We stopped Teaching C

Meeting Embedded Berlin 2018

We

Wouter van Ooijen

wouter.vanooijen@hu.nl

Jorn Bunk

jorn.bunk@hu.nl

HOGESCHOOL UTRECHT

Topics

- Context: institute, curriculum, trends
- Why skip C
- Experience
- Conclusions, questions

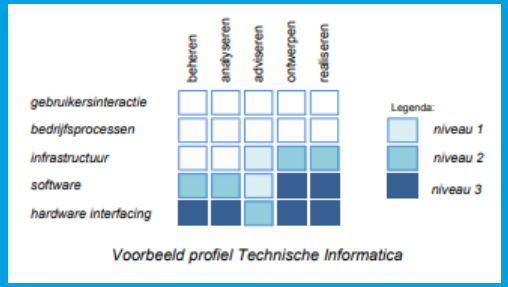
HOGESCHOOL UTRECHT

Dutch higher education

HOGESCHOOL UTRECHT

Universities
Universities of applied science (HBO, ~ polytechnic)
HBO Utrecht
institute HBO-ICT
specialization Computer Engineering (Technische Informatica)
~ 6 lecturers  

Our institute

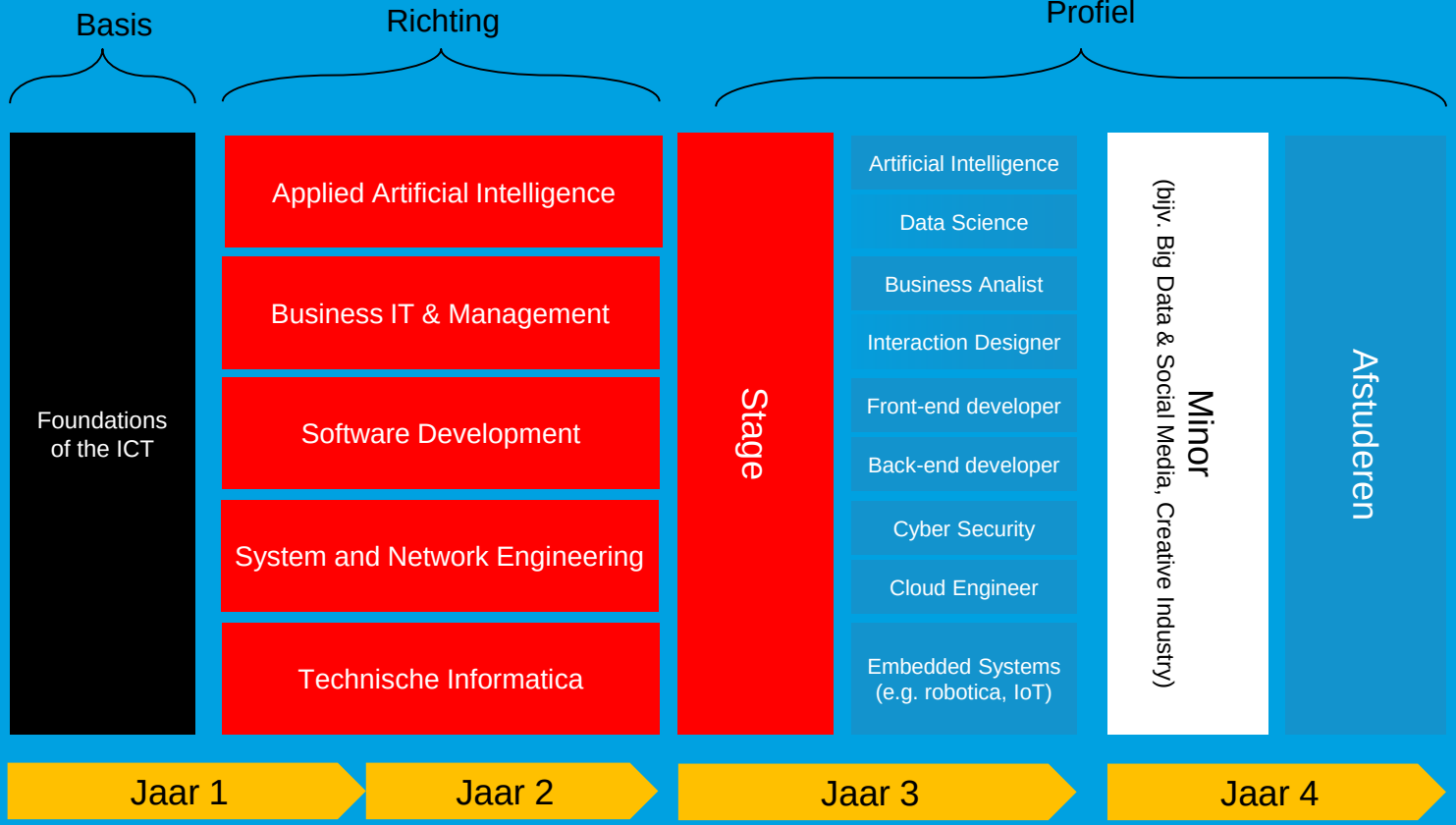


Voorbeeld profiel Technische Informatica

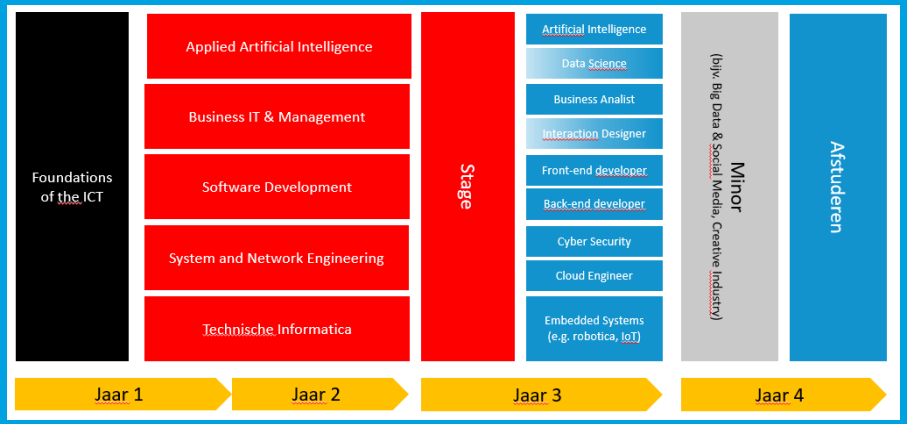
HBO-ICT, specializations:

- Applied Artificial Intelligence
- Buisiness Information Management
- Software & Information Engineering
- Systems & Network Engineering
- Technische Informatica (Computer Engineering)

Our institute



Our institute



4 y (4 * 60 = 240 EC)

- 0.5 y (30 EC) common introduction
- 1.5 y (90 EC) specialization (BIM, SNE, SIE, AAI, TI)
- 2.0 y (120 EC) internships, profile, minor

