Arrays

Lecture 14

Arrays

- An *array* is a container of variables, all in a sequence, all of the same type
- For example, say you need to keep track of the cost of 1000 items
- You could declare 1000 double variables: `double cost1, cost2, cost3, cost4, …`
- Or you could use an array!

Declaring Arrays (1)

- Declaring an array is similar to declaring other variables
- Start with the variable type (**`double`**, **`int`**, **`char`**, **`string`**, …)
- Then comes the array name (**`cost`**, **`x`**, **`vals`**, …)
- The new piece is the number of *elements* in the array, or the total number of values that the array can hold, which is put in brackets
- Example that declares an array named cost that holds 1000 **`double`** values:

```
double cost[1000];
```

Declaring Arrays (2)

- You can think of an array declaration as declaring the same number of individual variables

- Example declaring an array of 8 integers named counts:

```
int counts[8];
```

- This is similar (but not exactly the same) to declaring 8 separate integers:

```
int counts0, counts1, counts2, counts3,
    counts4, counts5, counts6, counts7;
```

Accessing Array Elements

- To actually use an individual element in the array, you specify the *index* of the element in brackets
- Be careful not to confuse the two uses of brackets (declaration versus use)
- Example array of 15 integers named values, and setting the value at index 7 to 10:

```
int values[15]; // declare any array of 15 ints named values
values[7] = 10; // give element at index 7 a value of 10
```

Arrays in Memory

Arrays are stored in memory so that all the elements in the array are next to each other, in order:

```
int counts[8];
```

address	value	variable
1000	5	counts[0]
1004	-8	counts[1]
1008	0	counts[2]
1012	-4	counts[3]
1016	17	counts[4]
1020	4	counts[5]
1024	103	counts[6]
1028	3	counts[7]
1032	42	
	...	

Array Elements

- Arrays start at index 0 and go through index size–1
- Arrays do NOT start at index 1!
- Array indices do not have to be hard coded, they can be any expression that evaluates to an integer
- Example of initializing an array so that all elements have an initial value of 0:

```
double temperatures[64];
for( int i=0; i<64; i++ )
{
    temperatures[i] = 0;
}
```

Out of Bounds Errors

- You always have to ensure that your program only uses valid elements/indices for an array
- You can never access an index of less than 0
- You can never access an index greater than or equal to the size of the array
- If you try to access an element outside of the bounds of the array, Visual Studio will give you a run-time debug error

```
int my_array[10];
my_array[0] = 5;  // ok
my_array[9] = -6; // ok
my_array[-1] = 0; // run-time error!
my_array[10] = 3; // run-time error!
```

Exercise

What is the output of the below code?

```
int vals[4];
vals[2] = 3;
vals[0] = 2;
vals[1] = 1;
vals[3] = vals[2];
for ( int i=0; i<4; i++ )
{
    cout << vals[i] << endl;
}
```

Answer

2

1

3

3

Exercise

Write a program that declares an array of 1000 integer values and initializes all 1000 values to 1

Answer

```
#include <iostream>
using namespace std;

int main()
{
    int a[1000];
    for ( int i=0; i<1000; i++ )
    {
        a[i] = 1;
    }
    return 0;
}
```

Initializing Arrays

- You can also initialize arrays when you declare them using special syntax
- Example:

```
int pages[6] = { 513, 343, 279, 409, 651, 222 };
```

- Above example is equivalent to:

```
int pages[6];
pages[0] = 513;
pages[1] = 343;
pages[2] = 279;
pages[3] = 409;
pages[4] = 651;
pages[5] = 222;
```

Array Elements

You can use any one element of an array anywhere you can use a variable of the same type

- Assigning values
- In equations
- With **`cin`** or **`cout`**
- With file streams
- As function arguments (by value or reference)

…

Example

```
#include <iostream>
using namespace std;
int do_something(int a, int& b);

int main()
{
	int x, vals[5];
	for ( int i=0; i<5; i++ )
	{
		vals[i] = ( i*i );
	}
	x = vals[4] * vals[3] + vals[1];
	vals[0] = x - vals[2];
	vals[2] = do_something( vals[1], vals[3] );
	for ( int i=0; i<5; i++ )
	{
		cout << "vals[" << i << "]=" << vals[i] << endl;
	}
	return 0;
}

int do_something(int a, int& b)
{
	b = a * 10 - 2;
	return ( a*b );
}
```

Arrays as Function Arguments

- Arrays can be passed as function arguments also
- Use *array* arguments, which are basically call by reference arguments
- Do NOT use the **&** symbol (you will get a build error)
- Any changes made to array elements in the function are permanent after the function is finished
- The array size is not directly accessible in a function, so you always have to pass the size as an additional argument

