NOVEMBER 11TH 2020

ELEMENTARY PROGRAMMING

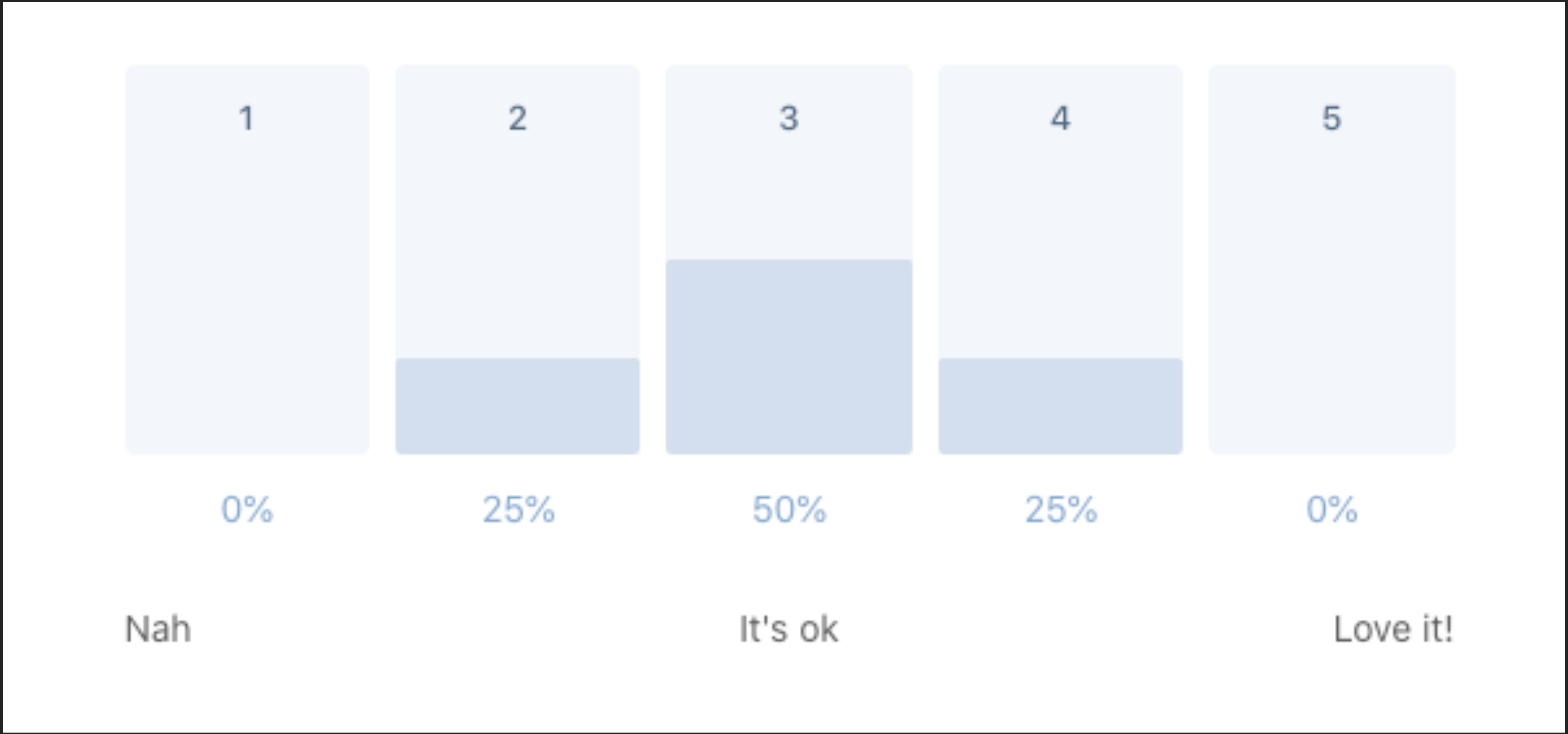
SOME COVID BEST PRACTICES BEFORE WE START

- 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 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/elemprog10

SECOND CODING ASSIGNMENT

- Opens at 10AM, November 4th 2020
- Closes at 8AM, November 18th 2020
- Here's the link: https://github.com/invasionofsmallcubes/elementary-programming-dtu/blob/master/assignments/assignment02/ASSIGNMENT.MD

THE PREPROCESSOR

- The preprocessor edits the program just before compilation
- The preprocessor changes the source code before compilation
- You need to be careful when you use the preprocessor because it could be you don’t spot the bugs right away

DIRECTIVES

- The directives are handled by the preprocessor
- There are many type of directives, we will look into three:
  - Conditional directives
  - Macro directives
  - Include directives

MACRO

- `#define` is a macro
- A macro is a name that represent something else:
  - A constant
  - An expression

MACRO WITH CONSTANT

- `#define TRUE 1`
- `#define FALSE 0`
- Wherever in the case one of this two words are used, the preprocessor will substitute their name with the respective values

MACRO WITH A FUNCTION

- `#define identifier(x1,x2,…,xn) replacement-list`
- We can actually declare a function using a macro

MACRO WITH A FUNCTION

- `#define MAX(x,y) ((x)>(y)?(x):(y))`
- This function tells me who's bigger between x and y
- As you can see there is not type defined

MACRO WITH A FUNCTION

- They are a generic function
- You can use MAX wherever > is handled by the type referred by x and y
- As you can see there is not type defined

MACRO WITH A FUNCTION

- In general macros with functions generate a larger compiled code
- You cannot use pointers

