Streams

1. Content

2. Objective

At the end of these exercises you should be able to:

- Use the string class of the standard library
- Use the stream classes of the standard library
- Take advantage of the stream 's inheritance tree whenever appropriate

We advise you to **make your own summary of topics** that are new to you.

3. Exercises

Give your **projects** the same **name** as mentioned in the title of the exercise, e.g. **StreamsBasics**. Other names will be rejected.

Always adapt the **window title**: it should mention the name of the project, your name, first name and group.

Create a blank solution W08

3.1. StreamsBasics

Add a new project with name **StreamsBasics** to the **W08** solution.

Copy the given **Resources** folder in this project's folder. It contains the file SoftwareQuotesInput.txt.

No need to add the framework files.

In this console application you'll make some basic exercises on console/file IO hereby using the function std::getline (Function getline)

```
istream& getline (istream&  is, string& str);
```

and on string manipulations.

3.1.1. Read a sentence from console

Read a sentence from the **console** and print it on the console.
This seems to be a simple exercise, however there are some peculiarities:

- The sentence can be spread over several lines.
- When it is spread over several lines, you need to add a space between 2 consecutive read lines.
- You can suppose that a sentence always starts on a new line.
- The end of a sentence is indicated by a full stop. (yes, the dot)

After having read the whole sentence, write it to the console without new lines.
The operator += allows you to concatenate strings (Concatenating strings). Or

you can send the read sentence parts to an std::stringstream object and then use its str() method - [str method of StringStream] - to get the internal string.

Example:

```
Before software can be reusable
it first has to be usable (Ralph Johnson).
```

Input: Sentence spread out over 2 lines and no space after the word reusable

```
Before software can be reusable it first has to be usable (Ralph Johnson).
```

Result: Sentence on 1 line

3.1.2. Read sentences from file

Then do the same but for quotes that are in the given text file **SoftwareQuotesInput.txt**. You read them and then write them to another file: **SoftwareQuotesOutput.txt**, in the form of one line per sentence.

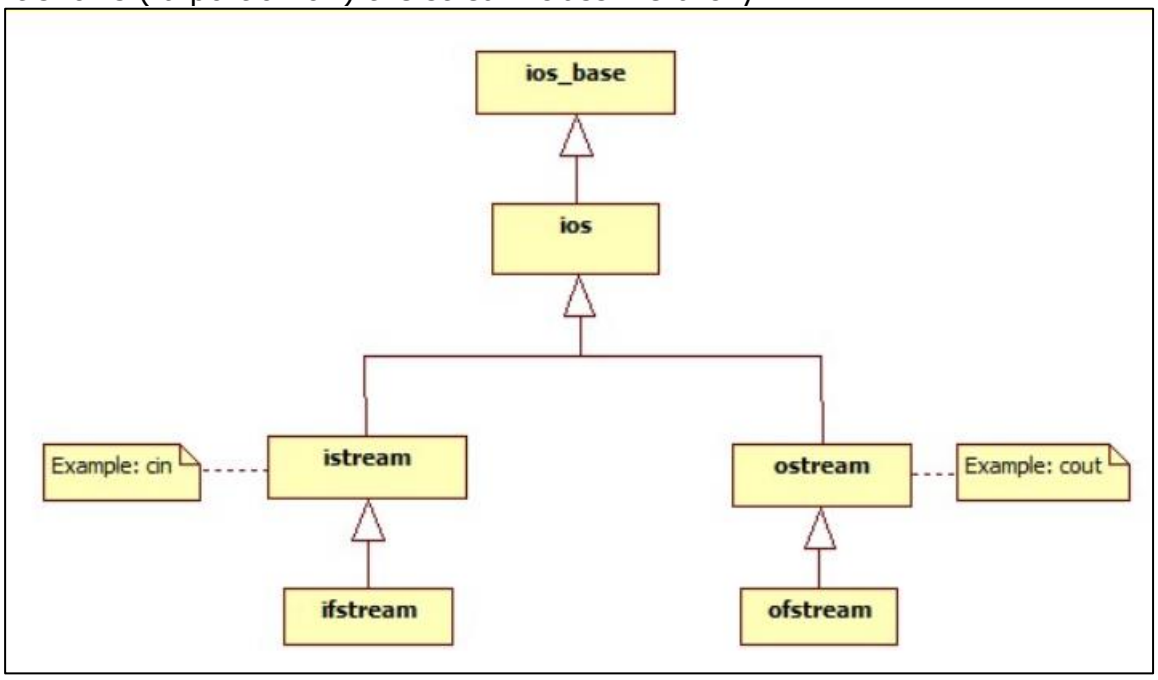
When you have finished these 2 exercises, you notice the same kind of code in both, the code that does the reading of lines from an input stream (ifstream or cin) and concatenates them into one string being the sentence.

