

ShapeLib

Generated by Doxygen 1.9.3

1 AK5PC - Shape 2D	1
1.1 What you may find in this project about	1
1.2 What you DON'T find in this project	1
1.3 Project structure	2
2 LICENSE	3
3 Namespace Index	5
3.1 Namespace List	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Class Index	9
5.1 Class List	9
6 File Index	11
6.1 File List	11
7 Namespace Documentation	13
7.1 ShapeLib Namespace Reference	13
7.1.1 Function Documentation	13
7.1.1.1 operator<<() [1/3]	13
7.1.1.2 operator<<() [2/3]	14
7.1.1.3 operator<<() [3/3]	14
8 Class Documentation	15
8.1 ShapeLib::Circle Class Reference	15
8.1.1 Detailed Description	16
8.1.2 Constructor & Destructor Documentation	16
8.1.2.1 Circle() [1/2]	16
8.1.2.2 Circle() [2/2]	17
8.1.3 Member Function Documentation	17
8.1.3.1 getBorderLength()	17
8.1.3.2 getCenterPoint()	18
8.1.3.3 getCircleVolume()	18
8.1.3.4 getRadius()	18
8.1.3.5 printInfo()	18
8.1.3.6 recalculateDim()	19
8.1.3.7 setCenterPoint()	19
8.1.3.8 setRadius()	19
8.1.4 Member Data Documentation	19
8.1.4.1 borderLength	19
8.1.4.2 centerPoint	20
8.1.4.3 circleVolume	20

8.1.4.4 dimsValid	20
8.1.4.5 radius	20
8.2 ShapeLib::Point Class Reference	21
8.2.1 Detailed Description	21
8.2.2 Constructor & Destructor Documentation	21
8.2.2.1 Point()	21
8.2.3 Member Function Documentation	22
8.2.3.1 horizontalDistance()	22
8.2.3.2 verticalDistance()	22
8.2.4 Member Data Documentation	23
8.2.4.1 x	23
8.2.4.2 y	23
8.3 ShapeLib::Rectangle Class Reference	23
8.3.1 Detailed Description	24
8.3.2 Constructor & Destructor Documentation	25
8.3.2.1 Rectangle() [1/2]	25
8.3.2.2 Rectangle() [2/2]	25
8.3.3 Member Function Documentation	26
8.3.3.1 getHeight()	26
8.3.3.2 getP1()	26
8.3.3.3 getP2()	26
8.3.3.4 getWidth()	27
8.3.3.5 printInfo()	27
8.3.3.6 recalculateDim()	27
8.3.3.7 setP1()	27
8.3.3.8 setP2()	28
8.3.4 Member Data Documentation	28
8.3.4.1 dimValid	28
8.3.4.2 height	28
8.3.4.3 p1	29
8.3.4.4 p2	29
8.3.4.5 width	29
8.4 ShapeLib::Shape Class Reference	29
8.4.1 Detailed Description	30
8.4.2 Constructor & Destructor Documentation	30
8.4.2.1 Shape()	30
8.4.3 Member Function Documentation	30
8.4.3.1 getID()	30
8.4.3.2 printInfo()	31
8.4.4 Member Data Documentation	31
8.4.4.1 ID	31

9 File Documentation	33
9.1 CMakeCCompilerId.c File Reference	33
9.1.1 Macro Definition Documentation	33
9.1.1.1 __has_include	34
9.1.1.2 ARCHITECTURE_ID	34
9.1.1.3 C_VERSION	34
9.1.1.4 COMPILER_ID	34
9.1.1.5 DEC	34
9.1.1.6 HEX	34
9.1.1.7 PLATFORM_ID	35
9.1.1.8 STRINGIFY	35
9.1.1.9 STRINGIFY_HELPER	35
9.1.2 Function Documentation	35
9.1.2.1 main()	35
9.1.3 Variable Documentation	35
9.1.3.1 info_arch	35
9.1.3.2 info_compiler	35
9.1.3.3 info_language_extensions_default	36
9.1.3.4 info_language_standard_default	36
9.1.3.5 info_platform	36
9.2 CMakeCXXCompilerId.cpp File Reference	36
9.2.1 Macro Definition Documentation	37
9.2.1.1 __has_include	37
9.2.1.2 ARCHITECTURE_ID	37
9.2.1.3 COMPILER_ID	37
9.2.1.4 CXX_STD	37
9.2.1.5 DEC	37
9.2.1.6 HEX	38
9.2.1.7 PLATFORM_ID	38
9.2.1.8 STRINGIFY	38
9.2.1.9 STRINGIFY_HELPER	38
9.2.2 Function Documentation	38
9.2.2.1 main()	38
9.2.3 Variable Documentation	38
9.2.3.1 info_arch	39
9.2.3.2 info_compiler	39
9.2.3.3 info_language_extensions_default	39
9.2.3.4 info_language_standard_default	39
9.2.3.5 info_platform	39
9.3 LICENSE.MD File Reference	39
9.4 main.cpp File Reference	39
9.4.1 Function Documentation	40

9.4.1.1 main()	40
9.5 README.md File Reference	40
9.6 circle.h File Reference	40
9.7 circle.h	41
9.8 point.h File Reference	41
9.9 point.h	42
9.10 rectangle.h File Reference	42
9.11 rectangle.h	43
9.12 shape.h File Reference	43
9.13 shape.h	44
9.14 circle.cpp File Reference	44
9.15 point.cpp File Reference	44
9.16 rectangle.cpp File Reference	45
9.17 shape.cpp File Reference	45
Index	47

Chapter 1

AK5PC - Shape 2D

Demonstration project for AK5PC course.

