

Algorithms and Programming IV

Determinism

Summer Term 2023 | 24.04.2023
Barry Linnert

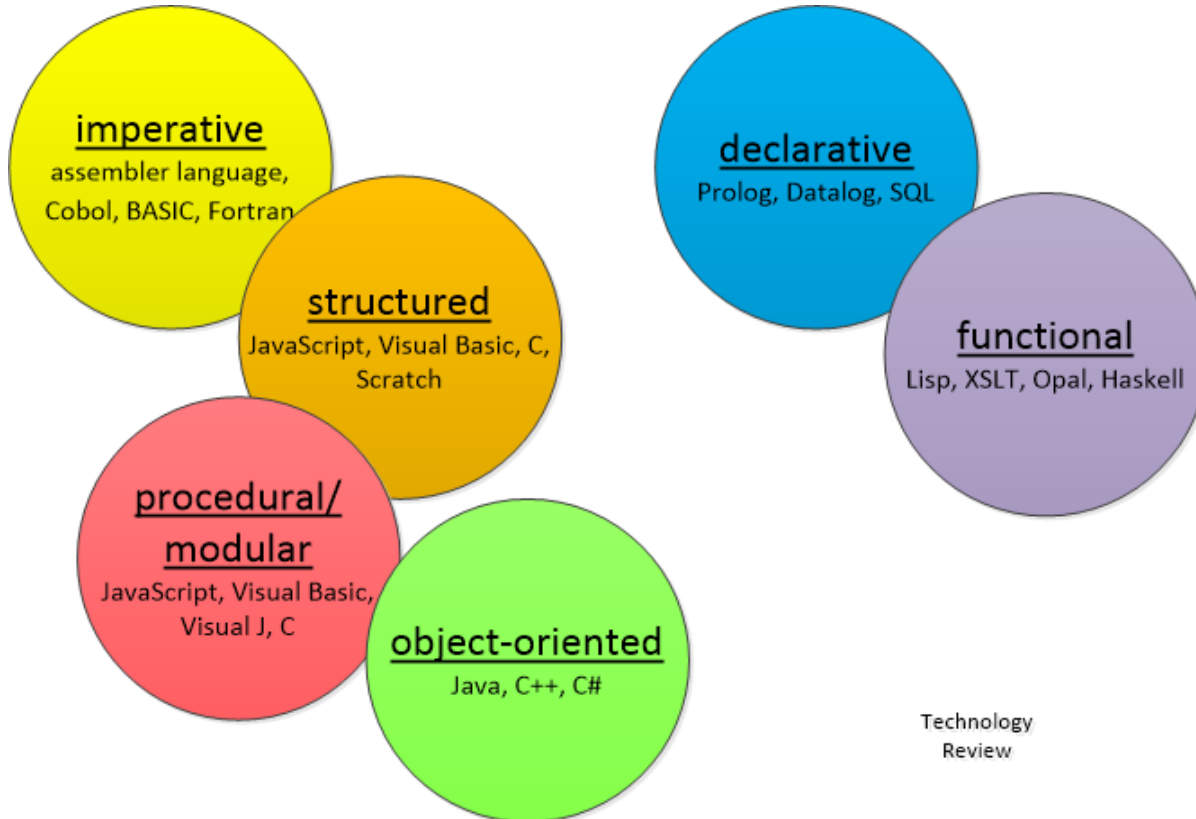
Objectives of Today's Lecture

- Defining the Requirements on Programming
- Specifying the Meaning of Correctness
- Identifying the sequential Processing as Foundation of Determinism
- Exploring existing Models to support Programming

Concepts of Non-Sequential and Distributed Programming

DETERMINISM

Programming Paradigms





Requirements for Programs

- A program should do what it is expected to do!
 - Functional requirements, such as
 - Scope of functions
 - Correctness

- A program should comply with certain requirements about its behavior.
 - Non-functional requirements, such as
 - Performance
 - Usability
 - Security
 - ...

Functional Characteristics

- Scope of Service
 - Requirements elicitation, design, project management, quality assurance
 - → LV (lecture course) Software Engineering 
- Correctness
 - Hoare logic (calculus), Simulation via pen and paper (Handsimulation) and Testing
 - select input values
 - apply algorithm
 - execute program
 - compare results
 - Problem: it works with existing program only 

Foundations of Building Correct Programs

1. Amount of operations changing the current state of the system in a known way
 - We have an idea (model) of the *change of state of the system* by the operation in a specific context (previous state of the system).
 - The operations are identified explicitly by a command (of the current programming language).
 - So, we do have a model of the execution (change of state of the system) of the commands.
2. Selection and order (sequence) of the operations (represented by the commands) based on the algorithm to be implemented
 - By connecting the operations together we build a model of the execution of the program.
 - The sequence of the commands represents this model of program execution.

