cmake-doxygen-example

1.0

Generated by Doxygen 1.8.20

1 CSCI 102 Lecture #9 Factory Class Example 1
1.1 Purpose/Overview . . . 1
1.2 Requirements . . . 1
1.3 Global Data/Functions . . . 1
1.4 Objects . . . 1
1.5 High-level architecture . . . 2
1.6 Function Descriptions . . . 2
1.7 User Interface . . . 2
1.8 Testing . . . 2

2 Hierarchical Index 3
2.1 Class Hierarchy . . . 3

3 Class Index 5
3.1 Class List . . . 5

4 File Index 7
4.1 File List . . . 7

5 Class Documentation 9
5.1 Circle Class Reference . . . 9
5.1.1 Detailed Description . . . 11
5.1.2 Constructor & Destructor Documentation . . . 11
5.1.2.1 Circle() [1/2] . . . 11
5.1.2.2 Circle() [2/2] . . . 11
5.1.3 Member Function Documentation . . . 12
5.1.3.1 getArea() . . . 12
5.1.3.2 getType() . . . 12
5.1.4 Member Data Documentation . . . 13
5.1.4.1 radius . . . 13
5.2 Point Class Reference . . . 13
5.2.1 Detailed Description . . . 14
5.2.2 Constructor & Destructor Documentation . . . 14
5.2.2.1 Point() [1/2] . . . 14
5.2.2.2 Point() [2/2] . . . 15
5.2.3 Member Function Documentation . . . 16
5.2.3.1 getX() . . . 16
5.2.3.2 getY() . . . 17
5.2.3.3 operator+=() . . . 17
5.2.3.4 operator=() . . . 18
5.2.3.5 setX() . . . 19
5.2.3.6 setY() . . . 19
5.2.4 Friends And Related Function Documentation . . . 19
5.2.4.1 operator"!= . . . 20

5.2.4.2 operator+ [1/2] . . . 20
5.2.4.3 operator+ [2/2] . . . 21
5.2.4.4 operator- . . . 21
5.2.4.5 operator<< . . . 22
5.2.4.6 operator== . . . 23
5.2.4.7 operator>> . . . 23
5.2.5 Member Data Documentation . . . 24
5.2.5.1 x . . . 24
5.2.5.2 y . . . 24
5.3 RandomSizeShapeFactory Class Reference . . . 25
5.3.1 Detailed Description . . . 25
5.3.2 Constructor & Destructor Documentation . . . 25
5.3.2.1 RandomSizeShapeFactory() . . . 25
5.3.3 Member Function Documentation . . . 26
5.3.3.1 createShape() . . . 26
5.4 Rectangle Class Reference . . . 27
5.4.1 Detailed Description . . . 30
5.4.2 Constructor & Destructor Documentation . . . 30
5.4.2.1 Rectangle() [1/2] . . . 30
5.4.2.2 Rectangle() [2/2] . . . 30
5.4.3 Member Function Documentation . . . 31
5.4.3.1 getArea() . . . 31
5.4.3.2 getType() . . . 32
5.4.4 Member Data Documentation . . . 32
5.4.4.1 length . . . 32
5.4.4.2 width . . . 32
5.5 Shape Class Reference . . . 33
5.5.1 Detailed Description . . . 35
5.5.2 Constructor & Destructor Documentation . . . 35
5.5.2.1 Shape() [1/2] . . . 35
5.5.2.2 Shape() [2/2] . . . 35
5.5.3 Member Function Documentation . . . 36
5.5.3.1 getArea() . . . 36
5.5.3.2 getCenter() . . . 37
5.5.3.3 getType() . . . 37
5.5.3.4 setCenter() . . . 38
5.5.4 Member Data Documentation . . . 38
5.5.4.1 center . . . 38
5.6 Triangle Class Reference . . . 38
5.6.1 Detailed Description . . . 41
5.6.2 Constructor & Destructor Documentation . . . 41
5.6.2.1 Triangle() [1/2] . . . 41

5.6.2.2 Triangle() [2/2] . . . 41
5.6.3 Member Function Documentation . . . 42
5.6.3.1 getArea() . . . 42
5.6.3.2 getType() . . . 43
5.6.4 Member Data Documentation . . . 43
5.6.4.1 base . . . 43
5.6.4.2 height . . . 43

6 File Documentation **45**
6.1 Include/circle.h File Reference . . . 45
6.2 Include/point.h File Reference . . . 46
6.3 Include/rectangle.h File Reference . . . 47
6.4 Include/shape.h File Reference . . . 48
6.5 Include/shapefactory.h File Reference . . . 48
6.6 Include/triangle.h File Reference . . . 49
6.7 Src/circle.cpp File Reference . . . 50
6.8 Src/main.cpp File Reference . . . 51
6.8.1 Function Documentation . . . 52
6.8.1.1 main() . . . 52
6.9 Src/point.cpp File Reference . . . 53
6.9.1 Function Documentation . . . 54
6.9.1.1 operator"!=() . . . 54
6.9.1.2 operator+() [1/2] . . . 54
6.9.1.3 operator+() [2/2] . . . 55
6.9.1.4 operator-() . . . 55
6.9.1.5 operator<<() . . . 56
6.9.1.6 operator==() . . . 56
6.9.1.7 operator>>() . . . 57
6.10 Src/rectangle.cpp File Reference . . . 57
6.11 Src/shape.cpp File Reference . . . 58
6.12 Src/shapefactory.cpp File Reference . . . 58
6.12.1 Function Documentation . . . 59
6.12.1.1 getRandomNumber() . . . 59
6.13 Src/triangle.cpp File Reference . . . 59

Index **61**

Chapter 1

CSCI 102 Lecture #9 Factory Class Example

1.1 Purpose/Overview

The purpose of this program is to demonstrate the power and flexibility of a factory class. The user will input names of shapes from the console and behind the scenes a factory will instantiate and return randomly sized shapes based on the input from the user. When the user finishes entering shapes, the program will also use polymorphism to print out the type and area of each shape that was created.

1.2 Requirements

The application shall accepts names of shapes from the user via the console.

The application shall gracefully report and handle invalid shape names without crashing.

The application shall use user input shape names to generate corresponding shape objects via a factory class.

The application shall provide a way for the user to indicate they are done entering shapes.

The application, upon normal exit, shall display a summary to the user of all the shapes that were created and their areas.

1.3 Global Data/Functions

There is no global data in this application. All the data is contained within objects.

1.4 Objects

See the individual object documentation pages for more info on each object.

Point - A class representing a 2D (x,y) point in space. Shape - An abstract base class used to represent a Shape. Triangle - A class to represent a triangle shape in space. Inherits from Shape. Rectangle - A class to represent a rectangle shape in space. Inherits from Shape. Circle - A class to represent a circle shape in space. Inherits from Shape. ShapeFactory - A factory class used for generating Shape pointers from strings.

1.5 High-level architecture

The main method of the program reads user input strings and passes them to the ShapeFactory object. The ShapeFactory object is responsible for knowing how to create Shapes from strings and it will return Shape pointers. Triangle, Rectangle and Circle are only ever reference inside the ShapeFactory. Everywhere else the abstract Shape interface is used to pass around data and polymorphically call functions on the individual shapes. Each Shape contains a Point object that represents its physical location in 2D space (its center point).

The main flow of the code is:

1. Prompt the user for a string
2. Pass the string to the ShapeFactory to create a Shape* 3a. If the shape string is bad, do nothing and reprompt the user. Go to 2. 3b. If the shape string is good, add the created shape to a vector. Go to 2. 3c. If the input string is empty, end the program and dump a summary of created shapes.

1.6 Function Descriptions

All functions are described in detail on the documentation pages for the classes that either contain them or are friends of them.

1.7 User Interface

The only user interface is that the user will be prompted to enter string representations of shapes (e.g. "triangle") and the application will handling all the heavy lifting of creating randomly sized shapes. If the user enters a bad shape, they will be reprompted. If the user enters nothing, the program will end and dump a summary.

1.8 Testing

Test the main code with all the known existing shape types (Triangle, Circle, Rectangle) and make sure that it works. Also try entering the same shape multiple times (e.g. make 3 Circles).

Test that the main code handles bad shape names gracefully. If a user enters a string like "asdfsda" that is not a known shape, check that the application displays an error and reprompts.