1.1 What you may find in this project about

Simplified demonstration of selected C++ features:

- Classes, abstract classes and basic inheritance
- Class members, member methods, functions and pure virtual functions
- Namespaces
- Simple constructors and constructor delegation
- Function overriding
- Simple operator overriding
- References and their usage
- Basic output stream usage
- etc.

1.2 What you DON'T find in this project

Because this project is most about C++ syntax and fundamental principles and is provided as supplemental material for the C++ classes, it's not focused on following things

- C++ best practices descriptions and demonstrations
- Application architecture recommendations
- Optimization guides

1.3 Project structure

The project is composed by CMake tool usage and is separated into two separate project. The first project is focused on [ShapeLib](#) and is placed with corresponding folder. It is configured as a shared library. The second project contains only short demonstration of the library written in [main.cpp](#). The file and configuration is placed in root folder.

```
..
.
+-- doc # project documentation
|   +-- output
|   |   +-- html      # generated documentation in HTML format
|   |   +-- latex     # generated documentation in LaTeX format (pdf)
|   +-- proj          # Doxygen project configuration
|
+-- ShapeLib          # the ShapeLib library project
|   +-- include       # header files
|   +-- src           # source files
|   +-- CMakeLists.txt # library project configuration
|
+-- main.cpp          # the main demonstration file
+-- CMakeLists.txt    # the configuration for demonsration project
```


Chapter 2

LICENSE

MIT License

Copyright (c) 2022 Ing. Peter Janku, Ph.D.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ShapeLib	13
------------------------------------	----

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ShapeLib::Point	21
ShapeLib::Shape	29
ShapeLib::Circle	15
ShapeLib::Rectangle	23

Chapter 5

Class Index

5.1 Class List

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

ShapeLib::Circle	
Simple class representing one Circle	15
ShapeLib::Point	
Simple class representing one point in cartesian coordinates	21
ShapeLib::Rectangle	
Simple class representing a rectangle defined by two points	23
ShapeLib::Shape	
Interface for any 2D shape in this library	29

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

CMakeCCompilerId.c	33
CMakeCXXCompilerId.cpp	36
main.cpp	39
circle.h	40
point.h	41
rectangle.h	42
shape.h	43
circle.cpp	44
point.cpp	44
rectangle.cpp	45
shape.cpp	45

Chapter 7

Namespace Documentation

7.1 ShapeLib Namespace Reference

Classes

- class [Circle](#)
Simple class representing one [Circle](#).
- class [Point](#)
Simple class representing one point in cartesian coordinates.
- class [Rectangle](#)
Simple class representing a rectangle defined by two points.
- class [Shape](#)
Interface for any 2D shape in this library.

Functions

- `std::ostream & operator<< (std::ostream &stream, Circle &circle)`
Stream operator for [Circle](#) class.
- `std::ostream & operator<< (std::ostream &stream, const Point &point)`
Stream operator for the [Point](#) class.
- `std::ostream & operator<< (std::ostream &stream, Rectangle &rect)`
Stream operator for [Rectangle](#) class.

7.1.1 Function Documentation

7.1.1.1 `operator<<()` [1/3]

```
std::ostream & ShapeLib::operator<< (  
    std::ostream & stream,  
    Circle & circle )
```

Stream operator for [Circle](#) class.

Parameters

<i>stream</i>	output stream
<i>circle</i>	the Circle object to be printed out

Returns

the original stream

7.1.1.2 operator<<() [2/3]

```
std::ostream & ShapeLib::operator<< (
    std::ostream & stream,
    const Point & point )
```

Stream operator for the [Point](#) class.

Parameters

<i>stream</i>	output stream
<i>point</i>	the Point object to be printed out

Returns

the original stream

7.1.1.3 operator<<() [3/3]

```
std::ostream & ShapeLib::operator<< (
    std::ostream & stream,
    ShapeLib::Rectangle & rect )
```

Stream operator for [Rectangle](#) class.

Parameters

<i>stream</i>	output stream
<i>rect</i>	the Rectangle object to be printed out

Returns

the original output stream

Chapter 8

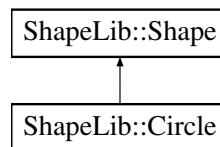
Class Documentation

8.1 ShapeLib::Circle Class Reference

Simple class representing one [Circle](#).

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

Inheritance diagram for ShapeLib::Circle:



Public Member Functions

- [Circle](#) (int [ID](#), int centerX, int centerY, int [radius](#))
The circle constructor.
- [Circle](#) (int [ID](#), const [Point](#) ¢er, int [radius](#))
The circle constructor.
- void [setCenterPoint](#) (const [Point](#) ¢erPoint)
Setter for the center point.
- void [setRadius](#) (int [radius](#))
Setter for the radius. It also invalidate internal dimensions.
- void [printInfo](#) () const override
Print out tha basic circle info (center point and radius)
- const [Point](#) & [getCenterPoint](#) () const
Getter for the center point.
- int [getRadius](#) () const
Getter for the radius dimension.
- double [getBorderLength](#) ()
This function returns the length of circle border. If the dimension is not valid, it is recalculated at first.
- double [getCircleVolume](#) ()
this function returns the volume of circle area. If the dimensions is not valid, it is recalculated at first.

