

Arrays in C++

July 19, 2010

What's an Array?

1d Array A location in memory with space for multiple variables of the same type.

Array Element One of the variables in an array.

What's an Array?

1d Array A location in memory with space for multiple variables of the same type.

Array Element One of the variables in an array.

- ▶ How are they declared?
- ▶ How are they initialized?
- ▶ How are specific elements accessed and modified?

Allocate Space for Arrays

These allocate **space** for the arrays, but do not specify element values.

Space for 3 integers, named intArray:

```
int intArray[3];
```

Space for 5 doubles, named dblArray:

```
const int DBLARRAY(5);  
double dblArray[DBLARRAY];
```

Allocate Array Space by Initialization

These allocate **space** implicitly by specifying the value of each array element.

Space for 3 integers, named `intArray`, with specific values:

```
int intArray[] = { 12, 0, 2 };
```

Space for 5 double values, named `dblArray`:

```
double dblArray[] = { 12, 0, 2.3, -5, 6 };
```

Error: arrays are not associative in C++

```
double array[] = { 1, 1.0, "3.14"};
```

Allocate Array Space by Size and Value

These allocate arrays of specific sizes, with values for initial elements.

All remaining elements are initialized to 0 (zero).

Space for 3 integers, named `intArray`, with specific values:

```
int intArray[3] = { 12, 0, 2 };
```

Space for 5 double values, named `dblArray`, with **some** initial values:

```
double dblArray[5] = { 12, 0, 2.3 };
```

Space for `LENGTH` bools, named `truthSequence`, **all** initialized to `false`*:

```
const int LENGTH = 1024;  
bool truthSequence[LENGTH] = { };
```

*Technically, I'm not convinced the last form is allowed by the C++ Language Specification, but practically all compilers allow this form.

Array Declaration Rule

Error: size and initial value list must agree

```
double oops[3] = { 4.5, 3.2, 2.1, 3.14 };
```

Error: at least one of size or initial values must be provided

```
int ugh[] = {}; // NO INITIAL VALUES!
```

Array Declaration Rule

Error: size and initial value list must agree

```
double oops[3] = { 4.5, 3.2, 2.1, 3.14 };
```

Error: at least one of size or initial values must be provided

```
int ugh[] = {}; // NO INITIAL VALUES!
```

Array sizes must be specified at array declaration —
explicitely or implicitly.

Try question 1.

Array Memory

```
1 const int SIZE(5);  
2 double array[SIZE] = { 2.3, -5.1, 6 };
```

Array Values:	2.3	-5.1	6	0	0	?
Element Number:	1	2	3	4	5	DNE
Element Offset:	0	1	2	3	4	5

Array Memory

```
1 const int SIZE(5);  
2 double array[SIZE] = { 2.3, -5.1, 6 };
```

Array Values:	2.3	-5.1	6	0	0	?
Element Number:	1	2	3	4	5	DNE
Element Offset:	0	1	2	3	4	5

What is special about the red 0s?

Array Memory

```
1 const int SIZE(5);  
2 double array[SIZE] = { 2.3, -5.1, 6 };
```

Array Values:	2.3	-5.1	6	0	0	?
Element Number:	1	2	3	4	5	DNE
Element Offset:	0	1	2	3	4	5

What is special about the red 0s?

Why DNE?

Array Memory

```
1 const int SIZE(5);  
2 double array[SIZE] = { 2.3, -5.1, 6 };
```

Array Values:	2.3	-5.1	6	0	0	?
Element Number:	1	2	3	4	5	DNE
Element Offset:	0	1	2	3	4	5

What is special about the red 0s?

Why DNE? What does the question mark mean?

Array Memory

```
1 const int SIZE(5);  
2 double array[SIZE] = { 2.3, -5.1, 6 };
```

Array Values:	2.3	-5.1	6	0	0	?
Element Number:	1	2	3	4	5	DNE
Element Offset:	0	1	2	3	4	5

What is special about the red 0s?

Why DNE? What does the question mark mean?

Which row of this table represents a segment of computer memory?

Array Memory

```
1 const int SIZE(5);
2 double array[SIZE] = { 2.3, -5.1, 6 };
```

Array Values:	2.3	-5.1	6	0	0	?
Element Number:	1	2	3	4	5	DNE
Element Offset:	0	1	2	3	4	5

What is special about the red 0s?

Why DNE? What does the question mark mean?

Which row of this table represents a segment of computer memory?

Why is Element Offset blue?

