# McMaster University Comp Sci 4TB3/6TB3, Winter Term 2017/18 — Lab 4
## For the Labs on January 20 -February 2,
## Due Monday, February 5, 11 pm

Eden Burton, Jenny Wang, Spencer Park
out of 20 points

- *Submission is to be done exclusively through Avenue. Submissions via e-mail will **not** be accepted. A **10% penalty** will be accessed for each day the lab is submitted after the due date*.
- This assignment requires access to a Linux, MacOS X, or some other Unix computer. You can log in remotely to either `moore.mcmaster.ca` or to `mills.mcmaster.ca` with ssh. Submissions are tested on `moore.mcmaster.ca`, please check if your submission works there.
- In this lab, you are allowed to work in pairs, provided that you split the work equally and arrive at a common understanding of the solution. However, in that case you must state in your submission the person you worked with, such that similarities in the solution will not be construed as Academic Dishonesty. Working in groups of three or larger is not allowed and will be considered Academic Dishonesty. If you look for someone to work with, we will try to find a match, please contact the TAs.
- You are allowed and encouraged to talk to everyone in the course to get a common understanding of the problem, but you can share partial solutions only with your collaborator, if you work in a pair. The final submission must be your own, that is, you cannot submit identical submissions with two names on them.
- The Tutorial Exercises will be presented in the tutorials; you need to submit only answers to the Lab Questions. In the labs, the solution to last week's lab questions are discussed and you can get help with this week's lab questions. Attendance at the labs is not checked.

**Tutorial Exercise 1** (XSD)**.** XML files are self-describing hierarchical structures, meaning that a grammar is not needed for parsing. As the structure is embedded in the file, some checks can immediately be performed by an XML parser, e.g.:

- XML documents must have a root element
- XML elements must have a closing tag
- XML elements must be properly nested
- XML attribute values must be quoted

For further checks, a grammar has to be supplied. XSD is a common format for writing grammars for XML files. XSD is like EBNF, but richer: for example, it is possible to specify a minimum and maximum on the number of repetitions and it is possible to specify that certain elements can appear in any order (thus allowing sets rather sequences to be specified). A syntactically correct XML file is *well-formed*. An XML file that is validated against an XSD grammar is *valid*. Suppose you want to describe the preferred contact of employees of a company:

- Each company consists of a sequence of entries.
- Each entry is either a person or a subdivision.
- A person has a name and either a telephone number or e-mail address.
- A subdivision has a name and consists of a sequence of entries.
- A name is an arbitrary string.
- An e-mail address has letters, "@", letters, ".", letters.
- A telephone number is a sequence of digits.

For example,

```
<employees>
 <person>
  <personname>emil</personname>
  <tel>123456789</tel>
 </person>
 <person>
  <personname>boss</personname>
  <email>boss@mcmaster.ca</email>
 </person>
 <subdivision>
  <subdivisionname>CAS students</subdivisionname>
  <person>
   <personname>shucai</personname>
   <email>yaos@mcmaster.ca</email>
  </person>
  <person>
   <personname>shuc</personname>
   <email>yao@mcmaster.ca</email>
  </person>
 </subdivision>
</employees>
```

**Lab Question 1** (Address Book in XML, 8 points)**.** You got tired of writing a regular expression each time you look for a friend and decided to keep an electronic address book. Naturally, XML comes to your mind. You have following structure for your contacts in mind:

- Your contacts consist of a list of pals.
- Each pal has in following order a name (a string), optionally a nickname (a string), optionally a birthday (a date), the favourite OS (either Mac or PC or Linux), an address (a string), optionally a postal code, and either an e-mail address or a phone number or both.
- Postal codes are of the form Z9Z 9Z9, where Z is a capital letter and 9 is a digit.

For strings and dates you can use xsd:string and xsd:date. For e-mail addresses and telephone numbers you can use strings (bonus points for a more precise specification). For example, your contacts may be:

```
<contacts>
   <pal>
      <name>Bogart, Humphrey</name>
      <nickname> Bogie </nickname>
      <birthday>1899-12-25</birthday>
      <os>Linux</os>
      <address>1280 Market Place, Casablanca</address>
      <phone>(905) 525 9140</phone>
   </pal>
   <pal>
      <name>Monroe, Marilyn</name>
      <os>Mac</os>
      <address>1280 Sunset Drive, Hollywood</address>
      <postal_code>X0X 0X0</postal_code>
      <phone>(323) 525 9140</phone>
      <e-mail>some@like.it.hot</e-mail>
   </pal>
</contacts>
```

Your task is to write an XSD grammar. There are numerous online resources on XML and XSD. Use `http://www.utilities-online.info/xsdvalidation/` to check your XSD grammar. Do not generate the grammar with a tool. (Generated grammars can be unreadable; the grammar should be around 40 lines long.)

**Lab Question 2** (Constant Evaluator with Type-Checking, 12 points)**.** Extend the grammar found on page 111 of the notes to support Boolean expressions. This necessitates distinguishing between Boolean and integer types and checking types. The Boolean type is represented with the characters "t" for true and "f" for false . They are given value attributes of 1 and 0 respectively.

You may use the sample code on Avenue as a starting point, or you may write your own parser from scratch. First, extend the scanner to recognize `&`, `|`, `~`, `=`, `#` as conjunction, disjunction, negation, equality, inequality. Include and document additional attribute rules to ensure that the new operators work only with Boolean operands.

As mentioned before, you can implement the parser in the language of your choice but you must provide a script called "script." which allows your program to run on a Linux server such as `mills.mcmaster.ca` hosted at the university. The program shall read from *standard input* until a newline character and write *"type:* ***synthesized attribute type*** *val:* ***synthesized attribute val*** *"* to *standard output*.

Note that the grammar does not include terminals such as spaces or the newline character. The following helps describe the expected behavour of the parser.

| *sentence* | *expected output* |
|---|---|
| $2*(3+3)$ | *type* : **integer** *val* : **12** |
| $t\&f$ | *type* : **boolean** *val* : **0** |
| $t\&1$ | *type* : **error** *val* : **n/a** |