(1) Introduction of the C++ Programming Language

Nico Ludwig (@ersatzteilchen)

TOC

- Cited Literature:
  - Bruce Eckel, Thinking in C++ Vol I
  - Bjarne Stroustrup, The C++ Programming Language
  - Pavel Mayer and Tim Pritlove https://cre.fm/cre063-die-programmiersprache-c-plus-plus

Initial Words

Yes, my slides are heavy.

I do so, because I want people to go through the slides at their own pace w/o having to watch an accompanying video.

On each slide you'll find the crucial information. In the notes to each slide you'll find more details and related information, which would be part of the talk I gave.

Have fun!

Why do we develop new Software?

- Before we start to understand how programming works, we have to understand why we do software development.
- Usually, the basic thing programs do, is solving a problem, which is hard/cumbersome/impossible to be done manually.
  - In most cases, if not all, this problem must be solved repeatedly.
- In other words, if we know an existing program that already solves the problem in question, we have nothing to program!
  - It sounds silly, but good programmers know, when they have nothing to program at all!
  - Many existing programs are reusable for a class of similar problems.
- So called standard software, like office products are extreme examples of reusable programs.
- A so called expert system is software working as a knowledge-based system.
  - Such a system needs to be "charged" with pretty simple basic rules of the expert knowledge domain.
  - Those rules are evaluated, e.g. via a predicate logic, to, e.g. derive a certain illness from a bunch of symptoms.
  - E.g. the system derives knowledge from its pre-defined rules. – An expert system is just an implementation of artificial intelligence.
- But in this course we'll more or less only write specialized programs to solve our special problems/assignments.
  - We're not going to learn how to write standard software!

How do we develop Software?

- Development of software is classically done in phases. Following phases could be identified for example:
- Phase 1: Formulation of the problem as functional specification or one or more user stories or one or more epics.
- Phase 2: Analysis of the problem to extract the algorithm.
  - Identify input data, the result (i.e. output data) and the processing approach.
- Phase 3: Graphical representation of the processing approach.
  - Graphical representations of processing approaches with flowcharts or Nassi–Shneiderman diagrams (NSDs or "structograms") can be useful.
- Phase 4: Implement the extracted algorithm in a programming language.
  - During programming, so called dry runs with value tables are useful to check algorithms for correctness.
- Phase 5: Program test
- Phase 6: Documentation

- The functional specification is called "Plichtenheft" in German.
- A dry run is called "Schreibtischtest" in German.

Phase 1: Formulation of the Problem – Challenge: Interpret an Update Log

- Real world programs can be quite complex, therefor it is required to define the customer requirement (c-requirement/Lastenheft).
  - The c-requirement can be very detailed, so that customer/contractor have a common understanding of the problem to be solved.
  - The c-requirement is provided by the customer, the contractor "answers" with a developer requirement (d-requirement/Pflichtenheft) "how will it be implemented".
  - => This follows the standardized definition of c- and d-requirement of the Software Requirements Specification (SRS) after the IEEE.

- Another popular way to formulate a smaller problem to be solved with a smaller program, is the definition of a user story.
  - A user story describes the problem to be solved in plain language from the perspective of the user, which has the problem.

- Let's assume this user story for the following discussion:
  - "As an administrator I want to know the latest updates on a PC, in order to get a report of that PC's state!"

- It is needed to interpret the file "Software Update.log", which looks like this:

```
2009-04-06 12:43:04 +0200: Installed "GarageBand Update" (5)
2009-04-06 12:44:34 +0200: Installed "iMovie Update" (8)
2009-07-30 13:11:28 +0200: Installed "iMovie Update" (9)
2009-04-06 12:43:31 +0200: Installed "iTunes" (8)
. . .
```


- How can we solve this problem?

Phase 2: Analysis of the Problem to extract the Algorithm – Input Data

- Initially, the file to be analyzed looks quite complex, but a closer look shows, that we only need to inspect few data.
  - Analysis of input data requires us to answer two questions basically.

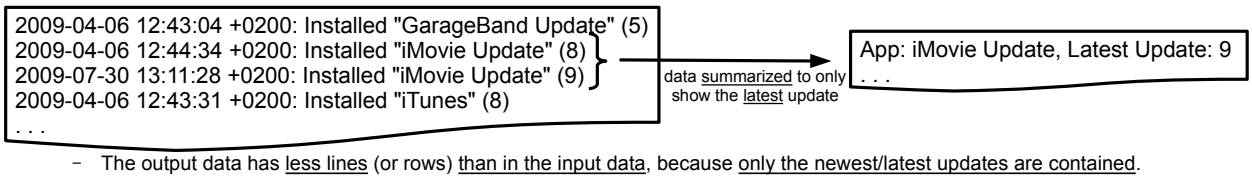
```
2009-04-06 12:43:04 +0200: Installed "GarageBand Update" (5)
2009-04-06 12:44:34 +0200: Installed "iMovie Update" (8)
2009-07-30 13:11:28 +0200: Installed "iMovie Update" (9)
2009-04-06 12:43:31 +0200: Installed "iTunes" (8)
. . .
```

- Question 1: What input data is required to solve the problem?
  - Each line represents a software update and each line is independent of the other lines.
  - The important portion of data can be found in the last parts (or columns) of each line, there we can find:
    - (1) the software name of the software being updated (blue boxes),
    - and (2) the version number of the update (red boxes) – the higher the number, the newer/later the update.
    - => The rest of the data in a specific line can be ignored for the analysis.
- Question 2: Of which data type is the input data?
  - Another important point is that the software name is textual data and the software version is numeric data.
- Mind, that all updates of a specific software item are listed in that file.
  - E.g. the above snippet contains two entries for "iMovie Update", one for version 8 and one for version 9.
  - This also means, that we have to analyze each line of the file, because we want to find the latest version reliably!

Phase 2: Analysis of the Problem to extract the Algorithm – Output Data

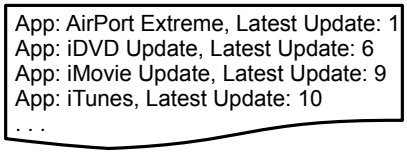
- Now we have to exactly define the presentation of the solution of the problem to the user, which satisfies the user story.
  - In "mathematical terms" the solution is the result of the program, in programing terms, it is the output data of the program.

- Only relevant data is presented, superfluous data eliminated



  - The output data has less lines (or rows) than in the input data, because only the newest/latest updates are contained.
  - Each row in the output data has only two portions of data: the name of the app and its newest/latest update's version number.

- The output data should be formatted and sorted by the software's (app) name respectively. Then we'll have this output:



Phase 2: Analysis of the Problem to extract the Algorithm – Processing Approach

- Now we have to think about the processing steps and their order to get and output the desired output data.

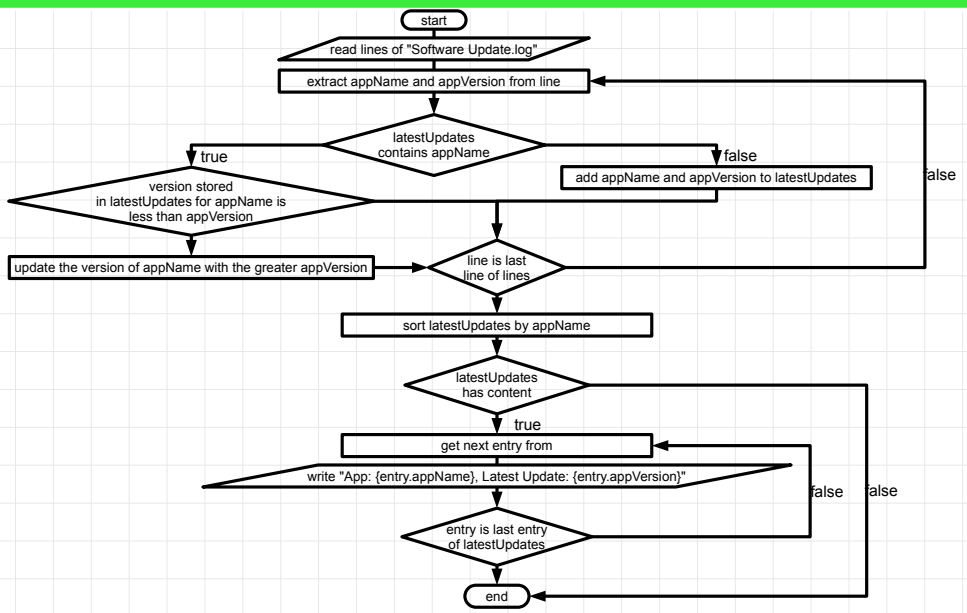
- (Step 1) We have to read each line of the log file.
  - (Step 2) For a specific line extract the software name and the software version number.
    - (Step 3) Then store the extracted software name and extracted software version number in a dictionary:
      - (Step 4) If the extracted software name is already stored in the dictionary check the version:
        - (Step 5) If it is less than the extracted version replace the stored version with the extracted version.
        - (Step 6) Else: do nothing!
      - (Step 7) Else: If the extracted software name is not stored in the dictionary:
        - (Step 8) Store the extracted software name and extracted software version number in the dictionary
    - (Step 9) If that line was not the last line, repeat all steps from (2) with the next line.
  - (Step 10) If that line was the last line:
    - (Step 11) Sort the dictionary by the software name.
    - (Step 12) For each dictionary's entry:
      - (Step 13) Format the data like "App: <software name>, Latest Update: <latest update>"
      - (Step 14) Print the formatted entry to standard out, so that the user can see the result.
- (Step 15) Program end.

Phase 3: Graphical Representation of the Processing Approach – Overview

- Sometimes, it can be useful to discuss the processing approach in a graphical way. There are some benefits:
  - There are some standardized diagram types, which are completely independent of a certain programming language.
  - The resulting diagrams can be more human readable.

- Now we'll look at two graphical representations:
  - Flowchart diagrams
  - Nassi-Shneidermann Diagram

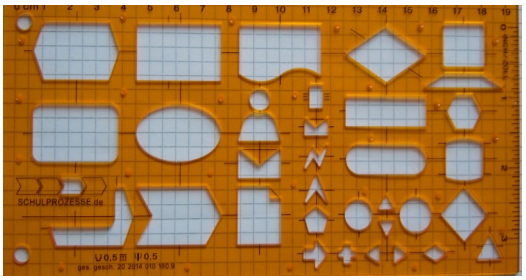
Phase 3: Graphical Representation of the Processing Approach – Flowchart



Phase 3: Graphical Representation of the Processing Approach – Some Words on Flowcharts

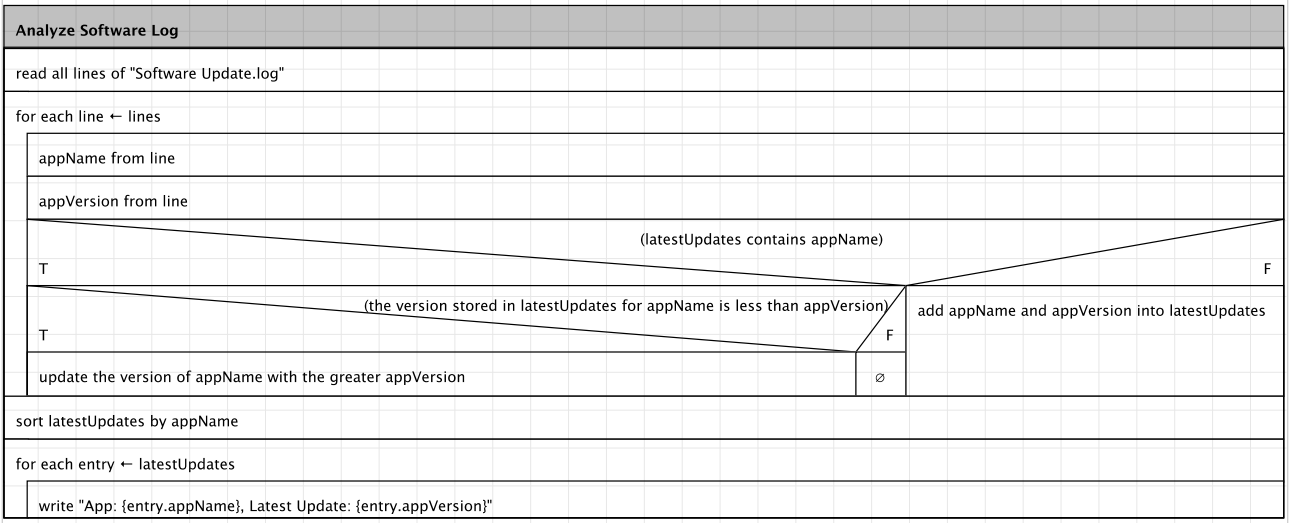
- Flowcharts are diagrams after the standard ISO 5807/DIN 66001, which show the flow of control in a "process".
  - The diagram just connects different symbols with connectors (w/ or w/o arrow tips) to depict the flow of control.
  - Flowcharts are not only used to visualize algorithms, but all kinds of control/information flow, e.g. business processes.

