Tasks related to input processing

Uwe Kastens

University of Paderborn
D-33098 Paderborn
FRG

$Revision: 3.7 $

Table of Contents

This library contains modules that support insertion of files into the currently processed input file. The name of the file to be inserted is computed from information of the current input file. Such computations have to be executed immediately while the input is being read (bottom-up in terms of tree construction). This library also contains a module that supports command line arguments of the generated processor which specify directories where files to be included are searched.

`Include` Insert a File into the Input Stream

`CurrTok` Accessing the Current Token

`CmdLineIncl` Command Line Arguments for Included Files

`Include` is the central module to solve the file inclusion task. It is based on the modules `CoordMap` and `GlaCorr` which are automatically instatiated when `Include` is used. (`CoordMap` and `GlaCorr` are not intended to be used directly in specifications.) The module `CmdLineIncl` can be used additionally to enable the command line feature.

The module `CurrTok` provides access to the current token, which has just been accepted by the scanner. It may supply the file name to the function provided by the `Include` module, if that function is issued in computations specified in the concrete grammar rather then in `.lido` computations.

1 Insert a File into the Input Stream

This module supports generating processors that include the contents of a file into the currently read input stream. The effect is the same as that known from the preprocessor `cpp`. There file inclusion is initiated by a `cpp` command like `#include "myfile"`, here it is initiated by a computation of the language processor. It usually takes the file name from the text of a token found in the input stream.

Such facilities are used, for example, in some systems for functional or logical languages, where commands like

```
use ["myfile"]
```

cause the inclusion of the given file at the place of the use command.

The file is included at the current position of the input stream when that computation is executed. That computation can either be specified within the concrete grammar to be initiated by the parser. Or it can be specified in a `.lido` specification. In the latter case the computation has to be executed while input is being read (and the tree is constructed bottom-up) to achieve the desired effect.

This module is instantiated without generic parameters by

```
$/Input/Include.gnrc:inst
```

The module provides the following function:

`int NewInput (char *name)`

It switches input reading to the file named `name`. When its end is encountered input reading is switched back to the next not yet processed token in the file that was the current when `NewInput` was called. The call of `NewInput` yields 1 if the file could be opened; otherwise it yields 0.

This facility is demonstrated by inserting a construct into our running example that allows to comprise sets of declarations on separate files and include them in the program text. Such declaration files may be used to specify common interfaces.

The file inclusion construct is introduced by the concrete productions

```
Declaration:   FileInclusion ';'.
FileInclusion: 'include' FileName.
```

We add a scanner specification for the terminal symbol `FileName`:

```
FileName: C_STRING_LIT       [c_mkstr]
```

Hence, an input program may contain file inclusion commands like `include "myfile";` wherever a `Declaration` is allowed.

The following `.lido` specification attaches the call of `NewInput` to the `FileInclusion` context:

```
ATTR InpFileEx: int;
RULE: FileInclusion ::= 'include' FileName COMPUTE
      .InpFileEx = NewInput (StringTable (FileName)) BOTTOMUP;

      IF (NOT (.InpFileEx),
      message (ERROR, CatStrInd ("can not open file ", FileName),
               0, COORDREF));
END;
```

The file name is specified to be the text of the `FileName` token. The computation has to executed while the input is read and the tree is built bottom-up. Hence it is specified `BOTTOMUP`.

Note: The above computation is executed when the `FileInclusion` tree node is built. Then the last token read from the current input stream is the terminating `;` of the `Declaration`, due to the lookahead of the parser. Hence, the contents of the included file is being processed as if it stood immediately following that `;` of the `Declaration`. Due to this reason it was essential to use two productions in the concrete grammar as stated above. A single production like

```
Declaration:  'include' FileName ';'.
```

instead would delay insertion until the token following that `;` has been read.

2 Accessing the Current Token

This module provides a function `GetCurrTok` that can be used to access the string of the current token read from input. The function is used for example to access the name of a file to be included if the switching computation is issued by the parser as specified in the concrete grammar.

This module is instantiated without generic parameters by

```
$/Input/CurrTok.gnrc:inst
```

The signature of the function is

`char *GetCurrTok (void)`
: It accesses the string of the current token, stores it in memory, and yields a pointer to the stored string as result.

In our running example we could achieve the same effect of file inclusion as described in (see Chapter 1 [Include], page 2), by specifying a computation in the concrete grammar rather than in a `.lido` specification:

```
Declaration: 'include'
             &'ChkFileOpnd(NewInput(GetCurrTok()), GetCurrTok());'
             FileName ';'.
```

The token of `FileName` that immediatly follows the computation is accessed. The call of `ChkFileOpnd` is assumed to check for success of the `NewInput` call.

3 Command Line Arguments for Included Files

Using this module introduces processing of a command line option into the generated processor: If the processor is called

```
processor.exe -Idirname filename
```

then files to be included are searched in the current directory and in `dirname`. This module provides specifications for the Eli's CLP tool to achieve the effect.

This module is instantiated without generic parameters by

```
$/Input/CmdLineIncl.gnrc:inst
```

The module provides a function that can be used together with calls of the `NewInput` function:

`char * FindFile (char *name)`
: It searches the file named `name` in the current directory and in the directories given by `-I` command line arguments.

In order to use this facility the `FindFile` function has to be applied to the argument of the function `NewInput`. Hence, the examples of the two previous sections had to be rewritten as

```
ATTR InpFileEx: int;
RULE: FileInclusion ::= 'include' FileName COMPUTE
  .InpFileEx = NewInput (FindFile (StringTable (FileName))) BOTTOMUP;

  IF (NOT (.InpFileEx),
  message (ERROR, CatStrInd ("can not open file ", FileName),
           0, COORDREF));
END;
```

for the case of the `.lido` specification of input switching as shown in (see Chapter 1 [Include], page 2), or

```
Declaration: 'include' &'NewInput(FindFile (GetCurrTok()));' FileName ';'.
```

for the case of the concrete grammar specification of input switching as shown in (see Chapter 2 [CurrTok], page 4).

Index