Riddle: Region Internal Database Description LanguagE

May 27, 1986

There are two forms of Riddle: "Human-Level Riddle" and "Machine-Level Riddle". The latter is a roughly syntax-free, highly encoded form of the latter, and is optimized for easy parsing and terseness. The former is what people will mostly use.

This is the definition for "Human-Level Riddle" (which we call simply "Riddle"):

A Riddle file consists of one or more Riddle statements. There are five types of statements: include, assignment, macro-assignment, object-use and object-definition.

The include statement has the form:

```
        include "<filename>";
```

where <filename> is the name of a file to include, expressed as a string constant. It does what you expect.

The assignment statement has the form:

```
        <name> = <expression>;
```

while the macro-assignment statement has the form:

```
        <name> @= <expression>;
```

The two are similar in that they both assign a value to an identifier, <name>. However, the nature of the value is different: The ordinary assignment statement computes the value of <expression> and makes that computed value the value of <name>. Further references to <name> result in this value. The macro-assignment statement, on the other hand, merely assigns the expression <expression> itself to be the value of name, and each time <name> is referenced, <expression> is recomputed and the resulting value used. In other words, macro-assignment defers the computation of <expression> and always results in the value that <expression> would have if it were substituted directly for <name>. If the values of any of the terms in <expression> change between references to <name>, then the corresponding value that results from a reference to <name> will also change.

The object-use statement describes an instance of an object which is to be inserted in the present context. Its form is:

```
        <objectCall> <objectClass> <globalObjectId> {
                <properties>
                <contents>
        }
```

The interpretation of this is: create an object of the sort indicated by <objectCall> with class <objectClass>, assign it the global object identifier <globalObjectId> and give it the property values specified by <properties> and the contents specified by <contents>. Any of these elements by be omitted, except that (ultimately) you must specify a class for the object, either directly, or somewhere in the chain of <objectCall>s.

<objectCall> has the form:

```
        @ <name>
```

and, if given, indicates that the object being defined is to be based on the previously defined object <name> that was created with an object-definition statement (described below). If omitted, the object specification is taken solely from the specified information that follows.

<objectClass> is an <expression> that specifies the class number of the object being created. If a class is not given, it is derived from <objectCall>. Ultimately, all object must have a class specified.

<globalObjectId> has the form:

```
        $ <expression>
```

The expression must result in a number. This number will become the object's database identifier key. It can be omitted (and, I think, usually will be), in which case the Riddle processor will assign one.

<properties> are the properties to be associated with the object. This is zero or more elements of the form:

```
        <offset>: <data>;
```

where <offset> is an expression resulting in a numeric offset into the object descriptor (usually this will be a single identifier, whose value is set up ahead of time, naming the offset of an object state variable). <data> is one or more <expression>s, separated by commas. Each property element assigns the given data elements into the successive bytes of the object descriptor indicated by <offset>.

<contents> has the form:

```
        [ <objectUse> <objectUse> ... ]
```

i.e., one or more object-use statements, enclosed in brackets. The object-use statements define objects which are to be created as the contents of the object being declared here. If no <contents> element is specified, then the object will have no objects in its contents vector (which it shouldn't have anyway unless it is some sort of container).

The object-definition statement has the form:

```
        <name> # <objectUse>
```

It defines <name> to be an object of the form indicated by the object-use statement <objectUse>. This object-use statement is exactly as described above.

Expressions can take the following forms:

```
        <name>                                  the value of identifier <name>
        <number>                                a numeric constant
        <string>                                a string constant
        ( <expression> )                        the value of the expression
        ~ <expression>                          bitwise complement
        - <expression>                          arithmetic negation
        <expression> + <expression>             addition
        <expression> - <expression>             subtraction
        <expression> * <expression>             multiplication
        <expression> / <expression>             integer division
        <expression> % <expression>             integer modulus
        <expression> & <expression>             bitwise AND
        <expression> | <expression>             bitwise OR
        <expression> ^ <expression>             bitwise XOR
```

A <name> can be any string of alphabetic characters, digits, and ’_’, that does not start with a digit.

A <number> can be any of the standard number formats we use: a decimal number (any sequence of decimal digits that does not start with 0), an octal number (any sequence of octal digits preceded by ’0’), a hex number (any sequence of hexadecimal digits preceded by ’0x’ or ’0X’), a binary number (any sequence of 0 and 1 preceded by ’0b’ or ’0B’), a quarter (any sequence of base-4 digits preceded by ’0q’ or ’0Q’) or an ascii character enclosed in single quotes (with the usual \ escape sequences).

