Stefan Rührup
Computer Networks and Telematics
University of Freiburg, Germany

Network Protocol Design
and Evaluation
Summer 2009

Exercise No. 4

May 29, 2009

Task 1 *ABNF*
An eMail server accepts a comma-separated list of the recipients' email addresses. Valid eMail addresses consist of a name, the at-symbol, and a domain name. Invalid addresses such as "@ietf.org" or "bg@ms..gov" should not be accepted. Write an ABNF specification for this list and implement a parser for your specified grammar.

1. Download the parser generator APG (http://www.coasttocoastresearch.com/) and build it using `./configure; make` (instructions given here refer to a Linux/Unix environment).
2. Create a subdirectory `MyParser` in the APG/Samples directory.
3. Write your ABNF specification and save it in a text file `List.bnf` in this directory. You might use the following rules:
   ```
   eol = [%x0d] %x0a
   at-sign = "@"
   period = "."
   separator = ","
   char = %x41-5A / %x61-7A
   digit = %d48-57
   ```
4. Save a copy of the file `main.cpp` that comes along with this exercise in the `MyParser` directory.
5. Create the parser:
   ```
   ../../Generator/release/apg /in:List.bnf /out:MyParser /cpp
   ```
6. Compile the parser:
   ```
   g++ MyParser.cpp -I ../../ApgLib/src/ -I ../../ApgUtilities/src/
   -L ../../ApgLib/release/libapg.a -c
   ```
7. Compile the parser:
   ```
   g++ main.cpp -I ../../ApgLib/src/ -I ../../ApgUtilities/src/ -I .
   -L MyParser.o -L ../../ApgLib/release/libapg.a
   -L ../../ApgUtilities/release/libapgutilities.a -o test
   ```
8. Test your application with a sample list saved in `List.txt` in the same directory.

The commands described here refer to APG 5.1 invoked from the APG/Samples/MyParser directory. You might as well use the Makefiles of the Currency example program.

Task 2 *CSN.1*
Consider the following specification in CSN.1. A data packet consists of a sequence of TLVs, where tag and value fields have a length of 1 octet each. In the value field we store bit strings, with padded bits at the end. The length field gives the length of the value field in octets, while the padding value contains the number of padded bits.

```
< data packet > ::= < date item >**;
< octet > ::= bit (8);
< data item > ::=  <Type : octet > < Length : octet > < Value >;
< Value > ::= <Padding : octet> < bit >** < spare padding > ;
< spare padding > ::= < bit** > = 0**;
```

With the given specification the decoder might read the first TLV and interpret all following bits as spare bits (PP..P is the padding value octet, x indicates a spare bit):

```
1       |2       |3       |4       |5       |6       |7       |8       |9 [octet]
8......1|8......1|8......1|8......1|8......1|8......1|8......1|8......1|8......1|
TTTTTTTT|LLLLLLLL|PPPPPPPP|VVVVVVVV|VVVVVVVV|TTTTTTTT|LLLLLLLL|PPPPPPPP|VVVVVVVV|
encoded string:
00100016|00000010|00000101|10101010|101xxxxx|00100016|00000001|00000011|10111xxx|
decoded string:
00100016|00000010|00000101|10101010|101xxxxx|xxxxxxxx|xxxxxxxx|xxxxxxxx|xxxxxxxx|
```

How can we make sure in the specification that the padding is limited and the next data element will be identified?

Task 3 *ASN.1*
Specify a data structure in ASN.1 where you can store a persons name and birthdate and also the names and birthdates of her mother and father and of their grandparents, great-grandparents and so on.

Task 4 *Tagging in ASN.1*
Tagging is used to disambiguate message field. However, tags are not always necessary. Which tags can be removed from the following specification?

```
Packet ::= [1] SEQUENCE {
    seqno [2] INTEGER,
    ttl   [3] INTEGER OPTIONAL,
    data  [4] DataType }

DataType := [5] CHOICE {
    plaintext  [6] PrintableString(SIZE(206)),
    ciphertext [7] OCTET STRING(SIZE(206)),
    publickey  [8] BIT STRING(SIZE(16)) }
```