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

Exam

Compilers and Interpreters

for Dataingenjörsprogrammet, and others

Monday October 28, 2019

Exam for:

DT125G Kompilatorer och interpretatorer, provkod 0100

Aids:	No aids.
Score requirements:	Maximum score is 36. To pass, at least 8 points are required on task 1, and at least 20 points in total.
Results:	Announced on the course website or by e-mail by Monday November 18, 2019.
Return of the exams:	Electronically through Studentforum.
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.
- Enter 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 (4 p)

In the following program, there are eight errors and warnings that will be reported, as shown by the comments in italics. Some of these errors and warnings may be detected in one of the compiler's phases, and some may be detected at other times. When will each of the errors and warnings be detected?

```
#include <stdio.h>

int main(void) {
    prontf("Starting...\n");    // 1: undefined reference to `prontf'
    int a = 17zzz;              // 2: invalid suffix on integer constant
    int b = ;                   // 3: expected expression before ';' token
    int c = 0;
    if (c == 0) {
        int d = a / c;          // 4: math exception
        int e, f, g;
        int h                   // 5: expected '=', ',' or ';' before 'e'
        e = 17;                 // 6: variable 'e' set but not used
        f = g;                  // 7: 'g' may be used uninitialized
        printf("d = %d, f = %d\n", d, f);
    }
    return "Done!";             // 8: return makes integer from pointer
}
```

Task 3 (6 p)

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

```
x = 0;
y = z * 2 - t - 2 * 2;
while (x < y) {
    if (x < 5) {
        x = x + 1;
    }
    else {
        x = x + 2;
        y = y + 1;
    }
}
```

a) Translate the program segment to an abstract syntax tree (by drawing it).

b) Translate the program segment to *either* postfix code for a stack machine *or* three-address code. (Not both!)

Scenario for task 4-6

Some biologists are making an inventory of the anmials in an area, and they need a language to write down and process their observations. The language should allow them to first declare a number of species, and then a number of observations of individual animals of those species. A complete input in this language might look like this:

```
species rabbit
species lion
declarations done
observation rabbit 1
observation lion 30
observation rabbit 3
observations done
```

This means that the animals we observe are rabbits and lions, and we have observed first a rabbit, then thirty lions, and finally three more rabbits. After listing the species, the "declarations" part of the input is ended with **declarations** and **done**. After listing the observations, the "observations" part of the input is ended with **observations** and **done**.

Animal names are single words that consist of lower-case letters.

It should be possible to write the input in free format, such as in most common programming languages, for example like this:

```
species rabbit species lion declarations

done observation rabbit 1
    observation

lion 30 observation                     rabbit 3 observations done
```

Task 4 (4 p)

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

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

Task 5 (4 p)

Write a grammar for the animal language. The start symbol should be **input**, which represents a complete input according to the scenario above.

Task 6 (8 p)

Write a predictive recursive-descent parser for the animal 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.