

# RoboRugby 2010

Robotics Design Project  
EEEN 10020

## Programming Tutorial 2



UCD School of Electrical,  
Electronic and Mechanical  
Engineering

Scoil na hInnealtóireacta  
Leictirí, Leictreonaí agus  
Meicniúla UCD

## Programming

- **Computer program is sequence of instructions**
  - basic steps which 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 algorithm - describe in English, diagram, etc.
  - then write instructions for computer
  - use language which computer understands



2

## Declaring Variables

- **Must declare variables at start of each function**
  - before any other instructions
- **int count;**
  - create an integer variable called `count`
  - stored as 16 binary digits (bits)
  - can hold values from -32,768 to +32,767
- **long bignum;**
  - create a long (32-bit) integer variable, `bignum`
  - range -2,147,483,648 to +2,147,483,647
- **float time;**
  - create a real variable (floating point) called `time`
  - precision about 7 decimal digits
  - magnitude from about  $10^{-38}$  to  $10^{+38}$



3

## Assigning Values: =

- **int count = 12; float time = 2.0;**
  - declare variable and give it an initial value
- **count = 0; bignum = 2050L;**
  - give existing variable a new value
- **count = (a + b) \* c;**
  - do arithmetic, put result in existing variable (assume a, b, c are also integer variables)
- **count = count + 2;**
  - change the value of count
- **variable on left of assignment gets new value**
  - read = as ← or *gets the value*



4

## 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



5

## Functions

- **C is based on functions**
  - section of program to perform a specific task
  - some are built in - e.g. `motor( )`, `cos( )`
  - recognise by brackets after name of function
  - you can write your own...
- **Defining functions:**
  - name the function - e.g. `forward`
  - specify what *arguments* it gets - integer, float
  - specify what it returns - can be nothing - `void`
  - specify what it does - { statements... }
  - *return* statement ends function, sends back value



6

```

void main() // main function – 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
  ao(); // turn off again
} // end of forward function

```



7

## Useful Functions

- **int start\_button(void)**  
– returns 1 if START button pressed, else 0
- **int stop\_button(void)**  
– returns 1 if STOP button pressed, else 0
- **int digital(int port\_number)**  
– returns 1 if switch connected to port is closed, else 0
- **int analog(int port\_number)**  
– returns integer 0-255 depending on voltage at port
- **float seconds(void)**  
– returns time since program started
- **void tone(float freq, float time)**  
– play sound of given frequency for given time



8

## Example Function - detect collisions

- Switches right and left, on ports 9 and 10  
– values checked using `digital()` function - 0 or 1  
– combined using arithmetic to get single value  
– that value is **returned** as result of function  
– no arguments needed - **void**

```

/* Function to detect collisions using 2 switches
returns 1 if hit left, 2 if hit right, 3 both, 0 none */

```

```

int checkSwitch( void ) // define function
{
  return digital(10) + 2*digital(9); // return value
} // end of checkSwitch function

```



9

## Flow Control - Decisions - if...

- ```

if ( count < 10 ) beep();

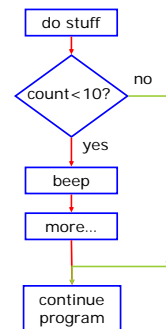
```
- “count < 10” has *boolean* value - true or false
  - if true, `beep()` will be done
  - need brackets around the decision part

```

if ( count < 10 )
{
  beep();
  count = count + 2;
  printf("message...\n");
}

```

- this form allows more to be done



10

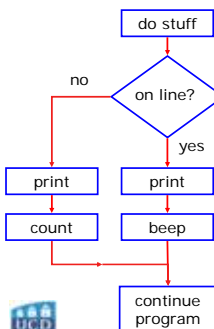
## Flow Control - More Decisions

- other comparisons:
  - `<=` less than or equal to
  - `>` greater than
  - `>=` greater than or equal to
  - `==` equal to (note double =)
  - `!=` not equal to
- `if ( start_button() ) beep();`
  - where is the boolean value?
  - `start_button()` function returns integer
  - integer treated as false if 0, otherwise true
  - same as `if ( start_button() == 1 ) beep();`
- `if ( !start_button() ) beep();`
  - opposite effect - read ! as “not”



11

## Flow Control - alternatives



12

```

if ( analog(2) < 20 ) // test sensor
{
  // do all this if value < 20
  printf("on white line\n");
  beep();
}
else // otherwise (optional)
{
  // do this...
  printf("off the line\n");
  lost = lost + 1;
}

```

## More Complex Decisions

- boolean operations:

- && AND
- || OR
- ! NOT

```
if ((analog(2) < 20) || (stop_button() ))
{ ... do something about it ...}
else if ((lost < 5) && (seconds() < timelimit))
{ ... do something about that ...}
else
{ ... do something else ...}
```

