MAC211 - Laboratório de Programação I

BCC – Primeiro Semestre de 2015

Projeto: Bombardeio Naval

1 Apresentação Geral

Este projeto, Bombardeio Naval, consiste na implementação de um jogo inspirado na famosa Batalha Naval.

A sua tarefa é atravessar uma determinada região de um oceano, onde está ocorrendo uma guerra naval, utilizando um barco a remo.

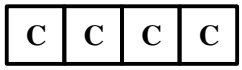
Para saber se você tem chances de sobrevivência nesta tarefa, escreva um programa na linguagem C, que simule esta situação.

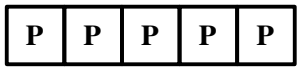
A guerra naval ocorre numa região retangular do Oceano Atlântico, descrita em um mapa, que será representado por uma matriz de caracteres contendo informações sobre as embarcações presentes na região.

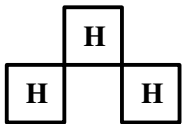
Os tipos das embarcações são os seguintes:

Submarino : 

Destroyer : 

Cruzador : 

Porta-Aviao : 

Hidro-Aviao : 

No início, na matriz que representa a região, as posições ocupadas pelas embarcações devem ter os caracteres identificando cada tipo de embarcação e as posições que representam água devem ter '.' , como no modelo a seguir:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	.	D	.	.	.	.	.	.	.	.	H	.	.	.	D	.	.	.
2	.	D	.	.	C	.	.	D	D	.	.	H	.	D	.	.	P	.
3	.	.	.	C	.	.	.	.	.	.	H	.	.	.	.	P	.	.
4	.	.	C	.	.	S	.	.	.	.	.	.	.	.	P	.	.	.
5	.	C	.	.	.	.	.	H	.	H	.	.	.	P	.	.	.	.
6	.	.	.	.	D	D	.	.	H	.	.	.	P	.	.	.	D	.
7	S	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	D	.
8	.	.	.	.	.	.	C	.	.	S	.	.	S	.	.	.	.	.
9	.	.	.	S	.	.	.	C	.	.	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	.	.	C	.	.	.	C	C	C	C	.	.
11	.	.	P	P	P	P	P	.	.	C	.	.	.	.	.	.	.	C
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	H	.	C
13	.	.	.	.	.	.	.	.	.	.	.	D	D	.	H	.	.	C
14	.	.	.	S	.	.	D	.	.	H	.	.	.	.	.	H	.	C
15	.	.	.	.	.	D	.	.	H	.	H	.	.	.	.	.	.	.

Inicialmente, posicione o seu barco, via teclado, em alguma coluna da primeira linha. O barco ocupa uma posição da matriz. Esta posição na matriz deve conter 'B' (posição atual do barco). Posições antigas serão marcadas ccom 'T'.

Seu objetivo é conduzir o barco até a última linha, através de remadas sucessivas. (Com isto, você terá atravessado a região.) Mas, durante a travessia, o mar não estará tranqüilo. Você terá que desviar de obstáculos (embarcações) que existem no caminho e, além disso, estará sujeito, a cada remada, a três tiros que podem acertar o seu barco. Cada tiro atinge uma única posição da matriz. (Se isto acontecer, você se deu mal e o jogo terá acabado!)

Os tiros são disparados aleatoriamente pelo computador (podendo haver repetições atingidas) e são tão fortes que se um tiro acertar parte de uma embarcação, esta embarcação deve afundar totalmente, e o local fica livre para a navegação do seu barco.

O jogo tem o seu andamento determinado a cada remada sua, até que você alcance a última linha da matriz, ou o seu barco afunde ou fique num beco sem saída por três jogadas consecutivas.

