Don Rodolfo P. Abril
Jerico L. Silapan

CS 150 WFVW
Machine Problem: Language Design

Asin USER MANUAL

A friendly note

We think that it would be best for this user manual to be read alongside the language design and functionalities report, so that information could be corroborated. ☺

Dependencies

As prerequisites, the software listed below should be installed on your system so that you could run the Asin interpreter.

- Python 3.5.2 or Python 3.4.3
- ply 3.10
- PyInstaller 3.3.1

The program will NOT work with Python 2. It is untested for Python 3.3 and other 3.4 and 3.5 versions, but should work for those (PyInstaller's constraints are that Python 3.3 – 3.5 be used when programming using Python 3).

Using the Asin interpreter

To use the interpreter, enter the following command in your terminal.

Unix-like systems:	`./asin <filename>`
Windows:	`asin.exe <filename>`

The file needs no extension such as Python's ".py", or C's ".c". However, to really get into our language, we ourselves like to use the superficial file extension ".asin".

A simple "Hello, world!" program

Writing a "Hello, world!" program in Asin is a simple one-liner, and would much remind any Python-fluent user of the latter's print function.

```
ilimbag(“Hello, world”);
```

This also serves to exhibit ilimbag as Asin's printing pseudo-function, of the form:

```
ilimbag(arg{, arg});
```

Comments

In Asin, comments are segments of code preceded by a hash symbol (#).

```
# This is an Asin comment
```

Assignment statements

A simple assignment statement in Asin takes the form of:

```
foo = 4;                        # integer
floo = 3.1416;                  # float
bar = “I am a string”;          # string
troo = Totoo;                   # Boolean, true
fool = Huwad;                   # Boolean, false
barr = [“I’m a list”, 4, 4.5];  # list
```

You can see above three assignments that pertain to the five datatypes of Asin: integer, float, string, Boolean, and list, all taken from Python as Asin was built on the latter. It is important to note that strings enclosed with single quotes will give errors.

When assigning user input to a variable, use the shown function:

```
foo = pahingi(“Optional prompt string”);
```

When manipulating the value of an element in a list, i.e. assigning a value to a list at a specific index, one would need to use the `palitan` function.

```
palitan(<list>, <index>, <value>);
```

Compound assignment statements which involve arithmetic operations are also available. You may check the “Mathematical operators” section for more information.

```
fooPlus = 0;       fooPlus += 2;       # fooPlus is now 2
fooMinus = 5.3;    fooMinus -= 2.3;    # fooMinus is now 3.0
fooTimes = 9;      fooTimes *= 5;      # fooTimes is now 45
fooDiv = 10;       fooDiv /= 5;        # fooDiv is now 2
fooFdiv = 34.2;    fooFdiv //= 3;      # fooFdiv is now 11.0
fooExp = 3;        fooExp **= 3;       # fooExp is now 27
fooMod = 76;       fooMod %= 8;        # fooMod is now 4
```

Accessing list elements

To access a list element, simply write the expression `<list>[<index>]` .

Conditional statements

An if-else statement in Asin takes the following form,

```
kapag (<condition>) { # if clause
      # do a thing
} ngunit kapag (<condition>) { # else-if clause
      # do some other thing
} ngunit kapag (<condition>) { # another else-if clause
      # do yet another thing
} kundiman { # else clause
      # just do something
}
```

where `<condition>` pertains to an expression that evaluates to a True or False (`Totoo` or `Huwad`) value. The else-if and else clauses can be omitted as needed.

Loop structures

Asin offers two loop structures: the while-loop and the for-loop.

Program a while-loop by adhering to this syntax:

```
hanggat (<condition>) {
      # loop body
}
```

The loop body will be executed until `<condition>` does not hold true anymore.

Write a for-loop using the following form:

```
sa bawat <identifier> sa [<start> : <end>] {
      # loop body
}
```

From `<start>` to `<end>`, the body will be run. Looping stops once the value attached to `<identifier>` is greater than `<end>`. To be precise, the loop body will be executed `<end> - <start> + 1` times.

Unfortunately, Asin does not have support for looping through iterables, like how one could in Python (`for i in <list>:. . .`).

Within any loop, one may use the break statement of Asin, `lumisan`.

```
hanggat (<condition>) {
      # do something
      kapag (<condition>) {
            lumisan;
      }
}
sa bawat <identifier> sa [<start> : <end>] {
      # more instructions
      kapag (<condition>) {
            lumisan;
      }
}
```

Constants

There are two constants in Asin: Pi and Euler's number.

```
foo = asin_pi;
bar = asin_e;
```

Function calls

In Asin, functions may be either expressions (i.e, assignable) or statements. With Asin being a fledgling project, we opted not to offer an implementation of user-defined functions (which would have added a great deal of complexity to the project), but we do believe that the language still satisfies the listed requirements, having chosen the path of creating a programming language of an imperative paradigm.

