Chapter 1

Libwindow - a small X11-library

This is a short and probably incomplete description of Libwindow, a small library which aims to present an easy `C++` interface to the X11 window system. If you want to use the library in your program, do not forget the following include directive.

#include <*IFM/window*>

1.1 The Window Class (*ifm::Wstream*)

Definition

The class *ifm::Wstream* represents a window that can be used for IO-operations and an associated graphic context. The graphic context defines the drawing color (default: black), line width (default: 1) and drawing mode (default: *GXcopy*) and affects any output operation. There are a number of member functions to change the state of the graphic context. With the window comes a Cartesian coordinate system where the origin sits in the lower left corner, i.e., the window resides in the all-positive quadrant. All output is internally buffered, so in order to make it visible on the display, this buffer has to be flushed. Many functions handle *expose-events*, that means if parts of the window get obscured, their content is redrawn automatically, as soon as they get visible (exposed) again.

Creation

ifm::Wstream w(string str = "ifm ::Wstream");

Creates a window with dimensions (512 x 512) and name *str* and positions it with upper left corner (100, 100) on the X display (your screen).

ifm::Wstream *w(int xsize, int ysize, string str = "ifm ::Wstream");*

Precondition: $10 \leq xsize$, $ysize \leq 2048$.
Creates a window with dimensions (xsize x ysize) and name *str* and positions it with upper left corner (100, 100) on the X display (your screen).

ifm::Wstream *w(int xpos, int ypos, int xsize, int ysize, string str);*

Precondition: $10 \leq xsize$, $ysize \leq 2048$.
Creates a window with dimensions (*xsize* x *ysize*) and name *str* and positions it with upper left corner (*xpos*, *ypos*) on the X display (your screen).

ifm::Wstream *w(ifm::Wstream);*

copy constructor.

ifm::Wstream&

w = ifm::Wstream

copy assignment.

Operations

int	*w.xmin()*	returns minimal x-xoordinate in *w*.
int	*w.xmax()*	returns maximal x-xoordinate in *w*.
int	*w.ymin()*	returns minimal y-xoordinate in *w*.
int	*w.ymax()*	returns maximal y-xoordinate in *w*.

ifm::Wstream&

& *w << ifm::Drawable d*

d is drawn into *w*.

ifm::Wstream&

& *w >> ifm::Getable d*

d is set from *w*.

ifm::Wstream&

w.flush() Buffer is flushed and all output drawn onto the display. Returns *w*.

ifm::Wstream&

w.endl() same as *flush*.

ifm::Wstream&

w.sync() Buffer is flushed, all output is drawn onto the display and all pending X-requests have been processed. Returns *w*.

ifm::Wstream&

w.clear() — Clears the window and flushes the buffer. Returns *w*.

ifm::Wstream&

w.wait(unsigned long microsec)

Flushes buffer and waits for *microsec* microseconds.

bool *w.check_key()*

Returns true, iff there is a KeyRelease event pending.

bool *w.check_mouse()*

Returns true, iff there is a MouseMotion event pending.

bool *w.check_mouse_click()*

Returns true, iff there is a ButtonRelease event pending.

int *w.get_key()* — Flushes buffer, waits for a KeyRelease event and returns the pressed key's ASCII-code. (65 $\Leftrightarrow$ A, 97 $\Leftrightarrow$ a). Expose events during the waiting period are handled.

void *w.get_mouse(int& x, int& y)*

Flushes buffer, waits for a MouseMotion event and sets (x, y) to the mouse position. Expose events during the waiting period are handled.

int *w.get_mouse_click(int& x, int& y)*

Flushes buffer, waits for a ButtonRelease event, sets (x, y) to the mouse position and returns the number of the pressed mouse button. (1 $\Leftrightarrow$ left, 2 $\Leftrightarrow$ middle, 3 $\Leftrightarrow$ right). Expose events during the waiting period are handled.

void *w.wait_for_mouse_click(int button = 0)*

Precondition: $0 \leq button \leq 3$.
Flushes buffer and waits until specified (0 $\Leftrightarrow$ any) mouse button gets released. Expose events during the waiting period are handled.

ifm::Wstream&

w.set_draw_mode(int m)

Drawing mode is set to *m*. (Possible values include *GXcopy*, *GXxor*, *GXand*, ...) Returns *w*.

ifm::Wstream& *w.set_line_width(int w)*

Precondition: w > 0.
Drawing line width is set to *w*. Returns *w*.

int *w.number_of_colors()*

Returns the number of available colors.

ifm::Wstream& *w.set_color(int c)*

Precondition: $0 \leq$ c $\leq$ number_of_colors().
Drawing color is set to *c*. The last two colors are always *black* and *white*, i.e., *set_color(number_of_colors())* selects *black*. The rest is evenly divided by interpolating the following colors in order: *red*, *orange*, *yellow*, *green*, *blue*, *magenta* and *purple*. Returns *w*.

