编译原理编程实践

Mandrill（山魈）语言编译器实现

编译原理课程组

福州大学计算机与大数据学院

2025 年 3 月 27 日

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。
- 它只有顺序、分支、循环三个核心语句。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。
- 它只有顺序、分支、循环三个核心语句。
- 仅支持整数的运算，字符常量、布尔类型均以整数存储和运算。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。
- 它只有顺序、分支、循环三个核心语句。
- 仅支持整数的运算，字符常量、布尔类型均以整数存储和运算。
- 简化运算符种类，只保留算术运算、关系运算，不讨论逻辑运算、位运算、地址运算。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。
- 它只有顺序、分支、循环三个核心语句。
- 仅支持整数的运算，字符常量、布尔类型均以整数存储和运算。
- 简化运算符种类，只保留算术运算、关系运算，不讨论逻辑运算、位运算、地址运算。
- 简化内存管理，只有全局变量（不考虑变量的作用域），没有数组、结构体、指针、引用等复杂类型。也没有函数调用等相关内容。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。
- 它只有顺序、分支、循环三个核心语句。
- 仅支持整数的运算，字符常量、布尔类型均以整数存储和运算。
- 简化运算符种类，只保留算术运算、关系运算，不讨论逻辑运算、位运算、地址运算。
- 简化内存管理，只有全局变量（不考虑变量的作用域），没有数组、结构体、指针、引用等复杂类型。也没有函数调用等相关内容。
- 取消变量的声明与定义、即写即用（类似 Python）。

概览

- Mandrill（山魈）是一门为了教学而生的命令式高级语言。
- 它只保留了命令式程序设计语言最基本的语法功能。
- 它只有顺序、分支、循环三个核心语句。
- 仅支持整数的运算，字符常量、布尔类型均以整数存储和运算。
- 简化运算符种类，只保留算术运算、关系运算，不讨论逻辑运算、位运算、地址运算。
- 简化内存管理，只有全局变量（不考虑变量的作用域），没有数组、结构体、指针、引用等复杂类型。也没有函数调用等相关内容。
- 取消变量的声明与定义、即写即用（类似 Python）。
- 输入输出功能由特殊关键字（保留变量）提供。

示例

N+1

```
1 output=input+1;
```

A+B

```
1 output=input+input;
```

GCD

```
1  a=input; b=input;
2  if (b > a) {
3      c = b; b = a; a = c;
4  }
5  while (b != 0) {
6      c = a % b;
7      a = b;
8      b = c;
9  }
10 output=a;
```

词法

字符集

- 字符分可见字符与不可见字符。
- 不可见字符有空白符（ASCII 码为 32）、换行符（ASCII 码为 10）、回车符（ASCII 码为 13）、制表符（ASCII 码为 9），它们只有分割词素（Token）的作用。
- 在 mandrill 源代码中可用的可见字符有：

```
abcdefghijklmnopqrstuvwxyz
0123456789
+-*/%
<>=!
()'\;
```

词法

词元

词元（token）是由语法定义的，不可再分割的词法单位，分标识符、关键字、运算符与常量四种。

词法

关键字

关键字（keyword）是具有特殊语法功能的词素。mandrill 的关键字只有七个：

`if`, `else`, `while`, `read`, `put`, `write`, `get`

词法

常量

常量（亦称字面值，literal）用于表示固定不变的值。mandrill 只有整数常量与字符常量。

整型常量

- 整数常量以十进制正整数表示（需要负数时，可以由“0”减正数得到）。
- 整数常量长度不超过 8 个字符。

字符常量

- 字符常量是用单引号围绕的可见字符、空格符或转义字符。
- 转义字符（Escape Character）用于表示一些无法或不方便在代码中直接输入的字符。mandrill 至少有 3 种转义字符：
  - `\n` 表示换行
  - `\\` 表示斜杠
  - `\'` 表示单引号

