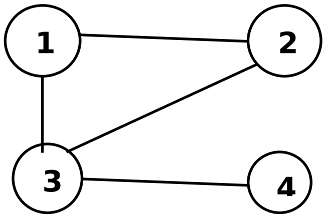
# Compiler Construction

Final Exam, Spring 2015

**Date:** Jun 12, 2015 **Marks:** 50 **Time: 3 hrs**

## Question 1 (5+10 marks)

An undirected graph can be represented in XML format. Consider the following graph, for example:



We can represent this graph using XML as follows:

```
<graph>
     <node> 1, 2, 3, 4 </node>
     <edge>
          (1, 2), (1, 3), (2, 3), (3, 4)
     </edge>
</graph>
```

Supposing you need to develop a parser to parse such strings, answer the following questions:

a) Identify all the tokens for the parser. Give regular definitions for any complex tokens. Assume the node-labels can be any text, for example Lahore, Islamabad, Karachi, etc.

b) Give a left-recurisve CFG for the translator.

## Question 2 (10 marks)

Consider the following grammar to generate a number along with its base. The base can be binary, octal or hexadecimal.

```
S → BASE  NUM
BASE → b | o | h
NUM → NUM  DIGIT
DIGIT → 0|1|...|9|A|B|...|F
```

Add semantic actions to compute equivalent decimal value for a given string. For example the input "o 123" shall be converted to "83" ... Do not check for invalid strings; assume the user enters correct data. Accumulate result as you visit each digit in the parse tree; do not use any function or trick to escape the usual recursive computation in a translation scheme.

## Question 3 (5+10 marks)

a) Give three-address code for the following C-like code:

```
sum = 0;
for (i = 1; i <= n; ++i) {
      sum = sum + i;
}
cout << sum;
```

b) Give a translation scheme to perform translations as in part a; i.e. translate any given C-like "for" statement  into equivalent three-address code. Use the following grammar for your scheme:

```
S → for ( A ; BE; A ) { L }   // add actions in this production
A → id = E        // no need to add actions here
BE → id RO id     // you may add actions here
L → L ; A         // no need to add actions
L → A       // no actions
```

Here the token RO represents any relational operator: <=, <, >=, >, ==, !=

## Question 4 (10 marks)

Consider a virtual machine that executes four-address code. All variables are global and are stored in a data section. The only data type available is Integer. Following is its code skeleton:

```
int *ds = new int[...]; // data section
int quad[...][5]; // 4-address code stored in quadruple
int pc = 0; // program counter
...
for (int pc = 0; quad[pc][0] != HALT; ++pc) {
      switch (quad[pc][0]) {
      case '+': ...
      case '-': ...
      case MNG: // Add code here!
      case LOSE: // Add code here!
      ...
      }
}
```

The machine supports several interesting instructions. One instruction is "mingle":

```
MNG  W, X, Y, Z
```

The instruction swaps the adjacent operands. For example, let the values of w, x, y, z are 10, 20, 30, 40 respectively; now after execution of the instruction the values will be 20, 10, 40, 30 respectively.

Another instruction is "lose". It is a branch instruction, and does not take any operand. Rather it jumps at a random address.

Give C/C++ code to execute these two instructions.