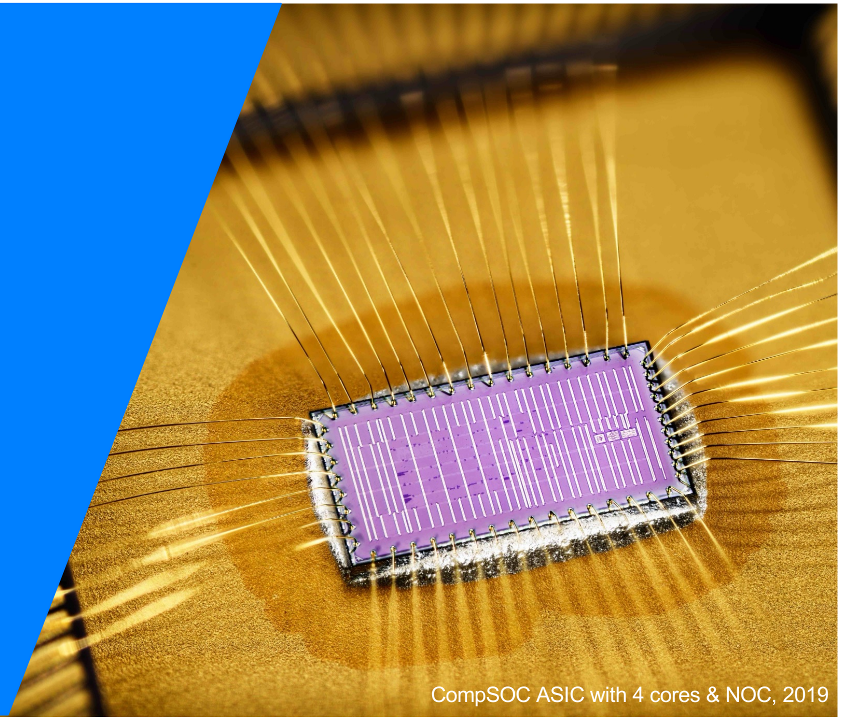
5EWC0
Programming & Engineering Challenge

Kees Goossens



CompSOC ASIC with 4 cores & NOC, 2019

https://kgoossens.estue.nl/docs/studiekeuzecheck/

Kees Goossens <k.g.w.goossens@tue.nl>
Electronic Systems Group
Electrical Engineering Faculty

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Let's try out the online quiz on menti.com

- where are you following to this lecture?
- go to www.menti.com and enter the code **2778 4171**

Mentimeter

- are you following this lecture on
  1. a computer / laptop
  2. tablet
  3. phone
  4. smart glasses

spectacles.com

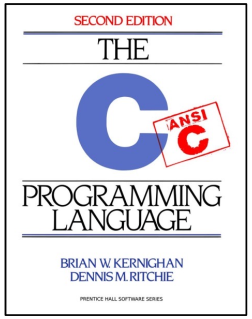
- I'll assume that on 1-3 you can program, later

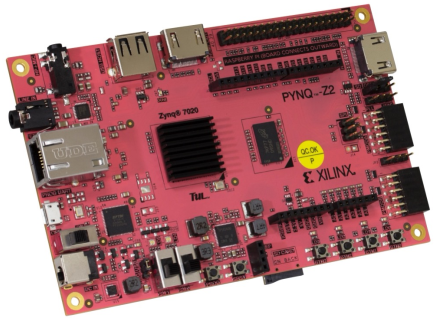
outline

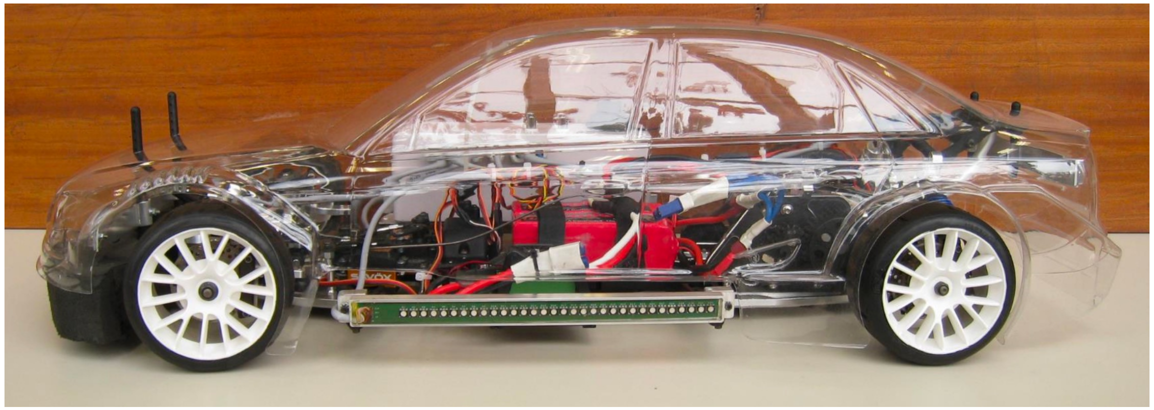
- computers are everywhere
- what is the course about
- why is programming important
- C programming language practicum

course goals

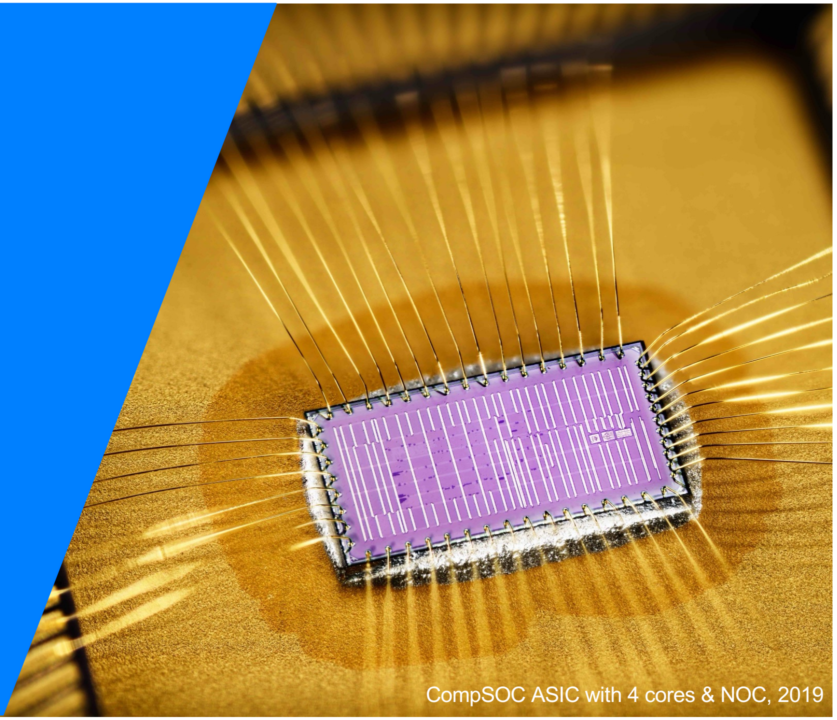
- to learn to
  - program in the C language
  - design a system from a real-world problem statement
    - EE: Rock Your Baby
    - AT: Energy Challenge







Computers



CompSOC ASIC with 4 cores & NOC, 2019

Kees Goossens <k.g.w.goossens@tue.nl>
Electronic Systems Group
Electrical Engineering Faculty