Here is a table of Asin's built-in functions:

Expressions:

• `bilang(<value>)`	# converts `<value>` to an integer
• `lutang(<value>)`	# converts `<value>` to a float
• `titik(<value>)`	# converts `<value>` to a string
• `halaga(<number>)`	# returns the absolute value of `<number>`
• `ibilog(<number>)`	# returns the value that `<number>` rounds off to
• `putulin(<realnum>)`	# truncates the fractional portion of `<realnum>`
• `kisame(<realnum>)`	# processes `<realnum>` through the ceiling function
• `sahig(<realnum>)`	# processes `<realnum>` through the floor function
• `iangat(<base>,<exp>)`	# raises `<base>` to the power of `<exp>`
• `ibaba(<number>,<base>)`	# returns the logarithm of `<number>` to the base `<base>`
• `parisugat(<number>)`	# returns the square root of `<number>`
• `maximo(<list>)`	# returns the largest value in `<list>`
• `minimo(<list>)`	# returns the smallest value in `<list>`
• `haba(<iterable>)`	# returns the length of `<iterable>`
• `silipin(<list>)`	# returns the last element of `<list>`
• `baligtad(<list>)`	# returns reversed `<list>`
• `nakaayos(<list>)`	# returns sorted `<list>`

Statements:

• `idagdag(<list>, <value>);`	# appends `<value>` to list
• `tanggalan(<list>);`	# removes the last value of `<list>` and returns it
• `baligtarin(<list>);`	# reverses a list in place
• `isaayos(<list>,<descend>);`	# sorts `<list>` in place, in ascending order `<descend>` is an optional argument; list will be sorted in descending order if this is `Totoo`
• `asin();`	# prints "Asin", a pile of salt, and a bit of project information. Just for fun.

* Note: some functions were omitted, placed in more appropriate sections (with respect to context) of this manual.

Mathematical operators

Asin offers similar mathematical operations as Python.

```
<expression> + <expression>
<expression> - <expression>
<expression> * <expression>
<expression> / <expression>
<expression> // <expression>
<expression> ** <expression>
<expression> % <expression>
-<expression>
```

Note: Expressions can be enclosed with parentheses. Asin follows the PEMDAS rule for operator precedence.

Logical operators

Asin's mathematical operations aren't the only ones inspired by Python (though these could also have been inspired by C).

```
<expression> == <expression>
<expression> != <expression>
<expression> > <expression>
<expression> >= <expression>
<expression> < <expression>
<expression> <= <expression>
<expression> at <expression>     # logical and
<expression> o <expression>      # logical or
hindi <expression>               # logical complement (negation)
```

When using these operators, and the operands are not primitives, be sure to enclose the expressions in parentheses.

```
e.g.    (1 < 2)                  # this is okay
        ((1 < 2) at (2 < 3))     # this is okay, too
        (1 < 2 at 2 < 3)         # but this will give you the wrong output
```

File Input/Output

Asin offers I/O methods besides the basic `ilimbag` and `pahingi()`.

• `buksan(<file>, <access>)`	# opens a file with access mode `<access>`
• `isara(<file>)`	# closes `<file>`
• `basahin(<file>)`	# reads all of `<file>'s` contents
• `linya(<file>)`	# reads a line from `<file>`
• `isulat(<file>, <string>)`	`# writes <string> to <file>`

Exceptions

• `Maalat`	# parent exception class to all other exceptions
• `MaalatAtNawawalangFile`	# raised when opening a file that is nonexistent
• `MaalatNaOperasyon`	# raised when performing operations between incompatible data types
• `MaalatNaAritmetika`	# raised when dividing by zero
• `MaalatNaSimbolo`	# raised when accessing an inexistent variable/function
• `MaalatNaLeksim`	# raised in the event of a lexical error
• `MaalatNaPalaugnayan`	# raised in the event of a syntax error
• `MaalatNaIndeks`	# raised when a getting an element from a list using a non-integer, or using an integer that is out of bounds

---

Questions you might ask

Q: Is it easy to learn Asin?

A: If you are fluent in C, C++, Java, and/or Python, then it should be fairly straightforward to use the Asin language.

Q: What are the different source files for?

A: Each source file is a singular module which altogether make the interpreter possible.

- `asin.py` is the main file that parses Asin programs and allows the execution of their instructions
- `asinlex.py` contains the tokens/lexemes and their definitions
- `asinyacc.py` contains the grammar rules and the abstract syntax tree
- `asinnodes.py` defines the various classes for abstract syntax tree nodes
- `asintable.py` has the class for the hash table, and built-in functions (to be stored in a table object) preset for Asin
- `asinerrs.py` details the different exceptions that are raised for errors when executing Asin programs
- `asinhelper.py` contains two classes that were separated from other source files since they caused errors due to circular module importation

Q: I can't compile/interpret the source files -> I can't get it working!

A: Make sure that you have Python 3.5.2 or 3.4.3 installed, not an older nor a newer version (source is only tested for these versions). See to it that you have the ply library installed. Make sure that you have PyInstaller on your system with pip3 for use with Python 3.X.

If you do confirm have these, and you're running `makefile` but nothing good happens for any reason, then you may create your own makefile with the contents:

```
pyinstaller --onefile asin.py
```

Save this as `makefile` (`makefile.bat` on Windows), then on the terminal, enter the command "`chmod 755 makefile`" (Unix-like OSes only). This will convert `makefile` to an executable, that can be run with the command `./makefile` (double clicking `makefile.bat` file on Windows).

Q: I've successfully executed the makefile… what now?

A: You will find the interpreter as an executable named "`asin`" in the "dist" directory. Proceed to dist, and run the interpreter with `./asin <filename>` (Unix-like) or `asin.exe <filename>` (Windows).

Have fun using Asin! ☺