Programming in C and C++: Supervision 3

Andrej Ivašković (ai294)
Compiled on: 28th October 2018

YOU MAY EMAIL ME YOUR WORK OR LEAVE IT IN MY PIGEONHOLE IN THE TRINITY COLLEGE GREAT COURT MAIL ROOM.
PLEASE SUBMIT THE ASSIGNED WORK AT LEAST 24 HOURS BEFORE THE SUPERVISION!

1 Before attempting the problems

These exercises concern the last part of the course. This is mainly concerned with core object-oriented parts of C++, including exceptions and templates. Hopefully, the previous supervisions has shown you some of the difficulties of using C – and now you should know how C++ enables you to solve some of them. There are fewer coding exercises this time, but they are likely to take longer.

Remember the issues with C semantics whenever you write any C or C++ code! An obvious instance of undefined behaviour is accessing array items of outside of bounds, but you should not forget about integer overflow.

2 Problems

Some of these questions have been taken from the exercises in the lecture slides. Credit is due to their authors.

1. Compare and contrast the following aspects of C semantics: *undefined behaviour*, *unspecified behaviour* and *implementation-defined behaviour*.
2. Compare and contrast passing by reference in C++ with passing pointers in C.
3. Explain the usage of the keyword `virtual` in C++, as well as what the underlying implemention of `virtual` is.
4. How is the *dreaded diamond* problem resolved in C++?
5. A commonly quoted good practice in C++ is declaring all destructors `virtual`. Explain why it makes sense to do this.

6. Suppose you have defined a class `A` and wish to allocate an object `a` of this class. Compare and contrast the syntax, allocation and deallocation when you want `a` to be allocated in stack, heap and static segments.

7. Implement classes that implement singly linked lists (`linked_list`) and dynamic arrays (`array_list`). They should have sensible constructors, destructors, and methods for adding a new element to the start of the list, adding a new element to the end of the array, inserting at a given position and removing the item at a given position.

   Initially implement them to support only `int` items, then extend them to support data of arbitrary type using templates.

   Overload the indexing operator `[]` in these classes so that `a[i]` returns the `i`-th item in `a`.

   You should throw exceptions when needed.

   Finally, introduce an appropriate abstract superclass `list` for these two classes.

   You don't need to care about thread safety.

   (You are, in essence, reinventing `std::vector`.)

8. Explain the RAII (*Resource Allocation is Initialisation*) pattern and describe two concrete scenarios in which you might use it.

9. Write a program that uses template metaprogramming in order to perform primality testing.

10. Read up on *move constructors* and the facilities that allow them to be written in C++. Compare and contrast them to copying constructors and briefly discuss when you would use one over the other.

11. Attempt past exam question: 2017 Paper 3′ Question 2.