2 Objetivos gerais do projeto

A disciplina de Laboratório de Programação do curso de Bacharelado em Ciência da Computação é a primeira disciplina cujo objetivo é com que o aluno faça um projeto grande (isso não quer dizer que outras disciplinas anteriores não possam fazer os alunos trabalhar bastante). Por projeto grande, entenda-se um projeto que envolva diversas funcionalidades, diversas estruturas de dados, manipulação de entrada/ saída com vários dispositivos, documentação para o usuário e para o desenvolvedor, aplicação de alguns princípios de engenharia de software,

principalmente para testes e etc. Um objetivo afim da disciplina é que os alunos divertam-se bastante. Assim, seguindo uma receita de sucesso, o projeto deste semestre será um jogo, desta vez com cenário de um conflito naval durante a segunda guerra. Para facilitar a vida do professor e do monitor da disciplina (e indiretamente a dos alunos) o desenvolvimento do projeto será feito e cobrado em partes ao longo do semestre e para efeitos de avaliação, serão considerados:

- Documentação de desenvolvedor
- Documentação de usuário
- Organização e clareza do código
- Uso correto de estruturas de dados
- Uso correto de técnicas de programação
- A independência dos módulos/bibliotecas
- O interface para comunicação entre os modelos

3 Detalhes de implementação

O seu programa deve utilizar, obrigatoriamente, pelo menos as funções descritas a seguir. Para cada uma delas, decida quais parâmetros são necessários, e não utilize variáveis globais.

(1) **leia_mapa**

Lê de um arquivo para uma matriz, o mapa da região cujo formato deve ser o mesmo do modelo descrito anteriormente. Suponha que a primeira linha desse arquivo contém as 'dimensões' da região (isto é, número de linhas e de colunas).

É claro que o nome desse arquivo deve ser fornecido pelo usuário.

(2) **escreva_mapa_tela**

Escreve na tela o mapa da região, destacando cada uma de suas posições, e escreve também os números associados a cada linha e coluna.

Como o jogador não pode saber onde estão as embarcações, inicialmente, todas as posições devem conter, por exemplo, o caractere '−'. Nas situações seguintes, o jogador pode ver todas as posições ocupadas pelo barco (isto é, as que contêm os caracteres 'T' e 'B'), e também as posições que foram atingidas por um tiro (isto é, as que contêm os caracteres '=', '∗', '+' e '!'). Nas demais posições continua aparecendo o caractere '−'.

(3) **escreva_mapa_arquivo**

Escreve a matriz representando o mapa para o arquivo que vai conter toda a saída do programa.

(4) **posiciona_barco**

Lê do teclado a posição inicial do barco e marca com 'B' esta posição na matriz. Note que o barco não pode começar num lugar ocupado por alguma embarcação.

(5) **rema_barco**

Movimenta o barco ou para baixo (na direção da última linha) ou na horizontal, tomando cuidado para ele não ir para uma posição ocupada por alguma embarcação.

O tipo do movimento deve ser lido do teclado; ou seja, o jogador deve digitar um dos seguintes caracteres: 'b', 'c', 'd' ou 'e', indicando, respectivamente, para baixo, para cima, para a direita ou para a esquerda.

As posições ocupadas pelo barco devem ser marcadas na matriz com 'T', exceto a posição atual que deve conter 'B'.

(6) **dispara_tiros**

Determina os três tiros a serem disparados pelo computador, a cada remada. Imprime as mensagens correspondentes ao efeito de cada tiro, e atualiza a matriz representando o mapa. Após cada tiro, escreva (para tela e arquivo) a matriz atualizada.

A função disparaTiros deve utilizar as funções descritas a seguir.

(7) **coordenadas_tiro**

Determina as coordenadas de um tiro, utilizando a função sorteia.

(8) **sorteia**

Tem como parâmetro um inteiro k, e devolve um inteiro no intervalo $[1, k]$ fornecido pela seguinte expressão

```
(int)(1 + (rand()/(RAND_MAX+1.0))*k) .
```

(9) **identifica_alvo_atingido**

Imprime as coordenadas de um tiro e a mensagem correspondente ao efeito desse tiro. Quando o tiro acertar uma embarcação, deve-se especificar o seu tipo.

Além disso, atualiza na matriz a posição atingida pelo tiro, com um dos seguintes caracteres:

'=' se o tiro atingiu a água;

'*' se o tiro atingiu alguma embarcação;

'!' se o tiro atingiu o barco;

'+' se o tiro atingiu alguma posição do caminho percorrido pelo barco, exceto a posição atual.

(10) **afunda_embarcação**

Afunda totalmente a embarcação atingida por um tiro; ou seja, atribui '$*$' a todas as posições da matriz ocupadas por essa embarcação.

Escreva uma função para afundar cada tipo de embarcação, exceto o submarino pois não há mais nada a afundar. Ou seja, defina as funções: **afunda_destroyer, afunda_cruzador, afunda_porta_aviao e afunda_hidro_aviao**.

