MUA 程序说明

尹嘉权 3120000419

基本要求如下：

MakeUp Programming Language

基本数据类型value

数字number，单词word，列表list，布尔bool

- 数字：以[0~9]或'-'开头，不区分整数，浮点数
- 单词：以双引号"开头，不含空格，采用Unicode编码
- 列表：以方括号[]包含，其中的元素以空格分隔；元素可是任意类型；元素类型可不一致

基本操作

基本形式：操作名 参数

操作名是一个不含空格的词，与参数间以空格分隔。参数可以有多个，多个参数间以空格分隔。每个操作所需的参数数量是确定的，所以不需要括号或语句结束符号

基本操作

- `//`：注释
- `make <word> <value>`： 将value绑定到word上。基本操作的单词不能用做这里的word。绑定后的word称作名字，位于命名空间
- `thing <word>`：返回word所绑定的值
- `: <word>`：与thing相同
- `erase <word>`：清除word所绑定的值
- `isname <word>`：返回word是否是一个名字
- `print <value>`：输出value
- `read`：返回一个从标准输入读取的数字或单词
- `readlinst`：返回一个从标准输入读取的一行，构成一个列表，行中每个以空格分隔的部分是list的一个元素
- 运算符operator
  - `add, sub, mul, div, mod`：`<operator> <number> <number>`
  - `eq, gt, lt`：`<operator> <number|word> <number|word>`
  - `and, or`：`<operator> <bool> <bool>`
  - `not`：`not <bool>`
- `random <number>`：返回[0,number>的一个随机数
- `sqrt <number>`：返回number的平方根
- `isnumber <value>`：返回value是否是数字
- `isword <value>`：返回value是否是单词
- `islist <value>`：返回value是否是列表
- `isbool <value>`：返回value是否是布尔量
- `isempty <word|list>`: 返回word或list是否是空
- `test <value>`：测试value是真是假
- `iftrue <list>`：如果之前最后一次test是真，则执行list
- `iffalse <list>`：如果之前最后一次test是假，则执行list

- `word <word> <word|number|bool>`：将两个word合并为一个word，第二个值可以是word、number或bool
- `list <list1> <list2>`：将list1和list2合并成一个列表，两个列表的元素并列，list1的在list2的前面
- `join <list> <value>`：将value作为list的最后一个元素加入到list中（如果value是列表，则整个value成为列表的最后一个元素）
- `first <word|list>`：返回word的第一个字符，或list的第一个元素
- `last <word|list>`：返回word的最后一个字符，list的最后一个元素
- `butfirst <word|list>`：返回除第一个元素外剩下的列表，或除第一个字符外剩下的单词
- `butlast <word|list>`：返回除最后一个元素外剩下的列表，或除最后一个字符外剩下的单词
- `item <number> <word|list>`：返回word或列表中的第number项字符或元素
- `repeat <number> <list>`：运行list中的代码number次
- `stop`：停止当前代码的执行。当前代码可能是run、repeat、if或函数中的代码
- `wait <number>`：等待number个ms
- `save <word>`：保存当前命名空间在word文件中
- `load <word>`：从word文件中装载内容，加入当前命名空间
- `erall`：清除当前命名空间的全部内容
- `poall`：列出当前命名空间的全部名字

函数定义和调用

定义

```
make <word> [<list1> <list2>]
```

word为函数名

list1为参数列表

list2为操作列表

调用

```
<functionName> <arglist>
```

<functionName>为make中定义的函数名，不需要双引号"

<arglist>是参数列表，<arglist>中的值和函数定义时的<list1>中名字进行一一对应绑定

函数相关的操作

- `output <value>`：设定value为返回给调用者的值，但是不停止执行
- `stop`：停止执行
- `local <word>`：设定该word为本地名字。参数也是本地名字

既有名字

系统提供了一些常用的量，或可以由其他操作实现但是常用的操作，作为固有的名字。这些名字是可以被删除（erase）的。

- `pi`：3.14159

- `if <bool> <list1> <list2>`：如果bool为真，则执行list1，否则执行list2。list均可以为空表
- `run <list>`：运行list中的代码

对于程序的几点说明：

1. 每个有效字符之间必须以空格隔开，尤其是“[” 和 “]”来表示列表list和参数表arglist的时候，比如说一个列表 [abc 123 [a b 1 2]]，“[” 和 “]”前后不能与内容相连接。

