Breaking Things in C

Part 3: The Breakening

https://github.com/TheRustyStorm/BreakingThingsInC

Peter Zdankin, 27.10.21

Thats me!

Peter Zdankin

- Ph.D. Student at Prof. Weis
- Worked a lot with C and Embedded Development at Prof. Schiele (He’s amazing!)
- Done almost every C error in existence!
- Did a lot of teaching since 2014
- Here to answer questions!

ASK ME QUESTIONS !

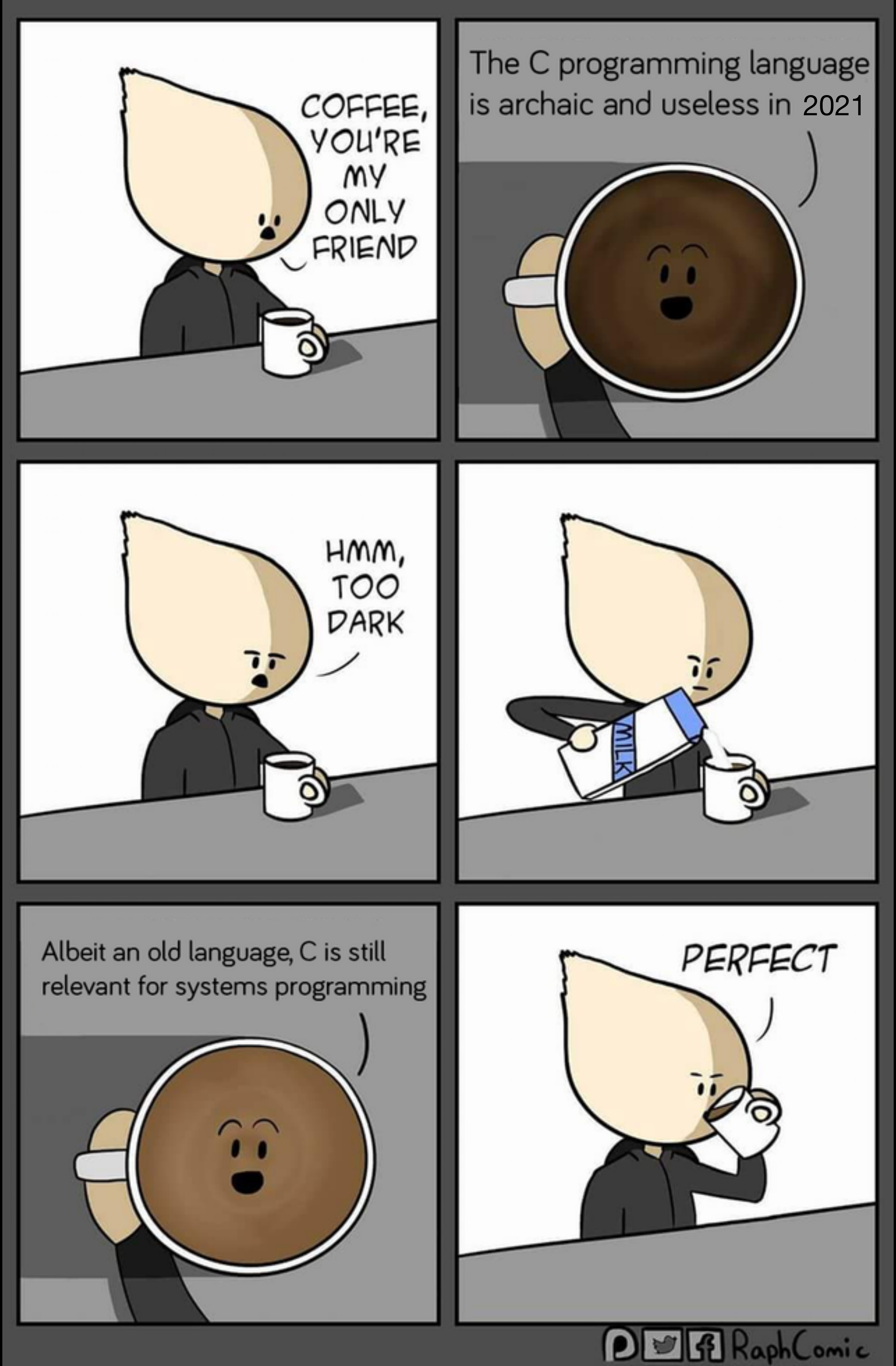
100%

Of this course is for fun and education

Week 1

Week 1

- History lesson
- Basics
- Compiler
- Editor


COFFEE, YOU'RE MY ONLY FRIEND
The C programming language is archaic and useless in 2021
HMM, TOO DARK
MILK
Albeit an old language, C is still relevant for systems programming
PERFECT
RaphComic

History yielded many programming languages

- Many vanished after some years of popularity
- Some remain in very obscure places (e.g. FORTRAN)
- Languages either evolve or go extinct
- What we use today could be dead in some years
- It has happened before

2014 - 2021

NOW

C++14 Perl 6 C++17 Fortran 2018 C++20

2010 - 2014

Rust Dart C++11 Kotlin Elm Elixir Julia Swift

2000 - 2010

C# D Scratch Scala C++03 Groovy F# Closure Nim Go

1990 - 2000

J Haskell Python Brainfuck Lua Java PHP Ruby Javascript

1980 - 1990

C
with classes

Objective C

C++

Bash

1970 - 1980

C Prolog Matlab

1960 - 1970

1950 - 1960

Fortran

COBOL

How can a language be fast?

