Arrays

- Build from fixed-size records.
- Constant access time given the index.
- Space efficient – arrays consist only of the data, so no space is wasted with links or formatting info.
- Easy to iterate over quickly, because of memory locality.
- Cannot adjust their size in the middle of a program's execution.
- *Dynamic arrays* double in size whenever insert index is out of bound (Java's `ArrayList` is dynamic, while `int[]` isn't).
- Java array max length is `Integer.MAX_VALUE = 2^31 - 1` (but could actually be a bit shorter because of reserved memory).
- Element at index i
    - Address - &A[i] or (A+i)
    - Value - A[i] or *(A+i)

```
int A[4];
A[0] = 1; A[1] = 3; A[2] = 4; A[3] = 5;
printf("%d\n", A); // Address of first element (base address of A)
printf("%d\n", *A); // Same as A[0] - the value of first element
printf("%d\n", A+1); // Address of second element
printf("%d\n", *(A+1)); // Same as A[1]
printf("%d\n", *(A+3)); // Same as A[3]
```

- 2D array
    - B[i][j] = *(B[i] + j) = *(*(B+i) + j)

```
int B[2][3]; // B[0] and B[1] are 1D arrays of 3 integers
int (\*p)[3] = B;
// B, &B[0] -> Address
// *B, B[0], &B[0][0] -> Address
// B+1, &B[1] -> Address
// *(B+1), B[1], &B[1][0] -> Address
// *(B+1)+2 -> Value
```