2. butfirst 和 butlast两个函数返回的list是原来的copy，而别的函数都是返回其本身（即指针）

3. 对于递归调用，设定的最大值为maxn = 100,000，这个数值代表的是所有堆栈的总大小，堆栈单方向增长，使用过的堆栈并不会被重用，这是为了记录返回值的时候防止前后读取不一致。比如说计算斐波拉切数列的时候，f(i) = f(i-1) + f(i-2)，如果像C/C++这样的语言f(i-1) 和 f(i-2) 应该在同样的堆栈地址，因为调用完f(i-1)之后，堆栈弹出对应的值之后返回f(i)，再调用f(i-2)，这就使得f(i-1)和f(i-2)在同一个堆栈地址段里面，但是程序里面实现的是单方向增长，即便已经退出过程/函数也不会对这个过程/函数所使用的值做回收。

4. 对于注释 // 采用的是忽略这一行后面的所有字符，用getline来实现。

5. 每一句代码的开头必须都是以过程的形式来输入，即不会有任何的返回值，比如说 print thing A， 而单独的一个thing A会报错；所有的带有返回值的函数都必须被需要用到的过程所接受，如果不匹配则报错；

6. 对于基础类型和全局变量的说明:

number类型分别int和double，在输入的时候进行判断，然后用type来标记是哪种类型：

```
class number{
public:
    static const int type_int;
    static const int type_double;
    int type;
    int key_int;
    double key_double;
}
```

list类型包含一个value值和一个指向下一个元素的指针，value类型是多样化的，所以在列表里面表示所有的基础数据类型。每一个list的结尾都是以一个”End of list”这样的word来标记。

```
class list{
public:
    value* key;
    list* next;
```

```
}
```

基础数据类型value，包含了题目要求的4种类型：

```
class value{
public:
    static const int
type_number,type_string,type_list,type_bool;
    int type;
    number key_number;
    bool key_bool;
    string key_string;
    list* key_list;
}
```

和number类似，也是通过使用一个type来标记对应是具体哪种类型。

全局变量：

全局命名空间：

```
extern map<string,value*> _namespace;
```

每一个堆栈对应的命名空间，用来记录参数和本地变量的值

```
extern map<string,value*> _namespace_stack[maxn];
```

每一个堆栈对应的父堆栈标记，比如说stop之后要跳转回去对应的父堆栈的代码段。

```
extern int _namespace_stack_parent[maxn];
```

每一个堆栈对应的返回值，当然也可以没有返回值，比如说过程。

```
extern value* _namespace_stack_ret[maxn];
```

记录test的结果，用来判断iftrue,iffalse

```
extern bool _test_ret;
```

totalstack标记单方向增长的堆栈，curstack代表当前是第几号堆栈，主要用于访问对应的参数和本地变量。

```
extern int totstack,curstack;
```

具体相关测试如下：

```
make "A 123 // A = 123
print thing A
123
print : A
123
make "W Aword // W = Aword
print : W
Aword
print : not_exist_variable
The parameter of colon, not_exist_variable doesn't exist in namespace
Program ended with exit code: 1
```

对于运算符操作

```
make "A 123
make "W aword
make "B true
make "L [ 123 abc [ 1 2 a b ] ]
print isname L
True
print isname AAA
False
make "Readme read
321
print : Readme
321
print add : A : Readme
444
print sub : A : Readme
-198
make "Double 123.333
print mul : A : Double
15170
make "TT true
make "FF false
print not : TT
False
print and : TT : FF
False
print or : TT : FF
True
print eq : TT : FF
The parameters of eq is not number | word, error
Program ended with exit code: 1
```

```
make "A 123
make "B 312.123
print eq : A : B
False
make "zero 0
print div : zero : A
0
print div : A : zero
Divide zero!!
Program ended with exit code: 1
```

```
make "size 100
make "rand random : size
print : rand
23
make "rand2 random : size
print : rand2
35
make "sqr1 sqrt : size
print : sqr1
10
make "sqr2 sqrt 10.2
print : sqr2
3.19374
```

```
print isnumber : rand
True
print isword : rand
False
print islist : sqr1
False
make "ll
[ 1 2 3
     [ 1 2 3 a b cc ]
]
print islist : ll
True
print isempty : ll
False
print isbool : ll
False
print : ll
[
1
2
3
[
1
2
3
a
b
cc
]
]
```

```
make "l2
[ this is l2 ]
print isword item 2 : l2
True
join : l2 0.123
print : l2
[
this
is
l2
0.123
]
print list : ll : l2
[
1
2
3
[
1
2
3
a
b
cc
]
this
is
l2
0.123
]
```

```
print first : l2
this
print last : l2
0.123
print butfirst : l2
[
is
l2
0.123
]
```

测试wait: 可以看到是延迟了10秒钟（10,000ms）才出现123

```
wait 10000
print 123
```

```
wait 10000
print 123
123
```

