9 Arrays II

(9.1) Naming variables. Here are the rules. A name for a variable, etcetera,

- Must begin with a letter, `a, ..., z, A, ..., Z,` or an underscore `_`. It would be very unhelpful to begin a name with an underscore.
- Can then contain any mixture of letters, underscores, and digits `0, ..., 9`.
- C is *case-sensitive,* meaning that `a` and `A` are not considered the same, and so on.

Variables can be grouped together in arrays. For example

```
int days_in_month [ 12 ];
char greeting[6];
```

An array declaration like `int a[4];` has two effects.

- Enough central memory is reserved to store all the array.
- The variable `a` actually contains the *address* of the first array element. If `x` is declared as an array of size 100, then its entries are accessible as `x[0]` up to `x[99]`. `x[100]` **is outside the array bounds.**

Important.
The first array element is indexed 0. Thus, for example, if `a` is an array of 234 elements, the first is a[0] and the last is a[233].
This is unlike most other programming languages.

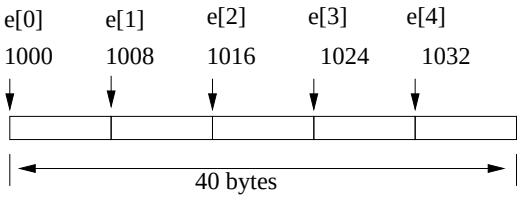
The memory used to store an array.
Recall that `char` occupies 1 byte, `short` 2 bytes, `int, long, float, address` 4 bytes, and `double` occupies 8 bytes.

Multiply these numbers by the number of elements in an array to get the total number of bytes needed.

For example,

```
char a[10];   /* occupies 10 bytes */
short b[3], c[4];  /* occupy 6 and 8 bytes respectively */
int d[1]; /* occupies 4 bytes */
double e [5];  /* occupies 40 bytes */
```

Address of an array element. With `a,b,c,d,e` as above, suppose that `e` begins at memory location 1000 (we give all these addresses as ordinary decimal numbers). Then the elements of `e` are arranged as illustrated.



In general,

> Address of i-th array element = address of 0-th element + $i \times b$, where b is the size of a single element, 1,2,4, or 8 bytes.