Ordered Execution of a Set of Commands

- Sequential processing of the commands using numerical ordering
- Example BASIC

```
10 REM This is a test program
20 PRINT "Hello World!"
30 INPUT "Please enter your name"; A$
40 PRINT "Hello, "; A$
```

How about other (imperative) programming languages?

Sequential Processing

- Example C

```
// first C-program

int main (void)
{
    int a = 0;
    int b = 0;

    a = 2;
    b = 3;

    a = a * b;

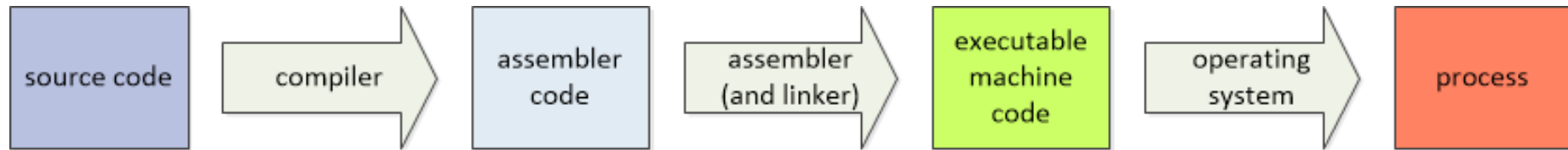
    return a;
}
```

- How is sequential processing ensured?

Programming

- With selection and the sequential order of the operations the state of the system and the state change can be anticipated in our minds.
 - That's called Programming model!
 - Building a (correct) programming model is called Programming.
 - It's the same approach as to check correctness of the program.
 - So, building a correct execution model of the program ensures the implementation of a correct program.
 - ... but is only the first step – Testing, testing, testing is also needed.
- But...
 - ... how is it ensured that the sequence of commands will be executed as expected?
 - ... where does the knowledge about the change of system's state by the commands comes from?

From Program to Process



From Program to Machine Program

<pre> // first program in C int main (void) { int a = 0; int b = 0; a = 2; b = 3; a = a * b; return a; } </pre>	<pre> 00000000 <main>: 0: e52db004 push {fp}; (str fp, [sp, #-4]!) 4: e28db000 add fp, sp, #0 8: e24dd00c sub sp, sp, #12 c: e3a03000 mov r3, #0 10: e50b3008 str r3, [fp, #-8] 14: e3a03000 mov r3, #0 18: e50b300c str r3, [fp, #-12] 1c: e3a03002 mov r3, #2 20: e50b3008 str r3, [fp, #-8] 24: e3a03003 mov r3, #3 28: e50b300c str r3, [fp, #-12] 2c: e51b3008 ldr r3, [fp, #-8] 30: e51b200c ldr r2, [fp, #-12] 34: e0030392 mul r3, r2, r3 38: e50b3008 str r3, [fp, #-8] 3c: e51b3008 ldr r3, [fp, #-8] 40: e1a00003 mov r0, r3 44: e24bd000 sub sp, fp, #0 48: e49db004 pop {fp}; (ldr fp, [sp], #4) 4c: e12fff1e bx lr </pre>
---	---

