HARMONICA

- LANGUAGE FOR PARALLEL COMPUTING

GUIHAO LIANG(GL2520),

JINCHENG LI(JL4569),

XUE WANG(XW2409),

ZIZHANG HU(CVN, ZH2208)

THE LANGUAGE

- **Motivation:**
  - Dominance of multi-processor architectures
  - Rise of distributed applications and computing on large data sets
  - Languages with built-in concurrency support are becoming increasingly popular.

THE LANGUAGE

- **Goal:**
  - Provide easy-to-use primitives for programming parallel programs
  - Handle large matrix operations / data frame manipulation / signal processing computations efficiently

THE LANGUAGE

- **Features:**
  - Concurrency support
  - First-class functions
  - Compound types (struct)
  - Standard math library for scientific computing
  - Container libraries (vector, binary search tree)

COMPILER STRUCTURE

- Scanner, Parser: Harmonica => AST
- Semant, Codegen: AST => LLVM module
- Clang: C => LLVM module
- LLVM Linker

RESPONSIBILITIES

Guihao Liang	parser, C bindings, pthread library, preprocessor
Jincheng Li	parser, semantic checking, first-class functions, vector/BST libraries
Xue Wang	testing, documentation, language design, parser
Zizhang Hu	parser, math library, semantic checking, code generation

FIRST-CLASS FUNCTIONS

- Functions are no different from variables
  - Can be passed as arguments

    ```
    void map(<int int> f, list[int] arr, int length);
    map(plus1, [1,2,3], 3);
    ```

  - Can be declared as variables and assigned different values

    ```
    bool bar(int x) { x == 3; }
    <int bool> foo = bar;
    ```

LAMBDA EXPRESSIONS

- In-line function definitions
  - Syntax: lambda => argument list => return type => expression

    ```
    <int int> plus1  = lambda (int x) int ( x + 1 );
    ```

  - Returns one single expression
  - No closure support right now. OCaml-LLVM seems to lack support for this.

PARALLEL AND MUTEX

- Lack of support on Ocaml-LLVM thread bindings, and LLVM system thread documents.
- Use Clang as another level of indirection: convert C program to LLVM.
- Using POSIX threads to implement parallel and mutex.
- Mutex is sort of same as POSIX's. It's used for concurrency control.
- Parallel takes a function object and a list of arguments, and then spawns threads.

```
# create 4 parallel thread to print out square.
void foo(int a)  { printi(a * a);
parallel(foo, [1,2,3,4], 4);
```

PARALLEL AND MUTEX

- `clang -c -pthread -emit-llvm bindings.c`
- Convert bingings.c to bingings.bc and embed it into LLVM

```
let llmem = L.MemoryBuffer.of_file “bindings.bc” in
let llm = Llvm_bitreader.parse_bitcode context llmem in
ignore (Llvm_linker.link_modules the_module llm
Llvm_linker.Mode.PreserveSource);
```

- Source in bindings.c

TEMPLATE AND PREPROCESSOR

- Preprocessor will do context macro replacement before compilation.
- `alias` directives will guide the preprocessor to process template program.
- `python preprocess.py $@ | ./harmonica.native`

```
alias T int

struct vector_T {
 list[T] elements;
 int length;
 int memsize;
};
```