打印当前的命名空间：

```
poall
The value of l2
[
this
is
l2
0.123
]
The value of ll
[
1
2
3
[
1
2
3
a
b
cc
]
this
is
l2
0.123
]
The value of pi
3.14159
The value of rand
23
The value of rand2
35
The value of size
100
The value of sqr1
10
The value of sqr2
3.19374
```

测试save和load：

先把他们存在一个文件里面，这里面命名为testing，然后erall清空所有的内容，poall打印测试，然后再load回来，之后使用poall来测试是否成功：

注意代码中save函数结束之前会把一个EOF这样的字符串写到文件里面，来标注文件结束，然后load的时候读取到EOF表示文件读取完毕。

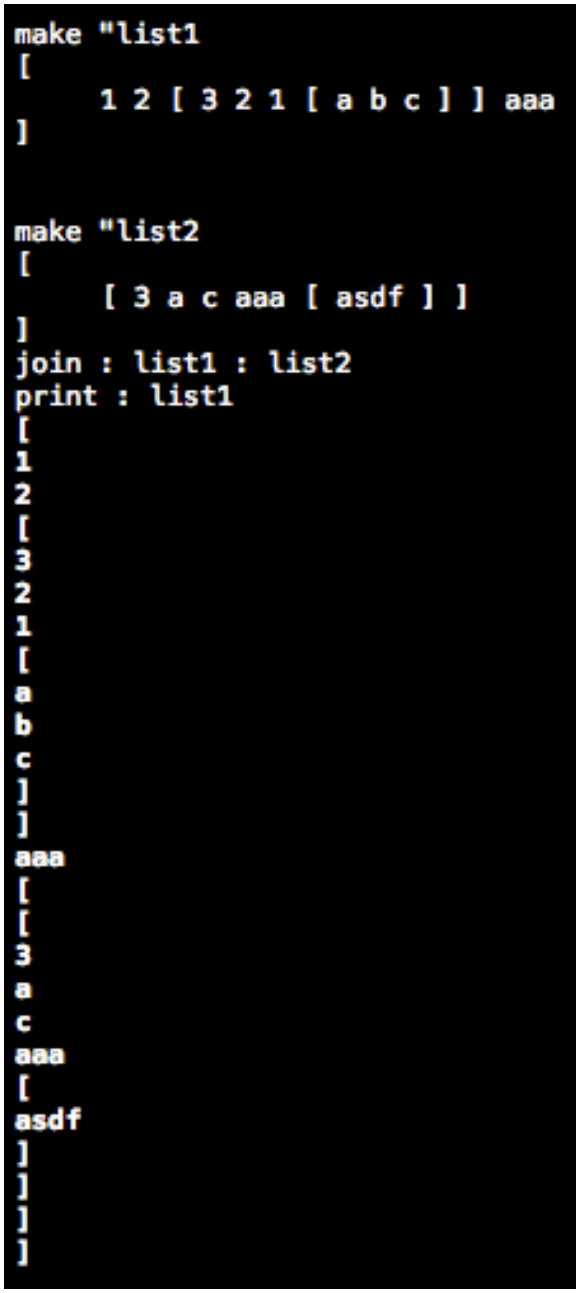
由于是在Xcode下测试，默认的路径是和机器配置有关，在我的机器上是：/Users/SmilENow/Library/Developer/Xcode/DerivedData/MUA-dwltroclwavoksgydsarukszrfil/Build/Products/Debug/testing

```
save testing
erall
poall
load testing
poall
The value of l2
[
this
is
l2
0.123
]
The value of ll
[
1
2
3
[
1
2
3
a
b
cc
]
this
is
l2
0.123
]
The value of pi
3.14159
The value of rand
23
The value of rand2
35
The value of size
100
The value of sqr1
10
The value of sqr2
3.19374
```

对于run和repeat函数测试：

```
make "pr [ print 123 ]
repeat 10 : pr
123
123
123
123
123
123
123
123
123
123
run : pr
123
```

对于list里面嵌套list等操作：



把list2作为一个元素插入到list1中，输出正确（输出格式没有去调整= =||）

对于iftrue，iffalse和函数的output,stop测试，这里面使用老师上课给的求阶乘和斐波拉切的函数：

factor代码如下：

```
make "factor
[
        [ n ]
```

```
        [
                test eq : n 1
                iftrue [ output 1 stop ]
                output mul : n factor sub : n 1
        ]
]
```