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

computing: the personal computer

TRS-80 Model 1
my first computer in 1980
4 KB, 1.7 MHz

IBM 5150 PC, 1981.
16 KB, 4.7 MHz

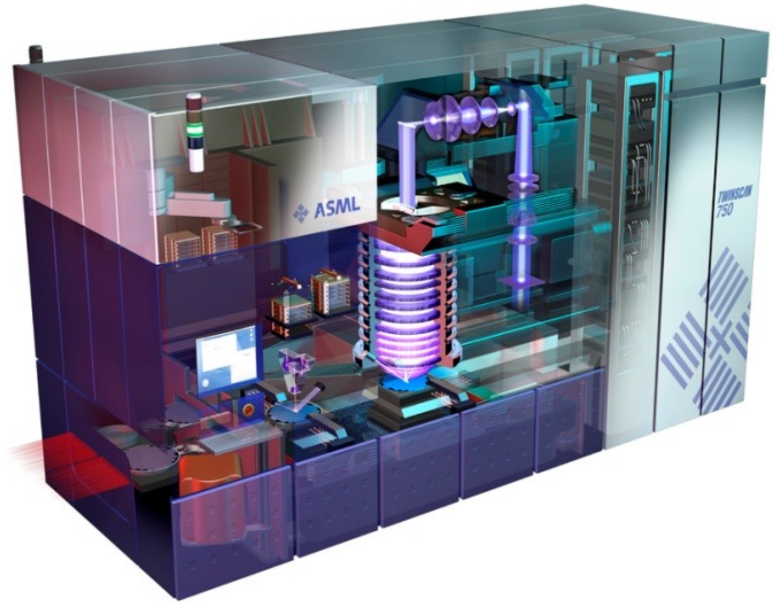
Osborne 1, the first “portable” computer
– only 10 kilos.

iPad mini, 256 GB, 2.5 GHz,
0.3 kilo

iPhone Pro, 512 GB, 3.1 GHz,
0.19 kilo

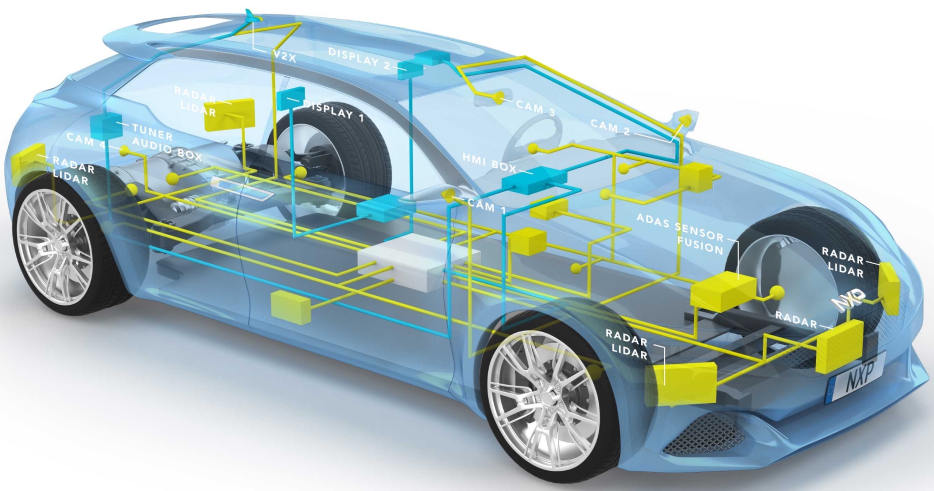
computing: embedded & cyber-physical systems

Philips medical imaging



ASML chip manufacturing

computing: cars contain networks of computers



up to 100 computers and
six networks in modern cars

TUE's STORM electric motorbike
– went around the world
– EE automotive students!

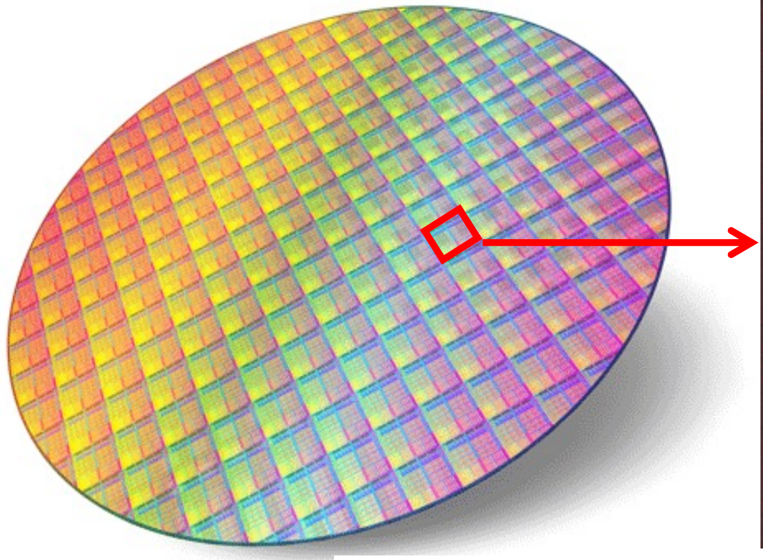
the importance of computers

- critical infrastructure
  - financial system
  - all communications, including Internet, satellites, your mobile phone
  - power grid
- cars, trains, planes, e-bikes, ...
- medicine, hospitals, smart pills (electronic, not psychedelic)
- microwave, washing machines, passports, clothes tags, ...
- ...

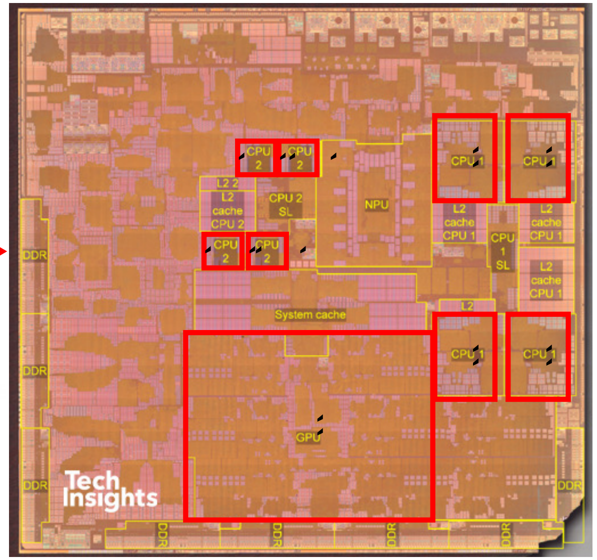
- invented & built by engineers (you!)
- any modern system is too complex to design & build without computers

designing & manufacturing chips

cool guy in a factory (“fab”)

