float

Floating-point numbers can be as large as 3.4028235E+38 and as low as -3.4028235E+38. They are stored as 32 bits (4 bytes) of information

float var =val;

enumeration

Instead of integers ~~or real numbers~~ we will enumerate the member of the data type

can now declare a variable as data type dir_t, in the same way as uint8_t,...).

```
enum dir_t { south, east, west, north };
             0b00   0b01  0b10  0b11
             0      1     2     3

dir_t facing;
facing = north;
if ( facing == north ){
    facing = south;
}
}
```

```
dir_t turnRight( dir_t someDir)
{ if( someDir ==
  else if
  :
void turnRight(){
return
}
```

0b00

$00000.00

Initialize elements of an array to zero

initialize 6 elements to zero

```
unit_8  myRobot[6];
unit_8  myRobot[6] = {0};
unit_8  myRobot[6] = {};
```

millis() data type ? unsigned long 32 bits uint32_t quiz

casting

casting

8 bit unsigned

randomSeed((uint8_t) millis());

32-bit unsigned

uint8_t(millis())

new school C++

old school C++

casting 32-bit unsigned to a 8-bit unsigned

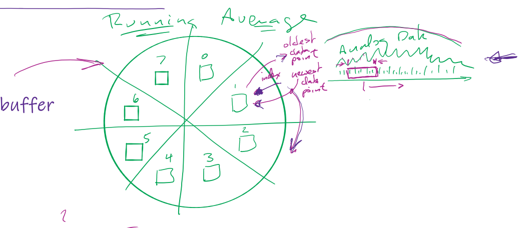
seed

equation ⟹ □□□□□□□

random number generator

Running Average

circular buffer



?

```
const uint16_t g_width = 8; // number of 10-bit ADC samples averaged
                            // over the data stream. maximum size = 64 (2^6)

void setup() {
    for (int i=0; i < g_width; i++) readADC(A0); // initialize buffer (opt.)
}

void loop() {
  uint16_t adc = readADC(A0);
}

uint16_t readADC(int sensorPin){
  static uint16_t window[g_width]={0}; // window into ADC data stream. [1]
  static int index = 0;                // index pointer into the buffer
  static uint16_t sum = 0;             // sum of the readings in the buffer

  uint16_t A_0 = window[index];          // oldest reading
  uint16_t A_1 = analogRead(sensorPin); // current reading
  sum += A_1 - A_0;                      // subtract out oldest reading and
                                         // replace with current reading

  // update
  window[index] = A_1;                // update buffer
  index++;                            // move pointer
  if(index >= g_width) index = 0;     // circular buffer [2]
                                      // return average reading across window
  return sum/g_width;                 // integer division (rounds down)[3]
}
```

8

save to SRAM

running sum

sum = sum + A1 - A0

1)

0 . . . 7 0 - - 7

2) Modulo Operator

$n \% 8 =$

$8\overline{)8}$ 1 r 0

$8\overline{)7}$ 0 r 7

$8\overline{)9}$ 1 r 1

n % 8;

C++ Programmer obscure & slow

3) n &= 0X000F;

obscure & fast in pipelined architecture

will this work ?

4) You could also use a shift operator in place of division (multiple of 2) Unfortunately, our processor does not have a barrel shifter. Alternative float version: return sum/float(g_width);

&

*

address of operators or reference operators &
https://www.cplusplus.com/doc/tutorial/pointers/

```
uint8_t  myvar = 25;
         foo = &myvar;
         blah = myvar
```

// datatype & declaration of foo covered shortly

myvar	~~0x0101~~ 0x0100	25	foo = 0x0100 blah = 25

Pointers or Dereference Operator *

blah = *foo   blah is equal to the value pointed to by foo

Summary

& is the address-of operator, and can be read simply as "address of"
* is the dereference operator, and can be read as "value pointed to by"

Declaring pointers

Due to the ability of a pointer to directly refer to the value that it points to, a pointer has different properties when it points to a char than when it points to an int or a float. Once dereferenced, the type needs to be known. And for that, the declaration of a pointer needs to include the data type the pointer is going to point to.

The declaration of pointers follows this syntax:

type * name;

Where type is the data type pointed to by the pointer. This type is not the type of the pointer itself, but the type of the data the pointer points to. For example:

uint8_t * foo;

foo points to a data type uint8_t

The size in memory of a pointer depends on the platform where the program runs.

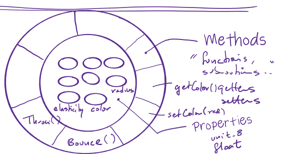
That the asterisk (*) used when declaring a pointer only means that it is a pointer (it is part of its type compound specifier), and should not be confused with the dereference operator seen a bit earlier, but which is also written with an asterisk (*). They are simply two different things represented with the same sign.

To see how confusing this can get, check-out Lecture 3 "Software Programming in C++: Introducition (https://www.arxterra.com/c-introduction/) and specifically chapter 11 "What is _SFR_Byte(sfr)?"

_SFR_BYTE(sfr) *(volatile uint8_t * uint16_t &(sfr))

OOPS

Object Oriented Programming



Methods
"functions, subroutines.."
getColor() getters
setters
setColor(red)
Properties
unit_8
float
radius
elasticity
color
Throw()
Bounce()