LABORATORY 13

Inheritance

Objective

In this week's lab you learn to create new objects using inheritance. Inheritance supports both abstraction and code reuse—two valuable programming techniques.

Key Concepts

- Inheritance
- Is-a relationship
- Code reuse
- Abstraction
- Base class
- Derived class
- Derived-class declaration
- Derived-class implementation

13.1 GETTING STARTED

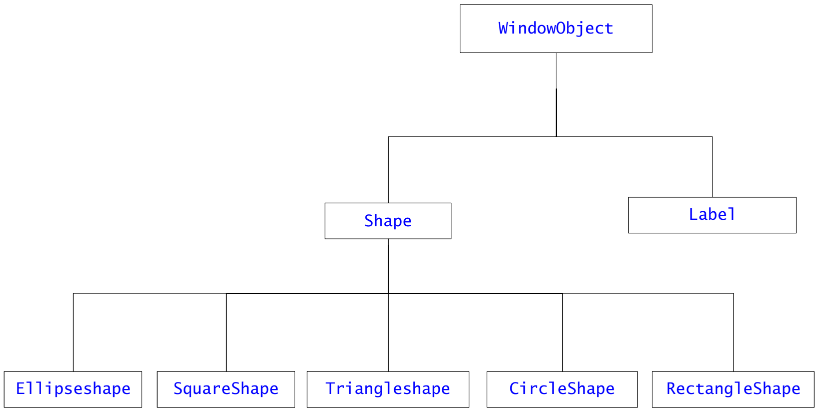
- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab13.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

13.2 INHERITANCE

A key feature of an object-oriented language is inheritance. *Inheritance* is the ability to define new classes using existing classes as a basis. Inheritance supports both abstraction and code reuse. For example, suppose you were going to develop classes for different kinds of bicycles. You might develop classes to represent mountain bicycles and road-racing bicycles. It may pay off to introduce a base class that contains the features common to all types of bicycles and to then use the base class to create specialized types of bicycles. Furthermore, this type of hierarchical abstraction of common features helps you understand the classes. For example, even if you do not know exactly what a hybrid bicycle is, because it is a bicycle, you know it must have two wheels, handlebars, and pedals.

The relationship between a hybrid bicycle and a generic bicycle is known as an *is-a* relationship—a hybrid bicycle is a bicycle. The relationship between a bicycle and its wheels is a *has-a* relationship—a bicycle has wheels.

An inheritance hierarchy is often presented pictorially. For example, the inheritance hierarchy of the EzWindows shapes is



We see that the base class is `WindowObject` and that the classes `Shape` and `Label` are derived from `WindowObject`. Thus a `Shape` is a `WindowObject`, and a `Label` is a `WindowObject`.

Designing an inheritance hierarchy is one of the keys to a good object-oriented design. Developing a flexible hierarchy is quite difficult, but in the long run, it can pay off by reducing both maintenance costs and future development costs.

- To get a feel for developing a hierarchy of objects based on the is-a relationship, develop an inheritance hierarchy for telephones. For each

✎ class of telephone in your hierarchy, give the attributes and behaviors of the class. ✓

13.3 THE MECHANICS OF INHERITANCE

- Open the file `rect.h`. Examine the `RectangleShape` class declaration. Answer the following questions.
  - — What is the base class for `RectangleShape`?

— Suppose a user defines a `RectangleShape` object name `R`. Determine all the messages that a client user can send `R`. That is, what member functions of `R` can a client user invoke?

- Show your answers to a laboratory instructor. ✓
- Open the file `rect.cpp`.
- Examine the constructor for `RectangleShape`. What constructors in the shape inheritance hierarchy are called to instantiate a `RectangleShape`?
- To verify your answer, open the project file `exp1.dsw`. For each constructor in the shape inheritance hierarchy, add an insertion statement such as

  ```
  cout << "Constructor XXX called" << endl;
  ```

  where `XXX` is the name of the constructor. Hint: Use the IDE's cut-and-paste feature to add the insertion statement. Run the program and write down the order in which the constructors were called. ✓
- Close the project `exp1.dsw`.

13.4 SHADOW BOXING

A useful type of window object is a shadowed rectangle.

Shadowed rectangles have a three-dimensional look. We can create this new type of window object easily using inheritance.

We will call this new shape `ShadowedRectangleShape`. The class `ShadowedRectangleShape` will be derived from `RectangleShape`. This is saying that a `ShadowedRectangleShape` is a kind of `RectangleShape`. For this exercise, a `ShadowedRectangleShape` will be identical in all respects to a `RectangleShape` except that when a `ShadowedRectangleShape` is drawn a black shadow is included. The shadow is offset 0.25 centimeters to the right and below the main rectangle.

- Create the class declaration for `ShadowedRectangleShape` in the file `shadowrect.h`. You may find it helpful to look at `rect.h`. If you need help ask your laboratory instructor.

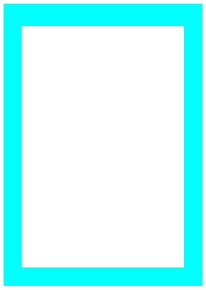
- Next implement `ShadowedRectangleShape`. Place the implementation of `ShadowedRectangleShape` in the file `shadowrect.cpp`. Again you will find it useful to use `rect.cpp` as a guide. Indeed, besides the constructors only the `Draw()` and `Erase()` member functions need to be implemented—all the other member functions are inherited.

- Open the project file `exp2.dsw`. Examine the program `exp2.cpp`. Demonstrate that your implementation of `ShadowedRectangleshape` ✎ works to the laboratory instructor. ✓

- Close the project `exp2.dsw`.

13.5 DON’T GET BOXED IN

Another handy type of window object is a box. As the following picture shows, a box is like a rectangle with no borders except that its middle is empty.



- Using the existing shape hierarchy and inheritance, it is easy to build a new box object. Design a simple box class. Your box class should be derived from `Shape`. (A box is not a kind of rectangle). In this `BoxShape` class, the walls of the box are always 0.3 centimeters thick. Before writing any code, answer the following questions.

— What member functions, if any, are specific to BoxShape?

— What data members, if any, are specific to BoxShape?

— Describe how you will draw a BoxShape.

✎ ▪ Show your answers to a laboratory instructor. ✓

▪ Open the project file exp3.dsw.

▪ Open the include file box.h. This file should be empty except for a comment. In this file, add your class declaration for BoxShape. If you are unsure how to declare the class, you may find it helpful to examine the declaration of RectangleShape in rect.h. Save the declaration of BoxShape in the file box.h.

✎ ▪ Explain your BoxShape class declaration to a laboratory instructor. ✓

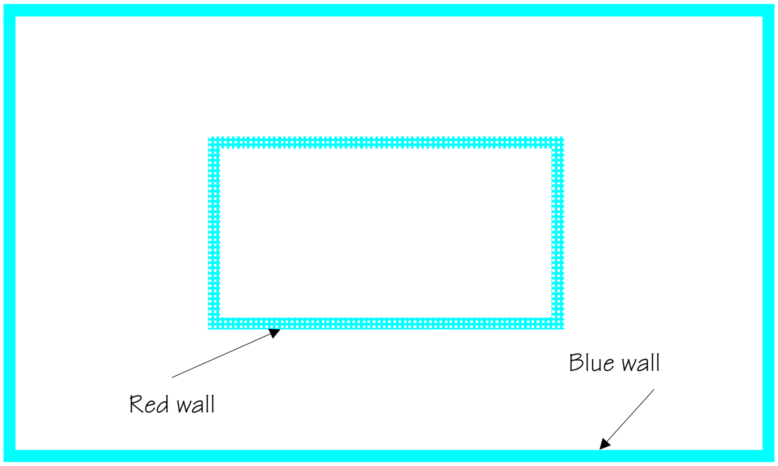
▪ Open the file box.cpp. Again, this file should be empty except for a comment. In this file, type in the implementation of the constructors and member functions for BoxShape. If you are unsure how to implement the BoxShape's constructor, you may find it helpful to examine the constructor for RectangleShape in rect.cpp.

