Meowlang

Language Guru: Carolyn Chen (cec2192)

Manager: Megan Frenkel (mmf2171)

System Architect: Lauren Pham (lyp2106)

Tester: Michelle Lin (ml4080)

Inspiration and Intention

Meowlang is an object-oriented esoteric programming language inspired by LOLCODE (Adam Lindsay)

- Utilizes Internet lolspeak: intentionally misspelled and grammatically incorrect natural language
- Intended to be humorous, absurd, yet functional
- Meowlang introduces powerful features such as Classes, Arrays and Built-in Functions
- We created a text-based RPG game

Example of LOLCAT meme and lolspeak

Meowlang in One Slide

1. **Import `KEWL_MODULE` module**
2. Set of keywords `HAI` and `KBYE` indicate scope
3. Main function declaration
4. Declare string (`YARN`) variable `message`
5. Assign value to `message`
6. `PSST` keyword indicates the beginning of a single-line comment
7. Call print function

```
1.GIMME KEWL_MODULE?

  HAI ITZ ME FUNC Main,
      ITZ ME YARN message.
      message IZ “Hello, World!”.
      PSST Print “Hello, World!”
      PURR Meow WIT message.
  KBYE
```

hello_world.meow

Meowlang in One Slide

1. Import `KEWL_MODULE` module
2. **Set of keywords `HAI` and `KBYE` indicate scope**
3. Main function declaration
4. Declare string (`YARN`) variable `message`
5. Assign value to `message`
6. `PSST` keyword indicates the beginning of a single-line comment
7. Call print function

```
  GIMME KEWL_MODULE?

2.HAI ITZ ME FUNC Main,
      ITZ ME YARN message.
      message IZ "Hello, World!".
      PSST Print "Hello, World!"
      PURR Meow WIT message.
2.KBYE
```

hello_world.meow

Meowlang in One Slide

1. Import `KEWL_MODULE` module
2. Set of keywords `HAI` and `KBYE` indicate scope
3. **Main function declaration**
4. Declare string (`YARN`) variable `message`
5. Assign value to `message`
6. `PSST` keyword indicates the beginning of a single-line comment
7. Call print function

```
  GIMME KEWL_MODULE?

3.HAI ITZ ME FUNC Main,
      ITZ ME YARN message.
      message IZ "Hello, World!".
      PSST Print "Hello, World!"
      PURR Meow WIT message.
  KBYE
```

hello_world.meow

Meowlang in One Slide

1. Import `KEWL_MODULE` module
2. Set of keywords `HAI` and `KBYE` indicate scope
3. Main function declaration
4. **Declare string (`YARN`) variable `message`**
5. Assign value to `message`
6. `PSST` keyword indicates the beginning of a single-line comment
7. Call print function

```
GIMME KEWL_MODULE?

HAI ITZ ME FUNC Main,
4.      ITZ ME YARN message.
        message IZ "Hello, World!".
        PSST Print "Hello, World!"
        PURR Meow WIT message.
KBYE
```

hello_world.meow

Meowlang in One Slide

1. Import `KEWL_MODULE` module
2. Set of keywords `HAI` and `KBYE` indicate scope
3. Main function declaration
4. Declare string (`YARN`) variable `message`
5. **Assign value to `message`**
6. `PSST` keyword indicates the beginning of a single-line comment
7. Call print function

```
GIMME KEWL_MODULE?

HAI ITZ ME FUNC Main,
    ITZ ME YARN message.
5.  message IZ "Hello, World!".
    PSST Print "Hello, World!"
    PURR Meow WIT message.
KBYE
```

hello_world.meow

Meowlang in One Slide

1. Import `KEWL_MODULE` module
2. Set of keywords `HAI` and `KBYE` indicate scope
3. Main function declaration
4. Declare string (`YARN`) variable `message`
5. Assign value to `message`
6. **`PSST` keyword indicates the beginning of a single-line comment**
7. Call print function

```
GIMME KEWL_MODULE?

HAI ITZ ME FUNC Main,
    ITZ ME YARN message.
    message IZ "Hello, World!".
6.  PSST Print "Hello, World!"
    PURR Meow WIT message.
KBYE
```

hello_world.meow

Meowlang in One Slide

1. Import `KEWL_MODULE` module
2. Set of keywords `HAI` and `KBYE` indicate scope
3. Main function declaration
4. Declare string (`YARN`) variable `message`
5. Assign value to `message`
6. `PSST` keyword indicates the beginning of a single-line comment
7. **Call print function**

```
GIMME KEWL_MODULE?

HAI ITZ ME FUNC Main,
    ITZ ME YARN message.
    message IZ "Hello, World!".
    PSST Print "Hello, World!"
7.  PURR Meow WIT message.
KBYE
```

hello_world.meow

The Syntax *cursed*

Meowlang is highly structured; keywords replace symbols with the goal of visually emulating natural language syntax