- Benefits:
  - They can be easily drawn without software by hand. Special stencils are also available.
  - Flowcharts can also easily be modified with rubber and pencil.
  - Because of its simplicity it is the ideal means to design algorithms on the fly.



- Downsides:
  - People say, that it is simple to design unstructured algorithms, which contain unconditional jumps (so called gotos) with flowcharts.
    - The diagram would in this case have many connection lines, which unconditionally connect rectangular statement symbols.

Phase 3: Graphical Representation of the Processing Approach – NSDs



Phase 3: Graphical Representation of the Processing Approach – Some Words on NSDs

- Nassi-Shneidermann Diagram (NSD, structogram) after the standard DIN 66261.

- Benefits:
  - NSDs can be read top down basically. With flowcharts instead, we have to follow the lines.
    - NSDs are more structured than flowcharts., hence the name "structogram".
  - NSDs support a very strict design, it helps implementing structured programs.

- Downsides:
  - NSDs are relatively difficult to draw. Changes in diagram often leads to a complete redraw.
  - Alas, software engineers will have to cope with NSDs in school at a certain point in time.
    - … and the need to draw an NSD is a frustrating task in an assignment or exam, because the draw process doesn't forgive errors!
    - Tip: If you can decide to draw a diagram (esp. NSD) or to write pseudo code in an exam choose to write pseudo code!

Phase 3: Graphical Representation of the Processing Approach – the Reality

- Nowadays, graphical representations of processing approaches at the design phase are used rarely.
- Diagrams are not suited for complex algorithms:
  - The required space is just too large, loosing its most important benefit being human readable.
  - Such diagrams are difficult to maintain.
- Generally, algorithms are different today:
  - Flowcharts and NSDs have been designed to support imperative, esp. structured programming.
    - Structured programming means to program only using sequences, branches and loops.
  - The usage of procedural, functional and object-oriented means to solve problems renders flowcharts and NSDs inappropriate.
  - Parallel programming to exploit the multicore processing power of modern CPU is often difficult to depict.
    - However, NSDs have been extended to express parallel algorithms in a limited way.
- Software development (SD) is different today:
  - Flowcharts and NSDs clearly rather support waterfall approaches on SD, but nowadays agile SD approaches are applied.
- Flowcharts still have a certain relevance to show a higher level of solution domains: service interaction, business processes and other workflows.
  - Mind that those modern incarnations of graphical representations base on flowcharts and not on NSDs!

15

- Nowadays a lot of (free) Computer Aided Software Engineering (CASE) tools are available, which allow drawing flowcharts and NSDs. Such tools also allow converting code to a diagram and vice versa.

Phase 4: Implement the extracted Algorithm in a Programming Language

- Sure, to solve the problem defined in the user story we're going to implement a program. But: what is a program?
  - A program is a logical series of commands to define work instructions to complete a task, which can be executed as algorithm.

- All right! And what is an algorithm exactly? An algorithm is
  - a series of exact commands (we already know this),
  - which certainly and repeatedly yields correct solutions for the problem
  - with a finite set of steps.
  - When an algorithm is written as program, "everything must be correct", because the computer cannot ask us for later corrections.

- A program should be written in a common way, that it can be applied for similar problems. – It should be reusable!
  - It makes sense to question ourselves early: is the way I wrote the program applicable to solve more complex problems?

- => We are going to implement the analyzed problem in a symbolic, i.e. "real" programming language (language).
  - A symbolic language allows to express processing steps of an algorithm with a set of clearly defined symbolic command names.
  - Usually, such a symbolic language is also called High-Level Language (HLL, German: "Hochsprache").

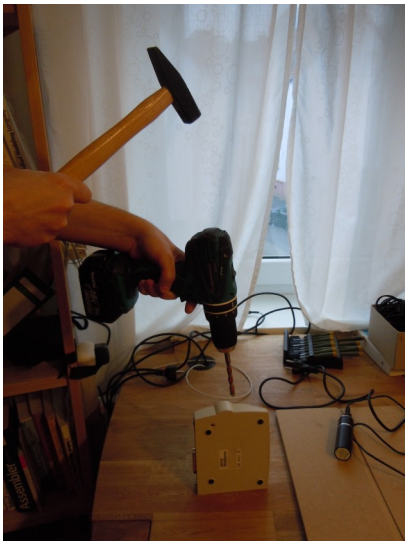
Phase 4: Implement the extracted Algorithm in a Programming Language

- But … which programming language (language) should we use to code this algorithm?
- Choosing a language is often driven by personal preferences.
  - Languages are just tools!
  - Craftspersons use different tools for different problems – a screwdriver isn't better than a hammer, its different!
  - When you only know how to use hammers, every problem might look like a nail – but not all problems are nails!
- On the following slides, we implement our problem with different languages.
  - Ideally, we should compare and choose languages regarding the problem we're going to solve. – Choose the right tool!
  - We'll also see, that our problem can be solved simpler in one language than in another.
- But before looking at other tools, let's first take a look at C++, the tool, we are going to learn in this course.
  - Yes, we choose also C++ to solve this problem, because we want to learn C++ in this course!

17

- Which programming languages are known to the audience?
- Now let's inspect how we can solve this problem with a computer program. – We are going to discuss solutions in some high level languages (HLLs) that are appropriate to solve this problem.
- What is a HLL?
  - A HLL allows expressing solutions in a more human readable way. Hardware near languages (e.g. assembly) are hardware specific and not standardized, HLLs are in opposite hardware independent and standardized (so are C/C++).
- What is a low level language?
  - A low level language is a machine near language that works only on one type of machine (assembly), so they are often not portable.

Phase 4: Implement the extracted Algorithm in a Programming Language



Avoid the hammer-effect: try to understand, what the problem is, not how it can be solved with a certain tool!

C++17

```
#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include <regex>
int main(int argc, char* argv[]) {
	// check arguments from console
	if (2 >= argc) {
		// open and check file
		std::ifstream logFile(argv[1]);
		if (logFile.good()) {
			std::map<std::string, std::string> latestUpdates;
			std::string aLine;
			do { // read a line
				std::getline(logFile, aLine);
				// parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
				std::cmatch res;
				std::regex rx(R"([^\"]*\"([^\"]*)\"\s*\(([^\(]*)\))");
				if (std::regex_search(aLine.c_str(), res, rx)) {
					// store the parsed data
					if (latestUpdates.end() == latestUpdates.find(res[1]) || std::stoi(res[2]) > std::stoi(latestUpdates[res[1]])) {
						latestUpdates[res[1]] = res[2];
					}
				}
			} while (logFile.good());
			// output the parsed data to console
			for (const auto& item : latestUpdates) {
				std::cout<<"App: "<<item.first<<", Latest Update: "<<item.second<<std::endl;
			}
		}
	}
}
```

Step 1: read all lines

Step 2: appName and appVersion from line

Steps 3-8: complete logic to care for the latestUpdates dictionary

Step 11: automatically sorts the latestUpdates dictionary

Steps 12-14: write each entry to standard out

- We can identify certain pieces in the code along the steps of the processing approach we have determined.
- Esp. because this is a symbolic code, it resembles partially spoken (English) language.
- The indentation of this code was also done along the processing steps to give the code a structure.
- The indentation of this code also resembles the cascaded style of the NSD we have designed.
- Following a structure of steps in program code is basically the "imperative programming paradigm".
- With this programming language we'll deal in this course.
- Application areas:
  - (C++ is one of the most complex programming languages.)
  - C/C++ count to the most applied programming languages today.
  - The huge success of C++ is due to the fact that it was based on C by design. – So C programmers could switch to C++ relatively easy, also porting of C code was not a big deal.

Phase 4: Implement the extracted Algorithm in a Programming Language

- Before we can put the presented C++ program into effect, it needs to be compiled.
  - Compilation means, that the symbolic C++ code is transformed into machine code.
  - C++ code is just written in a text file with the extension .cpp. Let's assume our program code resides in main.cpp.
  - A compiler is a program, which performs this transformation. A specific C++ compiler is gcc (for GNU Compiler Collection).
- Compile the C++ program in main.cpp separately, the compiled machine code is put into a binary file named main.o:

```
Terminal
NicosMBP:src nico$ g++ -c main.cpp -std=c++11
NicosMBP:src nico$
```

- We need a further immediate step in C++: we have to link main.o to the effective program file called "main":

```
Terminal
NicosMBP:src nico$ g++ main.o -o main
NicosMBP:src nico$
```

- After linking, we can execute main and pass the path to "Software Update.log" as argument:

```
Terminal
NicosMBP:src nico$ ./main "Software Update.log"
App: AirPort Extreme Update 2008-002, Latest Update: 1
App: Front Row Update, Latest Update: 1
App: J2SE 5.0 Release 4, Latest Update: 4
App: Keynote Update, Latest Update: 3
...
```


- In upcoming lectures, we will discuss separated compilation and linking in detail.
- Here the full command lines for separated C++ compilation and linking with gcc (macOS 10.14):
  - Here we use gcc's variant g++ on the command line. Often, g++ in an alias for gcc, which is predefined with some C++-standard options.
  - Compile (g++ is preferred):

```
g++ -c main.cpp -std=c++11
gcc -c main.cpp -std=gnu++11
```

  - Link (g++ is preferred):

```
g++ main.o -o main
gcc main.o -o main -lstdc++
```

Shell Script (Bash)

```
#!/bin/bash

# check arguments from console
if [ $# -gt 0 ] && [ -f "$1" ]
then
   # open file
   # parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
   # process input with unique sorting
   # output the collected data to console
   sed -En 's/.*"([^"]*)\"\s*(([^)]*))/\1:\2/p' "$1" \
      | sort -t':' -k1,1 -k2n,2r \
      | sort -u -t':' -k1,1 \
      | while read line
        do
            echo "$line" | sed -En 's/(.*):(.*)/App: \1, Latest Update: \2/p'
        done
fi
```