From Program to Machine Program

```
// first program in C
```

```
int main (void)
{
    int a = 0;
    int b = 0;

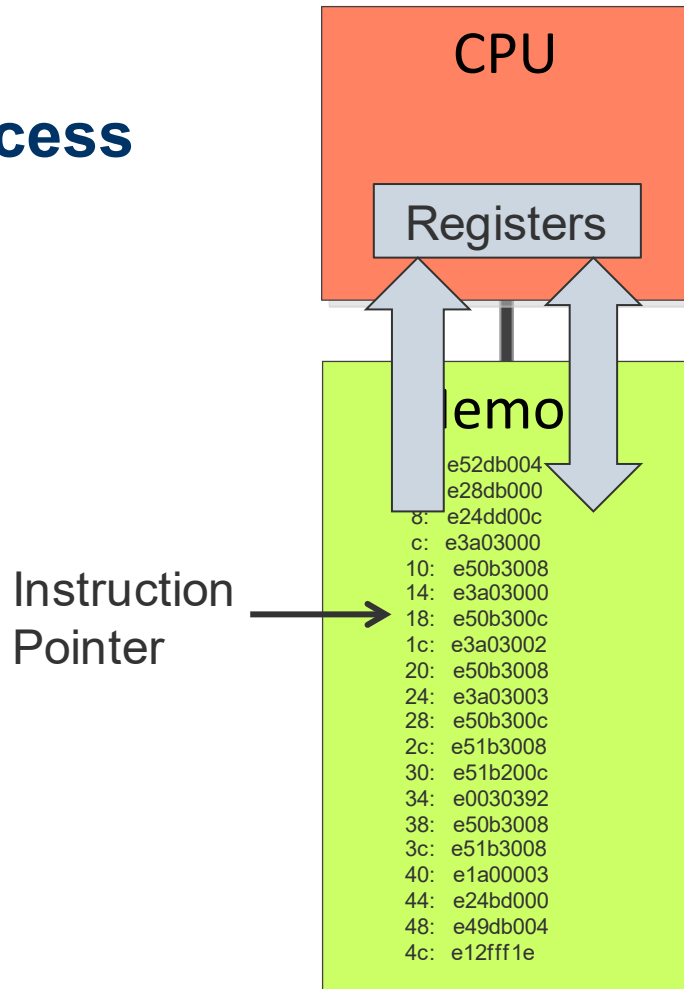
    a = 2;
    b = 3;

    a = a * b;

    return a;
}
```

```
00000000 <main>:
    0:  e3a00006      mov     r0, #6
    4:  e12fff1e      bx     lr
```

Process



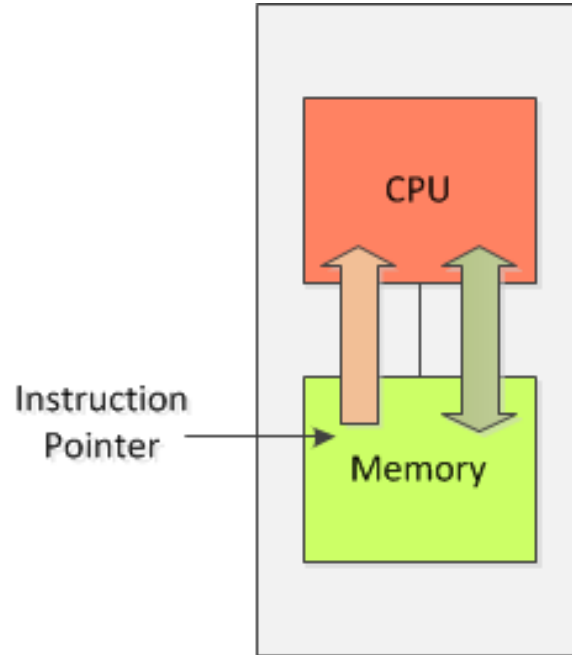
The automatic incrementation of the Instruction Pointer ensures the sequential execution of the instructions.

Execution of the Program

- All of the following components have work correctly to ensure that the commands will be executed as expected and in the determined sequence:
 - Compiler or Interpreter
 - Translates the commands to machine instructions
 - Linker
 - Builds a memory image with machine instructions and initial data
 - Operating system
 - Sets up the process out of the linked image
 - Sets up the execution environment
 - Hardware
 - Executes the machine instruction in an defined environment (by the OS)
 - Increments the Instruction Pointer automatically

Machine Model

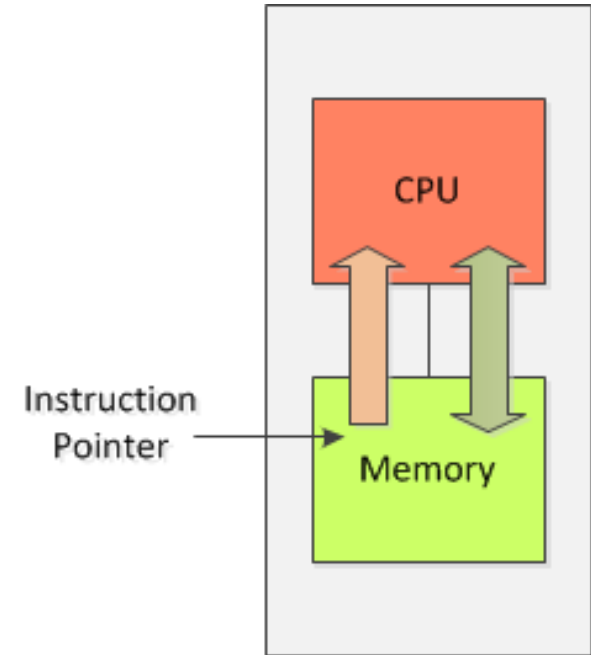
- The representation of the technical workflow
 - von Neumann architecture



wikipedia.org

Relations between the Models

- If there is a way to transform all of the models from programming model to execution model based on the machine model, then the correct execution of the algorithm implemented by the programming model can be assured.
- The transformation allows us to make statements about the behavior of the system controlled by the execution of our program.
- Based on these statements we can (mentally) anticipate the program execution and write our program.



