# Tutorial 5b & 6a/b

Vytautas Vislavicius

October 6, 2016

## 1 Classes (basic)

The aim of this tutorial is to write a simple class `rectangle` and utilize it in a function. Begin by checking out the svn directory
`svn co http://svncourse.hep.lu.se/svncourse/trunk MyLocalDirectory`
where `MyLocalDirectory` is a local directory you want to link to svn. If you have the directory checked out, updating it will suffice. Navigate to `Vytautas/Tutorial5b/` and you will find three files:

- `rectangle.h` – a header file with class declaration
- `rectangle.cxx` – a file with description of the class
- `run.cxx` – a steering macro (*i.e.* a macro that contains `main()` function

When starting from scratch, you would have to create these files yourself and implement the proper structure (see the lecture 5 slides), but this has been done. However, all the methods defined in `rectangle.cxx` class do not do anything – and the first task for you will be to implement the methods so the class does what you want it to do.

The final aim of the tutorial is to make an executable that would do the following:

- `./run X`, where X>0 is an integer number, would create an array of X rectangles with random size edges, calculate the area of each of them and then print out on the screen the rectangles (border sizes and areas) with largest and smallest areas. It also prints the sizes of these borders into a file `default.out`.
- `./run X OutputFileName.out`, where X>0 is an integer number and OutputFileName.out is some output file name, does exactly the same

what is described in 1st point, but border sizes are printed to file OutputFileName.out (as opposed to default.out).

- `./run X OutputFileName.out InputFileName.in` reads the border sizes of X rectangles from InputFileName.in, finds the ones with largest and smallest area, prints out the info of these rectangles on screen (size of edges + area) and stores the borders in OutputFileName.out as in previous cases.

This might sound scary at the moment, but hopefully with this tutorial we can go through it all step by step.

### 1.1

Start by editing the `rectangle.cxx` class. The comments in every method explain what the method should do. Once you are finished, you can compile the class by typing `g++ -c rectangle.cxx` to make sure that you do not have any errors. Once that is finished, you can try and see if it works fine. Compiling and running

```
g++ -o run run.cxx rectangle.cxx
./run
```

should produce output

```
Rectangle: a = 0.6, b = 0.5, S = 0.03
```

### 1.2

The function `GetRandomNumber(double max)` in `run.cxx` returns a random number in range from 0 to max. There is another function `MakeOneRectangle()` that is not implemented. Implement this function so that it creates a rectangle with random $a$ and $b$ (obtained with `GetRandomNumber()`) on the heap and return a pointer of it. Then edit your main function so that it creates a rectangle calling `MakeOneRectangle` function and prints its info out. Compile and test.

### 1.3

Modify your code so that calling `./run X` would create an array of X rectangles, all of random size, and print all of their info on the screen. Note that to create an array of X objects, one would have:

```
rectangle **myrecarray;
myrecarray = new rectangle*[X];
```

so that `myrecarray` can be treated as an array of pointers. Then for each element of `myrecarray` one can call

```
myrecarray[index] = MakeOneRectangle();
```

### 1.4

Modify the main() function, so that after the array is created, you loop through it and find the rectangles with smallest and largest areas. You want to keep the indeces of those rectangles (*i.e.* their position in an array) for future reference. A good starting point is to assume that the first element (one at index 0) has the largest AND the smallest area. Then you should loop through the rest of the array (indeces 1 to the end of array) and compare the areas to your initial guess. If the area of $i$-th rectangle is smaller(larger) than that of your initial guess, you then say that the $i$-th rectangle has the smallest (largest) area and compare with the rest of the array.

Once you are finished with looping over the array, you should have found and stored the indeces of the rectangles with smallest and largest areas. Print them on screen using rectangle::PrintShapeInfo() method.

### 1.5

Check `PrintToFile` and `ReadFromFile` commands. For former, one should first open a `std::ofstream` file and pass a pointer of it when calling `PrintToFile`. **Remember to close the file once all the writing is done**.

The `ReadFromFile` method is a private one and you cannot call it from the main() function. Instead, it should be implemented via `rectangle(std::ifstream *InFile)` constructor. You should open an `std::ifstream` file and pass a pointer of it to the constructor, *e.g.*

```
ifstream *InFile = new ifstream("dummy.in");
mynewrectangle = new rectangle(InFile);
mynewrectangle->PrintShapeInfo();
```

should create a rectangle with border sizes of 2 and 3 and output in on the screen.

### 1.6

Add the rest of command line arguments. Ultimately you want to call:

```
./run X [OutputFileName.out [InputFileName.in]]
```

where arguments in square brackets are optional. You have few possible cases:

- If the number of arguments is 2 (remember – first argument is always the name of the file!), then only number X is entered. You should then create an array of X rectangles (random border sizes), find the ones with largest and smallest areas and print their info into `default.out` file.
- If the number of arguments is 3, then X and OutputFileName.out is defined. You should do exactly the same as in previous point, but instead of printing the info into `default.out`, you should print it into `OutputFileName.out`
- If the number of arguments is 4, then X, OutputFileName.out and InputFileName.in are defined. In this case most of the code remains the same, but instead of creating the new rectangles with random coordinated (`MakeNewRectangle`), you should create them with `new rectangle(std::ifstream)` constructor. **Remember, for this to work you need to have ifstream opened.**

For InputFileName.in you can use a dummy file that came together with the files you have downloaded, *i.e.* call

```
./run X SomeOutputFileName dummy.in
```

Note that in dummy.in there are only 9 rectangles defined, so X should not be greater than 9.

## 2 Inheritance and polymorphism

Create a new class `triangle`. Ultimately, if you are finished with `rectangle` class, you can make a copy of it and rename it to `triangle` (you have to change names in both header and cxx files!). Try replacing rectangles in `run.cxx` file with triangles and make sure it runs properly.

### 2.1

Now that you have two classes that share many simmilar methods, create a base class `shape` and move all the relevant functions from `rectangle/triangle` to `shape` class. Make `rectangle/triangle` inherit from `shape` and remove all the redundant methods. Make sure that `run.cxx` is running fine!

### 2.2

You would like to compare the areas of shapes. You would like the following

```
rectangle->IsBiggerThan(rectangle);
rectangle->IsBiggerThan(triangle);
triangle->IsBiggerThan(rectangle);
triangle->IsBiggerThan(triangle);
```

to return `true`(`false`) if the object on the left-hand side has the larger(smaller) area than the one on the right-hand side.

### 2.3 Bonus

Now that you have two classes, printing info to files becomes little ambiguous: when printing info to a file, only base and height is stored. Hence, if you want to read it later, you do not know whether it corresponds to a triangle or rectangle. To fix this, for each print-out, you should also add an extra flag, e.g. 1 for `rectangle` and 2 for `triangle`. Try thinking of a way to implement it properly and, if you can, do it!