- Shell scripts run on the command line interface of an OS.
  Here the the GNU/Linux bash shell was used. It is widely applied (the usual default for Linux distributions) and available on almost all unixoide OS' (Unix often rather uses the Korn shell (ksh), macOS Catalina (10.15) uses zsh). This example uses script code following the POSIX standard. The Portable Operating System Interface or Portable Operating System Interface based on UNIX (POSIX) is an IEEE- and Open Group-standard for a platform-independent API to operate with the OS. In past, US government organizations made POSIX compliance a requirement for OS' they use.
- Shell script is not directly a programming language, but nevertheless a tool like a programming language.
  - It provides an execution context with commands being executed in the scripted order. But individual commands can also be executed in isolation and interactively in the shell.
  - Some commands are kind of integrated into the shell (highlighted in blue color). Own or 3rd party command line applications can be used like shell commands seamlessly.
  - Its main application area is in automation, administration and analysis affairs of filesystems and OS' (esp. for bulk operations). – The analysis of a log like in this example is a typical real life application of shell scripting.
    - Also GUIs can be put into effect from the shell with TCL/TK (Tool Command Language), which could easily be extended with C-functions. TK (Toolkit) was a set of C-functions, which allowed TCL to control X11 for graphical interactions. Even nowadays TCL/TK is good for prototyping.
- Shell scripting is very mighty, but contrived:
  - "Shell" is often used for the "command line", in this discussion the shell is the "interpreter" of scripts.
  - The name "Shell" stems from the idea that it acts like the outmost layer "around" the OS similar to the shell of an oyster.
  - The "terminal" used to be a pice of hardware presenting the Shell to the user. Nowadays the terminal is a piece of software executing the Shell to the user and the Shell can be bash, zsh, ksh, fish etc. (Terminal = application that can run/connect to/interact with different Shells)
  - Different commands must be used together, their results being combined with "pipes".
    - Pipes and FIFOs are file system objects and buffers.
    - Programs in the early 1970ies could not be very large and could also not use much memory. Therefor piping allowed independent programs to communicate with each other.
    - The basis for piping to work, is that programs use stdin and stdout to accept input and dump output. → Therefor those are called standard-in and -out.
    - Piping also allows programs to be written in different languages.
- Regular expressions (tools: (e)grep) and means for string manipulation are typical tools in the shell.
  - The commands applying these tools do often introduce their own micro-languages (like sed or awk), which are contrived and not always portable. The micro-languages were often programed with yacc (yet another compiler compiler, a so called "parser-generator"), which also be used for own programs.
  - awk (pronunciation [ɔːk], named after the first letters of the names of its inventors Alfred V. Aho, Peter J. Weinberger und Brian W. Kernighan) is a functional combination of sed, (e)grep and some database-tooling to process formatted text files in a very sophisticated manner.
- Nevertheless, shell scripting is not very good with text processing itself, therefor the mentioned programs have to be piped to work together. However, shell scripting is usually terrible with number processing.
- => Shell scripts are often very cryptic and difficult to debug.
- (It's interesting that many system administrators shun application development (i.e. "real" programming), but have no problem with shell scripting, which can become comparably complex to program.)
- When is shell scripting an appropriate tool:
  - If a set of instructions must be executed for multiple times
  - If data must be temporarily stored for further processing.
  - If prepared or interactive input from the user is required.
  - => For these kinds of tasks we find a lot of additional programming constructs, which are also present in other languages.
- When is shell scripting inappropriate:
  - Sure, we shouldn't confuse shell scripting with a HLL.
  - Sophisticated and fast graphics processing, time-critical applications, processing of large amount of data (e.g. with databases) are still to be tasks to be solved with HLLs.

Perl 5

```
# check arguments from console
if (0 < scalar(@ARGV)) {
    # open and check file
    open(logFile, $ARGV[0]) || die($^E);
    my %latestUpdates = ();
    # read a line
    while (<logFile>) {
        # parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
        if (/.*\"([^\"]*)\"\s*\(([^\)]*)\)/o) {
            if (!exists $latestUpdates{$1} || $2 > $latestUpdates{$1}) {
                $latestUpdates{$1} = $2;
            }
        }
    }
    close(logFile);
    # output the parsed data to console
    for my $appName (sort { "\L$a" cmp "\L$b" } keys %latestUpdates) {
        print "App: $appName, Latest Update: $latestUpdates{$appName} \n";
    }
}
```


- The performance and compactness on one hand but also the restrictions of sed and awk on the other hand inspired Larry Wall to develop Perl.
  - Esp. one problem was, that the shell could not hold status in between pipelining. To get along with this restriction, more information was piped from one program to the next.
- In Perl simple tasks can be solved simply and hard tasks are solvable.
  - (Yes, it should be said that before Perl, simple tools could be created with the yacc parser-generator.)
- Libraries for Perl can be found at the Comprehensive Perl Archive Network (CPAN)
- The syntax is intuitive but can become cryptic ("write only code"), because Perl provides many syntactic shortcuts.
  - Perl itself allows to solve even very small problems in completely different ways, people call that "There's more than one way to do it" (TMTOWTDI or TIMTOWTDI, pronounced Tim Toady).
  - The application of syntactic abbreviations can led to code, which nobody can read anymore: write only code. Perl developers develop their own style, based on the "tricks" the language provides: it can be difficult to combine the bases of only two developers.
- Regular expressions and means for string manipulation are intrinsic in Perl.
- Application areas:
  - Part of each GNU/Linux distribution.
  - Bioinformatics
  - In system administration, where it is used as "glue" for other programs.
  - CGI on the web server.
  - It is not so useful for big applications.

Python 2.7.1

```
import sys
import io
import re
# check arguments from console
if 0 < len(sys.argv):
    latestUpdates = {}
    parseRex = re.compile(r'[^\"]*\"(?P<appName>[^\"]*)\"\s*\((?P<versionNo>[^\(]*)\)')
    # open and check file, read a line
    for aLine in open(sys.argv[1]).readlines():
        # parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
        match = parseRex.search(aLine)
        if match:
            key = match.groups()[0]
            value = match.groups()[1]
            # store the parsed data
            if key not in latestUpdates or int(value) > int(latestUpdates[key]):
                latestUpdates[key] = value
    # output the collected data to console
    for key in sorted(latestUpdates):
        print 'App: {}, Latest Update: {}'.format(key, latestUpdates[key])
```

23

- Named after "Monty Python's flying Circus".
  Created by Guido van Rossum [ˈɣido vɑn ˈrɔsʏm] in 1991, which also provided the reference implementation.
  - The ancestor language was ABC.
- A modern multi paradigm language:
  - What programming paradigms are known to the audience?
    - Imperative, procedural, functional, object oriented (oo).
- Well suited for beginners, good readability:
  - When code exploits Python syntax to achieve good readability it is said to show "pythonic" style.
  - A syntax with semantic white-spaces (also called "off-side rule") that advances a prosaic programming style (white-spaces make up the "blocks" of Python). (But semantic white-spaces makes Python code difficult to generate.)
  - YAML (YAML Ain't Markup Language) also applies an offsite-rule to implement structure.
  - Other languages with off-side rule: Cobra (a successor of Python), F# (can be switched on and off) and in some corners of Haskell.
  - Problem of that syntax: if the off-side rule is hurt, a Python program will probably break, therefor unit tests are really required!
- Python 3:
  - In 2008 Python was reimplemented anew as Python 3. Alas, Python 3 is incompatible to Python 2. Nevertheless, Python 2 further-development is going on, because a lot of libraries important for industry are based on Python 2. E.g. on many Linux distributions we have different command line interpreters *python* and *python3*.
- Application areas:
  - Part of each GNU/Linux distribution.
  - Good libraries for web programming (Django) together with a JavaScript(-library) frontend.
  - AI-programming/machine learning: libraries like Caffe or TensorFlow, evaluation of numbers: libraries like Numpy and Pandas Those libraries are implemented in C to gain maximal performance at run time, but offer a pythonic interface.
  - Scripting language for some applications (e.g. Open Office and GIMP).
  - A .NET implementation is existent (IronPython).
  - There also exists an integration with Cocoa on macOS.

Ruby 1.8.7

```
# check arguments from console
if 0 < ARGV.length()
    parseRex = /[^\"]*\"([^\"]*)\"\s*\(([^\(]*)\)/
    latestUpdates = {}
    # open and check file, read a line
    IO.readlines(ARGV[0]).each { |aLine|
        # parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
        if parseRex.match(aLine)
                # store the parsed data
                if not latestUpdates.has_key?($1) or $2.to_i > latestUpdates[$1].to_i
                    latestUpdates[$1] = $2
                end
        end
    }
    # output the collected data to console
    latestUpdates.sort().each { |appName, versionNo|
        printf("App: %s, Latest Update: %s\n", appName, versionNo)}
end
```


- The name was inspired by Perl. And basically also designed as competitor of Perl (like Python) with the same application areas.
- Created by Yukihiro Matsumoto (Matz), a Japanese programmer, in 1995.
  - Very popular in Japan. The worldwide popularity raised, when the English documentation was available in 2000. Another milestone of popularity was the introduction of "Ruby on Rails" (Rails), which is a kind of innovative web framework (it uses the idea of "Convention over Configuration", which allows fast development). (Python's equivalent of Rails is "Django".)
- A modern multi paradigm language (but all types work oo).
- Matz took the "best" concepts from the programming languages Smalltalk, Lisp, Eiffel and Ada. E.g. the concept of blocks was taken from the Smalltalk programming language.
- Ruby is easy to learn, it follows the "principle of least surprise" (POLS) in syntax and semantics: "if I understand how the concept 'x' works, I will also understand like concept 'y' works". Yet Ruby is as mighty as Perl/Python.
- Application areas:
  - Good libraries for web programming (scaffolding with "Ruby on Rails"). But it should be said, that some shops went to Java or .NET after a while, because Ruby on Rails' performance can be a problem.
  - Similar to the application areas of Python.
  - Ruby is well suited for Domain Specific Languages (DSLs).
  - A .NET implementation is existent (IronRuby).
  - In Ruby there's no idea of byte code, so we have to pass source code to customers! In opposite Python does support byte code compilation.

Groovy 2.1.5

```
import java.nio.file.*

// check arguments from console and check the file
if (args[0] && Files.exists(Paths.get(args[0]))) {
    def pattern = /.*\"([^\"]*)\"\s*\(([^\)]*)\)/
    def latestUpdates = new TreeMap<String, String>(String.CASE_INSENSITIVE_ORDER)
    // open the file, read a line
    new File(args[0]).each {
        // parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
        def matcher = it =~ pattern
        if (matcher) {
            // store the parsed data
            def key = matcher[0][1]
            def value = matcher[0][2]
            if (!latestUpdates[key] || value < latestUpdates[key]) {
                latestUpdates[key] = value
            }
        }
    }
    // output the collected data to console
    latestUpdates.each { appName, versionNo -> println "App: $appName, Latest Update: $versionNo" }
}
```

25

- Designed/developed by James Strachan and Guillaume LaForge, it appeared in 2003.
- A multiparadigm language. It does esp. also support dynamic typing.
- Can be interpreted or compiled, so it can be used as scripting language.
- Based on the JVM.
  - It is counted as the second JVM language besides Java. Groovy is a first class citizen on the same VM, parallel to Java.
  - The resulting byte code is 100% compatible to Java's byte code. Java programs are able to use Groovy classes and vice versa.
- Is a kind of "Java++", which means a "better Java", because it addresses Java's shortcomings with new features: closures and an intuitive syntax for the most often used Java collections (lists and maps). – One could just take some Java code and "groove it up". Groovy's syntax beyond Java's syntax is rather derived from Ruby/Python than from C. Groovy also extends the Java Development Kit (JDK) with its own additions, which makes the Groovy Development Kit (GDK).
- Groovy is esp. interesting to build Domain Specific Languages (DSLs) on top of it with so called "builders".
- What kinds of programing languages did we discuss up to here?
  - => Scripting languages and/or interpreted languages.
- What kind of languages exist in opposite to these?
  - => Compiled languages
  - Now let's inspect these "real" programming languages.
- Scripting languages are good beginner languages, esp. Python and Ruby are backed by large communities (e.g. Rails).
  - Those languages need no intermediate compilation step, so programmers have an immediate feedback, because programs can be directly executed after being edited. – This is esp, true, because many scripting languages (SDKs and interpreters) are included in most OS' meanwhile. As we will see next, the source code for scripting languages is much shorter than that of "real" programming languages. Scripting languages are very productive but terse.
  - It tends to be difficult to code big applications with scripting languages, but this tendency seems to prove wrong (e.g. Rails).
  - Scripting, esp. with ubiquitous tools like awk and sed on the command line, allow to do complicated tasks that may never be repeated in future: such a task needs not to be a "real" program... The need to be a "real" program also depends on requirements such as performance (memory and execution time).
  - Disadvantages of esp. dynamically typed scripting languages: less support in IDE (code completion etc.).