MACRO WITHIN A MACRO

- You can use a macro within another macro:
  - `#define PI 3.14159`
  - `#define TWO_PI (2*PI)`

C PROVIDES FOR YOU SOME PREDEFINED MACROS

- `__LINE__` line number of file being compiled
- `__FILE__` name of the file being compiled
- `__DATE__` date of the compilation
- `__TIME__` time of the compilation
- `__STDC__` if compiler conform with C standard (C89 or C99)

YOU CAN REMOVE A MACRO

- `#undef` removes a defined macro
- `#define YEARS_OLD 12`
- `#undef YEARS_OLD`

CONDITIONAL DIRECTIVES

- Macros by themselves don’t mean so much
- With conditional directives these will probably make more sense
- They are `#if`, `#ifdef`, `#ifndef`, `#elif` , `#else`

USAGE OF #IF

```
#define DEBUG 1
int i = 0;
#if DEBUG
  printf("Value of i: %d\n", i);
#endif
```

boolean check

USAGE OF #IFDEF

```
#ifdef WIN32
 ...
#endif
#ifdef MAC_OS
 ...
#endif
#ifdef LINUX
 ...
#endif
```

USAGE OF #IF AND DEFINED()

```
#define DEBUG 1

#if defined(WIN32) && DEBUG
 ...
#endif
#if defined(MAC_OS)
 ...
#endif
#if defined(LINUX)
 ...
#endif
```

CONDITIONAL DEFINITION

```
#ifndef BUFFER_SIZE
#define BUFFER_SIZE 256
#endif
```

CONDITIONAL DEFINITION

- Checking that something is already defined is needed because of the nature of the preprocessor
- Sometimes we could define the same macro in different files and we can only have one working at every given time
- For example we could pass the BUFFER_SIZE from the last example from outside this way `gcc -DBUFFER_SIZE=256 myfile.c -o myfile`

WRITING LARGE PROGRAMS

- When we write large programs it's better to split a single source code in more than one file.
- We want to organise our code by functionality related to each other.
- Let’s do this with an example.

WRITING LARGE PROGRAMS

- When we write large programs it's better to split a single source code in more than one file.
- We want to organise our code by functionality related to each other.
- Let’s do this with an example.

LET'S CRYPT ASSIGNMENT

- String normalisation
- String obfuscation
- Combine everything

LET'S CRYPT ASSIGNMENT

- String normalisation
- String obfuscation
- Combine everything

HEADERS

```
// stringNormalization.h
#ifndef STRING_NORMALIZATION_H
#define STRING_NORMALIZATION_H
const char * stringNormalization(char * input);
#endif
```

We use headers to define our signatures for the functions

```
// stringObfuscation.h
#ifndef STRING_OBFUSCATION_H
#define STRING_OBFUSCATION_H
const char * stringObfuscation(char * input);
#endif
```

This is what the user of our libraries will see, not the actual implementation

SOURCE FILES

```c
// stringNormalization.c
#include "stringNormalization.h"
const char * stringNormalization(char * input) {
  return "stringNormalization\n";
}
```

We #include the headers in the source code and we actually do the implementation

```c
// stringObfuscation.c
#include "stringObfuscation.h"
const char * stringObfuscation(char * input) {
  return "stringObfuscation\n";
}
```

This will be compiled in object code

MAIN SOURCE FILE

```c
// main.c
#include <stdio.h>
#include "stringNormalization.h"
#include "stringObfuscation.h"
int main(void) {
    const char *strNorm = stringNormalization("hello");
    printf("%s", strNorm);
    const char *strObs = stringObfuscation("hello");
    printf("%s", strObs);
}
```

We #include the headers in the main file and we don't care about the actual implementation, we use the interfaces as defined by the headers

FINAL RESULT

```
clang stringNormalization.c stringObfuscation.c main.c -o main.out
```

- We give to the compiler (clang or gcc) the list of files to compile
- The compiler will get the stringNormalization.c and stringObfuscation.c and create object code for them, then, they will be linked to the main.c using the headers

LINKING

- Linking is the process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory and executed.
- Linking can be performed at compile time, when the source code is translated into machine code, at load time, when the program is loaded into memory and executed by the loader, and even at run time, by application programs.
- On early computer systems, linking was performed manually.
- On modern systems, linking is performed automatically by programs called linkers

HOW TO USE #INCLUDE

- `#include` is a directive that includes the content of a file in the current compiled code, like `#include <stdio.h>`
- Whenever we split in files we need to have a way to import them to actually be used. We use `#include`

HOW TO USE #INCLUDE

- If you use `#include <filename>` you are searching your system directory (for example on Linux is `/usr/include`)
- if you use `#include "filename"` you are searching current directory.

ANOTHER EXAMPLE WITH #INCLUDE

```
#if defined(IA32)
  #define CPU_FILE "ia32.h"
#elif defined(IA64)
  #define CPU_FILE "ia64.h"
#elif defined(AMD64)
  #define CPU_FILE "amd64.h"
#endif
#include CPU_FILE
```

VARIABLE MODIFIERS - STATIC

- When you declare a variable `static` it will exist for the entire duration of the execution of the program
- If you declare it `static` inside a function, the variable will be visible only inside the block but will be stored outside the call stack

FUNCTION MODIFIERS - STATIC

- A function can be `static`
- `static` means a function can only be called from the same file where it's declared
- It's a good practice to use `static` with functions because you keep the namespace clean and avoid unnecessary exposure.

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/elemprog10

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