Örebro University
School of Science and Technology
Thomas Padron-McCarthy (thomas.padron-mccarthy@oru.se)

Exam

Compilers and Interpreters

for Dataingenjörsprogrammet, and others

Thursday January 2, 2020

Exam for:

DT125G Kompilatorer och interpretatorer, provkod A001

Aids:	No aids.
Score requirements:	Maximum score is 43. To pass, at least 8 points are required on task 1, and at least 23 points in total.
Results:	Announced on the course website or by e-mail no later than January 23, 2020.
Return of the exams:	Electronically through the student portal "Studenttjänster".
Examiner and teacher on call:	Thomas Padron-McCarthy, phone 070-73 47 013.

- Write clearly. Solutions that can not be read can of course not give any points. Unclear and ambiguous wording will be misinterpreted.
- Write the personal exam code on each sheet submitted. Do *not* write your name on the sheets.
- Write on only one side of the paper. Do not use a red pen.
- Assumptions beyond those in the given problems must be stated.
- You are allowed to explain your solutions. Even an incorrect answer may give some points, if the key ideas were right.

GOOD LUCK!!

Formulas

1. Eliminating left recursion

A left-recursive grammar can be transformed to a grammar that is not left recursive. Assume that the grammar contains a rule (or, more correctly, two productions) like this:

```
A -> A x | y
```

A is a non-terminal, but **x** and **y** are any constructions consisting of terminals and non-terminals.

The rule is replaced by the following two rules (or, more correctly, three productions), that describe the same langugage, but are not left recursive:

```
A -> y R
R -> x R | empty
```

2. Left factorization

Assume that the grammar contains this rule (two productions):

```
A -> x y | x z
```

A is a non-terminal, but **x**, **y** and **z** are any constructions consisting of terminals and non-terminals.

Replace with these three productions:

```
A -> x R
R -> y | z
```

Task 1 (10 p)

A compiler's work is usually divided into a number of phases. Which are those phases? Explain shortly what each phase does. What is the input and the output of each phase?

Task 2 (6 p)

Here is a program segment written in a C-like language:

```
a = 1;
b = c - 2 * d - 3;
while (e * 4 < f * 5) {
    e = e + 6;
    f = f - 7;
}
g = 8;
```

Translate the program segment to *two* of the following three types of representations. (Should you answer with all three, the one with the highest points will be discarded.)

a) an abstract syntax tree (by drawing it)

b) postfix code for a stack machine

c) three-address code

Task 3 (5 p)

Explain the following compiler-related terms.

a) activation record (in Swedish: aktiveringspost)
b) call sequence (Swedish: anropskonventioner)
c) conservative garbage collection (Swedish: konservativ skräpsamling)
d) copy propagation
e) DAG

Task 4 (6 p)

Explain the difference between:

a) structural equivalence and name equivalence
b) static and dynamic (when talking about compilers)
c) an ambiguous grammar (in Swedish: tvetydig grammatik) and a FIRST() conflict

Scenario for task 5-7

We're working with machine translation between natural languages such as Swedish and English, and we have made a table with words in different languages:

Words				
Swedish	**English**	**German**	**Smurf**	**Cat**
bil	car	Auto	smurf	meow
skola	school	Schule	smurf	meow
hus	house	Haus	smurf	meow

We could store the table in a database and use SQL commands, but it is probably simpler to create a specialized input language for entering words and for searching. This input language should have three commands:

- **new language**, to add a new natural language to the system. Example:
  ```
  new language Norwegian
  ```
- **add word**, to add a new word with a translation to the system. This command should have the format **add word** *word-1* **in** *language-1* **=** *word-2* **in** *language-2*. Example:
  ```
  add word hus in Swedish = Haus in German
  ```
- **translate**, to perform a translation. This command should have the format **translate** *word* **from** *language-1* **to** *language-2*. Example:
  ```
  translate school from English to German
  ```

Task 5 (4 p)

a) (2p) Which terminals are needed to write a grammar for the input language in the scenario?

b) (2p) Out of these terminals, some will not have fixed lexemes. Write regular expressions for each such terminal.

Task 6 (4 p)

Write a grammar for the input language. The start symbol should be *command*, which represents *one* single command according scenario above.

Task 7 (8 p)

Write a predictive recursive-descent parser for the input language, in a language that is at least similar to C, C++, C# or Java. You do not have to write exactly correct program code, but it should be clear which procedures exist, how they call each other, and what comparisons with token types are made. You can assume there is a function called **scan**, which returns the type of the next token, and a function called **error**, which you can call when something went wrong and which prints an error message and terminates the program.
