# Arrays

Anna Simon

September 9, 2015

- Homework deadline extension: submit a request as your homework (I will get an email notification), then I can extend the resubmission deadlines individually.
- no office hours next Tuesday (Sep 15th).

# Syntax and initialization

An **array** is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

## Syntax

```
1 float detEnergy[8];
2 char myWord[20];
```

The indices of an N-element array run from 0 to N-1.

## Initializing an array

```
1 int nArray[5] = {3, 6, 8, 2, 9};
2 int nScore[] = {4, 8, 2}; //compiler will figure out the size of
      nScore
3 int nTime[10] = {0}; // initialize all the elements of the array to
      zero (requires the array size explicitly declared)
```

# Allocating and accessing a value

Allocating a value to an array element:

```
1 detEnergy[3] = 1.460;
2 myWord[10] = 'p';
3
4 //array elements can be accessed by a non-constant integer variable
5 int nIndex = 3;
6 detEnergy[nIndex] = 1.460;
```

The size of the array must be declared as a constant and has to be known at the time of compilation!

```
1 int anArray[5]; // Ok -- 5 is a literal constant
2
3 const int nArraySize = 5;
4 int anArray[nArraySize]; // Ok -- nArraySize is a variable constant
5
6 int nSize = 5;
7 int anArray[nSize]; // Not ok! -- nSize is not a constant!
```

# Arrays and loops

## Arrays and loops

Loops make manipulating array elements much easier:

```
1 const int nNumStudents = 5;
2 int anScores[nNumStudents] = { 84, 92, 76, 81, 56 };
3 int nTotalScore = 0;
4 for (int nStudent = 0; nStudent < nNumStudents; nStudent++)
5     nTotalScore += anScores[nStudent];
6
7 double dAverageScore = static_cast<double>(nTotalScore) /
      nNumStudents;
```

Modify the above code so that it also finds the best score in the class.

## Warning

Make sure that the loop iterates properly over the array elements! Accessing an non-existing array element results in a segmentation fault that is frequently difficult to localize in the code!

# Problem 1

## Vector algebra

Write a code that:

- asks the user for coordinates of two vectors in a 3D space,
- stores user's input in two one dimensional arrays, 3-element each,
- returns the results of: addition, subtraction, scalar product and vector product of these two vectors.

# Arrays and structs

Arrays can hold any data type, even structs:

```
1  struct Rectangle
2  {
3      int nLength;
4      int nWidth;
5  };
6
7  Rectangle asArray[5]; // declare an array of 5 sRectangle
8
9  asArray[0].nLength = 24;
10 asArray[0].nWidth = 7;
```

and can be an element of a struct:

```
1  struct Detector
2  {
3  int multiplicity;
4  float energy[4];
5  long int time[4];
6  };
7
8  Detector sClover; // declare a detector that stores four values of
       energy and time
9
10 sClover.energy[2] = 0.6;
11 sClover.time[2] = 12345;
```

# Multidimensional arrays

An element of an array can also be an array $\Rightarrow$ multidimensional array

```
1 int anArray[3][5];
```

where the first index can be thought of as a row number and the second is a column number.
Use nested braces to initialize an array:

```
1 int anArray[3][5] =
2 {
3 //[0][0],[0][1], ....
4 { 1, 2, 3, 4, 5, }, //row 0
5 //[1][0], [1][1], ...
6 { 6, 7, 8, 9, 10, }, // row 1
7 { 11, 12, 13, 14, 15 } // row 2
8 };
```

The whole multidimensional array can be initialized to zero:

```
1 int anArray[3][5] = { 0 };
```

**Multiplication table**

Write a code that prints out a multiplication table for all values between 1 and 9.

# Problem 2

## Matrix algebra

Write a code that takes two 3x3 matrices from the user. (NOTE: the `cin` command can read in multiple space-separated values). Then the code should:

- return transposed matrix of both input matrices and their traces,
- print out the result of addition and subtraction of the two input matrices,
- print out the product of two arrays to the screen.