# RoboRugby 2014

## Programming Tutorial

Brian Mulkeen

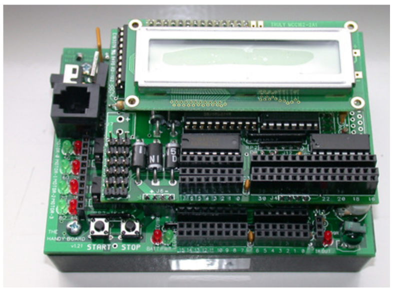
UCD School of Electrical,
Electronic and Communications
Engineering

Scoil na hInnealtóireachta
Leictrí, Leictreonaí agus
Cumarsáide UCD

## Computer

Handyboard: small computer, "brain" of robot

lots of sockets to connect motors, sensors, etc.



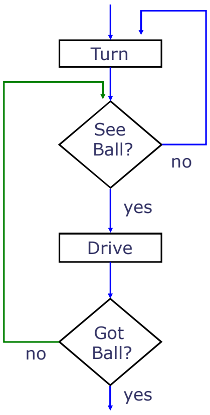
## Programming

- Computer program is sequence of instructions
  - instructions are basic steps that computer can take
  - you combine them to make it do what you want
- Algorithm
  - definition of process required
  - sequence of steps to solve problem
- Writing a computer program
  - first decide what you want it to do
    - define the algorithm - describe in English, diagram, etc.
  - then write instructions for the computer
    - use language that the computer understands

## Flow Chart

- One way of planning your program
  - boxes for actions
  - diamonds for decisions
- Graphical description of algorithm…

Not required,
but can be useful!

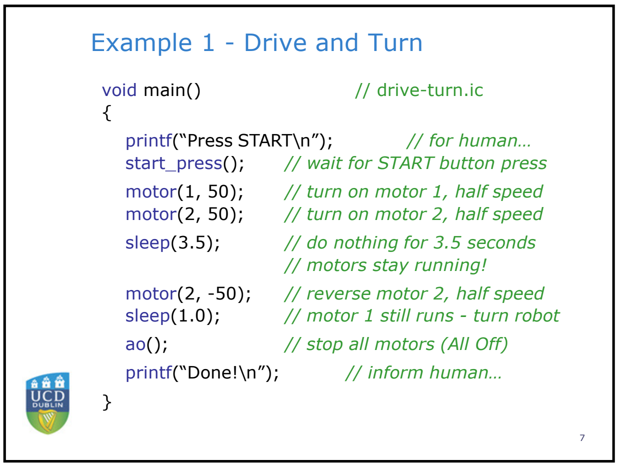


## Interactive C

- Programming language for Handyboard system
  - based on C - a standard programming language
    - used in Computer Science for Engineers module
  - some simplifications for small computer
  - many extra features for Handyboard
- Compiler
  - translates program into instructions that a computer can understand
  - fussy about syntax, spelling, punctuation…
- Interactive C system
  - allows instructions to be executed immediately!

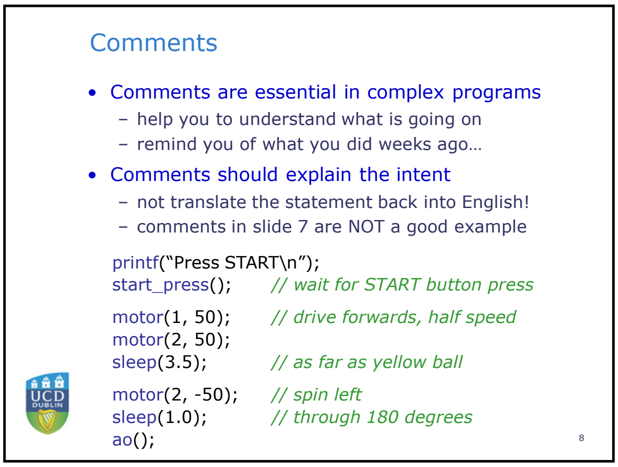
## Basic Rules - Syntax

- Instructions must end in ;
  - called *statements*
- { } used to group statements as a unit
- Case matters: Beep() is not same as beep()
- Comments added for human use
  - explain or remind what program does
  - anything after // is a comment – to end of line
  - anything between /* and */ is a comment
    - can be many lines, or small part of line



