编译原理

实验 2 报告

学　　号：20188068

班级序号：180235

姓　　名：孔天欣

提交日期：2020 年 11 月 10 日

成　　绩：

东北大学秦皇岛分校

实验 2 使用自顶向下语法分析实现基本编译器

【实验目的和要求】

掌握一个含有基本词法分析和语法分析功能的编译器的实现过程，重点掌握语法分析功能的实现，能够使用基本编译器实现对如下源程序的语法分析，并编写一个简单的 C++程序进行外部函数调用测试编译后生成的目标代码的准确性，进一步掌握编译原理的完整实现过程。

可以在已有语法分析功能的基础上设计实现更多的语法分析功能， 比如更多运算符和更多语句（条件语句、循环语句等）的语法分析功能（可选）。

【实验内容】

（1）需要编译的源程序示例：

```
def average(x y) (x+y)/2;

def sum(x y) x+y;
```

（2）针对上述源程序设计实现一个基本的编译器，并测试编译结果的准确性，可以使用反汇编工具查看生成的目标代码的汇编指令。

【实验环境】

Linux 操作系统（eg：Ubuntu），CLion 开发环境，LLVM 库，GCC 和 G++编译工具，CMake。

【实验结果】

（1）语法分析功能实现部分的过程描述

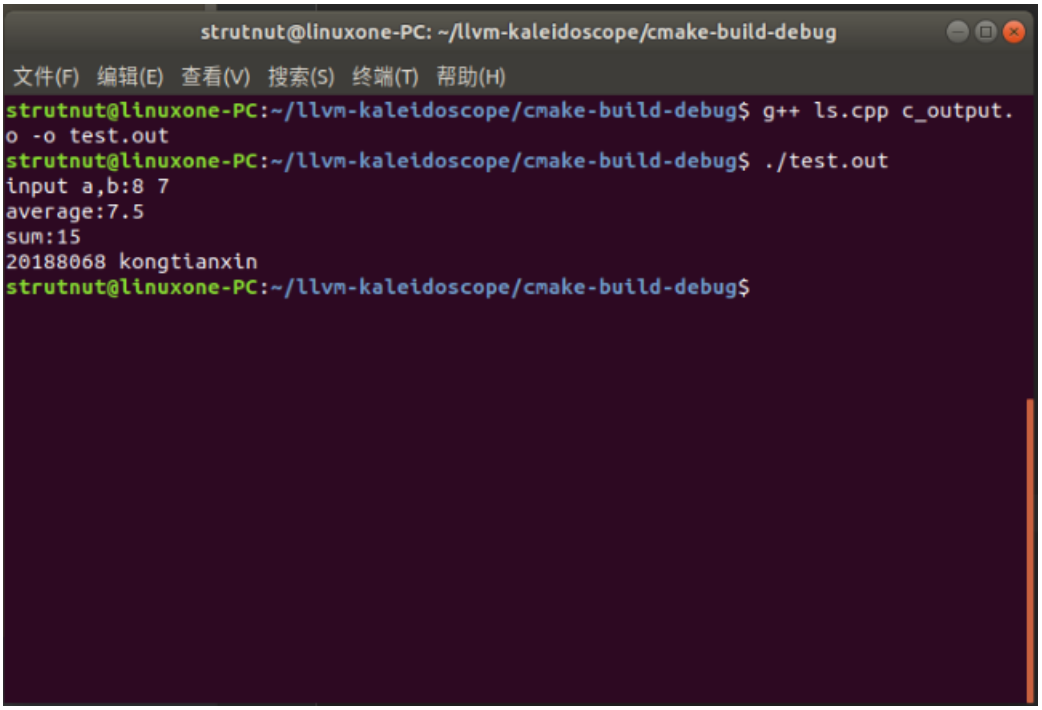
编译器首先采用词法分析 Lexer 获取 token 流，然后根据 token 流使用语法分析 Parser 生成抽象语法树 AST，它是基于自顶向下递归分析下降法 LL(1)和算符优先分析法进行构造的。主要包括 ExprAST 和它派生的 VariableExprAST （变量）、BinaryExprAST（运算）、CallExprAST（函数）等，每个 AST 都是一个实体类，并在其内部成员变量中存储这种抽象表示的具体数据或者还有其他 AST 结构，它们根据语法分析组合构造成抽象语法树。这种语法分析方法的大致流程是这样的：先读取一个 token，然后判别当前 token 的类别，然后调用相应的 parseExpr()方法，创建相应的 ExprAST 节点，以此递归下降继续分析，最后返回结果。