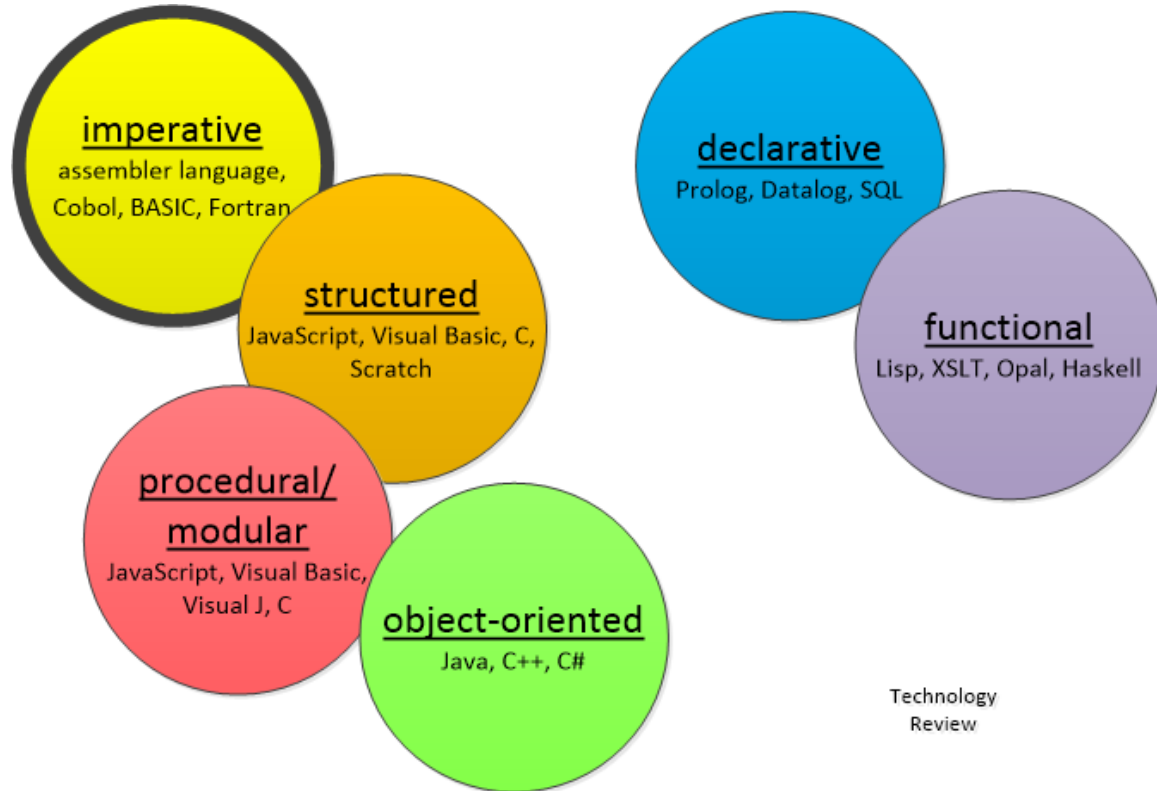
Determinism

- The (mental) anticipation of the program execution holds for all of the runs of our program.
- Definition: A **deterministic algorithm** is an algorithm which, given a specific input, will always produce the same specific output and will always fulfill changes to the state of the system in the same way (and order).
 - Using all of the models we can decide if an algorithm is deterministic.
 - If every step of transferring one model to another is performed correctly a deterministic algorithm is correctly implemented and the program will be executed correctly.
 - This program can be called a deterministic program.

Sequential Processing

- One foundation for deterministic execution of the programs is the sequential processing of operations, commands and instructions.
- Can it be assumed to have a sequential processing?