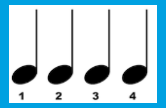
Computer Engineering

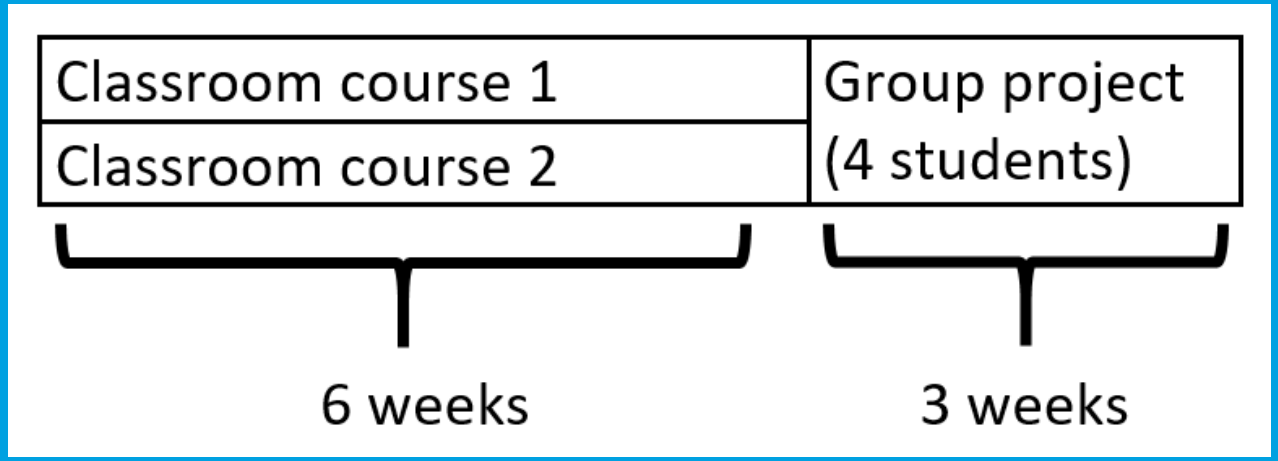
HOGESCHOOL UTRECHT

- ~ 60 students/y (2 or 3 classes of 20..30)
- Focus on close-to-the-hardware and resource-constrained:
  Embedded, micro-controllers, robotics, Linux
- Mostly C++, but also Python, Assembler (,C)

+ We work with physical stuff
- Technical work seems to be frowned upon (in our country)

Quarter rhythm





9 weeks

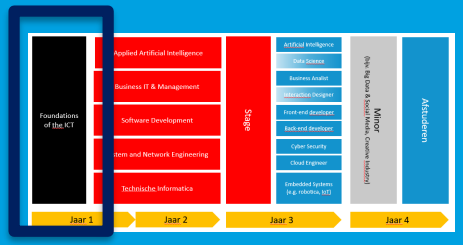
- 6 weeks: 2 classroom courses
- 3 weeks: 1 group project

Common first half-year

HOGESCHOOL UTRECHT

Q1

- Programming (Python)
- Computer systems & networks
- ICT and organization



Q2

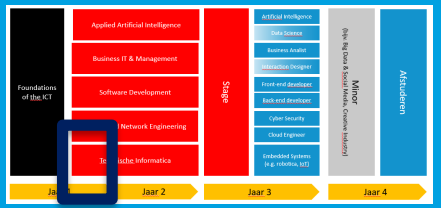
- Modelling (BPMN, use-cases, databases)
- ASK (Critical thinking, logics, statistics)
- Group project (collaboration, agile, professional skills)

Computer engineering – y1

HOGESCHOOL UTRECHT

Q3

- procedural programming (algorithmics, C++)
- operating systems (Linux, RaPi)
- Lego Robotics (Lego + BrickPi, C++)

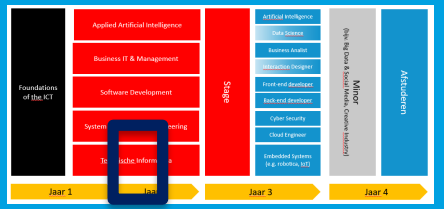


Q4

- OO programming and micro-controllers (classic C++, ArduinoDue)
- Digital and analog electronics, computer architecture
- 1st year final project (individual, ArduinoDue, C++)

Computer engineering – y2

HOGESCHOOL UTRECHT



Q5

- Low-level programming (assembly, C++, threading)
- Real-time modelling (UML & synchronization)
- Laser-Tag project (ArduinoDue, RTOS, modelling, waterfall, C++)

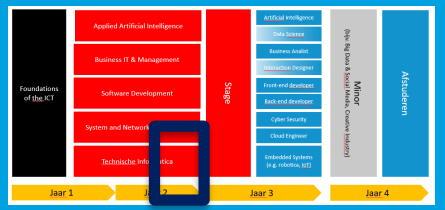
Q6

- High-level programming (C++, resources, STL, patterns, MSVS, SFML)
- Datastructures & algorithms (lists, trees, hashmaps; Python)
- Game project (SFML, PC/Linux/Mac, Agile/scrum, Fagan inspection, C++)

Computer engineering – y2

Q7

- MRB (sensors, actuators, control)
- Vision (image processing)
- Development department (applied research)
  - All students together, changing sub-teams
  - Proposals, team construction, work, dependencies
  - Continuous integration, tests, changing requirements
  - Self-organizing, scrum

