# C 语言
# 期末项目文档

## Mao 语言运行器

邱超凡
1454001
软件学院

# 目录

# 1. 功能与要求

## i. 基本功能

Mao 语言项目的基本要求是编写一个 C 语言程序，能够接受来自文件的输入，执行文件中的代码，输出正确的结果。

Mao 语言代码包含分离的两个部分：变量定义和语句执行部分。变量定义的部分里可以有两种类型标识符，即 int 和 double 两种类型。一行可以定义多个变量，不能在定义变量时给变量赋值。

语句执行部分包含基本的加减乘除、赋值号和括号。此外有 print 语句，比如 print(a)可以输出变量 a 的值。一行只会有一个语句。

## ii. 扩展

Mao 语言运行器完整地支持以上提供的所有功能，并且提供一部分的扩展功能。其中包括：

- 变量声明部分和表达式计算部分可以交替出现
- 格式书写自由，一行可包含很多个语句，一个语句亦可拆成很多行
- 除了传统的+、−、*、/符号之外，亦增添了+=、−=、*=、/=等运算符
- print 语句可以输出表达式，也可以输出以双引号包括的常量

字符串，常量字符串能够正确识别转义字符

- 支持以科学计数法表示的浮点数
- 变量名可以有任意长度，可以以英文字母或下划线开头
- 对于输入中的错误，程序会给出智能的提示
- 支持 C 和 C++风格的注释

## 2. 实现过程

i. 容器类型

为了方便程序当中动态内存的管理，专门编写了三个容器：

- qmem_t，一个元素为一定长度内存的链表
- qstr_t，基于 qmem_t 实现的字符串
- qmap_t，可以以 qstr_t 为键的散列表

以上三种类型都是泛型的。但 C 语言里的结构体无法保存类型信息，所以在定义和加入新值的时候，需要传输一个参数，即容器内变量的类型。而类型无法作为函数的参数，因此这里用宏来实现。而容器的复制也利用了不需考虑类型的 memcpy 函数。

qmem_t 类型实质上是一个指向 qmemory_struct 类型的指针，qmemory_struct 结构体里保存了每个块的长度、总长度、链表的头和尾。链表每个节点的类型是 qmem_node 结构体，由三个成员组

成，即指向上一个和下一个的指针和一个 void*，用以存储任意类型的数据。

举个例子，假如要利用 qmem_t 存储 int 类型的数据，我们定义的 qmem_t 指向的结构体里保存了 int 类型的长度，每一块的长度（默认是 16），当前正在读写的指针位置。假设我们已经往里面加入了 15 个 int，那么再假如数据，实质上不是在链表中插入一个新节点，而是直接写那个 void 指针指向的内存里的数据。写入之后，数据变成了 16 个，此时如果要再次插入，负责插入的函数会自动插入一个新的节点。删除的过程亦如是。当每块长度被设定为 1 时，整个 qmem_t 会退化成一个普通的链表。具体的实现过程可以参看源码目录里的 qmemory.h 和 qmemory.c 文件。

qstr_t 类型的实现基本基于 qmem_t 类型。

除此之外，这两种类型我都设计了对应的迭代器，即 qmem_iter_t 和 qstr_iter_t 两个结构体，并且提供了一些接口，方便遍历这两种类型。整个 Mao 运行器的动态内存容器基本都是基于这两种类型。

项目里保存变量的部分用到了 qmap_t 类型。qmap_t 的实现基于一个简单有效的散列算法：遍历字符串，每次乘上 131 再加上当前字符的 ASCII 值，最后模指针指向的内存的容量（默认是 512）。解决冲突的方式是将每个元素的类型都设置成 qmem_t，如果两个元素

的散列值相同，它们会被放在一个链表里。

ii. 词法分析

为了使得 Mao 的代码格式更加自由，在解析源代码之前，先将字符串进行词法分析，将字符串组织为一个 token 组成的序列。每个 token 是一个结构体，定义在 lex.h 里。这个头文件里还定义了一些宏，用以表示 token 的类型。每个 token 都有一个 unsigned 成员，记录 token 所在的行，方便出错提示。