▪ Save the implementation of BoxShape in the file box.cpp.

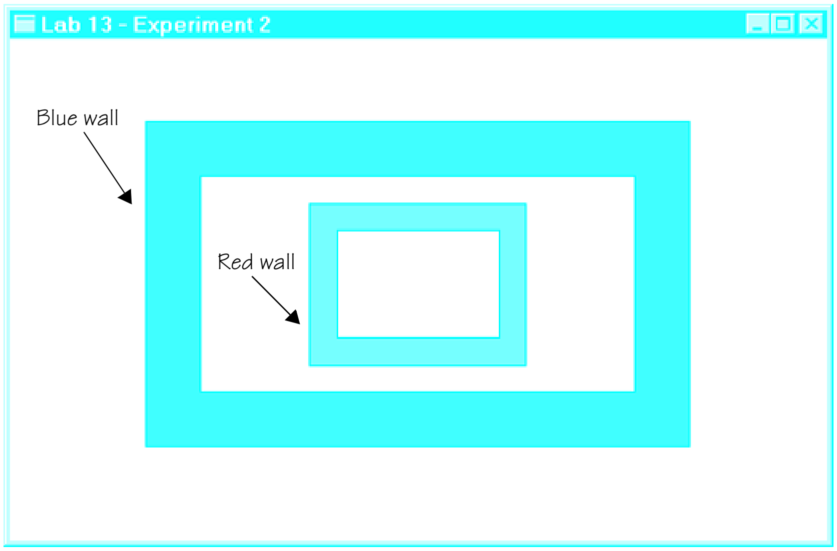
✎ ▪ Explain your implementation of BoxShape to a laboratory instructor. ✓

▪ Open the file exp3.cpp. This file contains a stub for the function ApiMain(). (A *stub* is a function where the body of the function has been omitted). A stub allows the program to be compiled, but the function does not do anything when called.

- To demonstrate the use of `BoxShape`, add code to the function `ApiMain()` that creates a diagram as shown below:



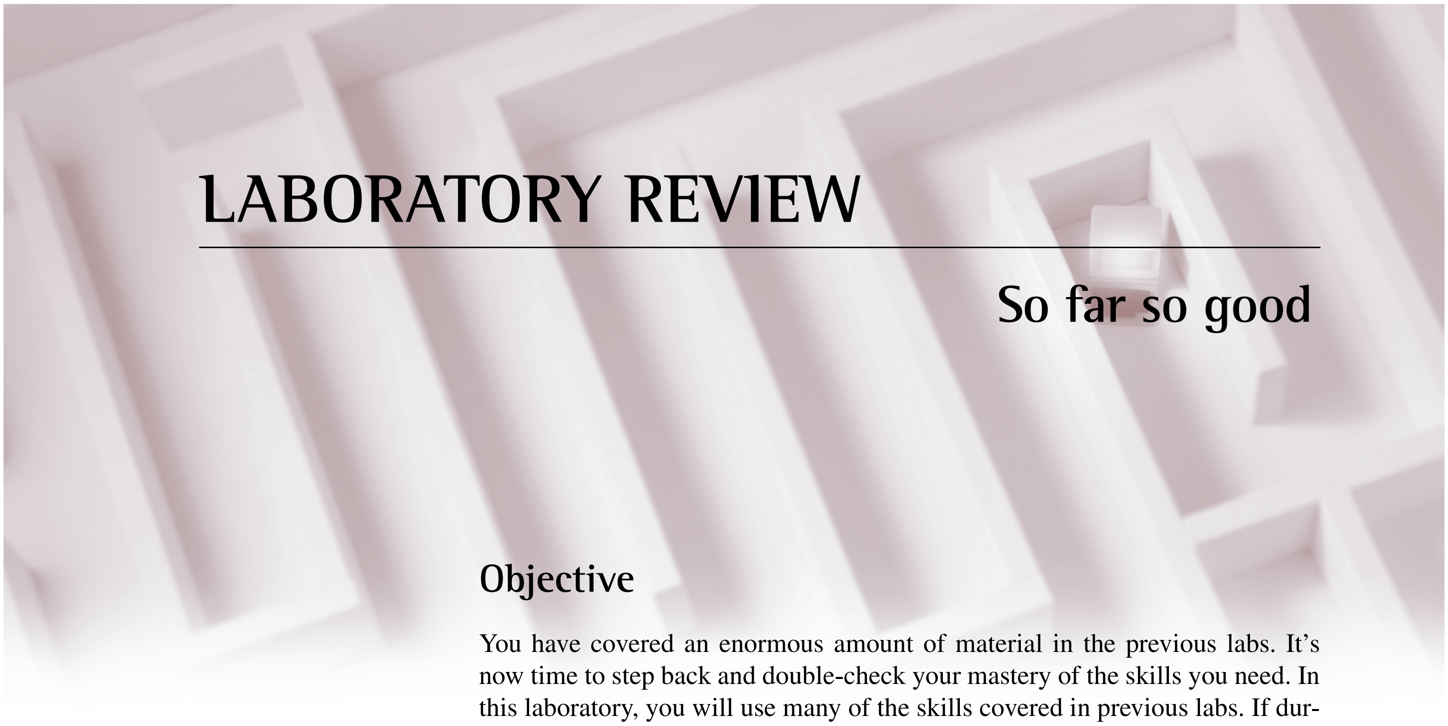
- ✎ Demonstrate your program to a laboratory instructor. ✓
- You can make `BoxShape` more flexible if you permit the user of a `BoxShape` to specify the thickness of the walls of the box when it is created. Modify `box.h` and `box.cpp` to include this extension.
- Demonstrate that your extension works correctly by modifying `exp2.cpp` so the walls of the outer box are 1 centimeter thick, and the walls of the inner box are 0.5 centimeters thick. The outer box is blue and is 10 centimeters by 6 centimeters. The inner box is red and is 4 centimeters by 3 centimeters. The window your program creates should look something like the following:



✎
- Demonstrate your program to the laboratory instructor. ✓
- Close the project `exp3.dsw`.

13.6 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.



LABORATORY REVIEW

So far so good

Objective

You have covered an enormous amount of material in the previous labs. It’s now time to step back and double-check your mastery of the skills you need. In this laboratory, you will use many of the skills covered in previous labs. If during any of the following activities, you get stuck or are unsure about how to do something, consult the laboratory instructor for help. Now is the time to correct any lingering uncertainties you might have about the C++ topics that have been covered thus far.

Key Concepts

- Arithmetic and assignment statements
- Conditional execution
- Iteration
- Reading files
- Using objects

R.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab08.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

R.2 SIMPLE ASSIGNMENT

- Write a program called `poly.cpp` that prompts for and extracts four floating-point values (`a`, `b`, `c`, and `x`) and then evaluates and displays the value of the following expression:

  $$ax^2 + bx + c$$

- Demonstrate your completed program to your laboratory instructor. ✓

R.3 CONDITIONAL EXECUTION

- Write a program called `date.cpp` that prompts for and extracts three integer values. We will refer to these values as `Month`, `Day`, and `Year`. As the names imply, these inputs represent a date. Your program should then output the date in the traditional written style. For example, if the inputs are:

  ```
  12 29 53
  ```

  your program should output

  ```
  December 29, 1953
  ```

  and exit. Your program need not check the validity of the year or day, but it should check the validity of the month. If the input month is incorrect, your program should output an error message. For example, if the inputs are:

  ```
  15 6 44
  ```

  your program should output

  ```
  Bad month: 15
  ```

- Demonstrate your completed program to the laboratory instructor. ✓

R.4 SIMPLE ITERATION

- Write a program called `line.cpp`. Function `main()` calls a function called `line()`. Function `line()` accepts two optional parameters `n` and `c`. The formal parameter `n` is an integer, and the formal parameter `c` is a **`char`**. The function `line()` outputs a line of characters. The line length is `n` characters. The character used to form the line is `c`. If function `line()` is called with no parameters, it outputs a line consisting of 10 asterisks. If function `line()` is called with a single parameter, it outputs `n` *'s.
- Demonstrate your completed program. Make sure your demonstration illustrates the various ways that function `line()` can be called. ✓

R.5 READING A FILE

- The file `pairs.dat` contains lines that consist of pairs of integers. Write a program called `math.cpp` that reads the file. For each pair of extracted numbers (call the input values `v1` and `v2`), your program should call `sumdiff()`. The function `sumdiff()` accepts the two numbers and computes the sum (`v1` + `v2`) and difference (`v1` - `v2`). The function passes these two computed values back to the calling function through two other parameters. Your function `main()` should print the two numbers and the sum and difference. The following line illustrates the output your program should produce.