Example

The following code reads in a Circle *c*, draws *c* and its bounding square and then tracks the mouse pointer by drawing line segments between consecutive positions until finally a mouse button is pressed.

```
#include <IFM/window>

int main()
{
    // define a 200 x 200 pixel window
    ifm::Wstream w(200, 200, "LibWindow-Example");

    // read in a circle
    ifm::Circle c;
    w >> c;

    // print c and its bounding square
    w << ifm::yellow << c << ifm::red
      << ifm::Rectangle(c.x() - c.r(), c.y() - c.r(),
                        c.x() + c.r(), c.y() + c.r())
      << ifm::flush;

    // tracks mouse pointer
    ifm::Point p_last(c.x(), c.y());
    do {
        int x, y;
        w.get_mouse(x, y);
        w << ifm::blue << ifm::Line(p_last.x(), p_last.y(), x, y)
          << ifm::flush;
```

```
        p_last = ifm::Point(x, y);
    } while (!w.check_mouse_click());

    return 0;
}
```

1.2 A default Window

ifm::Wstream   ifm::wio;

ifm::wio can be used whenever one default *ifm::Wstream* suffices. It is a so called *proxy*, i.e., the corresponding X-window and graphic context are created, when *ifm::wio* is used the first time. Consequently, if *ifm::wio* is not used anywhere, this creation does not happen and hence no window appears.

1.3 What can be drawn (*ifm::Drawable*)

Here is a list of classes/objects that can be drawn into an *ifm::Wstream* using the operator <<.

1.3.1 Points (*ifm::Point*)

ifm::Point   p(int x, int y);

Creates a point with Cartesian coordinates (x, y).

int	*p.x()*	Returns x-coordinate of p.
int	*p.y()*	Returns y-coordinate of p.

Example

The following code draws a point at coordinate (100, 100) and waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::Point(100, 100) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.2 Line Segments (*ifm::Line*)

ifm::Line l(int x1, int y1, int x2, int y2);

Creates a line from (*x1*, *y1*) to (*x2*, *y2*).

int	*l.x1()*	Returns x-coordinate of the first endpoint.
int	*l.y1()*	Returns y-coordinate of the first endpoint.
int	*l.x2()*	Returns x-coordinate of the second endpoint.
int	*l.y2()*	Returns y-coordinate of the second endpoint.

Example

The following code draws a line segment from (100, 100) to (200, 200) and waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::Line(100, 100, 200, 200) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

Notes

Line Segments are somewhat special in X, since they do not include their endpoints, but only the grid points in between. So if you write

```
    ifm::wio << ifm::Line(100, 100, 200, 100)
             << ifm::Line(201, 100, 300, 100)
             << ifm::flush;
```

there does not appear a continuous line segment (100, 100) $\longrightarrow$ (300, 100), it will have a one-pixel gap in the middle.

1.3.3 Rectangles (*ifm::Rectangle*)

ifm::Rectangle r(int x1, int y1, int x2, int y2);

Creates a rectangle with diagonal (*x1*, *y1*) $\longrightarrow$ (*x2*, *y2*).

int	*r.x1()*	Returns x-coordinate of the diagonal's first endpoint.
int	*r.y1()*	Returns y-coordinate of the diagonal's first endpoint.

int	*r.x2()*	Returns x-coordinate of the diagonal's second endpoint.
int	*r.y2()*	Returns y-coordinate of the diagonal's second endpoint.

Example

The following code draws a rectangle with lower left corner (100, 100) and upper right corner (200, 200) and waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::Rectangle(100, 100, 200, 200) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.4 Filled Rectangles (*ifm::FilledRectangle*)

ifm::FilledRectangle r(int x1, int y1, int x2, int y2);

Creates a filled rectangle with diagonal ($x1$, $y1$) $\longrightarrow$ ($x2$, $y2$).

int	*r.x1()*	Returns x-coordinate of the diagonal's first endpoint.
int	*r.y1()*	Returns y-coordinate of the diagonal's first endpoint.
int	*r.x2()*	Returns x-coordinate of the diagonal's second endpoint.
int	*r.y2()*	Returns y-coordinate of the diagonal's second endpoint.

