Chapter 1 Notes

September 10, 2017

1 Introduction

Here are my notes on the first chapter of the K&R book. I'm a C noob, so what follows should not be taken seriously. I tried to do the exercises, but I'm sure I have made lots of mistakes and what I made contains lots of bugs. But at least I have attampted. In this document I some of my thoughts and misconceptions that I had from reading the chapter and doing exercises.

2 Files

Source files can be two types:

- `.c`: usually contain procedure definitions
- `.h`: *header* files, usually contain function and variable declarations which are used in `.c` files. `.h` files are usually included using `#include`.

Files are compiled using the command `cc`, for example

```
~cc hello.c~
```

compiles the program contained in file `hello.c` and produces a `a.out` file which is an executable of the program.

But I usually use something like

```
~gcc -O0 -g hello.c -o hello~
```

because I use gcc and I like to be able to debug and have different executables in different files. I turn off the optimization in order to be able to inspect the executable binary (it is more predictable that way) and `-g` for debugging symbols. By the way, I see that there is even a `-ggdb` option for `gdb`, not sure if it really improves things (never tried it).

3 Functions

A program should have a `main` function. This is where it starts when it is called.

A function can have several arguments. In this case they are provided to the function when it's called. Arguments are passed by value, which means, that the local argument is a copy of the caller's variable. Same thing for arrays because array variables are in reality pointers to the first element.

A function can have its own variables. They are called "local variables" and are created when the function is called and is given control.

Variables in a function can also:

- static: they preserve their values between calls
- external: they can be shared between files and functions.

When a function is used before of its definition or its definition is in another file, a forward declaration is usually used. It's the same as a normal declaration, but with `;` instead of its body and `(void)` as its argument list if it has no arguments. Another name for this kind of declaration is *function prototype*.

4 Literals

1. Number literals

   It's very comlicated, but usually is just a number. If it has a point, then it is a float. Character literals are also number.

2. Character literals

   It can be either

   - a single character between ~'~
   - or a single escape sequence starting with \

3. Escape sequences

   They are used in a character or in a string literal. They correspond to a single character. They begin with a \ and then it's a character that is a character of special meaning or special character that is to be inserted litterally:

   - character with special meaning: `\n` for new line, `\t` for tab, `\0` for the `0`-value character.
   - character that has a special meaning within a character or string literal: \, single quote and double quote.

5 Input and Output

For printing, the `printf` function from a library is used. Its arguments is a string, containing special elements like `%s`, `%d`, `%f`, and corresponding variables or values, which are evaluated before passing, so an expression or a function can be used instead.

Printing a single character can be done with `putchar(c)`.

Reading a single character can be done with `getchar()`.

These functions can be used for file copying, character, line and word counting.

6 Comments

Multiline comments are put inside `/*` and `*/`.

7 Variable types

Variable can be of different types, like `int`, `float`, `char`, `short`, `long`, `double`. These types are signed nummerals and correspond to numbers of different size with or without floating point encoding. Integer negative numbers can be stored differently, so the result of casting to or from a negative number can be unpredictable. I think it's done this way in order to be like the hardware and so that the software is not forced to emulate a foreign behaviour.

8 Loops

There are three main kinds of looping in C:

- `for`: used when the test is a simple expression that is easy to evaluate and without side effects and there is a simple way to get to the next step. I think it's usually used when it is already known before the loop how many times it has to be executed.
- `while`: it is more like waiting for a condition to become false.

The third kind of loop is not mentioned in this chapter and I don't see it used very often and I think not many people like it. But in reality it sometimes can make the code more compact. It's like the `until` in Pascal.

9 Symbolic constants

Many people in the expanses of the internets say that it is not good to have literals that have a meaning inside the code. In order to avoid it they say that people must use the `#define` directive. On the other hand lots of people hate it and say that the right thing is to use the `const` keyword. This book doesn't

use the `const` keyword in the first chapter. Perhaps it's because `#define` is preferred or it is more often used or easier to understand... Or perhaps the book is old and the authors didn't know yet that it is not the right thing to do.

So this is how the `#define` directive works in this chapter:

```
#define NAME 123
```

I often see the define without anything after the `NAME`. I've seen it being used in header files in order to not include them twice and in checking of the OS.

10 Arrays

Arrays are a way to reference juxtaposed data of the same type. For example if we have several structs with the same structure, they can be referenced used arrays. But it also can be done with pointers, so it's just a way to make the code easier to read.

Arrays are pointers to the first element of an array. They are of the same type as the pointers. Pointer and array variables can be assigned to each other and be passed to functions as arguments without any warning.

Arrays are declare so:

```
int arr[n];
```

It means that the `int` pointer `arr` now points to a freshly created row of `n` contiguous `int` cells.

They can be accessed so: `arr[i]` (i-th element of the array, starting from 0) or so `*(arr + i)`.

A particular type of arrays is used to represent strings. It's a char array with a `\0` character after the string. This is the type of array created when string literals are assigned to variables or passed to a function (this is how the *hello world* program works).

Declaring arrays and pointers is very different: an array is a pointer, but not a pointer variable. Applying the `&` operator to it gives the location of the first element of the array.

11 Conclusion

So after doing the first chapter of the K&R book we are now ready to learn and discuss the real things about the C language. Reading it and doing the exercises makes us focus on some things we otherwise might have not noticed, and so we can get closer to understand the philosophy of C, the ways of planning and building C applications.