Protected Member Functions

- void [recalculateDim](#) ()
Recalculates the circle border length and area volume.

Protected Attributes

- [Point](#) [centerPoint](#)
Point defining the center of the circle.
- int [radius](#)
Dimension of circle radius.
- double [borderLength](#) = 0
Temporary storage of circle border length. Don't have to be correct!!!!
- double [circleVolume](#) = 0
Temporary storage of circle area volume. Don't have to be correct!!!!
- bool [dimsValid](#) = false
Information if circle area volume and border length is valid or not.

8.1.1 Detailed Description

Simple class representing one [Circle](#).

The circle is defined by a centre point and radius.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 [Circle\(\)](#) [1/2]

```
ShapeLib::Circle::Circle (
    int ID,
    int centerX,
    int centerY,
    int radius ) [inline]
```

The circle constructor.

This constructor is calling the second one with modified parameters. This can be done since C++11 standard and it's called constructor delegation.

Parameters

<i>ID</i>	Circle ID
<i>centerX</i>	The X coordinate of circle center
<i>centerY</i>	The Y coordinate of circle center
<i>radius</i>	The circle radius

8.1.2.2 Circle() [2/2]

```
ShapeLib::Circle::Circle (
    int ID,
    const Point & center,
    int radius ) [inline]
```

The circle constructor.

This constructor stores all [Circle](#)'s internal members as well as members defined in [Shape](#) class. It also calls constructor of [Shape](#) class. Finally, it recalculates the dimensions of the rectangle.

See also

[recalculateDim\(\)](#)

Parameters

<i>ID</i>	Circle ID
<i>center</i>	Circle center point

See also

[Point](#)

Parameters

<i>radius</i>	Circle radius
---------------	-------------------------------

8.1.3 Member Function Documentation

8.1.3.1 getBorderLength()

```
double ShapeLib::Circle::getBorderLength ( )
```

This function returns the length of circle border. If the dimension is not valid, it is recalculated at first.

See also

[recalculateDim\(\)](#)

Returns

current length of circle border.

8.1.3.2 getCenterPoint()

```
const Point & ShapeLib::Circle::getCenterPoint ( ) const [inline]
```

Getter for the center point.

Returns

Center point

8.1.3.3 getCircleVolume()

```
double ShapeLib::Circle::getCircleVolume ( )
```

this function returns the volume of circle area. If the dimensions is not valid, it is recalculated at first.

See also

[recalculateDim\(\)](#)

Returns

8.1.3.4 getRadius()

```
int ShapeLib::Circle::getRadius ( ) const [inline]
```

Getter for the radius dimension.

Returns

Size of the circle's radius

8.1.3.5 printInfo()

```
void ShapeLib::Circle::printInfo ( ) const [override], [virtual]
```

Print out the basic circle info (center point and radius)

Implements [ShapeLib::Shape](#).

8.1.3.6 recalculateDim()

```
void ShapeLib::Circle::recalculateDim ( ) [protected]
```

Recalculates the circle border length and area volume.

Based on radiusthis function recalculates the circle are volume and border lenght and store it in internal temporary variables.

See also

[borderLength](#)

[circleVolume](#)

8.1.3.7 setCenterPoint()

```
void ShapeLib::Circle::setCenterPoint (
    const Point & centerPoint )
```

Setter for the center point.

Parameters

<i>centerPoint</i>	Point defining the circle center
--------------------	--

8.1.3.8 setRadius()

```
void ShapeLib::Circle::setRadius (
    int radius )
```

Setter for the radius. It also invalidate internal dimensions.

Parameters

<i>radius</i>	the circle radius
---------------	-------------------

8.1.4 Member Data Documentation

8.1.4.1 borderLength

```
double ShapeLib::Circle::borderLength = 0 [protected]
```

Temporary storage of circle border length. Don't have to be correct!!!!

See also

[dimsValid](#)

8.1.4.2 centerPoint

```
Point ShapeLib::Circle::centerPoint [protected]
```

[Point](#) defining the center of the circle.

8.1.4.3 circleVolume

```
double ShapeLib::Circle::circleVolume = 0 [protected]
```

Temporary storage of circle area volume. Don't have to be correct!!!!

See also

[dimsValid](#)

8.1.4.4 dimsValid

```
bool ShapeLib::Circle::dimsValid = false [protected]
```

Information if circle area volume and border length is valido or not.

See also

[recalculateDim\(\)](#)

8.1.4.5 radius

```
int ShapeLib::Circle::radius [protected]
```

Dimension of circle radius.

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

- [circle.h](#)
- [circle.cpp](#)

8.2 ShapeLib::Point Class Reference

Simple class representing one point in cartesian coordinates.

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

Public Member Functions

- [Point](#) (int [x](#), int [y](#))
First parametric constructor.

Static Public Member Functions

- static int [horizontalDistance](#) (const [Point](#) &p1, const [Point](#) &p2)
Static function for horizontal distance of two point estimation.
- static int [verticalDistance](#) (const [Point](#) &p1, const [Point](#) &p2)
Static function for vertical distance of two point estimation.