Test that when the user enters an empty string, the program prints a summary and terminates.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Point 13
RandomSizeShapeFactory 25
Shape 33
  Circle 9
  Rectangle 27
  Triangle 38

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Circle 9
Point 13
RandomSizeShapeFactory 25
Rectangle 27
Shape 33
Triangle 38

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Include/circle.h . . . 45
Include/point.h . . . 46
Include/rectangle.h . . . 47
Include/shape.h . . . 48
Include/shapefactory.h . . . 48
Include/triangle.h . . . 49
Src/circle.cpp . . . 50
Src/main.cpp . . . 51
Src/point.cpp . . . 53
Src/rectangle.cpp . . . 57
Src/shape.cpp . . . 58
Src/shapefactory.cpp . . . 58
Src/triangle.cpp . . . 59

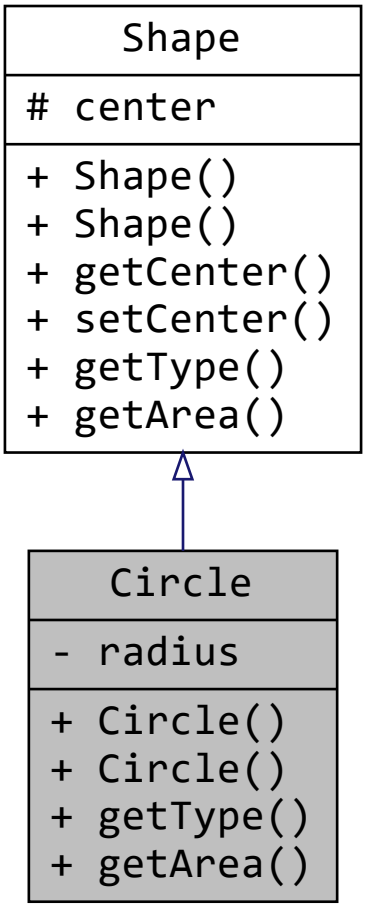
Chapter 5

Class Documentation

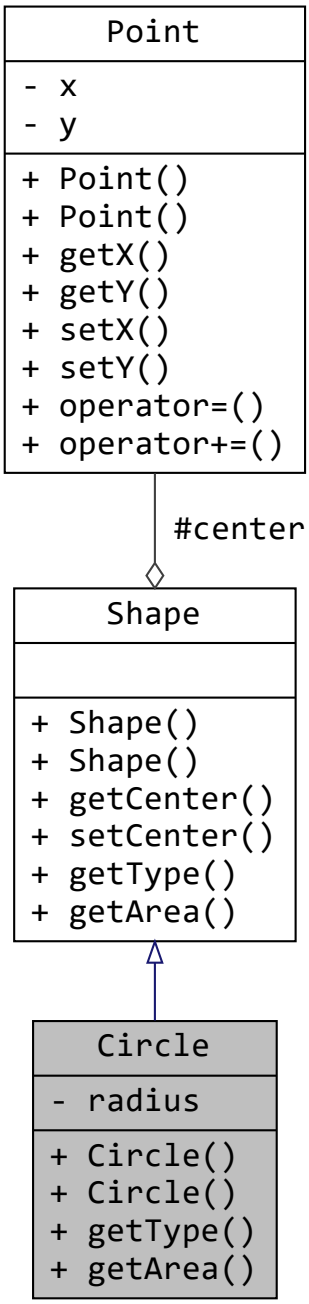
5.1 Circle Class Reference

`#include <circle.h>`

Inheritance diagram for Circle:



Collaboration diagram for Circle:



Public Member Functions

- Circle (int r, int x, int y)
- Circle (int r, Point c)
- virtual std::string getType () const
- virtual double getArea () const

Private Attributes

- int radius

Additional Inherited Members

5.1.1 Detailed Description

A class that represents a circle on a 2D plane. Its only member is its radius and it inherits all of its functionality from the Shape class.

Author

Brent Nash

Definition at line 16 of file circle.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Circle() [1/2]

```
Circle::Circle (
            int r,
            int x,
            int y )
```

Overloaded constructor that takes in a radius and an (x,y) coordinate.

Parameters

r	The radius of the circle (must be greater than 0)
x	The X coordinate of the circle's center point in 2D space
y	The Y coordinate of the circle's center point in 2D space

Definition at line 7 of file circle.cpp.

5.1.2.2 Circle() [2/2]

```
Circle::Circle (
            int r,
            Point c )
```

Overloaded constructor that takes in a radius and a center Point.

Parameters

r	The radius of the circle (must be greater than 0)
c	The center point of the circle in 2D space

Definition at line 14 of file circle.cpp.

5.1.3 Member Function Documentation

5.1.3.1 getArea()

```
double Circle::getArea ( ) const  [virtual]
```

Virtual function overridden from Shape. Returns the area of the circle as PI∗radius∗radius.

Precondition

The radius must be set to a valid number.

Postcondition

Does not change the object

Returns

The area of the circle as a floating point number.

Implements Shape.

Definition at line 27 of file circle.cpp.

5.1.3.2 getType()

```
string Circle::getType ( ) const  [virtual]
```

Virtual function overridden from Shape. Returns a string indicating what type of shape this object is.

Precondition

None

Postcondition

Does not change the object

Returns

The string "Circle" Get the Type object
std::string circle

Implements Shape.

Definition at line 21 of file circle.cpp.

5.1.4 Member Data Documentation

5.1.4.1 radius

```
int Circle::radius  [private]
```

The radius of the circle

Definition at line 21 of file circle.h.

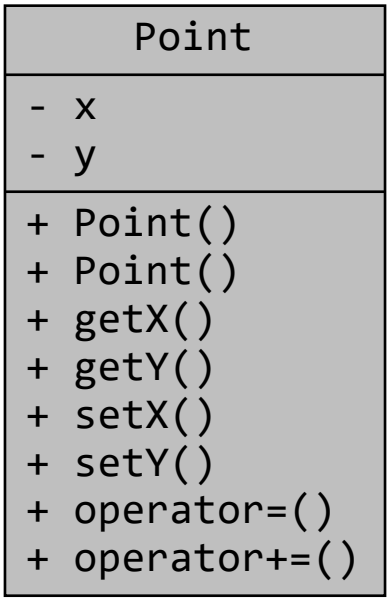
The documentation for this class was generated from the following files:

- Include/circle.h
- Src/circle.cpp

5.2 Point Class Reference

```
#include <point.h>
```

Collaboration diagram for Point:



Public Member Functions

- Point ()
- Point (int newx, int newy)
- int getX () const
- int getY () const
- void setX (int newx)
- void setY (int newy)
- Point & operator= (const Point &other)
- void operator+= (const Point &right)

Private Attributes

- int x
- int y

Friends

- Point operator+ (const Point &a, const Point &b)
- Point operator+ (const Point &a, const int &b)
- std::ostream & operator<< (std::ostream &out, const Point &pt)
- std::istream & operator>> (std::istream &in, Point &pt)
- bool operator== (const Point &a, const Point &b)
- bool operator!= (const Point &a, const Point &b)
- Point operator- (const Point &a)

5.2.1 Detailed Description

A basic class to represent a point somewhere in 2D space.

Author

Brent Nash

Definition at line 12 of file point.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Point() [1/2]

```
Point::Point ( )
```

Default constructor

Postcondition

The X and Y coordinates are initialized to zero

Definition at line 5 of file point.cpp.

5.2.2.2 Point() [2/2]

```
Point::Point (
            int newx,
            int newy )
```

Overloaded constructor

Parameters

newx	The initial value for the X coordinate
newy	The initial value for the Y coordinate

Definition at line 11 of file point.cpp.

5.2.3 Member Function Documentation

5.2.3.1 getX()

```
int Point::getX ( ) const
```

Accessor for the X coordinate

Precondition

None

Postcondition

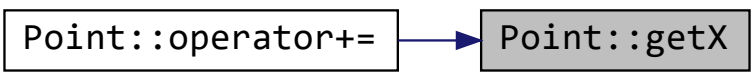
This method does not change the object

Returns

The integer X coordinate

Definition at line 17 of file point.cpp.