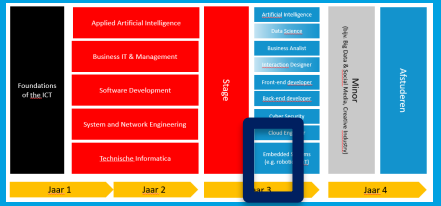


Q8

- Research
- Development department cont. – 15 EC total

Computer engineering – y3/4

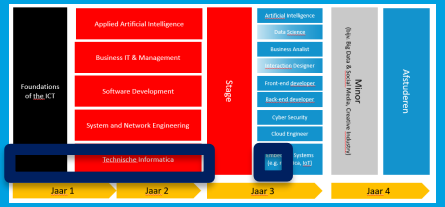
HOGESCHOOL UTRECHT



Q9 or later

- Advanced Programming
  Functional programming, introspection
  Python/C++ integration, test-strategy
- Applied AI
  neural networks, classifiers
- Inter-disciplinary project (20 EC)

Programming



HOGESCHOOL UTRECHT

Y	Q	Subjects	Language
1	1	Introduction	Python
	2		
	3	Basic algorithms, procedural	C ➔ C++
	4	OO, hardware abstraction	C++
2	5	Low level, asm, C/C++/asm interaction, templates	C++, assembler
	6	High level, STL, ownership,	C++
		Datastructures	Python
	7		
	8		
3	9	FP, introspection, Python-C++	Python, C++

Some ’recent’ trends

HOGESCHOOL UTRECHT

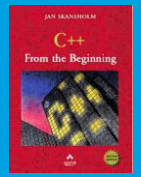
- More HBO-ICT integration
- First language (shared) Java ➔ Python
- One 2.5 EC C++ course (classic OO)
  ➔ 4 * 5 EC courses (each 50% C++): 4-fold increase
- Smallest hardware
  PIC16F630 (asm) ➔ 18F452 (C) ➔ ArduinoDue (C++)
- Direct HW register interfacing ➔ abstraction of the hardware

- Our own lab
- The students visit conferences

Course materials

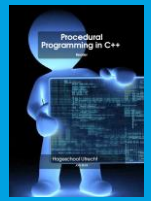
2005:

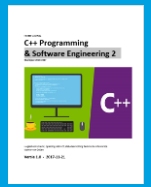
- Book by Skansholm (classic OO)
- Book ‘In zee met C’ (almost K&R C)



2018:

- Reader PPC
- Reader OOPC
- Reader CPSE1
- Reader CPSE2 (+ book)





Course materials

Why not use existing material?

- Material decides course content
- Academic ⇔ practical
- Often either for total beginner, or as 2nd language
- Not up-to-date with the latest standard
- English only (we prefer Dutch for y1)

Why skip C?

HOGESCHOOL UTRECHT

- Focus shift towards abstraction:
  hardware register interfacing ➔ hardware abstraction
- No subsequent C courses
- No C-only targets
- C++ has (slightly) less noise and clutter
- Avoid C ➔ C++ transition pain