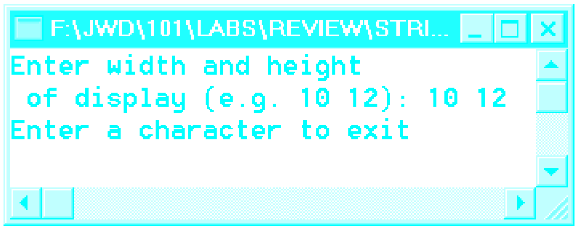
  ```
  v1 = 10; v2 = 5; v1 + v2 = 15; v1 - v2 = 5
  ```

  Your program should process all the value pairs in the file `pairs.dat`.

- When you have completed your program and verified that it works correctly, show your code to a laboratory instructor and demonstrate that it works. ✓

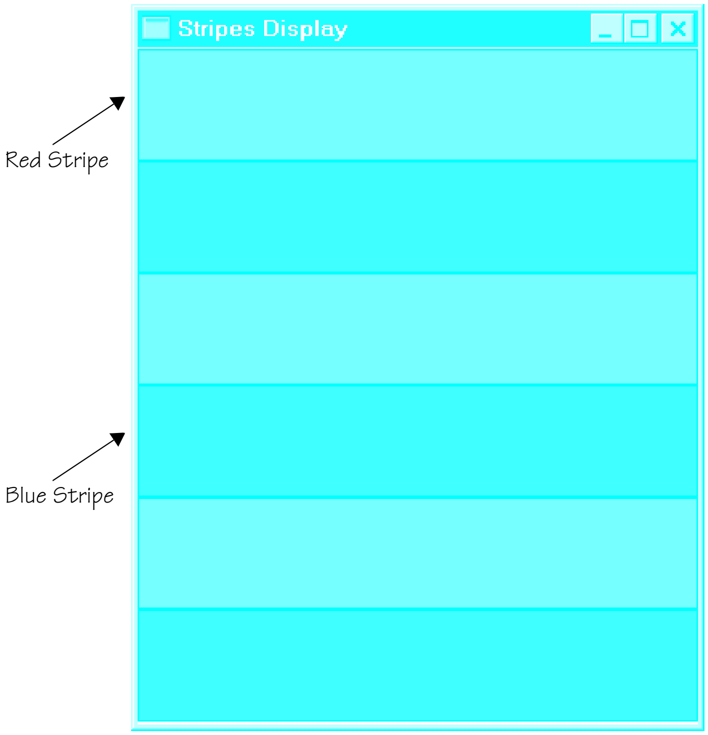
R.6 USING OBJECTS

- Write a program called `stripes.cpp`. For this program, you will need to set up a project file to use the EzWindows API. The program `stripes.cpp` in its function `ApiMain()` should prompt the user for the width and height of the display window to be created. In this display window (titled Stripes Display), your program should display six alternating red and blue stripes. The stripes should fill the display window. The following figures illustrate the behavior of the program.



```
Enter width and height
 of display (e.g. 10 12): 10 12
Enter a character to exit
```

The above input creates the following display window.



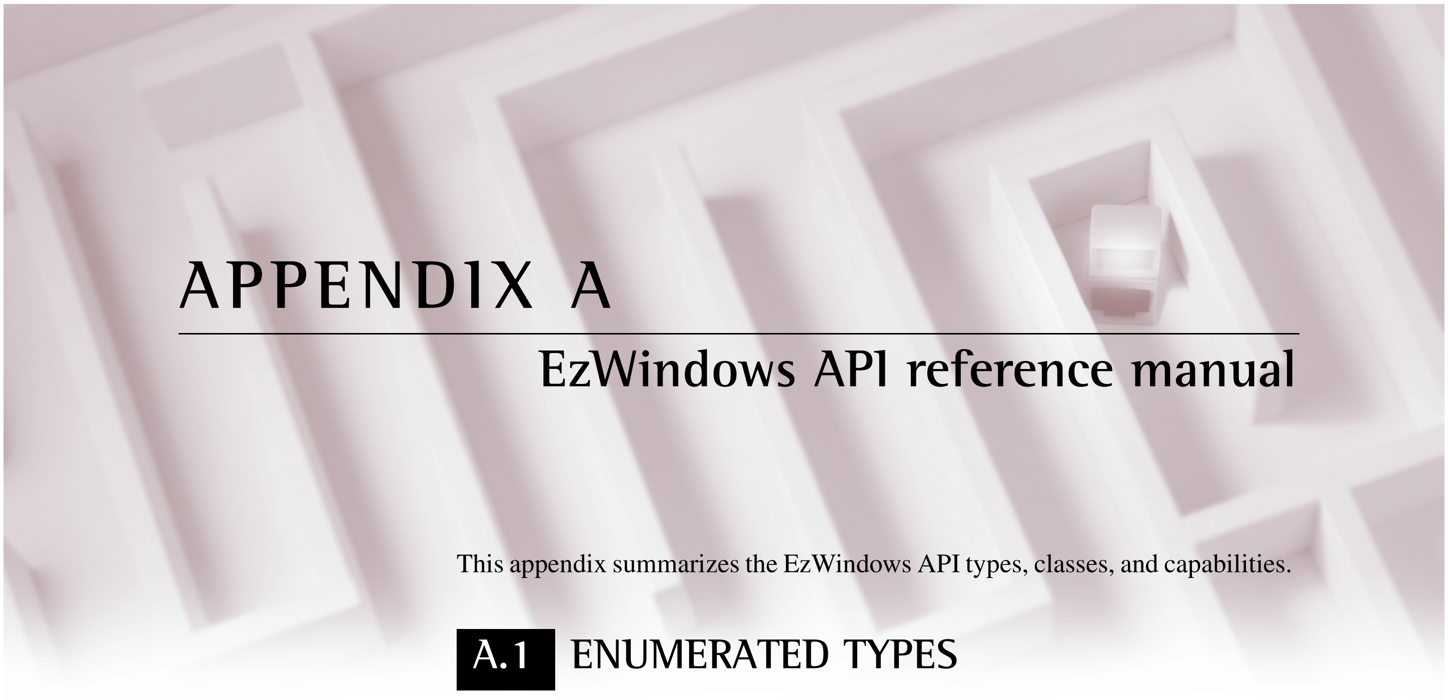
- Before attempting to solve this problem, sit down and sketch out on paper a preliminary design. Your preliminary design should include what objects you will need to create, where these objects will be created, and how you will draw the necessary rectangles. Your design sketch should contain details of how you will compute the coordinates of each rectangle. Discuss your design with your laboratory instructor before writing any code. ✓

- After completing your preliminary design, set up a project file to build the program. As a first step in completing your program, write the code necessary to prompt the user for the dimensions of the display window and to create the window. Do not worry about drawing the alternating stripes in the window at first. Demonstrate that your program creates an appropriately sized and titled window to your laboratory instructor. ✓

- Now complete the part of the program that draws the stripes in the window. If you are having any difficulty remembering how to create a `RectangleShape`, consult your textbook. If you are having trouble figuring out how to tackle the problem or you wish to discuss your proposed solution, consult the laboratory instructor. When you have completed the program, show your code to the laboratory instructor and demonstrate the program. ✓

R.7 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.