Here is the caller graph for this function:



5.2.3.2 getY()

```
int Point::getY ( ) const
```

Accessor for the Y coordinate

Precondition

None

Postcondition

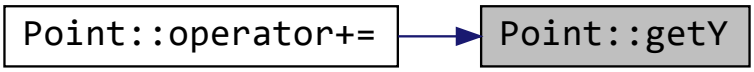
This method does not change the object

Returns

The integer Y coordinate

Definition at line 22 of file point.cpp.

Here is the caller graph for this function:



5.2.3.3 operator+=()

```
void Point::operator+= (
            const Point & right )
```

Overloaded += opreator used to increment the values in the current point by the coordinate values in another point. This will take each of the coordinates in "right" and add them to each of the coordinates in "this" respectively. This implements:

Point a, b; a += b;

Precondition

None

Postcondition

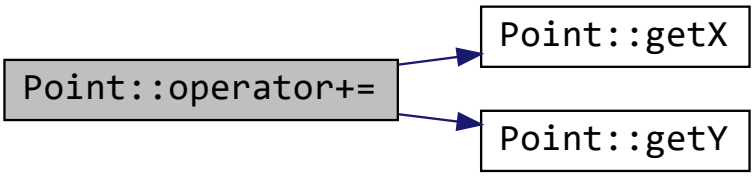
The values in "this" will have been incremented by the values in "right"

Parameters

right	The Point whose coordinates should be added to the coordinates of this Point (passed by const reference).

Definition at line 37 of file point.cpp.

Here is the call graph for this function:



5.2.3.4 operator=()

```
Point & Point::operator= (
            const Point & other )
```

Overloaded = operator used to set two Points equal to each other. Declared as a member function because it changes the current object. This implements:

Point a, b; a = b;

Precondition

None

Postcondition

The values stored inside "this" will be changed to the values stored inside of "other"

Parameters

other	The Point whose values should be copied into this Point (passed by const reference)

Returns

A reference to "this" so that we can chain '=' operations together.

Definition at line 43 of file point.cpp.

5.2.3.5 setX()

```
void Point::setX (
            int newx )
```

Mutator for the X coordinate

Precondition

None

Postcondition

The X coordinate will be updated to the value of "newx"

Parameters

newx	The new value for the X coordinate

Definition at line 27 of file point.cpp.

5.2.3.6 setY()

```
void Point::setY (
            int newy )
```

Mutator for the Y coordinate

Precondition

None

Postcondition

The Y coordinate will be updated to the value of "newy"

Parameters

newy	The new value for the Y coordinate

Definition at line 32 of file point.cpp.

5.2.4 Friends And Related Function Documentation

5.2.4.1 operator"!=

```
bool operator!= (
            const Point & a,
            const Point & b )  [friend]
```

Overloaded operator to compare two Points for inequality. Two points are not equal if any of their individual member coordinates are different. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; if(a != b)

Precondition

None

Postcondition

No changes

Parameters

a	The Point on the left side of the != operation (passed by const reference)
b	The Point on the right side of the != operation (passed by const reference)

Returns

A bool value of "true" if the Points are not equal. False otherwise.

Definition at line 71 of file point.cpp.

5.2.4.2 operator+ [1/2]

```
Point operator+ (
            const Point & a,
            const int & b )  [friend]
```

Overloaded + operator for adding an integer to a point. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a; int x; Point c = a + x;

Precondition

None

Postcondition

Does not change the current object

Parameters

a	The point on the left side of the + sign (passed by const reference)
b	The int on the right side of the + sign (passed by const reference)

Returns

A new point object created by adding b to each of the coordinates of a (e.g. a.x + b, a.y + b, etc.)

Definition at line 97 of file point.cpp.

5.2.4.3 operator+ [2/2]

```
Point operator+ (
            const Point & a,
            const Point & b )  [friend]
```

Overloaded + operator for adding two points together. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; Point c = a + b;

Precondition

None

Postcondition

Does not change the current object

Parameters

a	The point on the left side of the + sign (passed by const reference)
b	The point on the right side of the + sign (passed by const reference)

Returns

A new point object created by adding together the individual coordinates from a & b (e.g. a.x + b.x, a.y + b.y, etc.)

Definition at line 89 of file point.cpp.

5.2.4.4 operator-

```
Point operator- (
            const Point & a )  [friend]
```

Overloaded unary operator negation operator to multiply each coordinate of the input Point by -1. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; a = -b;

Precondition

None

Postcondition

No changes

Parameters

a	The Point on the right side of the - operation (passed by const reference)

Returns

A point whose values are the negation of the values in the input Point a (e.g. if (5,6) is input, then (-5,-6) should be returned).

Definition at line 58 of file point.cpp.

5.2.4.5 operator<<

```
std::ostream& operator<< (
             std::ostream & out,
             const Point & pt )  [friend]
```

Overloaded << operator for inserting a point in an output stream. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a; cout << a;

Outputs a Point in the form "(x,y)" (no quotes)

Precondition

None

Postcondition

The ostream "out" will have "pt" inserted into it

Parameters

out	The output stream being modified (passed by reference...will be changed!)
pt	The point being inserted into the output stream (passed by const reference)

Returns

A reference to the input parameter "out" so that we can chain multiple << operators together.

5.2.4.6 operator==

```
bool operator== (
            const Point & a,
            const Point & b )  [friend]
```

Overloaded operator to compare two Points for equality. Two points are equal if each of their individual member coordinates are equal. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; if(a == b)

Precondition

None

Postcondition

No changes

Parameters

a	The Point on the left side of the == operation (passed by const reference)
b	The Point on the right side of the == operation (passed by const reference)

Returns

A bool value of "true" if the Points are equal. False otherwise.

Definition at line 66 of file point.cpp.

5.2.4.7 operator>>

```
std::istream& operator>> (
            std::istream & in,
            Point & pt )  [friend]
```

Overloaded >> operator for extracting a point from an input stream. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a; cin >> a;

Expects a Point to be entered in the form "X Y" (no quotes).

Precondition

None

Postcondition

The point "a" will be changed by whatever information was extracting from the istream

Parameters

in	The input stream being modified (passed by reference...will be changed!)
pt	The point having input stream information extracted into it (passed by reference...will be changed!)

Returns

A reference to the input parameter "in" so that we can chain multiple >> operators together.

5.2.5 Member Data Documentation

5.2.5.1 x

```
int Point::x  [private]
```

The X coordinate in 2D space

Definition at line 17 of file point.h.

5.2.5.2 y

```
int Point::y  [private]
```

The Y coordinate in 2D space

Definition at line 19 of file point.h.

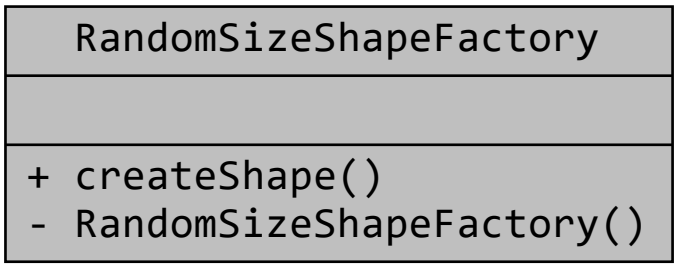
The documentation for this class was generated from the following files:

- Include/point.h
- Src/point.cpp

5.3 RandomSizeShapeFactory Class Reference

`#include <shapefactory.h>`

Collaboration diagram for RandomSizeShapeFactory:



Static Public Member Functions

- static Shape * createShape (std::string name)

Private Member Functions

- RandomSizeShapeFactory ()

5.3.1 Detailed Description

A factory class to generate Shapes with randomly generated sizes/dimensions.

Author

Brent Nash

Definition at line 13 of file shapefactory.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 RandomSizeShapeFactory()

`RandomSizeShapeFactory::RandomSizeShapeFactory ( )  [private]`

Default constructor. Private so no one can ever instantiate this class.

Definition at line 30 of file shapefactory.cpp.

5.3.3 Member Function Documentation

5.3.3.1 createShape()

```
Shape * RandomSizeShapeFactory::createShape (
            std::string name )  [static]
```

Static factory method for creating shapes. Takes in a string representation of a shape and returns a pointer to that particular shape. Only works for shapes that derive from the Shape object. Center points for returned shapes will always be (0,0).

Precondition

None

Postcondition

Does not change the object

Parameters

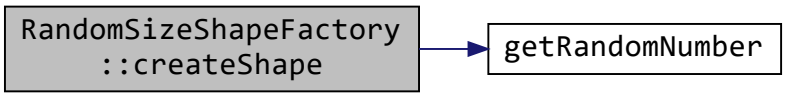
name	The name of the shape to generate (e.g. "triangle", "rectangle", or "circle"). Input is case-sensitive.

