CSE 351 Lecture 3 – Memory & Data II

Pointers and Pointer Arithmetic

Recall that ***pointers*** are special variables of *word size* that store addresses (*i.e.* "point" to data at that address). In C, we can declare a pointer variable using the following forms:

```
type* ptr;   // or equivalently, type *ptr
```

Notice that the pointer data type `type` encodes *size* information that, in combination with the location indicated by the value of `ptr`, specifies an exact piece of data in memory.

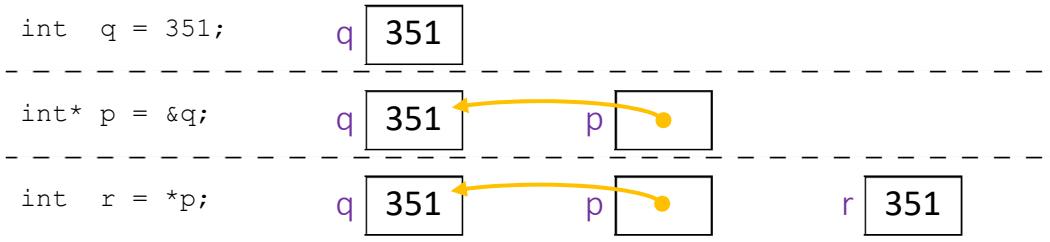
The *address-of operator* (***&***) goes in front of a variable name and returns the address of that variable in memory, which can be stored in a pointer of the appropriate type (*e.g.* `int q; int* p = &q;`).

The *dereference operator* (*******) is used to access the data that a pointer points to (*i.e.* it will grab the number of bytes specified by the width of `type` starting at the value stored in the pointer). For example, `int q = 351; int* p = &q; int r = *p;` stores 351 in `r`.

> **Note:** Here we are referring to when the symbol '`*`' is used as an operator (*i.e.* when it acts on a variable). When used in a variable declaration, such as `int *p`, the '`*`' symbol is part of the variable type (pointer to an int) and is NOT being used as an operator.

`NULL` is a symbolic constant that is specifically used for pointers. In C, `NULL` evaluates to zero and indicates a pointer to nothing. Dereferencing `NULL` results in a runtime error.

Pointers can be represented visually using ***box-and-arrow*** memory diagrams, where boxes represent variables (pointers and otherwise) and arrows visually connect stored addresses (*i.e.* the contents of pointer variables) and the addresses themselves in memory (*i.e.* what would be accessed upon dereference). Associated with each box are: (1) a value stored inside, (2) an address for its location in memory, and sometimes (3) a variable name, which is just an alias that your program uses for the value in the box.



Pointer arithmetic takes the size of the data type being pointed to into account by automatically scaling the arithmetic operation. You can think of this as a change in units from bytes into "things," where a thing is what's being pointed to. Common pointer arithmetic operations include adding integers (+/-) to pointers and subtracting two pointers of the same type. For integer pointers `p1` and `p2`, `p1+1` will evaluate to the stored address incremented by 1 `int` = 4 bytes and `p2-p1` will return the number of `int`s between the two addresses.

Arrays and Strings

Arrays are sets of contiguous locations in memory that store the same type of data object. Declaring an array in C follows the syntax `type array_name[num];` and sets aside `num*sizeof(type)` bytes of consecutive memory, where `num` is a constant like `5`. The size of an array cannot be changed after it is declared.

Array subscript notation `array_name[n]` refers to the `n`-th element of the array and is actually accessed via `*(array_name + n)` (*i.e.* using pointer arithmetic and dereferencing). However, there is no bounds checking in C, meaning using $n<0$ or $n \geq num$ is syntactically valid, though often leads to logical errors or bad memory accesses.

C does not support an explicit data type called `string`, though it somewhat confusingly does have a library header called `string.h` and uses terminology such as "string constants." "Strings" in C are actually arrays of characters that are terminated by the ***null character*** (*e.g.* `char str[] = "hi";`).

The *null character*, similar to `NULL`, evaluates to zero and is specifically used in character arrays. When `printf` is given the conversion character `%s` and a pointer to an array of characters, it will print out the characters until it finds the first null character.

A ***string literal*** (or *string constant*) is a sequence of characters surrounded by double quotes (*e.g.* `"hello, world!"`). A string constant is automatically stored in memory as an array of characters, terminated by `'\0'` (so the array length is the # of characters + 1), and can't be manipulated.