## Example 1 - Drive and Turn

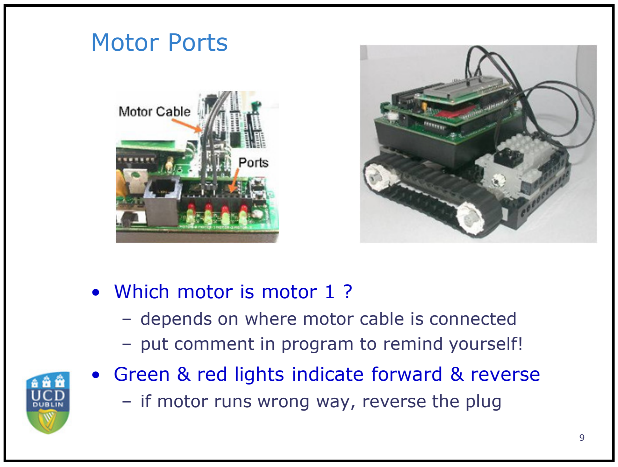
```
void main()                        // drive-turn.ic
{
    printf("Press START\n");           // for human…
    start_press();      // wait for START button press
    motor(1, 50);       // turn on motor 1, half speed
    motor(2, 50);       // turn on motor 2, half speed
    sleep(3.5);         // do nothing for 3.5 seconds
                        // motors stay running!
    motor(2, -50);      // reverse motor 2, half speed
    sleep(1.0);         // motor 1 still runs - turn robot
    ao();               // stop all motors (All Off)
    printf("Done!\n");          // inform human…
}
```



## Comments

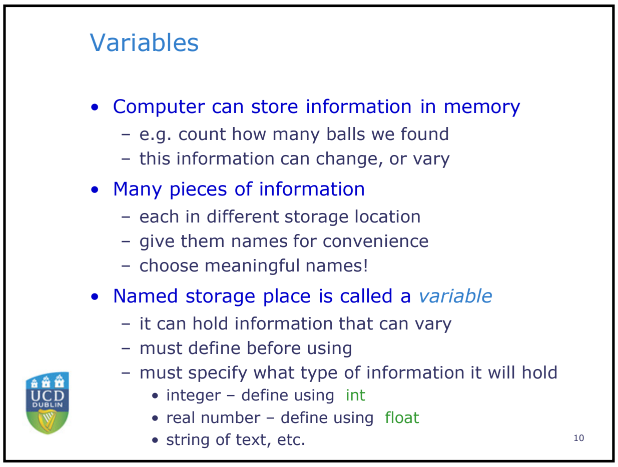
- Comments are essential in complex programs
  - help you to understand what is going on
  - remind you of what you did weeks ago…
- Comments should explain the intent
  - not translate the statement back into English!
  - comments in slide 7 are NOT a good example

```
printf("Press START\n");
start_press();      // wait for START button press
motor(1, 50);       // drive forwards, half speed
motor(2, 50);
sleep(3.5);         // as far as yellow ball
motor(2, -50);      // spin left
sleep(1.0);         // through 180 degrees
ao();
```



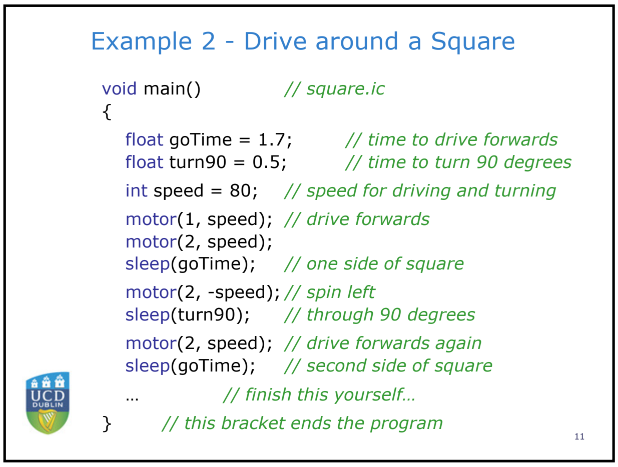
## Motor Ports

- Which motor is motor 1 ?
  - depends on where motor cable is connected
  - put comment in program to remind yourself!
- Green & red lights indicate forward & reverse
  - if motor runs wrong way, reverse the plug



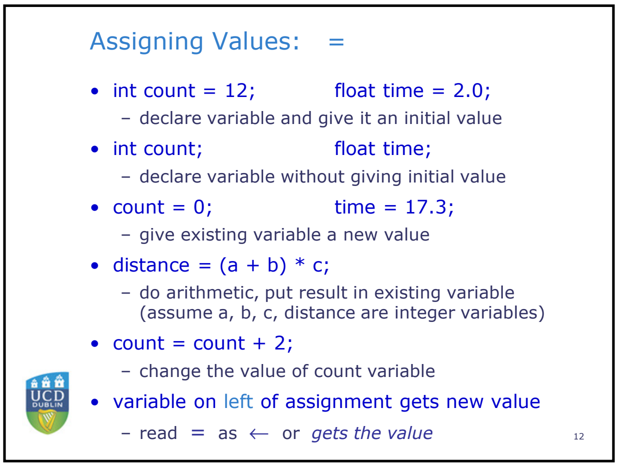
## Variables

- Computer can store information in memory
  - e.g. count how many balls we found
  - this information can change, or vary
- Many pieces of information
  - each in different storage location
  - give them names for convenience
  - choose meaningful names!
- Named storage place is called a *variable*
  - it can hold information that can vary
  - must define before using
  - must specify what type of information it will hold
    - integer – define using int
    - real number – define using float
    - string of text, etc.



