the encompassing loop *structure* – *i.e.* in this particular example, the complete loop contents would be returned.

Note that *Star_Base* faithfully returns context even in the processing of complex branching requests. Therefore if, for example, a save-frame pointer is returned as a data value following the processing of a request, the associated save-frame contents will be returned in full so that they are referenced in the returned STAR data structure.

5.2.3.5. Implementation issues

Star_Base is implemented in the C programming language, and exploits Gnu's *flex* and *bison* compiler-compiler system to generate a lexer and parser for the STAR File and a separate lexer and parser for the *Star_Base* query language.

The STAR File parser builds an *in-memory* representation (much like most programming-language compilers) of the file contents, and differs from similar applications that are based on a single pass over a stream (like SAX for XML applications).

While a system like *CIFtbx* retains a block copy of the STAR File in memory, the initial *Star_Base* processing removes all comments and formatting, and stores the meaningful tokens in a binary tree representation. For each STAR File container (global block, data block, save frame, loop or data item) there is a C structure defined. For each of these there are additional structures defined that hold sequences of containers. The nodes of this tree are populated with these structures. Each leaf of the tree is the data item consisting of the data name and its associated value. A binary tree of the global-block sequences is built in reverse order (that is, in an order reverse to that in which they appear in the file), making it simple to identify the global values in scope for a specific data block. It will be recalled that the STAR File semantics require a backward scan through the file to pick up the global blocks in scope.

The binary search algorithm employed is the classic *tsearch* of Knuth (1973), which is part of the standard C libraries. Given modern computer systems, the implementation is extremely fast and efficient. There are no files in existence whose size would test the limits of *Star_Base*.

The use of a binary tree simplifies the process by which a legitimate STAR File is returned as output by *Star_Base* and also how the scope over which the conditionals operate can be controlled by the user. The program stores references to the data nodes of the tree it needs to extract when outputting. Since the location in the original data tree is always stored, the program is easily able to reconstruct the correct structure of the file by walking the tree, identifying the nodes that need to be output in addition to the data.

Star_Base is by default the 'gold standard' for testing other applications for correctness with respect to the syntax and semantics of the STAR File. It can be said that the output of *Star_Base* is not optimal, since it is yet another STAR File and one which is devoid of the original comments and formatting. However *Star_Base* is in essence an API for STAR File applications, rather than a stand-alone program (although it is often used in that way). *Star_Base* was the platform from which BioMagResBank's *starlib* (Section 5.2.6.4) was developed.

5.2.4. Editing STAR Files with *Star.vim*

The *vim* editor supports syntax highlighting for a wide variety of languages through syntax definition rules. The definition rules for STAR Files are simple and small in number. The entire syntax is defined by 19 rules that include the regular expressions for the STAR File keywords, data names, numbers, single- and double-quoted strings, semicolon-delimited text and frame codes. The language for defining a syntax in *vim* is very simple and very powerful. The constructs allow the user to define the syntax precisely enough so that the system does not match patterns within other patterns, unless directed to.

There are three types of syntax items: **keyword**, **match** and **region**.

keyword can only contain keyword characters, no other syntax items. It will only match with a complete word (there are no keyword characters before or after the match). The rule for the STAR syntax is

```
syn keyword strKeyword   global_ save_ loop_ stop_
```

match is a match with a single regular expression (regexp) pattern. The rule for matching on a save-frame code is

```
syn match strFramecode  "$[^ \t]\+"
```

region starts at a match of the 'start' regexp pattern and ends with a match with the 'end' regexp pattern. Any other text can appear in between. There can be several 'start' and 'end' patterns in the one definition. A 'skip' regexp pattern can be used to avoid matching the 'end' pattern. There are a number of character offset parameters that allow the user to redefine the start and end of the matched text given the pattern that matches the regular expression. Quite separately, one can define the region for highlighting, which can be different from the matched text.

The rule that matches a double-quoted string is

```
syn region strString start=+^"+ start=+\s\+"+ms=e
          end=+" +me=e-1 end=+"\t+me=e-1 end=+"$+
           contains=strSpecial " skip=+\\\\\|\\"+
```

In this rule, the beginning of the pattern is a double quote that is either the first character on the line or that has one or more white-space characters before it. The beginning of the matched string (*ms*) is the end of the matched pattern (*e*). That is, the matched string begins at the quote. The end pattern is a double quote followed by a single space, a tab or an end of record. The end of the matched string (*me*) is one character less than the end of the matched pattern (*e*). That is, the trailing character after the closing double quote is not considered part of the matched string. By default the characters between location *ms* and *me* are highlighted. This too can be controlled, and by including $hs = ms + 1$ and $he = me - 1$ the highlighted text would not include the delimiting double quotes.

As these rules are based on regular expressions, there is no possibility of using them to validate the STAR File structure. However, problems in the structure are often identifiable by unexpected or irregular highlighted text [a fact often used in graphical CIF editors to help the user locate visually errors in syntax (see *e.g.* Section 5.3.3.1.4)].

5.2.5. Browser-based viewing with *StarMarkUp*

StarMarkUp is a Tcl/Tk program that takes any STAR File as input and outputs the contents as HTML. The output is a faithful copy of the input, and there is no reformatting or deletion of content.

During transformation, the contents can be cross-referenced against any other STAR File using HTML anchors. This feature is particularly useful when marking up a data file, since the data names contained within can be hyperlinked to their definition in their dictionary. Furthermore, the definitions contained within the dictionary can be hyperlinked to the DDL dictionary. *StarMarkUp* makes no presumptions about the version of DDL employed, the preferred dictionary structure or the specific application