The Kazm Programming Language

Aapeli Vuorinen (oav2108) - Systems Architect
Katie Kim (jk4534) - Manager
Molly McNutt (mrm2234) - Tester
Zhonglin Yang (zy2496) - Language Guru

October 2021

1 Overview

The Kazm programming language is a statically and strongly typed programming language which extends the C programming language with lightweight classes implemented as C-style structs that support methods and instances. General purpose programming functionality such as input/output will be provided through the inclusion of existing C libraries; however, we will not provide class inheritance or private methods.

Our motivation is to bridge the gap between C and C++. Kazm improves upon C as follows:

- Supports tuples (fixed-length arrays that allow the mixing of types)
- Gets rid of a C-style struct altogether – the Kazm language’s class is based on a C-style struct, but by default supports member functions and instances.

However, Kazm still supports C libraries, and does not deviate significantly from the syntax of C.

2 Language Details

2.1 Data types

The Kazm language provides four builtin primitive arithmetic data types: bool (8-bit), char (8-bit), int (64-bit), and double (64-bit). The language is also statically typed, thus the type of a variable is determined during compile time. Conversions between these primitive types are never done implicitly by Kazm, rather they must be done explicitly, for example through the builtin cast module.

String literals in code are entered using double quotes and are null-terminated of type char[]. Character literals (at most one character) are entered with single quotes and get type char.

2.2 Operators

The Kazm programming language has a wide range of operators to perform basic operations. The operators are grouped as following:

- Relational Operators: ==, !=, >, <, >=, <=
- Logical Operators: &&, ||, unary !
2.3 Data structures

The Kazm language provides the built-in data structures **array** and **tuple**. The array implementation will be same as in C. Elements in an array data structure need to be the same data type. By contrast, elements in a tuple structures can be different data types but their data types have to be **explicitly defined** when the tuple is declared.

```kazm
tuple(int, bool) a = (1, true);  
tuple(int, double, bool) b = (2, 3.5, false);  
int[] numbers[2] = [a[0], b[0]];  
bool[] booleans[2] = [a[1], b[2]]; 
```

2.4 Control flow statements

The Kazm language has basic control flow statements as following: **if..elif..else statement**, **for** loop, and the **while** and **do..while** loops; including the **break** statement.

2.5 Function declaration and function calls

Declaring functions and calling defined functions are supported in the Kazm programming language. The syntax of function definition and function call is C-like. Each file to be executed must also have a **main** function with signature `void main()`. When a Kazm program is invoked, this **main** function will be called.

```kazm
int sum(int[] list, int list_size) {
    int result = 0;
    for (i = 0; i < list_size; i++) {
        result += list[i];
    }
    return result;
}

void main() {
    int[] list[5] = [1, 6, 12, 17, 25];
    int sum = sum(list, 5);
    return 0;
}
```

2.6 Classes

The Kazm language has classes which are like structs in C language but support member functions to call directly on class objects. The keyword **me** is used for self-reference within an instance. Member variables are accessed using the **.** (dot) operator.

2.6.1 String Class

```kazm
//@ String.kazm

class String {
    char[] str;
    int len;

    // Constructor
    public String(char[] str) {
        me.str = str;
        if (str == null) {
```
me.len = 0;
} else {
    char c = 'a';
    int count = 0;
    char[] str = [a', 'b', 'c', '\0'];
    while (c != '\0') {
        count++;
        c = str[count];
    }
    me.len = count;
}
}

/* member function */
String concat(String you){
    char[] str2[me.len + you.len + 1];
    //"hello " world" 5, 6 ==> 12
    for (int i = 0; i<me.len; i++){
        str2[i] = str[i];
    }
    int start = me.len;
    for (int i = 0; i< you.len; i++){
        str2[start++] = you[i];
    }
    //[h, e, l, l, o, , w, o, r, l, d, '\0']
    me.len = me.len + you.len +1;
    str2[me.len-1] = '\0';
    str = str2;
}
}

2.7 Builtins
The language defines a few convenient builtins (probably not implemented in Kazm), namely the following modules and functions:

1. io: input/output functionality
   - void print({type0} arg0, {type1} arg1, {type2} arg2, ...): prints each argument one after the other
   - void println(...) like above, but end line terminated
   - {type} read.{type}() reads a value of type {type} from the prompt
   - exit(char[] msg) exits with an error message msg.

2. cast: casts types from one type to another.
   - {type2} {type1} to.{type2}({type1} arg) (e.g. double int_to_double(int arg)) casts type
     type1 to type type2.

3. math: math functions like floor, ceil, etc.
2.8 Modules and import system

Programs can import symbols from other files using a Python-style import syntax. There are two ways to import symbols:

1. import module imports all symbols from module, accessible as module.symbol
2. from module import symbol similar to above, but now accessible as symbol

A module is a source code file possibly in some subfolder, in a file ending in .kazm similar to Python (but not as messy).

2.9 External C library support

We also would like the Kazm programming language to be able to import external C libraries so that some additional functionality can be achieved in the Kazm language.

2.10 Comments

Kazm implements C-style comments:

/* multiple-line comment */
// single-line comment

2.11 Keyword List

int, double, bool, char, tuple, if, else, for, while, me, null, break, do, from, import

3 Example programs

We now list a variety of example programs to demonstrate language syntax and functionality.