Common Lisp (CL)

```
(let ((latestupdates
 (reduce
  (lambda (reducedupdates item)
   (if (gethash (first item) reducedupdates)
      (progn
       (setf (gethash (first item) reducedupdates) (append (gethash (first item) reducedupdates) (list (parse-integer (second item)))))
       reducedupdates)
      (progn
       (setf (gethash (first item) reducedupdates) (list (parse-integer (second item))))
       reducedupdates)))
  (map
   'CONS
   (lambda (line) ; parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
    (let ((keybegin (+ 1 (search "\"" line))))
     (let ((keyend (search "\"" line :start2 keybegin)))
      (let ((valuebegin (+ 1 (search "(" line :start2 keyend))))
       (let ((valueend (search ")" line :start2 valuebegin)))
        (list (subseq line keybegin keyend) (subseq line valuebegin valueend)))))))
   (with-open-file (stream (first *unprocessed-command-line-arguments*) :direction :input) ; open the file and read all lines
    (loop for line = (read-line stream nil) while line collect line)))
  :initial-value (make-hash-table :test 'equal))))
 (map
  nil
  (lambda (appname) ; output the collected data to console
   (format t "App: ~A, Latest Update: ~A ~%" appname (reduce #'max (gethash appname latestupdates))))
  (sort
   (loop for k being the hash-keys in latestupdates collect k)
   #'string<)))
```

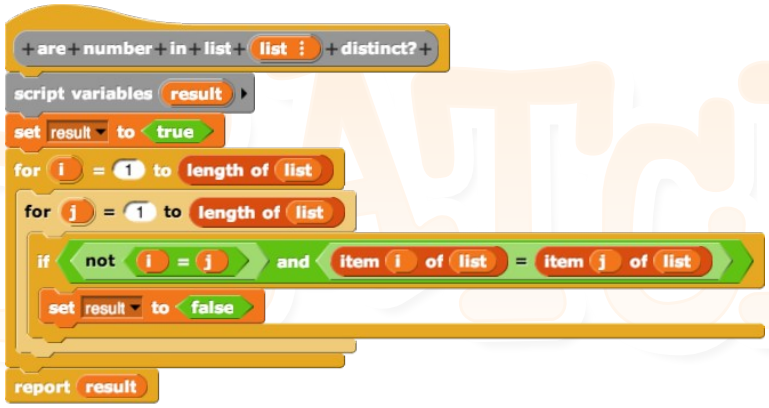

- Designed by John McCarthy.
- One of the oldest programing languages (Lisp appeared in 1958) together with Fortran (appeared in 1957). (ALGOL (ALGOrithmic Language) appeared in 1958 as well.)
- Originally, Lisp is the abbreviation for LISt Processing, therefor the historical writing is LISP.
- In principle, Lisp has only one syntactical concept: lists (The logo in the background of the slide is the graphical representation of Lisp's lists, which is used in literature: the "box notation".). – Code and data will be represented by the list concept; e.g. code can be used as data. The idea to represent code and data in the same way is called "homoiconicity" (Greek for "being self-representable"). The highlighted symbols are not directly reserved, but have a known meaning in Lisp. The highlighting improves the readability somewhat, because the first element of a list designates the operation (function name) the expression performs, this called prefix syntax. Lisp programmers long for "beautiful" programs.
  - Lisp allows working with the so called lambda calculus, which allows to express mathematics symbolically in a programming language. E.g. we could do differential and integral calculus in Lisp!
  - Some people say that Lisp has no syntax at all.
  - Lisp's lists are made up of items put into parentheses. – A Lisp program's source code is basically just list of further cascaded lists. This syntax scares many people, because there are a lot of parentheses in Lisp! To be frank, after a while the syntax is not really bad at all …
    - Lisp also provides macros for meta-programming.
  - In comparison to programming "block-oriented" languages, which feels like working with brick stones, programming Lisp feels more or less like working with modeling clay. (Hans Hübner)
  - "Lisp is a programmable programming language." (John Foderaro)
- Lisp is a so called functional programming (fp) language. Fp works like writing a gigantic formula or editing a spreadsheet. – Maybe spreadsheet programs (like Excel) are the most often used fp languages, they are even used by non-programers. In fp languages, control flow is often not directly present (for programmers of non-fp languages), instead expressions could have been evaluated in any order! (A horrible assumption for, e.g. a Basic programmer!)
  - Lisp applies a garbage collector to manage consumed memory automatically. This is required for fp languages, because they deal with a lot of temporary data!
  - The first fp language was Information Processing Language (IPL) it appeared in 1954.
- Lisp programs, and esp. isolated parts of its, i.e. the functions, can also be executed in isolation and interactively. Theoretically this is possible, because functions have no side effects and yield the same result when called with the same arguments. Technically this is possible, because Lisp provides a Read-Eval-Print-Loop (REPL), which allows interactive execution of functions and "free" Lisp-code on a kind of shell. – This can speed-up Lisp-development a lot, because pieces of code can be tested quickly.
- In the end of the 1970ies (until the end of the 1980ies), so called Lisp machines were produced.
  - Those computers featured CPUs, which were optimized to run Lisp programs.
  - Most important manufacturers: Symbolics Inc., Lisp Machines Inc
- Virtually Lisp is a family of programming languages.
  - **Common Lisp (CL)** is one of those dialects. It is a "kind of standard", based on some other Lisp dialects running on many mainframes (esp. the Maclisp family of dialects). People say, that CL was initiated US government (and its Lisp projects), because it was required to guarantee, that present List code can run under a "common Lisp" → CL.
  - Scheme is a "reduced" variant of Lisp. And Racket is another Lisp-dialect derived from Scheme.
  - DSSSL (Document Style Semantics and Specification Language) is a further subset of Scheme to develop SGML (Standard Generalized Markup Language) stylesheets.
  - CLOS (Common Lisp Object System) is an extension of CL, which adds support for oo programming.
  - Dylan was a language developed at Apple in the early 1990ies as primary software development language for the "Apple Newton". The Newton was one of the first PDAs (Personal Digital Assistant), i.e. an ancestor of the "Apple iPhone". Dylon took Lisp, but got rid of the many parentheses in favor to a block syntax. – Someone at Apple didn't like the parentheses.
  - Arc seems to be the next incarnation of Lisp, because it is in development by some of the drivers of Lisp, e.g. Paul Graham.
- Application areas:
  - Education (fp, maths in general)
  - Science, esp. artificial intelligence.

Smalltalk

```
FileStream fileNamed: fileName do: [ :fileStream|  |latestUpdates|
"open file"
    latestUpdates := Dictionary new.
    "read lines in loop"
    [ fileStream atEnd ] whileFalse: [ |parsedLine key value oldValue|
        "parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed 'Digital Camera Raw Compatibility Update (2)'"
        parsedLine := (fileStream nextLine findTokens: #($" $( $)).
        key := parsedLine at: 2.
        value := parsedLine at: 4.
        oldValue := latestUpdates at: key ifAbsent: [ nil ].
        (oldValue isNil or: [ oldValue asNumber < value asNumber ]) ifTrue: [
            "store the parsed data"
            latestUpdates at: key put: value.].].
    "output the collected data to the Transcript"
    latestUpdates keysSortedSafely do: [ :key|
        Transcript
            show: 'App: ',key,' Latest Update: ',(latestUpdates at: key);
            cr.].].
```


- Designed by Allen Curtis Kay et al. at XEROX PARC from 1969 to 1980. So it is a language created by a hardware company!
- Smalltalk is an oo language (Smalltalk's first versions were not oo). – We'll discuss how programming in an oo-language works in future lectures in great depth.
  - Mr. Kay is said to have coined the term "oo-programming". He regretted using that term in hindsight, he would better had called it "message-oriented programming".
  - Esp. UI frameworks written in Smalltalk were based on the so called "Model View Controller" (MVC) oo design pattern.
- The Smalltalk syntax is so simple that its rules can be printed on a post card and learned within one day. The highlighted symbols are not directly reserved, but have a known meaning in Smalltalk (the language has only six reserved symbols). The highlighting improves the readability somewhat.
- Smalltalk knows only oo-means to express code (e.g. objects and message calls) instead of control flow statements. Only following constructs are executable:
  - Sending a message to an object.
  - Assigning an object to a variable.
  - Returning an object from a method.
- Application areas:
  - The Smalltalk object system was virtually the first OS with a GUI (desktop metaphor etc.) and a sophisticated integrated development environment (IDE).
  - What is an IDE?
    - An IDE is a system/application that integrates all the tools we need for developing: editor, compiler, designer, debugger etc. Some IDEs: Visual Studio, Xcode, Eclipse.
    - Typically Smalltalk-only programmers step into the "4GL-trap": "IDE, technology, framework and language is all Smalltalk!" – 4GL-developers can't see a difference here, everything is integrated.
  - Education (it is suited for children learning a programming language).
  - Science, esp. artificial intelligence.
- Smalltalk is a great language, but when it was presented it was ahead of its time. Also the vendors, the religious "language war" in the 1990ies (esp. with C++) and the upcoming Java language killed Smalltalk.

Scratch 3.0



- Scratch was developed at the MIT as educational language. The idea of the team was to allow novices, esp. children, easy access to programming computers. The graphical syntax of Scratch features graphical components of different shapes and different colors (variables are brown, controls are yellow and statements are purple). And only components, which fit together graphically can be be combined to create meaningful programs, i.e. the grammar is expressed via matching graphical elements. The programming process is composing programs via drag'n'drop and editing properties in a graphical manner. However, Scratch has its limits, when hit, we have to move to another language.
- The Scratch's IDE runs in the web browser as pure JavaScript script.
- There also exist "more professional" graphical/visual programming languages like LabVIEW.

