FiM++ Proposals

Switch Statement

By Kyli Rouge:

```
In regards to value:
On the 1st hoof...
I said "Case 1".
On the 2nd hoof...
I said "Case 2".
If all else fails...
I said "Default".
That's what I did.
```

Arrays

By Kyli Rouge:

- ```
  Did you know that cake has many names?
  cake 1 is "chocolate".
  cake 2 is "apple cinnamon".
  cake name 3 is "fruit".
  I remembered cake 2.
  ```
  - Prints "apple cinnamon".
- ```
  Did you know that cake has the names "chocolate" and "apple cinnamon" and
  "fruit"?
  ```
- ```
  Did you know that my checklist checklist has phrases of phrases?
  my checklist checklist phrase 1 phrase 1 is "make a checklist".
  my checklist checklist phrase 1 phrase 2 is "write the checklist".
  my checklist checklist phrase 1 phrase 3 is "seal the checklist in wax".
  my checklist checklist phrase 1 phrase 4 is "send the checklist to you for
  ```
  ```

```
approval".
my checklist checklist phrase 2 phrase 1 is "water the potted plants".
my checklist checklist phrase 2 phrase 2 is "put away recently checked-in
books".
my checklist checklist phrase 2 phrase 3 is "go into town to meet Fluttershy".
my checklist checklist phrase 2 phrase 4 is "buy more parchment and ink".
I remembered my checklist checklist phrase 2 phrase 3.
```

- `Did you know that the addition table has numbers of numbers?`

```
the addition table 1 1 is 2.
the addition table 1 2 is 3.
the addition table 1 3 is 4.
the addition table 2 1 is 3.
the addition table 2 2 is 4.
the addition table 2 3 is 5.
the addition table 3 1 is 4.
the addition table 3 2 is 5.
the addition table 3 3 is 6.
I remembered the addition table 2 3.
```

  - Prints "5"

By Evan Rittenhouse:

Creating an array:

~~`I made a list of numbers called my combination containing 2 and 3 and 4.`~~ ~~`(I like this version of the literal)`~~

```
Did you know my combination is 2 and 3 and 4? P.S. In this case the compiler
recognizes the array literal and sets my combination to an array<number> type.
```

Referencing elements:

~~*<variable name><whitespace>* number *<whitespace><value: number><punctuation>*~~

~~In this case number dereferences a value in an array. a number is a literal numerical type. numbers refers to an array containing numerical types.~~

If no array type is given it is a `nothing` type. This array type is able to contain any primitive data type (or objects when they're implemented).

Multidimensional arrays:

Multidimensional arrays are technically just lists of lists. Remember the episode where Twilight was so organized she had a to do list with "make my to do list" on it? We could try this:

```
Did you know my checklist checklist is a list of lists of words?
P.S. This makes a two dimensional array. The sub-arrays are string-only arrays.
P.P.S. Alternatively you can declare the sublists as nothing type
Did you know my checklist checklist is a list of lists?
```

Referencing (and proposed literals):

Probably something like this:

```
Did you know the index of my book is the list "Canterlot history" and "Pre-Equestrian society" as well as "Page 214"
and "Page 338"?

I wrote the index of my book item 1,2.
```

OR better yet

```
I wrote the index of my book item 1.2.
```

Setting a function's return type:

By Evan Rittenhouse:

We could possibly use this instead of the one proposed in the main document I basically switched the positions of the return type and function name:

```
I learned to get a number with fancy mathematics using a number and a number.
I learned to get <return type> with <function name> using
<parameters><punctuation>
```

The user can access the parameters as an array (the parameters are practically an array literal). However, I still need to figure out what the name of said array would be.

Reading input directly into a variable, string manipulation

By Anwssl:

Idea: Have a way to combine output and input to create a prompt

- `I asked Spike "How many gems are left?".`
  - Prints "`How many gems are left?`" and then waits for the user to input a number, which it stores in the variable "`Spike`".

By Evan Rittenhouse:

Perhaps different parts of strings could be accessible. Tokenizer variable types?

For example, word, sentence, and letter could all be token types which let the program parse parts of strings in foreach loops. (see the for each loop section)

Maybe `and` can concatenate strings?

If we don't have one already, we should add a `letter` variable type for the representation of individual characters in a string.

Casting

By Kyli Rouge:

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

Example:

```
Did you know that Pinkie1 likes words?
Did you know that Pinkie2 likes the number 1?
Pinkie1 now likes the word Pinkie2.
```

- Sets the value stored inside `Pinkie1` to be a string representation of the value in `Pinkie2`, so "`1`".

For loops

By Kyli Rouge:

See my proposal on the wiki

For each loops

By Evan Rittenhouse:

For every <data type> in <variable name> ~~I would~~:

...

That's what I would do.

The individual elements in <variable name> are accessible via a temporary variable called "the item" , "it", "that <data type>" or something similar. For each loops can also work as string tokenizers.

For example:

```
Did you know Applebloom's essay topic is "My Sister, Applejack"?

For every word in Applebloom's essay topic,

I would write that word and ",".