词法

运算符

运算符是表示运算的符号。mandrill 只有两种运算类型：

- 算术运算
  - 加性：加号（+）、减号（-）
  - 乘性：乘号（*）、除号（/）、取模（%）
- 关系运算
  - 比较：大于（>）、小于（<）、大于等于（>=）、小于等于（<=）
  - 相等：等于（==）、不等于（!=）

词法

标识符

- 标识符（identifier）是用来为概念命名的词素。在没有自定义函数及自定义类型的前提下，标识符的唯一作用就是为变量命名。
- 标识符只允许使用小写英语字母（不允许用数字是为了方便词法分析）。
- 不同标识符数量不超过 1024 个，每个标识符长度少于 16 个字符。
- 关于变量，mandrill 有三项规定：
  - 使用变量不需要声明。
  - 所有变量均具有全局作用域。
  - 所有变量的初始值均为零。

词法

输入变量与输出变量

`read` 与 `write` 是特殊的变量，当 `write` 为赋值语句左值时，语义为以十进制整数的格式输出右值。例如以下语句的效果是向屏幕输出 30：

```
1 write = 10 + 20;
```

当 `read` 参与求值计算时，语义是从输入中以十进制整数的格式获取一个整数值，例如以下表达式的求值的效果是将输入的数字翻倍：

```
1 read * 2
```

词法

输入变量与输出变量

- 为了具备处理字符的能力，mandrill 还定义了 put 与 get 关键字，其语义与 write 及 read 类似，不同之处是 get 只从输入中获取一个字符，所获字符的 ACSII 码就是 get 的值，put 只会输出一个字符，这个字符 ACSII 码就是提供给 put 的右值数值。若提供给 put 的右值是个不合法或无法输出的字符，则不输出任何数据。
- 若 get 或 read 无法从输入中获得数据时，它们会返回数值零。
- 特别地，read 的行为可以参考`scanf("%d", &n);`；类似的，get 的行为可以参考`getchar();`；

语法

表达式

- 表达式（expression）是由运算符（operator）及运算元（operand）组成的序列，用于描述一个值的计算过程。
- mandrill 语言中表达式的巴科斯范式（BNF）表述如下（已考虑了所有运算符的优先级与结合性）：

```
1 expression ::= equality
2
3 equality ::= equality == comparison | equality != comparison
4 equality ::= comparison
5
6 comparison ::= comparison < term | comparison <= term
7 comparison ::= comparison > term |comparison >= term
8 comparison ::= term
```

语法

表达式

```
term ::= term + factor
term ::= term - factor
term ::= factor

factor ::= factor / primary
factor ::= factor * primary
factor ::= factor % primary
factor ::= primary

primary ::= constant | identifier
primary ::= read | write | put | get
primary ::= ( expression )
```

语法

表达式

- **单目运算**：为了化简编译的工作，mandrill 省略了两种常用的单目运算：负号及取反（逻辑非）。其实双目运算也能提供这两种运算：
  - 表达式 `-exp` 可以由 `(0 - (exp))` 代替。
  - 表达式 `!exp` 可以由 `((exp) == 0)` 代替。

语法

语句序列

mandrill 的程序是由一个**语句序列**组成的，即**顶层语法成分**（或称**起始符号**）为语句序列。一个语句序列可以为空，可以只含一条语句或多条语句。为了简化实现的难度，规定一个语句序列最多支持 256 条语句。

比如 while、if 等语句包含子语句序列，子语句序列一样受此条件制约。

语法

语句

mandrill 的语句仅有**赋值语句**、**循环语句**、**条件语句**三种。

语法

赋值语句

- 赋值（assignment）是指令式程序设计语言（Imperative Programming Language）最基本的功能。

  赋值语句 `::= lvalue = expression ;`

- 赋值的语义：
  1. 对 `expression` 求值，求值所得称为右值。
  2. 若 `lvalue` 是一个变量，则该变量的新值赋为右值。
  3. 若 `lvalue` 是 `write`，将右值以十进制整数输出。
  4. 若 `lvalue` 是 `put`，将右值解释为 ASCII 码，输出对应的字符。