Show that you can take advantage of the **stream inheritance tree**:

- The function std::getline requires an istream object as first parameter.
- The object std::cin is an instance of the istream class and the ifstream class inherites from istream.

The diagram below comes from https://www.cs.uic.edu/~jbell/CourseNotes/CPlus/FileIO.html.

It shows (a portion of) the stream class hierarchy.



Make one function that reads a sentence – possibly spread over several lines as described above - from any input stream, e.g. from std::cin, std::ifstream and that copies the whole sentence in the second parameter (sentence). Then use this function in both exercises.

```
void ReadSentence( std::istream& istream, std::string& sentence );
```

3.2. Streamed Shapes

3.2.1. General

In this project you:

- Create shape objects whose properties are read from the given file **Shapes.txt** at the start of the game.
- Save shape objects properties to a file - **SavedShapes.txt** - when the game stops.

3.2.2. Create the project

Add a new project with name **StreamedShapes** to the solution and delete the generated file **StreamedShapes.cpp**.

Customize the title of the window.

Copy **all the code files** from the previous project PolymorphismShapes into the folder of this project and add them to Visual studio.

Overwrite the **Game class files** with the given ones, they are in the folder: CodeAndResources/02_StreamedShapes

Also copy the given **Resources** folder into the project folder, it contains the txt file with shapes information.

3.2.3. Read shapes from file

a. Format of the shapes in the file

Have a look at the content of the given file **Shapes.txt**. Notice it contains information about shapes poured in XML format.

- Each shape is an XML element and starts with: **<NameOfShapeClass**
- This is followed by all property values in the XML attribute format **propertyName="propertyValue"**. You can assume that there are no spaces before nor after the equal sign and that the value is always enclosed in double quotes. Compound property types - such as Point2f and Color4f – have their components separated by a comma and follow the same sequence as their constructor parameter list.
- Each shape element ends with **/>**
- The shape information can be spread out over several lines.

You are not allowed to change the content of the file to simplify your code.

b. Game class

Have a look at the given Game class, it is extended with some functions, some of which - the ones that read and parse the shapes form file - need to be completed (see Task list of Game.cpp). It is up to you to complete them as described at the top of these function definitions.

ToColor

```
Color4f Game::ToColor( const std::string& colorStr ) const
{
    // TODO: 1a. Complete the ToColor function definition
    // The parameter contains a color in the format: "r,g,b,a"
    // This function converts this color into a Color4f type and returns it
```

Start with this function, there are several ways to complete this task, e.g.:

- Using Function find and Function substr of the std::string class
- Copying the string colorStr in a Stringstream and using Function getline with the ',' char as a delimiter. This is possible as StringStream inherits from istream and thus can be used as first parameter in the function getline.

```
istream& getline (istream&  is, string& str, char delim);
```

Then uncomment the corresponding test in the Game::BasicTesting function. It prints the result on the console.

```
void Game::BasicTesting() const
{
    // TODO: 1b. Uncomment this call of TestToColor
    TestToColor();
```

ToBool and ToPoint2f

Then complete in a similar way the **ToBool** and **ToPoint2f** functions, and uncomment the tests as well.

```
bool Game::ToBool(const std::string& boolStr) const
{
    // TODO: 2a. Complete the ToBool function definition
    // The parameter contains a bool in text form: "true" or "false"
    // This function converts this information into a bool type and returns it
```

```
Point2f Game::ToPoint2f( const std::string& point2fStr ) const
{
    // TODO: 3a. Complete the ToPoint2f function definition
    // The parameter contains a point in the format: "x,y"
    // This function converts this information into a Point2f type and returns it
```

GetAttribute

```
std::string Game::GetAttributeValue(const std::string& attrName, const std::string& element) const
{
    // TODO: 4a. Complete GetAttributeValue function
    // It looks for the value of the given attribute (attrName)
    // in the given element and returns it
```

In this function you need to look for the start and end of the attribute value. This means that you need to look for the " character which has a special meaning in string definition. It is used to indicate the start and the end of a string. There are 2 ways to solve this problem, try both:

- You can unescape the special meaning of " by preceding it with a \
- You can surround the string literal by **R"(** and **)"**

Raw string literal

Then uncomment the test of this function.