Any <expression> can be followed by a <typeSpecifier> of the form:

```
        .b
        .w
        .l
```

".b" indicates that the value is to be treated as a byte (the default). ".w" indicates that the value is to be treated as a word (two-byte quantity, high order byte first). ".l" indicates that the value is to be treated as a long (four-byte quantity, high byte to low byte).

A <string> is any sequence of characters enclosed in double quotes, with the usual \ escape sequences.

At the top-level, the only object-use statements allowed are ones that create region objects, i.e., ones in which the <objectType> is 0.

Object-definition statements may be used to declare completely or partially specified objects for use inside the regions.

Simple, no?

Also, of course, whitespace may be freely used for formatting, and comments may be inserted at almost any point bracketed by "/*" and "*/".

This is the definition for "Machine-Level Riddle":

Semantically, machine-level Riddle is a subset of the above, with the following differences:

```
        1) No assignment, macro-assignment or include statements are allowed.
        2) All expressions must be replaced with simple decimal numbers.
        3) All type specifiers must be explicit.
        4) The containment relationship is "unfolded", i.e., objects indicate
                which other object they are contained in, instead of listing
                the objects they themselves contain.
```

In other words, it’s just like the above, except that EVERYTHING must be already evaluated.

Syntactically, it is quite different, being encoded in a form that requires essentially no parsing beyond that already performed by conventional formatted input routines.

A Machine-level Riddle file consists of a number of lines, each of which contains one or more decimal integers, separated by spaces. Each line has the form:

```
        <codeNumber> <otherStuff>
```

Where <codeNumber> is a number that encodes the meaning of the line, and <otherStuff> is any number of additional numbers that are interpreted on the basis of <codeNumber>. By this means we can have "statements" without resorting to parsing. There are two basic machine-level Riddle "statements": object-use and object-definition.

The object-definition statement consists of a line of the form:

```
        1 <objectNumber>
```

followed by an object-use statement.

The ’1’ encodes the fact that this is an object-definition statement, and <objectNumber> is the identifier assigned to the object being defined. The body of the object-use statement that follows becomes the definition of object number <objectNumber>.

An object-use statement consists of a line of the form:

```
        2 <objectNumber> <class> <globalId> <container>
```

followed by zero or more property definitions (described below), followed by a line of the form:

```
        3
```

The ’2’ encodes the start of the object specification. <objectNumber> is the number of an object previously defined using an object-definition statement, on top of whose definition this object is to be constructed. <objectNumber> should be ’-1’ if no earlier object definition is to be referenced. <class> is the class number of the object being created, or ’-1’ if no class number is to be specified at this time. <globalId> is the global database identification number of the object being created, or ’-1’ if no such identifier is to be assigned at this time, or ’0’ if one should be generated and assigned automagically. <container> specifies the another object in which this object is to be contained. A negative number of the form, -N, indicates that this object is contained by the object with global object identifier N (this way we can refer to objects outside the Riddle file that are already in the database). A positive number M indicates the ordinal of the containing object in the Riddle file, i.e, the object described by the Mth line that started with a ’2’. A ’0’ indicates that this object is contained by the database at large. In such a case the object ought to be a region.

The ’3’ encodes the end of the object specification.

A property definition specifies one of the properties of an object. It is a line of the form:

```
        4 <offset> <dataCount> <data>
```

The ’4’ encodes that this line is a property definition. <offset> is a number that is the offset into the object instance vector corresponding to the property being specified here. <dataCount> is the number of data values being assigned. Its value is usually 1. <data> is precisely <dataCount> number specifiers that indicate the value(s) of the property specified by <offset>. Each such number specifier has the form:

```
        <type> <value>
```

Where type is 1, 2 or 4, indicating byte, word or long values respectively, while value is the actual number value.

For the convenience of the processor that reads this stuff, there are also lines of the form

```
        5 <number>
```

Which indicates that no further contents for object <number> will be found in this file. <number> is encoded as the <container> field above is encoded: a negative number -N indicates that there are no further contents objects for the object with global id N, while a positive number M indicates that there are no further contents for the Mth object specified in the file.