Details of kiteLib Foundation Class API

The main goals of designing the **Mywin** class were to enable the students to develop C++ programs with *Graphical User Interface* and facilitate learning about computer graphics. In my opinion, the **Mywin** class has achieved these goals. This manual was created to act as a quick reference to the member functions of the **Mywin** class. It is to be used in conjuction with the theory lecture discussions during the *C++ programming course*. Also note that the windows created as objects of this class should normally not be overlapping. While the program is running, these window objects should not be moved, resized, minimized or closed. Although, this is not prevented, the effect of these actions would be to clear the contents already displayed on the window object, instantaneously; this may cause the program to go haywire, unless the programmer has taken special care to tackle these situations.

'Mywin' Constructors:

The 'Mywin' class has two constructors. The default (parameterless) constructor creates a window object of 400 pixels width and 300 pixels height; the title of the window would be "KITE window". The other constructor has 5 parameters. To create a window object with the second constructor, use the format as in the following example-

```
Mywin w1("My Window", 100, 125, 400, 300);
```

The first parameter, is the string that would appear in the title bar of the resultant window. The second and the third parameters (which should be unsigned integers) are the x and y coordinates in pixels of the upper left corner of the window. In other words, using the second parameter, we are specifying the distance in pixels between left edge of the monitor screen & left edge of the window and using the third parameter, distance in pixels between top edge of the screen & top edge of the window. The last two parameters (again unsigned ints) specify the width and height in pixels of the desired window object.
Once the window is created, all the coordinates are 'window relative', i.e., the origin is upper-left corner of the client area of the window. The coordinates are measured in pixels. <u>This applies to all the functions described below.</u>

void activate( );

This function is useful, when you have created more than one window. By default, the active window would be the one created last. If you want to switch focus to other window, then call this member function of that window.

void clear( );

This function clears the window of the present contents and fills it with the current background color.

void clrnchars(int x, int y, int n);

This function clears a part of the window. It clears the space that would be approximately taken by 'n' characters, if they were displayed using *winPrint()* function, starting from (x, y) coordinate position. The band cleared gets the current background color.

void kiteFlush( ) ;
This function flushes the request queue to the X server.

COLORREF GetSetForeColor(COLORREF fg);

This function sets the color of the subsequently drawn text (using the *winPrint* function described elsewhere) to the argument color. It returns the previous text color. The color values are specified using *RGB()* function as a triad of red, green and blue color intensities, which range from 0 to 255. The default value of text color is black.

COLORREF GetSetBackColor(COLORREF bg);

This function sets the *background* color of the subsequently drawn text to the given argument color. It returns the current (previous to this setting) background color. The default value of text background color is white.

void winPrint(int x, int y, char *str);

This function prints the argument string at the given (x, y) coordinate position, in the current text color on the current text background color.

Go to the *Contents* page...

int kiteGetche(int x, int y);

This function shows a cursor at the given (x, y) position and will wait for the user to input some character. It will return the ASCII value of the character input. If the character is printable, then it will be echoed at cursor position, in current text color on the current text background.

int kiteGetchar(int x, int y);

This function is similar to 'kiteGetche' function; but here, the user has to press ENTER key after the character to indicate end of input.

POINT kiteGetch(char& ch);

This function waits for the user to either input a character or click the mouse.

If user presses a key on the keyboard, then its ASCII code is stored in the argument character variable. The character input is not echoed on the window. The return value is point (0, 0) in this case. On the other hand, if the user clicks mouse instead of using keyboard, then coordinates of the point where the mouse was clicked, are returned in pixel units. In the mouse click case, NULL character '\0' is stored in the argument *ch*.

int kiteGetint(int x, int y, int n=0);

This function waits for the user to input an integer. The cursor would be waiting at the given coordinate position (x, y). The input is terminated either when ENTER is pressed or when 'n' digits have been typed, whichever occurs earlier. A character which can not be part of an integer is ignored. The third argument has default value of 0, which means that there is no restriction on the number of digits and only ENTER can terminate the input. The function returns the integer input by the user.

double kiteGetdouble(int x, int y);

This function accepts a 'float' or a 'double' number at the given position (x, y). It returns the number that is input by the user as 'double'.

int kiteGetstr(int x, int y, char *str, int n);

This function waits for the user to input a string. The cursor would be waiting at the position (x, y) provided as first two arguments. The string input by the user is stored in the string variable 'str' to be given as the third argument. The fourth argument 'n' specifies the maximum number of characters that can be input. The input terminates when either the user types ENTER key or inputs (n-1) characters. The resultant string is null-terminated. The function returns the number of characters stored in 'str'.

Bool kiteIsHit( );

This function, when executed, checks if the user has either clicked left mouse button or pressed a key. If any of these events has occurred, it returns true; otherwise, it returns false. Note that the function does not wait for the user.