Example

The following code draws a filled rectangle with lower left corner (100, 100) and upper right corner (200, 200) and waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::FilledRectangle(100, 100, 200, 200)
             << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.5 Circles (*ifm::Circle*)

ifm::Circle c(int x, int y, int r);

Creates a circle with center (x, y) and radius r.

int	*c.x()*	Returns center's x-coordinate.
int	*c.y()*	Returns center's y-coordinate.
int	*c.r()*	Returns radius of c.

Example

The following code draws a circle with center (100, 100) and radius 20. Then it waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::Circle(100, 100, 20) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.6 Filled Circles (*ifm::FilledCircle*)

ifm::FilledCircle c(int x, int y, int r);

Creates a filled circle with center (x, y) and radius r.

int	*c.x()*	Returns center's x-coordinate.
int	*c.y()*	Returns center's y-coordinate.
int	*c.r()*	Returns radius of c.

Example

The following code draws a filled circle with center (100, 100) and radius 20. Then it waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::FilledCircle(100, 100, 20) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.7 Ellipses (*ifm::Ellipse*)

ifm::Ellipse e(int x, int y, int w, int h);

Creates an ellipse with center (x, y), width $2 \cdot w$ and height $2 \cdot h$.

int	*e.x()*	Returns center's x-coordinate.
int	*e.y()*	Returns center's y-coordinate.
int	*e.w()*	Returns half the width of e.
int	*e.h()*	Returns half the height of e.

Example

The following code draws an ellipse with center (100, 100), width 80 and height 50. Then it waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::Ellipse(100, 100, 80, 50) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.8 Filled Ellipses (*ifm::FilledEllipse*)

ifm::FilledEllipse e(int x, int y, int w, int h);

Creates a filled ellipse with center (x, y), width $2 \cdot w$ and height $2 \cdot h$.

int	*e.x()*	Returns center's x-coordinate.
int	*e.y()*	Returns center's y-coordinate.
int	*e.w()*	Returns half the width of e.
int	*e.h()*	Returns half the height of e.

Example

The following code draws a filled ellipse with center (100, 100), width 80 and height 50. Then it waits for a mouseclick to finish.

```
#include <IFM/window>

int main()
{
    ifm::wio << ifm::FilledEllipse(100, 100, 80, 50) << ifm::flush;
    ifm::wio.wait_for_mouse_click();
    return 0;
}
```

1.3.9 Manipulators

The manipulators listed here correspond to the respective member functions of *ifm::Wstream*.
The functionality can be looked up there.

ifm::Wstream& *ifm::Wstream*& *w* << *ifm::flush*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::endl*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::sync*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::clear*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::wait(unsigned long)*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::line_width(int)*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::draw_mode(int)*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::color(int)*

Shortcuts for Drawing Modes

ifm::Wstream& *ifm::Wstream*& *w* << *ifm::copy_mode*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::xor_mode_mode*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::or_mode*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::and_mode*

Shortcuts for Colors

ifm::Wstream& *ifm::Wstream*& *w* << *ifm::white*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::black*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::red*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::orange*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::yellow*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::green*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::lightgreen*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::blue*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::magenta*
ifm::Wstream& *ifm::Wstream*& *w* << *ifm::purple*

1.4 What can be read (*ifm::Getable*)

Here is a list of classes/objects that can be read from an *ifm::Wstream* using the operator >>.

ifm::Point, *ifm::Line*, *ifm::Rectangle*, *ifm::FilledRectangle*, *ifm::Circle*, *ifm::Ellipse*, *ifm::FilledCircle*, and *ifm::FilledEllipse*.

Chapter 2

Libturtle - Turtle Graphics

This is a short description of Libturtle, a small turtle graphic library based on Libwindow. If you want to use the library in your program, do not forget the following include directive.

#include <IFM/turtle>

Imagine a turtle walking on the Euclidean plane and leaving a trace behind it. At any time, the turtle has a certain position and a view direction. Initially, it is looking to the right. You can influence these parameters using the following functions.

void *ifm::forward(unsigned int i = 1)*

Move the turtle i steps in view direction.

void *ifm::left(int d = 1)*

Turn the turtle left by an angle of d degree.

void *ifm::right(int d = 1)*

Turn the turtle right by an angle of d degree.

When the drawing is complete, i.e., at the end of the program, it is shown in a window of 512×512 pixels, appropriately scaled to use the space available. A mouseclick then destroys the window and ends the program.

Example

The following code draws a regular pentagon. Then it waits for a mouseclick to finish.

```
#include <IFM/turtle>

int main()
```

```
{
    for (unsigned int i = 0; i < 5; ++i) {
        ifm::forward();
        ifm::left(72);
    }
    return 0;
}
```