在开始生成语法树之前，还需要驱动入口函数 MainLoop()来启动语法分析，除此以外，它还负责在解析到一些特殊 token 时调用相应的解析函数。

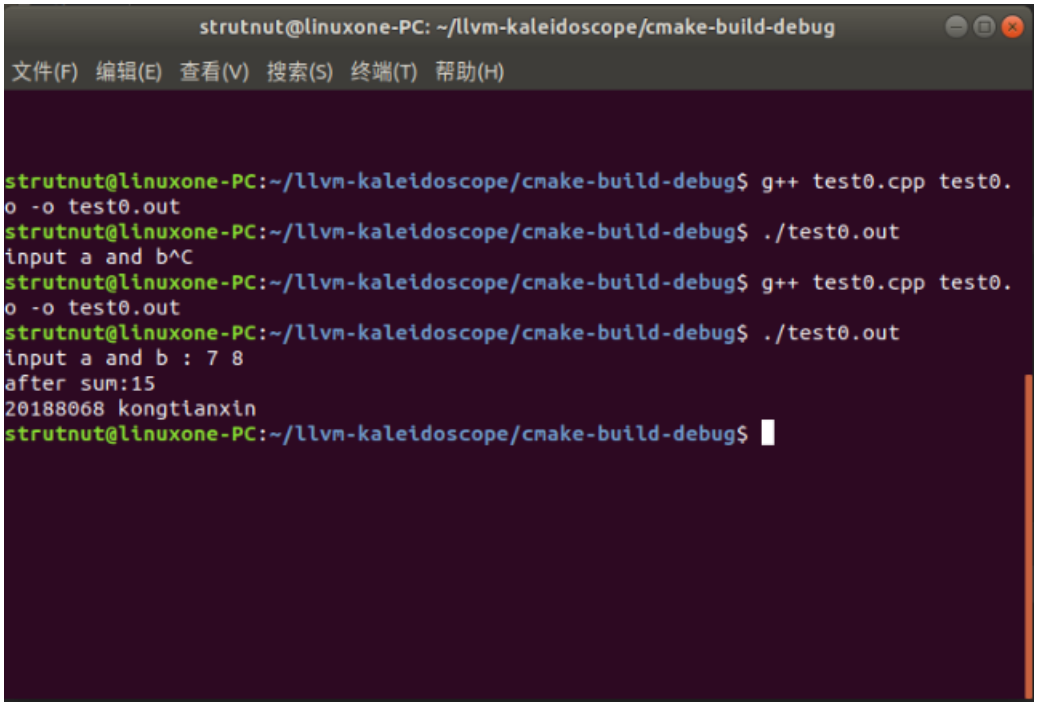
生成语法树完毕后，依靠 LLVM 进行中间代码生成 LLVM IR。

（2）测试程序的执行结果