What can you do that your program runs fast?

C

- Absolute efficiency
- High Level Assembler
- Almost nothing is made automatically
- Much behavior is undefined / compiler dependent behavior
- Errors are subtle and there are no exceptions. Unless your operating system kills the process, it continues running

Why bother?

- Fast
- Really fast
- As fast as you can possibly get without using assembler
- If nothing is done automatically, you define everything that happens -> low memory overhead
- For some use cases only option, e.g. Arduino

What will you learn?

- How to use C for practical problems
- How to debug archaic languages
- Writing fast and minimal code
- Good understanding of stack/heap/memory

Why do I use my free time to teach this, if there is a course that gives actual credits?

- Teach you how to do C in practice, asides lecture
- Show you how to be productive with C
- Explain how to find and fix bugs in C, which is ridiculously difficult
- Clean coding and TDD in practice

What can you do afterwards?

- Program on embedded devices like a pro
- Learn C++
- Tell all your friends that C is the best language ever

Getting started

- Download Visual Studio Code
- Only for Windows users: Download and install a C compiler (Mingw-64)
- Download Zeal/Dash and the C docset to look up the documentation

Hello World!

- Save file as test.c
- gcc test.c
- ./a.out

```
#include <stdio.h>

int main(void){
  printf(„Hello World!“);
  return 0;
}
```

Hello World!

- Save file as test.c
- gcc test.c
- ./a.out
- Output looks weird
- Hint: \n indicates a new line

```
#include <stdio.h>

int main(void){
  printf(„Hello World!“);
  printf(„Hello World!“);
  return 0;
}
```

Printf

- Printf is often used for formatted output
- E.g. we want to print the content of an int

-
  ```
  printf(„%i“, someInt);
  printf(„%c“, someChar);
  printf(„%s“, someString);
  printf(„%i + %i“, int1, int2);
  ```

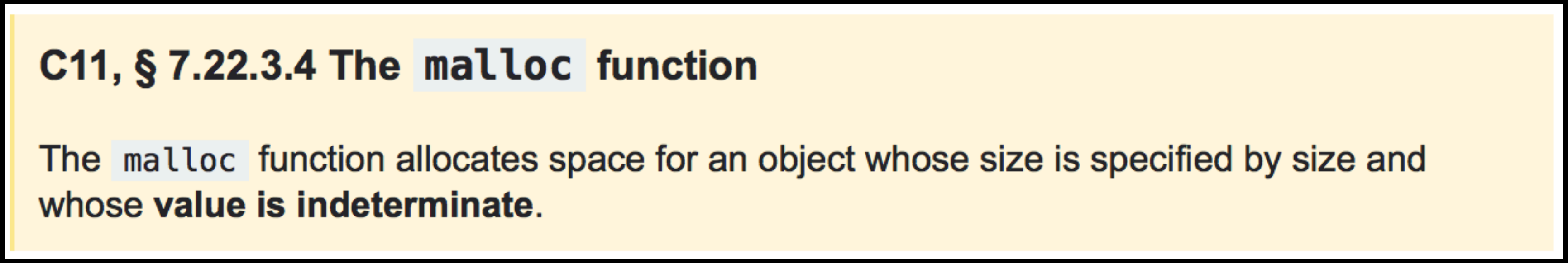
Undefined behaviour

- Save file as test.c
- gcc test.c
- ./a.out
- What is your result?

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  int* x;
  x = malloc(sizeof(int));
  printf("%i\n",*x );
  return 0;
}
```

Undefined?


C11, § 7.22.3.4 The `malloc` function

The `malloc` function allocates space for an object whose size is specified by size and whose **value is indeterminate**.

- C is a specification agreed upon by a committee
- They decide for example what should happen if you malloc
- Compiler manufacturers still need to declare some behaviour
- Therefore different compiler can yield different results

Types!

Tell me what you want (to store)

- Most of you likely come from Python

```
test = "hello"
test = 5
print(test)
```

- Python takes your code and interprets the details at runtime
  - What data do you want to store? Integer? Float? String?
  - How much memory do you need to store it?

Dynamic vs Static Typing

- Dynamic typic inspects each variable at runtime and **infers** an appropriate type
- Good for programming beginners and quick and dirty scripts!
- Bad for performance (EVERY. SINGLE. VARIABLE. Needs a type check while execution)

```python
def python_tease():
    while True:
        if random():
            mystery = [1,2,3]
        else:
            mystery = {1,2,3}
        mystery.remove(1)
```

Static Typing

We take matters in our own hand

- For each variable, we tell what type it is and how much space it needs!

```
int sum = 0;
float pi = 3.14;
char x = 'x';
```

- Could be annoying at first, but has 2 important benefits

  1. You don’t need to guess what’s in a variable

  2. The compiler also doesn’t need to guess..

Challenge

Check if a number is prime! (Here are some ingredients)

```c
#include <stdio.h>

// + add
// - subtract
// * multiply
// / divide
// % modulo (23 % 4 == 3, 16 % 5 == 1, 10 % 4 == 2)
// == (two '=' check if something is equal)
int sum_of_numbers(int x) {
    int sum = 0;
    for(int i = 0; i < x; i++){
        sum = sum + i;
    }
    return sum;
}

int main(){
    int result = sum_of_numbers(23);
    printf("%i\n",result);
}
```

Needed to print stuff on the screen