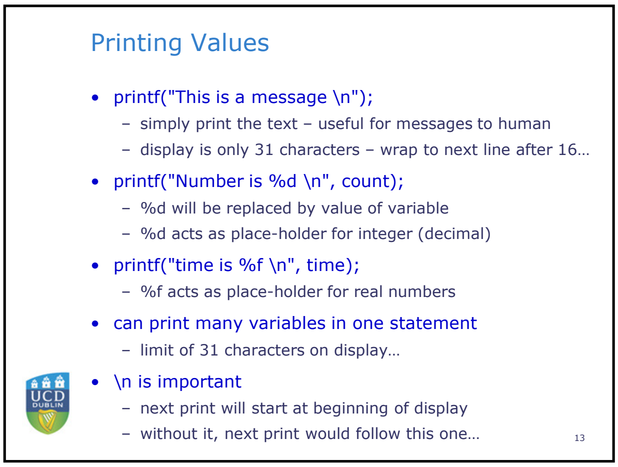
## Example 2 - Drive around a Square

```
void main()             // square.ic
{
    float goTime = 1.7;         // time to drive forwards
    float turn90 = 0.5;         // time to turn 90 degrees
    int speed = 80;     // speed for driving and turning
    motor(1, speed);  // drive forwards
    motor(2, speed);
    sleep(goTime);      // one side of square
    motor(2, -speed); // spin left
    sleep(turn90);      // through 90 degrees
    motor(2, speed);  // drive forwards again
    sleep(goTime);      // second side of square
    …              // finish this yourself…
}        // this bracket ends the program
```



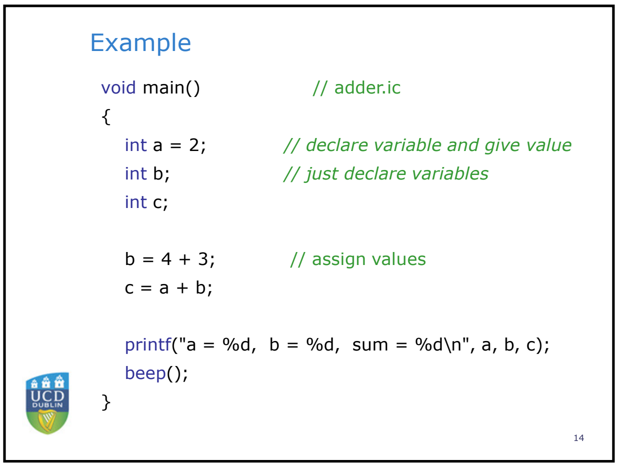
## Assigning Values:   =

- int count = 12;          float time = 2.0;
  - declare variable and give it an initial value
- int count;          float time;
  - declare variable without giving initial value
- count = 0;          time = 17.3;
  - give existing variable a new value
- distance = (a + b) * c;
  - do arithmetic, put result in existing variable (assume a, b, c, distance are integer variables)
- count = count + 2;
  - change the value of count variable
- variable on left of assignment gets new value
  - read = as ← or *gets the value*



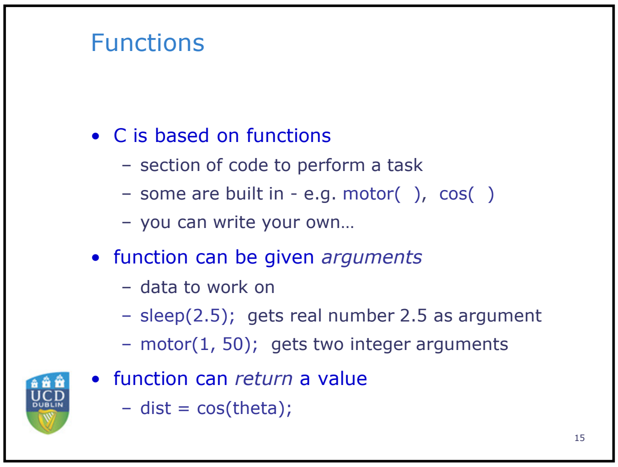
## Printing Values

- printf("This is a message \n");
  - simply print the text – useful for messages to human
  - display is only 31 characters – wrap to next line after 16…
- printf("Number is %d \n", count);
  - %d will be replaced by value of variable
  - %d acts as place-holder for integer (decimal)
- printf("time is %f \n", time);
  - %f acts as place-holder for real numbers
- can print many variables in one statement
  - limit of 31 characters on display…
- \n is important
  - next print will start at beginning of display
  - without it, next print would follow this one…