Sequential Processing and Programming Paradigms



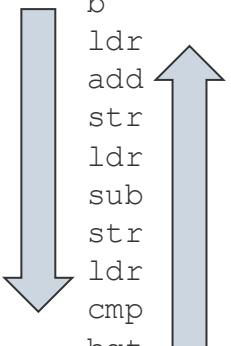
Technology
Review

Sequential Processing in further Programming Models – Loops

```
// Program in C with  
// while loop
```

```
int main (void)  
{  
    int a = 0;  
    int b = 0;  
  
    b = 3;  
  
    while (b > 0)  
    {  
        a += 2;  
        b--;  
    }  
    return a;  
}
```

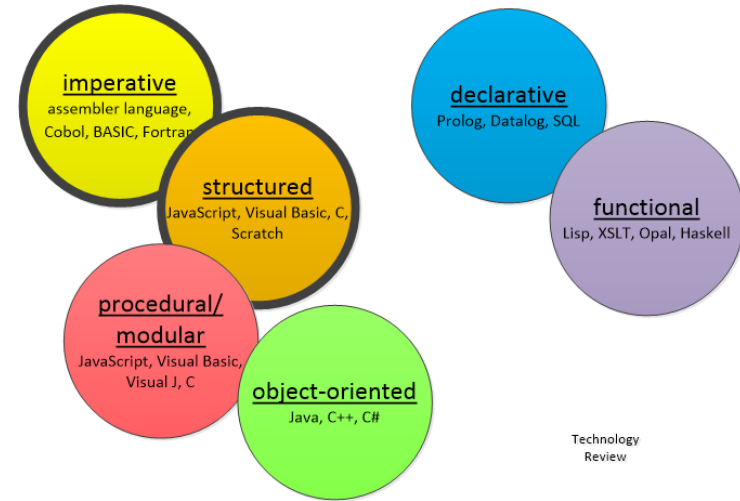
```
00000000 <main>:  
    0:     ...  
    c:   e3a03000        mov     r3, #0  
    10:  e50b3008        str     r3, [fp, #-8]  
    14:  e3a03000        mov     r3, #0  
    18:  e50b300c        str     r3, [fp, #-12]  
    1c:  e3a03003        mov     r3, #3  
    20:  e50b300c        str     r3, [fp, #-12]  
    24:  ea000005        b      40 <main+0x40>  
    28:  e51b3008        ldr     r3, [fp, #-8]  
    2c:  e2833002        add    r3, r3, #2  
    30:  e50b3008        str     r3, [fp, #-8]  
    34:  e51b300c        ldr     r3, [fp, #-12]  
    38:  e2433001        sub    r3, r3, #1  
    3c:  e50b300c        str     r3, [fp, #-12]  
    40:  e51b300c        ldr     r3, [fp, #-12]  
    44:  e3530000        cmp    r3, #0  
    48:  cafffff6        bgt    28 <main+0x28>  
    4c:  e51b3008        ldr     r3, [fp, #-8]  
    50:  e1a00003        mov    r0, r3  
    54:  ...
```



Sequential Processing in further Programming Models – Loops

```
// Program in C with  
// while loop
```

```
int main (void)  
{  
    int a = 0;  
    int b = 0;  
  
    b = 3;  
  
    while (b > 0)  
    {  
        a += 2;  
        b--;  
    }  
    return a;  
}
```



Sequential processing of commands and instructions is provided even with loops and jumps or conditional execution.

Sequential Processing in further Programming Models – Functions

```
// Program in C with function
```

```
int foo (int a, int b)
{
    return a * b;
}
```

```
int main (void)
{
    int a = 0;
    int b = 0;
    a = 2;
    b = 3;

    a = foo (a, b);

    return a;
}
```

