# Harmonica Language Proposal

language for parallel computing

**Guihao Liang(gl2520), Jincheng Li(jl4569), Xue Wang(xw2409), Zizhang Hu(CVN, zh2208)**

**Introduction:**

Motivation: With the dominance of multi-processor architectures and distributed applications, languages with built-in concurrency support are becoming increasingly popular. Harmonica is a language that borrows from features of Go and python to provide easy-to-use primitives for programming parallel programs.

Goal**:** compile to LLVM three address code.

Grammar: similar to C with some different keywords

**Features:**

- Support of concurrency (threading, channels).
- First-class functions
- Compound types (struct)
- Standard library for networking, scientific computing, and seamless/grubhub.

**Concurrency:**

We support concurrency by the `parallel` keyword which takes lists, and yields async channels.

**Potential Usage:**

As an imperative language with functional programming support and built-in concurrency support, Harmonica is perfect for server programming, data processing and scientific computation.

**primitive types:**

| **name** | **description** | **example** |
|---|---|---|
| int | integer | 1,2,3 |
| float64 | Double precision 64-bit floating point number | 0.0 |
| bool | Boolean value | True, False |
| string | String literals | “proposal” |
| list | Linked list | |

**Operator:**

| name | description | Applicable data types |
|---|---|---|
| + | String Concatenation | string |
| +,-,*,/ | Mathematical operators | int,float |
| % | Mathematical operators | int |
| and, or, not | Logical operators | bool |
| <,<=,>=,>,== | comparison operators | int,float,string |
| = | assignment | int,float,string,bool |

**function definition:**

```
type f(type arg1, type arg2) { body }
```

**Keyword:**
**Basic keyword:**

| name | description | syntax |
|---|---|---|
| const | Constant variable | const var a; |
| import | Import package | import Random |
| from | from | from Random import Gamma |
| . | Member Access | Gamma.mgf() |
| return | Return from function. | |
| Parallel | Parallelly execute. | |

**Compound type:**

| name | description | syntax |
|---|---|---|
| List | Collection of mutable ptr | [1, 2] |
| tuple | Collection of Immutable ptr | (1, 2) |

**control flow:**

| **name** | **description** | **syntax** |
|---|---|---|
| if...else  if...elif | Conditional branch | If(){}; else{};  if(){}; elif(){}; |
| for | Loop | for(){} |
| while | Loop | while(){} |

Comment single line: #
Block: “”” … “””

built-in functions:

| **Function** | **Input Data Type** | **Output Data Type** | **Description** |
|---|---|---|---|
| filter | list | list | filter(*func, []) |
| map | list | list | map(*func, []) |
| reduce | list | list | reduce(*func, []) |
| print | printable* | string | print(something) |

*printable: any primary type, any list of printables, any composition of printables

**Sample code:**

1.
```
int GCD (int a, int b) {
  if (a == b) {
    return a;
  } else {
    if (a > b) {
      GCD(a - b,b);
    } else {
      GCD(b - a,a);
    }
  }
}

parallel(GCD, [(5,25),(60,72),(1,9),(43,47),(88,48)], 5);
```

2.

```
from Gamma import gamma, mgf
from ElementaryMathematics import sqrt, mean

float[] random_nums = gamma(10, k=1.0, theta=2.0);
float[] second_moments = map(mgf, args=2, random_nums);
float[] averages = map(mgf, args=1, random_nums);

float E_sharpe = mean(averages/map(sqrt, second_moments));
```

3.

```
print(“Hello, world.”);
```

4.

```
parallel([f1,f2,f3,f4,f5],[a,b,c,d,e],5);
parallel(f,[a,b,c,d,e],5);
parallel(f,[a,b,c,d,e]);(default value: min(len(list),#available
threads))

chan c = parallel(f, [a, b, c], 3);

for (;;) {
      select {
            <-chan1:
            <-chan2:
      }
}

while ((int i = c.any())>= 0) List lst = c.fetch(i);
for (; c.ready(); ) List lst = c.get().toList();
Print(lst)
```