Returns

A pointer to a Shape object dynamically allocated on the heap if "name" is a valid, recognized shape type or NULL if "name" does not specify the name of a known shape.

Definition at line 36 of file shapefactory.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

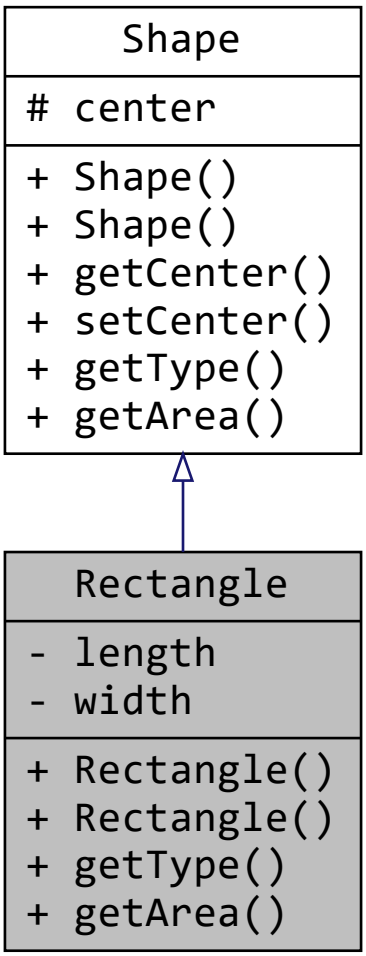
The documentation for this class was generated from the following files:

- Include/shapefactory.h
- Src/shapefactory.cpp

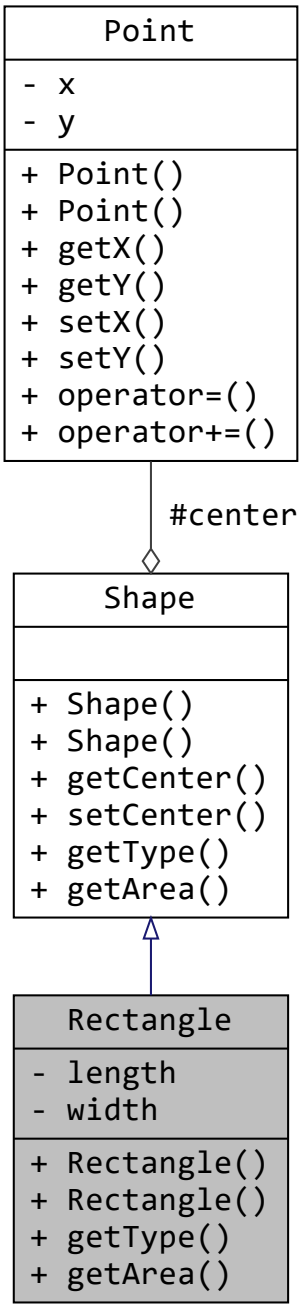
5.4 Rectangle Class Reference

```
#include <rectangle.h>
```

Inheritance diagram for Rectangle:



Collaboration diagram for Rectangle:



Public Member Functions

- Rectangle (int l, int w, int x, int y)
- Rectangle (int l, int w, Point c)
- virtual std::string getType () const
- virtual double getArea () const

Private Attributes

- int length
- int width

Additional Inherited Members

5.4.1 Detailed Description

A class that represents a rectangle on a 2D plane. Its only members are its length and width. It inherits all of its functionality from the Shape class.

Author

Brent Nash

Definition at line 14 of file rectangle.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Rectangle() [1/2]

```
Rectangle::Rectangle (
            int l,
            int w,
            int x,
            int y )
```

Overloaded constructor that takes in a length, a width, and an (x,y) coordinate.

Parameters

l	The length of the rectangle (must be greater than 0)
w	The width of the rectangle (must be greater than 0)
x	The X coordinate of the rectangle's center point in 2D space
y	The Y coordinate of the rectangle's center point in 2D space

Definition at line 6 of file rectangle.cpp.

5.4.2.2 Rectangle() [2/2]

```
Rectangle::Rectangle (
            int l,
```

```
int w,
Point c )
```

Overloaded constructor that takes in a length, a width, and a center Point.

Parameters

l	The length of the rectangle (must be greater than 0)
w	The width of the rectangle (must be greater than 0)
c	The center point of the rectangle in 2D space

Definition at line 13 of file rectangle.cpp.

5.4.3 Member Function Documentation

5.4.3.1 getArea()

```
double Rectangle::getArea ( ) const  [virtual]
```

Virtual function overridden from Shape. Returns the area of the rectangle as length∗width.

Precondition

The length and width must be set to valid numbers.

Postcondition

Does not change the object

Returns

The area of the rectangle as a floating point number.

Implements Shape.

Definition at line 26 of file rectangle.cpp.

5.4.3.2 getType()

```
string Rectangle::getType ( ) const  [virtual]
```

Virtual function overridden from Shape. Returns a string indicating what type of shape this object is.

Precondition

None

Postcondition

Does not change the object

Returns

The string "Rectangle"

Implements Shape.

Definition at line 20 of file rectangle.cpp.

5.4.4 Member Data Documentation

5.4.4.1 length

```
int Rectangle::length  [private]
```

The length of the rectangle

Definition at line 18 of file rectangle.h.

5.4.4.2 width

```
int Rectangle::width  [private]
```

The width of the rectangle

Definition at line 20 of file rectangle.h.

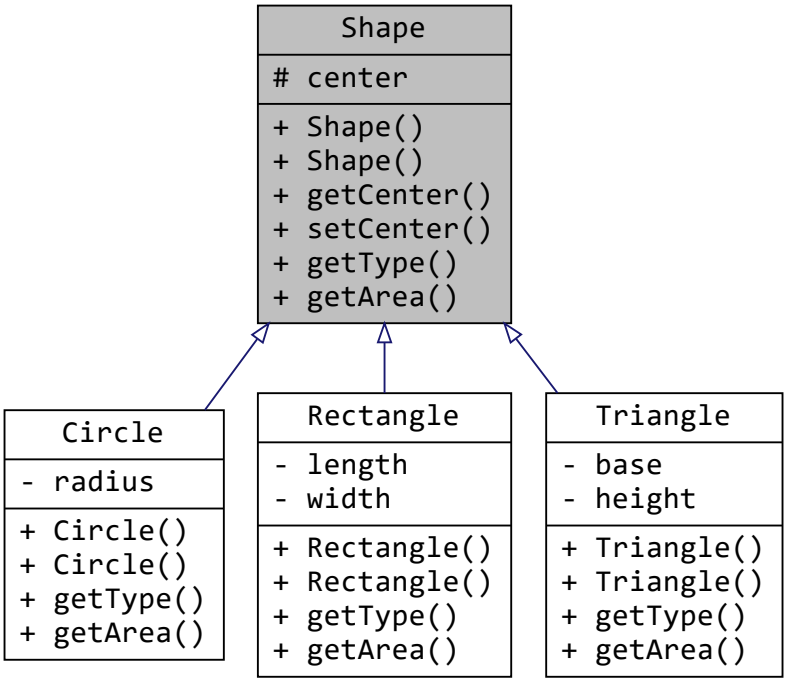
The documentation for this class was generated from the following files:

- Include/rectangle.h
- Src/rectangle.cpp

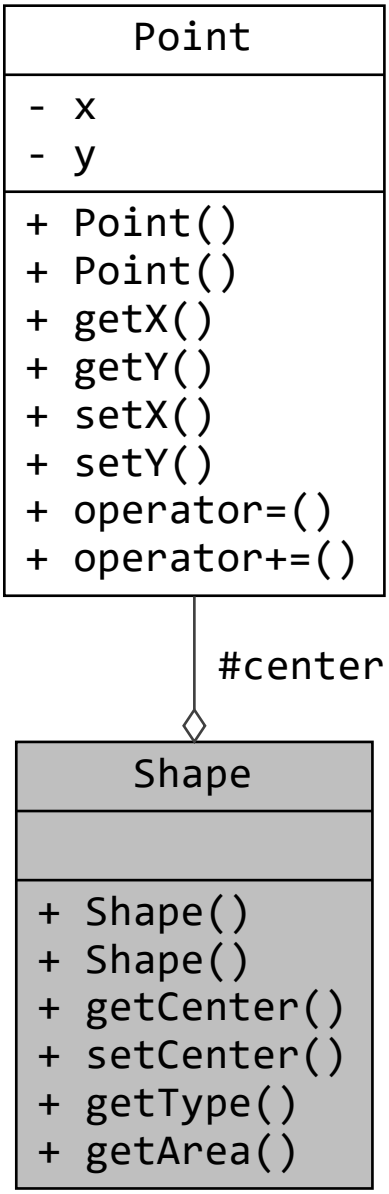
5.5 Shape Class Reference

`#include <shape.h>`

Inheritance diagram for Shape:



Collaboration diagram for Shape:



Public Member Functions

- Shape (int x, int y)
- Shape (Point c)
- Point getCenter () const
- void setCenter (Point c)
- virtual std::string getType () const =0
- virtual double getArea () const =0

Protected Attributes

- Point center

5.5.1 Detailed Description

An abstract base class that represents a shape on a 2D plane. Its only member is "center" which maintains the center point of the shape. It has pure virtual methods for identifying the type of shape and calculating the shape's area that must be implemented by subclasses.

