Pointers and Arrays

A pointer is a variable that contains the address of a variable.

Pointers and Addresses

The unary operator & gives the address of an object. The statement

```
p = &c;
```

assigns the address of `c` to the variable `p`, and `p` is said to "point to" `c`. The & operator only applies to objects in memory: variables and array elements. It cannot be applied to expressions, constants, or `register` variables.

The unary operator * is the **indirection** or **dereferencing** operator; when applied to a pointer, it accesses the object the pointer points to.

The following code is used to show how to use & and *.

```
int x = 1, y = 2, z[10];
int *ip;                    /* ip is a pointer to int */

ip = &x;                    /* ip now points to x */
y = *ip;                    /*y is now 1 */
*ip = 0;                    /* x is now 0 */
ip = &z[0];                 /* ip now points to z[0] */
```

Pointers and Function Arguments

Since C passes arguments to functions by value, there is no direct way for the called function to alter a variable in the calling function. To directly alter the value of the argument you must use pointers.

```
void swap(int *x, int *y) {
     int temp;

     temp = *x;
     *x = *y;
     *y = temp;
}
```

```
swap(&a, &b);          /* swap function calling sequence */
```

Pointers and Arrays

The declaration

```
int a[10];
```

defines an array `a` of size 10. The notation `a[i]` refers to the `i`-th element of the array. if `pa` is a pointer to an integer, declared as

```
int *pa;
```

then the assignment

```
pa = &a[0];
```

sets `pa` to point to element zero of `a`; that is, `pa` contains the address of `a[0]`.

Now the assugnment

```
x = *pa;
```

will copy the contents of `a[0]` into `x`.

If `pa` points to a particular element of an array, then by definition `pa+1` points to the next element, `pa+i` points `i` elements after `pa`, and `pa-1` points `i` elements before `pa`. Thus, if `pa` points to `a[0]`,

```
*(pa+1)
```

refers to the contents of `a[1]`, `pa+i` is the address of `a[i]`, and *(pa+i) is the contents of `a[i]`.

By definition, the value of a variable or expression of type array is the address of element zero of the array. Thus after the assignment

```
pa = &a[0];
```

`pa` and `a` have identical values. Since the the name of an array is a synonym for the location of the initiale element, the assignment `pa = &a[0]` can also be written as

```
pa = a;
```

A reference to `a[i]` can also be written as `*(a+i)`. If `pa` is a pointer, expressions may use it with a subscript; `pa[i]` is identical to `*(pa+i)`. In short, an array-and-index expression is equivalent to one written as a pointer and offset.