lab2 实验报告

201908010705 杨杰

实验要求

本次实验需要各位同学首先将自己的 lab1 的词法部分复制到 `/src/parser` 目录的 lexical_analyzer.l并合理修改相应部分，然后根据 `cminus-f` 的语法补全 syntax_analyer.y 文件，完成语法分析器，要求最终能够输出解析树。如：

输入：

```
int bar;
float foo(void) { return 1.0; }
```

则 `parser` 将输出如下解析树：

```
>--+ program
|  >--+ declaration-list
|  |  >--+ declaration-list
|  |  |  >--+ declaration
|  |  |  |  >--+ var-declaration
|  |  |  |  |  >--+ type-specifier
|  |  |  |  |  |  >--* int
|  |  |  |  |  >--* bar
|  |  |  |  |  >--* ;
|  |  >--+ declaration
|  |  |  >--+ fun-declaration
|  |  |  |  >--+ type-specifier
|  |  |  |  |  >--* float
|  |  |  |  >--* foo
|  |  |  |  >--* (
|  |  |  |  >--+ params
|  |  |  |  |  >--* void
|  |  |  |  >--* )
|  |  |  |  >--+ compound-stmt
|  |  |  |  |  >--* {
|  |  |  |  |  >--+ local-declarations
|  |  |  |  |  |  >--* epsilon
|  |  |  |  |  >--+ statement-list
|  |  |  |  |  |  >--+ statement-list
|  |  |  |  |  |  |  >--* epsilon
|  |  |  |  |  |  >--+ statement
|  |  |  |  |  |  |  >--+ return-stmt
|  |  |  |  |  |  |  |  >--* return
|  |  |  |  |  |  |  |  >--+ expression
|  |  |  |  |  |  |  |  |  >--+ simple-expression
|  |  |  |  |  |  |  |  |  |  >--+ additive-expression
|  |  |  |  |  |  |  |  |  |  |  >--+ term
|  |  |  |  |  |  |  |  |  |  |  |  >--+ factor
|  |  |  |  |  |  |  |  |  |  |  |  |  >--+ float
|  |  |  |  |  |  |  |  |  |  |  |  |  |  >--* 1.0
|  |  |  |  |  |  |  |  >--* ;
```

```
|  |  |  |  |  >--* }
```

请注意，上述解析树含有每个解析规则的所有子成分，包括诸如 `;` `{` `}` 这样的符号，请在编写规则时务必不要忘了它们。

2.1 目录结构

```
.
├── CMakeLists.txt
├── Documentations
|   ├── lab1
|   └── lab2
|       ├── readings.md         <- 扩展阅读
|       └── README.md           <- lab2实验文档说明（你在这里）
├── README.md
├── Reports
|   ├── lab1
|   └── lab2
|       └── report.md           <- lab2所需提交的实验报告（你需要在此提交实验报告）
├── include                      <- 实验所需的头文件
|   ├── lexical_analyzer.h
|   └── SyntaxTree.h
├── src                          <- 源代码
|   ├── common
|   |   └── SyntaxTree.c       <- 分析树相关代码
|   ├── lexer
|   └── parser
|       ├── lexical_analyzer.l <- lab1 的词法部分复制到这，并进行一定改写
|       └── syntax_analyzer.y  <- lab2 需要完善的文件
└── tests                        <- 测试文件
    ├── lab1
    └── lab2                     <- lab2 测试用例文件夹
```

2.2 编译、运行和验证

- 编译

  与 lab1 相同。若编译成功，则将在 `${WORKSPACE}/build/` 下生成 `parser` 命令。

- 运行

  与 `lexer` 命令不同，本次实验的 `parser` 命令使用 shell 的输入重定向功能，即程序本身使用标准输入输出（stdin 和 stdout），但在 shell 运行命令时可以使用 `<` `>` 和 `>>` 灵活地自定义输出和输入从哪里来。

  ```
  $ cd cminus_compiler-2021-fall
  $ ./build/parser                # 交互式使用（不进行输入重定向）
  <在这里输入 Cminus-f 代码，如果遇到了错误，将程序将报错并退出。>
  <输入完成后按 ^D 结束输入，此时程序将输出解析树。>
  $ ./build/parser < test.cminus # 重定向标准输入
  <此时程序从 test.cminus 文件中读取输入，因此不需要输入任何内容。>
  <如果遇到了错误，将程序将报错并退出；否则，将输出解析树。>
  $ ./build/parser test.cminus  # 不使用重定向，直接从 test.cminus 中读入
  $ ./build/parser < test.cminus > out
  <此时程序从 test.cminus 文件中读取输入，因此不需要输入任何内容。>
  <如果遇到了错误，将程序将报错并退出；否则，将输出解析树到 out 文件中。>
  ```

