UNIVERSITÄT DUISBURG ESSEN

NES Networked Embedded Systems

Programming in C/C++

Exercises 7
!!! Deadline: Tue, Dec 17th, 23:55 !!!

Task 1: Implement the following classes:

- class "`StaticBase`" with a static member function called `myFunc()` and a static member `int` variable.
- class "`PureBase`" with a constructor, a public member function called `myFunc()` and a private member `int` variable.
- class "`VirtBase`" with a constructor, a public non-virtual member function called `nvFunc()` , a virtual member function called `virtFunc()` and an `int` variable.
- class "`VirtDer`" that publicly inherits from `VirtBase` and has a non-virtual member function called `myFunc()`, a virtual member function which overrides the virtual function of the same name in `VirtBase` and an `int` variable.

All member functions have no parameters, return no value (i.e. `void`) and simply increment the `int` variable if called. In addition, all the functions include the following code:

```
#ifdef VERBOSE
     std::cout << "Called " << typeid(*this).name() << "::"
     << __func__ << std::endl;
#endif
```

For the `StaticBase::myFunc()` method, use directly the name of the class as a string since static methods do not have access to the `this` pointer.

In main() define the following:

```
PureBase pure;
VirtBase vbase;
VirtDer vder;
PureBase *ppure   = new PureBase();
VirtBase *pvbase  = new VirtBase();
VirtDer *pvder    = new VirtDer();
VirtBase *pvbaseder = pvder;
```

a) Now add the following to your main:

- Make all possible distinct function calls using the variables defined above (total should be 15). *Hint*: do not forget the static methods
- Group them according to the similarities of the calls and add a comment describing each group. See slide 54 of "Chapter 6 – Inheritance" in the lecture slides

Points: (code 15 pts, comments 10pts)

b) Use the StopWatch and the MEASURETIME macro from exercise 05 to measure the time it takes for each of the function calls you made in (a) above. Use 50.000.000 repetitions for each test. Wrap all of the code in this section inside a preprocessor directive `#ifdef TASK02 … #endif`.
Copy the macro definition into the source file and include the `StopWatch` class header `stopwatch.hpp` using the `#include` preprocessor directive. The description parameter for the macro should be the function call e.g. `pure.myFunc()`.

Run the measurements, then compare and interpret the results in your report.

Points: (code 25pts, comments 5pts, report 10pts)

c) Using `pvbase`, `pvder` and `pvbaseder` as defined before. Implement and describe (using comments) two possibilities to call the non-virtual member function of `VirtDer` using `pvbaseder`. Calls of the `myFunc()` should be safe, make sure the conversion succeeded before you make the call. Measure the time needed for both versions and report your observations! The description for your macro should be the type of cast you used.
The code in this section should be wrapped in the preprocessor macro `#ifdef TASK03`.

Are there any disadvantages of the faster version compared to the slower one? If so, describe an example in your report.

Hint: The == operator for the `type_info` object is overloaded as one would expect.
Points: (code 15pts, comments 5pts, report 15pts)

Submission: All your code for this task should be in one file called overhead.cpp. Your `StopWatch` class should be included with the `#include` directive.
Also create one Makefile which has different targets as follows:

- `all`: target should compile the code normally and define the variable VERBOSE using the compiler directives. The executable should be called **allcalls**
- `task02`: target should compile the code and define the variable TASK02 in the compilation instructions. The name of the executable should be **overhead**.
- `task03`: target should compile the code and define TASK03 variable during compilation. The name of the executable should be **casting**
- You can define variables through compile options to g++ such as `-DTASK02`
- Remember to include your `StopWatch` source and header files and include them in overhead.cpp
- The goal is the ability to use the single Makefile and your source code to compile different versions of the code and generate different executables. We will test this.

Notes:

- Try to rearrange the calls and rerun the program. You do not need to find a layout where all the calls result in correct timing (as it might be different on my platform anyway). Report on the differences between the groups and on possible outliers in your measurements.