# Building Programming Skills, 2021
# Point carefully



You are given many C programs involving pointers. These are teaching codes that I have used to explain pointers. For each program (say `foo.c`), run as:

```
gcc foo.c
./a.out
```

You will likely get many warnings, but ignore them. Each program declares and sets a collections of pointers. It then prints their values out, and does some dereferencing.

## 1 What should you do?

For each program, you should run and be able to explain the output. Since explicit pointer values (which are memory addresses) are being printed, the output will be different each time you run it. On the other hand, (typically) the *differences* between the memory addresses will be the same.

There are many questions and concepts that you should understand well. I list them below, in order of higher complexity. Start by understanding the simpler concepts and make sure you can explain the output of programs related to those concepts. Slowly, move on to the more challenging concepts.

1. Know the basic notion of a pointer. Understand the distinction between a pointer variable, its value (a memory address), and the content of what the pointer points to (obtained by derefencing).

2. Pointers to different types are not really "different". In all cases, they are memory addresses.

3. Understand that the content of a memory location (which is obtained by dereferencing) can be interpreted as an int, float, char, etc. There is no type associated with a memory location, even if it was created by declaring (say) an int. One can use a pointer to interpret that memory location in terms of a different type.

4. Understand that any array variable is really a pointer. Make sure you clearly know how pointer arithmetic works, and how that relates to reading arrays.

5. Understand the notion of a pointer to a structure, and how that relates to pointers of fields within that structure.

6. Know what `malloc` and `free` does. Understand the distinction between stack and heap memory. The former contains variables that are declared within a function call and these variables get deallocated when the function ends. The latter (heap memory) is allocated whem `malloc` is called. This memory is persistent, in that it stays allocated after the function ends. Memory leaks happen when this memory is not freed.

7. Understand how 2D arrays work. A 2D array is an array of pointers. When a cell of a 2D array is read, there is a "double pointer dereferencing.