

## CS3215: Specifying PKB API: Guidelines and Example

In Assignment 2, you will specify some of the PKB interfaces (API). You will complete specifications of PKB API in Assignment 3, based on supervisors' feedback.

This document contains guidelines for writing PKB API specifications.

### 1. General guidelines:

This template for PKB API specs is given in the Handbook:

```
ADT name {
  Overview: explain the rationale and responsibility of a component or ADT
  Public interface: here you will list interface operations documented as follows
  Operation header: returned-value operation-name (list of parameters with names and types)
  *Parameters (optional): unless it is not clear from the header, describe parameters
  Description: here you describe the effects of the operation (what the operation does) in
    terms of parameters, returned value and whatever else you need to explain the
    meaning of the operation; it is most important that you describe both normal and
    abnormal behavior (handled by assertions and exceptions)
}
```

1. Briefly review sections 9.6 and 9.7 in the Handbook.
2. *Overview* clause: Should be brief, but for more complex ADTs, such as AST, it useful to provide some sketches and/or examples in the *Overview*. Any conventions, names specific to a given ADT should be also described in the *Overview*.
3. Interface operations for ADTs.
  - a) Interface operations should be well chosen to provide an easy way to work with an ADT. The set of operations should be complete in the sense that SPA components that work with a given ADT (create it or access information from it) can do so by means of calling ADT interface operations.
  - b) Interface operations for ADTs should be abstract:
    - i) *implementation-independent*: should not make unnecessary assumptions about implementation details such as the actual data structures chosen for ADT implementation
    - ii) *programming-language-independent*: avoid specific C++ types
4. Adopt standard way of documenting ADTs and use them consistently throughout all PKB API specifications:
  - a) Use symbolic type names (e.g., INDEX, PROC, AST\_NODE, CFG\_NODE, etc. ) rather than specific C++ type names (e.g., int).
  - b) Adopt naming conventions (for operation names, arguments, etc.).
  - c) There should be uniformity in the description of ADTs: similar situations should be described in similar way. For example, VarTable should be similar to ProcTable. ADT Modifies for statements should be similar to ADT Modifies for procedures. ADTs Modifies should be similar to ADTs Uses.
5. *Description* of interface operations.

- a) Keep the set of operations for each ADT simple.
  - b) Give names to arguments of interface operations and use these names in the operation *Description*. This helps to describe operations in brief and clear way.
  - c) *Description* of interface operations should be simple, concise but precise, unambiguous, complete.
6. No need to describe obvious parameters under the *Parameters* clause
  7. Pay attention that interface operation specifications are understandable to others.

## 2. Sample specifications: VarTable

This an example only. You are not asked to follow blindly this example.

<b>VarTable{</b>
<i>Overview:</i> VarTable is used to keep all the variables that appeared in the program
<i>Public Interface:</i>
<b>INDEX</b> insertVar ( <b>STRING</b> varName); <i>Description:</i> <b>If</b> 'varName' is not in the VarTable, inserts it into the VarTable and returns its index. <b>Otherwise</b> , returns its INDEX and the table remains unchanged. <b>OR: If variable is in the table, returns -1 (special value) and the table remains unchanged.</b>
<b>INTEGER</b> getSize(); <i>Description:</i> Returns the total number of variables in VarTable
<b>STRING</b> getVarName ( <b>INDEX</b> ind); <i>Description:</i> Returns the name of a variable at VarTable [ind] If 'ind' is out of range, Throws: InvalidReferenceException
<b>INDEX</b> getVarIndex ( <b>STRING</b> varName); <i>Description:</i> <b>If</b> 'varName' is in VarTable, returns its index; <b>Otherwise</b> , return -1 (special value)
}