

Pointers

1 Computer Memory

Memory is arranged as a collection of cells numbered from 0 up to how ever many is available on the machine. A single variable is stored in a cell, or block of cells, depending on its size. A cell is big enough to store a single character; but a string is stored in a block of cells. Indeed, any array is stored as a block of cells.

So what happens when we define a variable: say `int A`? The compiler and operating system together choose a memory cell to store `A`—you can think of a name-tag stuck on the cell door. But this cell also has an *address*, given by `&A`. We used this in passing that information to `scanf`. .

2 Pointers, Addresses and Dereferences

A *pointer* is a variable that stores the *address* of another variable. Pointers use the `*` notation. In fact pointer syntax uses stars in two *different* meanings. We declare that a variable is a pointer by prefacing it with a star:

```
int *p;
```

Note that, while an address is really just a number, a pointer holds the address of a variable of a specific type. That is, the pointer `p` stores the address of a variable of type `int`. (Well, actually, `p` contains garbage until we initialize it.) We say it points to a specific cell in memory.

We initialize a pointer by setting it to the address of a variable. This can be done in *two* ways: either set it to the address of some variable using the `&` notation, or set it to another pointer. Here is the former:

```
int *p = &A;
```

Now, we need to be able to work with the variable that `p` points to. This value is accessed by `*p` (sometimes called *dereferencing*).

The following code creates a variable `x` and a pointer `p` to it. Once that is set up, changing the value of `*p` changes the value of `x` and vice versa.

```
int x = 2;
int *p = &x;
printf("%d",*p); // prints 2
*p = 11;
```

```
printf("%d",x);    // prints 11
x = -5;
printf("%d",*p);  // prints -5 because p still points to x
```

3 Pointer Arithmetic and Arrays

Incrementing a pointer moves it to the next “cell” in memory. (The actual change in address depends on the size of the variable the pointer points to.) This pointer arithmetic is useful in array manipulation. The pointer can be initialized with the name of the array—because the array name is equivalent to the address of the first cell.

The following code prints out the array:

```
int B[] = {1,4,9,16};
int *p = B;    // initialize p to point to start of array B
for(int i=0; i<4; i++) {
    printf("%d", *p);
    p++;
}
```

It is good practice to set unused pointers to NULL (a special constant defined in C to be equal to 0), to emphasize that we know it’s not yet or no longer usable.

4 Strings via Pointers

We saw that a string is stored as an array of `chars`. It turns out that any block of `chars` can be interpreted as a string. This block can be accessed via the array name, or more generally, via a pointer.

In general, a string function takes as its argument either an array name or a pointer to the first char in the string. Here is another implementation of the function to calculate the length of a string, using pointer arithmetic.

```
int strlen(char *s)
{
    int len=0;
    while ( *s != 0 ) {
        s++; len++;
    }
    return len;
}
```

So, usually when you see `char *P` written, the pointer `P` points to (the first char in) some string. But sometimes it really points only to a single char. Go figure!