Microcontroller Project
MXEN2003

Dr Nasrin Afsarimanesh

Curtin University

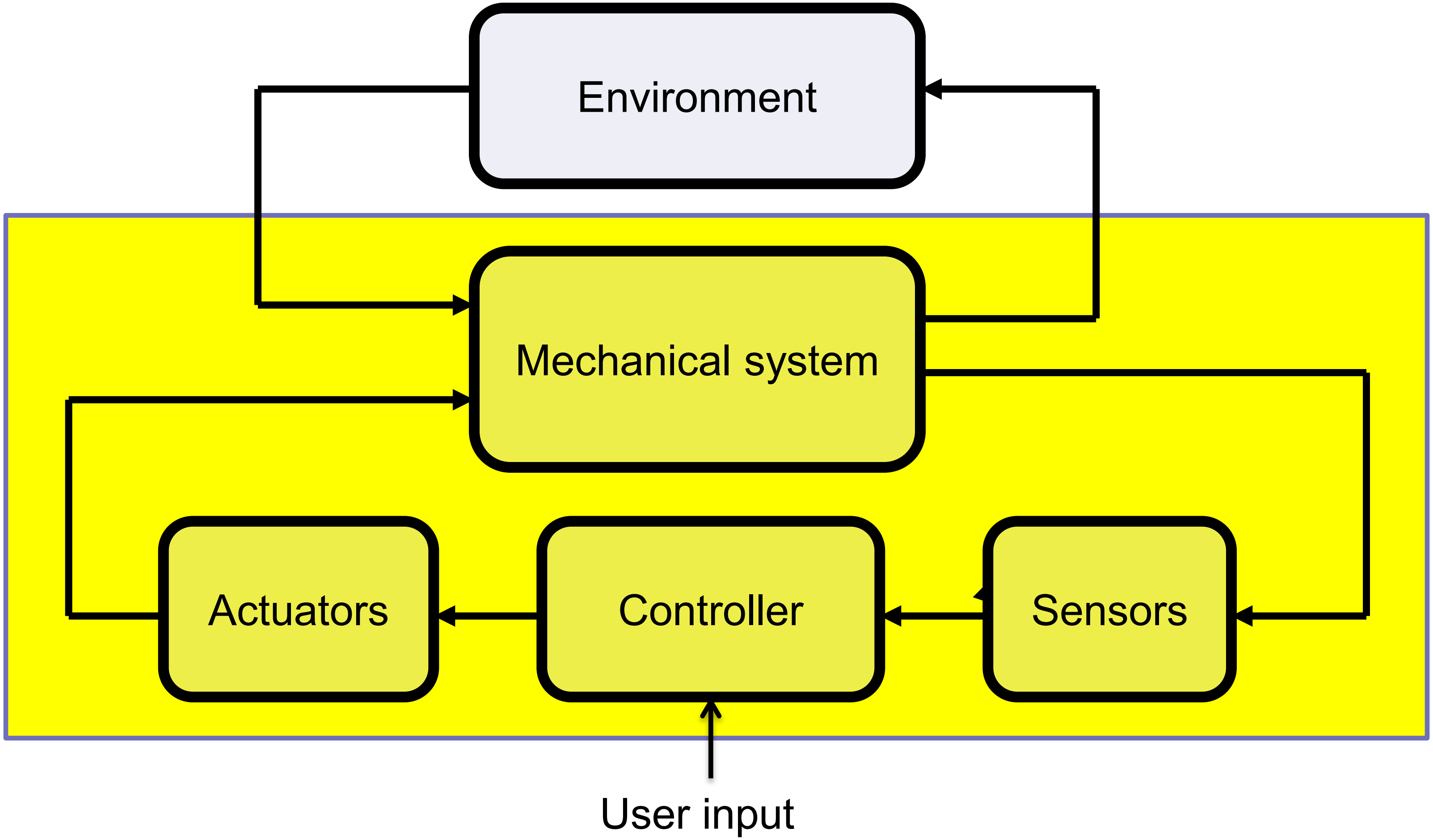
Overview

- Structure of a mechatronic system
- Examples
- Lab write-ups
- Data types in C
- Digital input and output
- Analog input
- Pseudocode and flowcharts