APPENDIX A

EzWindows API reference manual

This appendix summarizes the EzWindows API types, classes, and capabilities.

A.1 ENUMERATED TYPES

The EzWindows API defines three enumerated types: `color`, `WindowStatus`, and `BitMapStatus`.

Enumerated type `color` provides symbolic names for the possible colors that can be displayed in a `SimpleWindow`.

```
enum color { Black, White, Red, Green, Blue, Yellow,
 Cyan, Magenta};
```

Enumeration type `WindowStatus` defines the possible states for a `SimpleWindow` object

```
enum WindowStatus {WindowClosed, WindowOpen,
 WindowFailure};
```

where

- `WindowClosed` indicates an unopened window. Objects cannot be displayed in a window with this status.
- `WindowOpen` indicates an opened window. Objects can be displayed in a window with this status.
- `WindowFailure` indicates a failure state. Objects cannot be displayed in a window with this status.

Enumeration type `BitMapStatus` defines the possible states of a `BitMap` object

```
enum BitMapStatus {NoBitMap, BitMapOkay, NoWindow};
```

where

- `NoBitMap` indicates there is no bitmap to be displayed.
- `BitMapOkay` indicates there is a bitmap to display and an associated window.

- NoWindow indicates there is no associated window with the bitmap.

A.2 COORDINATE SYSTEM

Figure E.1 illustrates the EzWindows coordinate system. The origin is the upper-left corner of the screen. All coordinates are expressed as centimeters from the origin. The unit of measure for the size of EzWindows objects is also centimeters.

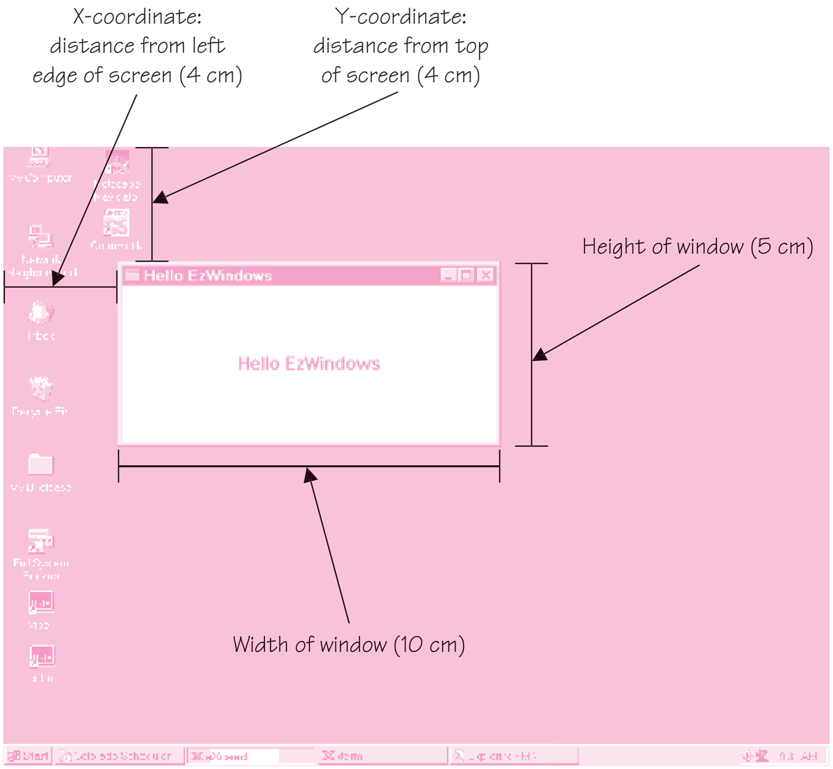
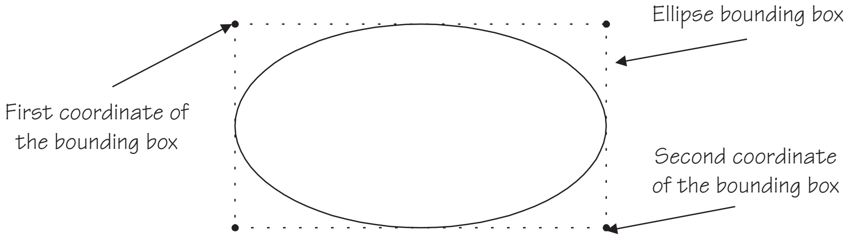


Figure A.1
The EzWindows coordinate system

Some EzWindows API functions use a bounding box to specify the size of an object. For example, the following diagram illustrates the bounding box for an ellipse.


Ellipse bounding box
First coordinate of the bounding box
Second coordinate of the bounding box

A bounding box is specified by giving the EzWindows coordinates of the upper-left and lower-right corners of a rectangle that bounds the shape.

A.3 CLASS POSITION

The class `Position` allows objects that represent the logical window coordinates of a window object to be defined and manipulated. The class provides two public constructors that are described below.

`Position::Position(`**`float`**` x = 0.0, `**`float`**` y = 0.0)`

Creates a `Position` object that associates the value of `x` with its x-coordinate and the value `of` y with its y-coordinate.

`Position::Add(`**`const`**` Position &p) `**`const`**

Creates a `Position` object that is a copy of `p`.

The `Position` class also provides two public members functions that are described below.

`int`` Position::GetXDistance() `**`const`**

Returns the x-coordinate of the position.

`int`` Position::GetYDistance() `**`const`**

Returns the y-coordinate of the position.

In addition, the + operator is overloaded to use `Position` objects as operands.

`Position `**`operator+`**`(`**`const`**` Position &a, `**`const`**` Position &b)`

Returns a position whose x-coordinate and y-coordinate are respectively the sum of `a`’s and `b`’s x-coordinates and `a`’s and `b`’s y-coordinates.

A.4 CLASS SIMPLEWINDOW

The class `SimpleWindow` allows objects that represent simple window displays to be defined and manipulated. The class provides several public constructors that are described below.

```
SimpleWindow::SimpleWindow(const char *WindowTitle
  = "Untitled", float Width = 8.0f,
  float Height = 8.0f,
  const Position &WindowPosn = Position(3.0f, 3.0f));
```

Creates a SimpleWindow for displaying graphical objects. Parameter WindowTitle is a pointer to the character string to be displayed in the title bar of the window. The default title is "Untitled". Parameter Width is the width of the window in centimeters. The default width is 8 centimeters. Parameter Height is the height of the window in centimeters. The default height is 8 centimeters. Parameter WindowPosition is the position of the window. The first coordinate is the distance in centimeters from the left edge of the screen. The second coordinate is the distance in centimeters from the top edge of the screen. The default position is (3.0, 3.0), which positions the upper-left corner of the window 3 centimeters from the left edge of the screen and 3 centimeters from the top edge of the screen.

```
SimpleWindow::SimpleWindow(const string &WindowTitle,
  float Width = 8.0f, float Height = 8.0f,
  const Position &WindowPosn = Position(3.0f, 3.0f));
```

Creates a SimpleWindow for displaying graphical objects. Parameter WindowTitle is a string to be displayed in the title bar of the window. Parameter Width is the width of the window in centimeters. The default width is 8 centimeters. Parameter Height is the height of the window in centimeters. The default height is 8 centimeters. Parameter WindowPosn is the position of the window. The first coordinate is the distance in centimeters from the left edge of the screen. The second coordinate is the distance in centimeters from the top edge of the screen. The default position is (3.0, 3.0), which positions the upper-left corner of the window 3 centimeters from the left edge of the screen and 3 centimeters from the top edge of the screen.