Manipulating (Partially Filled) Array Elements

```

1  /**
2   * Read in at most 8 percentiles from cin and print
3   * them back out converted to fractional equivalents.
4   */
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     const int MAXPOINTS( 8 );
11     double percents[MAXPOINTS]; // values will be read from cin
12
13     // prompt and read
14     cout << "Enter percents at the keyboard (with percent sign);" << endl;
15     cout << "Maximum " << MAXPOINTS << " points ,or 'end' to stop:" << endl;
16
17     // read, count with p
18     int p = 0;
19     while( p < MAXPOINTS ) {
20         char percent_sign;
21         if( !( cin >> percents[p] >> percent_sign ) ) {
22             // read failure
23             break;
24         }

```

Run

Edit

Answer question 5.

Beware...

- ▶ Array indices must be **integral** variables (bool, char, int).

- ▶ Arrays do not have a ()-style initialization syntax, so

Arrays **cannot** be initialized with

```
double array[]( {1.0,2,3} );           //will not compile
```

- ▶ An array **can not be printed** to the console or disk file with a **simple** cout statement:

```
cout << intArray << endl;
```

a for loop must be used to **iterate** and print each element individually.

- ▶ Arrays can not have multiple element values set with one assignment operation.

```
int intArray[3];
```

```
intArray = { 10, 11, 12 };           //will not compile
```

What's a 2d-Array?

2d Array An array of arrays!

- ▶ We need at least two sets of `{ }` to specify non-trivial initialization values,
- ▶ We need two sets of `[]` for declaration and element access, ...
- ▶ ... we treat these two `[]`-operators as the ROW and COLUMN specifier for the array.

Think of a 2d C++ array as a matrix in mathematics.

$$\text{array}[\text{ROWS}][\text{COLS}] \sim \text{array}[3][4] \sim \begin{bmatrix} 1 & 2 & 3 & 4 \\ -2 & 1 & 2 & 1 \\ -3 & -2 & 1 & 0 \end{bmatrix}$$

2d Array Example

```

1  /**
2  * Read in at most an N x N integer array and determine if
3  * it is symmetrical or not.
4  */
5  #include <iostream>
6  #include <cstdlib> // exit
7  using namespace std;
8
9  int main()
10 {
11     const int N( 8 );
12     int A[N][N]; // Matrix of max size NxN
13     int n; // user specified matrix size
14
15     // prompt and read
16     cout << "Enter the matrix dimension:_" << flush;
17     if( !( ( cin >> n ) && n >= 1 && n <= N ) ) {
18         cout << "Error: n is invalid_" << n << " " << endl;
19         // system("PAUSE"); // uncomment for windoze */
20         exit(1);
21     }
22     cout << "Enter the matrix_" << n << 'x' << n << "):_" << flush;
23     for( int i(0); i<n; i++ ) {
24         for( int j(0); j<n; j++ ) {

```

Run

Edit

Declaring 2d Arrays

Declare Space Only

```
2 // Declare space for arrays
3 int intArray[3][4];
4 const int ROWS(3), COLS(4);
5 double dblArray[ROWS][COLS];
```

$$\text{intArray}[3][4] = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$
$$\text{dblArray}[3][4] = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

Declaring 2d Arrays

Declare & Initialize All Elements

```

9 // Declare and initialize all values
10 const int ROWS(3), COLS(4);
11 int intArray[ROWS][COLS] = {
12     { 1, 2, 3, 4 },
13     { -2, 1, 2, 3 },
14     { -3, -2, 1, 2 }, // lingering comma OK
15 };
16
17 // Note the extra comma is ok...
18 double dblArray[2][2] = {
19     { 1, 1.0, }, // extra internal comma OK
20     { 2, 2.0, },
21 };
22
23 // Note the missing first [ROWS] parameter!
24 double dblArray2[][3] = {
25     { 1, 2, 3 },
26     { -2, 1, 2 },
27     { -3, -2, 1 },
28 };

```

$$\text{intArray}[3][4] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -2 & 1 & 2 & 3 \\ -3 & -2 & 1 & 2 \end{bmatrix}$$

$$\text{dblArray}[2][2] = \begin{bmatrix} 1.0 & 1.0 \\ 2.0 & 2.0 \end{bmatrix}$$

$$\text{dblArray2}[][3] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ -2.0 & 1.0 & 2.0 \\ -3.0 & -2.0 & 1.0 \end{bmatrix}$$