Observações:

(1) Suponha que as embarcações possam ser posicionadas ou na horizontal ou na vertical ou na diagonal, e que elas nunca se encostam (nem mesmo na diagonal), mas o barco pode encostar em qualquer embarcação.

Além disso, considere que as embarcações não se movimentam durante o jogo.

(2) Quando terminar o jogo, imprima uma mensagem descrevendo o que aconteceu com o barco.

(3) Este program deve gerar duas saídas.

- Saída para a tela, escrevendo o mapa que o usuário pode ver. Este mapa deve ser mostrado após cada movimento do barco, e também após cada tiro, com mensagens correspondentes.
- Saída para um arquivo, onde deverão ser escritos a matriz cada vez que ela é alterada, e as mensagens referentes aos tiros e movimentos do barco.

(4) A função `rand` devolve o próximo inteiro de uma seqüência de números inteiros pseudo-aleatórios no intervalo de 0 a `RAND_MAX`. A constante `RAND_MAX` representa o maior inteiro correspondente ao tipo *int*.

Para que seqüências diferentes sejam geradas cada vez que o programa é executado, pode-se utilizar a função `srand`, que estabelece o argumento como sendo a semente para a seqüência de números pseudo-aleatórios que serão devolvidos pelas chamadas subseqüentes à função `rand`.

A semente pode ser gerada, por exemplo, utilizando a função `time`, que devolve o número de segundos passados desde a meia-noite (00:00:00) do primeiro dia de janeiro de 1970, de acordo com o relógio do sistema. Para inicializar a semente, chame a função `srand`, uma única vez, no início da função principal, da seguinte forma:

```
srand ((unsigned int) time(NULL));
```

Para utilizar em seu programa as funções `rand` e `srand`, e a constante `RAND_MAX`, inclua o arquivo *stdlib.h*, e para utilizar a função `time`, inclua o arquivo *time.h*.

4 Variações

Use sua imaginação para introduzir variações. Qualquer tentativa bem sucedida de melhorar o programa trará nota de bônus. O objetivo dessa liberdade é fazer um jogo divertido de jogar e de ser programado!

Se alguém quiser reconhecer um click de mouse para as casas vizinhas válidas do barco, esta opção para o projeto pode ser implementada.

No exemplo de uso de OpenGL fornecido, há uma "placa" no canto superior esquerdo para escrever mensagens. Se quiser, pode modificar para colocar informações que achar interessante. Sinta-se à vontade para usar outras formas de apresentar estes dados, se preferir.

Não se esqueça de colocar um conjunto de texturas melhor do que a que está no exemplo fornecido.

5 Alguns conselhos

Não tentem fazer o jogo todo de uma vez. O projeto será dividido em partes por boas razões. Concentrem-se em cada uma delas e deixem as outras para o seu devido tempo. Procurem ser organizados, isto é, planejem como dividir as tarefas de programação, testes e documentação. Planejem a arquitetura dos módulos para eles se integrarem no final de um modo bastante suave. Se tudo for feito com o devido cuidado, mesmo que alguma alteração seja necessária em uma parte já pronta, ela não deverá comprometer os novos módulos. Documente tudo o que fizerem, idéias, desenhos, testes, código e etc. Para cada fase, faça um pequeno relatório onde descreve as principais decisões tomadas, as estruturas de dados escolhidas, o conjunto de módulos, algoritmos, etc. Basta um arquivo texto. Escreva um bom manual de usuário! É incrível o número de programas que são ignorados porque não é possível adivinhar como usá-los. Teste seu programa em diversas situações. Crie arquivos, rotinas, ou mesmo programas especiais para verificar cada função. Se você fizer alguma modificação, poderá rodar a bateria de testes novamente, praticamente sem esforço adicional. Recomenda-se utilizar um controlador de versões como o github ou svn. Curta seu programa quando ele estiver pronto!