Features of a mechatronic system

- A **mechanical system** – interacting with a known or unknown **environment**
- **Actuators** – perform physical actions on the system or environment
- **Sensors** – measure the state of the system or environment
  - Measure positions, speeds, temperatures, pressures, chemical states
- **Controller** – drives actuator actions based on the sensor measurements
  - Can be a computer, microcontroller, or a digital or analog electronic circuit
- **Design**

The structure of a mechatronic system

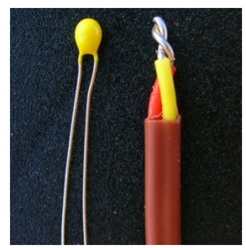


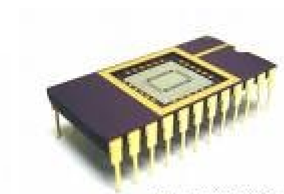
Feedback plays a critical role

Sensors

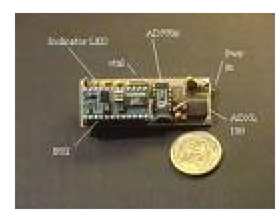
- Devices which measure some aspect of the state of the system
- Many sensor modalities







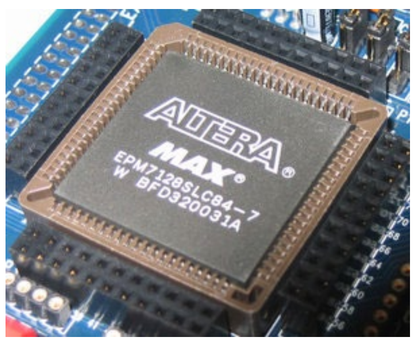



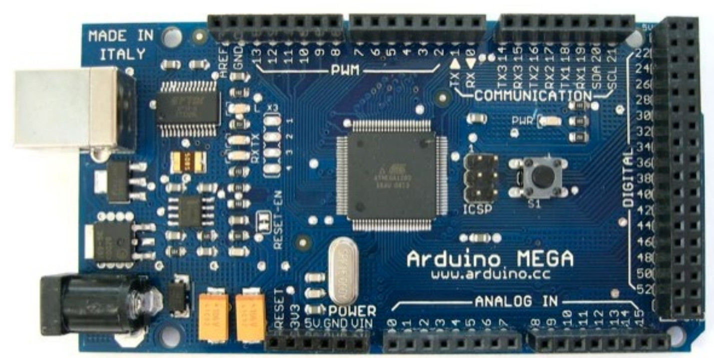
Actuators

- Devices which perform an “action” on the system

Controller

- The "brain"
  - Can incorporate electronics, computer systems, algorithms and software





Example: Washing machine

- Actuators:
  - AC or DC Motors
  - Water inlet/drain valves
- Sensors:
  - Water level
  - Load speed/balance
  - Temperature
- Control
  - Pre-defined programs implemented in a microcontroller

Example: Autonomous dock

- Task: move shipping containers from shipside to stacks, and from stacks to trucks *without direct human intervention*
- Motivation:
  - Improved efficiency
  - Safety
  - Lower cost
  - $Profit$

