HOCHSCHULE FÜR TECHNIK **STUTTGART** **SOMMERSEMESTER 2017**

PRÜFUNGSLEISTUNG im STUDIENGANG **S**oftware **T**echnology

MODUL:	CPL	FACH:	Concepts of Programming Langauges
DATUM:	10.07.2017	NAME:	
ZEIT:	08:30-10:30	SEMESTER:	ST 1
PRÜFER:	Prof. Dr. Ulrike Padó, Prof. Dr. Peter Heusch		

1) Lisp, C, Assembler
3) Assembler, C, C++
4) Python, Prolog, Lisp,
5) COBOL 357, C# 102, C++ 82
Python has less for example no types 35
smalltalk 6, lua 25
6) Oldest Assembly
newest Swift

MODUL-TEILLEISTUNGEN		TEILPUNKTE	GESAMTPUNKTE	NOTE
FACH 1:	CPL	_________	____________	_________

HILFSMITTEL: Orignal Slides + Notes + Seminar Papers

ANLAGEN:

Please read the following general rules before starting the exam:

- Put your name on every sheet that you use for noting your exam, but do not put your name on otherwise empty sheets
- Do not write final answers on concept paper, do not write final answers with pencils, do not use your own paper
- Write legibly, leave at least 5 mm of empty space between your lines and at least 40 mm of empty space at the right margin
- **Put the solutions for questions 5 and 6 on a separate sheet of paper. Failing to do so will result in 50% deduction of points for these two questions.**
- **The points for every question reflect the amount of minutes you need to answer it.**

1. Question (30P)

Answer the following questions concerning the languages from the seminar. Every question has 3 points. Some questions contain a limitation of results after the question. Failing to comply (and giving more names) results in fewer points! Also denote in one sentence why the language of your choice belongs into that list.

1) Which of the languages are macro languages or contain macro features (e.g. C or C++ are no macro languages, but contain macro elements). Give three names.
2) Contrary to most other programming languages FORTH does not allow you to use the name of the word being defined inside the definition to call the word recursively, but you must use the predefined word recurse. What does this tell you about the compilation process in FORTH?
3) Which languages from the seminar have been used to implement operating systems? Give three names.
4) Which languages would you use to solve problems of artificial intelligence? Give three names.
5) Which languages have a large number of keywords, which languages do only have few? Name two languages for every case.
6) Which language is the oldest, which is the newest one of all the languages of the seminar? Name one language for every case.
7) Describe typical features of interpreted and compiled languages. Give two names for every category.
8) Which languages are specialized on web environments? Give three names.

9) Which languages were designed for commercial programs? Give three names.

10) Describe in 80 to 100 words which language you liked most and why.

2. Question (20P)

A lot of modern programming languages support type inference. E.g. Java 8 allows you to write

```
IntStream.range(0,100).forEach((x) -> {System.out.printf("%d%n", x&5); })
```

while it does not allow you to write

```
IntStream.range(0,100).asDoubleStream().forEach((x) ->
{System.out.printf("%d%n", x&5); }),
```

because the binary operator & is not defined for values of type double. However, this practice has become mainstream only during the last ten years. Describe the difference between this way of type inference and weakly typed languages.

A language is stringly typed if type errors are aways detected.
It is weakly typed if types are only loosely checked.
Java ist almost stongly typed with a few exceptions.
The example above does not work because the type of x is checked at compile time and boolean does not work.
The type x is not specificly typed but javas type inference automaticly assings a data types based on
the context.
In weakly typed languages that wouldn't be done?

ab
ba
abba
baab

3. Question (20P)

Look at the following grammar:

S -> S'$

S'-> aS'b | bS'a | S'S' | ε

Describe the words that this grammar produces and show that this grammar is not LL(1)-parsable. Define an equivalent grammar that is LL(1)-parseable.

The second rule contains a left recusrive production. Therefore it is not LL(1) parsable.

Proof by creating first set h
FIRST(S') -> ... FIRST(S') = FIRST(S') = FIRST(S') ....

By reformulating the grammar as a tail recusring we can make it LL(1) parsable.

S' -> aS'bS'' | bS'aS''
S'' -> S' | epsilon

alpha = null
beta = aS'b and bS'a

4. Question (15P)

Look at the following context free grammar:

E → E + T | T

T → T * F | F

F → '(' E ')' | <num>

Construct the abstract and the concrete syntax tree for the input (2 + 3) * 4

5. Question (15P)

a) What is a recursive function, and why is tail recursion important? (3P)

b) Explain how we can convert recursive functions into tail-recursive functions. Use the following Scheme code in your explanation (you do not need to produce Scheme code of your own unless you wish to do so) (6P):

```
a. (define sum
        1. (lambda (n)  (if (= n 1) 1
    ii. (+ n (sum (- n 1))))))
```

c) We have seen functional elements in many languages that are not functional in basic design. Why do you think this is? Please explain your opinion. (6P)

6. Question (15)

a) Prolog and Scheme are very close to two mathematical concepts. Why is this, and why were those concepts chosen? (8P)

b) You are looking for a pet. It should be small, harmless and furry. Give the search tree that Prolog follows when asked to find all possible suggestions for a good pet from its knowledge base. Be sure to show the goal stacks and unification steps as well as success or failure of a search branch. (7P)

Knowledge Base:

```
pet(X):- small(X), harmless(X), furry(X).

small(fly).
small(tarantula).
small(guineaPig).

harmless(fly).
harmless(guineaPig).

furry(tarantula).
furry(guineaPig).
```

- Built around the concept of functional application (from λ calculus)

- Uses expressions (evaluate to a value) instead of statements (alter machine state)
- Uses recursion instead of iterative loops

????

pet(X)