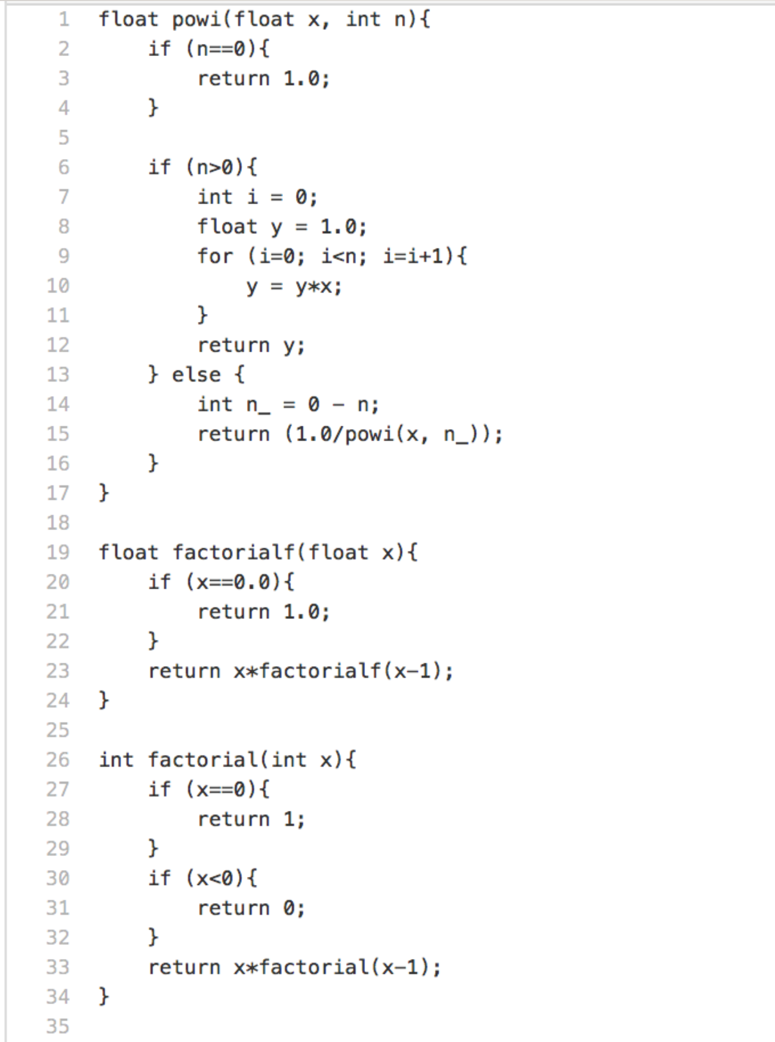
```
struct vector_int {
 list[int]
elements;
 int length;
 int memsize;
};
```

TESTING

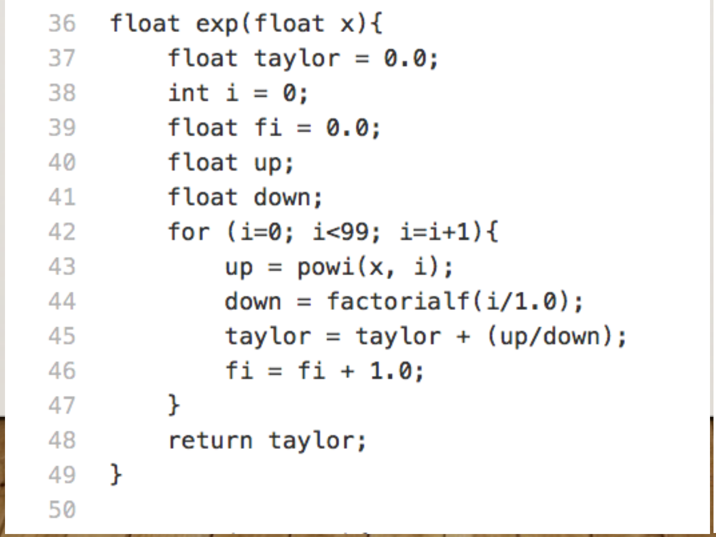
- Test-*.ha cases: expected-to-work
- Fail-*.ha cases: expected-to-fail

- Run ./testall.sh:
  - Takes all files starting with test- or fail- and ending with .ha.
  - Make executable, run them and redirect stdout to corresponding .out files
  - Check diff between these .out files to ref .out files
  - If no diff, delete .diff files, returns OK, else keep diff files return FAILED
  - All test information goes to testall.log

