Bill

Language Primer

Revision 1

J. Smith

Name:

bill Beginner – Intermediate Learning Language

Purpose:

It is a general purpose statically typed, with dynamic types, easy to use language.

Key uses:

- Learning basic to intermediate level programming.
- Pursuing enthusiast programming.
- Simple data management.
- Provide a “stepping stone” for transition to more complicated languages.

Features:

- portable
- a variety of useful data types
- extend-able data types
- plenty of conversion functions (dynamic types)
- simple to use
- object oriented foundation
- functional programming friendly
- constants
- all data types have string representations
- ease of static typing
- strong static and dynamic typing
- even primitives are objects
- manual garbage collection
- pass by value
- #, //, or /* */ denotes comments

all objects have the following default methods:

- constructor
- destructor
- string representation
- type

Types:

As all dynamic types are classes, they each have string representations. This allows for concatenating a number and a string, by coercion, as seen in the code samples.

Dynamic types are 64 bit. This ensures compatibility with 64 bit machines.

Built in:	
Primitives:	
uint64	static
uint32	static
uint16	static
uint8	static
int64	static
int32	static
int16	static
int8	static
float64	static
float32	static
char	static
null	static
sequence:	
array	static
char array	static
Note: the types listed below do not exist in Subset.	
Primitives:	
bool	dynamic
Integer	Dynamic (64)
float	Dynamic (64)
Compound:	
set	dynamic
sequence:	
string	dynamic
list	dynamic
tuple	dynamic
mapped:	
dictionary	dynamic
Standard Library:	
frozenset	dynamic
stack	dynamic
queue	dynamic
deque	dynamic

Figure 1: Bill Types

Reserved Words:

if	*condition*
elsif	*subsequent condition*
else	*convert to float (dynamic types only)*
return	*end a function*
break	*exit loop*
continue	*skip to next iteration*
try	*begin try block*

Likely there are more. The rule is function over keyword, where reasonable.

Note: *Reserved words in italics, are from Subset.*

Built in Functions:

str()	convert to string (dynamic types only)
int()	convert to integer (dynamic types only)
float()	convert to float (dynamic types only)
tuple()	convert to tuple (dynamic types only)
write()	*print (without newline)*
writeln()	*print (with newline)*
while()	*while loop*
type()	*get an object’s type*
for()	for loop
foreach()	foreach loop
catch()	*catch exception*
throw()	*throw exception*
exit()	*end program*

Likely there are more. The rule is function over keyword, where reasonable.

Note: *Functions in italics, are from Subset.*

Standard Library:

In addition to the usual contents:

math:

trig functions

string and regex:

chomp

match, replace, etc...

data types:

frosenset

stack

queue

deque ???

tools:

a unit test framework

a doxygen compatibility tool

file I/O

json

cpp_interface ???

Sample Code:

```
#!/usr/bin/env bill

# main.bill
# aka hello world

fun noval main():
{
    write('H');     // I haven't worked out char arrays yet.
    write('e');
    write('l');     #  This works too.
    write('l');
    write('o');     /* As does this. */
    write(' ');
    write('W');
    write('o');
    write('r');
    write('l');
    writeln('d');   // This one adds a newline.
    exit();         // defaults to 0
}
```

```
# Sample.bill
# sample class definition file.

# NOT in Subset!

/*  @class Sample
 *  @brief sample class example
 */
class Sample()
{
    /*  constructor
     *  @brief  set up vars
     *  @param  name    (str)       name of person
     *  @param  name    (uint8)     age of person
     *  @param  name    (dynamic)   other data      [default = false]
     */
    public fun noval construct(string: name, uint8: age = 0, dynamic: other =
false):
    {
        private var private string this.name    = name;
        private var uint8 this.age              = age;
        private var string this.other           = other;
    }

    /*  string  representation
     *  @brief  introduction
     *  @return (str)
     */
    public fun str str():
    {
        var string result   = 'Hello, my name is: ';
        result             +=  this.name;
        result             +=  '. My age is: ';
        result             +=  this.age;
        return result;
    }

    /*  getName
     *  @brief get name
     *  return (str)
     */
    public fun str getName():
    {
        return this.name;
    }

    /*  getAge
     *  @brief get age
     *  return (uint8)
     */
    public fun uint8 getAge():
    {
        return this.age;
    }

    /*  getOther
     *  @brief  get other
     *  return (str)
     */
    fun getOther():
    {
        return this.other;
    }

    /*  setAge
     *  @brief set age
     */
    public fun noval setAge(uint8: age)
    {
        this.age = age;
    }

    /*  destructor
     *  @brief del vars
     */
    fun void destruct()
    {
        del this.name;
        del this.age;
        del this.other;
    }
}
```

```
# whileloop.bill
# while loop syntax sample

fun noval main(argv):
{
    while(true):
    {
        # This is the loop that never ends.
    }

    while(1 > 3):
    {
        # This loop is skipped.
    }

    var dynamic int limit   = argv[1];
    var dynamic int i       = 0;
    while(i < limit):
    {
        writeln(i);
        i ++;
    }
}

# forloop.bill
# for loop syntax sample

fun noval main(argv):
{
    for(var int32 i = 0; i < 1; i += .01):
    {
        writeln(i);
    }

    # foreach (sequence types only)
    foreach(var str arg, argv):
    {
        writeln(arg);
    }
}
```

```
# exceptions.bill
# for exceptions sample

fun noval main(argv):
{
    fun noval mistake():
     {
        try:
        {
            doThis();
        }
        catch(RuntimeError e):
        {
            throw(RuntimeError(e));
        }
    }

    try:
    {
        mistake();
    }
    catch(RuntimeError e)
    {
        writeln(e);
        exit(1);
    }

    # Results are in a generic message, as none is defined here, and a backtrace.
}
```

```
# if_regex.bill
# regex and conditional sample

# regex NOT in Subset!

import Regex.bill;

fun noval main():
{
    var str pattern     = '/+d/';
    var str text        = 'Hello, my favorite number is 123.";
    NumberRGX           = Regex(pattern);
    var bool is_match   = NumberRGX.match(text);

    var replacement     = 'xyz';
    if is_match:
    {
        writeln(NumberRGX.replace(text, replacement));
    }
    else:
    {
        writeln('Sorry, no match.');
    }
    # result is "Hello, my favorite number is xyz."
}
```