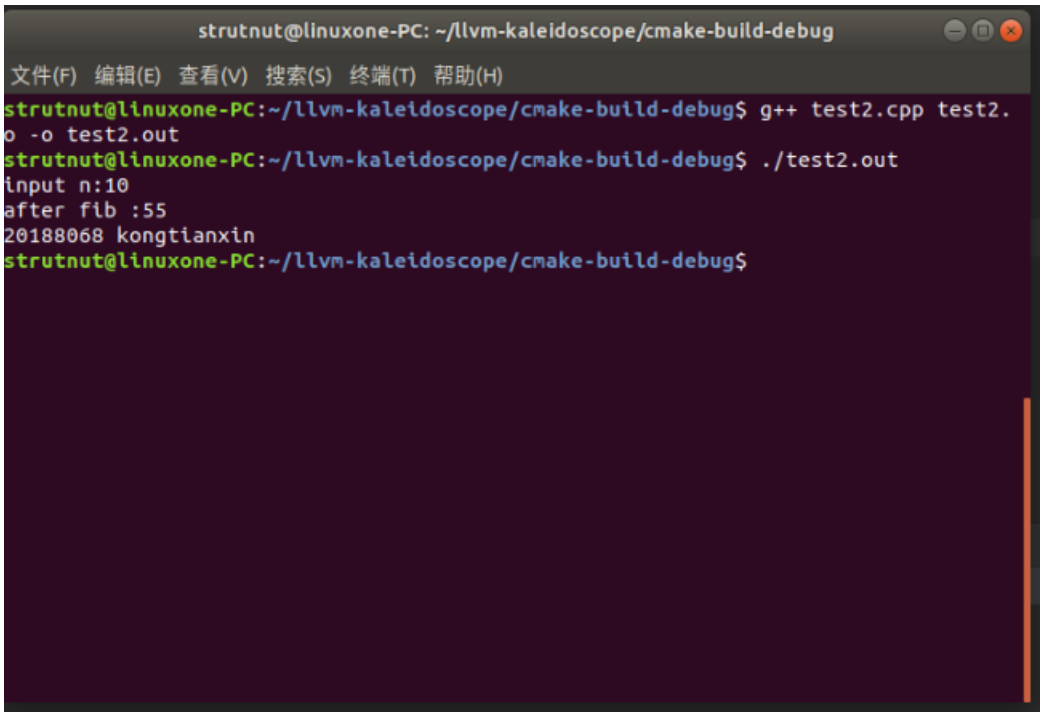
源代码 def average(x y) (x+y)/2; 对应的结果如下：



源代码 def sum(x y) x+y; 对应的结果如下：

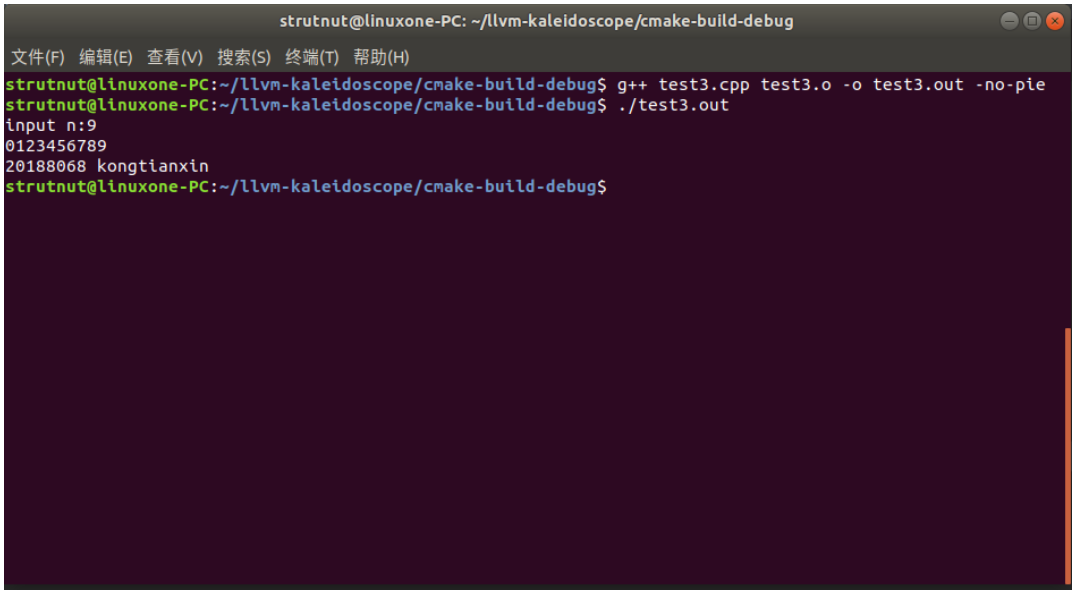


