NOVEMBER 11TH 2020

ELEMENTARY PROGRAMMING

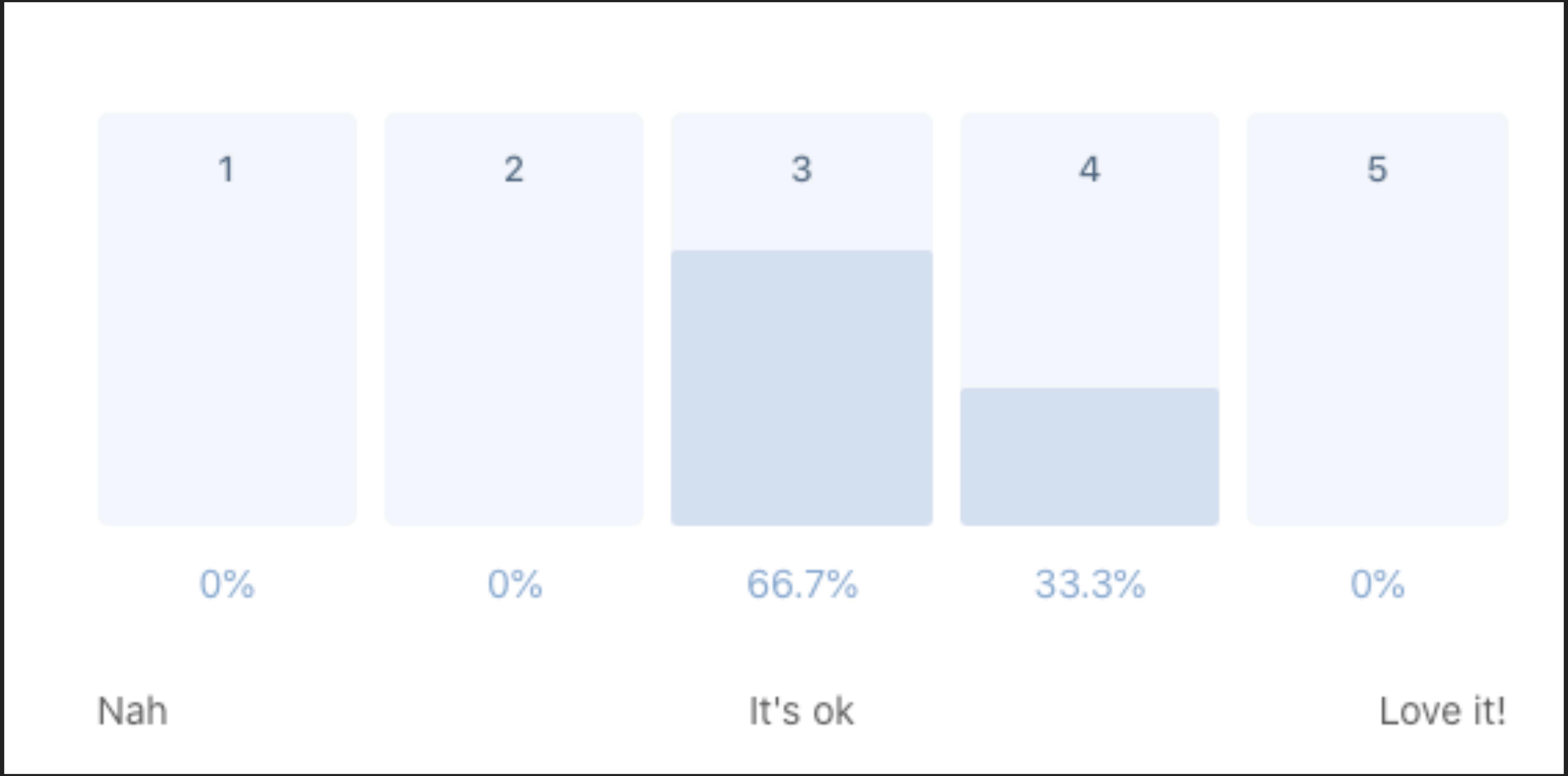
SOME COVID BEST PRACTICES BEFORE WE START