Public Attributes

- int [x](#)
coordinate X
- int [y](#)
coordinate Y

8.2.1 Detailed Description

Simple class representing one point in cartesian coordinates.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Point()

```
ShapeLib::Point::Point (
    int x,
    int y ) [inline]
```

First parametric constructor.

Because at least one parametric constructor is created, the compiler doesn't create the default constructor.

Parameters

x	coordinate
y	coordinate

8.2.3 Member Function Documentation

8.2.3.1 horizontalDistance()

```
int ShapeLib::Point::horizontalDistance (
    const Point & p1,
    const Point & p2 ) [static]
```

Static function for horizontal distance of two point estimation.

See also

[Point](#)

Parameters

<i>p1</i>	first input point
<i>p2</i>	second input point

Returns

horizontal distance between provided pints

8.2.3.2 verticalDistance()

```
int ShapeLib::Point::verticalDistance (
    const Point & p1,
    const Point & p2 ) [static]
```

Static function for vertical distance of two point estimation.

See also

[Point](#)

Parameters

<i>p1</i>	first input point
<i>p2</i>	second input point

Returns

vertical distance between provided points

8.2.4 Member Data Documentation

8.2.4.1 x

```
int ShapeLib::Point::x
```

coordinate X

8.2.4.2 y

```
int ShapeLib::Point::y
```

coordinate Y

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

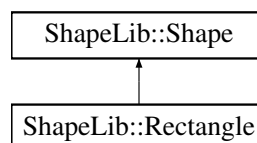
- [point.h](#)
- [point.cpp](#)

8.3 ShapeLib::Rectangle Class Reference

Simple class representing a rectangle defined by two points.

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

Inheritance diagram for ShapeLib::Rectangle:



Public Member Functions

- [Rectangle](#) (int id, int x1, int y1, int x2, int y2)
The rectangle constructor.
- [Rectangle](#) (int id, const [Point](#) &p1, const [Point](#) &p2)
The main rectangle constructor.
- void [printInfo](#) () const override
Print out basic rectangle info (ID, P2 and P2)
- int [getWidth](#) ()
This function returns the current width of the rectangle. If the already calculated width is incorrect(the rectangle points were redefined), the dimensions are recalculated at first.
- int [getHeight](#) ()
This function returns the current height of the rectangle. If the already calculated height is incorrect(the rectangle points were redefined), the dimensions are recalculated at first.
- const [Point](#) & [getP1](#) () const
Return the first point in 2D space, which defines the rectangle.
- const [Point](#) & [getP2](#) () const
Return the second point in 2D space, which defines the rectangle.
- void [setP1](#) (const [Point](#) &p1)
Setter for the first point of the rectangle.
- void [setP2](#) (const [Point](#) &p2)
Setter for the second point of the rectangle.

Protected Member Functions

- void [recalculateDim](#) ()
Recalculates the rectangle width and height.

Protected Attributes

- [Point](#) p1
The first point in 2D space.
- [Point](#) p2
The second point in 2D space.
- int [width](#)
Temporary storage of rectangle width. Don't have to be correct!!!!
- int [height](#)
Temporary storage of rectangle height. Don't have to be correct!!!!
- bool [dimValid](#) = false
Information if width and height is correct or not.

8.3.1 Detailed Description

Simple class representing a rectangle defined by two points.

See also

[Point](#)

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Rectangle() [1/2]

```
ShapeLib::Rectangle::Rectangle (  
    int id,  
    int x1,  
    int y1,  
    int x2,  
    int y2 ) [inline]
```

The rectangle constructor.

This constructor is calling the second one with modified parameters. This can be done since C++11 standard and it's called constructor delegation.

Parameters

<i>id</i>	Rectangle ID
<i>x1</i>	X coordinate of first rectangle's point
<i>y1</i>	Y coordinate of first rectangle's point
<i>x2</i>	X coordinate of second rectangle's point
<i>y2</i>	Y coordinate of second rectangle's point

8.3.2.2 Rectangle() [2/2]

```
ShapeLib::Rectangle::Rectangle (  
    int id,  
    const Point & p1,  
    const Point & p2 ) [inline]
```

The main rectangle constructor.

This constructor stores all [Rectangle](#)'s internal members as well as members defined in [Shape](#) class. It also calls constructor of [Shape](#) class. Finally, it recalculates the dimensions of the rectangle.

See also

[recalculateDim\(\)](#)

Parameters

<i>id</i>	Rectangle ID
<i>p1</i>	First point defining the rectangle.
<i>p2</i>	Second point defining the rectangle.

8.3.3 Member Function Documentation

8.3.3.1 getHeight()

```
int ShapeLib::Rectangle::getHeight ( )
```

This function returns the current height of the rectangle. If the already calculated height is incorrect (the rectangle points were redefined), the dimensions are recalculated at first.

See also

[recalculateDim\(\)](#)

Returns

current height of the rectangle.

8.3.3.2 getP1()

```
const Point & ShapeLib::Rectangle::getP1 ( ) const [inline]
```

Return the first point in 2D space, which defines the rectangle.

Returns

Const reference to the point

See also

[Point](#)

8.3.3.3 getP2()

```
const Point & ShapeLib::Rectangle::getP2 ( ) const [inline]
```

Return the second point in 2D space, which defines the rectangle.

Returns

Const reference to the point

See also

[Point](#)

8.3.3.4 getWidth()

```
int ShapeLib::Rectangle::getWidth ( )
```

This function returns the current width of the rectangle. If the already calculated width is incorrect(the rectangle points were redefined), the dimensions are recalculated at first.

