C K&R Notes for Chapter 3

October 2, 2017

So, another nice chapter from this wonderful book. It was more difficult than previous chapters and there is one exercise that I could not completely do (the first). This chapter was focusing on `goto` and its several replacements (I would say all except function calls which can also be implemented with `goto`'s).

Like many programming languages, C has statements and blocks. Statements correspond to things to do and blocks define several statements belonging to a separate namespace that should be executed in order compatible with how it is written in source code. Blocks are important here because they can be used instead of a single statement for condition and loop constructs.

It makes the for loop a little weird, because if it has a block, it defines two namespaces. For example, what if there is an `int i` in `for(<...>)` and one `int i` in the block? Normally they should just work independently...

Yeah, it's how it works. I made a little test with two for loops and 3 different *i* variables:

```
#include <stdio.h>

int main()
{
  for (int i = 0; i < 10; i++)
    {
      int i = 30;
      printf("%d\n", i);
      for (int i = 7; i < 1000; i *= 2)
          printf("\t%d\n", i);
    }
}
```

And yes, it really works how one would imagine it to work... That means it's ok to have `for (int i = <...>;<...>)` inside another `for int i = <...>;<...>)` if the `i` that is the most outside is not needed and used only for counting... [1]

[1] The for namespace is only implemented in new versions of C. I tried to code once without it and decided that if I use it the code looks slightly better...

So ok, . . . the ifs, elses and switches are really not that special and similar to what we have for example in Java.

So there is another thing to say about the `for` loop. On `continue` it's just as if we go to the very end of the block / statement. That means that the test and the incrementation parts are executed.

About the goto's. . . There is a reason why they exist. It seems that in very first programming languages it was the only thing that could make them useful: they didn't have loops, procedures, ifs. Only goto's and conditional goto's. As the languages evolved and became easier to use more and more constructs began to replace the goto. But there still remain things that are better done with goto. Even Java implemented goto's, but in a very restricted way.

An interesting use of goto's that I used in a project is, imagine you have a loop. And some functions are called from this loop. Imagine that it's like repl loop. It's given a function name and arguments, so this loop call the function with aguments supplied by the user. What if the arguments are bad? The loop doesn't know about it, only the function called knows. So it has to inform the user of the situation and go back to the main loop. It's difficult to do even with goto's. What is needed is a goto from a function to the beginning of the loop. In order to do it there is a thing called long jumps. It's functionality is exactly what is needed for this situation. The function `setjmp` defines the location where to return and `lonjmp` goes to that location taking care of namespaces, stack and everything. People call it the Exception Mechanism of C exactly because it allows to implement an exception-like behaviour in C.

At the beginning I said that I didn't finish the first exercise of the chapter, but there are some days left for this chapters, so I think I will have time to do it. This is kind of things that could make my computer wonder what's going on, so I am not sure if it will succeed. . .