COMSC-165 Lecture Topic 6
Arrays and Vectors in C++

Reference
Deitel, chapter 6.1-6.10
Tutorial

Concept Of An Array
a variable that can hold multiple values *at the same time*
  named just like any variable (like `a`)
  used "index" to access any specific value (like `a[3]`)
    index is zero-based whole number
a value at an index is called an "element" of the array
4 kinds of arrays: static, dynamic, vector, C++11 array

Array "Size" And Overrunning An Array
array "size" is *how many* values it can hold at once
e.g., for array `a` of size 100:
  use index range 0 to 99
`a[1000]` and `a[-100]` will compile
  ...and *sometimes* work!

BUT going outside the range leads to
  unpredictable results at runtime

Tracking Array Size
for static and dynamic, programmer has to track size
  using a separate `int` *or* a "sentinel"
for vector and C++11 array, arrays track their own size

Static Arrays In C and C++
fastest, and least overhead
declaration: `int a[100]; // sized to store 100 int values`
  or `char` or `double` or any data type...
  size can be a number or `const int`
    but *never* a variable `int`
first *element*: **a[0]**, last element: **a[99]** (for size 100)
curly-brace initialization

C++ `vector`s
"objects" that track their own size *and* are resizable

```
#include <vector>
using std::vector;
...
  vector<int> a(10);
  ...
  for (int i = 0; i < a.size(); i++)
  {
    ...  a[i]  ...
  }
  ...
  a.resize(20);
  ...
```

C++11 `array`s
like static arrays, but "objects" that track own size

```
#include <array>
  array<int, 10> a;
```

or `array<int, 10> x{{1, 2, 3, 4}};`

Arrays And C++11 Range for-loops
for vectors or C++11 arrays
`for (auto value: a)`
  and use "value" instead of "a[i]"

```
int matchThis = ...;
bool found = false; // an "accumulator"
for (auto value: a)
  if (value == matchThis)
    found = true;
```

Static and Dynamic Arrays As Function Parameters
memory address of the start of the array is shared
  and the data type of its elements
no pass-by-value option -- it's *always* by reference
options for sharing size:
  write size as whole number, everywhere
  an `int` parameter
  declare N globally
  mark the end with a sentinel value

```
int x[3] = {1, 2, 3};
int y[3] = {}; initialize all to zero
int z[] = {1, 2, 3};
char alphabet[26] = {'a', 'b', 'c'};
```

in C++11: `int a[]{1, 2, 3};` // no "equals" needed
`cout << a;` displays memory address of array
unless it's a `char` array: special case

Arrays And C/C++99 for-loops

```
for (i = 0; i < 100; i++)
  ...a[i]...
```

Summing An Array's Values

```
int sum = 0; // an "accumulator"
for (int i = 0; i < 100; i++)
  sum += a[i];
```

Searching An Array

```
int matchThis = ...;
bool found = false; // an "accumulator"
for (int i = 0; i < 100; i++)
  if (a[i] == matchThis)
    found = true;
```

Searching An Array For Min/Max

```
int min = a[0];
int max = a[0];
for (int i = 1; i < 100; i++)
{
  if (min > a[i]) min = a[i];
  if (max < a[i]) max = a[i];
}
```

Sorting An Array

```
for (int i = 0; i < 100; i++)
  for (int j = i + 1; j < 100; j++)
     if (a[j] < a[i]) // swap
     {
       int temp = a[i];
       a[i] = a[j];
       a[j] = temp;
     }
```

or `swap(a[i], a[j]);`
with `using std::swap;` in `#include <algorithm>`

Dynamic Arrays In C++

flexible sizing
declaration: `int* a = new int[n];` // sized to store n `int` values

the `const` keyword and arrays

syntax:
prototype: `void fun(int*, int);` // for array name and size
call: `fun(a, 100);`
function header: `void fun(int* a, int n);`

alternate syntax:
prototype: `void fun(int[], int);` // for array name and size
call: `fun(a, 100);`
function header: `void fun(int a[], int n);`

Vectors and C++11 Arrays As Function Parameters

no need to deal with size separately
pass-by-value is the default
use special syntax to pass by reference
prototype: `void fun(vector<int>&);`
call: `fun(a);`
function header: `void fun(vector<int>& a)`

prototype: `void fun(array<int, 100>&);`
call: `fun(a);`
function header: `void fun(array<int, 100>& a)`

Arrays As Function Returns

cannot return a static array

ok to return a dynamic array, but remember `delete []`

ok to return vector or C++11 array

2D Arrays

table with rows and columns of cells
declaration `int x[10][6]`
#rows (1st number)
#columns (2nd number)
reference `x[10][2]`, starting from 0,0
initialization

```
int x[2][3] = {{1,2,3},{4,5,6}};
int x[][3] = {{1,2,3},{4,5,6}};
int x[2][3] = {1,2,3,4,5,6};
int x[2][3] = {{1,2},{4,5}};
int x[][3] = {{1,2},{4,5}};
int x[2][3] = {};
void fun(char[][4]);
```

or `char` or `double` or any data type...
size can be a number or `const int`...
...or even an `int` variable!
no curly-brace initialization
to avoid "memory leak": `delete [] a;`

`void fun(char[][4][5]);`
3D arrays, etc

Printing the time and date in a C++ program

```
#include<iostream>
using std::cout;
using std::endl;

#include<cstdlib>

int main() // print COMPILE time and date
{
  cout << __TIME__ << " at " << __DATE__ << endl;
}
```

```
#include <iostream>
using std::cout;
using std::endl;

#include <ctime>

int main() // print CURRENT date/time
{
  time_t t;
  time(&t);
  cout << ctime(&t) << endl; // e.g., Fri May 02 17:57:14 2003
}
```