- If you fill ill, go home
- Keep your distance to others
- Wash or sanitise your hands
- Disinfect table and chair
- Respect guidelines and restrictions

REMEMBER TO BOOK YOUR SPOT TO DISCUSS THE SECOND ASSIGNMENT

- If you didn't receive my email please tell me I will resend the links
- You need to book an appointment otherwise no evaluation
  - Emanuele: **https://calendly.com/dierre/10min**
  - Alland: **https://calendly.com/a-kareem1991/02318_evaluering_1**
  - Patrick: **https://calendly.com/02318_opgave_eval_ph/10-min-eval**
  - Freja: **https://calendly.com/s200544/10-mins-samtale-om-aflevering**

FEEDBACK CHECK



NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog11

LOW LEVEL PROGRAMMING

- What we studied until now it’s helpful for most programs we want to make
- Some programs need to control the flow at bit level
- Example of programs that could need it are: system programs (OS), encryption, graphics, programs where you need to save space

LOW LEVEL PROGRAMMING

- What we studied until now it's helpful for most programs we want to make
- Some programs need to control the flow at bit level
- Example of programs that could need it are: system programs (OS), encryption, graphics, programs where you need to save space

LOW LEVEL PROGRAMMING

- Some of the techniques we will see depends on a deep knowledge on how memory is stored
- Memory is stored differently in different architectures
- Using this techniques will make probably your program non portable
- Avoid them if you can (to ease your life in maintaining your software)

BITWISE OPERATORS

- C has six bitwise operators
  - 2 shift operators
  - *complement*, *and*, *or* and *xor*
- Bitwise operators helps you deal with integers at bit level

BITWISE SHIFT OPERATORS

- With a shift operator we can transform the binary representation of an integer by shifting bits left or right
- The operators are `<<`(left) and `>>`(right)
- The operands can be of any integer type (including `char`)
- For portability, it's best to perform shifts only on `unsigned` numbers

BITWISE SHIFT OPERATORS

```
unsigned short i, j;
i = 13;       // (binary 00000000000001101)
j = i << 2;   // (binary 00000000000110100)
j = i >> 2;   // (binary 00000000000000011)
```

13

52

3

BITWISE SHIFT OPERATORS PRECEDENCE

bitwise shift operators have lower precedence

```
i << 2 + 1  →  i << (2 + 1)
```

REMAINING BITWISE OPERATORS

Symbol	Meaning
~	bitwise complement
&	bitwise and
^	bitwise exclusive or
|	bitwise inclusive or

Priority

Highest

Lowest

BITWISE COMPLEMENT

```
i = 21;     // (binary 0000000000010101)
k = ~i;     // (binary 1111111111101010)
```

Pro Tip: for portability reason you can setup all bits to 1, by using `~0`

BITWISE AND

```
j = 56;     // (binary 0000000000111000)
i = 21;     // (binary 0000000000010101)
k = i & j;  // (binary 0000000000010000)
```

BITWISE EXCLUSIVE OR

```
j = 56;     // (binary 0000000000111000)
i = 21;     // (binary 0000000000010101)
k = i ^ j;  // (binary 0000000000101101)
```

BITWISE INCLUSIVE OR

```
j = 56;     // (binary 0000000000111000)
i = 21;     // (binary 0000000000010101)
k = i | j;  // (binary 0000000000111101)
```

USING BITWISE OPERATORS TO ACCESS BITS

- When we do low level programming we want to store information inside each bit
- Using bitwise operators we can extract or modify data stored in a small number of bits
- Let’s some common operation on single bit

SET A SINGLE BIT (A MASK)

- Let's assume we are using a 16-bit unsigned short

```
i = 0x0000; // hex for 0000000000000000
j = 0x0010; // hex for 0000000000010000
i = i | j;  // hex for 0000000000010000
```

We can use this logic when we want to setup permissions on something each bit is a type of permission. You can write as well `i |= j`

CLEARING A BIT

- Let’s clean bit 4

```
j     = 0x0010; // hex for 0000000000010000
notJ = ~j;      // hex for 1111111111101111
i     = 0x00ff; // hex for 0000000011111111
i = i & notJ;   // hex for 0000000011101111
```

