Exercise Class 10 – Theory

0.1 Basic Pointers

Pointers are very similar to references, but different in the way that we can make them reference another variable, even after the initialization. Remember, once a reference is initialized, it will reference the same variable until the end of its own scope. The assignment operator `=` just changes the value of the variable that is referenced.

With pointers we have a different situation. We will be able to change the value of the variable the pointer points to (we call this variable target), and we will be able to make the pointer have another target. To implement this idea we need the concept of addresses, which on modern computers are typically 64-bit integers. However, when programming we almost never care about actual addresses. What we care is the objects that are stored in the memory that has these addresses. And this is what we will focus on when tracing.

How the code which manipulates the pointers look like? We obviously need a way to tell the program when to make the pointer point to another target and when to change the value stored in the target. We realize these by introducing two operators, the reference or address operator `&` and the dereference or value operator *.

0.2 Meanings of & and *

The symbol `&` can disorient many people approaching C++. It is important to realize that this symbol has 3 different meanings in C++, depending on the position:

1. the bitwise AND operator (e.g. `z = x & y;`)
2. to *declare* a variable as a reference (e.g. `int& y = x;`)
3. to *take the address* of a variable (address operator) (eg. `int *ptr_a = &a;`)

Similarly, the symbol `*` can be used:

1. as the arithmetic multiplication operator (e.g. `z = x * y;`)
2. to *declare* a pointer variable (e.g. `int *ptr_a = &a;`)
3. to *take the content* of a variable via its pointer (dereference operator) (e.g. `int a = *ptr_a;`)