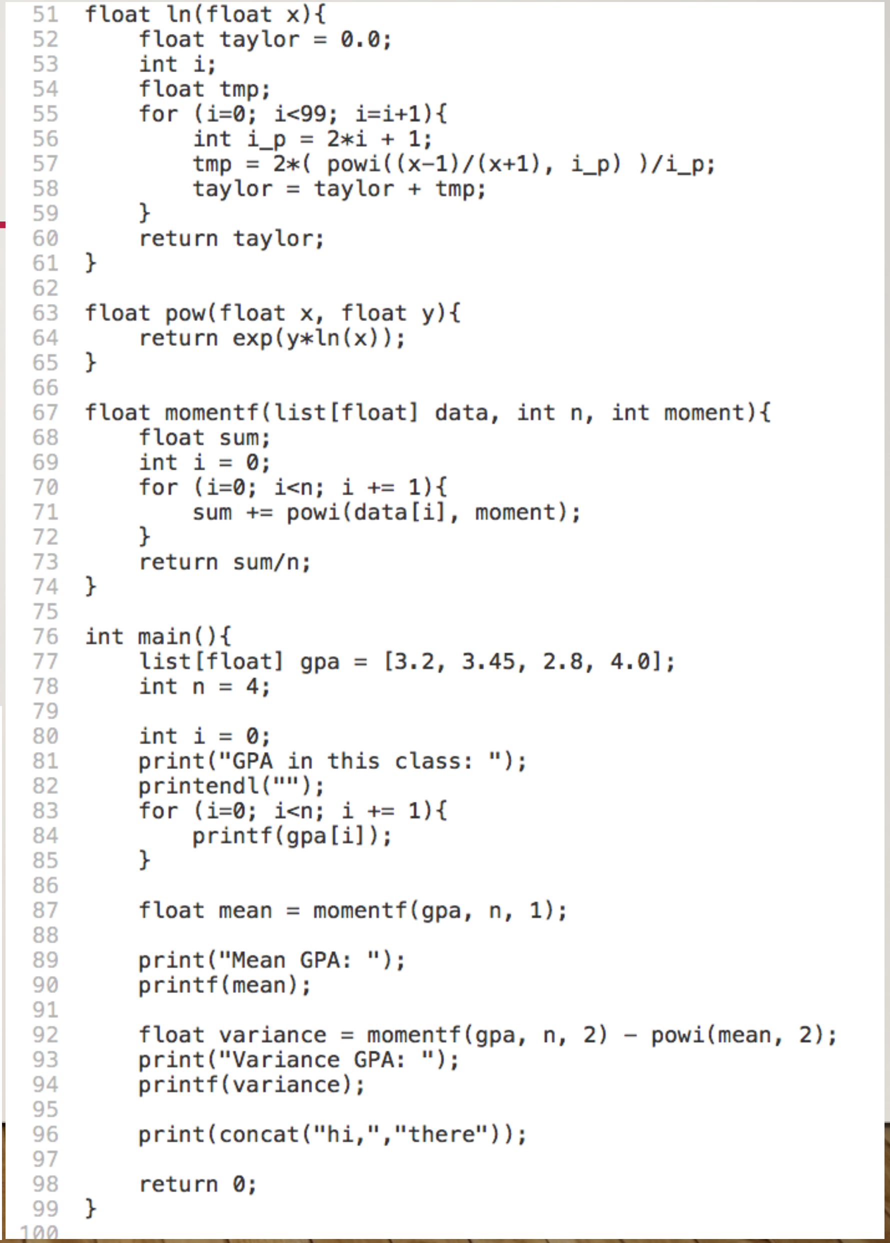
LIRBRARIES (MATH)



```
float powi(float x, int n){
    if (n==0){
        return 1.0;
    }

    if (n>0){
        int i = 0;
        float y = 1.0;
        for (i=0; i<n; i=i+1){
            y = y*x;
        }
        return y;
    } else {
        int n_ = 0 - n;
        return (1.0/powi(x, n_));
    }
}

float factorialf(float x){
    if (x==0.0){
        return 1.0;
    }
    return x*factorialf(x-1);
}

int factorial(int x){
    if (x==0){
        return 1;
    }
    if (x<0){
        return 0;
    }
    return x*factorial(x-1);
}

```



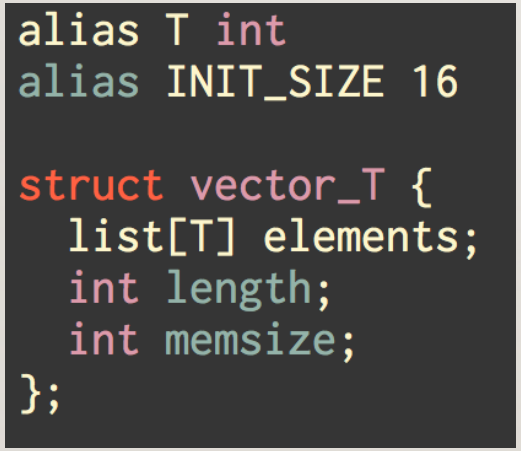
```
float exp(float x){
    float taylor = 0.0;
    int i = 0;
    float fi = 0.0;
    float up;
    float down;
    for (i=0; i<99; i=i+1){
        up = powi(x, i);
        down = factorialf(i/1.0);
        taylor = taylor + (up/down);
        fi = fi + 1.0;
    }
    return taylor;
}

