FFBB Programming Language Proposal

Bowen Chen, Joseph Yang, Jianan Yao, Xiaosheng Chen

{bc2916, zy2431, jy3022, xc2561}@columbia.edu

February 2, 2021

1 Overview

The FFBB programming language is an imperative language mainly based on the C programming language, with some other features inspired by Java.

It is a general-purpose programming language and even users with non-technical background will be able to study FFBB easily. FFBB will finish syntax-checking during compile time so that programmers won't waste too much time on syntax problem.

The general syntax and language features would be similar to those of the C programming language, with some other operators and features from Java (e.g. type declaration, comment, `void` keyword). Also, FFBB programming language accepts some functional programming features like higher-order functions. We hope that our language could combine the advantages of C and the flexibility of Python to some extent, with certain acceptable trade-off.

Our goals are:

- C-style language design with safe explicit type and easy compilation.
- Support of recursion and higher-order functions.
- Built-in data structures of list (array), dictionary and set like in Python. Implemented using hashtable to achieve high efficiency.

2 Language Details

2.1 Data Types and Operations

FFBB 's primitive data types are `integer`, `float`, `character`, and `boolean`. `String` is a built-in a class wrapping immutable arrays of characters. The language is in general not explicitly typed, and a wide range of automatic conversions will be performed. Control-flow consists of if-else statements, for loops, and while loops. FFBB will also support operators $=, ==, !=, +, -, *, /, \%, ++, --, +=, -=, <, >, =<$, and $>=$ for integers and floats, and $=, ==, !=, not, and$, and *or* for booleans. Other types may be added as needed. We will also have array types, but their exact syntax is yet to be determined. They will likely use an int[], float[], and char[] syntax. The language also supports custom-defined types, which can also function as keywords in typed functions.

Data Type	Description	Examples
int	An integral type, 4 bytes	3+4; x+=1; 5!=12
float	An float type, 4 bytes	3.0+4.0; 8.0+=1.0; 5.0!=12.0
boolean	store either true or false	x=true; true == true; !true
Character	A character type, 1 bytes	x='c'

2.2 Keywords

The following are reserved keywords: **while, for, in, if, else, rec, main, return, true, false, int, and, or, not, float, char, bool, string, void, def**

2.3 Control Flow

2.3.1 If...else...

```
if(condition) {} else {}
```

2.3.2 While Loops

```
while(condition) {}
```

2.3.3 For in Loops

```
for (i in range(2, n)) {}
```

Support iterating over List, Dictionary and Set.

2.4 Functions

2.4.1 Main

```
def void main(){}
```

Our language allows the users to include a `main` function in a file, providing an interface to execute the file.

2.4.2 Recursion

```
// recursive function to find gcd of two number
def rec int gcd(int a, int b) {
    if (b!=0) {
        return gcd(b, a%b); // general case
    } else {
        return a; // base case
    }
}
```

The system-reserved keyword `rec` is a required identifier for recursive functions.

2.4.3 High-order Function

Basically, our language could take parameter with function type in function definitions, meaning that our language accepts higher-order functions.

2.5 Comments

2.5.1 Single Line Comment

```
// single line comment goes here...
```

2.5.2 Multi-Line Comment

```
'''
multiple line comment
goes here...
'''
```

2.6 Built-in IO

2.6.1 Output

We would have a general purpose print function (i.e. works on all types).

2.7 Built-in Data Structures

The features of built-in data structures are described as follows:

2.7.1 List

```
// Init with n elements using ListCreate
String[] listA = ListCreate<string>(n);
// Init with [] format
int[] listB = [1, 2, 3];
// Update value at index
listA[0] = "0";
// Get length of a list
int len = ListLength(listA);
// Add element to the index position of the list
ListAdd(listA, "1", idx);
// Remove the first element of the list
ListRemove(listA, "1", idx);
// Create a list with [m,...,n-1], assuming m < n
int[] range = range(m, n);
```

2.7.2 Dictionary

```
// create a dictionary given types of key and value
Dict<int, string> dictA = DictCreate<int,string>();
```

```
// search key in dictionary, returns if key is found
bool found = DictFind(dictA, 3);
// insert (key, value) to dictionary, overwrite existing value
DictInsert(dictA, 3, "three");
// retrieve value by key in dictionary, throws error if key not exists
string s =  DictAt(dictA, 3);
// remove (key, value) from dictionary, throws error if key not exists
DictDelete(dictA, 3);
// get size of a dictionary
int len = DictLength(dictA);
```

2.7.3 Set

```
// Create a set given the type of elements
Set<int> setA = SetCreate<int>();
// Insert a new element to the set
SetInsert(setA, 3);
// Check whether an element is in the set
SetFind(setA, 3);
// Remove an element from the set
SetDelete(setA, 3);
// get size of a set
int len = SetLength(setA);
```

3 Examples

Here are some example codes to demonstrate our language.

```
def void swap(int[] A, int i, int j) {
    int t = A[i]; A[i] = A[j]; A[j] = t;
}

def int partition(int[] A, int p, int r) {
  int x = A[r];
  int i = p - 1;
  for (j in range(p, r + 1)) {
      if (A[j] <= x) {
```

```
            i++;
            swap(A, i, j);
        }
    }
    swap(A, i + 1, r);
    return i + 1;
}

// Recursive function to sort list A using quick-sort
def rec void quicksort(int[] A, int p, int r) {
    if (p < r) {
        int q = partition(A, p, r);
        quicksort(A, p, q-1);
        quicksort(A, q+1, r);
    }
}

def void main () {
    // Fibonacci number: Compute Nth value
    int n = 10;
    int[] f = ListCreate<int>(n);
    f[0] = 0;
    f[1] = 1;
    for (i in range(2, n)) {
        f[i] = f[i - 1] + f[i - 2];
    }
    print(f[n-1]);

    // Using quicksort
    int[] A = [4, 2, 7, 3, 1, 9, 6, 10, 5, 8];
    quicksort(A, 0, ListLength(A) - 1);
    for (a in A) {
        print(a);
    }
}
```