Example: Autonomous dock

https://www.youtube.com/watch?v=t8I-oOInbeA

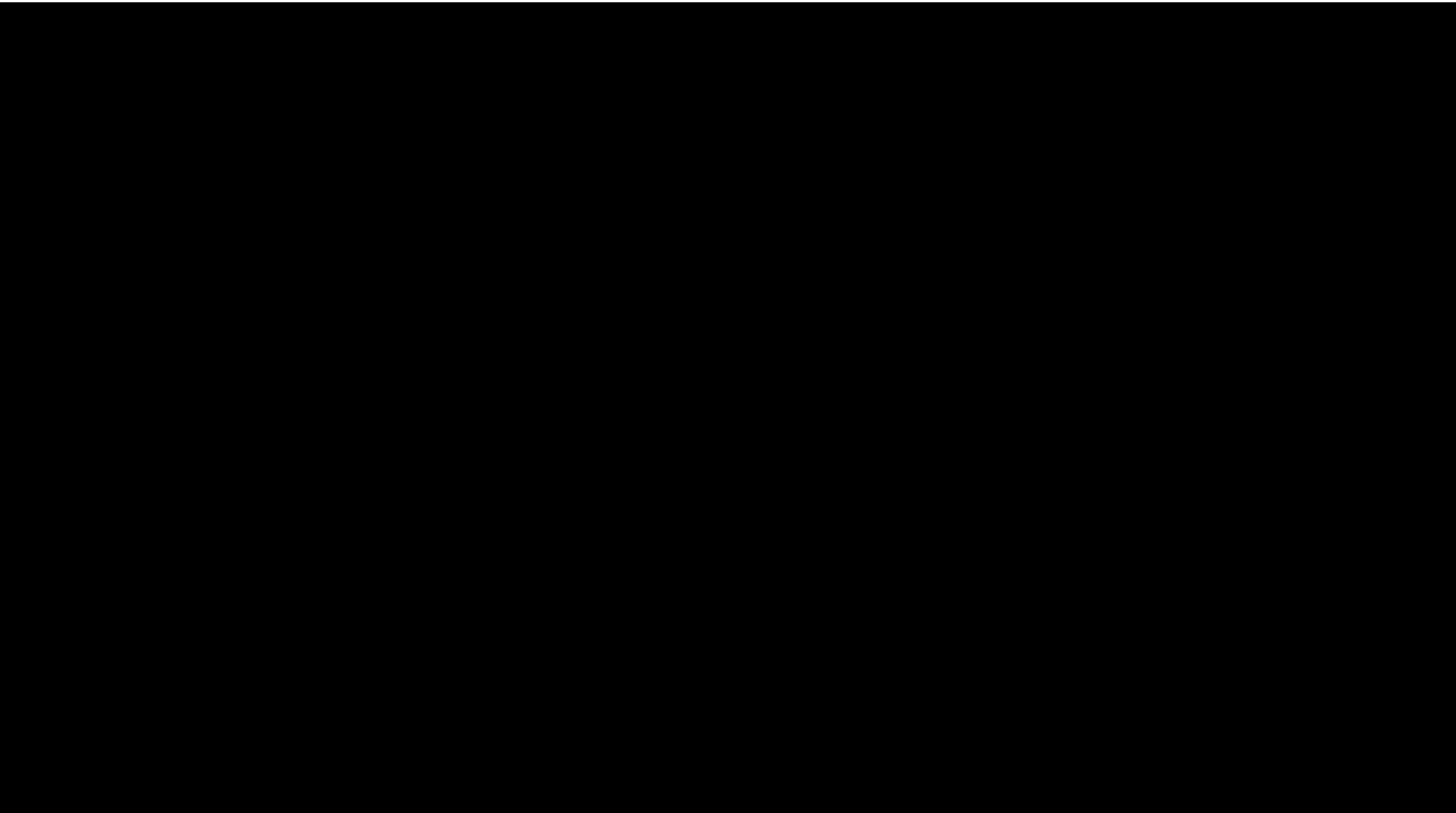
Autonomous *straddle carriers*

- Four SICK LIDARs
- RADAR beacon localization
- Inertial Navigation System (INS)
- GPS available but obsolete to RADAR

Urban Search and Rescue

- In extreme situations, the risk to human rescuers must be minimized
- Autonomous or teleoperated robots can be used to find victims, map environments and locate hazards with minimal risk to the human teams
- This means the human rescue effort can be better targeted

Urban Search and Rescue



https://www.youtube.com/watch?v=u8KrAAGzNMk

Laboratory marking scheme

- Five marks are available for each of the 8 weekly laboratories.
- Each mark will be awarded based on the demonstration of a technical outcome, or for showing an element of the required documentation.
- Note that a **complete documentation** for each task is expected in your lab book, even if only particular elements are awarded marks. This should include an **objective** for each lab session.
- Remember that most of the technical elements of the weekly labs will be incorporated in the final task, so you will need good documentation.

Data types

In C, unlike in Python, we must *declare* the *type* of a variable before it is used. Some of the datatypes available are integers (signed and unsigned), floating point numbers, characters, arrays and strings.

Most of the datatypes we use are some form of integer, and when programming microcontrollers, we are normally very specific about how big our integers are (e.g 8 bit, 16 bit etc).

Variables should also be initialised (given an initial value). This can be part of the declaration.