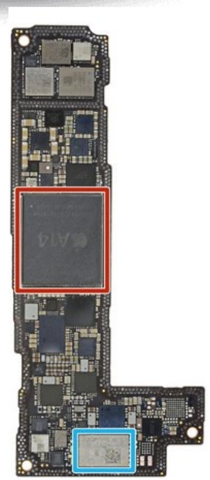


wafer

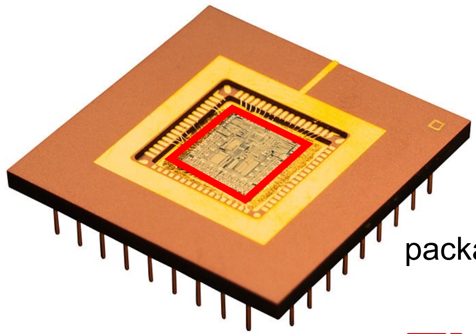


iPhone 12 A14 die, 4x CPU, 1x GPU

phone



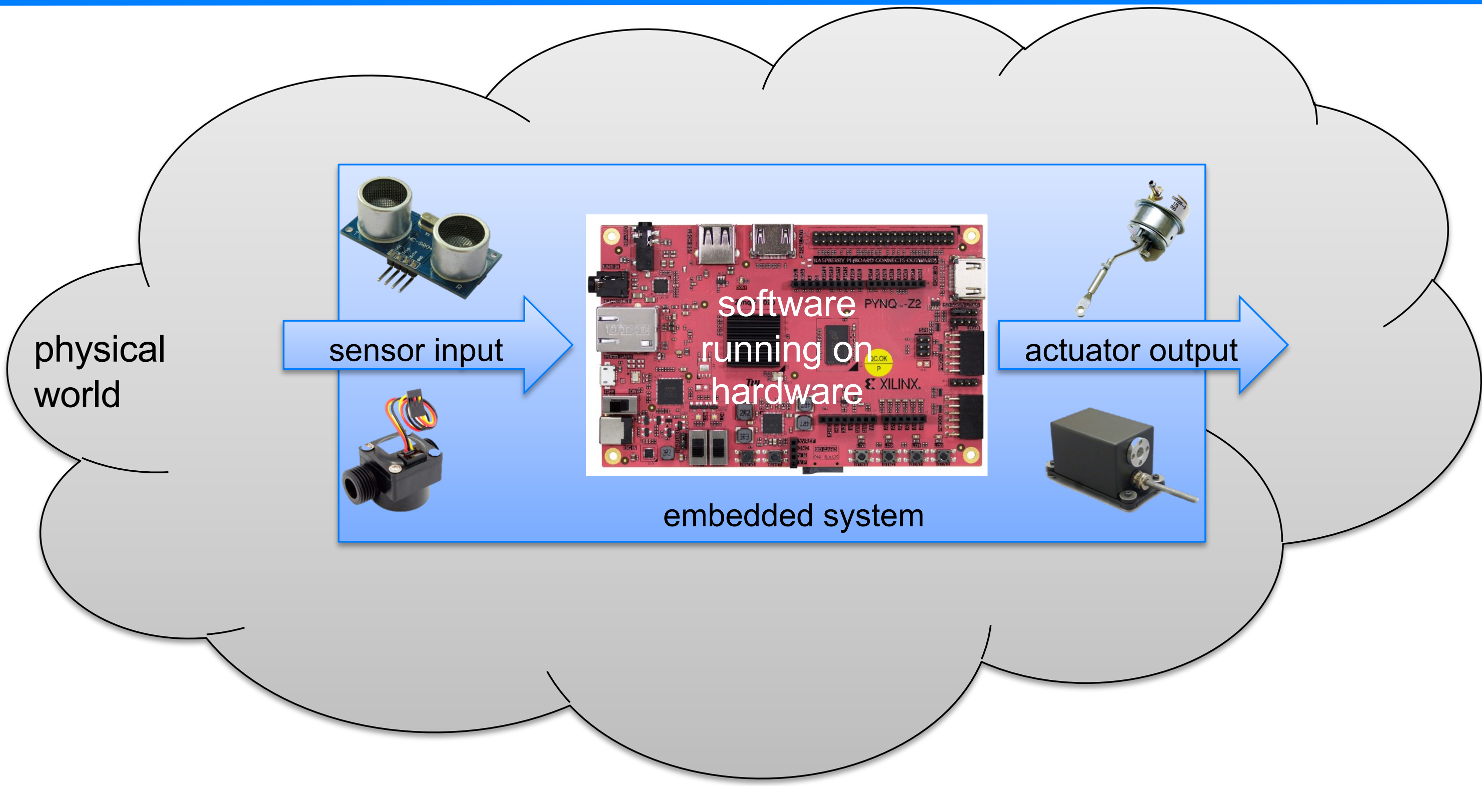
motherboard
or PCB

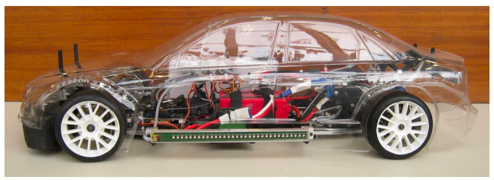


package

TU/e

Computer Architecture, Programming, Embedded Systems



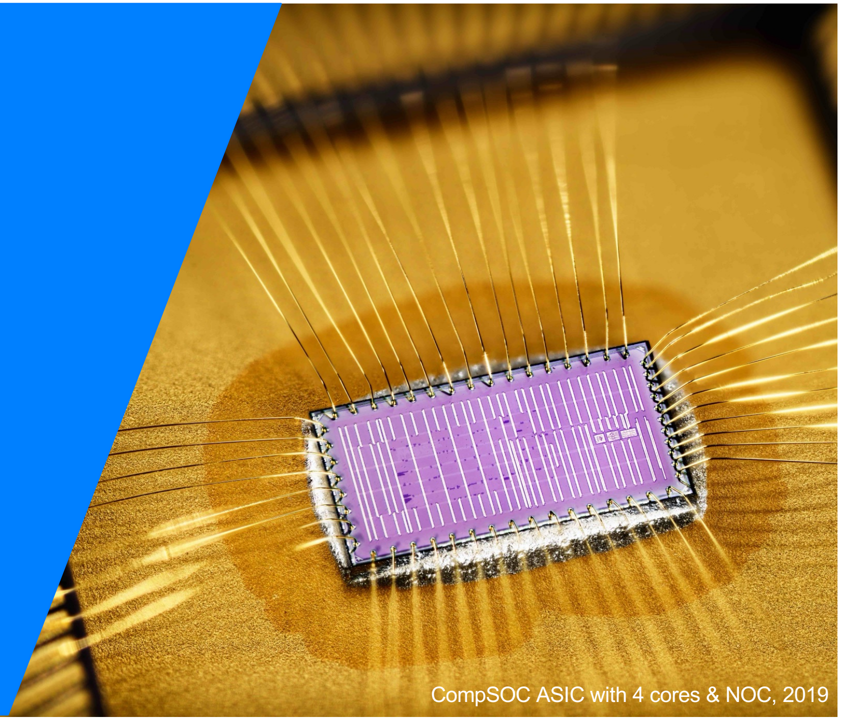


full picture will be covered in 3 courses:

- 5EWC0 programming
- 5EIC0 Computer Architecture I
- 5AIB0 Sensing, Computing, Actuating

5EWC0
Programming & Engineering Challenge

C programming



CompSOC ASIC with 4 cores & NOC, 2019

Kees Goossens <k.g.w.goossens@tue.nl>
Electronic Systems Group
Electrical Engineering Faculty

EINDHOVEN UNIVERSITY OF TECHNOLOGY

why is programming important anyway?