The SimpleWindow class also provides several public members functions that are described below.

```
WindowStatus SimpleWindow::Close();
```

Closes the window and makes it disappear. The return value is WindowClosed.

```
void SimpleWindow::Erase(const Position &UpperLeft,
  float Width, float Height);
```

Erases a rectangular region. The upper-left corner of the rectangle is specified by the Position UpperLeft. A rectangle Width centimeters wide and Height centimeters high is erased.

```
Position SimpleWindow::GetCenter() const;
```

Gets the location of the center of the window. The function returns a Position value that represents the logical coordinates of the center of the window, which are measured in centimeters from the left and top edges of the window.

```
float SimpleWindow::GetHeight() const;
```

Returns the height of the window in centimeters.

```
WindowStatus SimpleWindow::GetStatus() const;
```

Returns a `WindowStatus` value that represents the state of the window.

```
float SimpleWindow::GetWidth() const;
```

Returns the width of the window in centimeters.

```
float SimpleWindow::GetXPosition() const;
```

Returns the x-coordinate of the position of the window.

```
float SimpleWindow::GetYPosition() const;
```

Returns the y-coordinate of the position of the window.

```
void SimpleWindow::Message(
  const string &Msg = “Message”);
```

Pops up an alert window with a message. The parameter `Msg` is the character string to display in the alert window.

```
WindowStatus SimpleWindow::Open();
```

Makes window appear on the display and be enabled for displaying objects. The function returns a `WindowStatus` value that represents the state of the window.

```
void SimpleWindow::RenderEllipse(
  const Position &UpperLeft,
  const Position &LowerRight, const color &c,
  const bool Border = false);
```

Draws an ellipse. The bounding box is specified by the parameters `UpperLeft` and `LowerRight`. The ellipse is filled with color `c`. If `Border` is false, draw the ellipse without a border; otherwise, draw it with a black border.

```
void SimpleWindow::RenderPolygon(
  const vector<Position> &PolyPoints, int NPoints,
  const color &c, const bool Border = false);
```

Draws a closed polygon. The points of the polygon are held in the vector `PolyPoints`. The parameter `NPoints` is the number of points in the polygon. The polygon is filled with color `c`. If `Border` is false, draw the polygon without a border; otherwise, draw it with a black border.

```
void SimpleWindow::RenderPolygon(
  const Position PolyPoints[], int NPoints,
  const color &c, const bool Border = false););
```

Draws a closed polygon. The points of the polygon are held in the array `PolyPoints`. The parameter `NPoints` is the number of points in the polygon. The polygon is filled with color `c`. If `Border` is false, draw the polygon without a border; otherwise, draw it with a black border.

```
void SimpleWindow::RenderRectangle(
  const Position &UpperLeft,
  const Position &LowerRight, const color &c,
  const bool Border = false);
```

Draws a rectangle. The bounding box is specified by the coordinates `UpperLeft` and `LowerRight`. The rectangle is filled with color `c`. If

Border is false, draw the rectangle without a border; otherwise, draw it with a black border.

```
void SimpleWindow::RenderText(
  const Position &UpperLeft,
  const Position &LowerRight,
  const string &Msg = "Message",
  const color &TextColor = Black,
  const color &BackGroundColor = White);
```

Displays a text string in a window. Parameter UpperLeft is the position of the upper-left corner of the bounding box for the text message. Parameter LowerRight is the position of the lower-right corner of the bounding box for the text message. Parameter Msg is the string to be displayed in the window. The default message is "Message". Parameter TextColor is the color of the text message. The default text color is black. Parameter BackGroundColor is the background color for the text. The default background color is white.

```
void SimpleWindow::RenderText(
  const Position &UpperLeft,
  const Position &LowerRight,
  const char *Msg = "Message",
  const color &TextColor = Black,
  const color &BackGroundColor = White);
```

Displays a text string in a window. Parameter UpperLeft is the position of the upper-left corner of the bounding box for the text message. Parameter LowerRight is the position of the lower-right corner of the bounding box for the text message. Parameter Msg is a pointer to a character string to be displayed in the window. The default message is "Message". Parameter TextColor is the color of the text message. The default text color is black. Parameter BackGroundColor is the background color for the text. The default background color is white.

```
void SimpleWindow::SetMouseClickCallback(
  MouseClickCallbackFunction f);
```

Registers a callback for a mouse click. Function f() will be called when a mouse click occurs in the window. Function f() must be declared to take a single parameter of type **const** Position &, and it must return an **int**. The return value of f() indicates whether the event was handled successfully. A value of 1 indicates success, and a value of 0 indicates that an error occurred.

```
void SimpleWindow::SetRefreshCallback(
  RefreshCallbackFunction f);
```

Registers a callback for a refresh message. Function f() is called when the window receives a refresh event. The function f() must be declared to take no parameters, and it must return an **int**. The return value of f() indicates whether the event was handled successfully. A value of 1 indicates success, and a value of 0 indicates that an error occurred.

```
void SimpleWindow::SetQuitCallback(
  QuitCallbackFunction f);
```

Registers a callback for a quit message. Function f() is called when the window receives a quit event. The function f() must be declared to take no parameters, and it must return an **int**. The return value of f() indicates whether the event was handled successfully. A value of 1 indicates success, and a value of 0 indicates that an error occurred.

```
bool SimpleWindow::StartTimer(int Interval);
```

Starts timer running. Parameter Interval is the number of milliseconds between timer events. The return value indicates whether the timer was successfully started. A return value of true indicates success, and a return value of false indicates that the timer could not be set up.

```
void SimpleWindow::StopTimer();
```

Turns off the timer.

```
void SimpleWindow::SetTimerCallback(
  TimerTickCallbackFunction f);
```

Registers a callback for a timer tick. Function f() will be called when a timer tick occurs. The function f() must be declared to take no parameters, and it should return an **int**. The return value of f() indicates whether the event was handled successfully. A value of 1 indicates success, and a value of 0 indicates that an error occurred.

A.5 CLASS WINDOWOBJECT

Class WindowObject is the base class for class Shape. The class provides one public constructor.

```
WindowObject::WindowObject(SimpleWindow &w,
  const Position &p);
```

Creates a WindowObject that is centered at position p in window w.

The WindowObject class also provides several public members functions that are described below.

```
Position WindowObject::GetPosition() const;
```

Returns the position of the window object.

```
void WindowObject::GetPosition(float &XCoord,
  float &YCoord) const;
```

Returns the position of the window object. The x-coordinate is returned in XCoord, and the y-coordinate is returned in YCoord.

```
SimpleWindow& WindowObject::GetWindow() const;
```

Returns the window containing the WindowObject.

```
void WindowObject::SetPosition(const Position &p);
```

Sets the position of the WindowObject to p.

```
void WindowObject::SetPosition(float XCoord,
  float Ycoord);
```

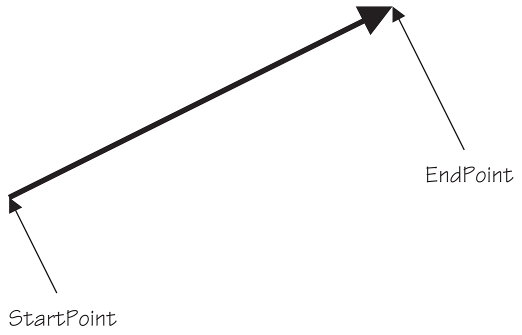
Sets the coordinates of the WindowObject to Position(XCoord, YCoord).

A.6 CLASS RAYSEGMENT

Class RaySegment represents rays in the SimpleWindow graphics system. Class RaySegment is derived from class WindowObject. A ray has a starting point that is a Position. This data member is inherited from WindowObject. Figure A.2 shows a RaySegment.