Declaring 2d Arrays

Declare & Use Default Zero Initialization

```

32 // Declare and initialize some values
33 double dblArray[5][3] = {
34   { 1, 2, 3 },
35   { 1, 2 },
36   { 0 },
37   {},
38   // MISSING 5th row (offset 4)
39 };
40
41 int intArray[5][3] = {};

```

$$\text{dblArray}[5][3] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{intArray}[5][3] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Limits on Implicitly Declared Size

```

2 // 2 implicit dimensions
3 // DOES NOT WORK!
4 double dblArray[][] = {
5     { 1, 1.0, },
6     { 2, 2.0, },
7     };
8
9 // Implicit columns
10 // DOES NOT WORK!
11 int intArray[4][] = {
12     { 1, 2, 3, 4 },
13     { -2, 1, 2, 3 },
14     { -3, -2, 1, 2 },
15     };
16
17 // Only implicit rows WORKS
18 double dblArray2[][3] = {
19     { 1, 2, 3 },
20     { -2, 1, 2 },
21     { -3, -2, 1 },
22     };

```

Only the first dimension of any multidimensional array can be implicitly determined by the compiler!

4: error: declaration of 'dblArray' must have bounds for all dimensions except the first
 11: error: declaration of 'intArray' must have bounds for all dimensions except the first

Try question 6 parts a–c, and f.

Magical C++ Strings

```
1 /**
2  * This application demonstrates the C++ string api
3  */
4
5 #include <cstdlib>
6 #include <iostream>
7 #include <string>
8 using namespace std;
9
10 int main( )
11 {
12     const string theString( "Ramblin'_wreck_from_Golden_Tech," );
13
14     // string objects support a member function length() and array (like)
15     // access to char values!
16     for( int i(0); i<theString.length(); /* empty */ ) {
17         cout << i << ":" << theString[i] << "\t";
18         if( !(++i%5) ) cout << endl;
19     }
20     cout << endl;
21
22     // the member function find(...) returns the index of a searched for string,
23     // or -1 if the string cannot be found.
24     cout << "wreck_is_at_index_" << theString.find( "wreck" ) << endl;
```

[Run](#)[Edit](#)

Cycling Through Arrays

Use `%` to keep index or offset arithmetic within the bounds of array dimensions.

Modulus is usually faster and more memory efficient than `if` statements.

Try question 7

Serialized Element Offsets in 2d Arrays

If the elements of an $M \times N$ 2d array are valued in (western) page-reading order beginning with 0...

$$\begin{bmatrix} 0 & 1 & \dots & N-1 \\ N & N+1 & \dots & \dots \\ \dots & \dots & MN-2 & MN-1 \end{bmatrix}$$

the row index of an element with value x is x/N ,

the column index of an element with value x is $x \% N$.

Binary Search

Suppose you have a sorted array of 12 numbers.

13	25	26	29	30	43	49	53	61	79	85	93
----	----	----	----	----	----	----	----	----	----	----	----

Is there an efficient way to determine if some other number, x is in the array?

Binary Search

Take $x = 54$ as an example.

We could start in the middle, noticing that $x > 43$

13	25	26	29	30	43	49	53	61	79	85	93
----	----	----	----	----	----	----	----	----	----	----	----

... then notice $x < 61$

13	25	26	29	30	43	49	53	61	79	85	93
----	----	----	----	----	----	----	----	----	----	----	----

... and now $x > 49$

13	25	26	29	30	43	49	53	61	79	85	93
----	----	----	----	----	----	----	----	----	----	----	----

... but $53 < x < 61$

13	25	26	29	30	43	49	53	61	79	85	93
----	----	----	----	----	----	----	----	----	----	----	----

$x = 54$ is **not** in the table, it **would have been** between 53 and 61.

13	25	26	29	30	43	49	53	61	79	85	93
----	----	----	----	----	----	----	----	----	----	----	----

Binary Search

Suppose a **haystack**: a **sorted** array of n numbers, and a **needle** (x in the previous example).

The **binary search** algorithm is the most efficient way to answer:

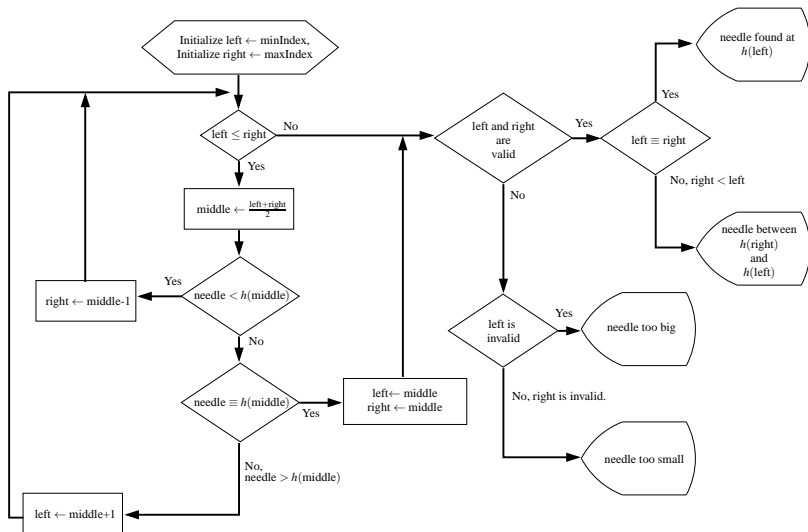
- ▶ is the needle in the haystack?
- ▶ is the needle too small for the haystack?
- ▶ is the needle too big for the haystack?
- ▶ or where it should be in the haystack?

Binary Search

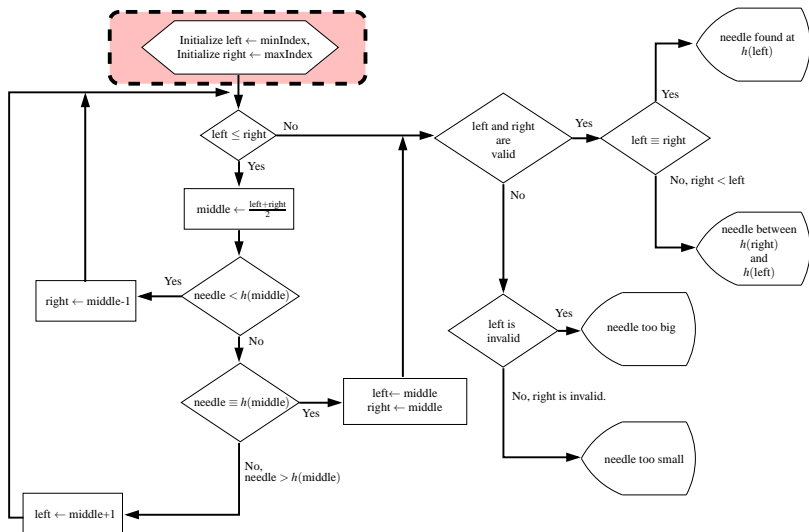
Suppose needle=54; ...

Loop	Variables			Haystack Element Indices											
	left	right	middle	0	1	2	3	4	5	6	7	8	9	10	11
1	0	11	$5 = \frac{0+11}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle > haystack[middle] \Rightarrow left = middle + 1;														
2	6	11	$8 = \frac{6+11}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle < haystack[middle] \Rightarrow right = middle - 1;														
3	6	7	$6 = \frac{6+7}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle > haystack[middle] \Rightarrow left = middle + 1;														
4	7	7	$7 = \frac{7+7}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle > haystack[middle] \Rightarrow left = middle + 1;														
	8	7		13	25	26	29	30	43	49	53	61	79	85	93