- engineering in general
  - problem analysis
  - structured approach & logical thinking
  - precision

especially so in programming
- the computer does exactly what you tell it do
- not what you intend or think to have programmed
- embedded systems are often safety critical
  – programming errors can lead to fatalities

- science relies on mathematics, modelling, & experiments
- EE also designs systems

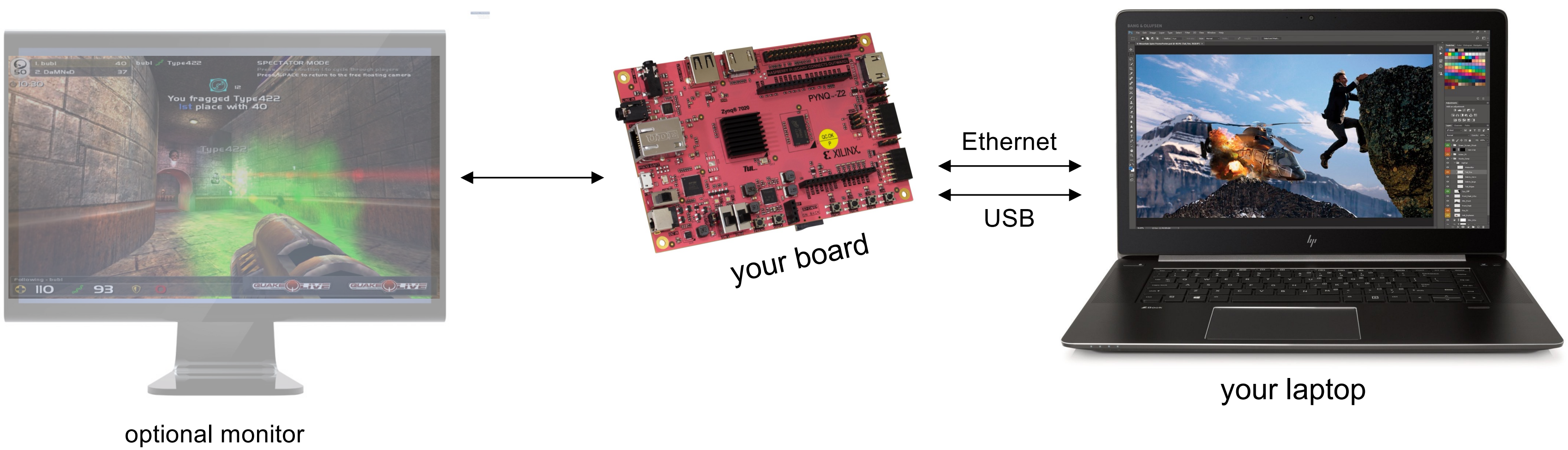
- automation is essential → programming!

in general, programming errors
→ failing or wrong experiments
→ invalid analysis
→ bugs in systems
→ incorrect conclusions

- programming is relatively easy
- the real skill is problem solving

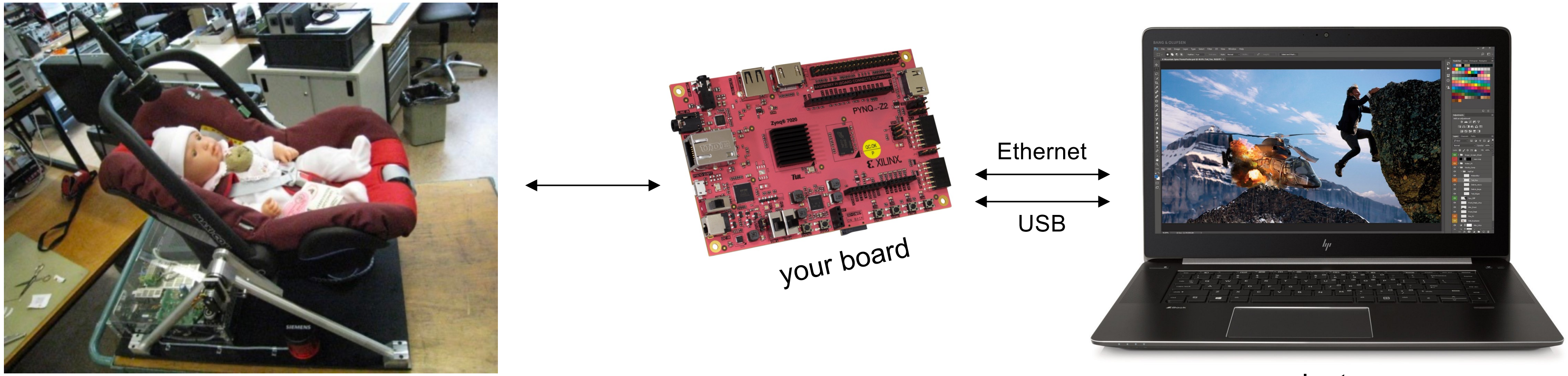
PYNQ-Z2 board – during the programming labs

- used in at least 3 of your courses
- dual-core ARM processor
- Ubuntu Linux
- programmable logic



PYNQ-Z2 board – in the challenges

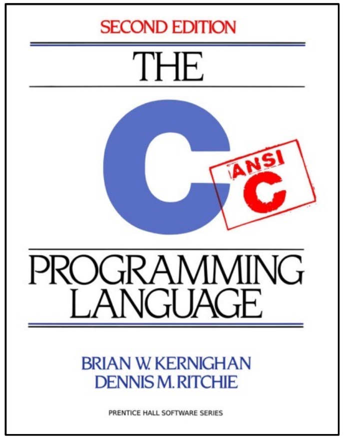
- used in at least 3 of your courses
- dual-core ARM processor
- Ubuntu Linux
- programmable logic



programming

- many different programming languages
- C
  - is the most widely used language in the world
  - especially for embedded systems
  - is at the boundary between hardware & software

- in this practicum
  - online interface
  - printing
  - variables
  - loops (automation)
  - input (data dependence)
  - conditionals (data dependence)



Kernighan & Ritchie, "C Programming Language", 2nd edition, ISBN: 978-0-13-110362-7