CreateShapesFromFile

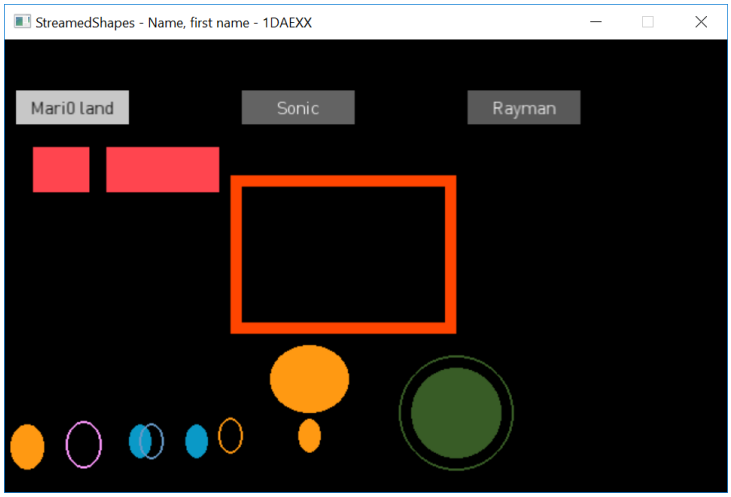
This function is called in the Game::Initialize function. It reads the shapes information from the given text file and creates Dae.. objects using the above functions you completed.

```
void Game::CreateShapesFromFile( const std::string& fileName )
{
    // TODO: 5. Read the XML shape elements from the given file
    // Call for each read DAE shape element the function CreateShape
```

Notice that the CreateShape function is already defined. You read the elements from the file and for each read element call the function CreateShape. It expects the shape information in XML format.

Tip: use **std::getline** using '>' as delimiter.

After you have defined this function, starting the application shows these shapes on the window.



3.2.4. Write shapes information to a file

When the game stops, all shapes have to be written to the file **SavedShapes.txt** using the same XML format.

a. Adapt the Dae classes

Make the Dae classes responsible for assembling their properties into an XML string.

They all need the common properties that reside in the DaeShape class. Therefor first add a method **PropertiesToString** to the abstract DaeShape class that returns an std::string object with these common properties, like this:

Center="260,340"

Width="100"

Height="30"

Color="0.39,0.39,0.39,1.0"

Then add to each concrete Dae… class a public method **ToString** which returns an std::string object with the shape information in the same format as the one used in the Shapes.txt file.

The DaeRectLabel needs to know the TextColor and Label values, add these properties.

b. SaveShapesToFile function

Then complete the last Task in the Game class.

```
void Game::SaveShapesToFile( const std::string & fileName ) const
{
    // TODO: 6. Complete the definition of SaveShapesToFile
    // Save all the shapes that are in the vector m_pShapes to the given file
    // in  the same XML format
```

Build, run and verify the content of the created file. Also test the correctness of this file by using it as input of this application. When you translated the shapes before stopping the application, starting the application with the Shapes.txt file content replaced by the SavedShapes.txt content should show the translated shapes at start.

Show your Dae... shapes code to the lecturer for immediate feedback.