EXAMPLE (1)

```c
#define BLUE  1 // 0000000000000001
#define GREEN 2 // 0000000000000010
#define RED   4 // 0000000000000100
int main(void) {
    unsigned i = 0;
    i |= BLUE;        // sets BLUE bit
    i &= ~BLUE;       // clears BLUE bit
    if(i & BLUE){}    // tests if BLUE is on
}
```

EXAMPLE (2)

```
#define BLUE  1 // 0000000000000001
#define GREEN 2 // 0000000000000010
#define RED   4 // 0000000000000100
int main(void) {
    unsigned i = 0;
    i |= (BLUE | GREEN);
    i &= ~(BLUE | GREEN);
    if(i & (BLUE | GREEN)){}
}
```

TESTING

- Testing is part of the software development lifecycle
- It's about verifying that all the conditions for the software to work are respected
- It's how you show that the software is working correctly

TESTING

- There are two main ways of testing a software:
  - Manual
  - Automatic

MANUAL TESTING

- It involves manually making all the checks by running the code with each significative input
- You need to do it after every change
- It's time consuming
- It's hugely error prone

AUTOMATIC TESTING

- It involves writing code the tests your expectations
- The software will do it for you
- The only time required is when your write the test
- Compared to manual testing is basically not error prone (you still need to write the correct expectations)

AUTOMATIC TESTING

- In real projects what you do is automatic testing
  - it's faster
  - less error prone
  - it's expected that you guarantee code quality in a measurable way

AUTOMATIC TESTING

- To do automatic testing generally we use a testing framework
- There are many testing framework on the market
- In general, though, you just need a way to run your expectations

AUTOMATIC TESTING

- We will use a framework suggested for microcontrollers
- It is contained in one header file
- It has low footprint
- The framework is called MinUnit (http://www.jera.com/techinfo/jtns/jtn002.html)

MINUNIT.H

```
#define mu_assert(message, test) do { if (!(test)) return message; } while  (0)
#define mu_run_test(test) do { char *message = test(); tests_run++;  \
                               if (message) return message; } while (0)
extern int tests_run;
```

MU_ASSERT

```
#define mu_assert(message, test)
    do {
        if (!(test))
            return message;
    } while (0)
```

MU_RUN_TEST

```
#define mu_run_test(test)
    do {
        char *message = test();
        tests_run++;
        if (message)
            return message;
    } while (0)
```

RUN_ALL_TESTS

```
int run_all_tests(char *(*all_tests)(void)) {
    char *result = all_tests();
    if (result != 0) {
        printf("%s\n", result);
    } else {
        printf("ALL TESTS PASSED\n");
    }
    printf("Tests run: %d\n", tests_run);

    return result != 0;
}
```

EXAMPLE

```c
#include <stdio.h>
#include <string.h>
#include “minunit.h"
#include “caesar.c"
int tests_run = 0;
static char * test_ac1() {
    char cryptedtext[100];
    int   ret         = caesar(1, "hello", cryptedtext);
    mu_assert("error, hello is not ifmmp", ret == 0);
    mu_assert(cryptedtext, strcmp(cryptedtext, "ifmmp") == 0);
    return 0;
}
static char * all_tests() {
    mu_run_test(test_ac1);
    return 0;
}
int main(int argc, char **argv) {
    return run_all_tests(all_tests);
}
```

PASSING ARGUMENTS TO A C PROGRAM

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Number of parameters is %d\n", argc);
    printf("The program name is %s\n", argv[0]);
    if (argc == 2) {
        printf("The argument supplied is %s\n", argv[1]);
    } else if (argc > 2) {
        printf("Too many arguments supplied, I only accept one.\n");
    } else {
        printf("Error: one argument expected.\n”);
    }
    printf("\nThis is all that was passed:\n”);
    for(int i = 0; i < argc; i++) {
      printf(" - %s\n", argv[i]);
    }
}
```

NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog11

SOME COVID BEST PRACTICES BEFORE WE LEAVE

- Disinfect table and chair
- Maintain your distance to others
- Wash or sanitise your hands
- Respect guidelines and restrictions outside