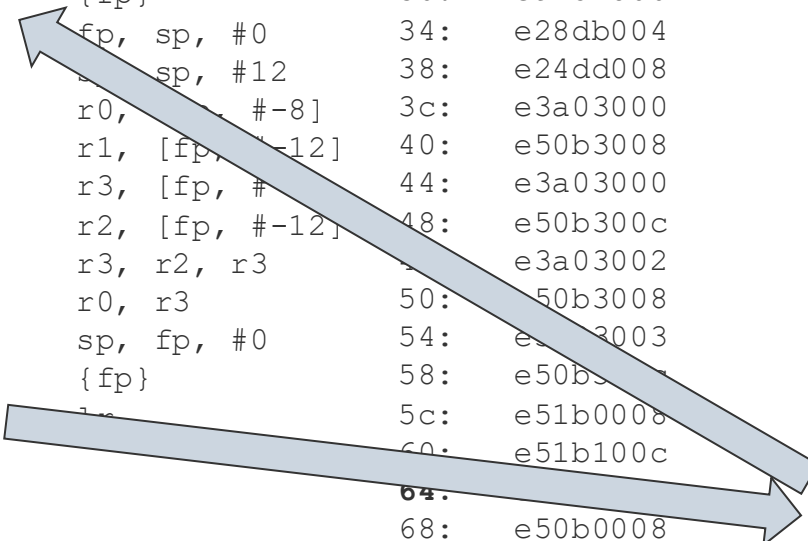
Sequential Processing in further Programming Models – Functions

```
00000000 <foo>:                                00000030 <main>:
  0:   e52db004      push   {fp}                                30:   e92d4800      push   {fp, lr}
  4:   e28db000      add    fp, sp, #0                          34:   e28db004      add    fp, sp, #4
  8:   e24dd00c      sub    sp, sp, #12                          38:   e24dd008      sub    sp, sp, #8
  c:   e50b0008      str    r0, [fp, #-8]                       3c:   e3a03000      mov    r3, #0
 10:  e50b100c      str    r1, [fp, #-12]                      40:   e50b3008      str    r3, [fp, #-8]
 14:  e51b3008      ldr    r3, [fp, #-8]                       44:   e3a03000      mov    r3, #0
 18:  e51b200c      ldr    r2, [fp, #-12]                      48:   e50b300c      str    r3, [fp, #-12]
1c:  e0030392      mul    r3, r2, r3                          4c:   e3a03002      mov    r3, #2
20:  e1a00003      mov    r0, r3                              50:   e50b3008      str    r3, [fp, #-8]
24:  e24bd000      sub    sp, fp, #0                          54:   e3a03003      mov    r3, #3
28:  e49db004      pop    {fp}                                58:   e50b300c      str    r3, [fp, #-12]
2c:  e12fff1e      bx     lr                                  5c:   e51b0008      ldr    r0, [fp, #-8]
                                           60:   e51b100c      ldr    r1, [fp, #-12]
                                           64:   ebfffffe      bl     0 <foo>
                                           68:   e50b0008      str    r0, [fp, #-8]
                                           6c:   e51b3008      ldr    r3, [fp, #-8]
                                           70:   e1a00003      mov    r0, r3
                                           74:   e24bd004      sub    sp, fp, #4
                                           78:   e8bd4800      pop    {fp, lr}
                                           7c:   e12fff1e      bx     lr
```


Sequential Processing in further Programming Models – Functions

```
00000000 <foo>:                                00000030 <main>:
  0:   e52db004      push   {fp}
  4:   e28db000      add    fp, sp, #0
  8:   e24dd00c      sub    sp, sp, #12
   c:   e50b0008      str    r0, [fp, #-8]
 10:   e50b100c      str    r1, [fp, #-12]
 14:   e51b3008      ldr    r3, [fp, #-8]
 18:   e51b200c      ldr    r2, [fp, #-12]
1c:   e0030392      mul    r3, r2, r3
 20:   e1a00003      mov    r0, r3
 24:   e24bd000      sub    sp, fp, #0
 28:   e49db004      pop    {fp}
2c:   e12fff1e      bx     lr

                                     30:   e92d4800      push   {fp, lr}
                                     34:   e28db004      add    fp, sp, #4
                                     38:   e24dd008      sub    sp, sp, #8
                                    3c:   e3a03000      mov    r3, #0
                                     40:   e50b3008      str    r3, [fp, #-8]
                                     44:   e3a03000      mov    r3, #0
                                     48:   e50b300c      str    r3, [fp, #-12]
                                     4c:   e3a03002      mov    r3, #2
                                     50:   e50b3008      str    r3, [fp, #-8]
                                     54:   e50b3003      mov    r3, #3
                                     58:   e50b300c      str    r3, [fp, #-12]
                                    5c:   e51b0008      ldr    r0, [fp, #-8]
                                    60:   e51b100c      ldr    r1, [fp, #-12]
                                    64:   e50b0008      bl    0 <foo>
                                     68:   e50b0008      str    r0, [fp, #-8]
                                     6c:   e51b3008      ldr    r3, [fp, #-8]
                                     70:   e1a00003      mov    r0, r3
                                     74:   e24bd004      sub    sp, fp, #4
                                     78:   e8bd4800      pop    {fp, lr}
                                    7c:   e12fff1e      bx     lr
```



Sequential Processing in further Programming Models – Functions

```
// Program in C with function
```

```
int foo (int a, int b)
{
    return a * b;
}
```

```
int main (void)
{
    int a = 0;
    int b = 0;
    a = 2;
    b = 3;

    a = foo (a, b);

    return a;
}
```

```
00000000 <foo>:
      0:  e0000091      mul     r0, r1, r0
      4:  e12fff1e      bx     lr
```