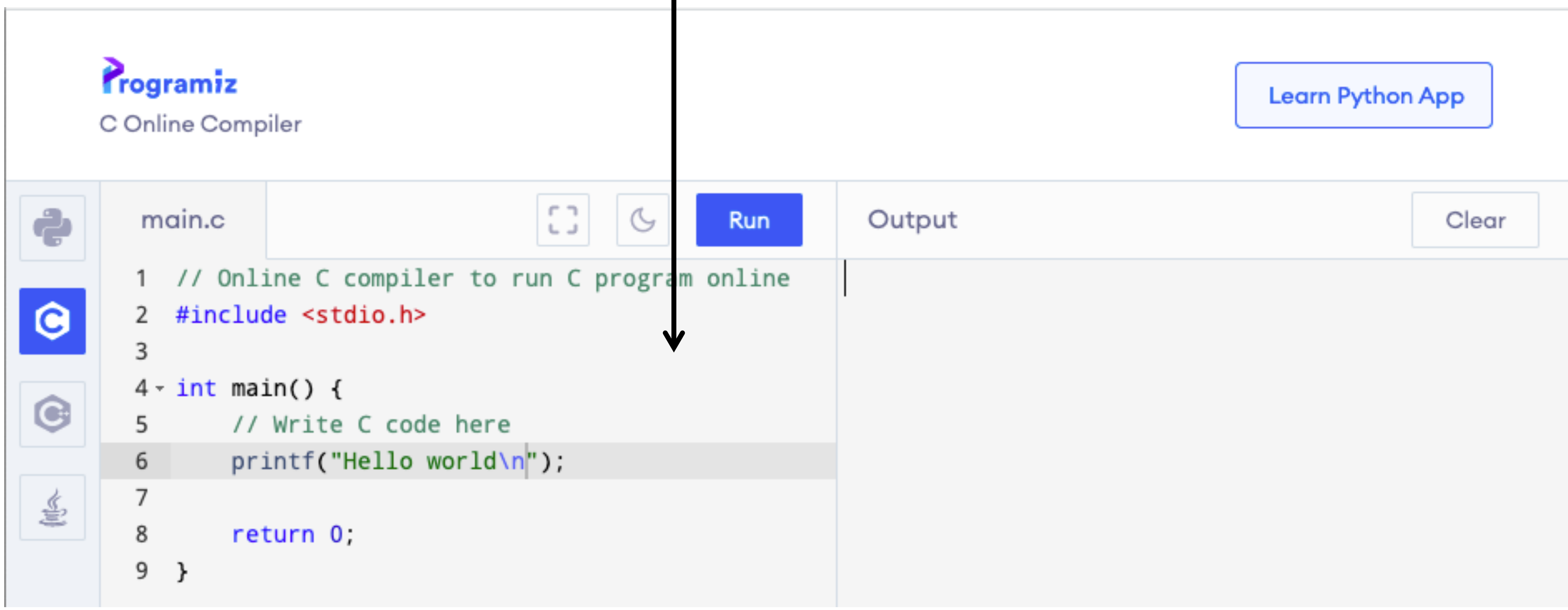
Mentimeter

- have you programmed before?
- what programming languages?

online C programming

- https://www.programiz.com/c-programming/online-compiler/

your program

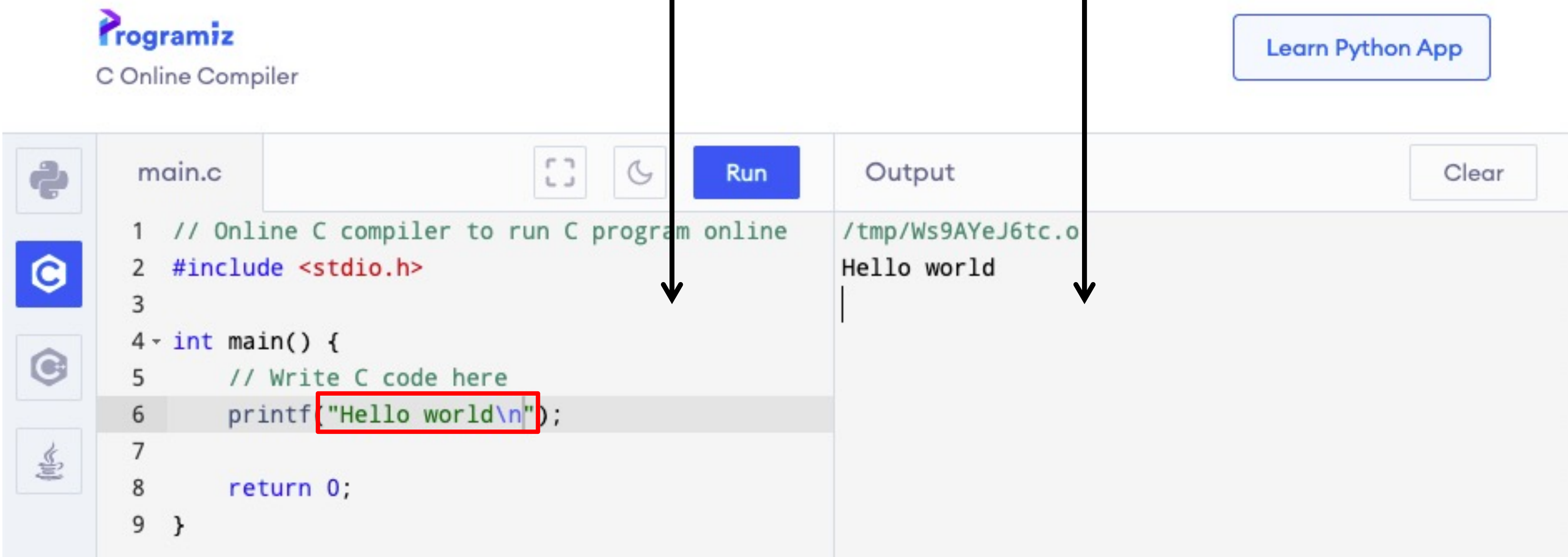


online C programming

- https://www.programiz.com/c-programming/online-compiler/
- press 'run'
  - compile program
  - run program
  - (input &) output shown on the right
  - change the text and rerun

your program

input for your program &
output from your program



Mentimeter

- did you run Hello YourName?

printing squares: printing

main.c

```c
#include <stdio.h>
int main (void)
{
  printf("printing squares up to 3\n");
  printf("square=0\n");
  printf("square=1\n");
  printf("square=%d\n", 4);
}
```

Output

```
/tmp/eT304r4LXB.o
printing squares up to 3
square=0
square=1
square=4
```

- functions:
  - always start from the `main` function
  - `printf, scanf`
- function contains one or more statements
- each terminated with a semicolon ;
- function arguments: `printf(format string, arguments)`

- now we must change the program
  every time we want to have a different size
- that's not very useful

- how to generalise for any number of squares?

printing squares: variables

main.c

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  size = 3;
  printf("printing squares up to %d\n", size);
  i = 0; printf("square=%d\n", i*i);
  i = 1; printf("square=%d\n", i*i);
  i = 2; printf("square=%d\n", i*i);
}
```

Output

```
/tmp/eT304r4LXB.o
printing squares up to 3
square=0
square=1
square=4
```

- a variable declaration gives a name to a memory location
- assign a value in the variable with =
  this stores the value in the memory
  - `size = 3;`
  - we can do this multiple times
- look up the current value of the variable in memory when it is used in an expression
  - e.g. `i*i`

- now it's just longer
- still need to change the program

printing squares: variables

main.c

Run

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  size = 3;
  printf("printing squares up to %d\n", size);
  i = 0; printf("square=%d\n", i*i);
  i = 1; printf("square=%d\n", i*i);
  i = 2; printf("square=%d\n", i*i);
}
```

Output

Clear

```
/tmp/eT304r4LXB.o
printing squares up to 3
square=0
square=1
square=4
```

Why do we have variables?

