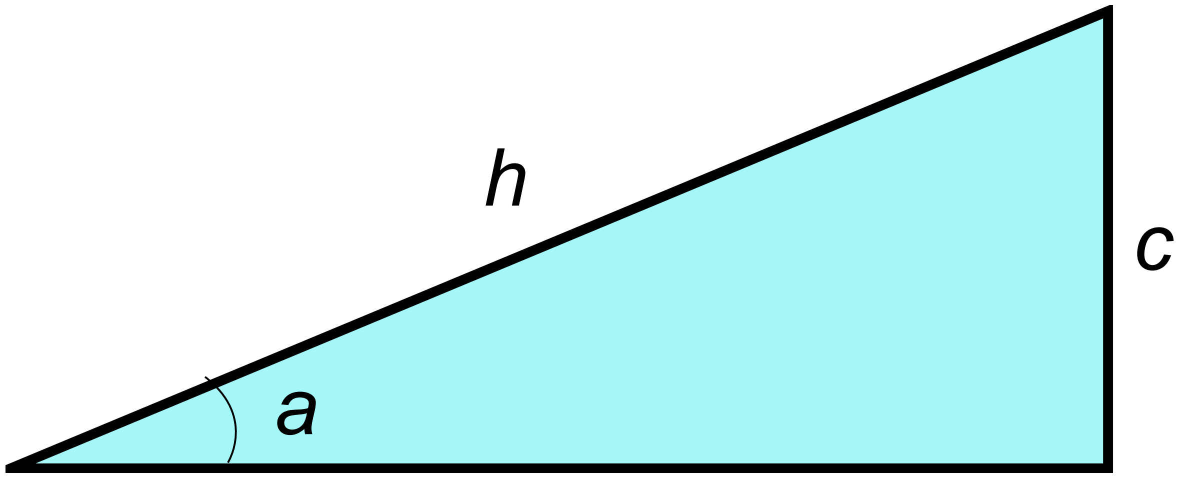
Rectangular triangle

Develop a program that goes through a set of rectangular triangles such as this:



For each triangle, the user will introduce through the keybard the hypotenuse *h* and the side *c* and then output the angle *a* opposed to the side, in degrees. The program will end when the user introduces the data of an imposible rectangular triangle.

Assume that you don't have available the implementation of the inverse sine, so you need to approximate it through its series expansion

$$\sin^{-1}(x) = \sum_{i=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1}.$$

Assume as well that you don't already have a function for calculating neither the factorial nor the power. We consider a sufficient approximation of the inverse cosine the use of the ten first terms of the series expansion.

Command line compilation

For distributing the code, put the specification of the functions you implement on a file named **`functions.h`** and their implementation on a file **`functions.cpp`**. The source code of the main program that solves the full proposed problem can be in a file named **`triangles.cpp`**. Therefore everything should compile with the following command line:

```
g++ triangles.cpp functions.cpp -o triangles
```

You can experiment with other source code distributions, and with manual step-by-step compiling.

Build system

Create a `CMakeLists.txt` file for compiling your code in the root folder of your program. If you have followed the code distribution suggested above, this file should look something like the following:

```
cmake_minimum_required(VERSION 3.15...3.30)
project(
    Triangles
    VERSION 1.0
    LANGUAGE CXX
)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Wpedantic -Werror")
add_executable(triangles functions.cpp triangles.cpp)
```

Tweak the file to match your particular code distirbution. If you create some test programs for your functions, add another `add_executable` line.

Once you have finished this `CMakeLists.txt`, open its containing folder in Visual Studio Code, and see how it generates its own internal build system. Then try to use its user interface to execute the program and test it.

Submission

Include all the source code files (`functions.h`, `functions.cpp` and `triangles.cpp` in our example) and the `CMakeLists.txt` for compilation. If you have (and you should) developed other test programs for your functions, include them as well.

Before submitting, remember to talk to your teacher and have a **small interview** describing and answering questions about the job you have done and the decissions you have made. This interview is *20%* of the total grade of the assignment, while the submitted material is evaluated for the other *80%*.

All source code files, must include a comment on top with the names and NIAs of the students involved on their development. Then, all source code files, including directory structure, must be compressed into a single `zip` file with the following naming:

- **`triangles_<nia1>_<nia2>.zip`**, where `<nia1>` and `<nia2>` are the NIAs of the involved students, if the work has been done in pairs. In this case, **only one** of the students must submit the work in Moodle.
- **`triangles_<nia>.zip`**, where `<nia>` is the NIA of the involved student if the work has been done individually.

The compressed file must not contain anything else besides the source code files. Particularly, do not submit any binary file, neither executable, library nor object. The submission of the `zip` file must be done through the corresponding task in Moodle before the stablished deadline.