Arrays in C++

declaration

```
int myInts[5];
```

initializing during declaration

```
int myInts[5] = { 1, 5, 2, 4, 3 };
```

in C++, arrays are not automatically initialized to 0
but we can do it ourselves: int myInts[5] = { 0 };

WARNING

we can access "elements" (or data or memory) beyond the array bounds (size)
C++ does not internally detect if we make such a mistake
so it is our responsibility to make sure we are not making a mistake

```
myInts[100] = 5; // valid!
```

no aggregate operation on arrays

```
arr1 = arr2; // illegal (well, not really but it does something we might not expect)
```

instead, use a loop to copy each individual element of the first array to the second

arrays are passed by reference only (in function calls)

parallel arrays

two or more arrays contain corresponding information
e.g.

```
int studentID[10];
int grade[10];
```

studentID[0] contains Joe's ID
grade[0] contains Joe's grade
there is a relationship between the arrays (each element location has a relationship)

2d arrays

arrays are useful to hold data in list form
sometimes data is in table form

Student	Test1	Test2
Joe	95	87
John	88	97
Sally	99	99
Susan	89	91

the headers on top and left are just there to make the information (integers) meaningful to us
we could, for example, store the grades in a single array
e.g.

95	87	88	97	99	99	89	91

but operations to manipulate the data would be more difficult
intuitively, we know that the first two elements corresponds to Joe's test grades
but how do we easily compute the index of a particular element (e.g. Joe's 2nd test grade)
it can be done, but it's not always easy to design or even particularly readable
2d arrays solve this problem by helping us look at data in tabular form
we can now think of not just rows but also columns

internally 2d arrays are stored as 1d arrays

syntax: declare a 2d array of 4 rows and 2 columns (like our table up above)

int grades[4][2]; // these are true 2d arrays unlike in java

note that in C++ arrays use row major order (rows specified first)

	0	1
0	95	87
1	88	97
2	99	99
3	89	91

and without the headers (note the row major format):

[0][0]	[0][1]
[1][0]	[1][1]
[2][0]	[2][1]
[3][0]	[3][1]

some examples

```
grades[1][0] = 88;
grades[3][1] = 91;
grades[0][1] = 87;
```

yields:

	0	1
0		87
1	88	
2		
3		91

array initialization during declaration

```
int grades[4][2] = { { 95, 87 }, { 88, 97 }, { 99, 99 }, { 89, 91 } };
```

2d array processing

rows:

```
for (int col=0; col<NUM_COLS; col++)
      arr[2][col] = 0;
```

columns:

```
for (int row=0; row<NUM_ROWS; row++)
      arr[row][1] = 0;
```

displaying/printing:

```
for (int row=0; row<NUM_ROWS; row++)
{
      for (int col=0; col<NUM_COLS; col++)
            cout << arr[row][col] << "\t";
      cout << endl;
}
```

input:

```
for (int row=0; row<NUM_ROWS; row++)
      for (int col=0; col<NUM_COLS; col++)
            cin >> arr[row][col];
```

sum by row:

```
for (int col=0; col<NUM_COLS; col++)
```

```
            sum += arr[2][col];
```

sum by column:

```
      for (int row=0; row<NUM_ROWS; row++)
            sum += arr[row][1];
```

largest element in each row:

```
      int max;
      for (int row=0; row<NUM_ROWS; row++)
      {
            max = 0;
            for (int col=0; col<NUM_COLS; col++)
                  if (arr[row][col] > max)
                        max = arr[row][col];
            cout << "max in row " << row << " is " << max;
      }
```

largest element in each column:

```
      int max;
      for (int col=0; col<NUM_COLS; col++)
      {
            max = 0;
            for (int row=0; row<NUM_ROWS; row++)
                  if (arr[row][col] > max)
                        max = arr[row][col];
            cout << "max in column " << col << " is " << max;
      }
```

main diagonal:

```
      for (int row=0; row<NUM_ROWS; row++)
            for (int col=0; col<NUM_COLS; col++)
                  if (row == col)
                        arr[row][col] = 0;
```

passing 2d arrays as function parameters

when passing regular (1d) arrays, we omit the dimension because it's a single row
but for 2d arrays, the compiler needs to know where one row ends and the next begins
so we must specify the number of columns
in general, we MUST specify the size of all dimensions except the first
e.g.

```
      f(grades);
      ...
      void f(int arr[][2])
      {
            ...
      }
```

rectangular arrays

we can extend to more than just 2d
we call 1d a list, 2d a table or matrix
what about 3d? a cube?

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

how do we define more dimensions?

```
data_type myArray[state][county][city][neighborhood][street][house][room][bookcase][shelf][book][chapter][page][row][col]
```

e.g.

```
      data_type myArray[0][1][1][3][12][4][2][0][4][8][13][7][32][7]
```

col 7 of row 32 on page 7 of chapter 13 in book 8 on shelf 4 of bookcase 0
in room 2 of house 4 on street 12 of neighborhood 3 in city 1 of county 1 in state 0

and we could go on and on and on...

character arrays (c-strings)

always terminated by the null character: '\0'
character: 'h'
c-string: “h”
it's actually 'h' and '\0' but we don't see the '\0': “h\0”
so “Hello World” is a c-string:
'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0'
same:

```
char name[10] = { 'J', 'o', 'h', 'n', '\0' };
char name[10] = "John";
char name[5] = "John";
char name[] = "John";
```

illegal:

```
char name[5];
name = "John";
```

aggregate operations are not allowed (this is an array, remember?)

c-string functions

```
#include <cstring>
```

suppose:

```
char name[5];
char name2[5];

strcpy(s1, s2); //copies s2 into s1 (length of s1 should be at least as large as s2)
strcpy(name, "John");
strcpy(name2, name);
strcpy(name, "Joe");
```

effect:

```
name = "Joe";
name2 = "John";
```

strlen(s) – returns length of s (excluding null '\0')

```
strlen(name);                  // returns 3
strlen("Juan Valdez");  // returns 11
```

strcmp(s1, s2) – compares s1 and s2 (returns < 0 if s1 < s2, > 0 if s1 > s2, 0 if s1 == s2)

```
strcmp(name, name2);    // returns < 0
strcmp("a", "b");       // returns < 0
strcmp("b", "a");       // returns > 0
strcmp("a", "a");       // returns 0
```

c-string input

```
char name[10];
cin >> name;      // someone typed John (notice there are no quotes)
```

'\0' is automatically appended
this is fine so long as there are no spaces
with spaces:

```
cin.getline(name, 10);  // reads up to 10 characters or until '\n'
```

e.g.

```
char name[10];

cin.getline(name, 10);        // I type Bill Nye
cout << "Hello " << name;     // output: Hello Bill Nye
```

more than one string:

```
char s1[5];
char s2[5];

cin.getline(s1, 5);     // I type a b
```

```
cin.getline(s2, 5);     // I type c d
cout << s1 << s2 << endl;
```

output:

a b c d

converting strings to c-strings

```
string s = "John";
char c[5];

strcpy(c, s.c_str());
```

arrays of c-strings

arrays of (arrays of characters)

char names[3][12];

names

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

strcpy(names[1], "John");

names

names[0]											
names[1]	J	o	h	n	\0						
names[2]											

arrays of strings (similar)

arrays of (arrays of characters)

string names[3];

names

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

names[1] = "John";

names

names[0]											
names[1]	J	o	h	n							
names[2]											

remember: strings do not end with a null character

HANDOUT arrays
HANDOUT command_line

HOMEWORK easy list