6 Segunda Fase — Parte Gráfica — Geração de imagens

O objetivo desta fase é mostrar na tela, com animação, o movimento dos objetos segundo feito na fase anterior. Para desenhar no `X11`, o prof. Marcos D. Gubitoso fez uma pequena biblioteca (`xwc`) de funções que simplifica bastante o uso das funções do `X11`. Qualquer sistema gráfico um pouco mais sofisticado precisa de um número muito grande de parâmetros para qualquer elemento desenhado na tela. No Windows, em particular, este número é maior ainda: cada janela pode ser configurada para ser apresentada em um número muito variado de dispositivos, com diferentes formatos de representação de cor, padrões de desenho, etc. Além disso, a tela pode ser local ou remota. A biblioteca `xwc` usa parâmetros default para a maioria dos casos e permite janelas apenas na máquina local. Por outro lado, fornece todas as informações para que as chamadas fundamentais da Xlib possam ser feitas. Existem outras bibliotecas gráficas muito mais poderosas e que podem ser usadas, se você preferir. Uma delas é a `OpenGL` que é descrita a seguir. A escolha dependerá da familiaridade que você já tiver com estas bibliotecas, pois a curva de aprendizado do `xwc` é bem pequena.

6.1 Desenhos

Faça os desenhos dos objetos que desejar, de preferência padrão xpm caso se use o `xwc` ou PNG caso se use o *OpenGL*, e para o fundo, sugiro escolher uma imagem única. No caso do barco, você deverá ter vários desenhos com diferentes orientações e usar o mais conveniente. Uma sugestão é construir um objeto 'barco' com um vetor de desenhos — digamos um para cada direção do barco: para baixo, para cima, para a direita, para a esquerda — e uma função de desenho que recebe como argumentos o barco, a posição e a orientação; ela então escolhe qual o desenho mais adequado no vetor.

7 OpenGL

7.1 OpenGL

Deixamos um pacote com uma descrição simples do *OpenGL/Glut* bem como um exemplo desenvolvido pelo professor Gubi à disposição na página do projeto. Se alguém tiver problemas na compilação ou execução deve relatar no fórum ou perguntar ao monitor. Em minha distribuição baseada na debian, tive que instalar primeiro os pacotes `freeglut3` e `freeglut3-dev` e demais pacotes de desenvolvimento dos quais eles dependem.

7.2 Comandos do usuário

O *OpenGL*, juntamente com o *freeGlut*, usa um laço contínuo para capturar eventos e redesenhar a janela. As ações do usuário são eventos tratados por funções especiais, chamadas *callbacks* e que devem ser registradas no início do programa.

É possível capturar eventos do teclado por duas funções, uma para teclas normais e outra para teclas especiais (veja o exemplo fornecido):

- `glutKeyboardFunc(`*nome da função*`);` — registra a função que trata das teclas normais, que podem ser representadas por um caractere. A função registrada irá receber 3 parâmetros: o caractere correspondente à tecla (`unsigned char`) e a posição (x, y) em pixels da ocorrência.

- `glutSpecialFunc(`*nome da função*`);` — idem, para as outras teclas, especiais, como *Page UP*, *Page Down*, setas, etc.

Se alguém quiser estender o jogo de forma a reconhecer o *mouse*, é possível capturar movimento e alteração no estado dos botões.

- `glutMotionFunc(`*função*`);` — a função recebe a nova posição do *mouse* quando ele se mover com algum botão pressionado. `glutPassiveMotionFunc` faz o mesmo, mas quando o *mouse* se move sem nenhum botão pressionado.

- `glutMouseFunc(`*função*`);` — a função recebe o código do botão (`GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` ou `GLUT_RIGHT_BUTTON`), um código indicando se o botão doi pressionado ou liberado (`GLUT_UP` ou `GLUT_DOWN`) e a posição do cursor no momento da ocorrência.

Existem diversas outras possibilidades de registro de *callbacks*, verifique a documentação e os exemplos.

O ciclo do jogo agora é comandado pelo *OpenGL/freeGlut*. Para isso é preciso registrar a função de temporização com

`glutTimerFunc(`*atraso*, *função*, `valor);`

Os parâmetros são:

1. *atraso* — tempo em milisegundos para o primeiro disparo acontecer.

2. *função* — função que deve ser chamada.

3. *valor* — um número inteiro que é passado para a *função* como argumento.

É importante que a *função* registre um novo disparo, ser se desejar a repetição, como é o nosso caso.