See also

[recalculateDim\(\)](#)

Returns

current width of the rectangle.

8.3.3.5 printInfo()

```
void ShapeLib::Rectangle::printInfo ( ) const [override], [virtual]
```

Print out basic rectangle info (ID, P2 and P2)

Implements [ShapeLib::Shape](#).

8.3.3.6 recalculateDim()

```
void ShapeLib::Rectangle::recalculateDim ( ) [protected]
```

Recalculates the rectangle width and height.

Based on p1 and p2 this function recalculates the rectangle width and height and store it in internal temporary variables.

See also

[width](#)

[height](#)

8.3.3.7 setP1()

```
void ShapeLib::Rectangle::setP1 (
    const Point & p1 )
```

Setter for the first point of the rectangle.

Parameters

<i>p1</i>	point defining the rectangle.
-----------	-------------------------------

8.3.3.8 setP2()

```
void ShapeLib::Rectangle::setP2 (
    const Point & p2 )
```

Setter for the second point of the rectangle.

Parameters

<i>p2</i>	point defining the rectangle.
-----------	-------------------------------

8.3.4 Member Data Documentation**8.3.4.1 dimValid**

```
bool ShapeLib::Rectangle::dimValid = false [mutable], [protected]
```

Information if width and height is correct or not.

See also

[recalculateDim\(\)](#)

8.3.4.2 height

```
int ShapeLib::Rectangle::height [mutable], [protected]
```

Temporary storage of rectangle height. Don't have to be correct!!!!

See also

[dimValid](#)

8.3.4.3 p1

`Point ShapeLib::Rectangle::p1` [protected]

The first point in 2D space.

8.3.4.4 p2

`Point ShapeLib::Rectangle::p2` [protected]

The second point in 2D space.

8.3.4.5 width

`int ShapeLib::Rectangle::width` [mutable], [protected]

Temporary storage of rectangle width. Don't have to be correct!!!!

See also

[dimValid](#)

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

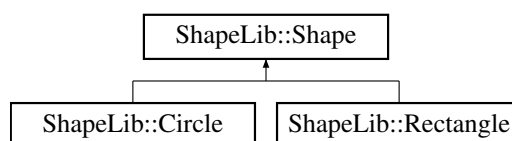
- [rectangle.h](#)
- [rectangle.cpp](#)

8.4 ShapeLib::Shape Class Reference

Interface for any 2D shape in this library.

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

Inheritance diagram for ShapeLib::Shape:



Public Member Functions

- [Shape](#) (int id)
Constructor of [Shape](#) class.
- virtual void [printInfo](#) () const =0
Virtual function for printing information about the shape into the console.
- int [getID](#) () const
Getter for shape's ID.

Protected Attributes

- int [ID](#)
internal ID of each shape

8.4.1 Detailed Description

Interface for any 2D shape in this library.

This function implements the basic ID management as well as virtual function [printInfo\(\)](#).

8.4.2 Constructor & Destructor Documentation

8.4.2.1 Shape()

```
ShapeLib::Shape::Shape (  
    int id ) [inline], [explicit]
```

Constructor of [Shape](#) class.

Parameters

<i>id</i>	an ID of the shape
-----------	--------------------

8.4.3 Member Function Documentation

8.4.3.1 getID()

```
int ShapeLib::Shape::getID ( ) const [inline]
```

Getter for shape's ID.

Returns

the ID of the shape

8.4.3.2 printInfo()

```
virtual void ShapeLib::Shape::printInfo ( ) const [pure virtual]
```

Virtual function for printing information about the shape into the console.

Implemented in [ShapeLib::Circle](#), and [ShapeLib::Rectangle](#).

8.4.4 Member Data Documentation**8.4.4.1 ID**

```
int ShapeLib::Shape::ID [protected]
```

internal ID of each shape

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

- [shape.h](#)

Chapter 9

File Documentation

9.1 CMakeCCompilerId.c File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define C_VERSION`

Functions

- `int main (int argc, char *argv[])`

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"`
- `char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"`
- `char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"`
- `const char * info_language_standard_default`
- `const char * info_language_extensions_default`

9.1.1 Macro Definition Documentation

9.1.1.1 `__has_include`

```
#define __has_include(  
    x ) 0
```

9.1.1.2 `ARCHITECTURE_ID`

```
#define ARCHITECTURE_ID
```

9.1.1.3 `C_VERSION`

```
#define C_VERSION
```

9.1.1.4 `COMPILER_ID`

```
#define COMPILER_ID ""
```

9.1.1.5 `DEC`

```
#define DEC(  
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

9.1.1.6 `HEX`

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```


9.1.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

9.1.1.8 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY_HELPER(X)
```

9.1.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

9.1.2 Function Documentation

9.1.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

9.1.3 Variable Documentation

9.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

9.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

9.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["  
  "OFF"  
"]"
```

9.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=  
  "INFO" ":" "standard_default[" C_VERSION "]"
```

9.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

9.2 CMakeCXXCompilerId.cpp File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define CXX_STD __cplusplus`

Functions

- `int main (int argc, char *argv[])`

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"`
- `char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"`
- `char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"`
- `const char * info_language_standard_default`
- `const char * info_language_extensions_default`