Example data types

- bool (1 bit)
- int8_t (char signed 8 bits, range -128 to 127)
- uint8_t (unsigned char 8 bits, range 0 to 255)
- int16_t (int signed 16 bits, range -32k to +32k)
- uint16_t (unsigned int unsigned 16 bit, range is 0 to 65k)
- int32_t (long signed 32 bit)
- uint32_t (unsigned long unsigned 32 bit)
- float (signed 32 bit floating point)
- double (signed 64 bit floating point)

(you may be more familiar with int, unsigned int, long, unsigned long in C – we prefer here to use types that specify the size of integer types)

Remember we are programming for an 8-bit processor. Stick to integer types if you can.

Variable declaration

Variables must:

- be declared and given a type

Variables must not be:

- *Reserved* word such as for, while

- Use descriptive variable names: e.g *output_string* instead of *S*

Examples of variable declarations

```
int8_t intOutputMask;          //signed 8 bit integer
intOutputMask=4;

uint8_t intOutputMask=0;       //unsigned 8 bit integer
                 //we can declare and initialise on one line

float TimingResult;            //floating point number
TimingResult=0.0365;           //we will usually avoid floats

bool PowerButtonFlag;          //Boolean variable
PowerButtonFlag=true;          //same as PowerButtonFlag=1

char strGreeting[];            //string variable (array of chars)
strGreeting="Hello, this is a greeting";
```

Data types: bool

```
bool Flag1;         //declares Flag1 as a bool
```

- A single bit, which takes the value 0 or 1

```
Flag1=1; //sets the variable Flag1 to 1, or "ON"
Flag1=0; //clears Flag1 to 0, or "OFF"
```

Data types: uint8_t (unsigned char)

```
uint8_t Num;  //declares Num as a byte
```

- 8 bit unsigned integers: 0-255
- Can assign using decimal, hexadecimal or binary:

```
Num=123;
Num=0x7B;
Num=0b01111011;
```

Data types: int16_t (int), uint8_t (unsigned int) and long

- int16_t (int): 16-bit **signed** integers, 65k values in total. The sign bit is bit 15 or MSB. The range is -32768 to +32767

- uint8_t (unsigned int): 16-bit **unsigned** integers, 65k values in total. Range 0 to 65535.

- int32_t (long): 32-bit **signed** integers. Range -2 147 483 648 to +2 147 483 647

- uint32_t (unsigned long): 32-bit un**signed** integers. Range 0 to +4 294 967 295

Data types: Integer overflows

```
uint8_t Num;                    //remember a byte is 0 to 255
Num=255;
Num=Num+1;
                                                  0
int16_t Num;                    //int is -32768 to 32767
Num=22767;
Num=Num+20000;
                                                  -22767
uint16_t Num;            //unsigned int is 0 to 65535
Num=27;
Num=Num-30;
                                                  65533
```

Data types: float and double

- float stores signed 32-bit fractional numbers.
- 1-bit ***sign***
- 8-bit ***exponent***
- 23-bit ***mantissa*** (or ***significand* – *the precision bits***)
- 7 significant figures : e.g **1.6777216**x10^(**255**)
- **Avoid** using float variables in real-time control applications because they are *much* slower than int or byte
- double stores 32-bit fractions (52 bit mantissa, 11 bit exponent)

Data types: char

- a char is simply an 8 bit signed integer, however it is often used to store ASCII characters
- A character array can be used to store a string
- Strings can be joined together

```
char myChar;          //a single character
char strName[30]; //a string of fixed length
char strName[]; //a string of unspecified length
//the compiler will decide how much memory to
  allocate
```

ASCII codes

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Data types: Arrays

- An array is a set of sequentially indexed elements having the same type.

```
uint8_t Array[10];          //declare 10 byte array
```

- Requires an integer type index

```
uint8_t c;          //indexing number
```

Example

```
for(c=0;c<10;c++)//display loop that repeats 10 times
  Array(c)=c;
// Array=[0 1 2 3 4 5 6 7 8 9]
```

- Avoid declaring very large arrays (limited RAM)

Casting data types

- Be very careful when using variables of different types! The result may not be what you expect

```
uint16_t uintVal;
uint8_t byteVal;
float floatVal;
char strVal;
floatVal=84.435;
byteVal=floatVal; //result is 84
strVal=byteVal;   //result is T
uintVal=17676;
byteVal=uintVal;  //result is 12: least
                             significant byte
```

Casting data type

- (char) var – converts var to a char type
- (uint8_t) var – converts var to a byte type
- (int16_t) var – converts var to an int type
- (int32_t) var – converts var to a long
- (float) var – converts var to a float

- Some casting is done automatically, but be careful if you mix data types: the result may not be what you expect.