```



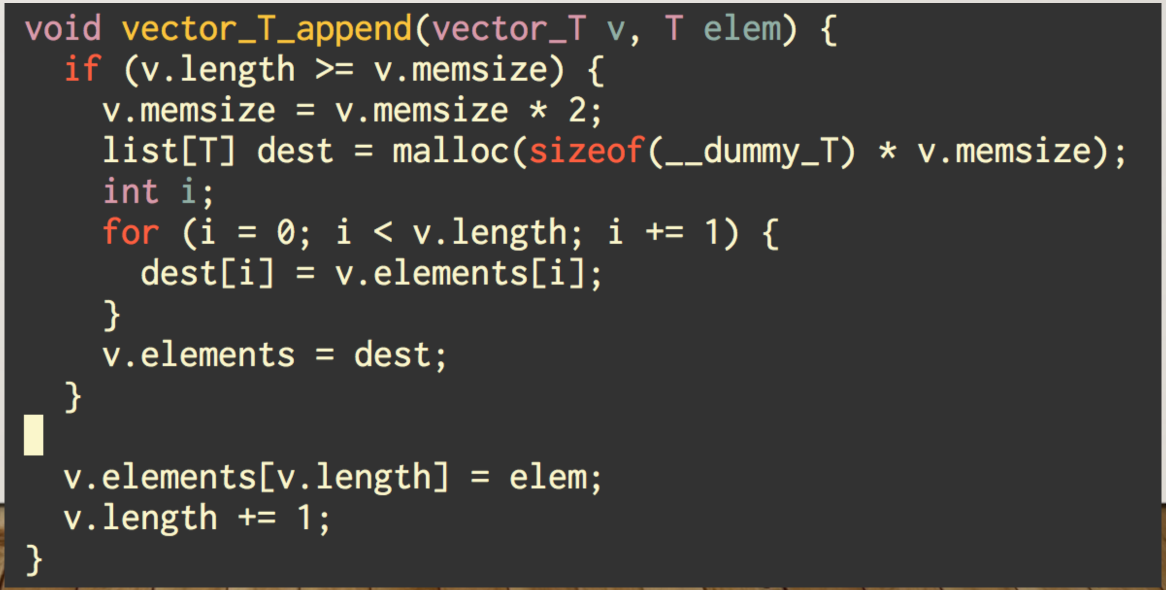
```
float ln(float x){
    float taylor = 0.0;
    int i;
    float tmp;
    for (i=0; i<99; i=i+1){
        int i_p = 2*i + 1;
        tmp = 2*( powi((x-1)/(x+1), i_p) )/i_p;
        taylor = taylor + tmp;
    }
    return taylor;
}

float pow(float x, float y){
    return exp(y*ln(x));
}

float momentf(list[float] data, int n, int moment){
    float sum;
    int i = 0;
    for (i=0; i<n; i += 1){
        sum += powi(data[i], moment);
    }
    return sum/n;
}

int main(){
    list[float] gpa = [3.2, 3.45, 2.8, 4.0];
    int n = 4;

    int i = 0;
    print("GPA in this class: ");
    printendl("");
    for (i=0; i<n; i += 1){
        printf(gpa[i]);
    }

    float mean = momentf(gpa, n, 1);

    print("Mean GPA: ");
    printf(mean);

    float variance = momentf(gpa, n, 2) - powi(mean, 2);
    print("Variance GPA: ");
    printf(variance);

    print(concat("hi,","there"));

    return 0;
}
```

LIBRARIES (VECTOR)

- Simple dynamic array container
- Uses preprocessor macros to accommodate different types
- Similar to how you would implement vectors in C



```
alias T int
alias INIT_SIZE 16

struct vector_T {
  list[T] elements;
  int length;
  int memsize;
};
```



```
void vector_T_append(vector_T v, T elem) {
  if (v.length >= v.memsize) {
    v.memsize = v.memsize * 2;
    list[T] dest = malloc(sizeof(__dummy_T) * v.memsize);
    int i;
    for (i = 0; i < v.length; i += 1) {
      dest[i] = v.elements[i];
    }
    v.elements = dest;
  }

  v.elements[v.length] = elem;
  v.length += 1;
}
```

LIBRARIES (BINARY SEARCH TREE)

- Basic BST with fine-grained locking

```
struct Node {
  int value;
  Node lchild;
  Node rchild;
  mutex lock;
};
```

- Safely handles operations from multiple threads

DEMO

FUTURE

- Channel
- Function Closure
- Modules and Namespaces
- Better Standard Libraries