源代码 def fib(x) if x < 3 then 1 else fib(x-1)+fib(x-2); 对应的结果如下：



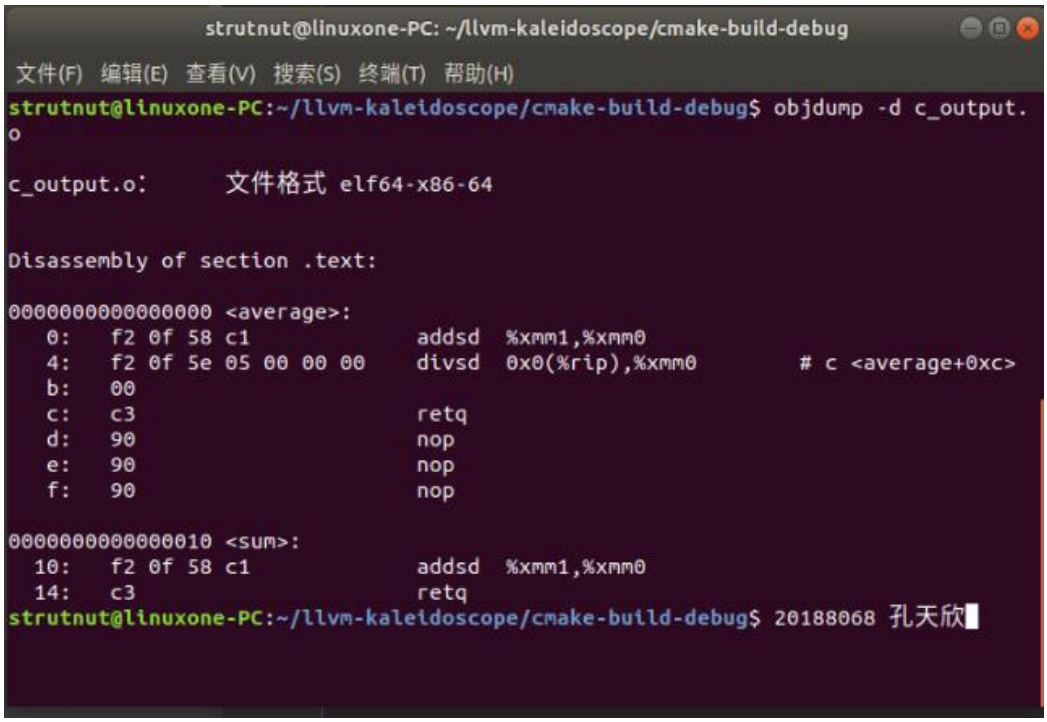
源代码 extern putchard(char); def printNumbers(n) for i = 0, i < n, 1.0 in putchard(48+i);