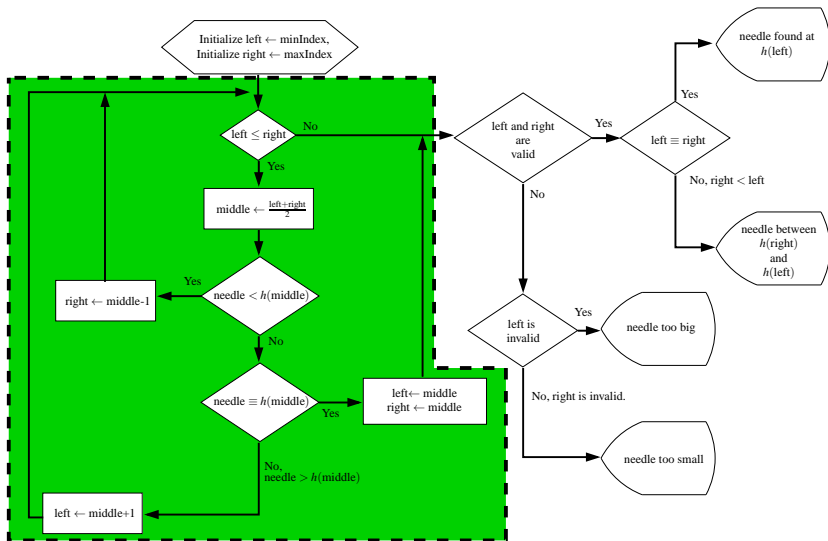
Binary Search



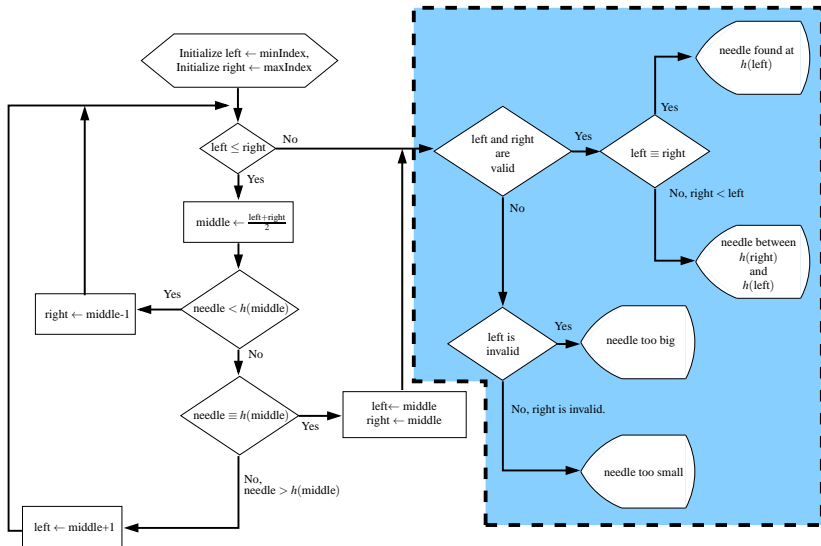
Binary Search — Initialize Variables



Binary Search — while () Loop Search



Binary Search — Interpret Results



Binary Search — Practice!

Suppose `needle=10`; ...

Loop	Variables			Haystack Element Indices											
	left	right	middle	0	1	2	3	4	5	6	7	8	9	10	11
1	?	?	?	13	25	26	29	30	43	49	53	61	79	85	93
	needle ??? haystack[middle] ⇒ ???														

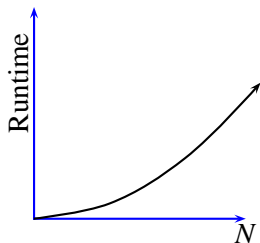
Binary Search — Practice!

Suppose `needle=10`; ...

Loop	Variables			Haystack Element Indices											
	left	right	middle	0	1	2	3	4	5	6	7	8	9	10	11
1	0	11	$5 = \frac{0+11}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle < haystack[middle] \Rightarrow right = middle - 1;														
2	0	4	$2 = \frac{0+4}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle < haystack[middle] \Rightarrow right = middle - 1;														
3	0	1	$0 = \frac{0+1}{2}$	13	25	26	29	30	43	49	53	61	79	85	93
	needle < haystack[middle] \Rightarrow right = middle - 1;														
	0	-1		13	25	26	29	30	43	49	53	61	79	85	93

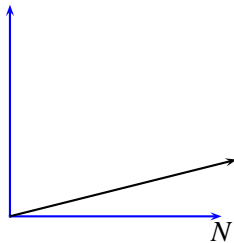
Big-O & Asymptotic Runtimes

Quadratic Time



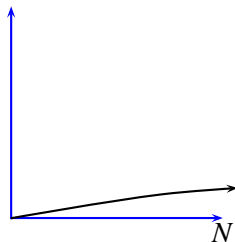
```
// Quadratic Time
for(int i=0; i<N; i++){
    for(int j=0; j<sqrt(N); j++){
        // Work done in j-loop
        // proportional to the
        // value of i
    }
}
```

Linear Time



```
// Linear time
for(int i=0; i<N; i++){
    // A constant amount of
    // work for i=0, i=1,
    // ... i=N-2, i=N-1
}
```

Log Time



```
// Log time
left = 0; right = N-1;
while( left <= right ) {
    middle = (left+right)/2;
    // Each iteration of
    // the loop reduces
    // the "size" of the
    // problem by 1/2.
}
```

Data Centric Programming

“Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won’t usually need your flowcharts; they’ll be obvious.”

— Fred Brooks in *The Mythical Man-Month*

Data centric programming puts the emphasis on the **data**, and less so on the **logic**. Some problems are difficult or **impossible** to approach from the vantage point of data centric programming.

When data centric programming can be used, it requires insight and a fair amount of thought. The reward is an application that is short, sweet, and to the point. The results can be surprising.

Paper-Rock-Scissors **without** `if/if-else` Statements

finis