Array T ypes

- a sequence of values of the ***same type***

- a structure of components (structured type)

- Syntax

```
<element type> <array name>[<dimension>]
```

  - o describes a collection of array elements, each of which may be used to store data values of type *<element type>*

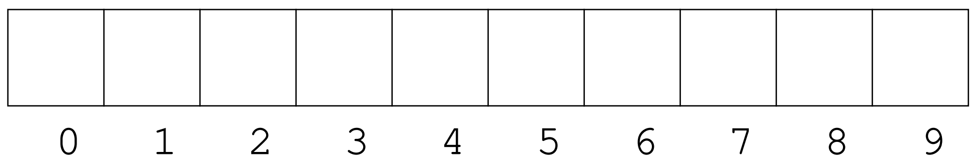
  - o *dimension* specifies the number of element contained in the array

    - must be a constant expression

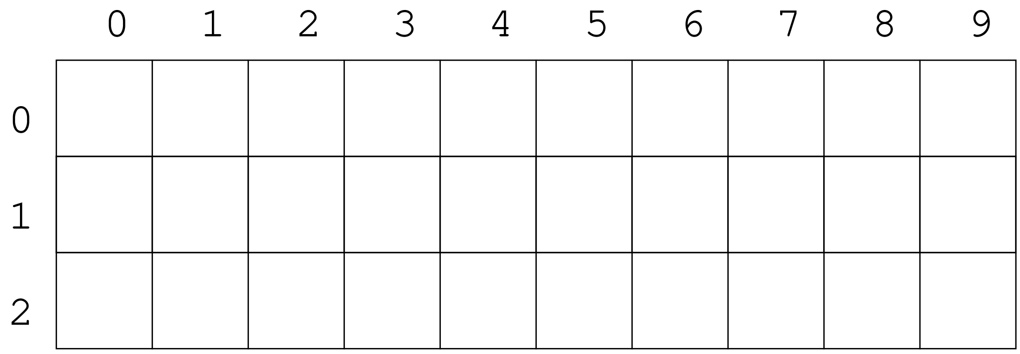
Array Types (Cont'd)

- Example:

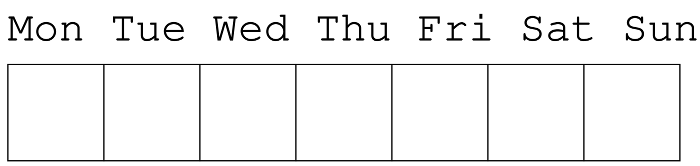
```
int A[10];
```




```
int A[3][10];
```



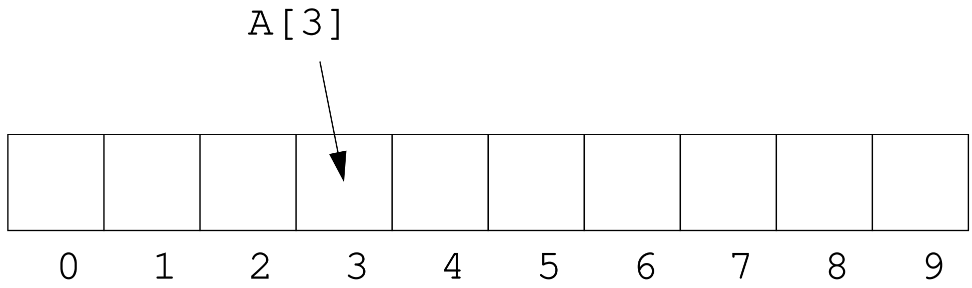

```
enum Day {Mon,Tue,Wed,Thu,Fri,Sat,Sun};
char Week[Sun+1];
```



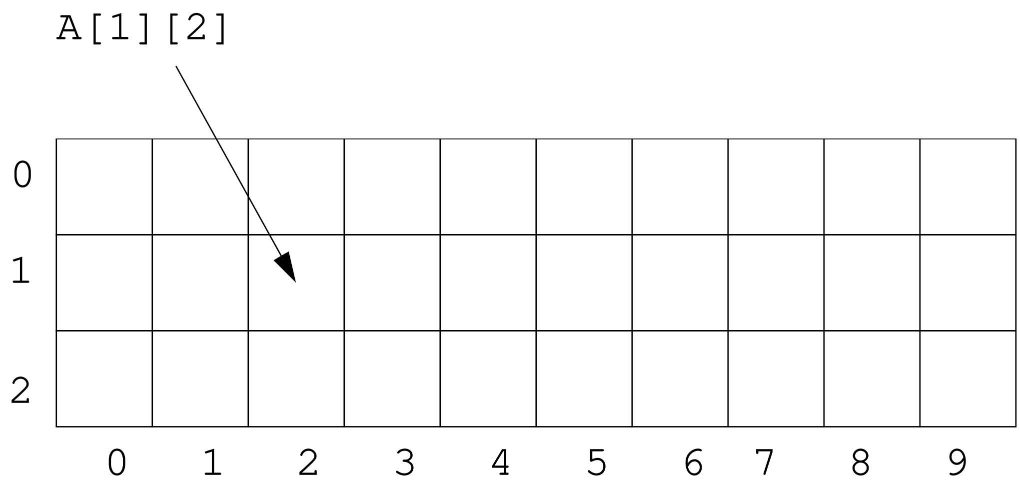
Subscripting

- referring to a component of an array

```
int A[10];
```




```
char A[3][10];
```