9.2.1 Macro Definition Documentation

9.2.1.1 `__has_include`

```
#define __has_include(  
    x ) 0
```

9.2.1.2 `ARCHITECTURE_ID`

```
#define ARCHITECTURE_ID
```

9.2.1.3 `COMPILER_ID`

```
#define COMPILER_ID ""
```

9.2.1.4 `CXX_STD`

```
#define CXX_STD __cplusplus
```

9.2.1.5 `DEC`

```
#define DEC(  
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \  
( '0' + ((n) / 1000000) % 10), \  
( '0' + ((n) / 100000) % 10), \  
( '0' + ((n) / 10000) % 10), \  
( '0' + ((n) / 1000) % 10), \  
( '0' + ((n) / 100) % 10), \  
( '0' + ((n) / 10) % 10), \  
( '0' + ((n) % 10))
```

9.2.1.6 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \  
( '0' + ((n)>>24 & 0xF)), \  
( '0' + ((n)>>20 & 0xF)), \  
( '0' + ((n)>>16 & 0xF)), \  
( '0' + ((n)>>12 & 0xF)), \  
( '0' + ((n)>>8  & 0xF)), \  
( '0' + ((n)>>4  & 0xF)), \  
( '0' + ((n)    & 0xF))
```

9.2.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

9.2.1.8 STRINGIFY

```
#define STRINGIFY(  
    X )  STRINGIFY_HELPER(X)
```

9.2.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X )  #X
```

9.2.2 Function Documentation

9.2.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

9.2.3 Variable Documentation

9.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

9.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

9.2.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["  
  "OFF"  
"]"
```

9.2.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["  
  "98"  
"]"
```

9.2.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

9.3 LICENSE.MD File Reference

9.4 main.cpp File Reference

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

Functions

- int [main](#) ()

9.4.1 Function Documentation

9.4.1.1 main()

```
int main ( )
```

9.5 README.md File Reference

9.6 circle.h File Reference

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

Classes

- class [ShapeLib::Circle](#)
Simple class representing one [Circle](#).

Namespaces

- namespace [ShapeLib](#)

Functions

- std::ostream & [ShapeLib::operator<<](#) (std::ostream &stream, Circle &circle)
Stream operator for [Circle](#) class.

9.7 circle.h

[Go to the documentation of this file.](#)

```

1 //
2 // Created by Peter Janků on 10.10.2022.
3 //
4
5 #ifndef SHAPE2D_CIRCLE_H
6 #define SHAPE2D_CIRCLE_H
7
8 #include "shape.h"
9 #include "point.h"
10
11 namespace ShapeLib {
12     class Circle : public Shape {
13     public:
14         Circle(int ID, int centerX, int centerY, int radius) : Circle(ID, Point(centerX, centerY),
15             radius) {}
16
17         Circle(int ID, const Point &center, int radius) : Shape(ID), centerPoint(center), radius(radius)
18         {}
19
20         void setCenterPoint(const Point &centerPoint);
21
22         void setRadius(int radius);
23
24         void printInfo() const override;
25
26         const Point &getCenterPoint() const {
27             return centerPoint;
28         }
29
30         int getRadius() const {
31             return radius;
32         }
33
34         double getBorderLength();
35         double getCircleVolume();
36
37     protected:
38         void recalculateDim();
39
40         Point centerPoint;
41         int radius;
42         double borderLength = 0;
43         double circleVolume = 0;
44         bool dimsValid = false;
45     };
46
47     std::ostream &operator<<(std::ostream &stream, Circle &circle);
48 }
49
50 // ShapeLib
51
52 #endif //SHAPE2D_CIRCLE_H

```

9.8 point.h File Reference

```
#include <iostream>
```

Classes

- class [ShapeLib::Point](#)
Simple class representing one point in cartesian coordinates.

Namespaces

- namespace [ShapeLib](#)

Functions

- `std::ostream & ShapeLib::operator<< (std::ostream &stream, const Point &point)`
Stream operator for the *Point* class.

9.9 point.h

[Go to the documentation of this file.](#)

```

1 //
2 // Created by Peter Janků on 08.10.2022.
3 //
4
5 #ifndef SHAPE2D_POINT_H
6 #define SHAPE2D_POINT_H
7
8 #include <iostream>
9
10 namespace ShapeLib {
11     class Point {
12     public:
13
14         Point(int x, int y) : x(x), y(y) {}
15
16         static int horizontalDistance(const Point &p1, const Point &p2);
17
18         static int verticalDistance(const Point &p1, const Point &p2);
19
20         int x;
21         int y;
22     };
23
24     std::ostream &operator<<(std::ostream& stream, const Point &point);
25
26 }
27
28 #endif //SHAPE2D_POINT_H

```

9.10 rectangle.h File Reference

```

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

```

Classes

- class `ShapeLib::Rectangle`
Simple class representing a rectangle defined by two points.

Namespaces

- namespace `ShapeLib`

Functions

- `std::ostream & ShapeLib::operator<< (std::ostream &stream, Rectangle &rect)`
Stream operator for *Rectangle* class.