对应的结果如下：

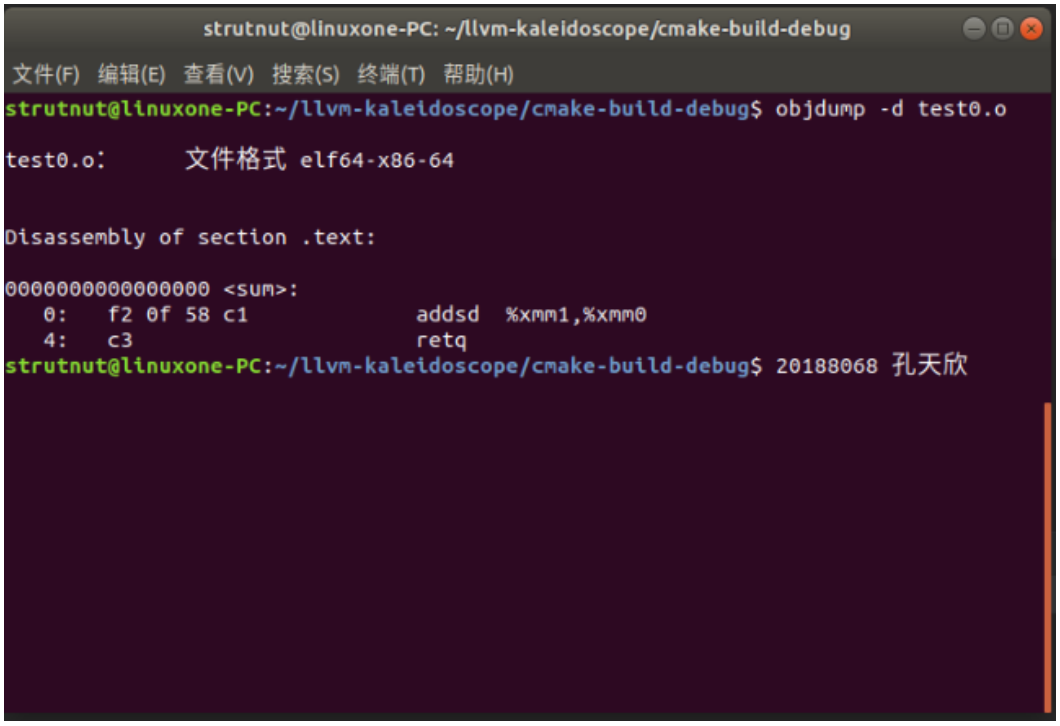


（3）源代码生成的目标指令反汇编后的汇编指令

源代码 def average(x y) (x+y)/2; 生成目标指令，反汇编后的汇编指令如下：



源代码 def sum(x y) x+y; 生成目标指令，反汇编后的汇编指令如下：



源代码 def fib(x) if x < 3 then 1 else fib(x-1)+fib(x-2); 生成目标指令，反汇编后的汇编指令如下：

```
strutnut@linuxone-PC: ~/llvm-kaleidoscope/cmake-build-debug
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)

strutnut@linuxone-PC:~/llvm-kaleidoscope/cmake-build-debug$ objdump -d test2.o

test2.o:     文件格式 elf64-x86-64


Disassembly of section .text:

0000000000000000 <fib>:
   0:   48 83 ec 18             sub    $0x18,%rsp
   4:   66 0f 28 d8             movapd %xmm0,%xmm3
   8:   f2 0f 10 0d 00 00 00    movsd  0x0(%rip),%xmm1        # 10 <fib+0x10>
   f:   00
  10:   f2 0f c2 c8 06          cmpnlesd %xmm0,%xmm1
  15:   f2 0f 10 05 00 00 00    movsd  0x0(%rip),%xmm0        # 1d <fib+0x1d>
  1c:   00
  1d:   66 0f 54 c8             andpd  %xmm0,%xmm1
  21:   0f 57 d2                xorps  %xmm2,%xmm2
  24:   66 0f 2e ca             ucomisd %xmm2,%xmm1
  28:   74 05                   je     2f <fib+0x2f>
  2a:   48 83 c4 18             add    $0x18,%rsp
  2e:   c3                      retq
  2f:   f2 0f 10 05 00 00 00    movsd  0x0(%rip),%xmm0        # 37 <fib+0x37>
  36:   00
  37:   f2 0f 58 c3             addsd  %xmm3,%xmm0
  3b:   f2 0f 11 5c 24 08       movsd  %xmm3,0x8(%rsp)
  41:   e8 ba ff ff ff          callq  0 <fib>
  46:   f2 0f 11 44 24 10       movsd  %xmm0,0x10(%rsp)
  4c:   f2 0f 10 44 24 08       movsd  0x8(%rsp),%xmm0
  52:   f2 0f 58 05 00 00 00    addsd  0x0(%rip),%xmm0        # 5a <fib+0x5a>
  59:   00
  5a:   e8 a1 ff ff ff          callq  0 <fib>
  5f:   f2 0f 58 44 24 10       addsd  0x10(%rsp),%xmm0
  65:   48 83 c4 18             add    $0x18,%rsp
  69:   c3                      retq
strutnut@linuxone-PC:~/llvm-kaleidoscope/cmake-build-debug$ 20188068 孔天欣
```

