.c*** e ****makefile*** ao sistema de controle de versão.
24. Recupere uma cópia de leitura do arquivo ***aula0704.c*** e uma cópia de escrita dos demais arquivos.
25. Inclua, no arquivo ***aula0701.h***, a definição do protótipo da função *DesenharPoligono*. Esta função deverá receber uma matriz de *pixels* (***tipoPixel***) correspondendo à configuração atual do monitor, suas dimensões reais (*numeroMaximoLinhas* e *numeroMaximoColunas*), o número de vértices do polígono, as informações (*linha*, *coluna*) sobre estes vértices e as cores correspondentes aos *pixels* apagados e acesos. 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 ZERO 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], /* E/S */
    unsigned numeroMaximoLinhas, /* E */
    unsigned numeroMaximoColunas, /* E */
    unsigned numeroVertices, /* E */
    unsigned linhasVertices [NUMERO_MAXIMO_LINHAS], /* E */
    unsigned colunasVertices [NUMERO_MAXIMO_COLUNAS], /* E */
    char *corPixelApagado, /* E */
    char *corPixelAceso /* E */);
```

26. Inclua, no arquivo de ***aula0701.c***, a implementação da função *DesenharPoligono*.
27. Crie o arquivo ***aula0705.c*** contendo o programa de testes para a função *DesenharPoligono*. Este programa deverá receber, através dos argumentos de linha de comando, o tempo de congelamento da exibição (us), as dimensões reais do monitor (número de linhas e número de colunas), o percentual de *pixels* com defeito, o percentual de *pixels* apagados, o número de vértices do polígono, os valores correspondentes aos vértices deste polígono e as cores correspondentes aos *pixels* apagados e acesos. A partir destes valores, o programa deverá executar a função *GerarDistribuicaoInicial* visando preencher a matriz com os dados correspondentes. A seguir deverá executar as funções *MostrarMonitor*, *DesenharPoligono* e *MostrarMonitor*.

Exemplo:

```
./aula0705 <tempo-congelamento> <numero-linhas> <numero-colunas> <percentual-defeituosos> <percentual-apagados> <numero-vertices> <linha-vertice-1> <coluna-vertice-1> <linha-vertice-2> <coluna-vertice-2> . . . <linha-vertice-N> <coluna-vertice-N> <cor-apagado> <cor-aceso>
```

```
./aula0705 5 100 200 10 80 3 5 70 30 105 30 35 cinza azul
```

28. Inclua as declarações necessárias nos arquivos de dependências.
29. Gere e teste as 20 versões do executável ***aula0705*** usando, na linkedição, a biblioteca ***libmonitor.a***.
30. Submeta os arquivos ***aula0701.h***, ***aula0701.c***, ***aula0705.c*** e ****makefile*** ao sistema de controle de versão.
31. Recupere uma cópia de leitura do arquivo ***aula0705.c*** e uma cópia de escrita dos demais arquivos.

32. Inclua, no arquivo **aula0701.h**, a definição do protótipo da função *PreencherPoligono*. Esta função deverá receber o tempo de congelamento da tela (em us), 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 (*linha* e *coluna*) sobre um *pixel* que esteja localizado dentro ou fora do polígono e as cores correspondentes aos *pixels* apagados e acesos. Se todos os argumentos forem válidos, a área interna ou externa ao polígono deverá ser preenchida de acordo com o algoritmo definido acima. A função deverá retornar ZERO ou o código de erro correspondente (todas as possíveis condições de erro detectáveis na função).

Se, durante o procedimento de preenchimento, um *pixel* defeituoso for 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 (useconds_t tempoEspera, /* E */
    tipoPixel monitor[NUMERO_MAXIMO_LINHAS ][NUMERO_MAXIMO_COLUNAS], /* E/S */
    unsigned numeroMaximoLinhas, /* E */
    unsigned numeroMaximoColunas, /* E */
    unsigned linha, /* E */
    unsigned coluna, /* E */
    char *corPixelApagado, /* E */
    char *corPixelAceso /* E */);
```

33. Inclua, no arquivo de **aula0701.c**, a implementação da função *PreencherPoligono*.
34. Crie o arquivo **aula0706.c** contendo a implementação do programa de testes para a função *PreencherPoligono*. Este programa deverá receber o tempo de congelamento da tela (em us), as dimensões reais do monitor (número de linhas e número de colunas), o percentual de *pixels* com defeito, o percentual de *pixels* apagados, as informações (linha, coluna) sobre um ponto qualquer do monitor (interno ou externo ao polígono), o número de vértices do polígono desejado, as informações (linha, coluna) sobre estes vértices e as cores correspondentes aos *pixels* apagados e acesos. A partir destes valores, o programa deverá executar a função *GerarDistribuicaoInicial* visando preencher a matriz com os dados correspondentes. A seguir deverá executar as funções *LimparMonitor*, *DesenharPoligono*, *MostrarMonitor* e *PreencherPoligono*.

Exemplo:

```
./aula0706 <tempo-congelamento> <numero-linhas> <numero-colunas> <percentual#defeituosos> <percentual-apagados> <linha-ponto>
<coluna-ponto> <numero-vertices> <linha#vertice-1> <coluna-vertice-1> <linha-vertice-2> <coluna-vertice-2> . . . <linha-vertice-N>
<coluna-vertice-N> <cor-apagado> <cor-aceso>
```

```
./aula0706 5 100 200 10 80 50 50 3 5 70 30 105 30 35 vermelho verde
```

35. Inclua as declarações necessárias nos arquivos de dependências.
36. Gere e teste as 20 versões do executável **aula0706** usando, na linkedição, a biblioteca **libmonitor.a**.
37. Submeta os arquivos **aula0701.h**, **aula0701.c**, **aula0706.c** e ***makefile** ao sistema de controle de versão.
38. Recupere uma cópia de leitura dos arquivos **aula0701.h**, **aula0701.c** e **aula0706.c** e uma cópia de escrita dos arquivos ***makefile**.
39. Limpe o diretório (make clean-all).