Arrays

Reading assignment: Textbook, Chapter 8, Sections 8.1 — 8.3 (pp. 381 — 394).

Arrays

Definition. An array is a **named** collection (sequence) of two or more adjacent memory cells, which are used to store values *of the same type*.

Arrays are *compound variables*: they store multiple values of the same type **under the same name**.

Properties of Arrays. Arrays have the following properties:

1. Type. The type of an individual value stored in the array.
2. Size. The size of an array is the number of values that can reside in it.
3. Name. The name of an array is a proper C identifier used to reference the array or any of its components in the program.

Array declarations. Arrays are special types of variables. To use them in a program, one must declare them first. The syntax of an array declaration is as follows:

```
<Type> <ArrayName> [<Size>]
```

Here,

`<Type>`	is the type of the array variable (e.g., `int`, `char` or `float`)
`<ArrayName>`	is a C identifier used to name the array variable.
`<Size>`	is a **constant** positive `int` value representing the size of the array

`<Size>` must be either an *int* constant or a `#define`d symbolic `int` constant.

Examples.

```
char Grades[29];  /* array of 29 grades */
unsigned char color[3]; /* array of three color component for an RGB color */
float y[100]; /* array of 100 floating point values (a sample of values of some function */
```

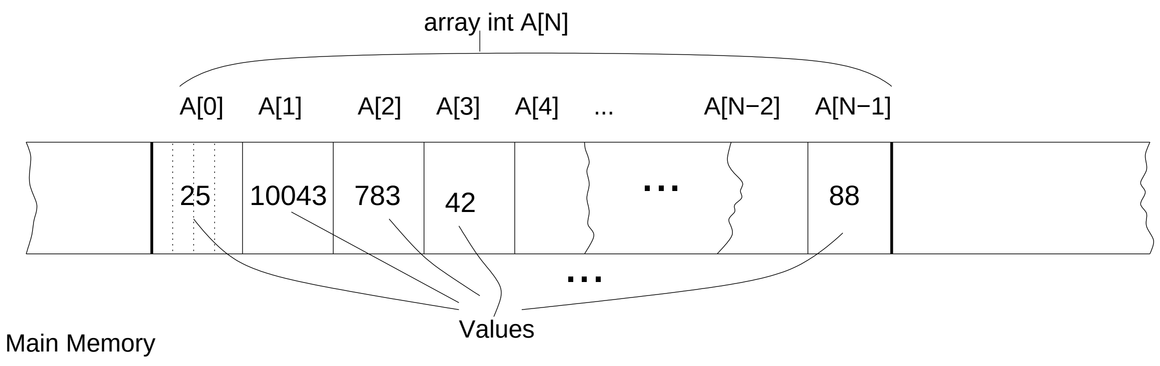

Figure 1: Arrays in a nutshell.

Array Elements. Each value stored in an array is called an array element. Within an array, array elements are identified by an index.

An array index is a number between 0 and size of the array -1.

Array elements are thus arranged in a sequence. Element with index 0 is the first, element with index 1 is the second, etc.. . . . This is illustrated on Figure 1.

Referencing an array element in a program. Individual array elements can be referenced as

```
<ArrayName>[<ArrayIndex>]
```

where `<ArrayName>` is the name of the array and `<ArrayIndex>` is the index of the array element you need in the array. `<ArrayIndex>` can be any `int` expression (i.e, **it DOES NOT NEED** to be constant.

Examples.

```
a[27]
b[12]
z[a+c-1]
```

Array references in programs. Array element references can be used both on the left- and on the right-hand sides of assignment statements. The uses of array element references are the same as the uses of variable names in those places:

- righthand-side of assignments. This is interpreted as the request to compute/return the value of the referenced array element.
- lefthand-side of assignments. This is interpreted as the request assign the value of righthand side of the assignment statement to the referenced array element.

