

Exercise

1) Build a project consisting of

- a class Rectangle, implementing separately the header file Rectangle.h and the source file Rectangle.C
- a main program myprog.C that uses the Rectangle class

Compile the Rectangle object, the main object, and link them

```
class Rectangle
{
public:
    Rectangle(double b=1, double h=1); //constructor

    void setBaseHeight(double, double); //set base and height of the rectangle
    double getDiag(void); //get the rectangle diagonal
    double getArea(void); //get the rectangle area
    double getB(void); //get the rectangle base
    double getH(void); //get the rectangle height

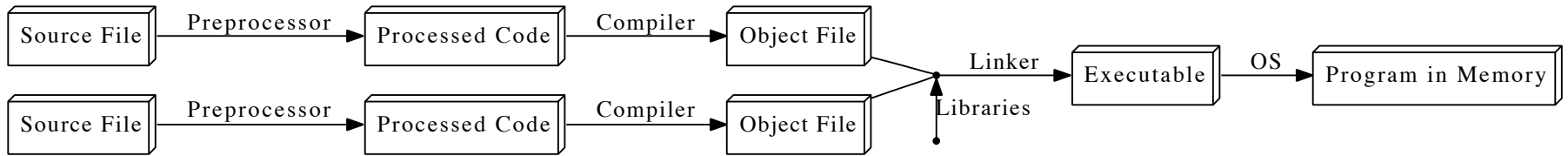
private:
    double base; double height;
    ~Rectangle(); //destructor
};
```

Exercise

1) Build a project consisting of

- a class Rectangle, implementing separately the header file Rectangle.h and the source file Rectangle.C
- a main program myprog.C that uses the Rectangle class

Compile the Rectangle object, the main object, and link them



Exercise

2) Use an external library to implement the getDiag function

GSL (GNU Scientific Library) is a library that provides basic and advanced mathematical and analysis tools. We could use

`double gsl_hypot(double x, double y)` that basically returns `sqrt(x*x+y*y)`

GSL libraries `libgsl` and `libgslcblas` and the headers are available in the lab computers

`/usr/include/gsl`

`/usr/lib64`

Exercise

3) Write a Makefile to automatically compile your project

Exercise

3) Write a Makefile to automatically compile your code

```
CXX          := g++
CXXFLAGS     := -O -Wall -pedantic
SRC          := ./src/
INC          := -I./inc -I/usr/include/gsl
LIB          := -L/usr/lib64 -lgsl -lgslcblas
OBJ          := ./obj/
EXE          := ./exe/

default: $(EXE)/myexe

$(OBJ)%.o: $(SRC)%.C
    @echo Compiling  $< ...
    @if ! [ -d $(OBJ) ] ; then mkdir -pv $(OBJ); fi
    $(CXX) $(CXXFLAGS) $(INC) -c -o $@ $<

$(EXE)/myexe: $(OBJ)rectangle.o $(OBJ)myprog.o
    @echo Linking  $^ to $@
    @if ! [ -d $(EXE) ] ; then mkdir -pv $(EXE); fi
    $(CXX) $(CXXFLAGS) $(INC) $(LIB) -o $@ $^

clean:
    rm -fv $(OBJ)*
    rm -fv $(EXE)*
```

Exercise

4) Implement, for the Rectangle class and for the main function, some debug printouts via a Preprocessor Directive

Exercise

5) Implement a 'static' method, `Rectangle::Merge()`, that accepts two `Rectangle` instances as inputs, checks if one of the two sides (base or height) is equal and, if yes, returns a new `Rectangle` object, of the two inputs.

In the main function, compute the Area and the Diagonal for this new `Rectangle`.

What happen if both the sides of the two inputs are different? What to return?

Exercise

6) Implement a method, overloading the '+' operator (*), acting on one Rectangle instance and *adding* to it another Rectangle instance:

```
Rectangle a;  
Rectangle b;  
Rectangle c = a+b;
```

(*) let's define "sum" as the sum of both the *base* and *height*.