```
</anthology>
```

The ability to use rules stating which elements can be nested within others to simplify markup is a very important characteristic of SGML. Before considering these rules further, you may wish to consider how text marked up in the form above could be processed by a computer for very many different purposes. A simple indexing program could extract only the relevant text elements in order to make a list of titles, or of words used in the poem text; a simple formatting program could insert blank lines between stanzas, perhaps indenting the first line of each, or inserting a stanza number. Different parts of each poem could be typeset in different ways. A more ambitious analytic program could relate the use of punctuation marks to stanzaic and metrical divisions.

Note that this simple example has not addressed the problem of marking elements such as sentences explicitly; the implications of this are discussed below in section 2.5.2, *Concurrent Structures*.

Scholars wishing to see the implications of changing the stanza or line divisions chosen by the editor of this poem can do so simply by altering the position of the tags. And of course, the text as presented above can be transported from one computer to another and processed by any program (or person) capable of making sense of the tags embedded within it with no need for the sort of transformations and translations needed to move word processor files around.

2.4 Defining SGML Document Structures: The DTD

Rules such as those described above are the first stage in the creation of a formal specification for the structure of an SGML document, or *document type definition*, usually abbreviated to *DTD*. In creating a DTD, the document designer may be as lax or as restrictive as the occasion warrants. A balance must be struck between the convenience of following simple rules and the complexity of handling real texts. This is particularly the case when the rules being defined relate to texts which already exist: the designer may have only the haziest of notions as to an ancient text's original purpose or meaning and hence find it very difficult to specify consistent rules about its structure. On the other hand, where a new text is being prepared to an exact specification, for example for entry into a textual database of some kind, the more precisely stated the rules, the better they can be enforced. Even in the case where an existing text is being marked up, it may be beneficial to define a restrictive set of rules relating to one particular view or hypothesis about the text—if only as a means of testing the usefulness of that view or hypothesis. It is important to remember that every document type definition is an interpretation of a text. There is no single DTD which encompasses any kind of absolute truth about a text, although it may be convenient to privilege some DTDs above others for particular types of analysis.

At present, SGML is most widely used in environments where uniformity of document structure is a major desideratum. In the production of technical documentation, for example, it is of major importance that sections and subsections should be properly nested, that cross references should be properly resolved and so forth. In such situations, documents are seen as raw material to match against pre-defined sets of rules. As discussed above, however, the use of simple rules can also greatly simplify the task of tagging accurately elements of less rigidly constrained texts. By making these rules explicit, the scholar reduces his or her own burdens in marking up and verifying the electronic text, while also being forced to make explicit an interpretation of the structure and significant particularities of the text being encoded.

2.4.1 An Example DTD

A DTD is expressed in SGML as a set of declarative statements, using a simple syntax defined in the standard. For our simple model of a poem, the following declarations would be appropriate:

```
<!ELEMENT anthology      - -  (poem+)>
<!ELEMENT poem           - -  (title?, stanza+)>
<!ELEMENT title          - O  (#PCDATA) >
<!ELEMENT stanza         - O  (line+)   >
<!ELEMENT line           O O  (#PCDATA) >
```

These five lines are examples of formal SGML element declarations. A declaration, like an element, is delimited by angle brackets; the first character following the opening bracket must be an exclamation mark, followed immediately by one of a small set of SGML-defined keywords, specifying the kind of object being declared. The five declarations above are all of the same type: each begins with an `ELEMENT` keyword, indicating that it declares an element, in the technical sense defined above. Each consists of three parts: a name or group of names, two characters specifying *minimization rules*, and a *content model*. Each of these parts is discussed further below. Components of the declaration are separated by white space, that is one or more blanks, tabs or newlines.

The first part of each declaration above gives the generic identifier of the element which is being declared, for example ‘poem’, ‘title’, etc. It is possible to declare several elements in one statement, as discussed below.

2.4.2 Minimization Rules

The second part of the declaration specifies what are called *minimization rules* for the element concerned. These rules determine whether or not start- and end-tags must be present in every occurrence of the element concerned. They take the form of a pair of characters, separated by white space, the first of which relates to the start-tag, and the second to the end-tag. In either case, either a hyphen or a letter O (for “omissible” or “optional”) must be given; the hyphen indicating that the tag must be present, and the letter O that it may be omitted. Thus, in this example, every element except `<line>` must have a start-tag. Only the `<poem>` and `<anthology>` elements must have end-tags as well.

2.4.3 Content Model

The third part of each declaration, enclosed in parentheses, is called the *content model* of the element, because it specifies what element occurrences may legitimately contain. Contents are specified either in terms of other elements or using special reserved words. There are several such reserved words, of which by far the most commonly encountered is #PCDATA, as in this example. This is an abbreviation for *parsed character data,* and it means that the element being defined may contain any valid character data. If an SGML declaration is thought of as a structure like a family tree, with a single ancestor at the top (in our case, this would be `<anthology>`), then almost always, following the branches of the tree downwards (for example, from `<anthology>` to `<poem>` to `<stanza>` to `<line>` and `<title>`) will lead eventually to #PCDATA. In our example, `<title>` and `<line>` are so defined. Since their content models say #PCDATA only and name no embedded elements, they may not contain any embedded elements.

2.4.4 Occurrence Indicators

The declaration for `<stanza>` in the example above states that a stanza consists of one or more lines. It uses an *occurrence indicator* (the plus sign) to indicate how many times the element named in its content model may occur. There are three occurrence indicators in the SGML syntax, conventionally represented by the plus sign, the question mark, and the asterisk or star.

Like the delimiters, these are assigned formal names by the standard and may be redefined with an appropriate SGML declaration.

The plus sign means that there may be one or more occurrences of the element concerned; the question mark means that there may be at most one and possibly no occurrence; the star means that the element concerned may either be absent or appear one or more times. Thus, if the content model for `<stanza>` were `(LINE*)`, stanzas with no lines would be possible as well as those with more than one line. If it were `(LINE?)`, again empty stanzas would be countenanced, but no stanza could have more than a single line. The declaration for `<poem>` in the example above thus states that a `<poem>` cannot have more than one title, but may have none, and that it must have at least one `<stanza>` and may have several.

2.4.5 Group Connectors

The content model `(TITLE?, STANZA+)` contains more than one component, and thus needs additionally to specify the order in which these elements (`<title>` and `<stanza>`) may appear. This ordering is determined by the *group connector* (the comma) used between its components. There are three possible group connectors, conventionally represented by comma, vertical bar, and ampersand.

What are here called “group connectors” are referred to by the SGML standard simply as “connectors”; the longer term is preferred here to stress the fact that these connectors are used only in SGML model groups and name groups. Like the delimiters and the occurrence indicators, group connectors are assigned formal names by the standard and may be redefined with an appropriate SGML declaration.

The comma means that the components it connects must both appear in the order specified by the content model. The ampersand indicates that the components it connects must both appear but may appear in any order. The vertical bar indicates that only one of the components it connects may appear. If the comma in this example were replaced by an ampersand, a title could appear either before the stanzas of a `<poem>` or at the end (but not between stanzas). If it were replaced by a vertical bar, then a `<poem>` would consist of either a title or just stanzas—but not both!

2.4.6 Model Groups

In our example so far, the components of each content model have been either single elements or #PCDATA. It is quite permissible however to define content models in which the components are lists