Examples. Array element references on the right:

```
x = a[1] + b[1];
y = a[x+1] - b;
z = a[17];
```

Array element references on the left:

```
 x[2] = 3;
 x[a] = 4;
 x[a-b+1] = 100;
```

Example. The following program computes the squares of numbers from 0 to 9, saves these values in an array and computes their sum.

```
#include <stdio.h>
int main() {

 int i;
 int squares[10];
 int sum = 0;

 for (i=0;i<10;i++) {
    squares[i] = i*i;
    printf("i*i = %d\n", squares[i]);
 }


 for (i=0;i<10;i++) {
   sum = sum+ squares[i];
 }

 printf("Sum of squares: %d\n",sum);

return 0;
}
```

Array initialization. Sometimes, it is convenient to declare an array whose initial state is known. This can be done using the following version of the array declaration statement:

```
 <Type> <ArrayName>[] = { <Value1>, ... , <ValueN>};
```

Notice that the array size is NOT necessary here.

Examples.

```
char grades[] = {'A','B','C','D','E'};
int eVotes[] = {1, 0, 1, 0, 0, 0, 1};
```

Remember that indexing of arrays starts with 0. So, in the example above, `grades[0] = 'A'` while `grades[1] = 'B'`.

Multidimensional Arrays.

Any array is a sequence of vales stored in consecutive memory cells. **One-dimensional arrays** represent this verbatim.

Sometimes, there is a need/desire to represent as arrays collections of values that are best viewed as multidimensional *tables* of data. This can be done by declaring arrays with multiple indexes:

```
<Type> <ArrayName>[<Size1>][<Size2>]...[<SizeN>];
```

Here is an example of a two-dimensional array representing a *tic-tac-toe* board:

```
char ticTacToe[3][3];
```

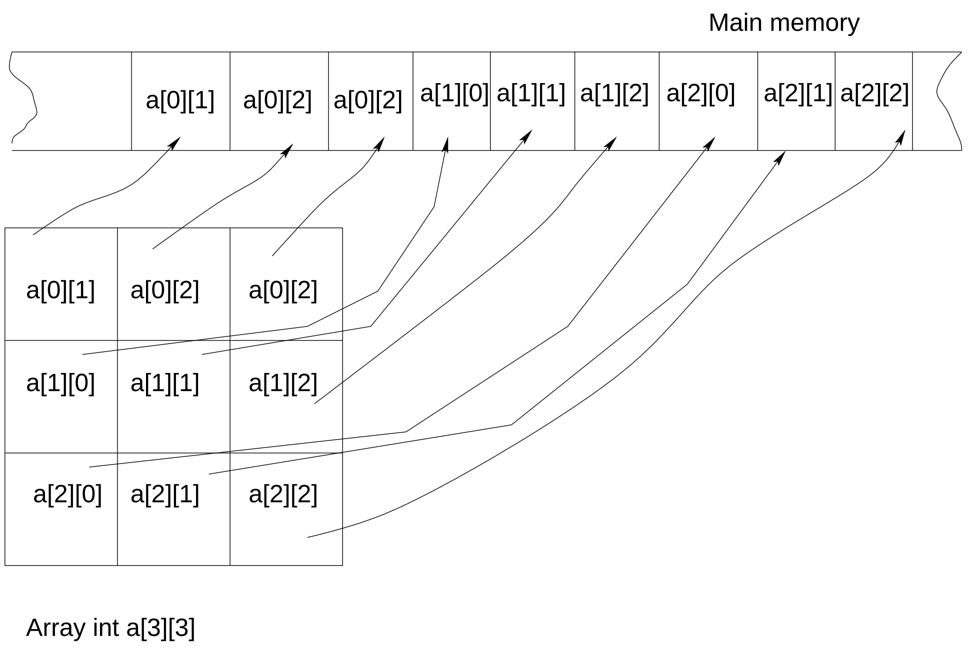

Figure 2: Storing multidimensional arrays in C.

The order of the array elements in the memory will be:

```
ticTacToe[0][0]
ticTacToe[0][1]
ticTacToe[0][2]
ticTacToe[1][0]
ticTacToe[1][1]
ticTacToe[1][2]
ticTacToe[2][0]
ticTacToe[2][1]
ticTacToe[2][2]
```

This is illustrated on Figure 2.