DECAF: Programming Language Proposal

Manager: Kylie Wu (kcw2141)
Language Guru: Hidy Han (yh2635)
System Architect: JiaYan Hu (jh3541)
Tester: Kimberly Tao (kmt2152)

Introduction

Introducing DECAF: "Easy enough to use without caffeine"

DECAF is a general-purpose, object-oriented language that compiles to LLVM. It is a language that will be intuitive to use for programmers who have previously used other high-level languages, such as Java, C, and Python. DECAF will extract a core subset of features from Java and Python and present these features in a concise semantic model.

More specifically, DECAF will support core object-oriented functionalities, such as inheritance and polymorphism. Additionally, DECAF will present flexible and robust built-in data structures, such as Python's list and tuple structures, which did not exist natively in Java.

Features & Syntax

Scope

DECAF's code blocks will be demarcated by curly braces `{}`. Code blocks, which can take the form of control flow or function definitions, can be nested within each other. Classes cannot be nested within one another.

Static Typing

All types must be defined when variables are declared, and the former may not be changed during the latter's lifetime, unless explicitly cast.

Language Features

- Inheritance and polymorphism
  - Method overriding
  - Interfaces and classes
- Method overloading

- Casting

  Requires explicit type casting for mixed type conversions.

- List comprehensions and functionalities such as `sort()`, `len()` and `append()`.
- Basic input and output

Operators

Type	Syntax	Description
Arithmetic	`+, -, *, /, ^,`%	Basic arithmetic operations. `x % 5`
Logical	`and, or, not`	Logical and, or, not. Equivalent to Java `&&, ||, !` respectively. `x < 3 and y > 7`
Boolean	`==, !=, <, >, <=, >=`	Basic boolean operations. `x != 5`
Assignment	`=`	Basic assignment statement. `x = 50;`
Return value	`->`	Declares a method's return value. `abs_val(int num) -> int`

Comments	`//, /* */`	Single line and multiline comments.
		
		`// Single line comment`
		`/*`
		`    This is a`
		`    multiline comment.`
		`*/`

Control Flow

Syntax	Example

`if, elseif, else`	Similar to Java `if`, `else if`, `else`.
	
	`if (x < 5) {`
	`    return x;`
	`}`

`while`	Similar to Java `while` loop.
	
	`while (x < 5) {`
	`    y = y * 2;`
	`    x = x + 1;`
	`}`

`for`	Similar to Java `for` loop.
	
	`for (i = 0; i < 10; i = i + 1) {`
	`    y = y * 2;`
	`}`

`break`	Similar to Java `break` and `continue`.
	
	`while (x < 5) {`
	`    if (y > 3) {`
	`        break;`
	`    }`
	`    elseif (z > 3) {`
	`        continue;`
	`    }`
	`}`

Built-In Data Types

Type	Description
`bool`	Boolean type. Can have values `true` or `false`.
`int`	Integer type.
`float`	Floating point type.
`char`	Represents a single character.
`string`	Represents a string of characters.
`Object`	An object with state and behavior.
`List/Tuple`	Similar to Python lists and tuples.

Keywords

Keyword	Description
`const`	Indicates that a field or variable cannot have its value changed.
`main`	Reserved keyword indicating the main function to run the program.
`void`	Indicates that the function does not return a value.
`return`	Returns the specified value.
`implements/extends`	Keyword indicating that a class implements an interface or extends another class.
`super`	Similar to Java's `super` keyword to access overridden methods in the superclass.
`interface/class`	Keyword marking a class or an interface.
`self`	Similar to `this` in Java referring to this object.

Example Programs

Hello World:

```
main() -> void {
            print("Hello world!\n");
}
```

Primitives:

```
// The following code block causes an error due to DECAF's choice to not support
// automatic type conversion.
int cost = 50;
float tip = cost * 0.15;

// The following code block is OK.
float cost = 50.0;
float tip = cost * 0.15;
```

Control Flow:

```
// Sample GCD program.
gcd(int x, int y) -> int {
    if (y == 0) {
        return x;
    }
    return gcd(y, x % y);
}

// Sample factorial program.
factorial(int n) -> int {
    int result = 1;
```

```
        for (int i = 2; i <= n; i = i + 1) {
            result = result * i;
        }
        return result;
    }
```

Lists & Tuples:

```
    // Sample list functionality: list comprehension.
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    doubled_evens = [n * 2 for n in numbers if n % 2 == 0];

    // Sample usage of tuples in returning multiple values.
    foo(float x, float y) -> (float, float) {
        return (x, y);
    }
```

Class Declaration & Instantiation:

```
    // Sample class demonstrating constructor overloading.
    class Cat {
            string breed;
            string color;
            int age;

            Cat() {
                    self.breed = "unknown";
                    self.color = "white";
                    self.age = 0;
            }

            Cat(string breed, string color, int age) {
                    self.breed = breed;
                    self.color = color;
                    self.age = age;
            }
    }

    // Instantiation
    Cat my_pet = Cat("Himalayan Cat", "white", 3);
    print(my_pet.color);
```

Inheritance:

```
class FictionalCat extends Cat {
        string reference;

        FictionalCat(string name, string color, int age, string reference) {
                super(name, color, age);
                self.reference = reference;
        }
}
```