Disassembly of section .text.startup:

```
00000000 <main>:
      0:  e3a00006      mov     r0, #6
      4:  e12fff1e      bx     lr
```

Sequential Processing in further Programming Models – Functions

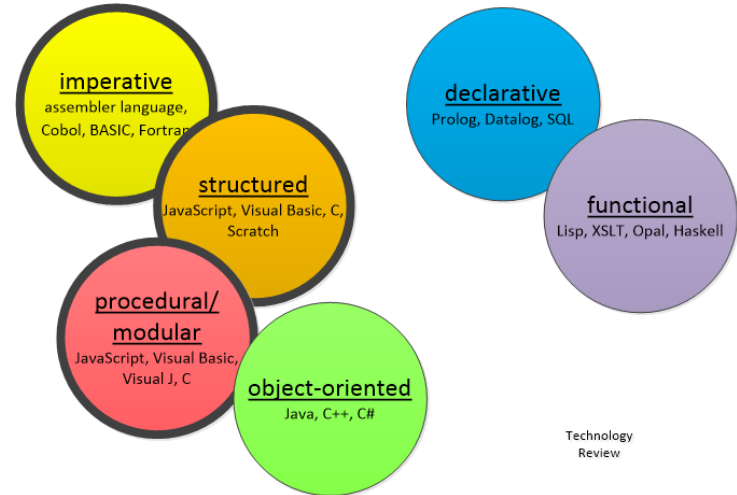
```
// Program in C with function
```

```
int foo (int a, int b)
{
    return a * b;
}

int main (void)
{
    int a = 0;
    int b = 0;
    a = 2;
    b = 3;

    a = foo (a, b);

    return a;
}
```



Sequential processing of commands and instructions is provided even with function calls.

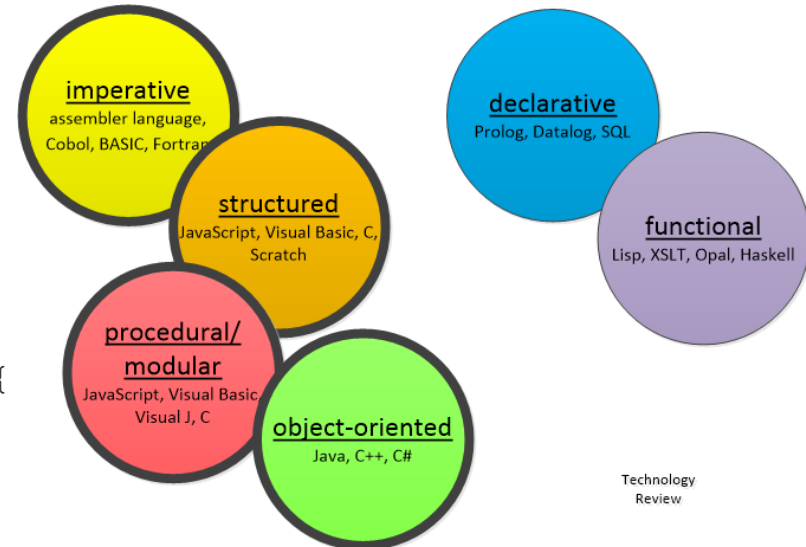
Sequential Processing in Object Oriented Programming Models – Classes/Objects

```
// simple program in Java
public class Rectangle {
    private int x, y;
    private int width, height;
```

```
    public Rectangle() {
        x = y = 0;
        width = 10;
        height = 10;
    }
    // methods ...
}
```

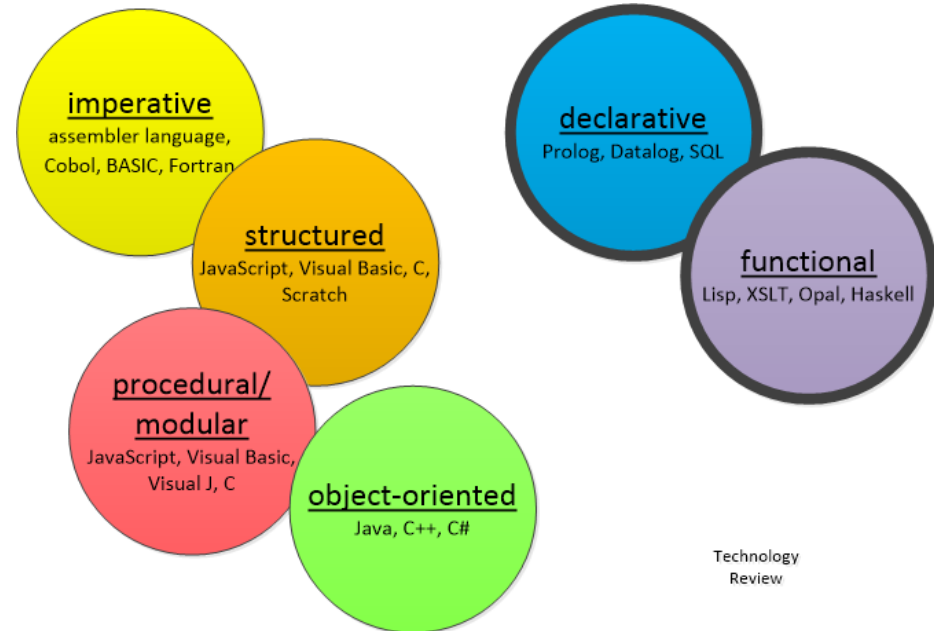
```
public class SimpleProgram {
    public static void main ( String[] args ) {
        Rectangle r = new Rectangle ();
    }
}
```