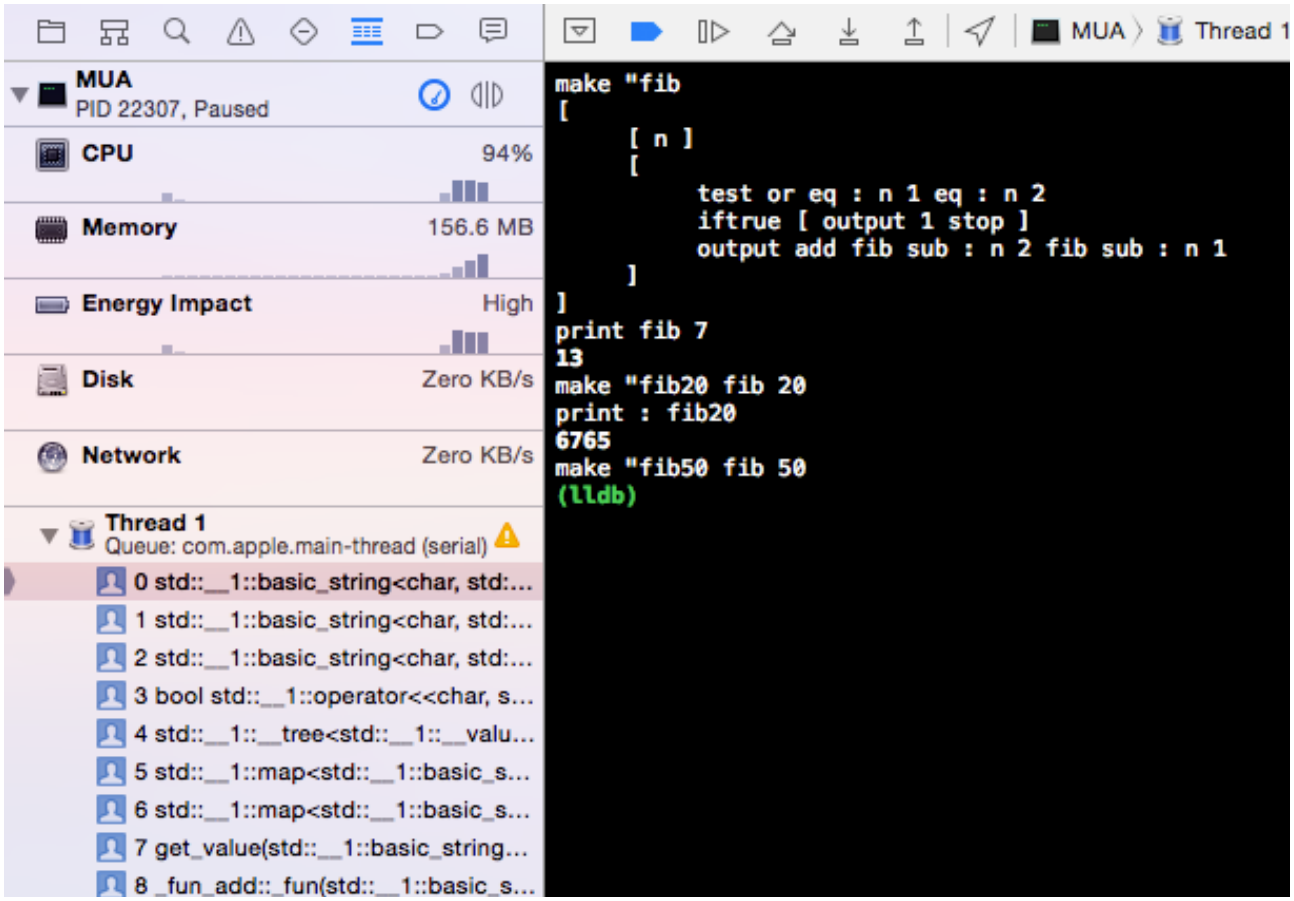
fib代码如下:

```
make "fib
[
        [ n ]
        [
                test or eq : n 1 eq : n 2
                iftrue [ output 1 stop ]
                output add fib sub : n 2 fib sub : n 1
        ]
]
```

测试结果:

```
make "factor
[
     [ n ]
     [
          test eq : n 1
          iftrue [ output 1 stop ]
          output mul : n factor sub : n 1
     ]
]
print factor 5
120
make "ten factor 10
print : ten
3628800
```

阶乘测试正确!



计算fib数列的时候，由于定义的单方向生长的堆栈长度是100k，而当计算fib(50)的时候早已超过100k（因为这样的代码算法的计算量是指数级别的，在coursera 的PPL上面的第二章也有同样的例子说明这个算法是不好的）

最后，程序的退出可以通过stop来退出（这里面相当于退出main函数一样）

```
stop
Exit successfullly.
Program ended with exit code: 0
```

后记：

整个MUA写下来遇到的问题很多，调程序的时间并不比写程序的时间短，同时，由于C++内存模型相对复杂，对于不少需要用到 itself (with pointer) 还是 copy 是需要斟酌的。同时程序中也对基础类型的运算进行了一定量的重载，一定程度上增加了可阅读性。总体下来还是收获蛮大的。^.^