语法

循环语句

- 语法

  循环语句 `::= while (expression) {` 循环体语句序列 `}`

- 与 C 语法不同之处在于：循环语句的花括号不可省略，必须明确指明循环体范围。
- 循环语句的语义：
  1. 对条件表达式求值。
  2. 若求值结果非零，则子语句（称之为循环体）被执行，反复这个过程，直到条件表达式为零时停止。

语法

条件语句

- 语法：

  选择语句 `::= if (`表达式`) { `语句序列` } else { `语句序列` }`

- 语义：先对表达式求值，若其值非零，则执行第一条语句序列，否则执行第二条语句序列。
- 由于没有逻辑与或非，相应的计算考虑使用算术操作代替，例如：
  1. 逻辑与：算术乘法，有一个是 0 结果即是 0；
  2. 逻辑或：算术加法，有一个是 1 结果非 0；
  3. 逻辑非：1-表达式结果，1 变 0，0 变 1。

示例

统计字符数量

```
1 while (get != 0) {
2     c = c + 1;
3 }
4 write = c;
```

示例
角谷猜想

```
1  n = read;
2  while (n > 1) {
3      if (n % 2 == 0) {
4          n = n / 2;
5      }
6      else {
7          n = 3*n + 1;
8      }
9      write = n;
10     put = '\n';
11 }
```

示例

计算水仙花指数

```
1 c = get;
2 while (c != 0) {
3     d = c - '0';
4     s = s + d * d * d;
5     c = get;
6 }
7 write = s;
```

代码管理要求

1. 使用 Git 版本管理软件。
2. 在 Bitbucket 上创建账号，建立名为 `mandrill2025` 的代码仓库。
3. 编写代码，并将代码推送到仓库。
4. 给任课教师的账号 `croissantfish` 开通可读权限。
5. 在共享文档上填写自己的 bitbucket 账号。
6. 按照每阶段测试的要求给提交版本的代码打上特定的标签。
7. 老师会在约定的时间从克隆仓库并检出（checkout）带有特定标签的版本，进行测试。
8. 测试过程由统一脚本自动化进行。

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;
- Git 常用命令:

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;
- Git 常用命令:
  - 提交文件到暂存区: `git add <file>`;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;
- Git 常用命令:
  - 提交文件到暂存区: `git add <file>`;
  - 提交一个版本: `git commit -m < 提交信息 >`;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;
- Git 常用命令:
  - 提交文件到暂存区: `git add <file>`;
  - 提交一个版本: `git commit -m < 提交信息 >`;
  - 推送到远端仓库: `git push -u origin main`;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;
- Git 常用命令:
  - 提交文件到暂存区: `git add <file>`;
  - 提交一个版本: `git commit -m < 提交信息 >`;
  - 推送到远端仓库: `git push -u origin main`;
  - 给版本打标签: `git tag <tagname> <commit id>`;

Git

- Git 是目前世界上最先进的分布式版本控制系统。
- 绝大多数互联网公司都采用 Git 作为版本管理的主要手段。
- 为什么需要版本控制?
  - 回溯修改的历史（提交与检出）;
  - 与他人协作时如何快速合并两个人的版本（比较和合并）;
  - 隔离开发与发布（分支与标签）;
- Git 常用命令:
  - 提交文件到暂存区: `git add <file>`;
  - 提交一个版本: `git commit -m < 提交信息 >`;
  - 推送到远端仓库: `git push -u origin main`;
  - 给版本打标签: `git tag <tagname> <commit id>`;
  - 推送标签到远端仓库: `git push --tags`;