- computer memory is a long list of locations (e.g. 16 MByte is 4194304 locations of 4 bytes each)
- without variables we would have to use & keep track of numerical addresses (0..4194303) instead of variable names such as i, size, ...
- if the program changes then the compiler may change the location of a variable
- variables allow clearer programs are quicker to write and easier to modify and maintain

printing squares: while loop

main.c

Run

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  size = 3;
  printf("printing squares up to %d\n", size);
  i = 0;
  while (i < size) {
    printf("square=%d\n", i*i);
    i = i +1;
  }
}
```

Output

```
/tmp/eT3O4r4LXB.o
printing squares up to 3
square=0
square=1
square=4
```

- we only write the `printf` statement only once
- even if the loop is executed thousands of times!

- now it's shorter
- but still need to change the program

printing squares

```
#include <stdio.h>
int main (void)
{
  int i, size;
  size = 3;
  printf("printing squares up to %d\n", size);
  i = 0;
  while (i < size) {
    printf("square=%d\n", i*i);
    i = i +1;
  }
}
// the loop is the same as:
i = 0;                       // store 0 in memory
printf("square=%d\n", i*i); // lookup 0 & multiply & print 0
i = i +1;                    // lookup star, add 1, store 1
printf("square=%d\n", i*i); // lookup 1 & multiply & print 1
i = i +1;                    // lookup star, add 1, store 2
printf("square=%d\n", i*i); // lookup 2 & multiply & print 2
i = i +1;
```

1. initialise variable i to 0
2. check if condition i<size is true
3. if yes, then execute the loop body
4. execute the iterator (i=i+1 ➔ i=1)

1. check if condition i<size is true
2. if yes, then execute the loop body
3. execute the iterator (i=i+1➔ i=2)

4. check if condition i<size is true
5. if yes, then execute the loop body
6. execute the iterator (i=i+1 ➔ i=3)

7. check if condition i<size is true
8. if no, exit the loop

- note that i equals 3 after the loop

printing squares: while loop

main.c

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  size = 3;
  printf("printing squares up to %d\n", size);
  i = 0;
  while (i < size) {
    printf("square=%d\n", i*i);
    i = i +1;
  }
}
```

Output

```
/tmp/eT3O4r4LXB.o
printing squares up to 3
square=0
square=1
square=4
```

Why do we have loops?

- to avoid duplicating repetitive code
- allow for a variable number of iterations without editing & recompiling the program (e.g. size)
- shorter programs are quicker to write, contain fewer errors, and are easier to maintain

printing sum

you can download this, to get started quicker: see
http://www.es.ele.tue.nl/~kgoossens/teaching/skc/

- change the program to print out the sum of the numbers 1 up to size
- for size == 3 the sum is 1+2 = 3
- for size == 1000 the sum is 1+2+...+999 = 499500

Mentimeter

- How did it go?

printing sum

main.c

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
  int i, size, sum;
  size = 1000;
  sum = 0;
  i = 1;
  while (i < size) {
      sum = sum + i;
      i = i + 1;
  }
  printf(" the sum up to %d is %d\n", size, sum);
}
```

Output

```
/tmp/apYyTf0Chh.o
the sum up to 1000 is 499500
```

- change the program to print out the sum of the numbers 1 up to size
- for size == 3 the sum is 1+2 = 3
- for size == 1000 the sum is 1+2+...+999 = 499500

printing squares: scanf

main.c

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  printf("up to what number? ");
  scanf("%d", &size);
  printf("printing squares up to %d\n", size);
  i = 0;
  while (i < size) {
    printf("square=%d\n", i*i);
    i = i +1;
  }
}
```

Output

```
/tmp/eT304r4LXB.o
up to what number? 10
printing squares up to 10
square=0
square=1
square=4
square=9
square=16
square=25
square=36
square=49
square=64
square=81
```

- we ask the user for input
- no need to change the program!

- you need to type in the number 10 (or another number)

printing squares: scanf

main.c Run

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  printf("up to what number? ");
  scanf("%d", &size);
  printf("printing squares up to %d\n", size);
  i = 0;
  while (i < size) {
    printf("square=%d\n", i*i);
    i = i +1;
  }
}
```

Output

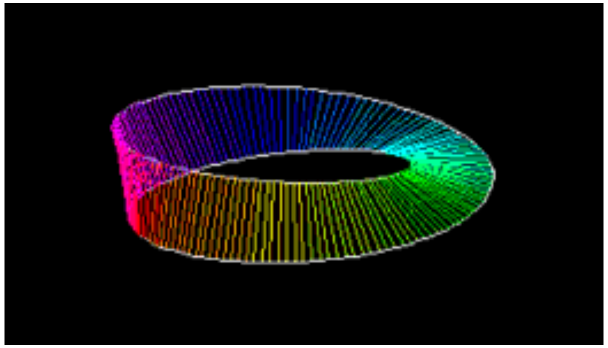
```
/tmp/eT304r4LXB.o
up to what number? 10
printing squares up to 10
square=0
square=1
square=4
square=9
square=16
square=25
square=36
square=49
square=64
square=81
```

Why do we have input (`scanf`) statements?

- one program be applied to different data every time it's run
- program can be interactive with user

Mentimeter

- what happens if you type in -10? why?



printing squares: if statements

main.c

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  printf("up to what number? ");
  scanf("%d", &size);
  if (size < 0) printf("Size must be positive\n");
  else {
   printf("printing squares up to %d\n", size);
    i = 0;
    while (i < size) {
      printf("square=%d\n", i*i);
      i = i +1;
    }
  }
}
```

Output

```
/tmp/eT3O4r4LXB.o
up to what number? -1
Size must be positive
```

- we need to check user input

printing squares: if statements

main.c

```c
#include <stdio.h>
int main (void)
{
  int i, size;
  printf("up to what number? ");
  scanf("%d", &size);
  if (size < 0) printf("Size must be positive\n");
  else {
   printf("printing squares up to %d\n", size);
    i = 0;
    while (i < size) {
      printf("square=%d\n", i*i);
      i = i +1;
    }
  }
}
```

Output

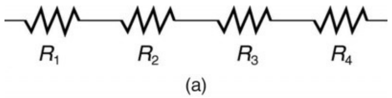
```
/tmp/eT3O4r4LXB.o
up to what number? -1
Size must be positive
```

Why do we have conditional (`if`) statements?

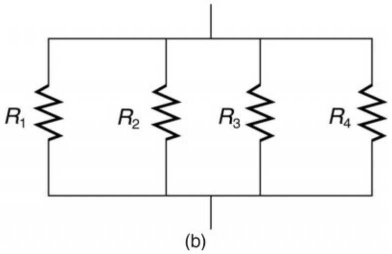
- program can react differently for different input data
- checking for invalid input
- dealing with exceptions, e.g. first and last iteration of a loop

simple EE problem

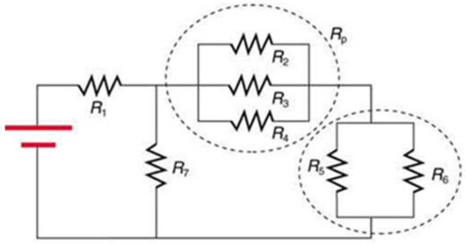
- consider an electrical circuit with resistors, which can be placed in
  - series



  - parallel



  - combinations

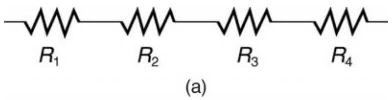


see: https://courses.lumenlearning.com/physics/chapter/21-1-resistors-in-series-and-parallel/

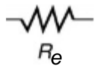
simple EE problem

- using Ohm's law any combination of resistors can be replaced by a single equivalent resistor

series:

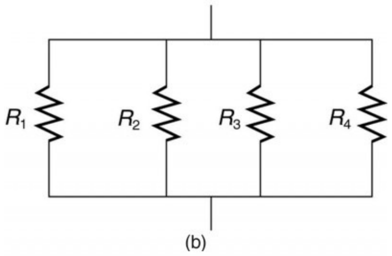


→

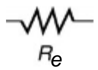


$R_e = R_1 + R_2 + R_3 + R_4$

parallel:



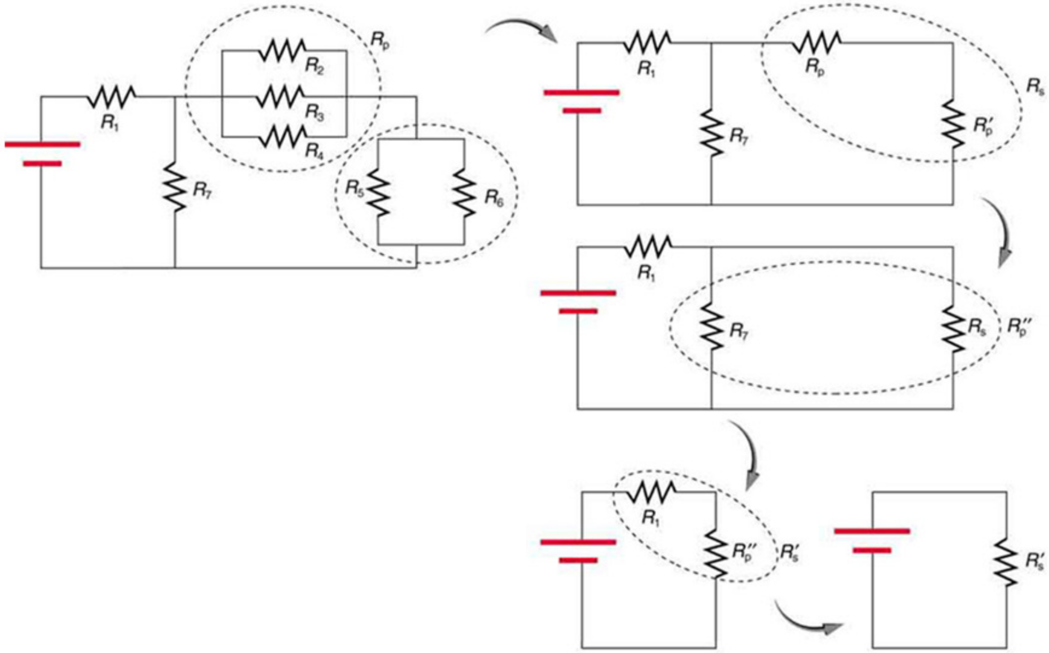
→



$1/R_e = 1/R_1 + 1/R_2 + 1/R_3 + /1R_4$

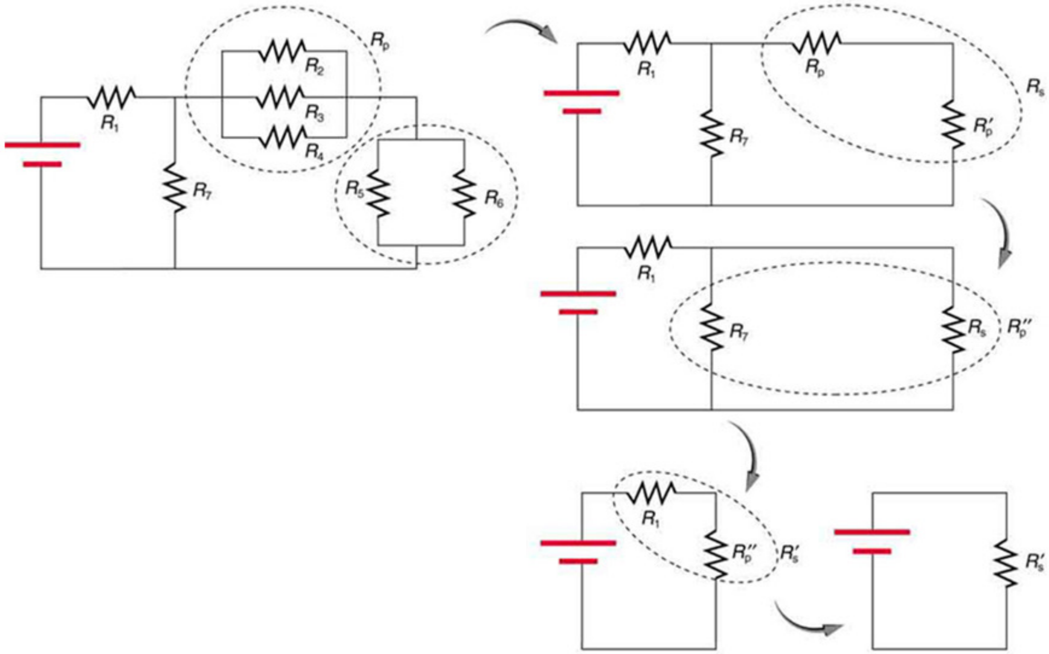
simple EE problem

- using Ohm's law any combination of resistors can be replaced by a single equivalent resistor



simple EE problem

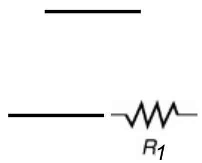
- we will write a program that computes the equivalent resistance
- as we add resistors (the arrows are reversed)



programming it

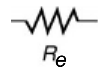
- write a program that
  - starts with a wire (resistance = 0)
  - then repeatedly asks for a resistor to be added either in series or parallel
  - computes the equivalent resistor

```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
```




```
requiv = 0
```

```
requiv = 0
rnew = 6
```



0

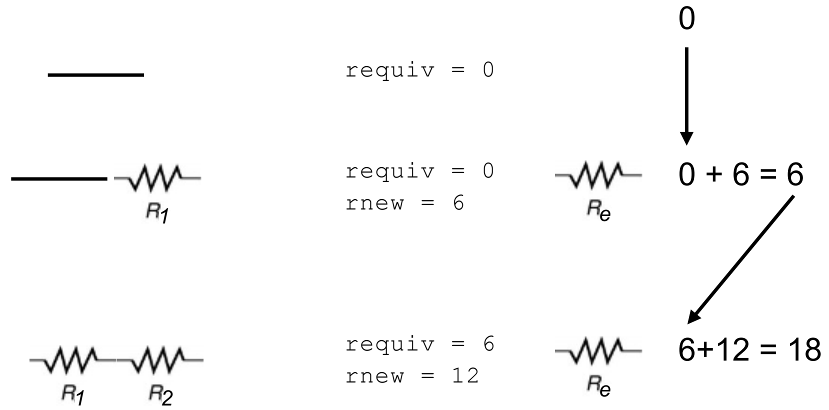
0 + 6 = 6

Electronic Systems

programming it

- write a program that
  - starts with a wire (resistance = 0)
  - then repeatedly asks for a resistor to be added either in series or parallel
  - computes the equivalent resistor