词法分析器会将输入的内容分类处理，如果是变量名，则归类成 identifier，如果是整数或者浮点数，就先保存，然后转换成数据存入 token 流。

词法分析器的实现在源码目录中的 lex.c 文件里。

iii. 构建表达式树

将字符串解析为一个以 qmem_t 类型作为容器的 token 流之后，将其传入函数 mao_parse，函数会根据 token 的类型判断语句的类型，选择恰当的函数进行处理。

如果是表达式，token 流会被传给 parse_expression 函数。函数会从表达式构建一个二叉树，交给负责计算的函数。具体把表达式解析成二叉树的过程是：

- 记录整个表达式是否被一个括号包住，如果是，去掉括号再

进行解析

- 寻找表达式中的等号，如果有，停止，然后分别解析左右
- 寻找最后一个括号外的优先级最低的运算符
- 如果整个表达式只有一个 token 并且是变量或数字，返回

iv. 变量系统

如上所述，Mao 语言运行时候的变量存储在一个 qmap_t 散列表里，散列表里的元素类型是 mvar，一个指向 mvar_struct 类型的指针。mvar_struct 结构保存了惟一的变量 id 和变量指向的对象。变量的类型和值都保存在它指向的对象中。对象是一个指向 mobject_struct 类型的指针。这两个类型都定义在 runtime.h 文件里。

在对表达式进行求值的时候，每次调用都会返回一个新的对象。计算的时候实质上没有变量和常量的区别，所有的计算都是对象的计算。

需要注意的是，用以计算的函数由于结构相似，因此是用同一个宏，和不同的关键字连接起来生成的。这样有助于简化代码。运算的代码定义在 object.c 文件中。

v. 内存管理与错误处理

Mao 运行器代码中大量运用了以上叙述的三个容器和迭代器，

方便了内存管理。而其他需要用到 malloc 函数的地方，对其进行了封装，定义了一个 qalloc 函数，当 malloc 分配失败时会自动退出。

在运算过程中产生的所有临时对象都会被注册到一个全局的 qmem_t 类型的表里。执行完每一句表达式后这个表里存储的指针指向的内存会被全部回收。

错误处理的宏定义在 error.h 中。

## 3. 开发过程中的收获

由于要实现通用的泛型容器，而 C 语言又不具备像 C++一样强大的模版功能，所以只能用 void 指针搭配宏来实现。因此我理解了 C 中指针强制类型转换的本质就是按照不同的方式来读写同一块内存。

而由于项目中有很多地方需要用到宏，而宏调试起来十分麻烦。因此也积累了一些调试技巧，比如 assert 的使用。同时认识到宏是一把双刃剑，既灵活，又容易造成麻烦的问题。

在开发过程中，处于提高代码质量的目的，查询了 C99 和 C11 标准提供的一些新特性。比如匿名联合体等也被运用到了项目当中。

由于此次项目的代码量比较大，因此模块化显得十分重要。因此在整个开发过程中，也加深了对“减少代码冗余”的理解。

## 4. 对项目的反思

整个项目的开发过程仍然有许多令人不满意之处。

首先是泛型容器的实现。尽管宏也能不低效地实现泛型容器的功能，但调试起来十分麻烦。并且在释放内存的时候，也无法智能地根据元素的类型执行对应的操作。比如要释放一个 qmem_t 容器，而里面的元素类型依然是 qmem_t，这样就必须要手动进行，先释放每个元素，再释放容器本身。

其次，项目中给表达式求值的方式需要反复地申请新内存，这样有可能会带来性能上的损失，也易造成内存泄漏。

同时，错误处理仍然没有一个很好的封装。项目中错误处理的实现只是在遇到错误的地方直接输出提示错误的语句，然后退出程序。一方面，输出错误提示的代码可以集中起来；另一方面，错误恢复也值得改进，遇到错误继续向下读取对用户而言会更加友好。