Author

Brent Nash

Definition at line 16 of file shape.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Shape() [1/2]

```
Shape::Shape (
            int x,
            int y )
```

Overloaded constructor that takes in an (x,y) coordinate.

Parameters

x	The X coordinate of the shape's center point in 2D space
y	The Y coordinate of the shape's center point in 2D space

Definition at line 6 of file shape.cpp.

5.5.2.2 Shape() [2/2]

```
Shape::Shape (
            Point c )
```

Overloaded constructor that takes in a center Point.

Parameters

c	The center point of the shape in 2D space

Definition at line 13 of file shape.cpp.

5.5.3 Member Function Documentation

5.5.3.1 getArea()

```
virtual double Shape::getArea ( ) const  [pure virtual]
```

Pure virtual function for Shape subclasses. Subclasses overriding this method should use it to calculate the area of the shape and return the area as a floating point number.

Precondition

Valid data must be set on the Shape object

Postcondition

Does not change the object

Returns

A floating point number representing the area of the Shape

Implemented in Triangle, Rectangle, and Circle.

Here is the caller graph for this function:

5.5.3.2 getCenter()

```
Point Shape::getCenter ( ) const
```

Accessor for the shape's center point.

Precondition

None

Postcondition

Does not change the object

Returns

The current center point of the shape as a Point object.

Definition at line 20 of file shape.cpp.

5.5.3.3 getType()

```
virtual std::string Shape::getType ( ) const  [pure virtual]
```

Pure virtual function for Shape subclasses. Returns a string indicating what type of shape this object is.

Precondition

None

Postcondition

Does not change the object

Returns

A string name of the current shape

Implemented in Triangle, Rectangle, and Circle.

Here is the caller graph for this function:

5.5.3.4 setCenter()

```
void Shape::setCenter (
            Point c )
```

Mutator for the shape's center point.

Precondition

None

Postcondition

The "center" member variable of Shape will be changed to the input value.

Parameters

c	The new center point for the Shape.

Definition at line 26 of file shape.cpp.

5.5.4 Member Data Documentation

5.5.4.1 center

```
Point Shape::center  [protected]
```

The center point of the shape as a 2D coordinate

Definition at line 21 of file shape.h.

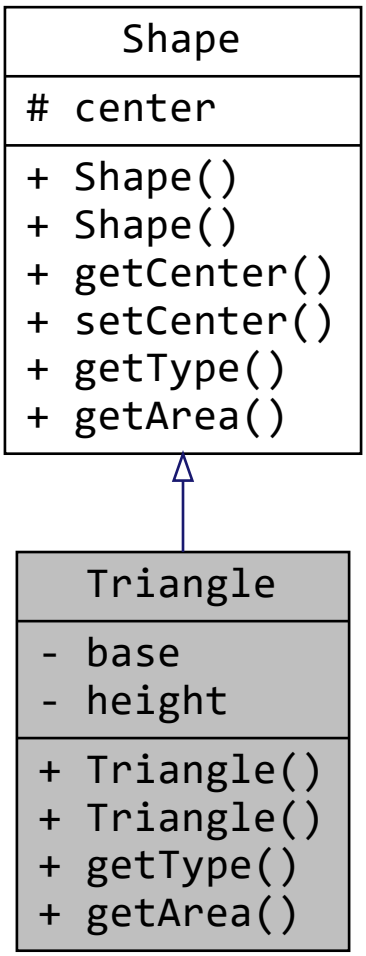
The documentation for this class was generated from the following files:

- Include/shape.h
- Src/shape.cpp

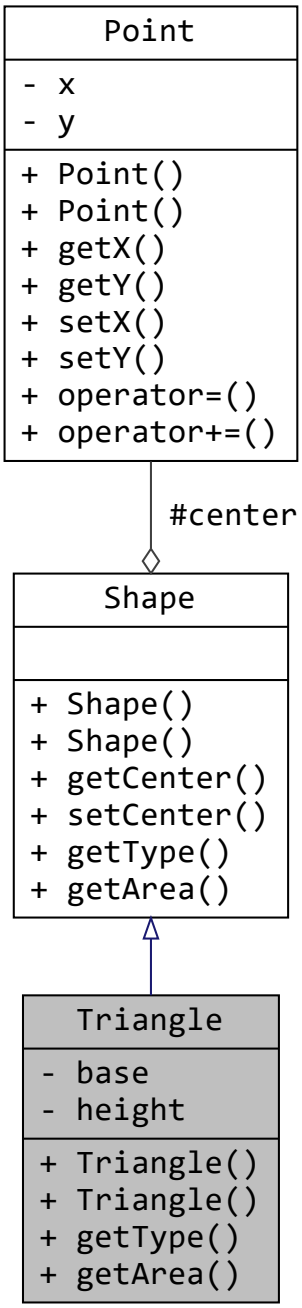
5.6 Triangle Class Reference

```
#include <triangle.h>
```

Inheritance diagram for Triangle:



Collaboration diagram for Triangle:



Public Member Functions

- Triangle (int b, int h, int x, int y)
- Triangle (int b, int h, Point c)
- virtual std::string getType () const
- virtual double getArea () const

Private Attributes

- int base
- int height

Additional Inherited Members

5.6.1 Detailed Description

A class that represents a triangle on a 2D plane. Its only members are its base and height. It inherits all of its functionality from the Shape class.

Author

Brent Nash

Definition at line 14 of file triangle.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Triangle() [1/2]

```
Triangle::Triangle (
            int b,
            int h,
            int x,
            int y )
```

Overloaded constructor that takes in a base, a height and an (x,y) coordinate.

Parameters

b	The base of the triangle (must be greater than 0)
h	The height of the triangle (must be greater than 0)
x	The X coordinate of the triangle's center point in 2D space
y	The Y coordinate of the triangle's center point in 2D space

Definition at line 6 of file triangle.cpp.

5.6.2.2 Triangle() [2/2]

```
Triangle::Triangle (
            int b,
```

```
        int h,
        Point c )
```

Overloaded constructor that takes in a base, a height, and a center Point.

Parameters

b	The base of the triangle (must be greater than 0)
h	The height of the triangle (must be greater than 0)
c	The center point of the triangle in 2D space

Definition at line 13 of file triangle.cpp.

5.6.3 Member Function Documentation

5.6.3.1 getArea()

```
double Triangle::getArea ( ) const  [virtual]
```

Virtual function overridden from Shape. Returns the area of the triangle as (1/2)∗base∗height.

Precondition

The base and height must be set to valid numbers.

Postcondition

Does not change the object

Returns

The area of the circle as a floating point number.

Implements Shape.

Definition at line 26 of file triangle.cpp.

5.6.3.2 getType()

```
string Triangle::getType ( ) const  [virtual]
```

Virtual function overridden from Shape. Returns a string indicating what type of shape this object is.

Precondition

None

Postcondition

Does not change the object

Returns

The string "Triangle"

Implements Shape.

Definition at line 20 of file triangle.cpp.

5.6.4 Member Data Documentation

5.6.4.1 base

```
int Triangle::base  [private]
```

The base of the triangle

Definition at line 18 of file triangle.h.

