

RoboRugby 2010

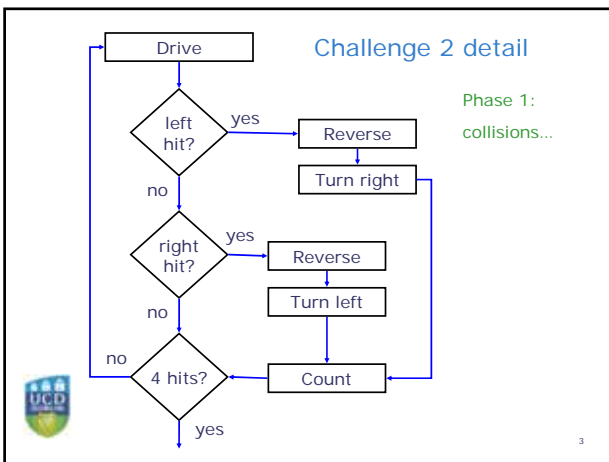
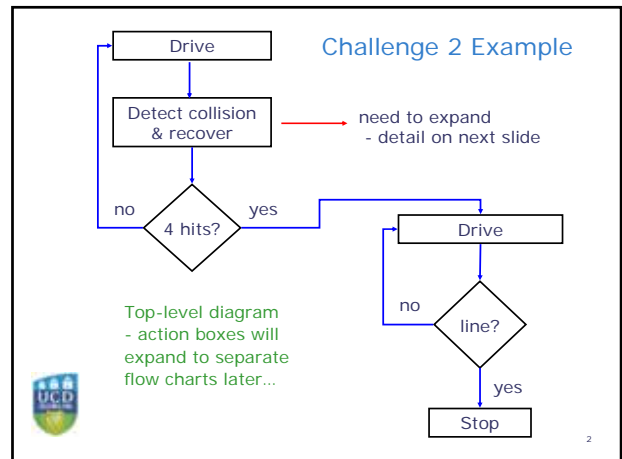
Robotics Design Project
EEEN 10020

Programming Tutorial 3



UCD School of Electrical,
Electronic and Mechanical
Engineering

Scoil na hInnealtóireacta
Leictir, Leictreonaí agus
Meicniúla UCD



Review - Flow Control - if

```
if (digital(7) && (!stop_button())) // test something
{
    // do all this if true
    printf("collision\n");
    beep();
}
else
{
    // do this if false
    printf("no collision\n");
}
```

Review - Flow Control - while ...

```
while( !digital(9) ) // test something
{
    // do this repeatedly while true
}
//only gets here if test is false
printf("collision\n"); // print message

// alternative if nothing to do...
while( !digital(9) ){ // wait until false
printf("collision\n"); // print message
```

Review - Flow Control - do ... while ...

```
do
{
    // do all this repeatedly
}
while( !digital(9) ) // while test is true
//only gets here if test is false
printf("collision\n"); // print message
```

- In competition program, loops should check time
 - no point in turning forever looking for beacon
- Suppose robot driving to find line:
 - if no line found in 20s, what has happened?

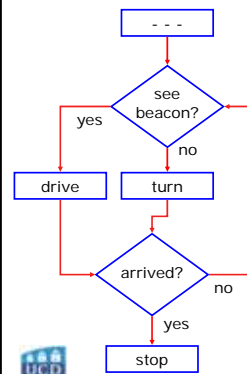
if () or while () ?

- Both make decisions, but different applications
- use `if ()` to make choices
 - do A or B or C, depending on conditions
 - example: if hit wall, reverse away
 - example: if see beacon, drive, otherwise turn
 - these things only happen once...
- use `while ()` to repeat things - loops
 - do TASK **until** something happens
 - TASK can have many steps...
 - TASK can include choices: `if ()` ...
 - TASK can include another loop: `while ()` ...
 - or TASK can be nothing - just wait...



7

if () or while () ?