Digital I/O for a microcontroller (review)

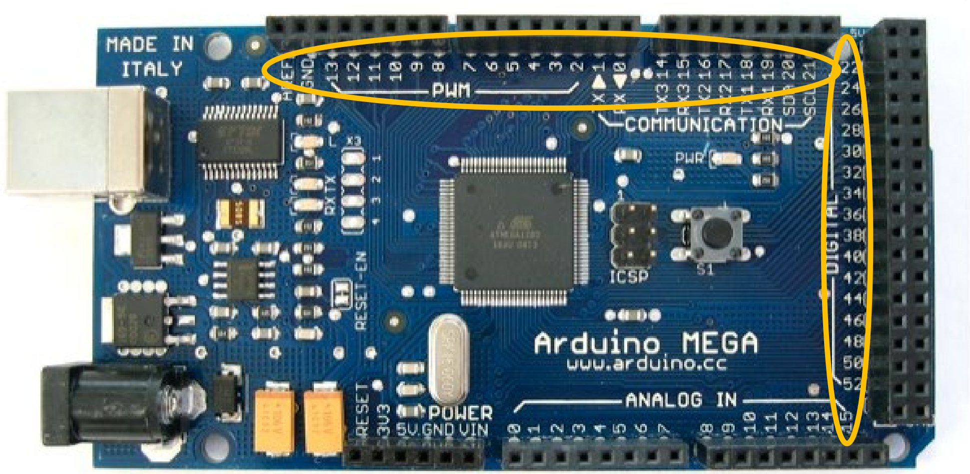
- Registers are memory locations are used for particular hardware purposes.
- Atmel microcontrollers have registers associated with each I/O port for **input**, **output** and **data direction.**
- Every port has a Data Direction Register (DDR), which tells the microcontroller whether each pin is used for **input** or for **output**
- E.g. `DDRA` sets all the bits on Port A
- `DDRA=0b11110000;`

```
    // sets the upper nibble on port A as
    // output, and the lower nibble as input
    // note 0b not in the C standard, but works for us
```

- `DDRK=0xFF;      // sets the whole of port K as output`
- `DDRK|=(1<<2);          // sets pin 2 port K as output`
- `DDRK&=~(1<<2);  // sets pin 2 port K as input`

I/O pin allocation (review)

- Arduino has its own pin numbering system, which is different from the microcontroller port labels
- To use the port labels (PORTA, PORTB etc), you will need to look up the pin associations, either on the Arduino schematic or other reference.
- A pin allocation document has been provided to you on Blackboard (see Lab 1)



Ports: Writing (review)

- Output ports can be accessed directly using the PORTx register (where x is the port letter)
  - Remember to set DDRx high before writing
- Entire ports can be written at once
  - `PORTA=0b10101010;   // binary`
  - `PORTA=0xAA;         // hexadecimal`
- Ports bits can be set individually using an OR operator or cleared (reset) using an AND
  - `PORTC=PORTC | 0b00000100;`
    `//sets the bit 2 only`
  - `PORTC=PORTC & 0b11111011;`
    `//clears the bit 2 only`
  - `PORTC=PORTC ^ 0b00000100;`
    `//toggles bit 2 only`

Ports: Writing (review)

- We can use the *bitwise shift* operators to do this more compactly (noting that **1= binary 00000001**):
  - `PORTC=PORTC | 0b00000100; //sets the bit 2 only`
    `// note above is not ANSI C standard`
  - `PORTC=PORTC|(1<<2);// identical using bit shift`
  - `PORTC|=(1<<PC2); // even more compact version`
  - `PORTC=PORTC&0b11111011;//clears the bit 2 only`
  - `PORTC=PORTC&~(1<<2); // identical to above`
  - `PORTC&=~(1<<PC2);  // identical to above`
  - `PORTC=PORTC^(1<<2);// toggles bit 2 only`
  - `PORTC^=(1<<PC2);   // even more compact version`

