Tutorial session 1
DOM and basics of XML parsing

1 XML lexing and parsing

In this excercise, we suppose a function `getchar()` which returns the next ASCII character of some input or *EOF* after the last character of the input has been read.

Questions :

1. Write a small function (in pseudo-language) that returns `true` the input is a well-formed XML opening tag ("`<foo>`").
2. Write a set of functions which check whether the sequence of characters in the input is compatible with an XML document (you only have to check for opening, closing tags and attributes ; you don't need to consider special tags such as `<? xml ... ?>`, processing instructions, CDATA sections, comments, entities, . . .).
3. What would you need to add to verify that the input is a *well-formed* XML document, that is, that every opening tag is correctly closed and that the document has a root element ?
4. (extra) modify your function to check the well formedness of the document.

2 UTF-8 sequences

UTF-8 is a standard for character encoding. It is *backward* compatible with 7 bit ASCII, which means that every character whose code is below 127 is the same, both in ASCII and UTF-8. UTF-8 was designed to take into account languages with wide range of characters (more than the 256 characters that one can represent using one byte, e.g. Chinese, Japanese, Korean, Arabic with ligatures, Mathematics, . . .). Hence any character of code 128 and above must be encoded. These extended characters can take up to 4 bytes. The encoding works as follows :

Character range	↔	bit representation	sample character
0-7F (0-127)	1	**0**xxxxxxx	A $=_{10}$ 65 $=_{16}$ 41 $=_2$ **0**1000001
80-7FF (128-2047)	2	**110**yyyxx **10**xxxxxx	Ł $=_{10}$ 321 $=_{16}$ 141 $=_2$ 1 01000001 **110**00101 **10**000001 $=_{16}$ C581
0800-FFFF (2048-65535)	3	**1110**yyyy **10**yyyyxx **10**xxxxxx	Ễ $=_{10}$ 7876 $=_{16}$ 1EC4 1EC4 $=_2$ 11110 11000100 **1110**0001 **10**111011 **10**000100 $=_{16}$ E1BB84
010000-10FFFF (65536-1114111)	4	**11110***zzz* **10***zz*yyyy **10**yyyyxx **10**xxxxxx	𝔸 $=_{10}$ 120120 $=_{16}$ 1D538 1D538 $=_2$ *1* 11010101 00111000 **11110***000* **10***01*1101 10010100 **10**111000 $=_{16}$ F09D94B8

Questions :

1. How many bytes are needed to encode the string `"Żubrówka"`, knowing that `'Ż'` is the 379th UTF-8 character and `'ó'` is the 243rd UTF-8 character.
2. In C or C++, strings are usually equivalent to arrays of bytes. Suppose you have a well-formed UTF-8 string represented as an array of bytes, terminated by the *NULL* (byte `00`) character. Write (informally) the algorithm computing the number of characters in this array.
3. Write the algorithm that gives you the position of the nth character of a given string. Is it as fast as for ASCII strings (plain C strings) ?
4. How could you store the UTF-8 string to get fast access to any character. How many bytes would then take the string `"Żubrówka"` in this data-structure. How many bytes are wasted by doing so ?

Note : In reality, handling UTF-8 strings is much more complicated than this, due to the presence of *combining* characters. For instance, the character number 769, `' ́'` (combining accute accent, which is different from the plain accute accent with code 180) can be used to put an accute accent on any character. Thus, the sequence `CC 81 40` which is composed of `CC 81` = `' ́'` and `40` = @ gives the character `'@́'` and should be counted as *a single character*.

3 Size of a DOM implementation

The DOM specification defines the various fields and methods that each type of node in an XML document should have. For *element* node and *text* nodes, these are :

Element :

`nodeName`	Tag of the element
`nodeValue`	`null`
`nodeType`	(constant) `ELEMENT_NODE`
`parentNode`	(pointer to) the parent
`childNodes`	(pointer to) the list of children
`firstChild`	The first node contained in `childNodes`
`lastChild`	The last node contained in `childNodes`
`previousSibling`	(pointer to) the previous sibling
`nextSibling`	(pointer to) the next sibling
`attributes`	(pointer to) the map of attribute nodes
`textContent`	Concatenation of the `textContent` attribute value of every child node, excluding COMMENT_NODE and PROCESSING_INSTRUCTION_NODE nodes.

Text :

`nodeName`	`"#text"`
`nodeType`	(constant) `TEXT_NODE`
`parentNode`	(pointer to) the parent
`childNodes`	`null`
`firstChild`	`null`
`lastChild`	`null`
`previousSibling`	`null`
`nextSibling`	`null`
`attributes`	`null`
`data`	A String including all character code contained in this node
`length`	The number of 16-bit units needed to encode all ISO 10646 character code contained in the character information items using the UTF-16 encoding.
`isElementContentWhitespace`	The element content whitespace property

1. Give a reasonable type for every attribute
2. Based on it, how much space would one `Element` object take in memory ? a `Text` object ? (you can assume that the size of an object is the sum of the size of its fields).
3. Consider the following document in Figure 1. What is the size in bytes of this document (supposing the encoding is *ASCII*).
4. What would be the size of its DOM representation, assuming that each element is represented by an *Element* object and each text content by a *Text* object (Note : UTF-16 encodes every character on *at least* two bytes and every character outside of the range `0-FFFF` with 4 bytes. It is compatible with 7 bit ASCII, meaning that every ASCII character whose code `xx` is below 127 is encoded as `00xx`).
5. Assuming that the factor between size on disk and size of the DOM is constant, what is the biggest document that you could parse with DOM on a machine with 2GB ram.

```
<addressbook>
␣␣<contact>
␣␣␣␣<name>
␣␣␣␣␣␣<first>John</first>
␣␣␣␣␣␣<last>Smith</last>
␣␣␣␣</name>
␣␣␣␣␣␣<tel>0206578913</tel>
␣␣␣␣␣␣<email>john@smith.com</email>
␣␣</contact>
␣␣<contact>
␣␣␣␣<name>
␣␣␣␣␣␣<first>Foo</first>
␣␣␣␣␣␣<last>Bar</last>
␣␣␣␣</name>
␣␣␣␣␣␣<tel>010203040506</tel>
␣␣␣␣␣␣<email>foobar@baz.whitehouse.gov.us</email>
␣␣</contact>
</addressbook>
```

FIG. 1 – Addressbook document