9.11 rectangle.h

[Go to the documentation of this file.](#)

```

1 //
2 // Created by Peter Janků on 08.10.2022.
3 //
4
5 #ifndef SHAPE2D_RECTANGLE_H
6 #define SHAPE2D_RECTANGLE_H
7
8 #include <iostream>
9 #include "shape.h"
10 #include "point.h"
11
12 namespace ShapeLib {
13     class Rectangle : public Shape {
14     public:
15
16         Rectangle(int id, int x1, int y1, int x2, int y2) : Rectangle(id, Point(x1, y1), Point(x2, y2))
17         {}
18
19         Rectangle(int id, const Point &p1, const Point &p2) : Shape(id, p1(p1), p2(p2)) {
20             recalculateDim(); }
21
22         void printInfo() const override;
23
24         int getWidth();
25
26         int getHeight();
27
28         const Point &getP1() const { return p1; }
29
30         const Point &getP2() const { return p2; }
31
32         void setP1(const Point &p1);
33
34         void setP2(const Point &p2);
35
36     protected:
37
38         void recalculateDim();
39         Point p1;
40         Point p2;
41         mutable int width;
42         mutable int height;
43         mutable bool dimValid = false;
44     };
45
46     std::ostream &operator<<(std::ostream &stream, Rectangle &rect);
47 }
48
49 #endif //SHAPE2D_RECTANGLE_H

```

9.12 shape.h File Reference

```
#include "point.h"
```

Classes

- class [ShapeLib::Shape](#)
Interface for any 2D shape in this library.

Namespaces

- namespace [ShapeLib](#)

9.13 shape.h

[Go to the documentation of this file.](#)

```

1 //
2 // Created by Peter Janků on 08.10.2022.
3 //
4
5 #ifndef SHAPE2D_SHAPE_H
6 #define SHAPE2D_SHAPE_H
7
8 #include "point.h"
9
10 namespace ShapeLib {
11     class Shape {
12     public:
13
14         explicit Shape(int id) : ID(id) {}
15
16         virtual void printInfo() const = 0;
17
18         int getID() const { return ID; }
19
20     protected:
21         int ID;
22     };
23 } // ShapeLib
24
25 #endif //SHAPE2D_SHAPE_H

```

9.14 circle.cpp File Reference

```

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

```

Namespaces

- namespace [ShapeLib](#)

Functions

- `std::ostream & ShapeLib::operator<< (std::ostream &stream, Circle &circle)`
Stream operator for [Circle](#) class.

9.15 point.cpp File Reference

```

#include "../include/point.h"
#include <cmath>

```

Namespaces

- namespace [ShapeLib](#)

Functions

- `std::ostream & ShapeLib::operator<< (std::ostream &stream, const Point &point)`
Stream operator for the [Point](#) class.

9.16 rectangle.cpp File Reference

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

9.17 shape.cpp File Reference

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

Namespaces

- namespace [ShapeLib](#)

Index

- `__has_include`
 - `CMakeCCompilerId.c`, [33](#)
 - `CMakeCXXCompilerId.cpp`, [37](#)
- `ARCHITECTURE_ID`
 - `CMakeCCompilerId.c`, [34](#)
 - `CMakeCXXCompilerId.cpp`, [37](#)
- `borderLength`
 - `ShapeLib::Circle`, [19](#)
- `C_VERSION`
 - `CMakeCCompilerId.c`, [34](#)
- `centerPoint`
 - `ShapeLib::Circle`, [20](#)
- `Circle`
 - `ShapeLib::Circle`, [16](#), [17](#)
- `circle.cpp`, [44](#)
- `circle.h`, [40](#), [41](#)
- `circleVolume`
 - `ShapeLib::Circle`, [20](#)
- `CMakeCCompilerId.c`, [33](#)
 - `__has_include`, [33](#)
 - `ARCHITECTURE_ID`, [34](#)
 - `C_VERSION`, [34](#)
 - `COMPILER_ID`, [34](#)
 - `DEC`, [34](#)
 - `HEX`, [34](#)
 - `info_arch`, [35](#)
 - `info_compiler`, [35](#)
 - `info_language_extensions_default`, [35](#)
 - `info_language_standard_default`, [36](#)
 - `info_platform`, [36](#)
 - `main`, [35](#)
 - `PLATFORM_ID`, [34](#)
 - `STRINGIFY`, [35](#)
 - `STRINGIFY_HELPER`, [35](#)
- `CMakeCXXCompilerId.cpp`, [36](#)
 - `__has_include`, [37](#)
 - `ARCHITECTURE_ID`, [37](#)
 - `COMPILER_ID`, [37](#)
 - `CXX_STD`, [37](#)
 - `DEC`, [37](#)
 - `HEX`, [37](#)
 - `info_arch`, [38](#)
 - `info_compiler`, [39](#)
 - `info_language_extensions_default`, [39](#)
 - `info_language_standard_default`, [39](#)
 - `info_platform`, [39](#)
 - `main`, [38](#)
 - `PLATFORM_ID`, [38](#)
 - `STRINGIFY`, [38](#)
 - `STRINGIFY_HELPER`, [38](#)
- `COMPILER_ID`
 - `CMakeCCompilerId.c`, [34](#)
 - `CMakeCXXCompilerId.cpp`, [37](#)
- `CXX_STD`
 - `CMakeCXXCompilerId.cpp`, [37](#)
- `DEC`
 - `CMakeCCompilerId.c`, [34](#)
 - `CMakeCXXCompilerId.cpp`, [37](#)
- `dimsValid`
 - `ShapeLib::Circle`, [20](#)
- `dimValid`
 - `ShapeLib::Rectangle`, [28](#)
- `getBorderLength`
 - `ShapeLib::Circle`, [17](#)
- `getCenterPoint`
 - `ShapeLib::Circle`, [17](#)
- `getCircleVolume`
 - `ShapeLib::Circle`, [18](#)
- `getHeight`
 - `ShapeLib::Rectangle`, [26](#)
- `getID`
 - `ShapeLib::Shape`, [30](#)
- `getP1`
 - `ShapeLib::Rectangle`, [26](#)
- `getP2`
 - `ShapeLib::Rectangle`, [26](#)
- `getRadius`
 - `ShapeLib::Circle`, [18](#)
- `getWidth`
 - `ShapeLib::Rectangle`, [26](#)
- `height`
 - `ShapeLib::Rectangle`, [28](#)
- `HEX`
 - `CMakeCCompilerId.c`, [34](#)
 - `CMakeCXXCompilerId.cpp`, [37](#)
- `horizontalDistance`
 - `ShapeLib::Point`, [22](#)
- `ID`
 - `ShapeLib::Shape`, [31](#)
- `info_arch`
 - `CMakeCCompilerId.c`, [35](#)
 - `CMakeCXXCompilerId.cpp`, [38](#)
- `info_compiler`