BASIC – Part I

```
' check arguments from console
If 2 <= UBound(args) Then
    ' open and check file
    Dim fileItem As FolderItem = New FolderItem(args(1),  FolderItem.PathTypeShell)
    If fileItem.Exists Then
            Dim fileStream As TextInputStream = TextInputStream.Open(fileItem)
            Dim latestUpdates As Dictionary = New Dictionary
            Dim allAppNames() As String
            Dim parseRex As RegEx = New RegEx

            parseRex.SearchPattern = "[^\""]*\""([^\""]*)\""\s*\(([^\(]*)\)"
            do
                ' read a line from the file
                Dim aLine As String = fileStream.ReadLine
                ' parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw
                ' Compatibility Update" (2)
                Dim match As RegExMatch = parseRex.Search(aLine)
                If Nil <> match And 2 <= match.SubExpressionCount Then
                    ' store the parsed data
                    Dim appName As String = match.SubExpressionString(1)
                    Dim versionNo As String = match.SubExpressionString(2)
```


- BASIC is a language, in which programming is a very productive process. It was created in 1964 by John G. Kemeny, Thomas E. Kurtz und Mary Kenneth Keller. One could say, that BASIC is the successor of Fortran (1957, FORmula TRANSlation).
  There exist many dialects (in this example: RealBasic) of the Beginner's All-purpose Symbolic Instruction Code (BASIC). – The reputation-problem of BASIC programmers: the word "Beginner" in the name of the language!
- Bill Gates and Paul Allen virtually created a BASIC interpreter for the MITS Altair 8800 in 1975, making BASIC "the programming language" for computer hobbyists (and in a sense the first programming language for PCs).
  - This variant of BASIC was originally called "Altair BASIC" and was the predecessor of "Microsoft BASIC".
  - Altair BASIC was Microsoft's (that time called "Micro-Soft") first product.
  - The Altair 8800 (powered by an Intel 8080 CPU) was published as a construction manual in the electronics magazine "Popular Electronics" in 1975. To be frank it was a very useless box, it contains only of LEDs for output and switches for inout. – This uselessness inspired nerds like Bill Gates to do something (useful) with Altair 8800 ...
  - BASIC's success was also based on the fact, that MS BASIC was distributed via piracy very quickly. – This piracy was really a good thing for the recognition of MS!
- Commodore BASIC (based on MS' BASIC) was the OS for Commodore's 8-bit line of computers (e.g. incl. the C64 (the BASIC ROM made 9kB) and the PET (Personal Electronic Calculator)).
- Originally, the syntax had no structural features, instead it used GOTOs and line numbers. Modern dialects have overcome this era and BASIC got procedural.
- It is an approachable language for end-users.
  - This approachability is also due to the fact that BASIC is case-insensitive. – Esp. beginners have often to cope with casing of variable and function names.
- BASIC code is rather noisy as there exist many keywords.

BASIC – Part II

```
                    If latestUpdates.HasKey(appName) Then
                        If (Val(versionNo) > Val(latestUpdates.Value(appName))) Then
                            latestUpdates.Value(appName) = versionNo
                        Else
                            allAppNames.Append(appName)
                            latestUpdates.Value(appName) = versionNo
                        End If
                    End If
                End If
            Loop Until fileStream.EOF
            ' output the collected data to console
            allAppNames.Sort
            For Each item As String In allAppNames
                Print("App: "+item+", Latest Update: "+latestUpdates.Value(item))
            Next
    End
End
```

Pascal – Part I

```
program UpdatesReport;
uses
    RegExpr, StrHashMap, SysUtils, Classes, Strings;
var
    logFile : Text;
    aLine : string;
    parseRex : TRegExpr;
    latestUpdates : TStringHashMap;
    appName : string;
    versionNo : PChar;
    allAppNames : TStringList;
    i : Integer;
begin
    // check arguments from console
    if 1 >= ParamCount() then
    begin
        // open and check file
        if FileExists(ParamStr(1)) then
        begin
                Assign(logFile, ParamStr(1));
                parseRex := TRegExpr.Create;
```


- Created by Niklaus Wirth in 1972. Pascal is named after the french mathematician Blaise Pascal.
- It was created as a didactic language; Pascal programs have a very strict structure in order to lead programmers to a structured programming style. One could say, that Pascal is the successor of ALGOL (1958, ALGorithmic Language).
    - A typical way documenting and learning Pascal's syntax is using syntax diagrams and a textual specification as Extended Backus–Naur form (EBNF) (extended by Mr. Wirth to better express iterations).
- Based on Pascal, Wirth developed Modula-2, which closed Pascal's gaps to allow system programming. Esp. does Modula-2 support modularization.
    - Borland licensed the (very fast) Pascal compiler and combined all tools developers need for their work (editor (working with shortcuts from WordStar) and debugger etc.), the concept of an Integrated Development Environment (IDE) was born.
    - The modern object oriented variant "Object Pascal" comes in many forms. The most renowned one is Delphi, which was originally designed by the designer of Turbo Pascal and C#, Anders Hejlsberg, at Borland.
    - Turbo Pascal 5.5 added object oriented features.
    - In the beginning of the 1990ies, Turbo Pascal was ported to Windows (before that it ran on the command line only, e.g. CP/M and MS DOS). – But Win32 programming on Turbo Pascal was basically as hard as it was using C. Therefor, Delphi was developed. Delphi was based on Object Pascal and Borland borrowed the ideas of a graphical integrated development environment (IDE), visual design of graphical user interfaces and a mighty object oriented programming model from Visual Basic from Microsoft. Visual Basic and Pascal allow what is called Rapid Application Development (rapid compared to the speed developing Win32).
    - Delphi has its strengths on working with databases.

Pascal – Part II

```
parseRex.Expression := '[^\"]*\"([^\"]*)\"\s*\(([^\(]*)\)';
latestUpdates := TStringHashMap.Create();
allAppNames := TStringList.Create();
Reset(logFile);
repeat
// read a line from the file
ReadIn(logFile, aLine);
// parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw
// Compatibility Update" (2)
parseRex.Exec(aLine);
if 2 = parseRex.SubExprMatchCount then
begin
        // store the parsed data
        appName := parseRex.Match[1];
        versionNo := GetMem(Length(parseRex.Match[2]));
        StrCopy(versionNo, PChar(parseRex.Match[2]));

        if latestUpdates.Contains(appName) then
        begin
            if StrToInt(String(versionNo)) > StrToInt(String(latestUpdates[appName])) then
            begin
```

```
                        FreeMem(latestUpdates[appName]);
                        latestUpdates[appName] := Pointer(versionNo);
                    end;
                end
            else
            begin
                    allAppNames.Add(appName);
                    FreeMem(latestUpdates[appName]);
                    latestUpdates[appName] := Pointer(versionNo);
            end;
end;
until Eof(logFile);
parseRex.Free();
Close(logFile);
// output the collected data to console
allAppNames.Sort();
for i := 0 to allAppNames.Count - 1 do
begin
    WriteLn('App: '+allAppNames[i]+', Latest Update: '+PChar(latestUpdates[allAppNames[i]]));
    FreeMem(latestUpdates[allAppNames[i]]);
```

Pascal – Part IV

```
            end;
                latestUpdates.Destroy();
                allAppNames.Destroy();
        end;
    end;
end.
```

C (C18) – Part I

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <stdbool.h>


struct pair {
    wchar_t* key;
    long version;
};

int comparePairs(const void* const lhs, const void* const rhs) {
    return wcscmp(((struct pair* const)lhs)->key, ((struct pair* const)rhs)->key);
}

int main(int argc, const char* argv[]) {
    // check arguments from console
    if (2 >= argc) {
        // open and check file
        FILE* const logFile = fopen(argv[1], "r");
        const size_t LINE_LENGTH = 256;
        wchar_t line[LINE_LENGTH];
        int nlatestUpdates = 0;
        struct pair* latestUpdates = NULL;

        if (NULL != logFile) {
            while (NULL != fgetws(line, LINE_LENGTH, logFile)) {
                // parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
                const wchar_t* const appNameBegin = wcschr(line, L'\"');
                const wchar_t* const appNameEnd = wcschr(appNameBegin + 1, L'\"');
                const size_t appNameLength = appNameEnd - 1 - appNameBegin;
                wchar_t* const appName = (wchar_t*)malloc(sizeof(wchar_t) * (appNameLength + 1));
                if (NULL != appName) {
                    wcsncpy(appName, appNameBegin + 1, appNameLength);
                    appName[appNameLength] = 0;
                    const wchar_t* const versionBegin = wcschr(appNameEnd, L'(');
                    const wchar_t* const versionEnd = wcschr(versionBegin + 1, L')');
                    const size_t versionLength = versionEnd - 1 - versionBegin;
                    wchar_t* const version = (wchar_t*)malloc(sizeof(wchar_t) * (versionLength + 1));
                    wcsncpy(version, versionBegin + 1, versionLength);
                    version[versionLength] = 0;
                    const long versionNo = wcstol(version, NULL, 10);
                    free(version);
```


- Created by Dennis M. Ritchie from 1969 – 1973 (i.e. he wrote the first C compiler). C was named as successor of the programming language "B" (also created by Dennis Ritchie).
- Ken Thompson coded the UNIX kernel for the PDP-11, one of the most popular mini computers, in C. UNIX is the basis of many modern OS' like GNU/Linux, Android, iOS, macOS, OS' which run in our cars and TVs. UNIX' derivates rule the world, thus C is a very very important language, and seems to be the most important HLL today.
  - The idea of C is also to write platform independent code:
    - Believe it or not, also Windows is still mainly written in C! Neither C++ not .NET are the main development tools used for its developments. Windows NT was entirely written in C, in order to make it portable. – It ran on Intel machines and was ported to the RISC platform (DEC Alpha machines).
- 1983, Thompson and Ritchie got the Turing award for inventing UNIX and C, basically.
- C is mighty and insecure, it might not be the correct tool to write secure software.
- C basically builds a very thin abstraction layer above assembly, so, C is a kind of portable assembly language.
- C is used to squeeze every cycle from the CPU for a program, but still have a run time library, that can be used from many HLLs.
  - Programs written in C work very good on systems with short resources.
- C is also well suited for programming OS kernels, it was basically designed to program the UNIX kernel with it.
- Originally, Brian W. Kernighan wrote tutorials for the programming language B and then wrote tutorials for C. Finally, Brian W. Kernighan came to the conclusion, that there might be a market for a "C book" to teach people C programming. → In 1978 Kernighan's and Ritchie's book about C-programming, "The C Programming Language" (colloquially called the "K&R C"), was issued. It was the result of combining Kernighan's tutorials and Ritchie's reference manual.
  - This book was the first book discussing a "Hello, World" example to introduce a programming language.
  - The C-coding convention "K&R" (for Kernighan and Ritchie) is named after the coding style used in the examples of this book.

```
                bool found = false;
                for (int i = 0; i < nlatestUpdates && !found; ++i) {
                    if (0 == wcscmp(appName, latestUpdates[i].key)) {
                        found = true;
                        if (versionNo > latestUpdates[i].version) {
                            latestUpdates[i].version = versionNo;
                        }
                    }
                }

                if (found) {
                    free(appName);
                } else {
                    if (NULL == latestUpdates) {
                        latestUpdates = malloc(sizeof(struct pair));
                    } else {
                        latestUpdates = realloc(latestUpdates, sizeof(struct pair) * ++nlatestUpdates);
                    }

                    if (NULL != latestUpdates) {
                        latestUpdates[nlatestUpdates - 1].version = versionNo;
                        latestUpdates[nlatestUpdates - 1].key = appName;
                    }
                }
            }
        }
        fclose(logFile);

        if (NULL != latestUpdates) {
            qsort(latestUpdates, nlatestUpdates, sizeof(struct pair), comparePairs);
            // output the parsed data to console
            for (int i = 0; i < nlatestUpdates; ++i) {
                printf("App: %S, Latest Update: %ld\n", latestUpdates[i].key, latestUpdates[i].version);
                free(latestUpdates[i].key);
            }
            free(latestUpdates);
        }
    }
}
return EXIT_SUCCESS;
}
```

JavaScript (on Node.js)

```
var readLine = require('readline');
var fs = require('fs');

// check arguments from console
if (2 <= process.argv.length) {
	// open the file
	var lineReader = readLine.createInterface({
		input: fs.createReadStream(process.argv[2])
	});

	var latestUpdates = [];
	// reads a line as callback
	lineReader.on('line', function(line) {
		// parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
		var match = /.*"([^"]*)"\s*\(([^\)]*)\)/.exec(line), key, value;
		if (match) {
			// store the parsed data
			key = match[1];
			value = parseInt(match[2]);
			if (!latestUpdates[key] || value > latestUpdates[key]) {
				latestUpdates[key] = value;
			}
		}
	});

