today:

- new topics:
  - two-dimensional arrays
  - `switch` statement
- review topics:
  - constants
  - string functions (see lecture IV.2 and textbook chapter 8)
  - cctype functions (see lecture IV.2 and textbook chapter 8)
- In class, we followed a comprehensive example using these elements. See the class web page for complete source code.

two-dimensional arrays

- we have already talked about *one-dimensional* arrays
- this is a data structure that groups together multiple elements of the same data type
- the grouping is done in "one dimension", e.g.:

  `mygrades =`

1	3	2	99	27

  which would be declared as:
  ```
  int mygrades[5];
  ```
- but sometimes you want to group elements together using two (or more) dimensions
- *two-dimensional* arrays are when you group elements together in two dimensions, like a matrix or a spreadsheet, e.g.:

  `allgrades =`

9	3	7	9	8
5	6	7	8	3
6	4	0	1	8
2	9	1	9	7

  which would be declared as:
  ```
  int allgrades[4][5];
  ```
- the entries in the two-dimensional array are referred to as *rows* and *columns*

- the *rows* run horizontally
- the *columns* run vertically
- in the example above, there are 4 rows and 5 columns
- note that in the declaration, the row dimension is declared first, as in
  ```
  int allgrades[4][5];
  ```
- you could also declare an array with 5 rows and 4 columns, like this:
  ```
  int allgrades[5][4];
  ```
- you address the elements in a two-dimensional array using two indexes, with the row index coming first, e.g.: `allgrades[0][0]` refers to the entry in the upper left corner of the array; in this case, it has the value 9

switch statement

- a `switch` statement is useful if you are making a choice between a number of options all concerning the value of a single, simple-typed variable
- it can replace multiple `if-else-if-else...` statements and tends to look neater
- but it can only replace multiple `if-else-if-else...` statements if the variable being compared in each statement is the same variable and it is of a simple data type (e.g., `int`, `char`, `bool`, etc.)

- for example:

```
int x;
string heading;
.
.
.
if ( x == 0 ) {
heading = "north";
}
else if ( x == 1 ) {
heading = "west";
}
else if ( x == 2 ) {
heading = "south";
}
else if ( x == 3 ) {
heading = "east";
}
```


which can be replaced with a `switch` statement, like this:

```
int x;
string heading;
.
.
switch ( x ) {
  case 0:
    heading = "north";
    break;
  case 1:
    heading = "west";
    break;
  case 2:
    heading = "south";
    break;
  case 3:
    heading = "east";
    break;
} // end of switch
```


- note the new keywords:
  - `switch` which begins the statement and indicates the name of the variable you want to compare
  - `case` which indicates the value that you want to compare the variable to
  - `break` which ends the clause that gets executed for each matching "case", i.e., when the value of the variable matches that specified in the enclosing `case`
- note that if you don't use a `break` command, then the program control will keep going at the end of one case and go into the code for the next case
  (there are times when you want this behavior, but most of the time you don't)

- note that the default case, when the value of the variable does not match any of those in the specified cases, is called `default`
- you replace the `case __` with `default`, like this:

```
switch( x ) {
  case 0:
    heading = "north";
    break;
  case 2:
    heading = "south";
    break;
  default:
    heading = "ERROR";
    break;
} // end of switch
```

constants

- we can define a *constant* using the keyword `const`, e.g.:

```
const int LENGTH = 7;
```

- by convention we give a constant a name that is all upper case
- we can then use `LENGTH` wherever we need the number 7, e.g.:

```
int grades[LENGTH];

for ( counter=0; counter<LENGTH; counter++ ) {
     sumOfGrades += grades[counter];
   }
```

- like a variable, a constant has a:
  - data type, e.g., int
  - name or *identifier*, e.g., LENGTH
  - value, e.g., 7
- UNLIKE a variable, the value of a constant CANNOT change whlie a program runs