Bitbucket
介绍

- Bitbucket 是由 Atlassian 开发和维护的 Git 代码托管服务，主要面向团队协作和 DevOps 需求。它提供了丰富的功能来管理代码、进行持续集成/持续交付（CI/CD）、以及项目管理。
- 免费、无用户数量限制。
- 方便国内用户访问。

Bitbucket

创建仓库并授予权限-步骤指南

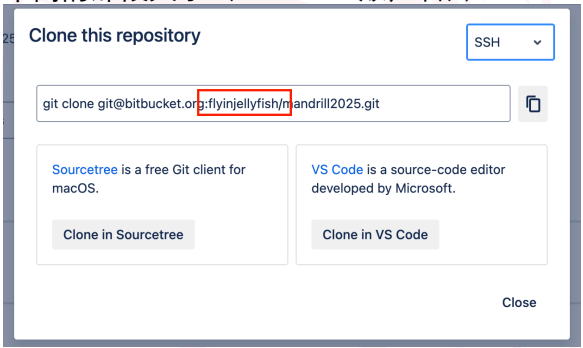
1. 访问网址`https://bitbucket.org`创建并登录个人账号；
2. 访问网址`https://bitbucket.org/youraccount/workspace/repositories/`，点击右上角的“创建仓库（Create repository）”；（`youraccount` 替换为你的实际用户名）
3. 仓库名字字段输入 `mandrill2025`；
4. 项目名称不影响，创一个新项目名字自定即可；
5. 取得项目链接，如：`https://bitbucket.org/youraccount/mandrill2025/`；
6. 在左侧边栏依次点击“仓库设置（Repository Settings）-仓库权限（Repository permissions）”进入仓库权限管理界面，然后点击右上角的“Add users or groups”，在弹出的窗口中搜索并添加账号“`croissantfish`”并赋予可读权限。
7. 注：现版本的 Bitbucket 可能需要先添加到工作空间才能添加到仓库权限。请向 `songyu-ke@outlook.com` 邮箱发出邀请，加入工作空间。

Bitbucket
填写问卷

填写问卷提供 Bitbucket 账号信息：

注意事项：

1. 账户名字不要填写邮箱，应填写 ssh 克隆连接中间的那段文字（bitbucket 账户名）；



测试脚本

```
#!/bin/bash
# 省略若干行环境设置
# $names为学生账户名构成的列表
for name in ${names[@]}; do
    # 省略若干行学生测试环境设置，如创建文件夹
    if [ ! -d $name ]; then
        _SC mkdir $name "&&" cd $name
        _SC git clone "git@bitbucket.org:${name}/mandrill2025.git"
        _SC cd ..
    fi
    _SC cd $name/mandrill2025
    _SC git checkout -f main # 现在git的分支默认是main，以前是master
    _SC git pull
    _SC git fetch --tags
    _SC git checkout -f <tag> # 这里会根据测试阶段不同选用不同的tag
```

测试脚本

```
16      # 用make命令进行构建
17      _SC make clean "&&" make
18      _SC cat ./finalvars.sh
19      # CCHK为可执行文件执行命令导出变量，由脚本finalvars.sh设置
20      # 不要求具体的可执行文件命名，通过变量自动调用
21      unset -v CCHK
22      set -x
23      source ./finalvars.sh #can't ensure SC here
24      set +x
25      echo CCHK=$CCHK
26      _SC cd bin
27      for filec in $(ls $normaldir/*.mx); do
28          # 省略若干行逐文件测试过程
29          timeout $timeo $CCHK <data.mx 1>assem.s
30      done
```

测试脚本

```
31      # 统计测试成绩
32      echo count: $score/$full_score >>$log_file
33      echo count: $score/$full_score
34      echo $name $score/$full_score >>$statistics
35      _SC cd ..
36      _SC git checkout -f master
37      cd $outdir
38  done
```

