**项目总结文档**

成员组成：

商文翰：2016302580076（组长）

曹鑫杰：2016302580075

姚连杰：2016302580077

胡红彬：2016302580078

陆艺璇：2016302580053

分工：

|  |  |
| --- | --- |
| 商文翰-076 | 词法分析器设计实现；  参与语法分析器的设计实现，主要负责：  抽象语法树AST的设计实现  部分Parse函数的实现：部分statement的parse函数实现（statemnet、if、while、block、print）  参与中间代码LLVM IR生成工作，主要负责Block、PRINT（包括PrintStatAST、TextPrintItemAST等）的实现； |
| 曹鑫杰-075 | 参与了语法分析器的设计实现，主要负责部分Parse函数的实现，包括所有expression和function相关的parse函数的实现（还实现了VSL中负表达式），以及部分statement的parse函数实现（Null、Return、Assign） |
| 姚连杰-077 | 主要承担了添加JIT支持和支持优化部分；  负责了VSL的+-\*/的实现（主要涉及中间代码生成部分）  参与中间代码LLVM IR生成工作，主要负责DeclExprAST、AssignStatAST的实现。 |
| 胡红彬-078 | 参与中间代码LLVM IR生成工作，主要负责，IF、WHILE部分的代码生成；  负责词法分析器的输出工作；  负责语法分析器的输出工作；  参与完成目标代码生成和执行（和陆艺璇合作完成）； |
| 陆艺璇-053 | 参与中间代码LLVM IR生成工作，主要负责NullStatAST、ReturnStatAST的中间代码实现，以及所有expression和function相关的codegen的修改。  参与完成目标代码生成和执行； |

**个人贡献的自我陈述：**

**商文翰：**

作为这次解释器实验的组长，我参与了大部分任务，并负责了主要的词法、语法分析器的设计，参与词法分析器、语法分析器的实现和中间代码LLVM IR生成工作。我完成了词法分析器，和抽象语法树的完整设计实现，并设计了语法分析器和中间代码生成的框架结构，参与了部分语法分析器和中间代码生成中难点的实现。比如，VSL 中print方法的实现、以及IF的debug等等。总的来说，自我感觉对整个团队还是很有贡献的。

当然，很多问题的解决我们所有人的合作，比如实现print过程中曾找到组员胡红彬来合作，胡红彬IF实现遇到bug来找我帮忙；但是我发现还存在一些问题，比如分工之后，除了我对所有人的代码比较熟悉（因为我做的设计），其他组员对其他人的代码不是很熟悉，遇到问题比较难互相帮助。

**曹鑫杰：**

在此次的解释器实验中，我参与了一些VSL设计文档的编写以及测试文档的编写，以及语法分析器的设计实现。具体来说主要负责了部分Parse函数的实现，其中包括了所有expression和function相关的Parse函数的实现，比如VSL中的负表达式。另外还有部分statement的parse函数的实现，例如Null，Return,Assign。最后我还参与了全部代码的测试。

**胡红彬：**

在此解释器实验中，我参与了生成中间代码、词法信息输出、语法树输出以及目标代码生成。除了完成这些任务，还会和小组成员一起完成大家无法独立解决的问题，并在网络上大量寻找相关信息以求解决问题。同时也参与了其他同学的一部分前期工作，比如print部分，打印text的语法分析以及中间代码生成。也参与了目标代码生成后的链接生成可执行文件的测试，但是在虚拟机上使用gcc链接会出现不兼容的问题，最后只能向陆艺璇寻求帮助。

**陆艺璇：**

在此解释器实验中，我参与了生成LLVM IR的工作，并完成了目标代码生成，编写简单的程序进行测试。在此过程中，遇到难以解决的问题，我会积极联系其他小组进行沟通和解决，或是从stackoverflow寻求解决方法。如生成.o文件时，llvm官方教程说是Ctrl+D，实际上windows系统上所需操作位Ctrl+z。通过阅读和学习组员的其它代码，我还掌握了应如何实现词法分析器、语法分析器。我打算今后有空继续完善此编译器，想办法做出UI界面。

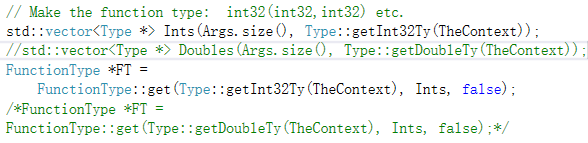
**挑战和解决思路：**

**1.LLVM万花筒所有数据类型是double,而VSL中使用的是int类型**

**挑战：** llvm教程中全程使用的都是double，很多地方都对数据类型有隐含要求，换为int类型就无法使用，而且还需要将int改为相应的llvm类型，中间遇到了非常多bug，（如类型不匹配等等）

**解决思路：**找到所有原本为double或float的部分，通过vs的转到定义功能，去查看llvm相关部分的源代码，找到int可以使用的函数使用（下图展示了部分修改后的成果）。





**2. 挑战：**一开始并没有很好地了解到中间代码的生成过程，由于万花筒给出了IF语句的中间代码部分，因此if部分的中间代码生成比较好完成，但是没有while语句的参考，感觉没有什么突破口。

**解决思路：**后来认真阅读if语句中间代码生成部分的代码，才慢慢了解到if的中间代码生成是通过分块来完成的，也大致有了一定的理解，于是也将while分块生成中间代码。

**3.IF中间代码生成中一直出现类型不匹配的bug**

**挑战：**if语句的中间代码生成已经根据教程进行了合适的修改，但是始终存在这个bug，怎么也找不到原因

**解决思路：**仔细研究LLVM，发现其PHI节点要求IF A THEN B ELSE C FI中，ABC是同一类型，而VSL中A是expression，BC是statement，所以让StatAST继承ExprAST。而且PHINode还对其ABC的codegen()的返回值有要求，需要修改其他codegen()的代码（如：ReturnStatAST的codegen()本来设计的返回 Builder.CreateRet(Val),但是不被PHINode兼容，所以改成返回Exp->codegen() ）。

**4.PRINT实现困难重重！！！**

**挑战：**Print函数的实现没有任何思路，找不到突破口；找到思路后，遇到很多打印问题，例如打印数字、表达式

**解决思路：**首先是在stack overflow上寻找解决思路，借鉴了一些想法，由此找到突破口，设计实现了print函数，但是存在很多打印数字和表达式的问题。深入了解调用函数对参数的需求后，发现，参数必须先压入字符串，字符串中包含%d，最后在压入对应个数的值才可以。

**5.挑战：**生成目标代码时，要跳出mainloop循环必须输入文件结尾标志符，按照万花筒上我们输入CTRL+D但是无法跳出循环。  
**解决思路：**后来在网上查阅得知EOF是文件结尾标志符，同时windows下文件结尾标志符为CTRL+Z

**6.挑战：**生成目标文件.o后我们发现.o文件的大小并不会随着输入的VSL代码改变，由此我们推断.o文件只是一个空壳，并非实际的目标代码。  
**解决思路：**后来比较万花筒生成目标文件的代码我们发现，万花筒在生成目标文件代码时并没有JIT优化支持，同时我们的mainloop循环中如果读取到函数定义那么调用函数HandleFuncDefinition，其中函数最后调用了InitializeModuleAndPassManager，由此推断出每次生成中间代码后，TheModule将会被初始化，最终生成的.o文件也应该是一个错误的目标文件。因此我们注释掉了这段初始化TheModule的代码，发现我们可以成功生成正确的目标文件。