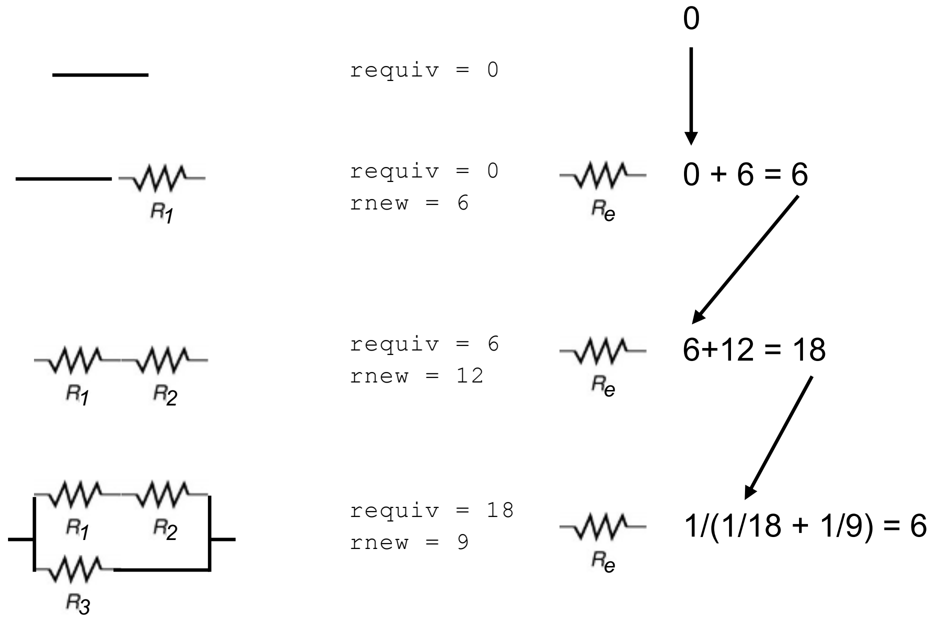
```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 12
Equivalent resistor is 18.000000
```



programming it

- write a program that
  - starts with a wire (resistance = 0)
  - then repeatedly asks for a resistor to be added either in series or parallel
  - computes the equivalent resistor

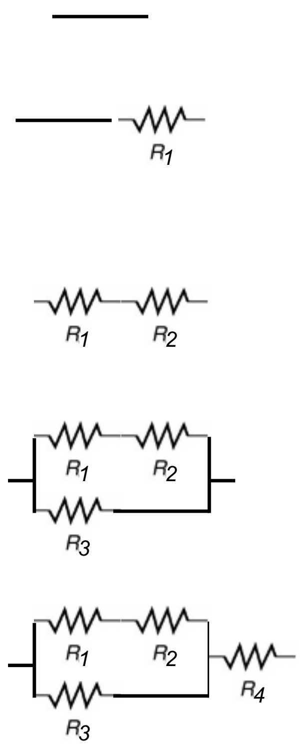
```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 12
Equivalent resistor is 18.000000
Series (0), Parallel (1), or Quit (2)? 1
Resistance? 9
Equivalent resistor is 6.000000
```



programming it

- write a program that
  - starts with a wire (resistance = 0)
  - then repeatedly asks for a resistor to be added either in series or parallel
  - computes the equivalent resistor

```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 12
Equivalent resistor is 18.000000
Series (0), Parallel (1), or Quit (2)? 1
Resistance? 9
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 3
Equivalent resistor is 9.000000
Series (0), Parallel (1), or Quit (2)? 2
Bye!
```




```
requiv = 0

requiv = 0
rnew = 6

requiv = 6
rnew = 12

requiv = 18
rnew = 9

requiv = 6
rnew = 3

requiv = 9
```

0

0 + 6 = 6

6+12 = 18

1/(1/18 + 1/9) = 6

6 + 3 = 9

Ohm's law


```
#include <stdio.h>
int main (void)
{
  float rnew, requiv = 0.0;
  int cmd = 0;
  while (cmd != 2) {
      printf("Equivalent resistor is %f\n", requiv);
      printf("Series (0), Parallel (1), or Quit (2)? ");
      scanf("%d", &cmd);
      if (cmd == 0 || cmd == 1) {
          printf("Resistance? ");
          scanf("%f", &rnew);
      }
      ... insert your code here ...
  }
  printf("Bye!\n");
}
```

floating point (real) number

```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 12
Equivalent resistor is 18.000000
Series (0), Parallel (1), or Quit (2)? 1
Resistance? 9
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 3
Equivalent resistor is 9.000000
Series (0), Parallel (1), or Quit (2)? 2
Bye!
```

you can download this, to get started quicker: see
https://kgoossens.estue.nl/docs/studiekeuzecheck/

Mentimeter

- did you know Ohm's law?
- is the assignment clear?

Mr. Resistance (and voltage and current)

too easy? compute PI

- use the Leibniz formula
- https://en.wikipedia.org/wiki/Leibniz_formula_for_pi

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

```
#include <math.h> // for M_PI=3.1415...
#include <stdio.h>
int main (void) {
  double mypi;
  ...
  // to print the currently computed value of mypi
  // and the difference with the real value M_PI
  printf("%30.28f %+30.27f\n", mypi, mypi-M_PI);
  ...
}
```

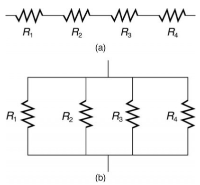
now let's program!

Mentimeter

- the problem was ...

Ohm's law


```
#include <stdio.h>
int main (void)
{
  float rnew, requiv = 0.0;
  int cmd = 0;
  while (cmd != 2) {
      printf("Equivalent resistor is %f\n", requiv);
      printf("Series (0), Parallel (1), or Quit (2)? ");
      scanf("%d", &cmd);
      if (cmd == 0 || cmd == 1) {
          printf("Resistance? ");
          scanf("%f", &rnew);
      }
      if (cmd == 0) requiv = requiv + rnew;
      if (cmd == 1) requiv = 1.0/(1.0/requiv + 1.0/rnew);
  }
  printf("Bye!\n");
}
```




```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 12
Equivalent resistor is 18.000000
Series (0), Parallel (1), or Quit (2)? 1
Resistance? 9
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 3
Equivalent resistor is 9.000000
Series (0), Parallel (1), or Quit (2)? 2
Bye!
```

- looks good, but this program contains a bug
- what happens when we immediately insert a resistor in parallel?

Ohm's law

```
#include <stdio.h>
int main (void)
{
  float rnew, requiv = 0.0;
  int cmd = 0;
  while (cmd != 2) {
      printf("Equivalent resistor is %f\n", requiv);
      printf("Series (0), Parallel (1), or Quit (2)? ");
      scanf("%d", &cmd);
      if (cmd == 0 || cmd == 1) {
          printf("Resistance? ");
          scanf("%f", &rnew);
      }
      if (cmd == 0) requiv = requiv + rnew;
      if (cmd == 1) {
          if (requiv == 0.0) requiv = rnew;
          else requiv = 1.0/(1.0/requiv + 1.0/rnew);
      }
  }
  printf("Bye!\n");
}
```

```
Equivalent resistor is 0.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 6
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 12
Equivalent resistor is 18.000000
Series (0), Parallel (1), or Quit (2)? 1
Resistance? 9
Equivalent resistor is 6.000000
Series (0), Parallel (1), or Quit (2)? 0
Resistance? 3
Equivalent resistor is 9.000000
Series (0), Parallel (1), or Quit (2)? 2
Bye!
```

- exception: the first resistor always replaces the wire
- (better) alternative: use `requiv = (requiv * rnew)/(requiv + rnew)`

Mentimeter

- conclusions

information

- see https://kgoossens.estue.nl/docs/studiekeuzecheck/
- for this presentation and some more information on C programming

- thank you for participating, and I hope to see you in September!