通过灵活使用重定向，可以比较方便地完成各种各样的需求，请同学们务必掌握这个 shell 功能。

此外，提供了 shell 脚本 `/tests/lab2/test_syntax.sh` 调用 `parser` 批量分析测试文件。注意，这个脚本假设 `parser` 在 `项目目录/build` 下。

```
# test_syntax.sh 脚本将自动分析 ./tests/lab2/testcase_$1 下所有文件后缀为 .cminus 的文件，并将输出结果保存在 ./tests/lab2/syntree_$1 文件夹下
$ ./tests/lab2/test_syntax.sh easy
  ...
  ...
  ...
$ ls ./tests/lab2/syntree_easy
  <成功分析的文件>
$ ./tests/lab2/test_syntax.sh normal
$ ls ./tests/lab2/syntree_normal
```

- 验证

  本次试验测试案例较多，为此我们将这些测试分为两类：

  1. easy: 这部分测试均比较简单且单纯，适合开发时调试。
  2. normal: 较为综合，适合完成实验后系统测试。

  我们使用 `diff` 命令进行验证。将自己的生成结果和助教提供的 `xxx_std` 进行比较。

```
$ diff ./tests/lab2/syntree_easy ./tests/lab2/syntree_easy_std
# 如果结果完全正确，则没有任何输出结果
# 如果有不一致，则会汇报具体哪个文件哪部分不一致
# 使用 -qr 参数可以仅列出文件名
```

  `test_syntax.sh` 脚本也支持自动调用 `diff`。

```
# test_syntax.sh 脚本将自动分析 ./tests/lab2/testcase_$1 下所有文件后缀为 .cminus 的文件，并将输出结果保存在 ./tests/lab2/syntree_$1 文件夹下
$ ./tests/lab2/test_syntax.sh easy yes
  <分析所有 .cminus 文件并将结果与标准对比，仅输出有差异的文件名>
$ ./tests/lab2/test_syntax.sh easy verbose
  <分析所有 .cminus 文件并将结果与标准对比，详细输出所有差异>
```

  请注意助教提供的 `testcase` 并不能涵盖全部的测试情况，完成此部分仅能拿到基础分，请自行设计自己的 `testcase` 进行测试。

2.3 提交要求和评分标准

- 提交要求

  本实验的提交要求分为两部分：实验部分的文件和报告，git提交的规范性。

  - 实验部分:
    - 需要完善 `./src/parser/lexical_analyzer.l` 文件;
    - 需要完善 `./src/parser/syntax_analyzer.y` 文件;
    - 需要在

```
./Report/lab2/report.md
```

撰写实验报告。

- 实验报告内容包括:
  - 实验要求、实验难点、实验设计、实验结果验证、实验反馈(具体参考report.md);
  - 实验报告不参与评分标准，但是必须完成并提交.
- 本次实验收取 `./src/parser/lexical_analyzer.l` 文件、`./src/parser/syntax_analyzer.y` 文件和 `./Report/lab2` 目录

- git提交规范:
  - 不破坏目录结构(`report.md` 所需的图片请放在 `./Reports/lab2/figs/` 下);
  - 不上传临时文件(凡是自动生成的文件和临时文件请不要上传);
  - git log言之有物(不强制, 请不要git commit -m 'commit 1', git commit -m 'sdfsdf'，每次commit请提交有用的comment信息)

- 评分标准
  - git提交规范(6分);
  - 实现语法分析器并通过给出的 easy 测试集(一个3分，共20个，60分);
  - 通过 normal 测试集(一个3分，共8个，24分);
  - 提交后通过助教进阶的多个测试用例(10分)。
- 迟交规定
  - `Soft Deadline`：【未定】
  - `Hard Deadline`：【未定】
  - 补交请邮件提醒TA:
    - 邮箱：shibc@hnu.edu.cn
    - 邮件主题：lab2迟交-学号
    - 内容：包括迟交原因、最后版本commitID、迟交时间等
  - 迟交分数
    - x为迟交天数(对于 `Soft Deadline` 而言)，grade满分10

```
final_grade = grade, x = 0
final_grade = grade * (0.9)^x, 0 < x <= 7
final_grade = 0, x > 7
```

- 关于抄袭和雷同

  经过助教和老师判定属于作业抄袭或雷同情况，所有参与方一律零分，不接受任何解释和反驳。

实验难点

- 确定 `token` 和 `type` 包含哪些元素（实际上就是确定终结符和非终结符集合）；
- 明白 `node` 函数的作用，了解如何通过 `node` 函数构造语法分析树；
- 知道如何将文法产生式转换为 `bison` 语句（有一个要注意的地方是空串怎么处理）；
- 看懂 `pass_node` 函数，了解 `bison` 与 `flex` 之间是如何协同工作。