7.3 SDL

Alternativamente ao Glut, que toma conta do gerenciamento das janelas e dos eventos de entrada e saída, pode-se usar o pacote *SDL* (*Simple Directmedia Layer*). Eventualmente, o monitor trará mais informação a respeito.

8 XWC

Para começar, pegue o `xwc` no link do projeto e expanda o arquivo (`tar xf`). Em minha distribuição baseada na debian, tive que instalar primeiro o pacote `libxpm-dev` e demais pacotes de desenvolvimento dos quais ele depende, para só então rodar com sucesso o script `configure`, que gera o Makefile, e por fim o `make`:

Exemplo:

```
eu@minha.maquina$ tar xf xwc.tar
eu@minha.maquina$ ./configure
./configure
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking how to run the C preprocessor... gcc -E
checking for X... libraries , headers
checking for XFlush in -lX11... yes
checking for XpmReadFileToPixmap in -lXpm... yes
checking for tan in -lm... yes
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
```

```
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for unistd.h... (cached) yes
configure: creating ./config.status
config.status: creating Makefile
eu@minha.maquina$ make
cc -Wall -g  -I -g -O2   -c -o teste.o teste.c
cc -Wall -g  -I -g -O2   -c -o xwc.o xwc.c
cc -Wall -g  -I -g -O2 -o teste  teste.o  xwc.o  -lm -lXpm -lX11
cc -Wall -g  -I -g -O2   -c -o teste2.o teste2.c
cc -Wall -g  -I -g -O2 -o teste2 teste2.o xwc.o  -lm -lXpm -lX11
cc -Wall -g  -I -g -O2   -c -o teste3.o teste3.c
cc -Wall -g  -I -g -O2 -o teste3 teste3.o xwc.o  -lm -lXpm -lX11
eu@minha.maquina$
```

Rode os programas de teste (`teste`, `teste2` e `teste3`), examine o fonte e tente fazer suas próprias modificações. O arquivo `xwc.h` serve como documentação parcial das funções. Uma documentação mais detalhada se encontra em seção a seguir. Nos exemplos (`teste*.c`) você provavelmente vai encontrar exemplos para tudo o que precisará fazer no projeto. O `teste2` mostra uma animação e o `teste3` mostra como construir uma imagem de modo simples, usando o formato xpm, que tanto pode ficar em arquivo como ser definida em memória, e ter uma máscara para desenho. Veja também a função `usleep` para controlar o tempo de animação ou, melhor ainda, tente entender como fazer funcionar a `setitimer`. Veja um exemplo em `timer.c`. Como o `xwc` ainda é uma biblioteca de nível relativamente baixo, desenvolva uma ou mais camadas de abstração para tornar a programação mais conveniente e modular, como desenho de objetos por exemplo.

8.1 Dicas do `xwc`

Existe um tipo básico para todas as janelas e regiões desenháveis: WINDOW. Ponteiros para este tipo descrevem objetos que podem ser usados em qualquer função de manipulação de imagens. Em particular, uma janela é uma variável do tipo WINDOW *. Existem mais tipos de ponteiros para WINDOW:

1. PIC — corresponde a uma área de rascunho de desenhos. Você pode escrever, copiar ou desenhar em um objeto destes sem que suas modificações apareçam na tela. Depois você pode copiar o pedaço de interesse para uma janela ativa.

2. MASK — indica uma máscara de desenho. Uma máscara é uma imagem preto e branco, onde os pontos brancos correspondem a uma região transparente. Uma máscara está sempre associada a um PIC. Com isto é possível construir *sprites*, isto é, imagens com forma não retangular.

8.2 Funções gerais

As funções disponíveis na biblioteca são as seguintes:

- `WINDOW* InitGraph(int WIDTH, int HEIGHT, char *name)`
  Cria uma janela de altura HEIGHT e largura WIDTH, com nome name. Esta função deve ser chamada pelo menos uma vez antes de qualquer outra!!!

- `void WCor(WINDOW *win, int c)`
  Define a cor para a janela win.

- `void WPlot(WINDOW *win, int x, int y, int c)`
  Desenha um ponto de cor c, na posicao (x,y) da janela win.

- `void WLine(WINDOW *win, int x1, int y1, int x2, int y2, int c)`
  Desenha uma linha de cor c, na janela win, entre os pontos (x1,y1) e (x2,y2).