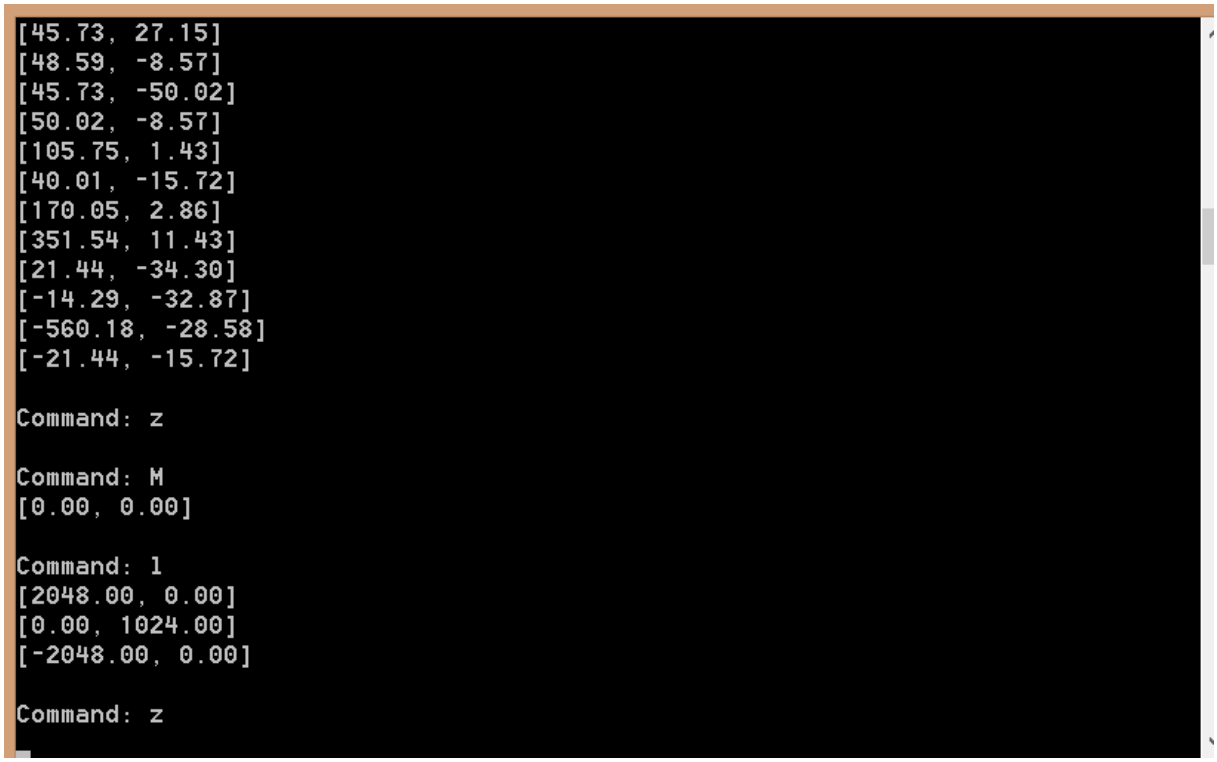
3.3. ParseSvgFile

3.3.1. Create the project

Add a new project with name **ParseSvgFile** to the solution.

No need for framework files, however copy and add **structs.h** and **cpp**.

In this **console** application you parse an SVG file and print the parsed information as you can see in the screenshot below.



Two files are given, try both:

- **Triangle.svg** which contains the description of a triangle
- **Level.svg** which contains the description of a level

When opening these files in a web browser the shapes are drawn.

Open the files with a text editor like notepad and have a look at the **path** element. It contains a **d** attribute which contains path instructions.

A d attribute exists of a list of commands, see Attribute d of Path element for more info.

Parse this **d attribute** and show the commands in the console. Also show the vertices following the m, M, L or l command. The file **dAttribute.txt** contains a more readable version of the d attribute. Use it to check the output of your application.

You can re-use functionality from the Shapes application like:

- Reading the Path element
- GetAttributeValue: to get the content of the d attribute of the Path element.
- StringToPoint2f: to parse the vertices

4. Submission instructions

You have to upload the folder *1DAExx_08_name_firstname.* This folder contains 2 solution folders:

- W08
- Name_firstname_GameName which also contains your up-to-date report file.

Don't forget to clean the projects before closing them in Visual Studio.

5. References

5.1. Input/Output

5.1.1. The library

http://www.cplusplus.com/reference/iolibrary/

5.1.2. The operator bool

http://en.cppreference.com/w/cpp/io/basic_ios/operator_bool

5.1.3. Formatting flags

http://www.cplusplus.com/reference/ios/ios_base/fmtflags/

5.2. Strings

5.2.1. Concatenating strings

http://www.cplusplus.com/reference/string/string/operator+=/

5.2.2. Function find

http://www.cplusplus.com/reference/string/string/find/

5.2.3. Function substr

http://www.cplusplus.com/reference/string/string/substr/

5.2.4. Stringstream

http://www.cplusplus.com/reference/sstream/stringstream/

5.2.5. Function getline

http://www.cplusplus.com/reference/string/string/getline/

5.2.6. String is a container

http://www.cplusplus.com/reference/string/string/?kw=string

http://en.cppreference.com/w/cpp/string/basic_string

5.2.7. String offers several search functions

http://en.cppreference.com/w/cpp/string/basic_string

5.2.8. Raw string literal

https://en.cppreference.com/w/cpp/language/string_literal

5.3. Stringstream

5.3.1. str method of StringStream

http://www.cplusplus.com/reference/sstream/stringstream/str/

5.4. Useful functions in cctype

http://www.cplusplus.com/reference/cctype/

5.5. Svg format

5.5.1. Attribute d of Path element

http://www.w3.org/TR/SVG/paths.html#PathData