Dr. Ole Klein, Stefan Meggendorfer
IWR, Heidelberg University
Submission date 09.12.2020

Exercise Sheet 4

Exercise 1: C++ Quiz (5 points)

Here are some additional questions from https://cppquiz.org for you to answer:

Question 1: https://cppquiz.org/quiz/question/124 (template template parameters)

Question 2: https://cppquiz.org/quiz/question/32 (constructor calls)

Question 3: https://cppquiz.org/quiz/question/125 (static in templates)

Question 2: https://cppquiz.org/quiz/question/17 (con-/destructors and inheritance)

Question 5: https://cppquiz.org/quiz/question/208 (inserting elements into maps)

Questions 2 and 4 discuss important points about constructors and destructors, while questions 1 and 3 are about minor details that might be interesting. What would you say about question 5?

Exercise 2: Linked List (Const Correctness) (5 points)

On the previous sheet you programmed a linked list to practice the interaction of constructors, destructors and pointers. We now want to extend this implementation, practicing const correctness and clean encapsulation.

(a) The `const` keyword has already been added to some of the methods and their arguments. Check if these qualifiers make sense and whether some are missing elsewhere, discuss:

- Which methods should or should not be `const`, and why?
- What is a good choice for the parameters and return values?
- What about data members of `List`, and what about the `Node` class?

(b) Add a `const` method `max()` that returns the maximum of the stored values, as long as the list is non-empty[1]. For efficiency, this value should be cached and only recomputed when it it requested and has become invalid in the meantime.

- What effect does the keyword `mutable` have and why is it needed here?
- What would be inefficient about updating the stored value every time the list is modified in some way?

(c) A function for printing the list could look as follows:

```
1 void printList (const List& list)
2 {
3   for (const Node* n = list.first(); n != 0; n = list.next(n))
4     std::cout << n->value << std::endl;
5   std::cout << "max: " << list.max() << std::endl;
6 }
```

- What exactly do the two instances of `const` refer to, especially the second one?
- Test your implementation with this function.

[1] Throwing an exception for empty lists would be good style, but here you might just return zero instead.

Exercise 3: STL Container Types (10 points)

In the lecture the different types of containers of the Standard Library have been discussed. Each type offers certain guarantees and is ideally suited for specific use cases. Here is a list of scenarios, what type of container would you use and why?

(a) You are writing a finite difference discretization of the Laplace equation, and you are using a structured grid. For each grid node, the solution at this point is stored. All nodes of the grid are numbered consecutively and their number is known a priori.

Which type of container is suitable for storing the node values?

(b) You are writing a Quicksort sorting algorithm, and you want to store the pivot elements in a stack structure to avoid recursive function calls.

Which type of container is well suited to save the pivot elements and which to store the data itself?

(c) When implementing direct solvers for sparse matrices, it is common to first minimize the bandwidth of the matrix in order to reduce the storage requirements of the resulting LU decomposition. One method for this is the Cuthill-McKee algorithm. In the course of this algorithm elements have to be buffered in a FIFO (first-in, first-out) data structure. The order of the elements in this FIFO does not change.

What type of container would you use as a FIFO?

(d) In your finite difference program, the solution is defined a priori on a part of the boundary. Nodes in this part of the boundary (Dirichlet nodes) aren't real degrees of freedom and must be treated differently when assembling the finite difference matrix. The number of these nodes is small compared to the total number of nodes.

You want to dynamically read the list of Dirichlet nodes and their solution values from a configuration file and make them accessable node by node. When solving linear PDEs the limiting factor is typically the memory, as you want to solve very big problems.

In what container you would store the indices of Dirichlet nodes and their values?

Exercise 4: Pointer Puzzle (5 points)

Remark: you may actually try this with your compiler.

Look at the following program:

```
1 void foo ( const int** );
2
3 int main()
4 {
5   int** v = new int* [10];
6   foo(v);
7 }
```

The compiler will exit with an error message, because you make a `const int**` out of the `int**`, something like this:

```
1 g++ test.cc -o test
2 test.cc: In function 'int main()':
3 test.cc:6: error: invalid conversion from 'int**' to 'const int**'
4 test.cc:6: error:   initializing argument 1 of 'void foo(const int**)'
```

Actually, shouldn't it always be possible to convert from non-`const` to `const` ...? Why doesn't this apply here?

Tip: It's clear why the following program doesn't compile:

```
const int* bar ();

int main()
{
  int** v = new int* [10];
  v[0] = bar();
}
```

What is the relation between this program and the one above?