That's what I would do.
```

The output is "My,Sister,Applejack"

By Kyli Rouge:

See my proposal on the wiki

Regular Expressions:

By Evan Rittenhouse:

We have to figure out objects before I get into this.

String manipulation: Replace all curse words with "buy some apples"

By Kyli Rouge:

```
        (will be completed once string manipulation is implemented)
<insert import of a string helper class>

Dear Curse Words: I'll be rid of you one day!

Today I learned how bad you are.
Did you know that you are a word?
I heard you.
Did you know that goodness is an argument?
As long as <insert statement about looping through words in the string>:
("As long as you included/had certain words:" Where "certain words" is the array of
curse words, and "included" or "had" would induce a method to check for certain array
values.)
goodness becomes how to check curse words using <a particular word in the string>.
("how to check curse words" is a method in the "Curse Words" class)
If goodness is correct then: <that particular word in the string> becomes "buy some
apples".
(needs an else statement...)
That's all about how bad you are.

Your faithful student, Digit Shine.
```

Importing external Classes

By Kyli Rouge

- Remember when I wrote about<whitespace>**<class name>**<punctuation>
- Remember when I wrote about **Hello World**?

```
Dear Princess Celestia: Echoing.

Today I learned how to say things twice.
I said Hello World. (Imported from the Hello World class)
I said Hello World.
That's all about how to say things twice.

Your faithful student, Kyli Rouge.
```

Anwssl's random list of random ideas/questions that he comes up with when he's bored and adds to whenever he thinks of new random ideas/questions

by Anwssl/Draco Icarus

(seriously, feel free to move this wherever is convenient and/or comment/change whatever just mark your change somehow unless it's a spelling/grammar correction)

- The name for the external math class being "complicated math" or "math" (for higher complexity math like exponents and square roots and whatnot) Current ideas: "complicated math" (0 votes); "math" (0 votes); "Fancy mathematics" (3 votes)
  - Resolved: Current choice for external math class name: "Fancy mathematics"

- List of assigned synonyms for keywords to remove stress from the compiler and make updating the compiler easier ("is" being list element 1.001 and "was" being list element 1.002, for example; both are in the list section "1.###" and thus pertain to the same word/function)
  - Resolved: <no description>

- Certain words or sections of phrases that are ignored by the compiler (would ignore "made" in "made of" for "types made of types" and just interpret "types of types")
  - Resolved: I was derping, these would be synonyms.

- Would there ever be a need to have a method that starts like this? : "Applejack learned how to fly." Possibly referencing a method for a certain variable?

- Possible new syntax for "for" loops; rather, a proposal for all loops, but specifically "for" loops (it would work best with them). My idea is that it'd set itself up, like, you'd say something like "I <obligatory action> 5 times." and it would set up a for loop that has 5 iterations. So it'd end up changing it to something like this (in java): for (int i = 0; i < 5; i++) {<loop body>}. However, I can't think of any statements to end the loop on other than this: "That's what I <obligatory action>." Any suggestions?
  (examples of <obligatory action>: "did";"read";"saw" etc.)

- Custom objects are needed, so I'm going to try to address that here. A custom object, by tradition is instantiated by using a custom class as an interface/guide (note to self and readers: see Kyli's documentation about interface declaration). Like in java, declaring a custom object would require constructing a variable of the interface class, and thus would have syntax like that of variable declaration (i.e. "Did you know *Applejack likes* 42?"). However, the custom object would have to be of type <customClass> and thus the type in the variable declaration statement would be changed to whatever the custom type is ("Did you know Applejack's favorite horseshoe is the Horseshoe

back-left?”). This requires the interpreter/compiler to allow for the creation of objects without a given primitive type (i.e. “boolean”, “char”, “int” etc.). But wait, isn’t this already done through the implementation of a default null type until specification or declaration? Actually, I don’t think it is. But that’s not necessary because all types root back into bits and most custom types are combinations of the primitive types in some form or another that doesn’t require the custom object to show itself as a unique type. So what am I even talking about anymore? Well, I’m not sure, back to the topic. Because the type would be custom, we must assume that it has to be placed specifically in substitute for the object type in the variable declaration phrase. If the original syntax would be this “Did you know <varName> <type> is <value>?” then the new syntax with the custom object (in this case, object “Horseshoe”) and the given arguments would be “Did you know <varName> Horseshoe is back-left?”. And that would call the custom class “Horseshoe” and give it the parameter(s) “back-left” to construct itself. (I’m typing this as I go by the way, so don’t be surprised if I go back and readdress stuff.) Now I’m going to try to write a code segment to explain what I mean: (I still haven’t gone back and checked my syntax against Kyli’s interface documentation)

    - Dear Princess Celestia: a Horseshoe.
      (This is an example of a sole custom object. Any advice would be appreciated.)
      I learned about a Horseshoe using a word. That word was boot. That’s all I learned about a Horseshoe. I learned what a Horseshoe was with the word boot using boot. Did you know boot becomes “the horseshoe on the “ and the word boot and “ hoof”? Then you get boot. That’s all I learned about what a Horseshoe was.

      Your faithful student, Draco Icarus.

  And here’s an example of the implementation of this class in a main class:

    - Dear Princess Celestia: On The Subject Of Horseshoes:
      Today I learned about Applejack. Did you know Applejack’s favorite horseshoe is the Horseshoe ”front-right”? I said what Applejack’s favorite horseshoe was. That’s all I learned about Applejack.

      Your faithful student, Draco Icarus.

  Anyways, I hope this code explains what I mean better than my huge paragraph does. Any suggestions and/or criticisms are (as always) appreciated.

- Personally, I like the idea of the source code file being a *.pony file, but that might just be me.

- What do you think of a “goto” statement like in DOS? Maybe like, “I then went back to the 1st thing I did.” or something to that effect. And then I think about labels in the program, maybe statements declaring time could be used. “It was NOON when I finished.” and then something like “I went back to what I had done at NOON.” Of course, my syntax here would be atrocious, but you get the point. I’m not even sure if this has been addressed before or not, lol. And this sounds a lot like “continue” in java... While I’m on the subject of cheap programming shortcuts for the lazy coder, what about treating input strings as reference names? Like, in java, prompt for input and then call method with a methodName that matches the inputted string. Or something of that nature. You get my point.
    - Resolved: use a method

Compiler/Executor

By Evan Rittenhouse:

I’ve been playing with the idea of just making both a compiler AND an interpreter. Therefore we wouldn’t have to

compile the .fim files into java .class files (and we wouldn't need to look up formatting in the .class file)
Of course, *.fim files are source code, but the compiler is called Spike.jar, compiled files have a .burp file extension, and the interpreter is called Celestia.jar.
Now, the compiled *.burp files will be in a tag-based format similar to HTML, but is not meant to be human-readable so there are no spaces. I can't really explain it, so I'll put up my compiled ideas on the subject.

The tags are as follows:

- cls- Class declaration
- cle- End class declaration
- clr- class reference (followed by either a variable reference or a function reference IFF the function/variable exists outside the main class [class ID 1] which extends Princess Celestia)
- vd- Variable declaration, followed by a number (the variable's ID)
- vde- Variable declaration end
- vr- Variable reference
- fd- Function declaration (followed by an ID given by the compiler)
- fde- Function declaration end
- frt- function return type (followed by an unclosed literal tag denoting type)
- fp- function parameters
- fpe- function parameter end
- fa- function access (executes a function, followed by the function's ID) (usually preceded by a class reference)
- fae- function access end
- l- literal (followed by a character representative of the literal type. e.g. ln for numbers, lb for booleans, ls for strings, etc.)
- le- literal end
- m- main function
- me- main function end
- o- output (followed by a literal or variable reference)
- i- input (followed by a variable reference)
- s- statement
- se- statement end
- n- null
- vs- set value
- rs- result (of addition, subtraction, etc. Order of operations preserved using prefix notation).
- rse- result end
- tvd, tvde, tvr, etc. - temporary variable calls. Defined by an ID and an unclosed literal tag denoting the temporary variable type.

Therefore, a hello world program's *.burp file would look like this:

cls1msols48656c6c6f20576f726c6421lsesemecle

In a more human- readable format:

```
cls 1
    m
        s
            o
                ls 48-65-6c-6c-6f-20-57-6f-72-6c-64-21 lse
        se
    me
cle
```

And a program that involves a function taking two parameters and adding them looks like this:

cls1fd1frtlnfptvd1lntvdetvd2lntvdefpesfrrs+tvr1tvr2rsesefdemsofa1ln2lneln3lnefaesemecle

In a human readable version:

```
cls 1
    fd 1
        frt ln
        fp
            tvd 1 ln tvde
            tvd 2 ln tvde
```

```
        fpe

        s
            fr rs
                + tvr 1 tvr 2
            rse
        se
    fde

    m
        s
            o
                fa 1
                    ln 2 lne
                    ln 3 lne
                fae
        se
    me
cle
```

By Kyli Rouge:

Supercompressed .FR file

A .FR file (abbreviation for "**F**riendship **R**eport") is proprietary FiM++ bytecode and must be read and executed by a virtual machine.

- Whitespace and Comments are entirely removed (except for programmer name)
- Keywords are represented by Unicode characters starting at `U+0001`, which represent their function, not the actual used keyword (any two synonyms are compiled to the same character).
    - Binary prefix operators which have a partner infix operator (think add 12 and 2) are converted to a single infix operator (think 12+2)
- Variables, class names, and method names are compiled into hex digits
    - surrounded by the Unicode character `U+F000`
- Literals are kept as-is, with any source quotes removed
    - surrounded by `U+F001`
- Punctuation is compiled into `U+FFFF`

Example

Hello World.FPP (190B)

```
Dear Princess Celestia:Hello World!

Today I learned how to say hello world!
I said “Hello, World!”!
That's all about how to say hello world.

Your faithful student, Kyli Rouge.
```

Would compile into: Hello World.FR (93B (51% compression); follow the link to download the actual thing, as Google Docs doesn’t display it properly)

```
0□1□2□+Hello, World!□!2□ □ Kyli Rouge□
```

Compile to JS

Why not? More at: http://fimpp.wikia.com/wiki/FiM%2B%2B_Wiki:Proposals/Compiler#.JS_file