Figure A.2

An EzWindows RaySegment



The RaySegment class provides two public constructors that are described below.

```
RaySegment::RaySegment(SimpleWindow &w,
  const Position &StartPoint, const Position &EndPoint,
  const color &c = Black, float Thickness = 0.1f,
  bool Arrowhead = false);
```

Creates a RaySegment object to represent a ray. The ray is contained in SimpleWindow w. Its starting position is StartPoint, and its ending position is EndPoint. The ray has color c, which defaults to black. The ray is Thickness centimeters thick. The default thickness is 0.1 centimeters. If Arrowhead is true, the ray is drawn with an arrowhead at its ending point; otherwise, the ray has no arrowhead. The default is no arrowhead.

```
RaySegment::RaySegment(SimpleWindow &w, float StartX,
  float StartY, float EndX, float EndY,
  const color &c = Black, float Thickness = 0.1f,
  bool Arrowhead = false);
```

Creates a RaySegment object to represent a ray. The ray is contained in SimpleWindow w. Its starting point is Position(StartX, StartY), and its ending point is Position(EndX, EndY). The ray has color c, which defaults to black. The ray is Thickness centimeters thick. The default thickness is 0.1 centimeters. If Arrowhead is true, the ray is drawn with an arrowhead at its ending point; otherwise, the ray has no arrowhead. The default is no arrowhead.

The RaySegment class provides several public member functions that are described below.

```
void RaySegment::ClearArrowhead();
```

Sets the ray to be drawn without an arrowhead.

```
void RaySegment::Draw();
```

Draws the ray in its associated window.

```
void RaySegment::Erase();
```

Erases the ray.

```
color RaySegment::GetColor() const;
```

Returns the color of the ray.

```
Position RaySegment::GetEndPoint() const;
```

Returns the ending point of the ray.

```
void RaySegment::GetEndPoint(float &x, float &y) const;
```

Returns the ending point of the ray in x and y.

```
float RaySegment::GetLength() const;
```

Returns the length of the ray in centimeters.

```
void RaySegment::GetPoints(Position &Start,
  Position &End) const;
```

Returns the starting and ending points of the ray.

```
Position RaySegment::GetStartPoint() const;
```

Returns the starting point of the ray.

```
void RaySegment::GetStartPoint(float &x,
  float &y) const;
```

Returns the starting point of the ray in x and y.

```
float RaySegment::GetThickness() const;
```

Returns the thickness of the ray.

```
bool RaySegment::HasArrow() const;
```

Returns true if the ray has an arrow; otherwise, it returns false.

```
void RaySegment::SetArrowhead();
```

Sets the ray to be drawn with an arrowhead.

```
void RaySegment::SetColor(const color &c);
```

Sets the color of the ray to c.

```
void RaySegment::SetEndPoint(const Position &p);
```

Sets the ending point of the ray to p.

```
void RaySegment::SetEndPoint(float x, float y);
```

Sets the ending point of the ray to Position(x, y).

```
void RaySegment::SetPoints(const Position &StartPoint,
  const Position &EndPoint);
```

Sets the ray's starting point to StartPoint and its ending point to EndPoint.

```
void RaySegment::SetStartPoint(const Position &p);
```

Sets the starting point of the ray to p.

```
void RaySegment::SetStartPoint(float x, float y);
```

Sets the starting point of the ray to Position(x, y).

```
void RaySegment::SetThickness(float t);
```

Sets the thickness of the ray to `t`. The units of thickness is centimeters.

A.7 CLASS SHAPE

Class `Shape` is the base class for classes `CircleShape`, `EllipseShape`, `RectangleShape`, `TriangleShape`, and `SquareShape`. The class provides a public constructor that is described below.

```
Shape::Shape(SimpleWindow &w, const Position &p,
  const color &c = Red);
```

Creates a `Shape` object that is centered at position `p` in window `w`. The color of the object is `c`, which by default is the value `Red`.

The `Shape` class provides several public member functions that are described below.

```
void Shape::ClearBorder();
```

Set the shape to not have a border.

```
virtual void Shape::Draw() = 0;
```

Member function `Draw()` is a pure virtual function.

```
color Shape::GetColor() const;
```

Returns the color of the object.

```
bool Shape::HasBorder() const;
```

Returns true if the shape has a border; otherwise, it returns false.

```
void Shape::SetBorder();
```

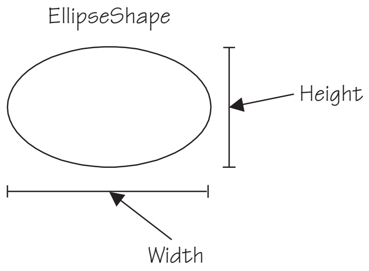
Sets the shape to have a border.

```
void Shape::SetColor(const color &c);
```

Sets the color of the object to `c`.

A.8 CLASS ELLIPSESHAPE

Class `EllipseShape` is derived publicly from class `Shape`. An EzWindows `EllipseShape` is shown below.



The class `EllipseShape` has the following public constructor:

```
EllipseShape::EllipseShape(SimpleWindow &w,
  const Position &p, const color &c = Red,
  float Width = 1.0f, float Height = 2.0f);
```

Creates an EllipseShape object to represent an ellipse. The ellipse is centered at position p in window w. The ellipse has color c, which by default is the value Red. The ellipse has width Width and height Height. The default values of parameters Width and Height are 1.0 and 2.0, respectively. Parameters Width and Height are centimeters.

The class EllipseShape also has the following public member functions:

`void EllipseShape::Draw();`

Draws the ellipse in its associated window.

`void EllipseShape::Erase();`

Erases the ellipse from its associated window.

`float EllipseShape::GetHeight() const;`

Returns the height of the object in centimeters.

```
void EllipseShape::GetSize(float &Width,
  float &Height) const;
```

Returns the width and height of the object in centimeters.

`float EllipseShape::GetWidth() const;`

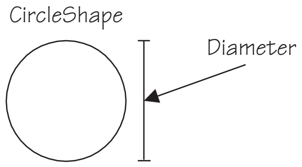
Returns the length of the object in centimeters.

`void EllipseShape::SetSize(float Width, float Height);`

Sets the width of the ellipse to Width and the height of the ellipse to Height. Parameters Width and Height are centimeters.

A.9 CLASS CIRCLESHAPE

Class CircleShape is derived publicly from class Shape. An EzWindows CircleShape is shown below.



Class CircleShape has the following public constructor:

```
CircleShape::CircleShape(SimpleWindow &w,
  const Position &p, const color &c = Red,
  float Diameter = 1.0f);
```

Creates a CircleShape object to represent a circle. The circle is centered at position p in window w. The circle has color c, which by default is the value Red. The circle has diameter Diameter. The default value of Diameter is 1.0. Parameter Diameter is centimeters.

The class CircleShape also has the following public member functions:

`void CircleShape::Draw();`

Draws the circle in its associated window.

```
void CircleShape::Erase();
```

Erases the circle from its associated window.

```
float CircleShape::GetDiameter() const;
```

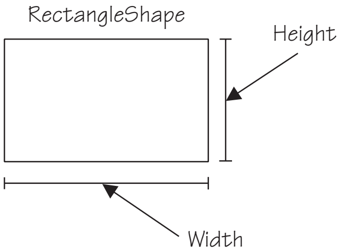
Returns the diameter of the circle in centimeters.

```
void CircleShape::SetSize(float Diameter);
```

Sets the diameter of the circle to `Diameter`. Parameter `Diameter` is centimeters.

A10 CLASS RECTANGLESHAPE

Class `RectangleShape` is derived publicly from class `Shape`. An EzWindows `RectangleShape` is shown below.