- **HAI** and **KBYE** are used to indicate scope, replaces curly braces
- **ITZ ME** - used in function, class and variable declarations
- **IZ** - assignment operator, replaces equals sign
- “.” - period indicates the end of a statement
- Case-sensitive, whitespace insensitive

```
GIMME KEWL_MODULE?

HAI ITZ ME FUNC Main,
    ITZ ME YARN message.
    message IZ “Hello, world!”.
    PURR Meow WIT message.
KBYE
```

Thoughtful use of whitespace and conventions maximizes readability!

Language Highlights

Built-in Types:	**Features**:	**Built-in Funcs/Keywords**:
• Strings (`YARN`) • Integers (`NUMBR`) • Floats (`NUMBAR`) • Booleans (`BOO`)	• Arrays (`BUCKET`) • Classes • Imports (`GIMME`) • Casting	• Print (`Meow`) • Scan (`Scan`) • Concatenation (`CAT`) • String Compare (`SAEM`)

Unique to Meowlang, unsupported by LOLCODE

Feature: Primitive Type Casting

- Casting between:
  - Int
  - Float
  - String
- Using:
  - Built in LLVM functions
  - C standard library functions
  - Custom C functions
- Float to Int: truncation
- String is malloc'd on the heap and must be freed

```
ITZ ME NUMBR int_var.
ITZ ME NUMBAR float_var IZ 2.84534534.

PURR Meow WIT float_var.    PSST Prints 2.84534534
int_var IZ NUMBR float_var.
PURR Meow WIT int_var.      PSST Prints 2
```

```
ITZ ME YARN string_var.
ITZ ME NUMBR int_var IZ 203423.

PURR Meow WIT int_var.      PSST Prints 203423
string_var IZ YARN int_var.
PURR Meow WIT string_var.   PSST Prints 203423
BLEEP string_var.
```

New Feature: String Concatenation (CAT)

- Concatenating string with String, Int, Float
- Codegen builds a function call to a custom string concatenation function written in C
- Autocasting for Int and Float to String wraps the operand in the A.Cast Binop type before recursively calling the expression builder
- Free allocated memory

```
ITZ ME NUMBAR flt IZ 2.0.
ITZ ME YARN str IZ " <- float to string.".
ITZ ME YARN flt_str_concat.

flt_str_concat IZ CAT flt AN str.
PURR Meow WIT flt_str_concat.
PSST Prints "2.0 <- float to string."

BLEEP flt_str_concat.
```

```
| SBinop(((A.String, _) as e1), A.Concat, ((_, _) as e2)) ->
   let lhs = expr builder e1 env
   and rhs = expr builder (A.String, SCast(A.String, e2)) env in
   L.build_call strcat_func [| lhs ; rhs |] "strcat_call" builder
```

New Feature: Arrays (BUCKET)

- Arrays live in **heap memory** to allow for variable-sized arrays
- Array **contents**:
  - Primitive types
  - Variable objects
  - User-defined objects
- Array **initialization**:
  - Initialize all, none, or some elements
  - Array size must be specified**
- Array **access** and **assignment**

```
MAEK animals NEW BUCKET OF YARN HOLDS 3,
    WIT "Cats"
    AN "Dogs".

animals[2] IZ "More dogs".

PURR Meow WIT animals[0]. PSST Prints “Cats”
PURR Meow WIT animals[1]. PSST Prints “Dogs”
PURR Meow WIT animals[2]. PSST Prints “More dogs”

BLEEP animals.
```

**It is actually possible to declare a new array with both size and contents unspecified without the `MAEK` keyword in this way: `ITZ ME BUCKET OF YARN strings`. In this case heap memory is not yet allocated and thus doing so effectively creates just a pointer to an array, without the actual memory for the array created. The use of this option should be limited to returning arrays from functions.

