**MiniC语言编译器说明**

1. MiniC编译器框架
2. 源代码说明
3. MiniC编译器使用说明
4. MiniC虚拟机说明

1.用MiniC编译器编译MiniC程序的过程如下：

字节码(虚拟机可执行)

语法树

语义分析

代码生成

语法分析

词法分析

语法树

MiniC源代码

符号流

和常规的编译器一样，MiniC编译器使用lex和yacc工具先对MiniC源代码进行词法分析和语法分析，然后对生成的语法树进行语义分析检查其中的语法错误。最后再通过语法树生成MiniC虚拟机可执行的字节码。代码生成的过程会在之后的内容中简述。

2.整个MiniC编译器以及虚拟机的源代码列表如下：

|  |  |
| --- | --- |
| ├─compiler  │ create.c  │ display\_tree.c  │ error.c  │ fix\_tree.c  │ generate.c  │ interface.c  │ makefile  │ minic.h  │ minic.l  │ minic.y  │ queue.c  │ string.c  │ util.c  │ y.output  │  ├─display  │ back.php  │ bootstrap.min.css  │ disassemble  │ disassemble.php  │ jquery-2.1.4.js  │ jquery.jOrgChart.css  │ jquery.jOrgChart.js  │ lex  │ lex.php  │ page.html  │ refresh.php  │ test.php  │ test.txt  │  ├─include  │ MC.h  │ MEM.h  │ MVM.h  │ MVM\_code.h  │ MVM\_dev.h  │ node.h  │ share.h  │  ├─main  │ helloworld  │ main.c  │ makefile  │ minic.exe  │ minic\_test  │ quicksort  │ struct  │  ├─memory  │ Makefile  │ memory.c  │ memory.h  │ storage.c  │  ├─mvm  │ execute.c  │ heap.c  │ makefile  │ mvm\_pri.h  │ native.c  │  └─share  disassemble.c  Makefile  opcode.c | compiler目录  创建语法分析用的结构的实现  compiler报错的实现  语义分析部分实现  代码生成实现  接口实现，比如创建虚拟机，进行编译等  Compiler目录的makefile  编译器用的头文件  词法分析使用的lex文件  语法分析用的yacc文件  对字符串的处理的实现文件  minic头文件中部分函数的实现  Include目录，包含被多个模块用到的头文件  包含编译器相关的声明  内存管理模块的相关声明  虚拟机结构的相关声明  虚拟机字节码以及虚拟机类型结构的声明  虚拟机用到的一些接口函数的声明  一些公用的函数和结构体  main目录  helloworld样例程序  程序入口  main目录的makefile  可执行文件  minic\_test样例程序  快排样例程序  结构体样例程序  memory目录，内存管理模块  makefile  部分内存管理的实现  内存管理函数的声明  存储管理的实现  mvm目录，虚拟机模块  虚拟机执行的实现  堆管理的实现  makefile  虚拟机的一些声明  原生函数的实现  share目录  将生成的字节码反汇编打印出来  makefile  字节码的可读形式的声明 |

3.MiniC编译器使用说明：

（1）点击上传按钮输入需要编译的文件，文件后缀名为.mc

（2）点击编译按钮完成编译

（3）显示程序的Token Stream

（4）点击右侧箭头查看语法分析树

（5）点击右侧箭头查看报错信息或运行结果

（6）点击左侧箭头可查看前面阶段的显示信息

（7）点击清空按钮可重新编译

4.MiniC VM说明：

VM主要结构：

|  |
| --- |
| Struct MVM\_VirtualMachine**{**  Stack stack**;**  Heap heap**;**  Static static\_v**;**  int pc**;**  Function **\***function**;**  Int function\_count**;**  **}** |

1.stack:运行时，局部变量、函数的参数和返回信息均需在栈上创建空间。

2.heap:堆是通过引用访问的内存区域，字符串和数组保存在堆中。

3.static\_v:保存程序的全局变量。

4.pc:程序计数器。

5.function:程序中所有函数的数组，保存所有的函数信息。

6.function\_count:程序中函数的个数。

VM可执行文件（字节码和一些其他信息）的格式：

1.constant\_pool：常量池，保存常量的内存空间。

2.global\_variable：保存全局变量列表。

3.function：保存函数的信息，主要是函数的代码。

4.code：保存顶层结构的代码。

5.line\_number：保存字节码与之对应的源代码的行号。

VM字节码生成和运行原理：

在生成语法树之后，语法树优先从最深层次开始遍历。常量或者变量就直接把值保存到栈中。双目运算符以先左后右的顺序保存到栈上，从栈顶端的两个元素开始计算，并将结果保存到栈中。

例如对于表达式：1 + 2 \* 3 – 4

会产生这样一棵语法树：

常量节点生成字节码push值、运算符节点生成代表运算符的字节码，将会生成以下字节码（以下字节码只是用来举例说明，并不是和MiniC字节码完全一样的）：

|  |  |
| --- | --- |
| 1: push 1  2: push 2  3:push 3  4:mul  5:add  6:push 4  7:sub | #将1入栈  #将2入栈  #将3入栈  #将栈顶的两个元素进行乘法运算，并将结果入栈  #将栈顶的两个元素进行加法运算，并将结果入栈  #将4入栈  #将栈顶的两个元素进行减法运算，并将结果入栈 |

运行时，栈经过每一步时的状态如下：

栈顶两元素进行减法

将4入栈

栈顶两元素进行加法

栈顶两元素进行乘法

将3入栈

将2入栈

将1入栈

3

4

7

7

6

1

3

2

1

2

1

1