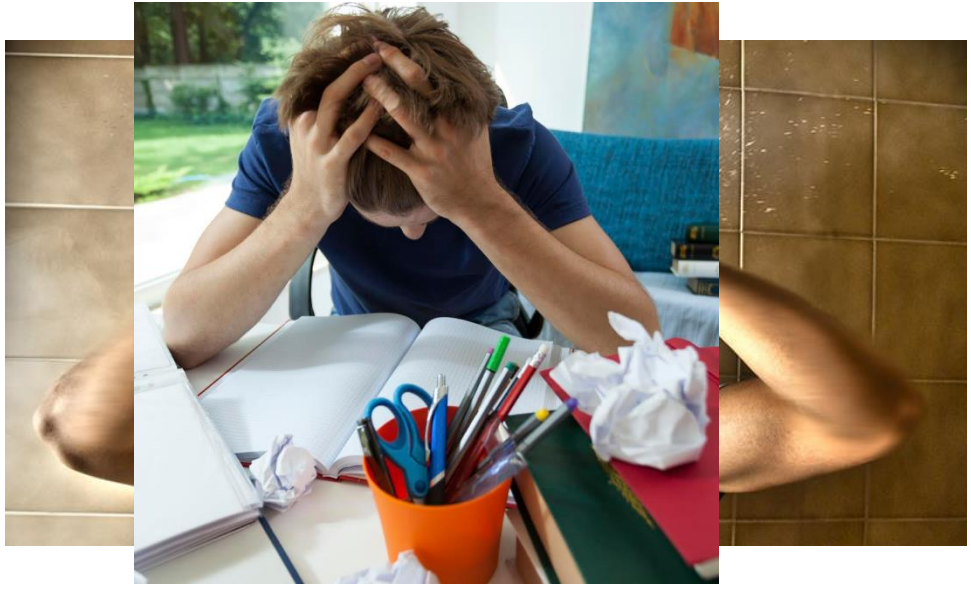
Cost: loose pure C programming

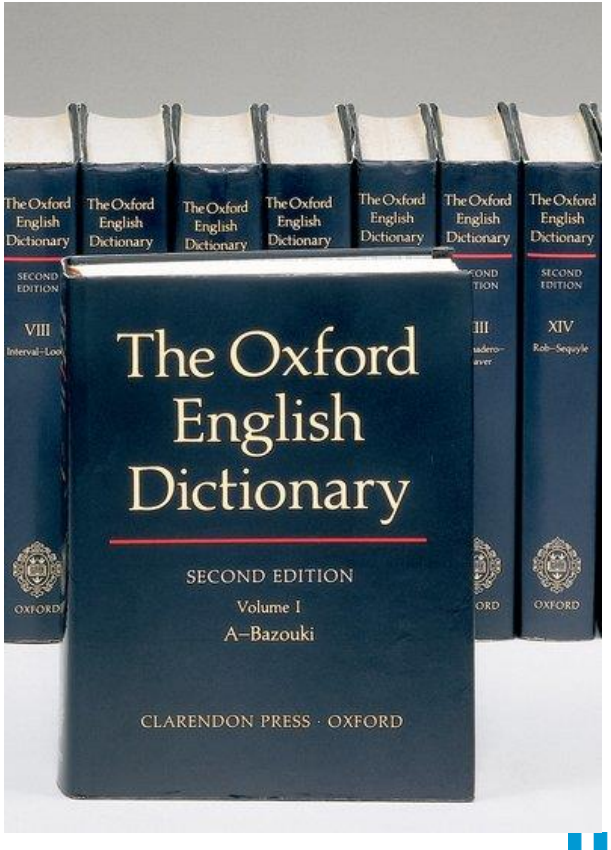
Why is programming so hard?

What to learn?

For the students there are two challenges:

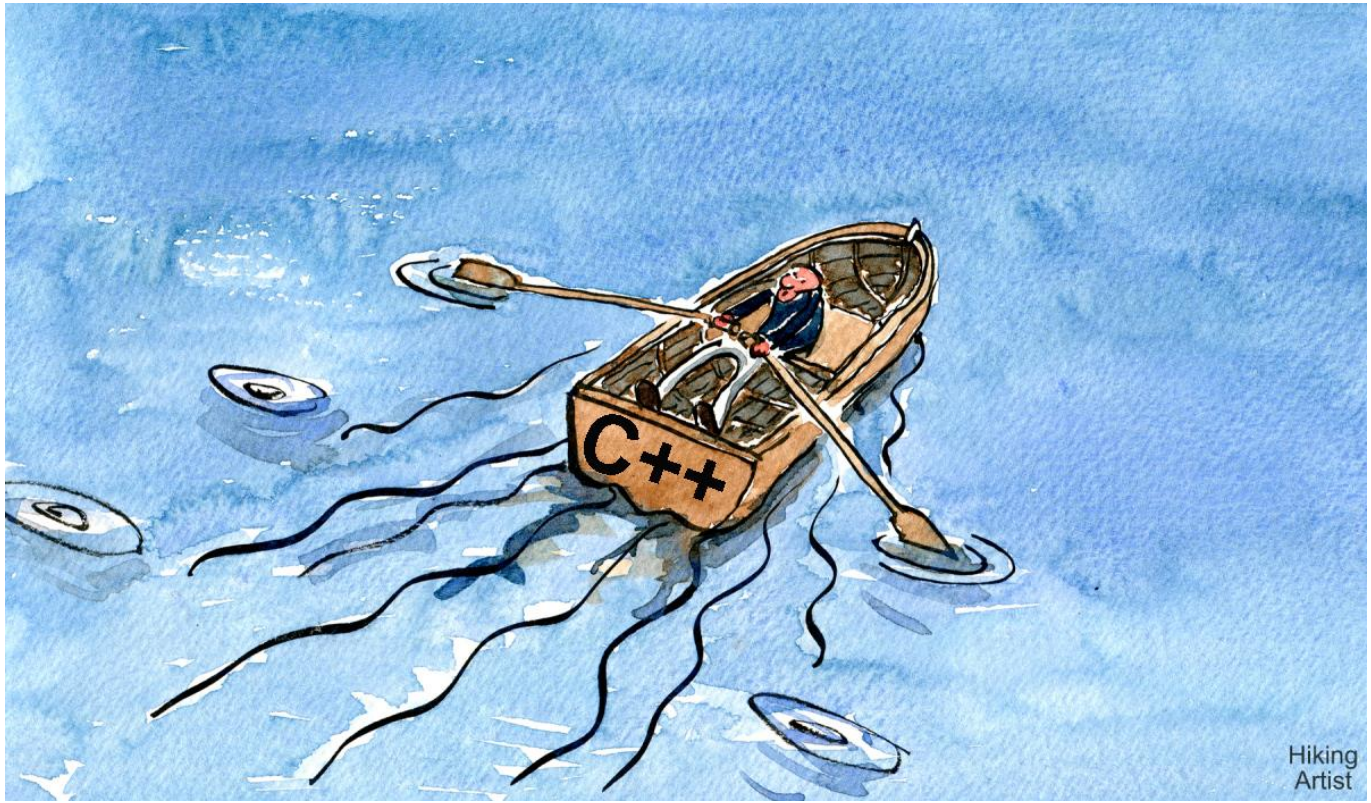
1. Syntax
2. (Procedural) Problem solving





Designing the new course

Designing the new course



Designing the course

We removed from the course:

- Switch
- Do-while
- Static variabeles
- Main function parameters (argc, argv)
- MACRO’s
- Arrays / c-string
- Pointers
- Printf() / scanf()
- And a lot of (for now) irrelevant details.

The old reader: integers

type	size (aantal bytes)	minimum	maximum	format specifier
int	4 (eis: >= 2)	-2147483648 (-2^31)	2147483647 (2^31-1)	%d
unsigned int	4 (eis: >= 2)	0	4294967295 (2^32-1)	%u
short	2 (eis: >= 2)	-32768 (-2^15)	32767 (2^15-1)	%hd
unsigned short	2 (eis: >= 2)	0	65535 (2^16-1)	%hu
long	4 (eis: >= 4)	-2147483648 (-2^31)	2147483647 (2^31-1)	%ld
unsigned long	4 (eis: >= 4)	0	4294967295 (2^32-1)	%lu
long long	8 (eis: >= 8)	-9223372036854775808 (-2^63)	9223372036854775807 (2^63-1)	%lld
unsigned long long	8 (eis: >= 8)	0	18446744073709551615 (2^64-1)	%llu

The new reader: integers (1/2)

```
#include <iostream>
using namespace std;

int main(){
    int number = 6;
    number = number + 1;
    number = number * 2;
     cout << "number: " << number << "\n";
     // number: 14

     int x = 8 - 4;
    cout << "x: " << x << "\n";
     // x: 4
}
```

The new reader: integers (2/2)

```
#include <iostream>
using namespace std;

int main(){
    int y = 4.9;
     cout << "y: " << y << "\n";
     // y: 4

    int z = 21;
     int result = z / 6;
     cout << "result: " << result << "\n";
     // result: 3
}
```

