today: math operations and function arguments

- random numbers
- math operators
- data type conversion
- function arguments

random numbers

- computers can generate "random" numbers, which is like picking a number by rolling dice
- there are two steps necessary for generating random numbers:
  1. *seeding* the random number generator
  2. picking the random number
- the **seed** is used to create a sequence of "pseudo random numbers"
  you can always get the same sequence again if you use the same seed!
- the random numbers generated are integers (of type `int`)
- `void srand( long seed )`
  is used to seed (initialize) the random number sequence
- `int rand()`
  returns a random integer between 0 and `RAND_MAX`, where `RAND_MAX` is a constant defined by the C language
- you can *scale* the result returned from `rand()` to get a number in the range you want (e.g., $0..10$)

random numbers: example

```
// initialize random seed
srand( time( NULL ));

// find random initial location
x = rand() % 10;
y = rand() % 10;
display();
```


math operators

- the mathematical operators in C are:

$+$	unary plus
$-$	unary minus
$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
$\%$	modulo

- there are also *math functions* defined in a standard C library called `math.h`
- these include, for example:
  - `double sqrt( double x )`
  - `double pow( double x, double y )`
  - `double sin( double x )`
- these take *arguments* and return values, e.g.:
  `double f = sqrt( 4.0 );`

data type conversion

- remember we talked earlier about data types, like `int` and `double` and how much they can be used to store integer (whole) or real numbers
- in C and C++, when you are doing math, you need to make sure that you are working with numbers that are all the same *data type*
- internally, C converts everything to `double` and then does the math, and then converts the result back to `int` if necessary
- but you have to explicitly tell the computer to make the conversion so that things are converted the way you want
  also, some compilers check to make sure that arguments are the right data types
- in order to convert explicitly, you put the data type you want to convert to in parenthesis and put that before the value you want to convert, e.g.,:
  `double f = sqrt( (double)4 );`

math example

- modify the roomba program so that the robot starts in a random location
- pretend that the roomba's charging station is located at position $(0, 0)$;
  extend the program so that it calculates the distance from the roomba's current location to its charging station;
  some definitions will be helpful:
  - *Euclidean distance*: $d = sqrt(x^2 + y^2)$
  - *Manhattan distance*: $d = x + y$
- modify the program so that the initial location and the location of the charging station are computed randomly;
  how would you need to change the formulas above?
  - *Euclidean distance*: $d = sqrt((x_1 - x_0)^2 + (y_1 - y_0)^2)$
  - *Manhattan distance*: $d = abs(x_1 - x_0) + abs(y_1 - y_0)$
- how would you translate the formulas from math language to the C or C++ programming language?

functions and function arguments

- we have talked about *functions* and used them already in the first homework (e.g., `display()`)
- we also talked briefly about the `return` statement
- the power of functions is that they can "take arguments" and "return values"
- this means you can use the same function code to perform the same tasks on different values, e.g.,:

```
double f;
f = sqrt( 4.0 );
cout << "f = " << f << endl;
f = sqrt( 15.7 );
cout << "f = " << f << endl;
```


function arguments

- you can write your own functions (like we did with `display()`)
- you can also write your own functions so that they take arguments
- example:

```
int move( char c ) {
  if ( c == 'F' ) {
    cout << "moving forward...\n";
    y = y + 1;
  }
  return 0;
} // end of move()
```

- in this example, the function `move()` takes one argument `c`
- note that the function definition includes the data type of the argument in the header
- you can have multiple arguments, separated by commas, e.g.:
  `int moveSteps( char c, int num_steps ) { ... }`

function return values

- we have used the statement `return 0;` to end all our functions (before the last curly bracket }})
- however, you can tell the function to return a value
- example:

```
int move( char c ) {
  if ( c == 'F' ) {
    cout << "moving forward...\n";
    y = y + 1;
  }
  return y;
} // end of move()
```

- note that the data type of the value returned must be the same as the data type of the function
- you can define functions of any type, e.g.: `char direction() { ... }`