- CMakeCCompilerId.c, [35](#)
 - CMakeCXXCompilerId.cpp, [39](#)
- info_language_extensions_default
 - CMakeCCompilerId.c, [35](#)
 - CMakeCXXCompilerId.cpp, [39](#)
- info_language_standard_default
 - CMakeCCompilerId.c, [36](#)
 - CMakeCXXCompilerId.cpp, [39](#)
- info_platform
 - CMakeCCompilerId.c, [36](#)
 - CMakeCXXCompilerId.cpp, [39](#)
- LICENSE.MD, [39](#)
- main
 - CMakeCCompilerId.c, [35](#)
 - CMakeCXXCompilerId.cpp, [38](#)
 - main.cpp, [40](#)
- main.cpp, [39](#)
 - main, [40](#)
- operator<<
 - ShapeLib, [13](#), [14](#)
- p1
 - ShapeLib::Rectangle, [28](#)
- p2
 - ShapeLib::Rectangle, [29](#)
- PLATFORM_ID
 - CMakeCCompilerId.c, [34](#)
 - CMakeCXXCompilerId.cpp, [38](#)
- Point
 - ShapeLib::Point, [21](#)
- point.cpp, [44](#)
- point.h, [41](#), [42](#)
- printInfo
 - ShapeLib::Circle, [18](#)
 - ShapeLib::Rectangle, [27](#)
 - ShapeLib::Shape, [31](#)
- radius
 - ShapeLib::Circle, [20](#)
- README.md, [40](#)
- recalculateDim
 - ShapeLib::Circle, [18](#)
 - ShapeLib::Rectangle, [27](#)
- Rectangle
 - ShapeLib::Rectangle, [25](#)
- rectangle.cpp, [45](#)
- rectangle.h, [42](#), [43](#)
- setCenterPoint
 - ShapeLib::Circle, [19](#)
- setP1
 - ShapeLib::Rectangle, [27](#)
- setP2
 - ShapeLib::Rectangle, [28](#)
- setRadius
 - ShapeLib::Circle, [19](#)
- Shape
 - ShapeLib::Shape, [30](#)
- shape.cpp, [45](#)
- shape.h, [43](#), [44](#)
- ShapeLib, [13](#)
 - operator<<, [13](#), [14](#)
- ShapeLib::Circle, [15](#)
 - borderLength, [19](#)
 - centerPoint, [20](#)
 - Circle, [16](#), [17](#)
 - circleVolume, [20](#)
 - dimsValid, [20](#)
 - getBorderLength, [17](#)
 - getCenterPoint, [17](#)
 - getCircleVolume, [18](#)
 - getRadius, [18](#)
 - printInfo, [18](#)
 - radius, [20](#)
 - recalculateDim, [18](#)
 - setCenterPoint, [19](#)
 - setRadius, [19](#)
- ShapeLib::Point, [21](#)
 - horizontalDistance, [22](#)
 - Point, [21](#)
 - verticalDistance, [22](#)
 - x, [23](#)
 - y, [23](#)
- ShapeLib::Rectangle, [23](#)
 - dimValid, [28](#)
 - getHeight, [26](#)
 - getP1, [26](#)
 - getP2, [26](#)
 - getWidth, [26](#)
 - height, [28](#)
 - p1, [28](#)
 - p2, [29](#)
 - printInfo, [27](#)
 - recalculateDim, [27](#)
 - Rectangle, [25](#)
 - setP1, [27](#)
 - setP2, [28](#)
 - width, [29](#)
- ShapeLib::Shape, [29](#)
 - getID, [30](#)
 - ID, [31](#)
 - printInfo, [31](#)
 - Shape, [30](#)
- STRINGIFY
 - CMakeCCompilerId.c, [35](#)
 - CMakeCXXCompilerId.cpp, [38](#)
- STRINGIFY_HELPER
 - CMakeCCompilerId.c, [35](#)
 - CMakeCXXCompilerId.cpp, [38](#)
- verticalDistance
 - ShapeLib::Point, [22](#)
- width
 - ShapeLib::Rectangle, [29](#)

x

ShapeLib::Point, [23](#)

y

ShapeLib::Point, [23](#)