5.6.4.2 height

```
int Triangle::height  [private]
```

The height of the triangle

Definition at line 20 of file triangle.h.

The documentation for this class was generated from the following files:

- Include/triangle.h
- Src/triangle.cpp

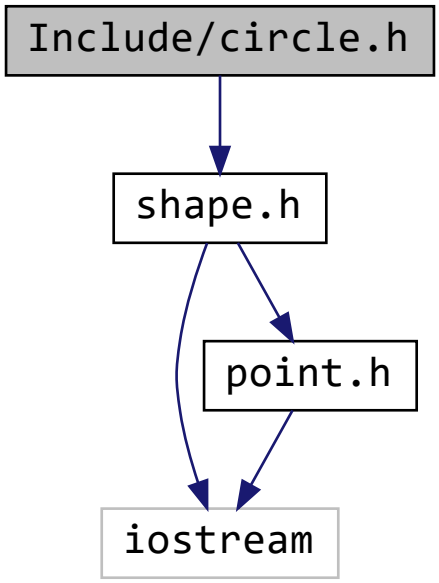
Chapter 6

File Documentation

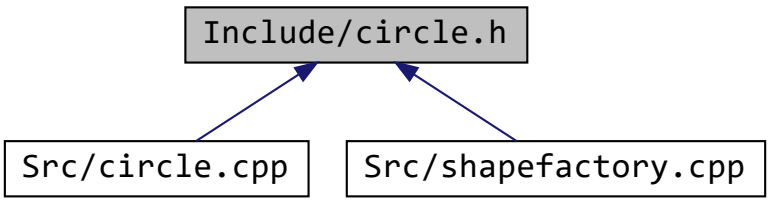
6.1 Include/circle.h File Reference

```
#include "shape.h"
```

Include dependency graph for circle.h:



This graph shows which files directly or indirectly include this file:

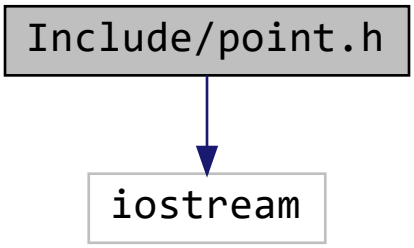


Classes

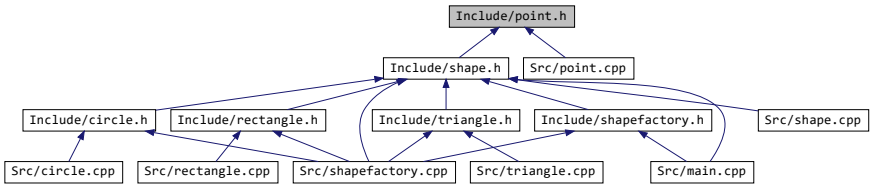
- class Circle

6.2 Include/point.h File Reference

`#include <iostream>`
Include dependency graph for point.h:



This graph shows which files directly or indirectly include this file:



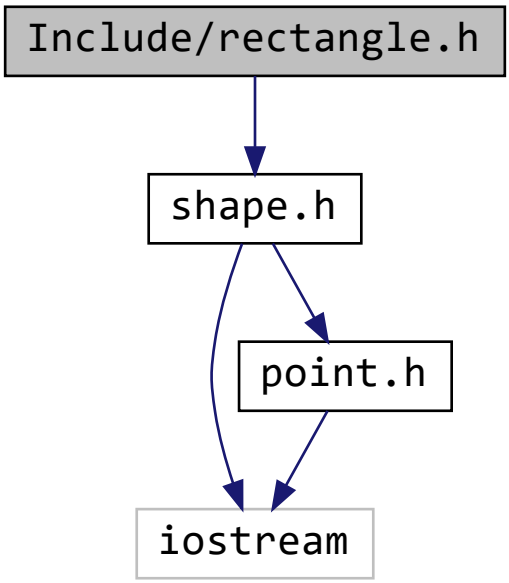
Classes

- class Point

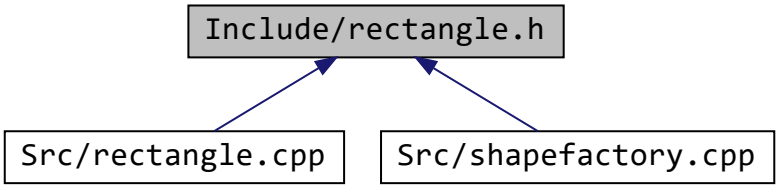
6.3 Include/rectangle.h File Reference

```
#include "shape.h"
```

Include dependency graph for rectangle.h:



This graph shows which files directly or indirectly include this file:



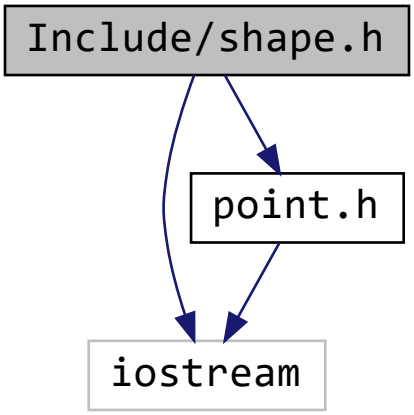
Classes

- class Rectangle

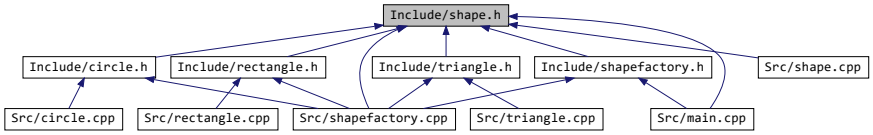
6.4 Include/shape.h File Reference

```
#include <iostream>
#include "point.h"
```

Include dependency graph for shape.h:



This graph shows which files directly or indirectly include this file:



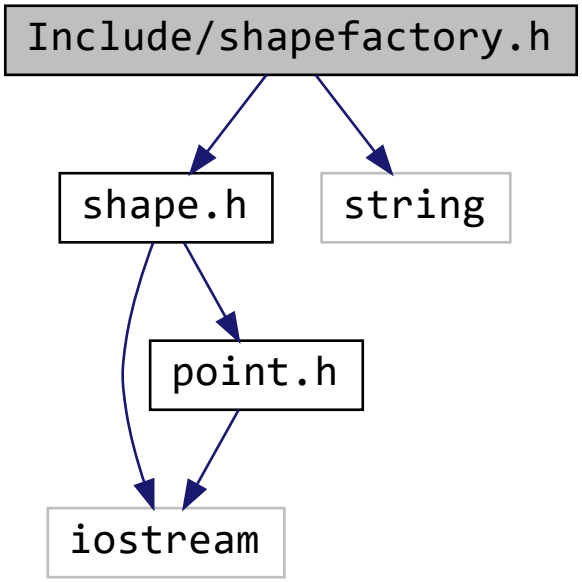
Classes

- class Shape

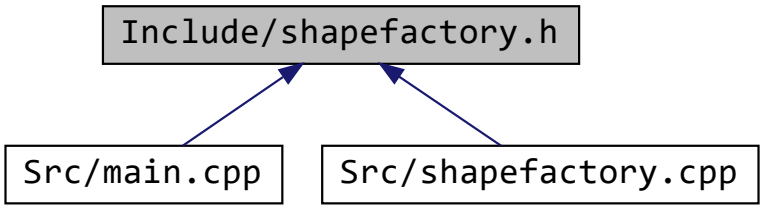
6.5 Include/shapefactory.h File Reference

```
#include "shape.h"
#include <string>
```

Include dependency graph for shapefactory.h:



This graph shows which files directly or indirectly include this file:



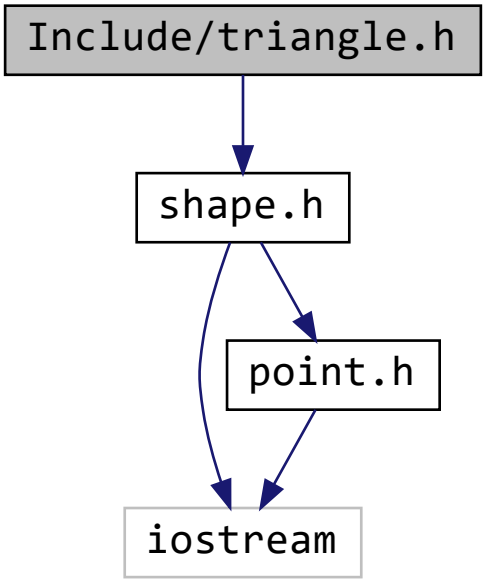
Classes

- class RandomSizeShapeFactory

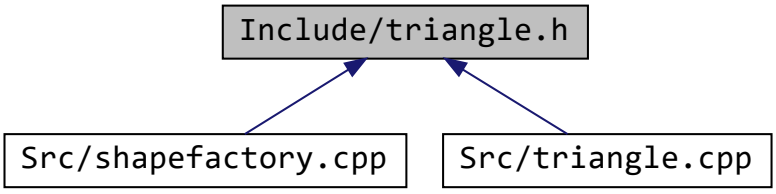
6.6 Include/triangle.h File Reference

```
#include "shape.h"
```

Include dependency graph for triangle.h:



This graph shows which files directly or indirectly include this file:



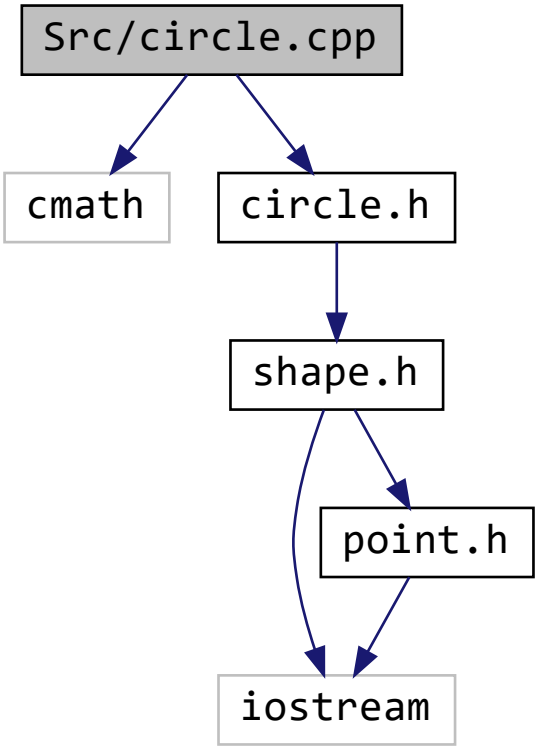
Classes

- class Triangle

6.7 Src/circle.cpp File Reference

```
#include <cmath>
#include "circle.h"
```

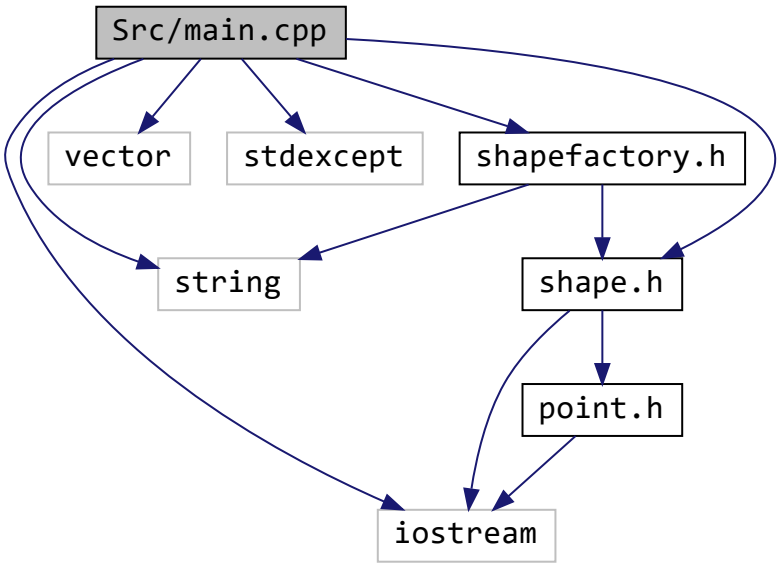
Include dependency graph for circle.cpp:



6.8 Src/main.cpp File Reference

```
#include <iostream>
#include <string>
#include <vector>
#include <stdexcept>
#include "shape.h"
#include "shapefactory.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

6.8.1 Function Documentation

6.8.1.1 main()

```
int main ( )
```

Main method

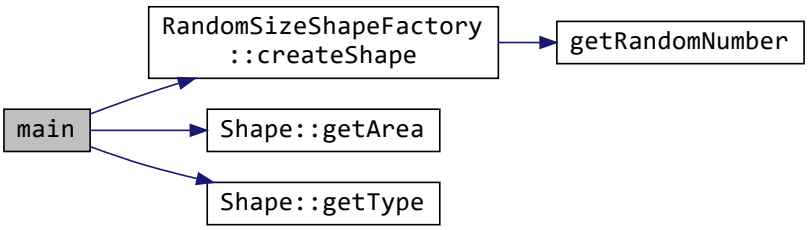
Prompt the user to enter names of shapes. For each valid shape the user enters, create a randomly sized version of that shape and put it in our list of shapes. When the user is done (they enter nothing), print out all the shapes and their areas.

Returns

The exit status of the program as an integer

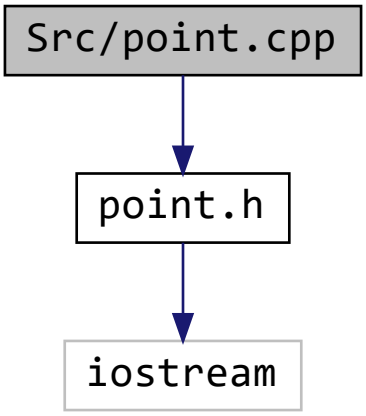
Definition at line 101 of file main.cpp.

Here is the call graph for this function:



6.9 Src/point.cpp File Reference

`#include "point.h"`
Include dependency graph for point.cpp:



Functions

- Point operator- (const Point &a)
- bool operator== (const Point &a, const Point &b)
- bool operator!= (const Point &a, const Point &b)
- ostream & operator<< (ostream &out, const Point &pt)
- istream & operator>> (istream &in, Point &pt)
- Point operator+ (const Point &a, const Point &b)
- Point operator+ (const Point &a, const int &b)

6.9.1 Function Documentation

6.9.1.1 operator"!=()

```
bool operator!= (
            const Point & a,
            const Point & b )
```

Overloaded operator to compare two Points for inequality. Two points are not equal if any of their individual member coordinates are different. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; if(a != b)

Precondition

None

Postcondition

No changes

Parameters

a	The Point on the left side of the != operation (passed by const reference)
b	The Point on the right side of the != operation (passed by const reference)

Returns

A bool value of "true" if the Points are not equal. False otherwise.

Definition at line 71 of file point.cpp.

6.9.1.2 operator+() [1/2]

```
Point operator+ (
            const Point & a,
            const int & b )
```

Overloaded + operator for adding an integer to a point. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a; int x; Point c = a + x;

Precondition

None

Postcondition

Does not change the current object

Parameters

a	The point on the left side of the + sign (passed by const reference)
b	The int on the right side of the + sign (passed by const reference)

Returns

A new point object created by adding b to each of the coordinates of a (e.g. a.x + b, a.y + b, etc.)

Definition at line 97 of file point.cpp.

6.9.1.3 operator+() [2/2]

```
Point operator+ (
            const Point & a,
            const Point & b )
```

Overloaded + operator for adding two points together. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; Point c = a + b;

Precondition

None

Postcondition

Does not change the current object

Parameters

a	The point on the left side of the + sign (passed by const reference)
b	The point on the right side of the + sign (passed by const reference)

Returns

A new point object created by adding together the individual coordinates from a & b (e.g. a.x + b.x, a.y + b.y, etc.)

Definition at line 89 of file point.cpp.

6.9.1.4 operator-()

```
Point operator- (
            const Point & a )
```

Overloaded unary operator negation operator to multiply each coordinate of the input Point by -1. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; a = -b;

Precondition

None

Postcondition

No changes

Parameters

a	The Point on the right side of the - operation (passed by const reference)

Returns

A point whose values are the negation of the values in the input Point a (e.g. if (5,6) is input, then (-5,-6) should be returned).

Definition at line 58 of file point.cpp.

6.9.1.5 operator<<()

```
ostream& operator<< (
            ostream & out,
            const Point & pt )
```

Definition at line 76 of file point.cpp.

6.9.1.6 operator==()