代码需要准备的文件

1. Makefile 文件，用于告诉 make 命令如何编译你们的代码，编译的可执行文件和字节码要拷贝到根目录下的 `bin` 目录；
2. xxxvars.sh 文件，用于导出环境变量，告诉 bash 如何执行你们的代码；
3. 代码输入输出通过测试脚本的命令自动重定向到对应文件，你们写的代码从 stdin 读取输入并输出到 stdout 即可；

Makefile 示例

Java 语言

```
1  BASE_PATH = cn/edu/sjtu/songyuke/mental
2  all:
3      # 调用子目录下的Makefile文件递归构建
4      $(MAKE) -C src all
5      mkdir -p bin/cn/edu/sjtu/songyuke/mental
6      ... # 省略10多行创建目录命令
7      if [ ! -d bin/$(BASE_PATH)/ast ]; then mkdir bin/$(BASE_PATH)/ast;
8      ... # 省略10多行复制文件命令
9      cp src/$(BASE_PATH)/ast/*.class bin/$(BASE_PATH)/ast/
10     ... # 复制其他必要文件
11     cp src/antlr-4.5.3-complete.jar mips_built_in.s built_in.mx bin/
12     $(MAKE) -C src clean
13 clean:
14     $(MAKE) -C src clean
15     -rm -rf bin
```

Makefile 示例

C 语言

```
1  all:
2      # 调用子目录下的Makefile文件递归构建
3      $(MAKE) -C src all
4      gcc main.c -o main
5      mkdir -p bin
6      cp main bin/main
7  clean:
8      $(MAKE) -C src clean
9      -rm -rf bin
```

Makefile 示例

Python

```
1 all:
2     # 调用子目录下的Makefile文件递归构建
3     $(MAKE) -C src all
4     mkdir -p bin
5     cp src/*.py bin/
6 clean:
7     $(MAKE) -C src clean
8     -rm -rf bin
```

xxxvars.sh 示例

Java 语言：

```
#!/bin/bash
export CCHK="java -cp .:antlr-4.5.3-complete.jar\
 cn.edu.sjtu.songyuke.mental.core.Run"
```

C 语言：

```
#!/bin/bash
export CCHK="./lexer"
# 假设为需要运行的可执行文件为lexer.
```

语言：

```
#!/bin/bash
export CCHK="python ./lexer.py"
# 假设为需要运行的python脚本为lexer.py。
```

词法分析器要求

- Git 标签为 `lexer`，设置 `CCHK` 变量的脚本名为 `lexervars.sh`
- 从 stdin 读入源代码，输出到 stdout。
- 输入的源代码保证不存在词法错误。
- 使用指定格式按顺序将分割好的词元输出即可。
- 格式要求：
  1. 每一行输出一个词元，格式为 `[< 类型 >:< 值 >]`。
     注意：`[`，`]`，`:` 均不是 mandrill 语言使用的可见字符，因此用它们作为限定符和分隔符。
  2. 类型有 `keyword`（关键字）、`literal`（常量）、`op`（运算符）、`id`（标识符）四种。
  3. 值对应如下：
     - 如果词元类型为关键字、运算符、标识符，则输出对应的串，如 `if`，`<`，`a`；
     - 如果词元类型为常量，则 `0x` 开头的十六进制值（a-f 使用小写字母，字符常量转换为 ASCII 码后再转十六进制值）；

词法分析器要求

示例

输入：

```
1 write = read + read;
```

输出：

```
1 [keyword:write]
2 [op:=]
3 [keyword:read]
4 [op:+]
5 [keyword:read]
6 [op:;]
```

语法分析器实现要求

- Git 标签为 `parser`，设置 `CCHK` 变量的脚本名为 `parservars.sh`
- 从 stdin 读入源代码，输出到 stdout。
- 输入源代码保证没有词法错误，但不保证没有语法错误。
- 如果输入的源代码没有语法错误，输出 PASS，否则输出 ERROR。

