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 29, 2018

Exam for:

DT125G Kompilatorer och interpretatorer, provkod 0100
DT3030 Datateknik C, Kompilatorer och interpretatorer, provkod 0100

→ This exam is also available in a Swedish version.

Aids:	No aids.
Score requirements:	Maximum score is 36. To pass, at least 18 points are required.
Results:	Announced on the course website or by e-mail by Monday November 19, 2018.
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 (8 p)

This is a program segment from a C-like language.

```
if (a < b) {
    while (c > d) {
        b = a + 1;
        a = c * (b + c);
        a = a + 1;
        d = c * (b + c);
    }
    b = a + 1;
}
c = a + 1;
```

a) Translate the program segment to *either* a abstract syntax tree (by drawing the tree) *or* postfix code for a stack machine. (Not both!)

b) Translate the program segment to three-address code. Identify the basic blocks and draw the flow graph. (The flow graph, with the three-address code in it, is sufficent as answer.)

c) Show an optimization that can be done within one of these basic blocks.

Scenario for task 3-6

Soon it's Halloween, and the children make a "candy round" where they walk around to the neighbors and ask "trick or treat". To document their activities they need a special Halloween language, where a complete input might look like this:

```
began candy round;
treat: three candies;
trick: made some ghostly sounds;
treat: one kilo of chocolate;
treat: a carrot;
trick: set fire to the neighbors car;
went home;
```

The input describes a candy round. It starts with **began candy round;** and ends with **went home;**. In between, there are zero or more notes about **trick** or **treat**. Such a note begins with either **trick** or **treat**, a colon (**:**), one or more words that indicate what happened, and a semicolon (**;**).

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

```
began candy round ; treat : three candies ; trick
: made some ghostly
sounds ; treat : one kilo of chocolate
; treat
: a carrot ; trick : set fire
to the neighbors car ; went home ;
```

Task 3 (3 p)

a) (1p) One of the terminals, that is, types of tokens, that are needed to write a grammar for the Halloween language is **word**. That is simply one or more lower-case letters, **a** to **z**. (We only use the English alphabet.) Write a regular expression that describes what such a word can look like.

b) (2p) Which other terminals, except **word**, are needed to write a grammar for the language?

Task 4 (4 p)

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

Task 5 (3 p)

A parse tree (sometimes called a "concrete syntax tree") contains nodes for all non-terminals. Draw the parse tree for this candy roound, according to your grammar from the task above:

```
began candy round;
treat: one chewing gum;
went home;
```

Task 6 (8 p)

Write a predictive recursive-descent parser for the Halloween 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.