```
bool operator== (
            const Point & a,
            const Point & b )
```

Overloaded operator to compare two Points for equality. Two points are equal if each of their individual member coordinates are equal. Not a member function, but declared as a friend so it can access member variables. This implements:

Point a, b; if(a == b)

Precondition

None

Postcondition

No changes

Parameters

a	The Point on the left side of the == operation (passed by const reference)
b	The Point on the right side of the == operation (passed by const reference)

Returns

A bool value of "true" if the Points are equal. False otherwise.

Definition at line 66 of file point.cpp.

6.9.1.7 operator>>()

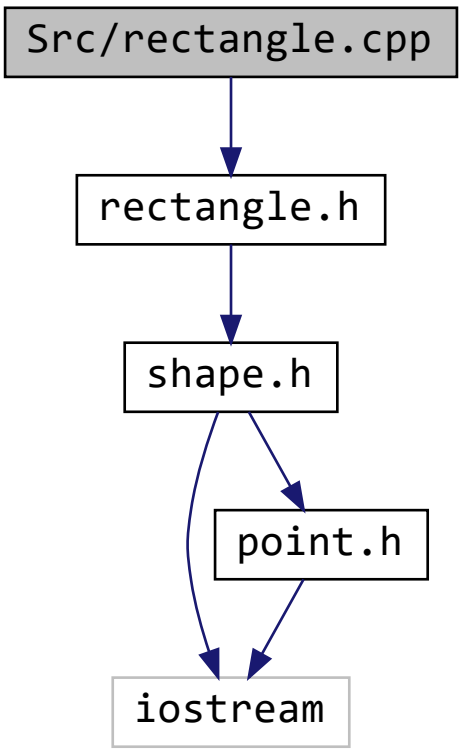
```
istream& operator>> (
            istream & in,
            Point & pt )
```

Definition at line 82 of file point.cpp.

6.10 Src/rectangle.cpp File Reference

```
#include "rectangle.h"
```

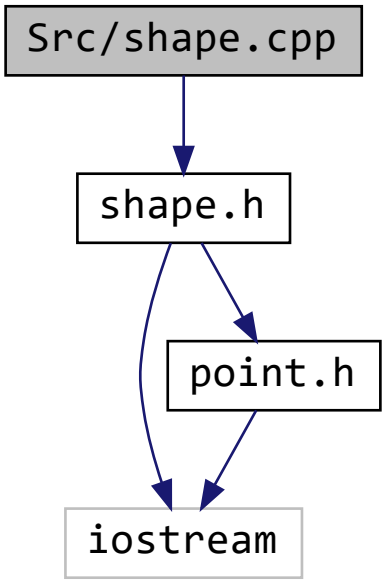
Include dependency graph for rectangle.cpp:



6.11 Src/shape.cpp File Reference

```
#include "shape.h"
```

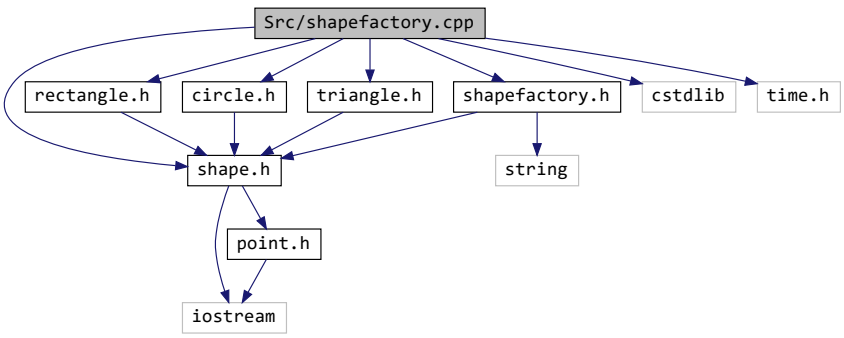
Include dependency graph for shape.cpp:



6.12 Src/shapefactory.cpp File Reference

```
#include "shapefactory.h"
#include "shape.h"
#include "rectangle.h"
#include "circle.h"
#include "triangle.h"
#include <cstdlib>
#include <time.h>
```

Include dependency graph for shapefactory.cpp:



Functions

- int getRandomNumber ()

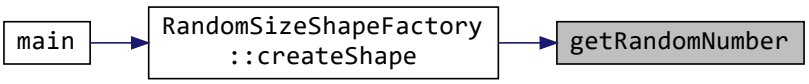
6.12.1 Function Documentation

6.12.1.1 getRandomNumber()

```
int getRandomNumber ( )
```

Definition at line 16 of file shapefactory.cpp.

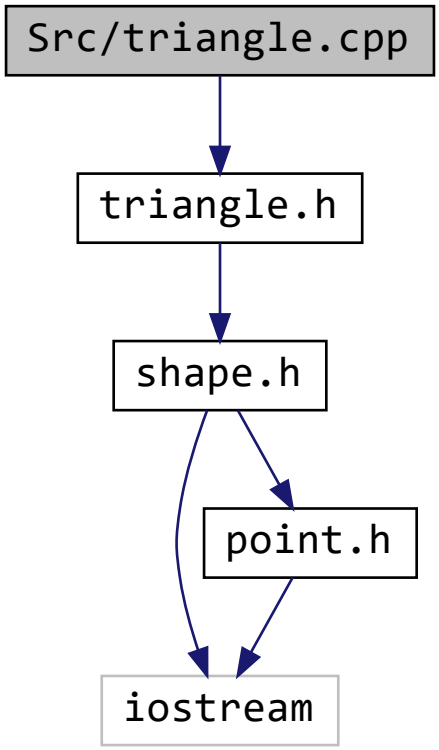
Here is the caller graph for this function:



6.13 Src/triangle.cpp File Reference

```
#include "triangle.h"
```

Include dependency graph for triangle.cpp:



Index

base
    Triangle, 43

center
    Shape, 38
Circle, 9
    Circle, 11
    getArea, 12
    getType, 12
    radius, 13
createShape
    RandomSizeShapeFactory, 26

getArea
    Circle, 12
    Rectangle, 31
    Shape, 36
    Triangle, 42
getCenter
    Shape, 36
getRandomNumber
    shapefactory.cpp, 59
getType
    Circle, 12
    Rectangle, 31
    Shape, 37
    Triangle, 42
getX
    Point, 16
getY
    Point, 16

height
    Triangle, 43

Include/circle.h, 45
Include/point.h, 46
Include/rectangle.h, 47
Include/shape.h, 48
Include/shapefactory.h, 48
Include/triangle.h, 49

length
    Rectangle, 32

main
    main.cpp, 52
main.cpp
    main, 52

operator!=
    Point, 19
    point.cpp, 54
operator<<
    Point, 22
    point.cpp, 56
operator>>
    Point, 23
    point.cpp, 57
operator+
    Point, 20, 21
    point.cpp, 54, 55
operator+=
    Point, 17
operator-
    Point, 21
    point.cpp, 55
operator=
    Point, 18
operator==
    Point, 23
    point.cpp, 56

Point, 13
    getX, 16
    getY, 16
    operator!=, 19
    operator<<, 22
    operator>>, 23
    operator+, 20, 21
    operator+=, 17
    operator-, 21
    operator=, 18
    operator==, 23
    Point, 14
    setX, 18
    setY, 19
    x, 24
    y, 24
point.cpp
    operator!=, 54
    operator<<, 56
    operator>>, 57
    operator+, 54, 55
    operator-, 55
    operator==, 56

radius
    Circle, 13
RandomSizeShapeFactory, 25
    createShape, 26

RandomSizeShapeFactory, 25
Rectangle, 27
getArea, 31
getType, 31
length, 32
Rectangle, 30
width, 32

setCenter
Shape, 37
setX
Point, 18
setY
Point, 19
Shape, 33
center, 38
getArea, 36
getCenter, 36
getType, 37
setCenter, 37
Shape, 35
shapefactory.cpp
getRandomNumber, 59
Src/circle.cpp, 50
Src/main.cpp, 51
Src/point.cpp, 53
Src/rectangle.cpp, 57
Src/shape.cpp, 58
Src/shapefactory.cpp, 58
Src/triangle.cpp, 59

Triangle, 38
base, 43
getArea, 42
getType, 42
height, 43
Triangle, 41

width
Rectangle, 32

x
Point, 24

y
Point, 24