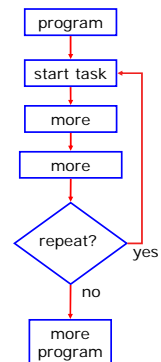


13

## Flow Control - Loops

- Need to repeat certain tasks
  - hunt for balls, over and over
  - collide, recover, drive on, etc.
- key word is **while**
  - two forms of loop
  - first form:

```
do
{
    all these things
}
while (something is true);
```



14

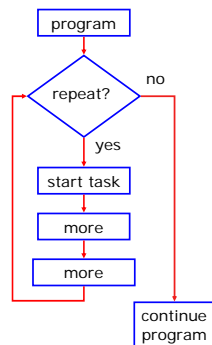
## Flow Control - Loops

- key word is **while**
  - two forms of loop
  - second form:

```
while (something is true)
{
    do all these things
}
```

- loop can be empty
 

```
while (something is true)
{ } // do nothing
```



15

## Flow Control - Do-While Loop

```
int count = 0;
do
{
    printf("twice %d is %d\n", count, 2*count);
    sleep(1.5);
    count = count + 1;
}
while (count <= 10);
```

- repeats all instructions as long as  $count \leq 10$ 
  - only checked at **end** of loop each time
  - action always happens at least once
- count must change inside loop!



16

## Examples using While

```
while(!stop_button()) // loop until STOP pressed
{ // instructions here get done many times
}
```

Can do nothing - effect is to wait:

```
while( !start_button() ) { } // wait for START
or
while( !start_button() ); // alternative empty loop
```

This is how the start\_press() function works:

```
while( !start_button() ); // wait for START
while( start_button() ); // wait for release
beep(); // beep & waste a little time
```



17

```
/* Example using one switch to detect collision.
   Assume switch on port 10, motors on 1 and 2 */
void main()
{
    start_press(); //wait for START
    motor(1,100);
    motor(2,100); // drive forwards, full speed

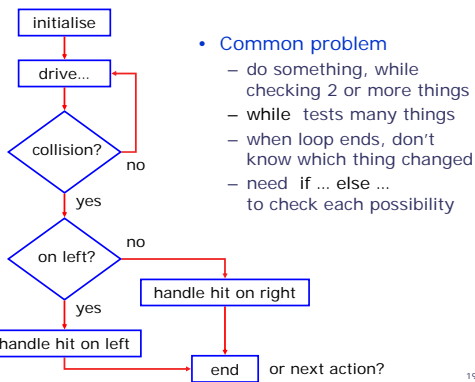
    do { } // do nothing (keep going)
    while( !digital(10)); // until hit something

    ao(); // stop - we have hit something
    beep(); // double beep
    sleep(0.3);
    beep();
    printf("oops!\n"); // print message
} // end of main
```



18

### Flow Chart for Handling Collisions



- **Common problem**
  - do something, while checking 2 or more things
  - while tests many things
  - when loop ends, don't know which thing changed
  - need if ... else ... to check each possibility



19

### Example - using function

```

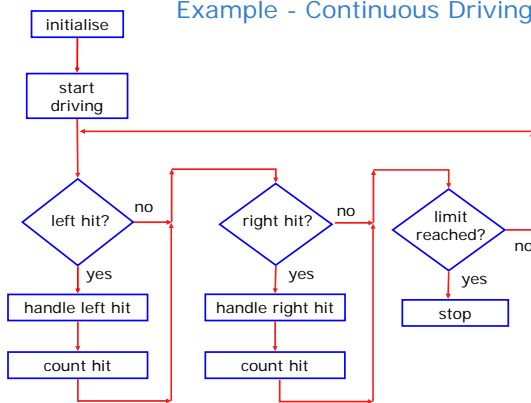
void main()
{
  motor(1,100);
  motor(2,100); // drive forwards
  while( checkSwitch()==0 ) {} // do nothing until hit
  ao(); // then stop motors
  printf("Collision\n");
  // reverse and turn away from obstacle
  if (checkSwitch()==1) // hit on left
  { - - reverse and turn right - - }
  else // must be hit on right or both
  { - - reverse and turn left - - }
} // end of main
  
```

- **Function hides detail from main program**
  - returns one number to indicate state of switches



20

### Example - Continuous Driving



### Example - Continuous Driving

```

int count = 0; // create counter, start at 0
drive(100); // start driving
do // repeat all that follows
{
  if (hit on left)
  {
    deal with it - stop, reverse, turn, drive
    count = count + 1;
  }
  else if (hit on right) { same idea }
}
while (count < 17); // until hit enough times
ao(); // then stop (or whatever...)
  
```



22