语法分析器实现要求

示例

输入 1:

```
1 write = read + read;
```

输出 1:

```
1 PASS
```

输入 2:

```
1 write read + read;
```

输出 2:

```
1 ERROR
```

语义分析器及解释器实现要求

- Git 标签为 `interpreter`，设置 `CCHK` 变量的脚本名为 `interpretervars.sh`。
- 从 stdin 读入源代码，输出到 stdout，额外有一个输入文件为 mandrill.in。
- 如果输入的 mandrill 代码有使用 read 或者 get 读取输入，则从文件 mandrill.in 读取，测试脚本将保证代码执行目录存在 mandrill.in 这一文件。
- 输入的源代码保证不存在语法和词法错误。
- 保证源代码在给定输入的情况下不会发生死循环。

虚拟机及代码生成阶段实现要求

- Git 标签为 `final`，设置 `CCHK` 变量的脚本名为 `finalvars.sh`。
- 注意，由于此阶段涉及虚拟机对编译结果的执行，为此，需要额外设置一个虚拟机运行命令的变量 `VMRUN`[1]。
- CCHK 从 stdin 读入源代码，输出字节码到 stdout。
- VMRUN 应支持一个命令行参数 `<filename>`，为读取的字节码文件名，然后从 stdin 读入对应输入，将程序运行结果输出到 stdout 中。（类似 Python 的运行命令 `python main.py`）
- 输入的源代码保证不存在语法和词法错误。
- 保证源代码在给定输入的情况下不会发生死循环。

[1] `export VMRUN=...`

Mandrill 虚拟机

- Mandrill 虚拟机采用类似 Java 最初版本的虚拟机架构，即虚拟机主要维护一个栈，用于存储操作数。
- 此外，虚拟机还会维护两个分别用于存储代码和变量数据的存储区以及一个程序计数器（Program Counter，PC）。
- 程序计数器是一个 32 位整数，指向代码存储区中下一轮执行指令的第一字节地址，从 0 开始，每条指令 8 个字节，正常指令执行后 PC 自动加 8。跳转和分支求值指令会修改 PC 的值。后续字节码会详细介绍。
- 当 PC 恰好等于 `0xFFFFFFFF` 时，虚拟机退出并返回 0。即程序执行的最后一条指令强制为 `jump 0xFFFFFFFF`。

虚拟机指令

Mandrill 虚拟机的指令一般由 ‘命令 + 操作对象 ‘构成，其中命令为固定单词，操作对象为立即数、运算符类型以及内存地址构成。接下来将介绍每一个指令。

- 数据加载指令：用于从数据存储区对应位置读取数据并存入操作数栈。
  格式：`dload x`，`x` 为变量编号，从 0 开始。
- 数据存储指令：用于将数据（立即数）压入操作数栈。
  格式：`dstore x`，`x` 为立即数。
- 数据写入指令：用于将操作数栈顶的元素写入对应存储位置并将其弹出。
  格式：`dwrite x`，`x` 为目标变量编码。

虚拟机指令

- 求值指令：根据立即数 x 进行对应操作。
  如果 x 对应二元运算，则取出操作数栈顶两个元素进行对应的计算后将结果存回操作数栈顶。
  如果 x 对应条件跳转运算，则取出三个操作数，第一个元素为表达式的结果。若表达式结果不为零，将 PC 设置为第二个元素（THEN 分支），否则设置为第三个元素（ELSE 分支）。
  格式：`eval x`，x 为立即数，运算类型与对应的值参考后面字节码介绍。
- 直接跳转指令：该指令将直接把 PC 设置为 x。
  格式：`jump x`。
- 无操作指令：不执行任何操作。
  格式：`nop 0`，后面的操作对象可以是任意整数，均应当被忽略。