源代码 extern putchard(char); def printNumbers(n) for i = 0, i < n, 1.0 in putchard(48+i);

生成目标指令，反汇编后的汇编指令如下：

```
strutnut@linuxone-PC: ~/llvm-kaleidoscope/cmake-build-debug
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
test3.o:     文件格式 elf64-x86-64


Disassembly of section .text:

0000000000000000 <printNumbers>:
   0:   48 83 ec 18             sub    $0x18,%rsp
   4:   f2 0f 11 44 24 08       movsd  %xmm0,0x8(%rsp)
   a:   66 0f 57 d2             xorpd  %xmm2,%xmm2
   e:   90                      nop
   f:   90                      nop
  10:   f2 0f 11 54 24 10       movsd  %xmm2,0x10(%rsp)
  16:   66 0f 28 c2             movapd %xmm2,%xmm0
  1a:   f2 0f 58 05 00 00 00    addsd  0x0(%rip),%xmm0        # 22 <printNumbers+0x22>
  21:   00
  22:   e8 00 00 00 00          callq  27 <printNumbers+0x27>
  27:   f2 0f 10 54 24 10       movsd  0x10(%rsp),%xmm2
  2d:   f2 0f 10 44 24 08       movsd  0x8(%rsp),%xmm0
  33:   f2 0f c2 c2 06          cmpnlesd %xmm2,%xmm0
  38:   f2 0f 10 0d 00 00 00    movsd  0x0(%rip),%xmm1        # 40 <printNumbers+0x40>
  3f:   00
  40:   f2 0f 58 d1             addsd  %xmm1,%xmm2
  44:   66 0f 54 c1             andpd  %xmm1,%xmm0
  48:   66 0f 2e 04 25 00 00    ucomisd 0x0,%xmm0
  4f:   00 00
  51:   75 bd                   jne    10 <printNumbers+0x10>
  53:   66 0f 57 c0             xorpd  %xmm0,%xmm0
  57:   48 83 c4 18             add    $0x18,%rsp
  5b:   c3                      retq
  5c:   90                      nop
  5d:   90                      nop
  5e:   90                      nop
  5f:   90                      nop

0000000000000060 <__unnamed_1>:
  60:   50                      push   %rax
  61:   f2 0f 10 05 00 00 00    movsd  0x0(%rip),%xmm0        # 69 <__unnamed_1+0x9>
  68:   00
  69:   e8 00 00 00 00          callq  6e <__unnamed_1+0xe>
  6e:   58                      pop    %rax
  6f:   c3                      retq
strutnut@linuxone-PC:~/llvm-kaleidoscope/cmake-build-debug$ 20188068 孔天欣
```

（4）心得体会

通过本次实验，本人通过阅读并理解现有的简单编译器源码的词法分析和语法分析过程，成功运行了基于万花筒语言的编译程序，实现了该语言的相关功能，还成功拓展了 if-else,for 循环等控制流程的支持。在这次实验后，本人对编译原理的理解不再囿于书本上的理论知识，还能够将理论知识运用到实践中去，进而对编译原理的过程和原理得到了更深入的理解，同时还认识到设计一个能够实现基本功能的编译器并非想象中的难度巨大，在既有成熟框架的支持下可以较为轻松地实现语言功能的拓展。