Example

```
#include <iostream>
using namespace std;

void fill_array(int a[], int size);

int main()
{
	int my_array[12];
	fill_array( my_array, 12 );
	for ( int i=0; i<12; i++ )
	{
		cout << "my_array[" << i << "]=" << my_array[i] << endl;
	}
	return 0;
}

void fill_array(int a[], int size)
{
	cout << "Please enter " << size << " integers:" << endl;
	for ( int i=0; i<size; i++ )
	{
		cin >> a[i];
	}
}
```

Another Example

```
#include <iostream>
using namespace std;
void fill_array(int a[], int size);
void print_array(int a[], int size);
int main()
{
	int my_array[12];
	fill_array( my_array, 12 );
	print_array( my_array, 12 );
	return 0;
}
void fill_array(int a[], int size)
{
	cout << "Please enter " << size << " integers:" << endl;
	for ( int i=0; i<size; i++ )
	{
		cin >> a[i];
	}
}
void print_array(int a[], int size)
{
	for( int i=0; i<size; i++ )
	{
		cout << "a[" << i << "]=" << a[i] << endl;
	}
}
```

Exercise

Write a function named **`add_one()`** that increments every value in an array by one. The array must be passed as an argument to **`add_one()`**, and the function should work for arrays of any size.

Answer

```
#include <iostream>
using namespace std;

void print_array(int a[], int size);
void add_one(int arr[], int size);
int main()
{
     int b[10] = { 1,2,3,4,5,6,7,8,9,10 };
     add_one( b, 10 );
     print_array( b, 10 );
     return 0;
}
void print_array(int a[], int size)
{
     for ( int i=0; i<size; i++ )
     {
          cout << "a[" << i << "]="
               << a[i] << endl;
     }
}
```

```
void add_one(int arr[], int size)
{
     for ( int i=0; i<size; i++ )
     {
          arr[i]++;
     }
}
```

Partially Filled Arrays

- Arrays do not have to be completely "full"
- Sometimes only some of the entries in the array have values
- You have to be careful to ensure that you only use elements that have been initialized
- Example: reading an unknown number of values from the user or a file

Example

```
int main()
{
	int inputs[20];
	int next_value;
	int last_index;

	cout << "Enter up to 20 positive values, stopping with a negative value:\n ";
	for ( last_index=0; last_index<20; last_index++ )
	{
		cin >> next_value;
		if ( next_value < 0 )
		{
			break;
		}
		inputs[last_index] = next_value;
	}

	print_array( inputs, last_index );

	return 0;
}
```

Using CTRL-D

- Recall that you can use the **`>>`** operator with **`ifstream`** variables to continue reading values from a file until **`EOF`** (end of file) is reached

- You can do the same with **`cin`**, by using a special key combination that tells C++ that there are no more values to read from the keyboard

- Hold down the Control key (**`Ctrl`**) and hit the D key

- That will ensure that the next **`cin >>`** operation will break out of a loop

Example

```
int main()
{
     int inputs[20];
     int last_index;

     cout << "Enter up to 20 values, stopping with CTRL-D: " << endl;
     last_index = 0;
     while ( cin >> inputs[last_index] )
     {
          last_index++;
          if ( last_index == 20 )
          {
               break;
          }
     }

     print_array( inputs, last_index );

     return 0;
}
```

Searching an Array

- Sometimes you want to search an array for a particular value or target
- Look through every element and return the index of one matching element (usually the first)
- If no element matches the target then usually return -1, since that is never a valid index

Example

```
#include <iostream>
using namespace std;

int search_array(int a[], int size, int target);
int main()
{
	int values[10] = { 4,11,-3,0,46,11,9,-77,3,11 };
	int target_value, index;

	cout << "Enter a value to search for: ";
	cin >> target_value;

	index = search_array( values, 10, target_value );
	if ( index == -1 )
	{
		cout << target_value
		     << " was not found" << endl;
	}
	else
	{
		cout << target_value
		     << " was first found at position "
		     << index << endl;
	}
	return 0;
}
```

```
int search_array(int a[],int size,int target)
{
	for ( int i=0; i<size; i++ )
	{
		if ( a[i] == target )
		{
			return i;
		}
	}
	return -1;
}
```

Wrap Up

- Arrays are useful when you need to keep track of many related values
- Arrays are almost always used together with loops
- Array elements can be used anywhere a single variable of the same type can be used
- Entire arrays can be passed to functions as array arguments, which are similar to call by reference arguments