Methods in classes or objects correspond to functions.



Functional and Logical Languages

- In functional and logical (declarative) languages, the runtime environment takes care of the translation into (sequential) machine code.



Non-Deterministic Algorithms

- In real world applications external events often prevent the deterministic execution of a program.
 - user input
 - response of (external) devices
 - timers
- So, the strict definition of determinism is not applicable for most of the programs.

What does this means for programming and correctness of our program?

Non-Deterministic Algorithms and Programs

- Programs can (and should) be split into reasonable pieces (modules).
- These modules should provide a deterministic behavior by their own,
 - using distinct interfaces,
 - by encapsulating internal data
 - (see Lecture on “Software Engineering”)
- Knowledge about every single step in the executing of a program is often not needed as long as the results are correct (and the same applies for every other run with the same input).

Determined Algorithms and Programs

- Definition: A **determined algorithm** is an algorithm which, given a specific input, produces always the same specific output.
 - Thus, all deterministic algorithms are determined algorithms, but not all determined algorithms are deterministic.
 - Of course, this definition holds for programs too.
 - (It's not always usual to distinct determined algorithms from deterministic algorithms – but we do.)
 - With a more general assumption of input data the definition can be applied to many even broader ranges of applications.

Programs with Stochastic Variables

```
// Test for random variable
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

int main (void)
{
    int random_nr = 0;
    srand ((unsigned) time (NULL));
    random_nr = rand ();
    if (random_nr < RAND_MAX / 2)
        printf ("The random number is in the lower half! \n");
    else
        printf ("The random number is in the upper half! \n");
    return 0;
}
```

Does this program provides deterministic execution?

Programs with Stochastic Variables

```
// Test for random variable
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

int main (void)
{
    int random_nr = 0;
    srand ((unsigned) time (NULL));
    random_nr = rand ();
    if (random_nr < RAND_MAX / 2)
        printf ("The random number is in the lower half! \n");
    else
        printf ("The random number is in the upper half! \n");
    return 0;
}
```

With the same start parameters (input data), each program run provides the same results and the same final state **with respect to the distribution(s) of the random variables!**

With these preconditions we do have a determined program.

Programs using ML Algorithms

- Artificial Intelligence
 - Yes, but...
 - The amount of input data is the amount of all data ever processed as input data!

<https://www.newscientist.com/article/mg24132214-100-the-best-image-recognition-ais-are-fooled-by-slightly-rotated-images/>





Concepts of Non-Sequential and Distributed Programming

SUMMARY AND TAKE AWAYS

Correctness and Determined Program Execution

- With correct transformation of all of the models from programming model to execution model based on machine model the correct and determined (sometimes even deterministic) execution of the program can be assured.
 - The program given a specific input, will always produce the same specific output.
 - The sequential execution of the commands and instructions are one foundation of the reliable change of the system's state.
 - We can make reasonable assumptions about the program behavior at the process of programming.
 - Afterwards the program can be tested and checked with respect to the machine model.
- Models help us to abstract from real world and provide reasonable assumptions about the system and the state changes of the system.

Requirements for Programs

- A program should do what it is expected to do!
 - Functional requirements, such as
 - Scope of functions
 - Correctness
- A program should comply with certain requirements about its behavior.
 - Non-functional requirements, such as
 - Performance
 - Usability
 - Security
 - ...

Concepts of Non-Sequential and Distributed Programming

NEXT LECTURE

Algorithms and Programming IV

Concurrency

Summer Term 2023 | 26.04.2023

Barry Linnert