实验设计

- `token` 包括Lab1写的 `IDENTIFIER INTEGER FLOATPOINT LPARENTHESE RPARENTHESE LBRACE RBRACE LBRACKET RBRACKET ARRAY ADD SUB MUL DIV LT LTE GT GTE EQ NEQ ASSIN SEMICOLON COMMA ELSE IF INT RETURN VOID WHILE FLOAT` 和新增加的 `MAIN`。

- `type` 包括 `type-specifier relop addop mulop declaration-list declaration var-declaration fun-declaration local-declarations compound-stmt statement-list statement expression-stmt iteration-stmt selection-stmt return-stmt expression simple-expression var additive-expression term factor integer float call params param-list param args arg-list program`。
- `node` 函数是用来构造语法分析树，它以参数 `name` 命名父节点，同时为父节点连接子节点。
- 将文法产生式转换为 `bison` 语句的关键在于明白当前节点使用 `$$` 代表，而已解析的节点则是从左到右依次编号，称作 `$1`, `$2`, `$3`。
- `bison` 与 `flex` 之间通过 `pass_node` 函数协同工作。
- 注意词法分析器中对识别 `error` 后返回值的处理。

实验结果验证

- 原始测试样例

  easy测试集：

```
(base) yj@myubuntu:~/Documents/MyDoc/HNU课程/大三上/编译原理/cminus_compiler-2021-fall$ ./tests/lab2/test_syntax.sh easy yes
[info] Analyzing array.cminus
[info] Analyzing call.cminus
[info] Analyzing div_by_0.cminus
[info] Analyzing expr-assign.cminus
[info] Analyzing expr.cminus
[info] Analyzing FAIL_array-expr.cminus
error at line 2 column 17: syntax error
[info] Analyzing FAIL_decl.cminus
error at line 2 column 10: syntax error
[info] Analyzing FAIL_empty-param.cminus
error at line 1 column 10: syntax error
[info] Analyzing FAIL_func.cminus
error at line 1 column 18: syntax error
[info] Analyzing FAIL_id.cminus
error at line 1 column 6: syntax error
[info] Analyzing FAIL_local-decl.cminus
error at line 4 column 5: syntax error
[info] Analyzing FAIL_nested-func.cminus
error at line 3 column 13: syntax error
[info] Analyzing FAIL_var-init.cminus
error at line 2 column 8: syntax error
[info] Analyzing func.cminus
[info] Analyzing if.cminus
[info] Analyzing lex1.cminus
[info] Analyzing lex2.cminus
[info] Analyzing local-decl.cminus
[info] Analyzing math.cminus
[info] Analyzing relop.cminus
[info] Comparing...
[info] No difference! Congratulations!
```

  normal测试集：

```
(base) yj@myubuntu:~/Documents/MyDoc/HNU课程/大三上/编译原理/cminus_compiler-2021-fall$ ./tests/lab2/test_syntax.sh normal yes
[info] Analyzing array1.cminus
[info] Analyzing array2.cminus
[info] Analyzing func.cminus
[info] Analyzing gcd.cminus
[info] Analyzing if.cminus
[info] Analyzing selectionsort.cminus
[info] Analyzing tap.cminus
[info] Analyzing You_Should_Pass.cminus
[info] Comparing...
[info] No difference! Congratulations!
```

- 自己编写的测试样例

  array-expr.cminus

  decl.cminus
  empty-param.cminus
  func.cminus

  id.cminus

  local-decl.cminus
  nested-func.cminus
  var-init.cminus

```
(base) yj@myubuntu:~/Documents/MyDoc/HNU课程/大三上/编译原理/cminus_compiler-2021-fall$ ./tests/lab2/test_syntax.sh mycase
[info] Analyzing array-expr.cminus
[info] Analyzing decl.cminus
[info] Analyzing empty-param.cminus
[info] Analyzing func.cminus
[info] Analyzing id.cminus
[info] Analyzing local-decl.cminus
[info] Analyzing nested-func.cminus
[info] Analyzing var-init.cminus
```

实验反馈

通过本次实验，我有了如下收获：

1. 熟悉了语法分析的基本原理，语法分析的过程，以及语法分析中要注意的一些问题。
2. 学会了使用`bison`自动生成语法分析器的方法。
3. 对`bison`与`flex`之间是如何协同工作的有了更深一步的理解。
4. 学到了 shell 的重定向功能，对stdin、stdout有了更深一步的理解。通过灵活使用重定向，可以比较方便地完成各种各样的需求。
5. `node`函数是可变参数函数，复习了可变参数函数的定义与用法。