- `void WRect(WINDOW *win, int x, int y, int w, int h, int c)`
  Desenha um retângulo no ponto (x,y) de altura h e largura w.

- `void WFillRect(WINDOW *win, int x, int y, int w, int h, int c)`
  Idem, mas o retângulo é cheio.

- `void WArc(WINDOW *win, int x, int y, int a1, int a2, int w, int h, int c)`

- `void WFillArc(WINDOW *win, int x, int y, int a1, int a2, int w, int h, int c)`
  Desenha um arco de elipse na posição (x,y), inscrito em um retângulo de altura h e largura w, iniciando no angulo a1 e terminando em a2. Os ângulos sao dados em graus/64, isto é 64 corresponde a um grau. Exemplo, para desenhar um quadrante de um circulo: WFillArc(win, 100,100, 0, 90*64, 20, 20, 3);

- `void WClear(WINDOW *win)`
  Limpa (apaga o conteúdo) a janela win.

- `void WPrint(WINDOW *win, int x, int y, char *msg)`
  Escreve msg na posição (x,y) da janela win.

- `PIC NewPic(WINDOW *win, int w, int h)`
  Cria um PIC no formato interno da janela win, de largura w e altura h.

- `void FreePic(PIC pic)`
  Destrói o PIC.

- `void PutPic(PIC pic1, PIC pic2, int x0, int y0, int w, int h, int x, int y)`
  Desenha o PIC pic2 em pic1, posição (x,y). x0, y0, w e h indicam o pedaço (retangular) de pic2 a ser usado.

- `int WNamedColor(char *name)`
  Escolhe uma cor pelo nome. Retorna seu valor. Or nomes das cores se encontram no arquivo /etc/X11/rgb.txt e são padronizadas.

- `void WShow(WINDOW *win)`
  Mostra a janela na tela.

- `void WHide(WINDOW *win)`
  Esconde a janela
- `void WDestroy(WINDOW *win)`
  Destrói a janela.
- `void CloseGraph()`
  Finaliza o sistema gráfico

8.3 Máscaras

- `MASK NewMask(WINDOW *w, int wd, int h)`
  Cria uma mascara nova, de largura wd e altura h. A janela w indica a estrutura de base para a máscara.
- `void SetMask(PIC p, MASK mask)`
  Ativa a mascara para o objeto p. p pode ser a janela base ou outra qualquer.
- `void UnSetMask(PIC p)`
  Desativa a mascara do objeto p. XPM XPM é um formato gráfico simples e flexível para pequenas imagens. Estas funções permitem sua manipulação. No formato Xpm, a cor transparente é indicada por 'None'. Esta informação é usada para especificar uma máscara ao mesmo tempo que se indica a figura.
- `PIC ReadPic(WINDOW *w, char *fname, MASK m)`
  Lê um arquivo xpm no arquivo fname e retorna o PIC correspondente. Retorna a mascara associada em 'm', se este for diferente de NULL. A janela w serve para indicar a estrutura a ser usada.
- `PIC MountPic(WINDOW *w, char **data, MASK m)`
  Idem, mas lê da memória (data) ao invés do arquivo. Cada elemento de data corresponde a uma linha do arquivo. Veja os exemplos.
- `int WritePic(PIC p, char *fname, MASK m)`
  Escreve o conteúdo do PIC p no arquivo fname, eventualmente mascarando com m, se m for diferente de NULL. Teclado A interface de teclado é bem simples, mas usável.
- `void InitKBD(WINDOW *w)`
  Inicializa a captura de eventos de teclado
- `int WCheckKBD(WINDOW *w)`
  Verifica se existe alguma tecla disponível
- `KeyCode WGetKey(WINDOW *w)`
  Pega a próxima tecla e retorna o código X. Veja em /usr/X11R6/ include/X11/keysymdef.h, ou o arquivo correspondente.
- `KeySym WLastKeySym()`
  Retorna o KeySym da última tecla lida.

8.4 Miscelânea

Estas funções servem para obter informações mais detalhadas,que permitem o uso direto das funções do Xlib. Veja as páginas de manual para maiores detalhes.

- `Display *GetDisplay()`
  Retorna o display utilizado.
- `GetDraw(w)`
  Retorna o "drawable".
- `GetGC(w)`
  Retorna o "contexto gráfico" (gc).