New Feature: Classes

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

- **User-defined** using function-like **HAI-KBYE** syntax, using keyword **CLASS**

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

- **User-defined** using function-like **HAI-KBYE** syntax, using keyword **CLASS**
- **Instance variables** support, default values are optional

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

- **User-defined** using function-like **HAI-KBYE** syntax, using keyword **CLASS**
- **Instance variables** support, default values are optional

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

- **User-defined** using function-like **HAI-KBYE** syntax, using keyword **CLASS**
- **Instance variables** support, default values are optional
- **Methods** also supported, with function-like syntax, direct access to instance variables; can call other methods on same object (or others)

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

- **User-defined** using function-like **HAI-KBYE** syntax, using keyword **CLASS**
- **Instance variables** support, default values are optional
- **Methods** also supported, with function-like syntax, direct access to instance variables; can call other methods on same object (or others)

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

Conversion of method → function happens during **AST → SAST** transformation in semant.ml.

① **Method lifting:** Make methods top level functions that take an object (struct pointer) as argument

② **Call site adjustments:** Adjust method calls to use new functions

Codegen defines new **struct** for each class.

```
HAI ITZ ME CLASS MOUSE,

    ITZ ME NUMBR cookies.
    ITZ ME NUMBR glasses_of_milk IZ 0.

    HAI ITZ ME FUNC Set_Num_Cookies
          WIT NUMBR cookies_given,
        cookies IZ cookies_given.
    KBYE

    HAI ITZ ME NUMBR FUNC Get_Num_Cookies,
        GIVE cookies.
    KBYE

    HAI ITZ ME FUNC Incr_Cookies,
        ITZ ME NUMBR existing_cookies IZ
          PURR Get_Num_Cookies IN HERE.
        cookies IZ SUM OF existing_cookies AN 1.
    KBYE

KBYE
```

New Feature: Classes

- Allocated on the heap, making use of keywords:
  - **MAEK** + **NEW** == “malloc”
  - **BLEEP** == “free”
- Constructor support (optional) using assignment-like expression with **WIT, AN** and **IZ**
- Access variables and methods using keyword **IN** with object identifier.

```
MAEK Jerry NEW MOUSE,
     WIT cookies IZ 5
     AN glasses_of_milk IZ 10.

cookies IN Jerry.
PURR Get_Num_Cookies IN Jerry.

BLEEP Jerry.
```

New Feature: Imports

- Import statements are always located at the beginning of a source file
- Syntax: **GIMME <MODULE_NAME>?**

module_name.meow

- Importing files containing function and class identifiers already in use will result in a compiler error

```
GIMME COLORS?
GIMME SHAPES?

HAI ITZ ME FUNC Main,
     PURR Get_Colors.
     PURR Get_Shapes.
KBYE
```

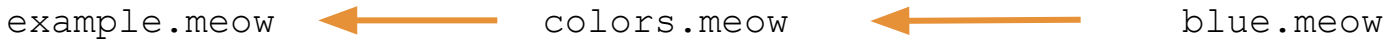
New Feature: Imports

```
GIMME COLORS?

HAI ITZ ME FUNC Main,
     PURR Get_Color.
KBYE
```

```
GIMME RED?
GIMME BLUE?
GIMME GREEN?
```

```
HAI ITZ ME FUNC Get_Color,
     ITZ ME YARN blue IZ “blue”.
     PURR Meow WIT blue.
KBYE
```



- A file being imported may also have imports
- `example.meow` imports `colors.meow` imports `blue.meow`
- No import hierarchy within a program

New Feature: Imports

1. AST is passed through separate `imports.ml` module
   a. Performs import-related semantic checks
   b. Generates AST for each imported files, appending to original AST
2. New AST is then passed to `semant.ml`

- Recursion allows for imports in imports
- ASTs are stored in a hashtable, with the module file path as the key
  - Addresses circular imports
  - Supports future project expansion

Testing

- Regression Test Suite contains:
  - Test_programs containing tests that are expected to pass and fail
  - Test_output containing the expected output of each test file
  - Shell scripts to automate testing, and allow for specified run-types
    - -a for printing out the AST
    - -s for semantic checking
    - -c for compiling to LLVM and printing the output
- Repetition and Separation
  - For every added functionality we would add many parsing/semantic tests, making sure it worked on its own before going on to the next functionality

Testing Process - Continued

Added new code:

- Getting “Hello World” to print
- Other functions besides printing
- Classes, objects
- Binop/Unop Operators
- Arrays
- For loops
- Conditionals
- etc.

Every time something new was added:

- Checked semantics by adding to sast.ml and codegen.ml
- Create tests expected to work
- Create tests expected to generate all possible specific errors

See if it works:

- Add expected output to test_output file
- Compare the expected output with actual output using shell scripts
- Look at pretty printing in pretty.ml for hints
- Make changes according to what we observe

Process loops until we are done!

Thank You

A big thank you to our professor and to our TAs (especially to Hans for guiding us through this project)!!

Program Demo