	// output the collected data to console, when the file is closed as callback
	lineReader.on('close', function() {
		var appNames = [], i = 0, key, index;
		for (key in latestUpdates) {
			appNames[i++] = key;
		}
		appNames.sort();

		for (index in appNames) {
			console.log('App: '+appNames[index]+', Latest Update: '+latestUpdates[appNames[index]]);
		}
	});
}
```


- Brendan Eich developed the language Mocha, later know as LiveScript, later known as JavaScript for the Netscape in 1995 at a hectic pace. People say, the first draft was ready in 10 days. The problem they wanted to solve is that bare HTML to implement web pages was too static.
- The language core of JavaScript is standardized as ECMAScript (ECMA 262).
- JavaScript has nothing to do with Java (it is no subset)! – Maybe the name JavaScript was chosen, to ride the marketing wave of Java in the mid 1990ies whilst Netscape cooperated with Sun Microsystems. But this is also true for JavaScript's style, originally Brendan wanted to create a Lisp-like language, but he was forced by "the management" to design it Java-like.
- JavaScript was originally designed as "language to be integrated", esp. to run in the web browser (originally in Netscape Navigator) to give websites interactive and lively capabilities, i.e. to manipulate the DOM (Document Object Model) at run time. – These abilities gave JavaScript a push, when the term "Web 2.0" was coined.
  - Other languages designed to be integrated in to an environment are Lua and TCL.
  - Nowadays JavaScript does also run on servers and even in micro controllers.
  - JavaScript is also interpreted in software like PDF readers meanwhile, which can also be a security issue like MS Office macros.
  - Many programming languages can x-compile JavaScript, e.g. Kotlin.
- Node.js (i.e. JavaScript was integrated into web browsers and then also into Node.js):
  - Meanwhile, JavaScript has transitioned from a scripting-ability embedded in the browser to a programming language for full-stack development. So, JavaScript can be used for developing on the server-side and former client-side developers using JavaScript can become back-end developers. Usually JavaScript is used on the server-side using Node.js.
  - The shown example makes use of Node.js, which enables accessing the local file system, so that we can read the "Software Update.log". Node.js is a system, which allows server code to be written in JavaScript. – JavaScript running in the browser can not read arbitrary files on the local file system.
  - Node.js' basis is the V8 JavaScript engine, which is part of the browsers Chrome and Chromium.
  - Node.js was implemented by Ryan Dahl in 2009. Dahl abandoned the further development of Node.js and started over with "Deno" to address the issues we found in Node.js.
  - "JavaScript on the server" lead to an enormous further development of JavaScript.
- JavaScript looks simple, but writing stable code needs to get used to. Some of JavaScript's constructs felt so strange with a "bit of familiarity", that some programmers didn't recognize it as "real" language (due to the hectic pace at JavaScript's design phase). – Douglas Crockford's book "JavaScript: The Good Parts" (2008) tried to support a better professional understanding of JavaScript.
  - "JavaScript looks like C, but is rather to be understood as a variant of Lisp."
  - JS comes w/o a standard library, it only has some support for arrays, strings and regular expressions.
  - When JS is used in the browser, it must often use the DOM (Document Object Model). DOM is *the* aspect, which lead to JS' bad reputation. Some JS "libraries", like jQuery, helped to encapsulate and improve JS' DOM handling.
    - Originally, JS was introduced to manipulate the DOM to modify the page at run time.
  - The need to program functionally in JavaScript, refrains "foreign" developers from using JavaScript, because traditional imperative programming, esp. if I/O and other blocking operations are concerned, don't work any longer. – Instead callbacks or promises must be used, which makes the code looking strange in opposite to, e.g. C. JavaScript provides function literals to use code as data.
    - ECMAScript 6 added support for "real" lambdas, which supersede function literals is most places.
  - JavaScript also supports object-oriented programming:
    - Originally, JavaScripts provides prototype-based object-orientation, which does not use predefined classes.
    - JavaScripts type-system is dynamic.
    - ECMAScript 6 added support for class-based object-orientation.

Objective-C 2 (automatic reference counting, Cocoa of macOS 10.13)

```
#import <Foundation/Foundation.h>

int main(int argc, char *argv[]) {
        // check arguments from console
        if (2 <= argc) {
            // open and check file, read the complete file into an NSString
            NSError* fileError;
            NSString* logFileContents = [NSString stringWithContentsOfFile:@(argv[1]) encoding:NSUTF8StringEncoding error:&fileError];
            if (!fileError) {
                NSArray<NSString*>* lines = [logFileContents componentsSeparatedByCharactersInSet:[NSCharacterSet newlineCharacterSet]];
                NSRegularExpression* regex = [NSRegularExpression regularExpressionWithPattern:@"[^\"]*\"([^\"]*)\"\\s*\\(([^\\(]*)\\)" options:0 error:nil];
                NSMutableDictionary<NSString*, NSString*>* latestUpdates = [NSMutableDictionary<NSString*, NSString*> new];
                for (NSString* aLine in lines) {
                    // parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera RawCompatibility Update" (2)
                    NSArray* matches = [regex matchesInString:aLine options:0 range:NSMakeRange(0, [aLine length])];
                    if (1 == [matches count]) {
                        NSTextCheckingResult* result = matches[0];
                        if (3 == [result numberOfRanges]) {
                            NSString* appName = [aLine substringWithRange:[result rangeAtIndex:1]];
                            NSString* versionNo = [aLine substringWithRange:[result rangeAtIndex:2]];
                            // store the parsed data
                            if (!latestUpdates[appName] || [versionNo intValue] > [latestUpdates[appName] intValue]) {
                                latestUpdates[appName] = versionNo;
                            }
                        }
                    }
                }
                // output the collected data to console
                for (NSString* appName in [[latestUpdates allKeys] sortedArrayUsingSelector:@selector(localizedCompare:)]) {
                    NSLog(@"App: %@, Latest Update: %@", appName, latestUpdates[appName]);
                }
            }
        }
        return 0;
}
```


- Created by Brad Cox and Tom Love from about 1980 to 1986.
- C++ was not the only approach to extend C with oo-features:
  C++' extensions are Simula-based and Objective-C's (Obj-C) extensions are Smalltalk-based.
  - Obj-C adds a rather thin layer above the C programming language.
  - Originally, Cox was a Smalltalk programmer, but he was forced to program in C at the ITT Corporation's Programming Technology Center. He didn't like C, which leads him to programming a C-preprocessor, the OOPC (Object-oriented Preprocessor), it was implemented using bare Unix tool such as awk, sed and of course a C compiler. During that time another ancestor of C, C++, was not yet known to him. Cox and Tom Love founded their own company, Stepstone, which brought Obj-C to market.
- The origins of Obj-C can be found in the NeXTSTEP OS, therefor many Obj-C's types have the prefix "NS" (NeXTSTEP). The NS framework and its additions from Apple, which form the Cocoa framework, are very consistent, but it takes a while until one understands its "patterns".
- Sun and Apple together defined the OpenStep standard, which specified an API for communication between OS-kernel and applications.
- It is a modern oo language. A "fallback" to C or C++ is possible.
- Very efficient; one of the most efficient dynamically typed and C-based languages.
- The syntax needs to get used to.
- Virtually only dominant on macOS and iOS (well, and NeXTSTEP OS also).
  - Beginning with the Cocoa version of macOS 10.6, Obj-C 2 code can be compiled with or without garbage collection.
  - If Apple hadn't chosen Obj-C as first class development language on its platforms, Obj-C would have passed away meanwhile.
  - Meanwhile Apple introduced another development for its platform besides Obj-C: Swift. Swift seams to better suit to modern programming, because it adopts features from other popular languages (esp. a mix of all programming paradigms: imperative, procedural, functional and oo) and simplified memory management.
- Besides Cocoa on macOS etc. there also exists a free and platform-independent implementation of the OpenStep oo-interface named GNUStep. GNUStep adds many extensions to OpenStep adding similar features as Cocoa, esp. UI components.

Swift 4 (Cocoa of macOS 10.14)

```
import Foundation

// check arguments from console
if 2 <= CommandLine.arguments.count {
	do {
		let data = try String(contentsOfFile: CommandLine.arguments[1], encoding: .utf8)
		let lines = data.components(separatedBy: .newlines)
		var latestUpdates = [String : String]()
		let pattern = "[^\"]*\"(?<appName>[^\"]*)\"\\s*\\((?<versionNo>[^\\(]*)\\)"
		let regex = try! NSRegularExpression(pattern: pattern, options: [])

		for aLine in lines {
			// parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera RawCompatibility Update" (2)
			let matches = regex.matches(in: aLine, options: [], range: NSRange(location: 0, length: aLine.count))
			if 1 == matches.count {
				let result = matches[0]
				if 3 == result.numberOfRanges {
					let appName = (aLine as NSString).substring(with:result.range(at: 1))
					let versionNo = (aLine as NSString).substring(with:result.range(at: 2))
					// store the parsed data
					if nil == latestUpdates[appName] || Int(versionNo)! > Int(latestUpdates[appName]!)! {
						latestUpdates[appName] = versionNo
					}
				}
			}
		}
		// output the collected data to console
		for appName in latestUpdates.keys.sorted() {
			print("App: \(appName), Latest Update: \(latestUpdates[appName]!)")
		}
	} catch {
		print("error")
	}
}
```


- Swift was introduced in 2014.
- Apple invented Swift as newer language (compared to Obj-C) for iOS/macOS/tvOS/watchOS.
  - In comparison to Obj-C Swift's readability is much clearer and simpler, where Swift's run time performance is at least on par with Obj-C.
- The current (2016) problem of Swift is, that its syntax is still evolving and many modifications in past, lead to serious source-incompatibilities.
- However, Apple's Swift documentation is excellent.
- Application areas:
  - Swift is suited for beginners, but mostly a language for switchers from Obj-C, because some of Swift's constructs address working with Obj-C interfaces to use present libraries.

Java 12 – Part I

```
import java.nio.file.*;
import java.util.*;
import java.util.regex.*;

public class Program {
	public static void main(String[] args) throws Exception {
		// check arguments from console and check the file
		if (0 < args.length && Files.exists(Paths.get(args[0]))) {
			// open the file, read all lines
			final Map<String, String> latestUpdates = new TreeMap<>(String.CASE_INSENSITIVE_ORDER);

			final Pattern parseRex = Pattern.compile("[^\"]*\"(?<appName>[^\"]*)\"\\s*\\((?<versionNo>[^\\(]*)\\)");

			for (final String aLine : Files.readAllLines(Paths.get(args[0]))) {
				// parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital
				// Camera Raw Compatibility Update" (2)
				final Matcher matcher = parseRex.matcher(aLine);
				if (matcher.matches()) {
					// store the parsed data
					final String key = matcher.group("appName");
					final String value = matcher.group("versionNo");
					if (!latestUpdates.containsKey(key) || Integer.parseInt(value) > Integer.parseInt(latestUpdates.get(key))) {
						latestUpdates.put(key, value);
					}
				}
			}
```


- Designed by James Gosling for embedded devices at Sun Microsystems (now Oracle) in 1995. So it is a language created by a hardware company like Smalltalk!
  - Java was inspired by Smalltalk and C++.
- The name "Java" is from "Java coffee", but it was randomly chosen. Java has nothing to do with JavaScript.
- Java is the main programming language of the Java platform.
- (Java solves the same problems .NET does.)
- Java is multi paradigmatic, but mostly oo with a mighty class library. The class library is built on oo design patterns, which leads to a very clean design.
- Compiled Java code (byte code) is platform independent and can run on any OS that provides an interpreter for the byte code.
- (A good way to learn Java for absolute beginners is the BlueJ IDE and belonging to tutorials.)
- Application areas:
  - Common application programming tasks.
  - Web programming: applets, servlets and JSP (servlets and JSP are parts of "enterprise Java" (JEE)).
  - Good acceptance, because there is a good documentation available as well as free IDEs (Eclipse, NetBeans etc.).
  - The Blu-ray Disc (BD) standard defines Blu-ray Disc Java (BD-J) as framework for the application layer for BD (e.g. for menu-driven applications). Therefore Java is highly integrated into BD-players.
  - The progress of Java is somewhat slow because of the Java Community Process (JCP), where the community decides about language changes. For C# the progress is faster, because there are less deciders (mainly only Anders Hejlsberg).
    - However, this is getting better right now, actually, it is promised to deliver a new java version every 6 months.
  - Some Java APIs are very elaborate and mighty, because of this, these APIs are sometimes said to be "over engineered".