Ports: Reading

- Input ports can be interrogated via the register PINx, (where x is the port letter)
  - Remember to set DDRx low before reading
- PINA returns the (8 bit) value of Port A
- Some typical commands...

```
bool x;  //declare x as boolean (also not ANSI C)
uint8_t y;//declare y as byte (unsigned 8bit int)
DDRA=0;  // set PORTA to input (read) mode
x=PINA & 0b00000100;
   // x is now equal to the value on port A pin 2
x=PINA & (1<<PA2); // identical to above command
y=PINA;  // y is now equal to all bits on port A
if (PINA & (1<<PA2))  // do things when input high
else     // do things when input low
```

Ports: Reading

- If a particular pin is configured for reading (i.e the DDR is set low), then you use the appropriate PIN register to check for its value. The PORT register can then be used to enable an internal pullup resistor (e.g if the input is a button to ground)

```
bool x;  //declare x as boolean (also not ANSI C)
uint8_t y;//declare y as byte (unsigned 8bit int)

DDRA=0;  // set PORTA to input (read) mode

PORTA|=(1<<PA2);      // enable pullup on pin 2
x=PINA & (1<<PA2);    // read boolean value on pin 2
```

Analogue input and the ADC

- The ATmega2560 has 16 pins for the analogue to digital converter (ADC) - the PORT F and K pins
- We can read 16 separate analogue signals
- Some functions to access the ADC are provided to you: please see adc.h and adc.c
- The analog system is setup using the adc_init() function
- We can use the adc_read(n) function to read from pin n (where n is the analogue pin number on the board)
- adc_read(n) returns a 10-bit integer (0-1023), scaled from
  - 0 to 5V in the default mode provided
  - It is possible to configure the ADC to use an external voltage reference or an internal 1.1V reference

Example

```
static uint16_t W;
static uint8_t channel = 0;

int main()
{
   adc_init();              // initialise ADC to read analog
   serial0_init();          //initialise serial subsystem


    while (1)
    {
      W = adc_read(channel);   //read a value from analog
                               //channel 0
      serial0_write_byte(W);   //write digital value
                               //to serial terminal
    }
}
```

Pseudocode

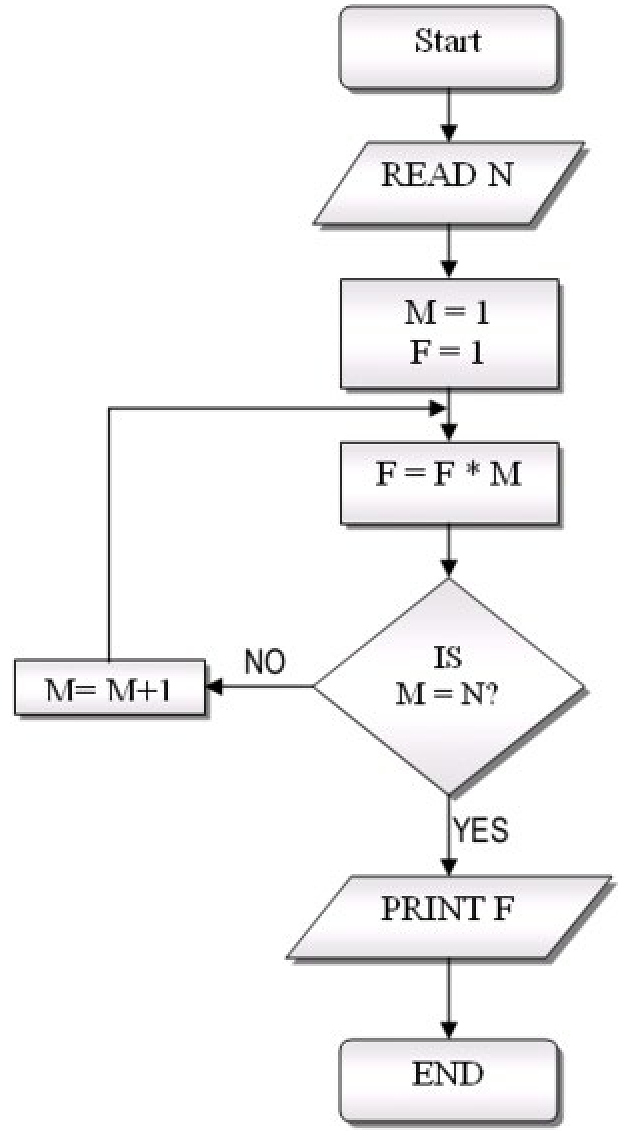
- An informal, compact, readable, high-level description of a computer algorithm
- May make use of keywords from a particular language (e.g BASIC)

```
READ string S
N=0
output_string=""
FOR N=0 to length(S)
     IF string(N)='i' THEN
        append 'u' to
        output_string
     ELSE
        append string(N) to
        output_string
     END IF
END
PRINT output_string
```

Fish and chips → Fush and chups!!!

