Functions
Introduction to Functions

• C is a simple language

• Its utility is extended through the use of re-usable functions

• Some functions are found in standard libraries like stdio.h and math.h

• Users can write functions too – for many good reasons!
“Intrinsic” Functions:

While the text makes reference to intrinsic functions, it’s more accurate to call them the C Standard Library Functions.

These are not part of the C language, but reside in a library of useful functions that evolved along with C and is now also standardized across compiler distributions and hardware platforms.

To use these functions it was necessary to first include a header file so the compiler would recognize their input/output interfaces. The actual code is pre-compiled and is linked to your source code to make a working program.
C Standard Library Functions

“Intrinsic” Functions:
While the text makes reference to intrinsic functions, it’s more accurate to call them the C Standard Library Functions.

These are part of the C language library. They evolved from the earlier machine-dependent library and now also compile and run on hardware platforms that are not the original hardware.

To use these, you first have to include the necessary header files in your program. You would reference input/output actual code and is linked to your source code to make a working program.

Common C Standard Libraries we will use:
- stdio.h
- math.h

Best to get in the habit of including these in every program you write!

You don’t even know WHERE these files are on galileo in order to use them.
functions from the C math library

note that C's math functions take and return parameters that are of type double:

```
double sqrt(double x);  //prototype for sqrt function
```

You'll find a line like this in the math.h header file.

The compiler reads this from <math.h>, then when it encounters a call to sqrt() in your code, it can check that you are calling it correctly:

- giving the right number of parameters,
- using the output value properly
- etc...

For example:

```
int i = sqrt(10.);
float q = sqrt(10., 2.);
```

This will generate a warning.

This will generate an error.
User-Defined Functions

• Why would one want to write their own functions?

  – Avoid duplicating the same code many times within a program: streamlined simplicity

  – Re-usable functions are easier to maintain and modify

  – Functions are portable – can be used in other programs

  – Simplicity: code something intricate once and call it via a simple single line rather than multi-line complications
User-Defined Functions: Example Use Case

Why Write Functions?

```c
#include <stdio.h>
#include <math.h>

int main () {
    double x0 = 0.0;
    double y0 = 0.0;

    double x1 = 1.0;
    double y1 = 2.0;

    double x2 = 4.0;
    double y2 = 1.0;

    double x3 = 3.0;
    double y3 = 0.0;

    double d01 = sqrt( (x1-x0)*(x1-x0) + (y1-y0)*(y1-y0) );
    printf ("d01 is %f\n",d01);

    double d12 = sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) );
    printf ("d12 is %f\n",d12);

    double d23 = sqrt( (x3-x2)*(x3-x2) + (y3-y2)*(y3-y2) );
    printf ("d23 is %f\n",d23);

    return(0);
}
```

Consider this program, which calculates the total distance for a trip through four points.

Notice that the program repeatedly uses similar statements to calculate the lengths of the segments of the trip.

If we ever needed to change the program (say, to print travel times) we'd need to remember to modify each of these statements.

No functions – repetitive!
Three main parts: Prototype, Definition, and Calls

User-Defined Functions: Use Case Example

Designing a Function:

```c
#include <stdio.h>
#include <math.h>

double distance(double xstart, double ystart, 
                double xend, double yend);

int main () {
    // (coordinates omitted for brevity)...
    double d01 = distance(x0,y0,x1,y1);
    printf ("d01 is %f\n",d01);

    double d12 = distance(x1,y1,x2,y2);
    printf ("d12 is %f\n",d12);

    double d23 = distance(x2,y2,x3,y3);
    printf ("d23 is %f\n",d23);

    return(0);
}
```

To make things better, we can create a new function, called “distance”, to calculate the distance.

Function prototype

Using (“calling”) the function

We could easily modify the distance function to return, say, travel time (adjusted for a headwind from a given direction!). We'd only need to make the change in one place: the function definition.
Prototype, Arguments and Return:

```c
#include <stdio.h>
#include <math.h>

double distance(double xstart, double ystart,
                  double xend, double yend);

int main () {
    // (coordinates omitted for brevity)...
    double d01 = distance(x0,y0,x1,y1);
    printf ("d01 is %f\n",d01);
    
    double d12 = distance(x1,y1,x2,y2);
    printf ("d12 is %f\n",d12);
    
    double d23 = distance(x2,y2,x3,y3);
    printf ("d23 is %f\n",d23);
    
    return(0);
}
```

The prototype defines the syntax for the function. (What arguments it takes, and what type of data it returns.)

The names of the arguments in prototype, function call and function definition don’t need to match, but the types do.

Our function takes four “doubles” as arguments, and returns a double.

```c
double distance (double xinit, double yinit,
                 double xfinal, double yfinal) {
    double d;
    d = sqrt( (xfinal-xinit)*(xfinal-xinit) +
              (yfinal-yinit)*(yfinal-yinit) );
    return(d);
}
```
User-Defined Functions: Reusable?

```
#include <stdio.h>
#include <math.h>

double distance(double xstart, double ystart, double xend, double yend);

int main () {
    // (coordinates omitted for brevity)...
    double d01 = distance(x0,y0,x1,y1);
    printf("d01 is \%f\n",d01);
    double d12 = distance(x1,y1,x2,y2);
    printf("d12 is \%f\n",d12);
    double d23 = distance(x2,y2,x3,y3);
    printf("d23 is \%f\n",d23);
    return(0);
}

double distance (double xinit, double yinit, double xfinal, double yfinal ) {
    double d;
    d = sqrt( (xfinal-xinit)*(xfinal-xinit) + (yfinal-yinit)*(yfinal-yinit) );
    return(d);
}
```

Some day, the prototype for your function could be moved into an external header file, to be included as needed... and the function itself could be added to your own library of functions, for later use. We'll see how to do this later.

```
mylib

distance()

etc...
```