Institut für Informatik — TUM
Scientific **C**omputing in **C**omputer **S**cience
R. Wittmann

WT 2015

Tutorial
(Advanced Programming)
Worksheet 3:

Assignment 1: Integral datatype properties

In this assignment we discuss the influence of some properties of the primary datatypes. First of all, think about how you can compute the maximum und minimum number which can be represented by a certain datatype. Do this calculation for the following datatypes, including both signed and unsigned variants:

- char
- short
- int
- size_t

Verify your computations by comparing with the reference values provided by **s**td::numeric_limits.

Now, print out all integral values which can by represented by **s**igned char, including the minimum and maximum value. You may cast your current value to **s**igned int as otherwise your output might turn out to be very cryptic.

What happens if you pass the maximum/minimum value? Can you explain the cause of this behavior? What are potential problems caused by this effect? How can you fix or even avoid it?

Assignment 2: Floating-point datatype properties

For this assignment we will investigate some arithmetic properties when working with floating point numbers instead of integral numbers. We consider numbers which are far apart: e.g. small and large values in the same operation. It is possible that certain arithmetic properties do not hold anymore, like identity relationships. For example: executing an operation and reversing it might not yield the same source terms.

Consider the following test:

1. Define a base value (e.g. 1.0) and a delta value (e.g. 1e-2 or 1e-20)
2. Perform an arithmetic operation with these two values to produce a third value
   - newvalue = base + delta
   - newvalue = base * delta
3. Reverse the operation by subtracting either the base value or the delta to reconstruct one source term. Use division for the multiplication part.

4. Check if the reconstructed source term is equal to our actual source term.

Test this procedure for rather small and rather large delta values. Automatize this test procedure in a way that you can find the smallest or largest delta value for a given base value for both multiplication and division.

Also try these tests for different base values (e.g. 1.0e4). Do you recognize a certain pattern with respect to the accuracy range? How does it depend on the chosen datatype (float or double)?

Assignment 3: Alignment properties

In this exercise we want to study how the compiler arranges variables in memory. Note that it might be especially interesting to do these tests in different environments (linux, windows, mac osx, *bsd) or even on different architectures (arm, ppc, sparc, x86, x86-64) and compare the results. Consider the following four steps:

1. Define a new function which determines the byte alignment $(1, 2, 4, \ldots, 2^n)$ for a given (void*) address.
   Use C++ style casts (`reinterpret_cast` , `dynamic_cast`, `static_cast`) to convert the address to `size_t` on which you may perform arithmetic operations. Then, find the largest power of two which divides the address without a remainder.

2. Define another arbitrary function (without return value) and declare several local variables inside. Make sure you have at least one **int**, one **double** and one **char**.

3. Now, for every variable in your function, print its address, its current value, its alignment and its size.

Try different compiler optimizations, including `-Os`, `-O0`, up to `-O3` and architecture information using `-march=native` (for gcc/clang compiler), `-xHOST` (for icc compiler). Answer the following questions:

- Do you recognize a certain alignment pattern when looking at variables of the same type? What about different types?
- How does the order of the variables change and is this behaviour beneficial? Did you expect a different ordering?
- How do the properties of the variables change with compiler optimization?