3.1 random library

// random.kazm
/*
Package ‘random’ implements basic random functions
*/
from math import floor
import cast

/*
Returns a random double between 0 and 1
*/
double rand_uniform() {
    // From some external library?? Mercenne Twister?
    // TODO
    return 0.5;
}

/*
Returns a random integer in with 0 <= n < upper_bound
*/
int rand_int(int upper_bound) {
    return cast.double_to_int(floor(random_uniform() * upper_bound));
}
Returns 1 with probability p, otherwise 0

```c
int rand_bernoulli(p) {
    if (random_uniform() < p) {
        return 1;
    } else {
        return 0;
    }
}
```

```c
int rand_binomial(n, p) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        count += rand_bernoulli(p);
    }
    return count;
}
```

### 3.2 Number guessing game

```c
// guess.kazm
from io import read_int, print, println;
from random import rand_int;

void main() {
    int value = rand_int(0, 100);
    do {
        print("Please guess my number! It’s between 0 and 100!");
        int guess = read_int();
        if (guess == value) {
            println("That’s right!!");
        } elif (guess < value) {
            println("That’s a bit too low");
        } else {
            println("That’s too big!");
        }
    } while (guess != value);
    println("Congrats, you won :)");
}
```

### 3.3 COVID SIR model simulator

```c
// covid.kazm
/*
This example is a basic COVID-19 simulator using a simplified SIR model.
```
The dynamics are as follows:

1. There are $n$ students and $m$ classrooms
2. For each day, $t=1,\ldots,t_{\text{max}}$
   a. Each student randomly chooses a classroom to go to
   b. For each infected student in the classroom they go to, they have a probability of $p\%$ of getting infected
   c. They enter an incubating state
3. If a student is incubating on a given day, they have a probability of $q\%$ of becoming infectious the next day
4. If a student is infectious on a given day, they have a probability of $w\%$ of becoming removed (no longer susceptible) the next day

```python
import cast
from io import read_int, print, println, exit;
from random import rand_int, rand_binomial, rand_uniform;
println("Welcome to the COVID simulator");

int read_int_between(int min, int max) {
    int input = read_int();
    if (input > max) {
        exit("Number too large");
    } elif (input < min) {
        exit("Number too small");
    }
    return input;
}

int main() {
    int n = read_int_between(1, 1000);
    print("Please enter the number of classrooms:");
    int m = read_int_between(1, n);
    print("Please enter number of infected people at time 0:");
    int infectious0 = read_int_between(1, n-1);
    print("Please enter number of time steps:");
    int t_max = read_int_between(1, 10000);
    print("Please enter number of classes:");
    int t_max = read_int_between(1, 10000);
    print("Please enter infection probability as a percentage (e.g. 50 for 50%):");
    int p100 = read_int_between(0, 100);
    print("Please enter probability of becoming infectious after being incubating as a percentage (e.g. 50 for 50%):");
    int q100 = read_int_between(0, 100);
    print("Please enter probability of becoming removed after being infectious as a");
```
percentage (e.g. 50 for 50%);"
int w100 = read_int_between(0, 100);

double p = cast.int_to_double(p100) / 100.0;
double q = cast.int_to_double(q100) / 100.0;
double w = cast.int_to_double(w100) / 100.0;

int[] infectious[];
int[] susceptible[];
int[] incubating[];
int[] recovered[];

infectious[0] = infectious0;
susceptible[0] = n;
incubating[0] = 0;
recovered[0] = 0;

for (int t = 0; t < t_max-1; t++) {
    println("Starting simulation step ", t+1);
    println("There are ", susceptible[t], " susceptible students");
    println("There are ", incubating[t], " incubating students");
    println("There are ", infectious[t], " infectious students");
    println("There are ", recovered[t], " recovered students");

    infectious[t+1] = infectious[t];
susceptible[t+1] = susceptible[t];
incubating[t+1] = incubating[t];
recovered[t+1] = recovered[t];

    int[] susceptible_students_in_rooms[m];
    int[] infectious_students_in_rooms[m];

    // allocate susceptible students to rooms
    for (int i = 0; i < susceptible[t]; i++) {
        // pick a random room for each student
        susceptible_students_in_rooms[rand_int(m)]++;
    }

    // allocate infectious students to rooms
    for (int i = 0; i < infectious[t]; i++) {
        infectious_students_in_rooms[rand_int(m)]++;
    }

    // for each room, do a sim
    for (int j = 0; j < m; j++) {
        for (int inf_i = 0; inf_i < infectious_students_in_rooms[j]; inf_i++) {
            for (int sus_i = 0; sus_i < susceptible_students_in_rooms[j]; sus_i++) {
                // get infected w.p. p
                if (rand_uniform() < p) {
                    incubating[t+1]++;
                    susceptible[t+1]--;
                    break;
                }
            }
        }
    }
}
// handle students going inc -> inf
for (int i = 0; i < [t]; i++) {
    if (rand_uniform() < w) {
        incubating[t+1]--;
        infectious[t+1]++;
    }
}

// handle students going inf -> rec
for (int i = 0; i < incubating[t]; i++) {
    if (rand_uniform() < w) {
        infectious[t+1]--;
        recovered[t+1]++;
    }
}

println("Finished simulation step ", t+1);
}

println("Simulation complete!");
println("There are ", susceptible[t_max-1], " susceptible students");
println("There are ", incubating[t_max-1], " incubating students");
println("There are ", infectious[t_max-1], " infectious students");
println("There are ", recovered[t_max-1], " recovered students");

// TODO: plotting?

return 0;