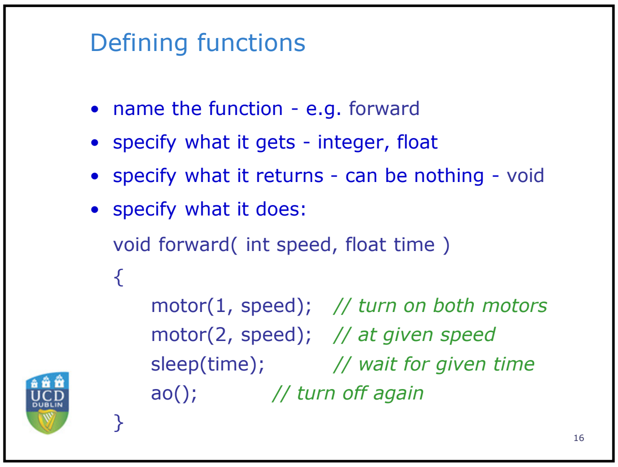
## Example

```
void main()                 // adder.ic
{
    int a = 2;          // declare variable and give value
    int b;              // just declare variables
    int c;

    b = 4 + 3;          // assign values
    c = a + b;

    printf("a = %d,  b = %d,  sum = %d\n", a, b, c);
    beep();
}
```



## Functions

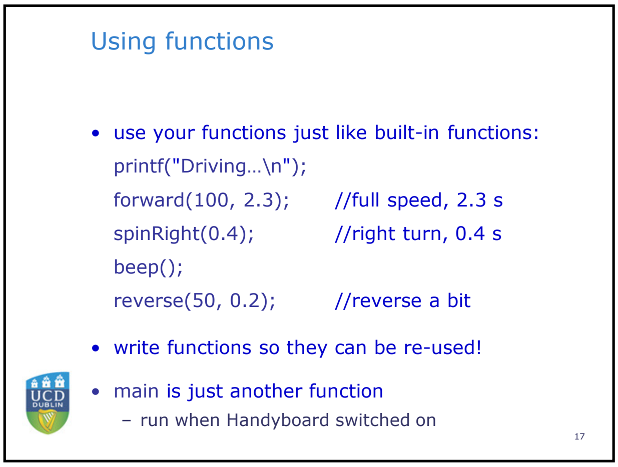
- C is based on functions
  - section of code to perform a task
  - some are built in - e.g. motor( ), cos( )
  - you can write your own…
- function can be given *arguments*
  - data to work on
  - sleep(2.5); gets real number 2.5 as argument
  - motor(1, 50); gets two integer arguments
- function can *return* a value
  - dist = cos(theta);



## Defining functions

- name the function - e.g. forward
- specify what it gets - integer, float
- specify what it returns - can be nothing - void
- specify what it does:

```
void forward( int speed, float time )
{
    motor(1, speed);   // turn on both motors
    motor(2, speed);   // at given speed
    sleep(time);       // wait for given time
    ao();          // turn off again
}
```

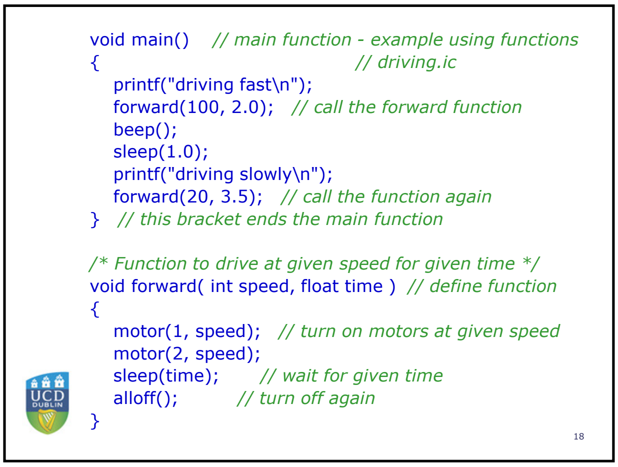


## Using functions

- use your functions just like built-in functions:

```
printf("Driving…\n");
forward(100, 2.3);      //full speed, 2.3 s
spinRight(0.4);         //right turn, 0.4 s
beep();
reverse(50, 0.2);       //reverse a bit
```

- write functions so they can be re-used!
- main is just another function
  - run when Handyboard switched on



```
void main()    // main function - example using functions
{                                   // driving.ic
    printf("driving fast\n");
    forward(100, 2.0);   // call the forward function
    beep();
    sleep(1.0);
    printf("driving slowly\n");
    forward(20, 3.5);   // call the function again
}   // this bracket ends the main function

/* Function to drive at given speed for given time */
void forward( int speed, float time )  // define function
{
    motor(1, speed);   // turn on motors at given speed
    motor(2, speed);
    sleep(time);       // wait for given time
    alloff();          // turn off again
}
```