Compiler Construction — XML File Processing

1 Aim

The aim of the exercise is to improve skills for creation of scanners and parsers augmented with semantic analysis. A program should be created in the course of the exercise that should print the file taking into consideration indentation of tags while at the same time checking correctness of a simplified XML file.

The exercise is also meant as a test for greater self-reliance of students, so only an instruction manual is provided, and there are no files with a skeleton of the program. The `Makefile` file can be copied and adapted from previous exercises. Test files are provided.

2 Lexical Analysis

The scanner should be put into `x.l` file.

2.1 Declarations

In the declarations section:

1. A file `defs.h` holding a macrodefinition of the greatest acceptable string length `MAXSTRLEN` should be included.
2. A file `x.tab.h` with definitions from the parser should also be included.
3. The syntax of an identifier should be named as an identifier is part pf several constructions to be recognized. The first character of the identifier can be a letter, a colon, or an underscore; further characters can additionally include digits and the minus sign.
4. Start conditions `ST_COMMENT` (comment contents), `ST_PI` (processing instruction contents), and `ST_TAG` (tag contents) should be declared.

2.2 Rules

During the lexical analysis:

1. Comments are removed; comments start with ”`<!--`” strings, and end with ”`-->`” preceded with characters different from the minus sign.
2. Unfinished comment is detected.
3. The start of a processing instruction in form of ”`<?`” string with an identifier that follows is detected. It is returned as a final symbol (token) `PI_TAG_BEG`, and its value is the identifier text.
4. The contents of a processing instruction is ignored.
5. The end of processing instruction ”`?>`” is detected and returned as `PI_TAG_END`.
6. The start of an opening tag in form of the character ”`<`” with the following identifier is detected and returned as the `STAG_BEG` token. Its value is the text of the identifier.
7. The start of a closing tag in form of the string ”`</`” with the following identifier is detected and returned as the `ETAG_BEG` token. Its value is the text of the identifier.
8. The contents of tags is ignored.
9. The end of a tag in form of ”`>`” is detected and returned as the `TAG_END` token.
10. The end of an empty token in form of ”`/>`” is detected and returned as the `ETAG_END` token.

11. Strings "`&lt;`", "`&gt;`", "`&amp;`", "`&apos;`", and "`&quote;`" are returned as the token `CHAR` with the value of the characters "<", ">", "&", apostrophe and double quote respectively.
12. Sequences of spaces and horizontal tabulations at the beginning of lines are ignored.
13. The new line (line feed) character is returned as itself.
14. The carriage return character is ignored.
15. Spaces and horizontal tabulation characters are returned as the `S` token. Their value is the matched character.
16. The remaining characters are returned as the `CHAR` token with the value being the matched character.

2.3 Functions

The `yywrap()` function returns the value of 1, and it detects unfinished comments, processing instructions or tags.

3 Parsing

The parser should be put into `x.y` file.

3.1 Declarations

In the declarations section:

1. The header files of libraries `stdio.h` and `string.h`, as well the header file `defs.h` containing a macrodefinition of MAXSTRLEN constant should be included.
2. The variable `level` with the initial value of 0 representing the nesting level of tags should be declared.
3. The variable `pos` with the initial value of 0 representing the current text column should be declared.
4. Constants `INDENT_LENGTH` and `LINE_WIDTH` representing the width of a single indentation (e.g. 2) and the maximal number of characters in a line (e.g. 78) should be defined.
5. A prototype of the function `indent()` that adds indents with the parameter being the indentation level should be typed in.
6. The `%union` directive with a character array field named `s` capable of holding `MAXSTRLEN`+1 characters should be typed in.
7. `PI_TAG_BEG`, `PI_TAG_END`, `STAG_BEG`, `ETAG_BEG`, `TAG_END`, `ETAG_END`, `CHAR` i `S` should be declared as string tokens.
8. Grammar variables `start_tag`, `end_tag`, and `word` should be declared as having string values.

3.2 Rules

1. An XML document consists of an introduction and an element.
2. The introduction consists of a sequence of processing instructions and new line characters.
3. A processing instruction consists of a beginning of the processing instruction (`PI_TAG_BEG`) and the end of that instruction (`PI_TAG_END`).
4. An element consists of an empty tag or from a pair of elements.
5. An empty tag consists of the empty tag beginning (`STAG_BEG`) and the end of the empty tag (`ETAG_END`).
6. A pair of elements consists of the opening tag, the contents, and the ending tag.

7. An opening tag consists of the tag beginning (`STAG_BEG`) and the tag end (`TAG_END`).
8. An ending tag consists of the ending tag beginning (`ETAG_BEG`) and the tag end (`TAG_END`).
9. A contents is a sequence of elements, white spaces (`S`), words (sequences of characters different from white spaces) and new line characters.

3.3 Functions

The following should be defined:

1. `main()` function that calls `yyparse()` function,
2. `yyerror()` function that prints the given message and returns 0,
3. `indent()` function that adds indentation.

4 Tasks

Tasks to be completed:

1. implement the scanner,
2. implement the parser that checks syntactic correctness of the analyzed XML file,
3. check matching of a pair of tags and print a message when mismatch occurs,
4. print found tags with their names while ignoring the tag interior,
5. print found processing instructions with their names while ignoring their interior,
6. print strings and processing instructions with indentation reflecting their nesting level,
7. print the text between the tags with indentation reflecting its nesting level,
8. print the text between the tags with indentation reflecting its nesting level so that it does not go beyond the declared line width (`LINE_WIDTH`). Line breaking can be simplified, i.e. whole words can be moved to subsequent lines — dividing words into syllabels is not necessary.

5 Grading

Tasks 1–7 are worth 1 point each. The last task merits 3 points.