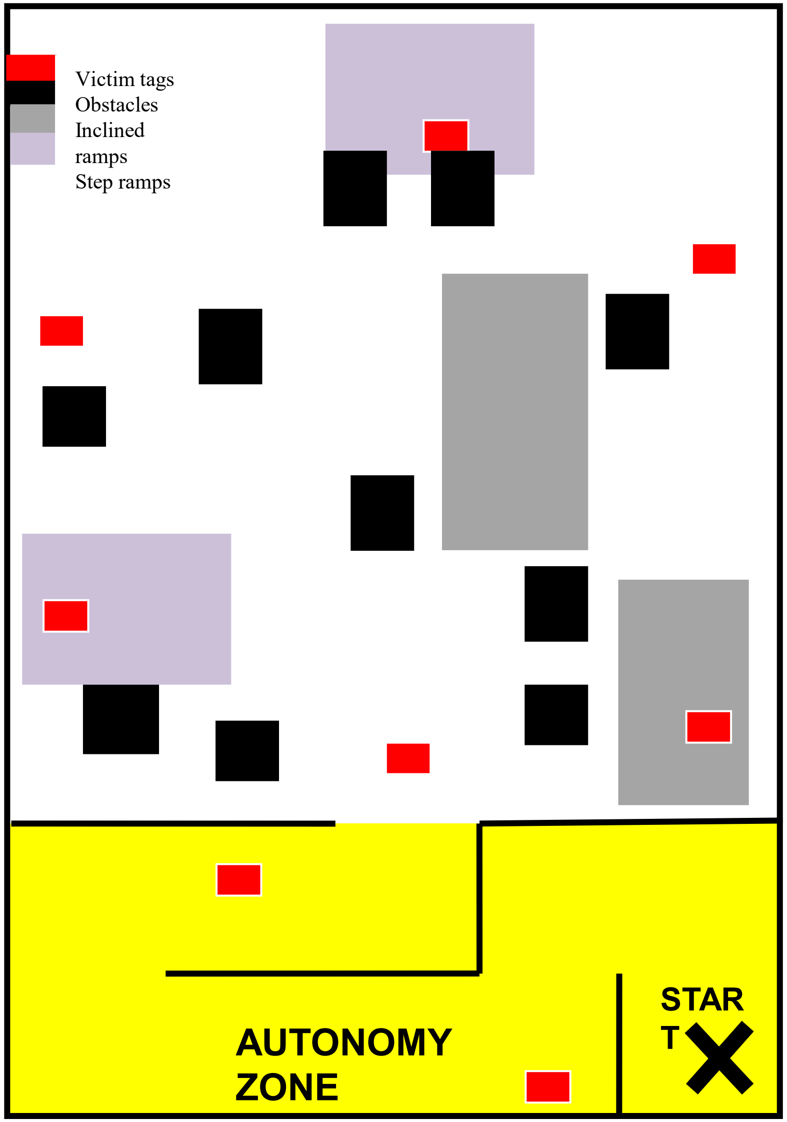
Flowcharts: graphical pseudocode

- Arrows represent a sequence of execution, or *flow of control*
- Diamonds represent decisions, or *conditional flow*
- Rectangles represent *processing* – a more complex process may be abstracted
- Parallelograms represent *input* or *output*
- Rounded rectangles represent *start* or *end* of execution



Project task: Urban Search and Rescue

- A building has collapsed, and is hazardous to enter. Remote and autonomous technology is required to search for victims, to minimize the risk to human rescuers.
- Design a remotely controlled robot, which will traverse the search zone, equipped with a wireless camera and range sensors.
- Due to a radio blackout, part of the search area may require autonomous navigation.



Project competition