Initialization

```
int A[10] = {9,8,7,6,5,4,3,2,1,0}
```

0	1	2	3	4	5	6	7	8	9
9	8	7	6	5	4	3	2	1	0

Assignment

```
int A[10] = {9,8,7,6,5,4,3,2,1,0}

A[3] = -1;
```

0	1	2	3	4	5	6	7	8	9
9	8	7	-1	5	4	3	2	1	0

Array Data Types - Example

```
// Counts lower case letters in a file
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>

// External file names
#define in_file "test"

main() {
     int count[26]; // holds letter count
     ifstream ins;  // ins is input stream

     char next;            // next character

     // open input file, exit on error
     ins.open (in_file);
     if (ins.fail ())
          cerr << "***ERROR: Cannot open "
               << in_file << endl;

     // initialize count
     for (int i=0; i<=25; i++)
          count[i] = 0;

     ins.get(next);  // get first character

     while (!ins.eof()) {
          if (islower(next)) // lower case?
               count[int(next)-int('a')]++;
          ins.get(next);  // get next character
     }

     // print table of letter count
     for (char c='a'; c<='z'; c++)
          cout << c << "   "
               << count[int(c)-int('a')] << endl;

     ins.close();   // close input file stream
     return 0;
```

Arrays as Arguments

- pass individual elements of an array to a function
  - Example:

```
void order(int& x, int& y) {
  int temp;

  if (x > y) {
    temp = x; x = y; y = temp;
  }
  return;
} // end order

main() {
  int A[10] = {9,8,7,6,5,4,3,2,1,0};

  order(A[3], A[5]);
  return 0;
}
```

Before:

9	8	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	8	9

After:

9	8	7	***4***	5	***6***	3	2	1	0
0	1	2	3	4	5	6	7	8	9

Arrays as Arguments (Cont'd)

- pass an **entire** array to a function
  - never by value
  - simulated pass by reference (address of first element)
  - Example

```
// Find the largest value in an array
// input:   n - number of elements to check
//          a[] - array of elements
// output:index to the largest element
int max_element(int n, const int a[]) {
  int max_index = 0;

  for (int i=1; i<n; i++)
    if (a[i] > a[max_index])
      max_index = i;

  return max_index;
} // end max_element;

#include <iostream.h>
main() {
  int A[10] = {9,8,7,6,5,4,10,2,1,0};

  cout << A[max_element(10,A)] << endl;
  return 0;
}
```

Searching

Problem Definition:

Search an *unordered* array of integers for a value and return its index if the value is found. Otherwise, return -1.

0	1	2	3	4	5	6	7
10	7	9	1	17	30	7	6

Solution Outline:

```
Start with the first array element (index 0).

While value not found and more array element do
  Try next element (increment index);

If value not found then
  return position to -1
else
  return index;
```

Searching (Cont'd)

```
// Searches an unordered array of integers
int search(int data[], // input: array
           int size,   // input: array size
           int value   // input: value to find
           )   // output: index if found
               //         otherwise return -1
{
     int index = 0;    // initialize index to 0

     while ((data[index] != value) &&
            (index < size))
          index++;

     if (data[index] != value)
          return -1;
     else
          return index;
} // search
```

```
#include <iostream.h>
int main() {
     int A[10] = { 5,2,4,6,9,2,25,5,18,3 };
     int x;

     cin >> x;
     cout << search(A,10,x) << endl;

     return 0;
}
```

Example

```
// Add a[i] and b[i] and store the sum in c[i]

void  add_array
         (int size,        // in: array size
          const float a[],// in: first array
          const float b[],// in: second array
          float c[] )     // out: result array

// array elements with subscripts ranging from
// 0 to size-1 are added element by element
// Pre: a[i] and b[i] (0<=i<=size-1) are defined
// Post: c[i] = a[i] + b[i] (0<=i<=size-1)

{
    int i;

    // Add a[i] and b[i] and store result in c[i]
    for (i=0; i < size; i++)
      c[i] = a[i] + b[i];

    return;

}
```

Storage of Multidimensional Arrays

- Example:

```
char a[3][3];
```

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]

- most compiler store multidimensional arrays in adjacent memory cells

  o ***row-major*** order

[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]
0	1	2	3	4	5	6	7	8

```
offset = ix3 + j
```

  o column-major order

[0][0]	[1][0]	[2][0]	[0][1]	[1][1]	[2][1]	[0][2]	[1][2]	[2][2]
0	1	2	3	4	5	6	7	8

```
offset?
```

Initializing & Passing Multidimensional Array

```
// Initializing multidimensional arrays
#include <iostream.h>

             size of subsequent subscript required

void printArray(int [][3]);

main() {
     int a1[2][3] = { {1,2,3}, {4,5,6} },
         a2[2][3] = { 1,2,3,4,5,6 },
         a3[2][3] = { 1,2,3,4 },
         a4[2][3] = { {1,2}, {4} };

     cout << endl; printArray(a1);
     cout << endl; printArray(a2);
     cout << endl; printArray(a3);
     cout << endl; printArray(a4);

     return 0;
}

void printArray(int a[][3]) {
     for (int i=0; i<=1; i++) {
          for (int j=0; j<=2; j++)
               cout << a[i][j] << ' ';
          cout << endl;
     }
}
```

Using 1-D array instead of m-D array

- instead of declaring

```
int a[2][3][4];
```

- we can declare

```
int a1[24];
```

- whenever we want to access component

```
a[i][j][k]
```

- we access component

```
a1[i*12 + j*4 + k]
```