Bool kiteIsHit(POINT& p1, char& ch);

This function, like its parameterless overloaded version, checks if the user has either clicked left mouse button or pressed a key. If any of these events has

occurred, it returns true; otherwise, it returns false. More over, it stores the user input information in the argument variables. If user has pressed a key on the keyboard, then its ASCII code is stored in the argument character variable. The character input is not echoed on the window. The POINT argument variable is set to point (0, 0) in this case. On the other hand, if the user clicks mouse instead of using keyboard, then coordinates of the point where the mouse was clicked, are stored in argument *p1*. and NULL character '\0' is stored in the argument *ch*.
Note that the function does not wait for the user.

int strpixlen(char *str);

This function returns the length in pixels of the band that the argument string would occupy, when printed with *winPrint( )* function.

int getWinWidth( );

This function returns the width of the client area of the window object, in pixels.

int getWinHeight( );

This function returns the height of the client area of the window object, in pixels.

void point(int x, int y, COLORREF c);

This function draws a point (i.e. puts ON the pixel), at pixel position (x, y) given as first two arguments, in the color given as third argument.

void line(int x1, int y1, int x2, int y2, COLORREF c = RGB(0, 0, 0),

int penWidth = 0, int penStyle = PS_SOLID);

This function draws a line segment between points (x1, y1) and (x2, y2), the coordinates of which are given as the first four arguments. The line is drawn in the color given in the fifth argument (default, BLACK). You can specify the width of the line in pixels, in the sixth argument. The default value zero, means one pixel wide line will be drawn. You can also specify the line style in the seventh argument. The following macros can be used to set the line style-

PS_SOLID to draw usual solid line (default case).

PS_DASH -- -- -- -- -- -- -- -- --

PS_DOT . . . .

PS_DASHDOT -- . -- . -- . -- . -- . --

PS_DASHDOTDOT -- .. -- .. -- .. -- .. -- ..

Note that line width greater than one pixel, is applicable only to PS_SOLID. If you set the line width to 3 and line style to PS_DASH, then solid line with 3 pixels width will be drawn and not a dashed line.
(Till the writing of this manual, the *penWidth* and *penStyle* parameters are ineffective.)

Go to the *Contents* page...

void circle(int xc, int yc, int r, COLORREF c = RGB(0, 0, 0));

This function draws a circle at the center specified by the first two arguments and with the radius specified by the third argument. The color of the circle circumference would be specified in the fourth argument. If it is not specified, the default color is black.

void ellipse(int xc, int yc, int xr, int yr, COLORREF c = RGB(0, 0, 0));

This function draws an ellipse at the center specified by the first two arguments. The horizontal radius is specified by the third argument, while the vertical radius is specified by the fourth argument. If the vertical and horizontal radii are equal, then a circle would be drawn. The fifth argument specifies the color of the ellipse.

void polygon(POINT *apt, int n, COLORREF c = RGB(0, 0, 0));

This function produces a closed polygon. The vertices of the polygon are specified by the POINT array in the first argument. The second argument specifies the number of points to be considered out of this array. The last arguments specifies the color of the polygon.

void arc(int xc, int yc, int xr, int yr, int start_ang, int included_ang,

COLORREF c = RGB(0, 0, 0));

This function draws a part of an ellipse. The first four arguments have same work as the first four arguments of the 'ellipse' function. The fifth argument specifies the start angle and the sixth argument specifies the included angle of the arc. These angles are measured from positive x-axis in counter-clockwise direction and are to be specified in degrees. The last argument specifies the color of the arc.

void filledRectang(int xLeft, int yTop, int width, int height,

COLORREF c = RGB(255, 255, 255), int pattern = -1);

This function produces a solid or filled rectangle. The color of the filling is given by the fifth argument. The first two arguments specify the coordinates of the upper left corner of the rectangle. The third and the fourth arguments specify the width and the height of the rectangle. By default, the rectangle will be filled with solid color; but if you specify the last (sixth) argument as one of the following possible macros, then the rectangle will be hatched, instead of solid-filled.

HS_HORIZONTAL, HS_BDIAGONAL, HS_VERTICAL, HS_FDIAGONAL, HS_CROSS, HS_DIAGCROSS.

See for yourself, how these various patterns are, by using them. (Note: Till the writing of this manual, the parmeter *pattern* is ineffective.)

void filledEllipse(int xc, int yc, int xr, int yr,

COLORREF c = RGB(255, 255, 255), int pattern = -1);

This function produces a solid or filled-up ellipse. The first five arguments have same meaning as those of the 'ellipse' function. The last argument specifies the fill-pattern as in the 'filledRectang' function. (Note: Till the writing of this manual, the parmeter *pattern* is ineffective.)

void filledPolygon(POINT *apt, int n,

COLORREF c = RGB(255, 255, 255), int pattern = -1);

This function produces a filled-up polygon. The vertices of the polygon are specified by the POINT array in the first argument. The second argument specifies the number of points to be considered out of this array. The last two arguments are same as the last two arguments of the 'filledRectang' function.
(Note: Till the writing of this manual, the parmeter *pattern* is ineffective.)

void close( );

This function closes and destroys the window object.

Go to the *Contents* page...