Next time: TCS 4.
Exam is coming up. Will cover TCS 1-4 and LCR 1-4.
We will be scheduling a review session (probably in labs).
1/3 short answer, 1/3 code reading, 1/3 code writing.

Program Design

Example: Program to draw regular polygons (i.e., all the sides are the same length and angles are equal).

Decide on exact specifications.

Number of sides: min 3, max is less clear.
Given the control limitations of these robots (which we don't know exactly) we can't draw very short lines accurately, and we can't turn through very small angles accurately, so this limits the max number of sides we can hope to do.

Standard engineering problem: you have idealized mathematical formulas, but you also have the real physical objects, which have additional limitations. You have to make sure that what you are trying to do is realistic.

Based on our experience, we might pick a max number, say 8.

What are the allowable lengths? We surely want to agree on a minimum length, but we have to figure out what's reasonable.

It's important for both the implementer and the user to agree in advance on the specifications and workout any uncertainties. This doesn't mean it can never change. But if either sees a reason to change it, you need to go back and agree on

the specs. Bottom line: don't make assumptions.

Top-down:
Obviously to draw a polygon, we need to be able to draw a straight line of a given specified length. And we need to be able to turn through a specified angle. But the Myro software does not provide these operations. We have reduced drawPolygon to two simpler problems, but we have to implement these too. Fortunately we can see that it's easy to implement these in terms of Myro operations.

When you write the for-loop to draw the sides, you have to be careful not to make an off-by-one error. One way to avoid this is to check small special cases (e.g., 3, 4). Do the same to make sure you have the turn-angle computation correct.

Random Numbers

Often we want a source of random numbers (unpredictable numbers) in our programs. They have many uses:
(1) Games.
(2) Simulations. E.g. simulating traffic flow around a city.
(3) There are some algorithms (e.g., stochastic optimization algorithms) the require randomness.
(4) etc.

The int function rand() (in <cstdlib> = "C standard library") gives you a random number in the range 0 to some big number.

What we usually use in computer science are pseudo-random numbers, which means they are not really random, but they look random. rand() is a pseudo-random number generator, which computes, by an algorithm, numbers that look

random.

This calculates the same sequence of numbers every time you run the program, unless you seed the random number generator at the beginning of the program. This starts it at a different place than the beginning of its sequence.

srand(<seed>); // sets the seed
srand(0); // starts at the beginning again

So the trick is to seed the random number generator with an unpredictable number.

```
int seed;
cout << "Enter any number: \n";
cin >> seed;
srand (seed);
```

To get real unpredictability:

```
#include<ctime>
….
srand(time(NULL)); // seeds the RNG with the time in msec.
```

Usually we want integers in some range, say 1 to 6 for dice.
Sometimes we want random doubles, e.g. a number between 0.0 and 1.0.

Notice that when you compute N % M you get a number in the range 0 to M-1. This is because N%M is the remainder after dividing M by N.

How about a random double in the range 0.0 to 1.0?
Here we can use a built-in constant called RAND_MAX = the maximum possible random number.

rand() / (double) RAND_MAX