Class `RectangleShape` has the following public constructors:

```
RectangleShape::RectangleShape(SimpleWindow &w,
  const Position &p, const color &c = Red,
  float Width = 1.0f, float Height = 2.0f);
```

Creates a `RectangleShape` object to represent a rectangle. The rectangle is centered at position `p` in window `w`. The rectangle has color `c`, which by default is the value `Red`. The width of the rectangle is `Width`. The height of the rectangle is `Height`. The default values of `Width` and `Height` are 1.0 and 2.0, respectively. Parameters `Width` and `Height` are centimeters.

```
RectangleShape::RectangleShape(SimpleWindow &w,
  float XCoord, float YCoord, const color &c = Red,
  float Width = 1.0f, float Height = 2.0f);
```

Creates a `RectangleShape` object to represent a rectangle. The rectangle is centered at `Position(XCoord,YCoord)` in window `w`. The rectangle has color `c`, which by default is the value `Red`. The width of the rectangle is `Width`. The height of the rectangle is `Height`. The default values of `Width` and `Height` are 1.0 and 2.0, respectively. Parameters `Width` and `Height` are centimeters.

The class `RectangleShape` also has the following public member functions:

```
void RectangleShape::Draw();
```

Draws the rectangle in its associated window.

```
void RectangleShape::Erase();
```

Erases the rectangle from its associated window.

```
float RectangleShape::GetHeight() const;
```

Returns the height of the rectangle in centimeters.

```
void RectangleShape::GetSize(float &Width,
  float &Height) const;
```

Returns the width and height of the rectangle in centimeters.

```
float RectangleShape::GetWidth() const;
```

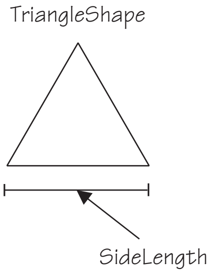
Returns the width of the rectangle in centimeters.

```
void RectangleShape::SetSize(float Width,
  float Height);
```

Sets the width of the rectangle to width and the height of the rectangle to Height. Parameters Width and Height are centimeters.

A.11 CLASS TRIANGLESHAPE

Class TriangleShape is derived publicly from class Shape. An EzWindows TriangleShape is shown below.



Class TriangleShape has the following public constructor:

```
TriangleShape::TriangleShape(SimpleWindow &w,
  const Position &p, const color &c = Red,
  float SideLength = 1.0f);
```

Creates a TriangleShape object to represent an equilateral triangle. The triangle is centered at position p in window w. The triangle has color c, which by default is the value Red. The length of a side of the triangle is SideLength. The default value of SideLength is 1.0. Parameter SideLength is centimeters.

The class TriangleShape also has the following public member functions:

```
void TriangleShape::Draw();
```

Draws the triangle in its associated window.

```
void TriangleShape::Erase();
```

Erases the triangle from its associated window.

```
float TriangleShape::GetSideLength() const;
```

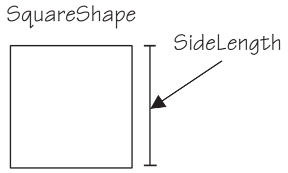
Returns the side length of the triangle in centimeters.

```
void TriangleShape::SetSize(float SideLength);
```

Sets the side length of the triangle to SideLength. Parameter SideLength is centimeters.

A.12 CLASS SQUARESHAPE

Class SquareShape is derived publicly from class Shape. An EzWindows SquareShape is shown below.



Class SquareShape has the following public member functions.

```
SquareShape::SquareShape(SimpleWindow &Window,
  const Position &Center, const color &c = Red,
  float Side = 1.0f);
```

Creates a SquareShape object to represent a square. The square is centered at position p in window w. The square has color c, which by default is the value Red. The length of the side of the square is SideLength. The default value of SideLength is 1.0. Parameter SideLength is centimeters.

The class SquareShape also has the following public member functions:

```
void SquareShape::Draw();
```

Draws the square in its associated window.

```
void SquareShape::Erase();
```

Erases the square from its associated window.

```
float SquareShape::GetSideLength() const;
```

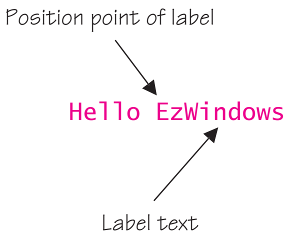
Returns the side length of the square in centimeters.

```
void SquareShape::SetSize(float SideLength);
```

Sets the side length of the square to SideLength. Parameter SideLength is centimeters.

A.13 CLASS LABEL

Class `Label` is publicly derived from `WindowObject`. An EzWindows `Label` is shown below.



Class `Label` has the following public constructors:

```
Label::Label(SimpleWindow &w, const Position &p,
  const string &Text, const color &TextColor = Black,
  const color &BackGroundColor = White);
```

Creates a `Label` object to represent a text message. The message is contained in the `string` object `Text`. The message is centered at position `p` in window `w`. The color of the message text is `TextColor`. The default color of the message text is black. The message has background color `BackGroundColor`, which by default is white.

```
Label::Label(SimpleWindow &w, float XCoord,
  float YCoord, const string &Text,
  const color &TextColor = Black,
  const color &BackGroundColor = White);
```

Creates a `Label` object to represent a text message. The message is contained in the `string` object `Text`. The message is centered at position `(XCoord, YCoord)` in window `w`. The color of the message text is `TextColor`. The default color of the message text is black. The message has background color `BackGroundColor`, which by default is white.

```
Label::Label(SimpleWindow &w, const Position &p,
  const char *Text, const color &TextColor = Black,
  const color &BackGroundColor = White);
```

Creates a `Label` object to represent a text message. The `char` pointer `Text` is a pointer to the text message to display. The message is centered at position `p` in window `w`. The color of the message text is `TextColor`. The default color of the message text is black. The message has background color `BackGroundColor`, which by default is white.

```
Label::Label(SimpleWindow &w, float XCoord,
  float YCoord, const char *Text,
  const color &TextColor = Black,
  const color &BackGroundColor = White);
```

Creates a `Label` object to represent a text message. The `char` pointer `Text` is a pointer to the text message to display. The message is centered at position `(XCoord, YCoord)` in window `w`. The color of the message text is `TextColor`. The default color of the message text is black. The message has background color `BackGroundColor`, which by default is white.

The class `Label` also has the following public member functions:

`void` `Label::Draw();`

Draws the label in its associated window.

`void` `Label::Erase();`

Erase the label from its associated window.

`color Label::GetColor()` **`const`**`;`

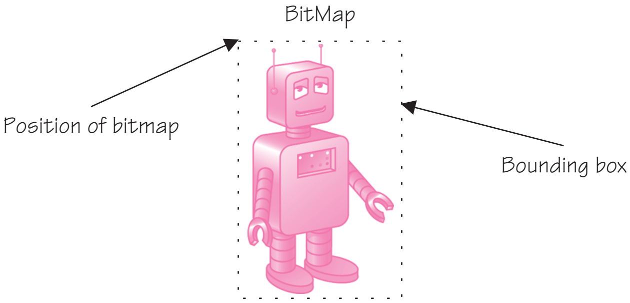
Returns the background color of the label.

`void` `Label::SetColor(`**`const`** `color &c);`

Sets the background color of the label to `c`.

A.14 CLASS BITMAP

Unlike the window shape objects (i.e., `RectangleShape`, `EllipseShape`, `CircleShape`, etc.), a bitmap is positioned using the upper-left corner of its bounding box. An EzWindows `BitMap` is shown below.



Class `BitMap` has the following public constructors:

`BitMap::BitMap();`

Creates a `BitMap` object with `BitMapStatus` `NoBitMap`. The object is not associated with any window.

`BitMap::BitMap(SimpleWindow &w);`

Creates a `BitMap` object with `BitMapStatus` `NoBitMap`. The object is associated with window `w`.

`BitMap::BitMap(SimpleWindow *w);`

Creates a `BitMap` object with `BitMapStatus` `NoBitMap`. The object is associated with the window pointed to by `w`.