The old reader: output

```
int b = 14;
printf("The square of %d is %d", b, b*b);

printf("%#x %#o", 90, 420);
```

decimaal	%d
hexadecimaal	%x , met prefix 0x : %#x
octaal	%o , met prefix 0 : %#o
long hexadecimaal	%lx , met prefix 0x : %#lx
long long hexadecimaal	%llx , met prefix 0x : %#llx

The new reader: output

```
#include <iostream>
using namespace std;

int main(){
    char letter = 'd';

     cout << "letter: " << letter << "\n";
    // letter: d

     char result2 = 'c' - 'a';
    cout << "result2: " << int(result2) << "\n";
    // result2: 2
}
```

The old reader: input

```
int day, month, year;
int n;

printf("Datum in format: dd/mm/yyyy: ");
n = scanf("%d/%d/%d", &day, &month, &year);
printf("n: %d\n", n);
printf("The date is: %02d/%02d/%4d\n", day, month, year);
```

The new reader: input

```
#include <iostream>
#include <string>
using namespace std;

int main(){
    float price;
    int quantity;
    cout << "Enter price and quantity: ";
    cin >> price >> quantity;

   float total_price = price * quantity;
    cout << "Total price: " << total_price << "\n";
}
```

The new reader: input

```
#include <iostream>
#include <string>
using namespace std;

int main(){
    string sentence;
    cout << "Enter a sentence: \n";
    getline(cin, sentence);

   cout << sentence;
}
```

C-array

```
//get all elements bigger than x from array:
int BiggerThan(int src[], int src_size, int dest[], int x){
    int j = 0;
    for(int i=0; i < src_size; i++){
        if(src_size > x){
            dest[j] = src_size[i];
            j++;
        }
    }
    return j;
}
```

Vector

```
//get all elements bigger than x from vector:
vector<int> BiggerThan(vector<int> v, x){
    vector<int> result;
    for(unsigned int i=0; i<v.size(); i++){
        if(v[i] > x){
            result.push_back(v[i]);
        }
    }
    return result;
}
```

Other changes:

- Learning by example, not by long explanations
- Reader only contains correct code.
- Exercises:
  - Other exercises types (e.g. reading exercises)
  - Each exercise has one learning goal.

Remaining challenges

- Better (and more) exercises
- Vector and recursion

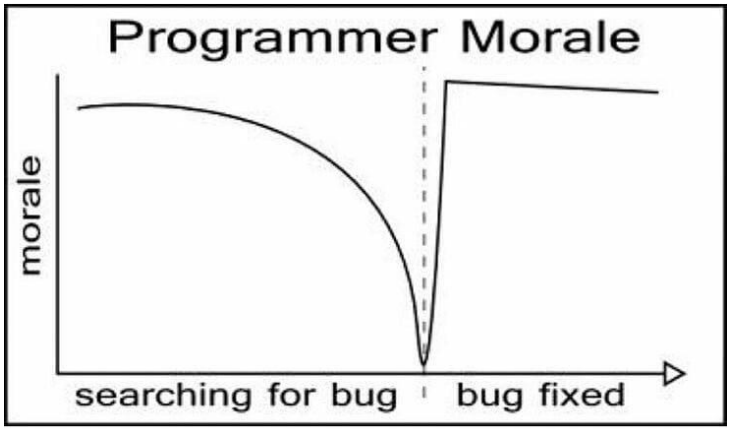
Vector and Recursion

```
int sum(const vector<int> & numbers, int size){
    if (size <= 0)
        return 0;
    return numbers[size-1] + sum(numbers, size-1);
}
```

Results:

Students became better in
- procedural program solving.
- debugging the logic in code.

Students
- are more confident.
- have more fun.



No transition pain in first week next C++ course.

We stopped Teaching C

- So far the effect seems to be positive (1y, 60st)
- Better start with the better-C subset than C-then-C++
- We don’t cater for all (C is for electro?)

Questions?

HOGESCHOOL UTRECHT

We have a few…

HOGESCHOOL UTRECHT

- What do you (industry) need?
  - C, C++, both, something else, everything
  - Specialists, generalists, profssional skills
  - Electronics, HW interfacing, SW
  - Specific experience (frameworks, libraries, tools, IDEs, targets)
- Is there a future for hard-technical work in western europe?