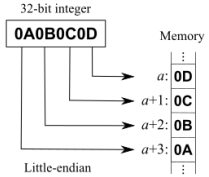
虚拟机指令

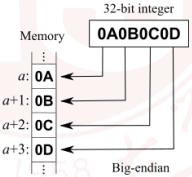
- 整数输入指令：操作数无用。
  格式：`geti x`，`x` 为 dummy（无用），将从 stdin 读入一个十进制整数的数据存入栈顶。
- 字符输入指令：操作数无用。
  格式：`getc x`，`x` 为 dummy，将从 stdin 读入一个字符并把它对应的 ASCII 值存入栈顶。
- 整数输出指令：操作数无用。
  格式：`puti x`，`x` 为 dummy，将栈顶元素弹出并以十进制的形式输出到 stdout 中。
- 字符输入指令：操作数无用。
  格式：`putc x`，`x` 为 dummy，将栈顶元素弹出并判断它的值是否为合法的 ASCII 字符，若合法则输出对应字符到 stdout 中，否则什么都不做。

Mandrill 虚拟机字节码

指令结构

- 简化实现，所有指令的字节码均为二进制存储的 64 位整数。其中前 32 位为指令，后 32 位为操作数。
- 虽然操作数均为 32 位，但不同指令含义不同，可能对应立即数、运算类型、变量地址或者跳转地址。
- **所有 Mandrill 字节码中涉及的指令值、立即数、地址等均为大端序（Big Endian）。**





Mandrill 虚拟机字节码

文件结构

- 使用 .mbc 作为字节码的文件后缀。
- 文件由文件头（File Header）和文件体（File Content）构成。

Mandrill 虚拟机字节码

文件头

- Mandrill 虚拟机的字节码文件头共计 32 个字节，包含 16 字节魔数、4 字节版本号、4 字节数据区大小定义、4 字节代码区大小定义，以及最后 4 个字节的填充（任意内容）。
- 字节码文件使用 `MANDRILLBYTECODE` 作为魔数（共 16 个字节）。
- 紧跟魔数的是字节码版本，目前版本为 1，使用 4 个字节的 32 位整数表示（大端序）。
- 版本号后面的是数据区定义，使用一个 32 位整数表示（大端序）表示数据区所需大小（字节数），用于告知虚拟机需要申请多大的内存来存储变量的值。注意，Mandrill 所有变量初始化为 0。越界访问应当直接拒绝并报错。
- 数据区定义后紧跟一个 32 位整数表示（大端序）表示代码区大小（字节数）。

Mandrill 虚拟机字节码

文件体

- 从第 33 个字节开始存储字节码指令。
- 每条指令均为二进制存储的 64 位大端序整数，前 32 位表示指令类型，后 32 位表示立即数、操作地址等附加信息。
- 超过代码区定义大小的指令被视为多余部分，虚拟机应将其忽略。

Mandrill 虚拟机字节码

操作码

指令编码：

- 0x00000000 nop
- 0x00000001 dstore
- 0x00000002 dload
- 0x00000003 dwrite
- 0x00000005 eval
- 0x00000006 jump
- 0x00000007 geti
- 0x00000008 getc
- 0x00000009 puti
- 0x0000000A putc

eval 指令操作数：

- 加法：0x00010001
- 减法：0x00010002
- 乘法：0x00010003
- 整除：0x00010004
- 求余：0x00010005
- 大于：0x00010006
- 小于：0x00010007
- 大等于：0x00010008
- 小等于：0x00010009
- 等于：0x0001000A
- 不等于：0x0001000B
- 条件跳转：0x0001000C

实验报告要求

撰写一个 Mandrill 语言编译器完整实践过程的报告，包括但不限于

- 技术路线：词法分析器设计，语法及语法分析器设计（用 LL(1)、LR 还是 LALR 算法等）；
- 实现的优化功能：常量压缩、短路表达式（自动检测逻辑与、或并在确定结果的情况下提前返回结果）等；
- 实现过程的心得体会；
- 致谢；

祝愿大家能够借本课程实践项目的机会巩固理论知识，提高编程水平，为未来的学习、研究、工作打下坚实的基础。