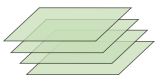
The Cellophane sheet visualization

Complied with LaTeX on February 7, 2013

Disclaimer

This paper contains original unverified research. The author is not responsible for any damages caused to intellectual property by reading, sharing or distibuting this paper, both legally and illegally. Use at your own risk!

Preface



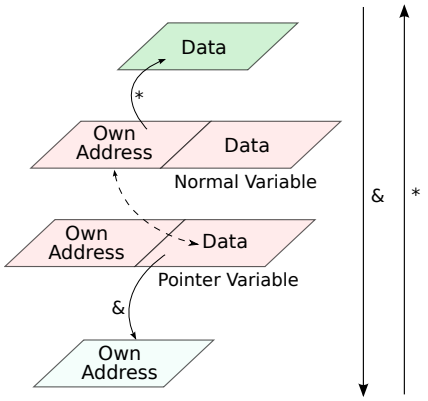
The objective of this paper is to make life with pointers easier. This article applies mainly to C and C++ but the syntax can be extended to just about any programming language which uses the kind of pointers that C/ C++ uses. The cellophane sheet technique is a powerful technique of visualising and evaluting complicated expressions like &(**(*a+5)+6) easily. Although such applications may be attributed to academic importance, this technique was created primarily to help the beginner grasp the concept of pointers.

What is a variable?

All memory locations can be, for the purpose of illustration, broken up into two parts: The address part which contains the address of the memory location and the data part which contains the data stored in that memory location.[1] When a "variable" is declared, it means that the compiler simply gives the user the convinience of naming the memory location with a name. For example, in the statement "int a" picks a suitable random memory location from the memory of the computer and names it "a" for the purpose of convinience of the programmer.

[1] In reality, address isn't "stored" in the memory location. Address is just the inherent property of every memory location

What then, is a pointer?



A pointer, or a pointer variable is also a variable. It behaves almost exactly like a normal variable. The only difference is that when an integer variable stores a number like 97 in its data part, a pointer variable stores a number like 7eb012c3[2] which refers to the address of another "variable". Attempting to print a variable prints its data part. Using the (&) operator on a variable returns its address part. Using the (*) operator on a variable finds another variable with the address mentioned in the data part of the variable. It returns the data part of the variable found. For example, in "int *a, b; a = &b;", the data part of *a* contains the address of *b*. Therefore *a would first find b and then return b's data part. In this example, we could say that *a* is "pointing to" *b*.

Variable types

Every variable type has two portions in the specification: The data type and the "pointer level".[3] The pointer level indicates how many times the (*) operator needs to be used in order to get to the real data and not just another address. For example, "int **a" makes variable *a* of data type int and pointer level 2 (or simply int**). It is hence a "pointer to a pointer". This means that a is pointing to the variable *a, which in turn is again pointing to **a. The data part of **a contains the real data.

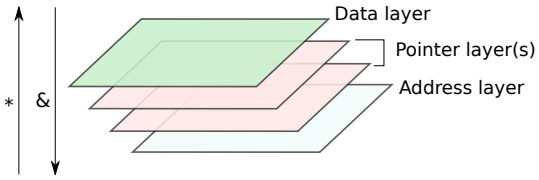
In the case of variables of pointer level 0 or "normal variables", the data type indicates how many bytes of memory are required to store the variable's data and how the data's variable is stored.

In the case of variables of pointer level >=1, the memory allocated and the method of storing the address is constant. The data type simply indicates the data type of the variables pointed to by the pointer. "float *a" for example makes variable *a* a pointer of level 1 and data type float. This means that it is "pointing to" a float variable of level 0. Note again that in both "float *a" and "int *b", *a* and *b* are of the same size.

[2] On a 32-bit system

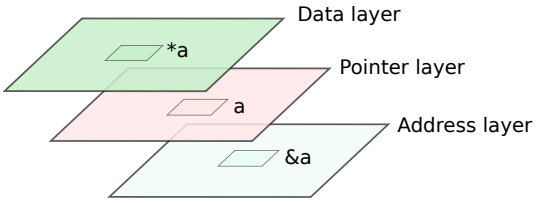
[3] Pointer level is a term coined by the author for the purpose of explanation only

Familiarization with the notation



The whole memory can be visualized as a set of layers. There are a minimum of two layers: the data layer and the address layer. As we add levels of pointers, the number of layers increase. The (&) operator enables one to move down layers and the (*) operator enables them to move up layers. Each of these layers is made of thin transparent cellophane sheets so one can look through the layers.

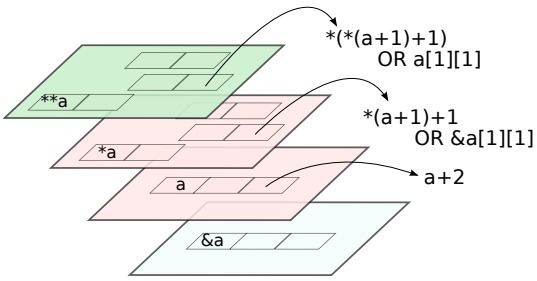
A simple pointer (declared using int *a) has been represented in the cellophane sheet layer notation in the figure on the right. There is only one pointer layer as a is a pointer variable.



Examples and exercises

Example

Consider the declaration int a[3][2]. Clearly, a is of type int**. For the visualization, the number of layers equals 2+(highest pointer level) = 2+2 = 4. The lowest layer is the address layer, the highest is the data layer and there are two intermediate pointer levels which refer to a, *a and their neighbours. First analyse what the declaration does. It creates three consecutive memory locations in the second layer, three sets of two consecutive memory locations in the third layer. We now create the data and address layers as shown. Think of the model as a tree where the base three branches branch off to two sub-branches each. The rest of the information can be deduced from the diagram as shown.



Exercises

Visualize: &(*a), *(&a), ***a, &(**(*a+5)+6), **(a[1]+1), *a[0][1]+1+4