The class `BitMap` also has the following public member functions:

`bool` `BitMap::Draw();`

Attempts to display the bitmap object to the associated window. The `BitMapStatus` of the object must be `BitMapOkay` for the display to be successful. If the bitmap is displayed, the function returns true; otherwise, the function returns false.

bool BitMap::Erase();

Overwrites the bitmap on the display by drawing a white rectangle of the same size. If the bitmap is successfully erased, the function returns true; otherwise, the function returns false.

bool BitMap::IsInside(**const** Position &p) **const**;

Returns true if position p lies within the bitmap; otherwise, the function returns false.

float BitMap::GetHeight() **const**;

Returns the height of the bitmap in centimeters.

Position BitMap::GetPosition() **const**;

Returns the position of the bitmap.

void BitMap::GetSize(**float** &Width,
  float &Height) **const**;

Returns both the width and height of the bitmap in centimeters.

BitMapStatus BitMap::GetStatus() **const**;

Returns the current BitMapStatus value associated with the object.

float BitMap::GetWidth() **const**;

Returns the width of the bitmap in centimeters.

float BitMap::GetXPosition() **const**;

Returns the distance from the upper-left corner of the bitmap to the left edge of the associated window. The distance is in centimeters.

float BitMap::GetYPosition() **const**;

Returns the distance from the upper-left corner of the bitmap to the top edge of the associated window. The distance is in centimeters.

BitMapStatus BitMap::Load(**const** string &Filename);

Uses the file whose name is Filename to set the bitmap. If the file contains a valid bitmap, the status of the object is set to BitMapOkay; otherwise, the status of the object is set to NoBitMap.

BitMapStatus BitMap::Load(**const** **char** *Filename);

Uses the file whose name is pointed to by character string Filename to set the bitmap. If the file contains a valid bitmap, the status of the object is set to BitMapOkay; otherwise, the status of the object is set to NoBitMap.

void BitMap::SetPosition(**const** Position &p);

Sets the position of the bitmap to p.

void BitMap::SetWindow(SimpleWindow &w);

Associates the bitmap with window w. The BitMapStatus of the bitmap is set to NoBitMap.

A.15 CLASS RANDOMINT

Class RandomInt provides the ability to produce uniform random numbers in a specified interval. The class has the following public constructors:

RandomInt::RandomInt(**int** a = 0, **int** b = RAND_MAX);

Creates a RandomInt object that generates pseudorandom numbers in the inclusive interval (a, b). The default interval is (0, RAND_MAX). The value RAND_MAX is defined in stdlib.h.

RandomInt::RandomInt(int a, int b, unsigned int Seed);
Creates a RandomInt object that generates pseudorandom numbers in the inclusive interval (a, b). The pseudorandom-number generator is initialized with the value in Seed.

The class RandomInt also has the following public member functions:

int RandomInt::Draw();
Returns the next pseudorandom number.

unsigned int EzRandomize();
Generates a new seed value for the pseudorandom-number generator. Returns the new seed.

int RandomInt::GetLow() const;
Returns the low endpoint of the interval.

int RandomInt::GetHigh() const;
Returns the high endpoint of the interval.

void RandomInt::SetInterval(int a, int b);
Sets the inclusive interval for RandomInt object to (a, b).

void RandomInt::SetSeed(unsigned int Seed);
Sets the pseudorandom-number generator seed to Seed.

A.16 MISCELLANEOUS FUNCTIONS

long GetMilliseconds()
Returns the value of a timer that is ticking continuously. The resolution of the timer is milliseconds.

void Terminate()
Sends a terminate message to the EzWindows window manager.

CHECK-OFF SHEET: LABORATORY 1

1.1 IDENTIFICATION

Name: E-mail: Name: E-mail:
Section:

1.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Directory manipulation ________________________

Ran "Hello, World" ________________________

Number fun ________________________

Representation problems ________________________

CHECK-OFF SHEET: LABORATORY 2

2.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

2.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

A, b, c solving ______________________

Wrote results ______________________

Ran original lawn ______________________

Ran modified lawn ______________________

CHECK-OFF SHEET: LABORATORY 3

3.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

3.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Justified testing ________________________

Restructured nand.cpp ________________________

Understood the debugger ________________________

Prediction ________________________

Zeroed out ________________________

Overlapping rectangles ________________________

CHECK-OFF SHEET: LABORATORY 4

4.1 IDENTIFICATION

Name: E-mail: Name: E-mail:
Section:

4.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Sum modifications ______________________

For sum ______________________

Counting ______________________

Concentricity ______________________

Counting words ______________________

Fraction 1/8 ______________________

Fraction 1/10 ______________________

Fixed upper.cpp ______________________

CHECK-OFF SHEET: LABORATORY 5

5.1 IDENTIFICATION

Name: E-mail: Name: E-mail:
Section:

5.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

String comparisons ____________________

String length ____________________

String concatenation ____________________

String concatenation (+) ____________________

Word count ____________________

Word count with check ____________________

CHECK-OFF SHEET: LABORATORY 6

6.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

6.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Problem 1 ______________________

Problem 2 ______________________

Problem 3 ______________________

Problem 4 ______________________

Problem 5 ______________________

Problem 6 ______________________

Problem 7 ______________________

Problem 8 ______________________

Problem 9 ______________________

Problem 10 ______________________

CHECK-OFF SHEET: LABORATORY 7

7.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

7.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Problem 1 ____________________

Problem 2 ____________________

Problem 3 ____________________

Problem 4 ____________________

Problem 5 ____________________

IsEndOfSentence function ____________________

LineSpace function ____________________

Update function ____________________

Working program ____________________

Modification ____________________

CHECK-OFF SHEET: LABORATORY 8

8.1 IDENTIFICATION

Name: E-mail: Name: E-mail:
Section:

8.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Ifndef purpose ______________________

Knew members ______________________

Made blue ______________________

No inspectors ______________________

Called constructor ______________________

No mutators ______________________

Double or nothing ______________________

Made members ______________________

Line segment ______________________

CHECK-OFF SHEET: LABORATORY 9

9.1 IDENTIFICATION

Name: E-mail: Name: E-mail:
Section:

9.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Rationality ________________________

Made GCD ________________________

Reduce code ________________________

Reduce locations ________________________

Ran reduce ________________________

Operator overloading ________________________

CHECK-OFF SHEET: LABORATORY 10

10.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

10.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Bitmap discussion ______________________

Bitmap modification ______________________

Flipping implementation ______________________

Timer event discussion ______________________

Timer event modification ______________________

Eyes discussion ______________________

Eyes implementation ______________________

CHECK-OFF SHEET: LABORATORY 11

11.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

11.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Modified element listing ____________________

Minimum ____________________

Labeled minimum ____________________

Summing ____________________

Re-deja vu ____________________

Reversing ____________________

Exhaustive searching ____________________

Binary searching ____________________

Modified binary searching ____________________

CHECK-OFF SHEET: LABORATORY 12

12.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

12.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Indexing ________________

Bad At ________________

Duplication ________________

Display functions ________________

Insertion operators ________________

Mean ________________

Resizing ________________

Iterators ________________

Mergesort ________________

Measuring performance ________________

CHECK-OFF SHEET: LABORATORY 13

13.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

13.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Telephone hierarchy ______________________

RectangleShape declaration______________________

RectangleShape constructor______________________

Shadow demonstration ______________________

BoxShape design ______________________

BoxShape class declaration ______________________

BoxShape implementation ______________________

BoxShape demonstration ______________________

Flexible demonstration ______________________

CHECK-OFF SHEET: LABORATORY 14

R.1 IDENTIFICATION

Name:
E-mail:

Name:
E-mail:

Section:

R.2 CHECK-OFFS

As you complete various portions of the laboratory, you will be instructed to ask a laboratory instructor to initial your progression on this sign-off sheet. If you need help, please ask a classmate or laboratory instructor for assistance. Hand in this sheet at the end of the laboratory session.

Poly ______________________

Date ______________________

Line ______________________

Summer ______________________

Stripes design ______________________

Stripes part 1 ______________________

Stripes part 2 ______________________