8

- Flow chart
- Both shown as diamonds
 - since represent decisions
- Typical `if ()`
 - two or more branches
 - branches go forwards
 - paths converge again later
- Typical `while ()`
 - exactly two branches
 - one branch goes back to form loop

Programs with Several Stages

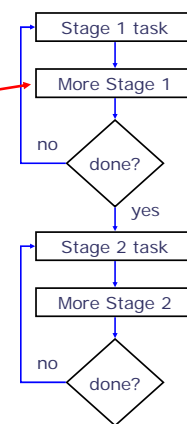
- Important in competition
 - strategy will have a few different stages
 - want to do one task for a while, then move on
- While loop repeats one task
 - conditions in `while()` to decide when to move on
 - include all conditions - check later: which one?
 - example: move on to next task if
 - hit obstacle
 - or see beacon
 - or 10 seconds elapsed
- Finish loop for one task before starting next
 - do NOT try to handle stage 2 inside loop for stage 1



9

Flow Chart

- Top-level diagram
- This will expand
 - may include other loops
 - may include a few tasks
- Suppose driving to beacon to score collected balls
 - when does this stage end?
 - beacon signal lost?
 - hit obstacle?
 - detect scoring zone?



10

Review - Functions

- Can write functions to perform specific tasks
 - useful for common tasks - use many times
 - useful to hide detail, simplify main program
- function can be given data to use: *arguments*
 - `motor(1, 50);` gets two integer arguments
- function can *return* a value
 - `light = analog(3);`
- write functions so they can be re-used
- put comments at start to explain everything
- `main()` is just another function
 - run when Handyboard switched on



11

Example Function

```

/* Function to check line sensor on port 2.
Returns 1 (true) if on line, else 0 (false) */

```

```

int onLine( void )
{
    if (analog(2) < 20) return 1; // true
    else return 0; // false
}

```

```

// Example using function

```

```

if ( onLine() ) drive(100);
else turnLeft(50);

```



12

Example Function

```
/* Function to check line sensors on ports 2 (left) & 3.
Returns 3 if on line, 2 off on right, 1 off on left,
0 if lost. Hides detail of sensors and thresholds. */
int checkLine( void )
{
    if ((analog(2)<20) && (analog(3)<20)) return 3;
    else if (analog(2)<20) return 2; // off on right
    else if (analog(3)<20) return 1; // off on left
    else return 0; // no sign of line
}
```

// note no need to check both sensors every time...



13

Local Variables

- Variables created within a function:
 - local to that function
 - cannot be accessed by other functions
 - other functions can have variables with same name - no conflict

```
void main(void)          int drive( int speed)
{
    int ann = 20;        int ann;
    int bob;             motor(1, speed);
    bob = drive(ann);    ann = analog(3);
    ann = ann + 3;       return ann;
    ...                  }
...                      }
```



14

Global Variables

- Variables created outside a function:
 - at start of program - before main()
 - available to all functions in program
 - can **not** have local variables with same name
 - use only when absolutely necessary!

```
int light;               void drive( int speed)
void main(void)          {
{                          motor(1, speed);
    int ann = 20;         light = analog(3);
    drive(ann);           }
    if (light > 20)
    {
    ...
}
```



15

Defining Timeout Functions

- Define functions to set and check time limits
 - limit value stored in *global variable* - accessible to both functions

```
// function to set time limit - stored as maxtime
void timelimit(float limit)
{
    maxtime = seconds() + limit; // set end time
}

// function to check time limit - return true if time up
int timeup(void)
{
    return (seconds() >= maxtime);
}
```



16

Using Timeout Functions

```
// define global variable to hold time limit
float maxtime; // defined outside any function

void main( )
{
    ...
    spinRight(50); // start robot turning
    timelimit(5.0) // allow 5 seconds
    while ( checkbeacon(2) != mybeacon
            && !timeup() ) { }
    // wait until found or 5s passes
    drive(100); // drive to beacon or elsewhere
    ...
}
```



17

Good programming - #define

```
#define LEFT_MOTOR 1
#define RIGHT_MOTOR 2
#define LEFT_SWITCH digital(9)
#define RIGHT_SWITCH digital(10)

• use at start of program
  ▪ compiler will replace LEFT_MOTOR with 1,
    wherever it occurs
  ▪ normal to use capital letters - not required
  ▪ cannot put comment on same line!

motor(LEFT_MOTOR, 100); //spin right
motor(RIGHT_MOTOR, -100);

...
if (LEFT_SWITCH || RIGHT_SWITCH) { collision } 18
```



18