```
                // output the collected data to console
                for (final String appName : latestUpdates.keySet()) {
                      System.out.printf("App: %s, Latest Update: %s %n", appName, latestUpdates.get(appName));
                }
          }
    }
}
```

C# 6 – Part I

```
using System;
using System.Text.RegularExpressions;
using System.Collections.Generic;
using System.IO;

public class Program
{
	public static void Main(string[] args)
	{
		// check arguments from console
		if (0 < args.Length)
		{
			IDictionary<string, string> latestUpdates = new SortedDictionary<string, string>();
			Regex parseRex = new Regex(@"[^\""]*\""(?<appName>[^\""]*)\""\s*\((?<versionNo>[^\(]*)\)");
			// open and check file, read a line
			foreach (string aLine in File.ReadAllLines(args[0]))
			{
				// parse the line: e.g. 2009-04-06 12:42:58 +0200: Installed "Digital Camera Raw Compatibility Update" (2)
				Match match = parseRex.Match(aLine);
				if (match.Success)
				{
					// store the parsed data
					string appName = match.Groups["appName"].ToString();
					string versionNo = match.Groups["versionNo"].ToString();
					if (latestUpdates.ContainsKey(appName))
					{
						if (int.Parse(versionNo) > int.Parse(latestUpdates[appName]))
						{
							latestUpdates[appName] = versionNo;
						}
					}
					else
```


- Created by Microsoft and mainly designed by Anders Hejlsberg in 2000. Interestingly Hejlsberg worked for Borland until he went to Microsoft in 1996. At Borland he created Turbo Pascal and Delphi. Many features of Delphi have been transferred to C#!
- C# (not necessarily the newest version of C#) is standardized in ECMA-334 and in ISO/IEC 23270:2006.
- Meanings of the name "C sharp": a "sharp" kind of the C programming language or C with four pluses.
- It's the main programming language of the "language diverse" .NET platform.
- Modern multi-paradigm language, a mixture of Java and C++ in syntax. It looks like Java with some additions to satisfy legacy Win(32) stuff (the legacy stuff is more or less a .NET-platform thing and not directly related to C# as a language).
- C-style syntax, simple to use.
- Application areas:
  - All kinds of application systems.
  - Good acceptance, because there is a good documentation available as well as some high quality IDEs allowing Rapid Application Development (RAD). (RAD was already present in Visual Basic (VB) and Delphi, its success is continuing in IDEs for C#.)

```
                        {
                            latestUpdates.Add(appName, versionNo);
                        }
                    }
                }
                // output the collected data to console
                foreach (KeyValuePair<string, string> item in latestUpdates)
                {
                    Console.WriteLine($"App: {item.Key}, Latest Update: {item.Value}");
                }
            }
        }
    }
```

Phase 5: Program Test

- After having the coded processing approach and the code also compiles successfully, we have to check the program.
- The test-input data should cover extreme and edge cases, which make up critical data:
  - wrong data, wrong/unexpected format
  - "Update Software.log" is empty or very large
  - infrastructure problems: "Update Software.log" not existent, or we have no read access
- To check the program, we can use value tables and the code to make dry runs.
  - With a dry run we check the correctness of a program mentally, with a value table. I.e. we do not use a computer!
- Then the real software product will be tested on a real PC:
  - Tests on the supported operating systems (OS).
  - Different input data, incl. extreme and edge cases under "real" conditions.
  - In this phase of testing it is important, that people different from the original implementors perform the tests.
  - => Such tests are usually done by special personnel of the quality assurance department.

Phase 5: Program Test – Dry Run and Value Table

- Now we'll make a dry run of this simplified part of our program (e.g. w/o output, but only the processing algorithm):

```
1:    do {
2:          std::getline(logFile, aLine);
3:          std::cmatch res;
4:          if (std::regex_search(aLine.c_str(), res, regex)) {
5:                if (latestUpdates.end() == latestUpdates.find(res[1]) || std::stoi(res[2]) > std::stoi(latestUpdates[res[1]])) {
6:                      latestUpdates[res[1]] = res[2];
7:                }
8:          }
9:    } while (logFile.good());
```

- Input values: a smaller version of the "Software Update.log" file with these two lines:

```
2009-04-06 12:43:04 +0200: Installed "GarageBand Update" (5)
2009-04-06 12:44:34 +0200: Installed "iMovie Update" (8)
```

- Following the program execution would result in such a value table.

Executed Program Line	Variables	Values	Conditions
2	*aline*	"2009-04-06 12:43:04 +0200: Installed \"GarageBand Update\" (5)"	
4	*res[1]* *res[2]*	"GarageBand Update" "5"	
5	*latestUpdates*	{empty}	*latestUpdates.end() == latestUpdates.find(res[1])* is true
6	*latestUpdates*	{"GarageBand Update" : "5"}	
2	*aline*	"2009-04-06 12:44:34 +0200: Installed \"iMovie Update\" (8)"	
4	*res[1]* *res[2]*	"iMovie Update" "8"	
5	*latestUpdates*	{"GarageBand Update"}	*latestUpdates.end() == latestUpdates.find(res[1]) is true*
6	*latestUpdates*	{"GarageBand Update" : "5", "iMovie Update" : "8"}	

Phase 5: Program Test – Dry Run and Value Table

- Benefits
  - It could be done without a computer.
  - Can be used for documentation.
- Downsides
  - It is a lot of work! E.g. a line of code could do many algorithmic things at once.
  - Esp. syntactic errors can almost not be spotted.
- Today we rather use:
  - (1) Log/trace messages, which print the values of variables of an algorithm to standard output, a file or a graphical user interface.
    - "printf()-debugging"
    - A downside of this approach is, that adding new messages requires to change the code and to be able to change the code at all.
  - (2) A debugger to see the values of an algorithm during it is running on the computer (i.e. no dry run, but a "wet" run).
    - A debugger allows line-by-line step-through execution of source code while the program is running, to examine or even modify variables.
    - There exist visual debuggers for IDEs and non-visual debuggers for console usage, e.g. on remote or customer machines.
    - Debuggers are easy and fast/spontaneous to use, esp. they work in "real time".
    - Most debuggers also allow to add logging messages, without changing the code at all!
  - Nowadays, log/trace outputs or debuggers (in IDEs) should be used in favor to dry runs and value tables.

- Some older Basic dialects allowed to turn on the "trace mode". Tracing then output the number of the executed line on the screen respectively. - Tracing was esp. needed to check the correctness of gotos, which tended to be incorrect quite often. The trace mode was activated with the TRON (TRace ON) command, the term "TRON" is actually the inspiration of the science fiction movie "Tron" in 1982. (Tracing was deactivated with the TROFF (TRace OFF) command.)

Phase 6: Program Documentation

- Technically, for other developers to allow maintenance and further development:
  - The initial user story.
  - The symbolic program code itself.
  - Flowcharts and NSDs.
  - Test plans, test input data

- User-facing documentations:
  - Installation documentation
  - User manual.

- => To be frank software engineering is a holistic activity and spans other phases below phase 1 and above phase 6.

Roots of C++ – Part I

- Dr. Bjarne Stroustrup wanted to code a simulation of a distributed computer system (sending messages etc.):
  - In the 1970ies he started this project using Simula 67, an object-oriented programming language.
  - Basically, Simula was too slow for the task (some say due to Simula's garbage collector) and the project was going to fail.
- Simula was appropriate to formulate a simulation, but its execution time was too slow. => It was the wrong tool for the task!
- Therefor, Dr. Stroustrup reimplemented the system in BCPL (Basic Combined Programming Language)
  - BCPL was an predecessor of the programming languages "B" and "C".
- BCPL was fast enough at execution time, but programming BCPL, esp. programming this simulation, was terrible.
- After these experiences, Stroustrup experimented with C.
  - C is a successor of BCPL.
  - C adds the concept of a (small) type system to BCPL.
  - C and BCPL have in common a very fast execution time.
- He decided to extend C with means to better support writing simulations with it. => He created his own tool!

48

- Simula provides classes and polymorphisms, concepts, which were included into C++ as well. Additionally, Simula provides technologies to support communication patters via queues and coroutines – excellent features to code distributes systems!
- Bjarne Stroustrup is Danish. Simula was designed by a Norwegian team, mainly Ole-Johan Dahl and Kristen Nygaard.
- In BCPL basically only used processor word to store data, the context this data was used determined the type, e.g. an integer or pointer etc. This is also the reason, the compiler was so small, simple and fast. – But it came to the prize, that programing was not type safe: if data was used in the wrong context, this could lead to a crash!
- C++ was not the only way to add oo extensions to C. C++ is a C-extension based on Simula; Objective-C is a C-extension based on Smalltalk. In short we'll also discuss Objective-C a little.

Roots of C++ – Part II

- Stroustrup extended C with the support of object-oriented (oo) features in the type system by so-called classes.
  - The class concept was borrowed from Simula.
  - Stroustrup created a frontend for C, cfront, which translated the extended C into plain C.

- The new language, i.e. the extended C was named "C with classes" in 1979.

- In 1983, "C with classes" was renamed to C++. (This was an idea from Rick Mascitti, one of Stroustrup's assistants.)
  - In 1985, cfront was published as a commercial C++ compiler.

- C++ needed a long evolution from 1979 to C++98 as a standard.
  - It took so long, because C++' design is driven pragmatically. – It wasn't created on the green table my mathematicians.
  - C++ is standardized in ISO/IEC 14882.
  - In the beginning of the 1990ies, i.e. before the standard was finished, the usage of C++ already took of.
    - In 1990 the book "The Annotated C++ Reference Manual" (ARM) (Margaret A. Ellis, Bjarne Stroustrup) was released.
    - Also C++' integration of oo was a driver for C++' success: In the 1990ies there was an "oo-hype", which was boosted by C++.
  - The standardization of new features is still in progress.

- That C++ was adopted pre-standard is due to its pragmatic design, we will discuss this concept right now.

Bjarne Stroustrup's Philosophy

- As pragmatic language, it was designed after Stroustrup's philosophy of empiricism:
  - Empiricism is insight by experience. Concepts, that work in the reality stay in.
  - Experience teaches, what people need – even if Stroustrup don't like it.
  - Concepts must survive in the reality and people should not be forced to use a specific solution.
  - Esp. the decision to use C as a basis was driven by experience: C just had a high acceptance this time.

- Some examples of the spectrum of philosophies:
  - Empiricism: insight by experience (small everyday-problems are more important than cognitive concepts)
  - Rationalism: insight by reason, thinking and guidelines
  - Idealism: insight by ideas, perception is a projection of ideas (there are no perceptions, but only cognitive concepts)

- The pragmatic approach in C++ also shows in the bare bandwidth of features it offers:
  - Macros for the module system and code generation.
  - Working with raw memory via low-level arrays and pointers.
  - Maximal abstraction and semantics is possible with templates.
  - However, an aim of C++: There should be no further language between C++ and machine language, neither now nor in future.

- Concepts must survive in the reality check, some languages like, e.g. Dylan didn't pass this reality check.
- Idealists may force people to do things, because they think the have the "Holy Grail" they only want "the best". This may lead to catastrophes: innocent people will suffer and it will lead to delusion and perdition of the idealists. – This is the experience of the European history.

C++ is a "Superset" of C

- Early C++ compilers translated C++ to C; the resulting C was then compiled.
- C++ is more robust than C.
  - C++ is type-safer than C, e.g. pointer types are strictly typed.
  - Functions must be declared in C++ before they can be called.
  - C++ offers alternatives for C's unsafe macro operations.
- But C++ is more complex than C.
  - One gets C programs easier to compile than C++ programs.
- C++ adds means of abstraction and object-orientation to C.
  - Classes, abstraction, encapsulation, inheritance and polymorphism are available.
  - STL types add utilities to C++, so that it can be used for many tasks.
- C++ compilers are very much slower than C compilers, because parsing the complex language is difficult.
- Some C++-Features were back-added to C: Stroustrup says, that C++ is the better C!

- The early Cfront compiler, which translated C++ code to C code, was abandoned in 1993, because some C++ features could not be easily implemented. → C++ has grown to complex!
- Meanwhile C++ is no longer a superset of C, because after C++ "branched" from C, C was extended with many new features not shared by C++.

Excursus: The Objective-C Story in Short

- Originally, Brad Cox was a Smalltalk programmer, which worked at the ITT Corporation.
- At the ITT's Programming Technology Center he was forced to program in C.
- He didn't like C, but was pretty convinced of the features in Smalltalk, esp. the object-oriented features.
- This leads him to program a C-preprocessor, the Object-Oriented Preprocessor (OOPC) in 1980.
  - It was implemented using bare Unix tools such as awk, sed and of course a C compiler.
  - During that time another ancestor of C, namely C++, was not yet known to him.
- Cox and Tom Love founded their own company, Stepstone, which brought Objective-C to market in 1984.
- In fact, Objective-C (1980-1984) was present before C++ (1983-1985), but people went to C++.
  - All C-programmers know the book "The C Programming Language" (Brian W. Kernighan and Dennis Ritchie, 1978).
  - Stroustup's book "The C++ Programming Language" (1986) was written in the same style (same format, font formatting, chapter 0).
  - => The present literature that time brought C programmers rather to C++ that to Objective-C.

- ITT is for International Telephone & Telegraph.
- The book "The C Programming Language" is often just called the "K&R book" after the authors' initials.

Some C++ Features

- A C++ distribution provides libraries (standard-library, h-files, lib-files, dlls), compiler and linker (sometimes debugger).
- C++ code can be platform independent, but the resulting program can be natively compiled to a specific platform.
- Top features of C++:
  - Generic programming with templates to program mighty and high performance libraries.
  - Class-based object-orientation to program maintainable abstractions.
  - Control of the life time of objects to program resource saving and reliable code.
- Versatility:
  - 1. Multi-paradigm: imperative, procedural, functional, object-oriented and generic.
  - 2. From low-level (embedded: registers and raw memory) to high-level (applications).
  - 3. Abstraction: from compile time evaluated templates to run time executed Run Time Type Information (RTTI) and dynamic_cast.

C++ Versatility:
C++ is a language you can write lyric poetry or a legal text.

- C++ provides no graphical or GUI features, other frameworks go there:
  - Microsoft: GDI+, MFC
  - Trolltech: QT
  - Apple: Cocoa
  - GNU/Linux: GTK+, Motif

- An important component of C++' standard library is the Standard Temple Library (STL), which is a fully library based on C++' templates.

C++ – Plainly Spoken...

- C++ bugs are easy to produce, but hard to find.
  - (The problem in C++ is the "C".)
  - C++ has undefined behavior in some of its corners. (However, less undefined behavior than in C.)
  - There are language exceptions, but no standard runtime.
  - Dynamic memory must be freed manually.
  - There exist only few guarantees.
  - Good experience, patience and tools are required.

Disapproving saying:
"If C++ was a hammer, every problem looks like a thumb."

- Why one should learn it?
  - C/C++ help to understand low-level mechanisms.
  - C/C++ are the most efficient high-level programming languages available.
  - C/C++ and its programmers have a high reputation in the market.
  - A C/C++ compiler is contained in many Linux distributions, to compile the code of a component instead to load it as RPM.
    - E.g. to compile the kernel after one added libraries.
    - Also the make-system is closely related to C/C++.
  - We can use a lot of other C-code or libraries.

- As programming languages, C++ and esp. C, are very terse and compact in syntax.
- Often, there is no agreement about what a string is, is it a 0-terminated char[], or is it a const char*?.

Learning C++

- Learning C++ is like learning martial arts:
  - After a year you think you can handle C++, but you learned so many tricks in the mean time, that old code looks weird.
  - You are collecting more and more belts and learn aspects of the language. – A master's degree is possible after 3-4 years.
- There exist no common conventions in C++!
  - The C++ compiler were most often behind the standard, people didn't know, how to use features "the best way".

- Comparing learning C++ is appropriate. People, which have a master's degree or dan can move very elegantly and fall without hurting themselves and smash brick stones without effort. They hold their style for the only applicable one, and this is ok, because they spend their whole lifetime improving their skills. – However, experience shows, that those people often fail in street fights, because their opponents might not the same "fine" styles and make the better fight.

What kind of Tool is C++? – Which Problems can it solve?

- One of the most versatile languages on the market.
- Hardware-near code (drivers etc.) and embedded systems.
- Complete operation systems.
- Very large/complex realtime systems.
- Desktop applications.
- Plug-In and component technologies (MS Office, COM, ActiveX etc.).
- Where is C++ not an appropriate choice?
  - Security critical applications. C++ is very hardware near, most stack overflow-based vulnerabilities are due to C++ software.
  - Servers written in C++ are problematic. C++ is notorious for many memory leaks (even very proficient programmers produce them).
    - The problem is, that servers run for a very long time and memory leaks could accumulate, until the server crashes.
    - However, it should be said, that server architectures are meanwhile run in a way, that servers are redundant and are automatically started, if crashed.

- As programming languages, C++ and esp. C, are very terse and compact in syntax.
- All currently available OS' are written in assembly and/or C and/or C++ (and/or one of C/C++' dialects).
- There exists "embedded C++", which is C++ lacking some features, e.g. templates.

C++ has Siblings and or Successors

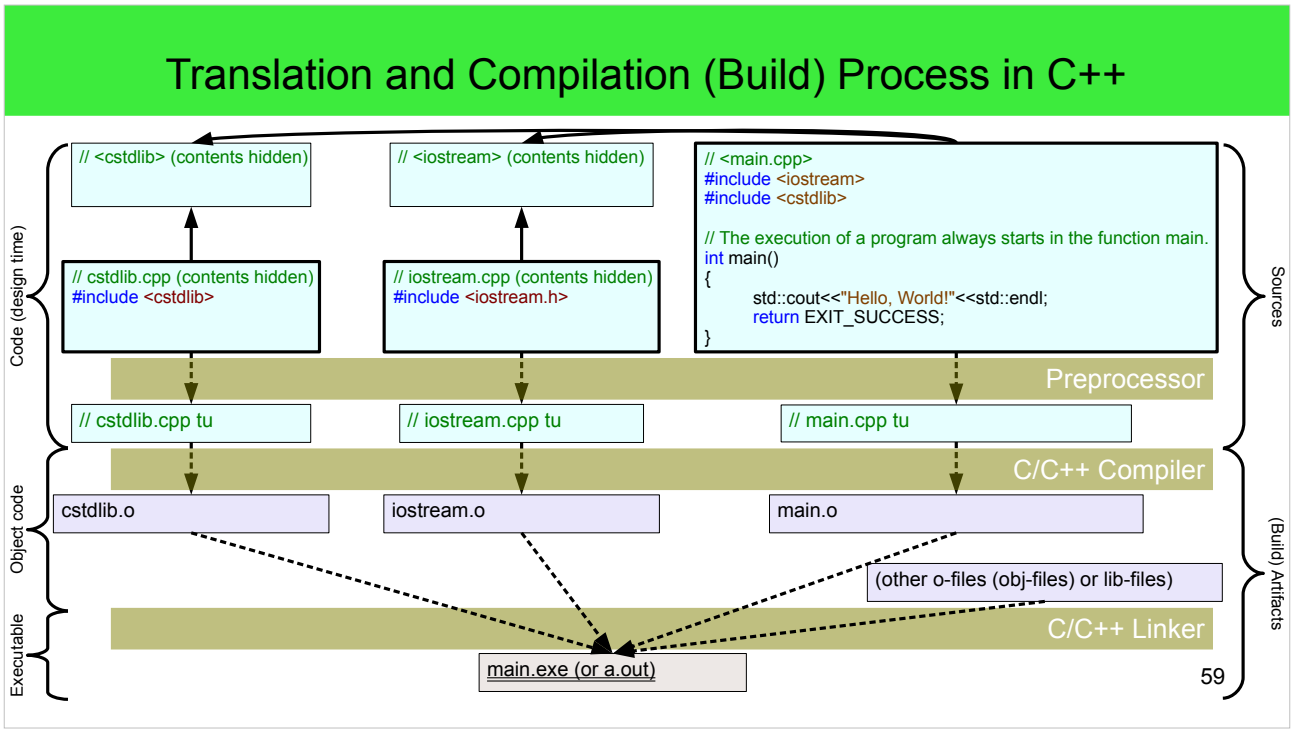
- C++17 (2017)
  - The current version, but we'll use C++ 98 in this course.
- Objective-C (1986)
  - A combination of C and Smalltalk. Used for Cocoa programming on macOS.
- C++/CLI (Common Language Infrastructure) (2005)
  - Microsoft's extended variant of C++ for .NET development.
- C++/CX (Component Extensions) (2011)
  - Microsoft's variant of C++ primarily for COM development on WinRT (Windows 8).
- D (2001)
  - Digital Mars' further development of C++.

Our First C++ Program

```
// <main.cpp>
#include <iostream>
#include <cstdlib>


// The execution of a program always starts in the function main.
int main()
{
	std::cout<<"Hello, World!"<<std::endl;
	return EXIT_SUCCESS;
}
```


- What's "*std::cout*" and "*EXIT_SUCCESS*" and where does it come from?
- Show this in the IDE:
  - Create a new project.
  - Create a new code file and write source code into that file.
    - C/C++ source code is written/organized in text files.
  - Start the program in the console.
  - Start the program in the debugger.



- What is a compiler?
  - C/C++ source code is written/organized in text files.
  - The compiler checks the C/C++ syntax and translates the C/C++ code into machine code.
  - What's the "opposite"? What's the difference?
  - Interpreted programming languages are getting a greater appeal, because code can be quickly changed, being effective w/o a compilation/linking phase. Interpreting also opens a door to platform independent code. Some languages/platforms use a combination of compiled and interpreted code (Java, .NET).
- What kind of code do we have on both sides of the "compiler line"?
  - Source code (textual code) vs. binary code (object code or machine code).
- The difficulty to find errors in a C/C++ program increases with the phases: Compile time, link time and run time. Because in each phase we've a greater "logical distance" to the original source code.
- Also important is the aspect of separated compilation; it will be discussed in a future lecture.

Books on C++

- There exists a plethora of books...
- The C/C++ language:
  - Recommended Book for this course: Bruce Eckel, Thinking in C++ Vol I
  - Bjarne Stroustrup, The C++ Programming Language
- Advanced and industry quality C/C++:
  - Bruce Eckel, Thinking in C++ Vol II
  - John Lakos, Large Scale C++ Software Design
- Computing in general:
  - Charles Petzold, Code
  - Rob Williams, Computer System Architecture

Thank you!