Lab1 实验文档

0. 实验目标

本次实验需要同学们从无到有完成一个完整的 Cminus-f 解析器，包括基于 `flex` 的词法分析器和基于 `bison` 的语法分析器。

本次实验提供了若干文档，请根据实验进度和个人需要认真阅读。

- 实验要求（本文档）
- 实验基础知识（进行试验前请首先仔细阅读）
- Flex matching
- Flex regular expressions
- 拓展阅读（选读）

在提交实验前，请务必仔细阅读本文档，确保已经完成了所有实验要求、自己的代码可以通过所有测试。

1. 词法分析器

本部分需要各位同学根据 `Cminus-f` 的词法补全 `src/parser/lexical_analyzer.l` 文件，完成词法分析器。在 `lexical_analyzer.l` 文件中，你只需补全模式和动作即可，能够输出识别出的 `token`，`text`,`line(刚出现的行数)`，`pos_start(该行开始位置)`，`pos_end(结束的位置,不包含)`。比如：

文本输入：

```
1  int a;
```

则识别结果应为：

```
1  Token         Text      Line    Column (Start,End)
2  280            int      0       (1,4)
3  284              a      0       (5,6)
4  270              ;      0       (6,7)
```

必须维护正确的：

- `token`
- `text`

选择维护的（方便你debug，测试）：

- `line`
- `pos_start`
- `pos_end`

具体的需识别token请参考基础知识。

提示：

1. 在编写本部分前，需要首先修改 `.y` 文件。具体怎么做请参见基础知识。
2. 在进入实验下一部分前，你可以使用我们提供的 `lexer` 程序进行调试。参见本文档 3.2 节。

3. token编号是自动生成的，`make`后，可在`build/syntax_analyzer.h`中找到。每次修改token后，都应该重新`make`后再进行对照。

特别说明对于部分token，我们只需要进行过滤，即只需被识别，但是不应该被输出到分析结果中。因为这些token对程序运行不起到任何作用。根据 token 定义顺序不同，输出的 token 编号也可能不同，是正常现象。

2. 语法分析器

本部分需要同学们完成`src/parser/syntax_analyzer.y`。与词法分析器相同，你只需要根据代码中的提示和基础知识中给出的文法填写相应的规则即可。

如果实现正确，该语法分析器可以从 Cminus-f 代码分析得到一颗语法树。例如输入

```
int main(void) {
    return 0;
}
```

可以得到如下语法树

```
>--+ program
|  >--+ declaration-list
|  |  >--+ declaration
|  |  |  >--+ fun-declaration
|  |  |  |  >--+ type-specifier
|  |  |  |  |  >--* int
|  |  |  |  >--* main
|  |  |  |  >--* (
|  |  |  |  >--+ params
|  |  |  |  |  >--* void
|  |  |  |  >--* )
|  |  |  |  >--+ compound-stmt
|  |  |  |  |  >--* {
|  |  |  |  |  >--+ local-declarations
|  |  |  |  |  |  >--* epsilon
|  |  |  |  |  >--+ statement-list
|  |  |  |  |  |  >--+ statement-list
|  |  |  |  |  |  |  >--* epsilon
|  |  |  |  |  |  >--+ statement
|  |  |  |  |  |  |  >--+ return-stmt
|  |  |  |  |  |  |  |  >--* return
|  |  |  |  |  |  |  |  >--+ expression
|  |  |  |  |  |  |  |  |  >--+ simple-expression
|  |  |  |  |  |  |  |  |  |  >--+ additive-expression
|  |  |  |  |  |  |  |  |  |  |  >--+ term
|  |  |  |  |  |  |  |  |  |  |  |  >--+ factor
|  |  |  |  |  |  |  |  |  |  |  |  |  >--+ integer
|  |  |  |  |  |  |  |  |  |  |  |  |  |  >--* 0
|  |  |  |  |  |  |  |  >--* ;
|  |  |  |  |  >--* }
```

这一部分必须严格遵守我们给出的语法，输出必须与标准程序输出完全一致。

2.1 思考题

本部分不计入评分，出题的本意在于想要帮助同学们加深对实验细节的理解，欢迎有兴趣和余力的同学在报告中写下你的思考答案，或者在论坛中分享出你的看法。

1. 基础知识中的计算器例子的文法中存在左递归，为什么 `bison` 可以处理？（提示：不用研究 `bison` 内部运作机制，在下面知识介绍中有提到 `bison` 的一种属性，请结合课内知识思考）
2. 请在代码层面上简述下 `yylval` 是怎么完成协同工作的。（提示：无需研究原理，只分析维护了什么数据结构，该数据结构是怎么和 `$1`、`$2` 等联系起来？）
3. 在计算器例子中，除 0 时会发生什么？如果把 `yylval` 修改为整形（`int`，`long` 等），这时候又会发生什么？
4. 能否修改计算器例子的文法，使得它支持除数0规避功能？

3. 实验要求

3.1 目录结构

与本次实验有关的文件如下。

```
.
├── CMakeLists.txt
├── Documentations
|   └── 1-parser                        本次实验的所有文档
├── Reports
|   └── 1-parser
|       └── README.md                   在这里写实验报告 <-- 修改
├── include
|   └── syntax_tree.h
├── shell.nix
├── src
|   ├── CMakeLists.txt
|   ├── common
|   |   ├── CMakeLists.txt
|   |   └── syntax_tree.c               语法树的构造
|   └── parser
|       ├── CMakeLists.txt
|       ├── lexical_analyzer.l          词法分析器          <-- 修改
|       └── syntax_analyzer.y           语法分析器          <-- 修改
└── tests
    ├── CMakeLists.txt
    └── parser
    |   ├── CMakeLists.txt
    |   ├── easy                        简单的测试
    |   ├── normal                      中等复杂度的测试
    |   ├── hard                        复杂的测试
    |   ├── lexer.c                     用于词法分析的调试程序
    |   ├── parser.c                    语法分析
    |   ├── syntree_*_std               标准程序输出结果
    |   └── test_syntax.sh              用于进行批量测试的脚本
```

3.2 编译、运行和验证

项目构建使用 `cmake` 进行。

- 编译

```
$ cd 2022fall-Compiler_CMinus
$ mkdir build
$ cd build
$ cmake ..
$ make
```

如果构建成功，会在该目录下看到 `lexer` 和 `parser` 两个可执行文件。

  - `lexer`用于词法分析，产生token stream；对于词法分析结果，我们不做考察
  - `parser`用于语法分析，产生语法树。

- 运行

```
$ cd 2022fall-Compiler_CMinus
# 词法测试
$ ./build/lexer ./tests/parser/normal/local-decl.cminus
Token         Text  Line    Column (Start,End)
280            int  0   (0,3)
284           main  0   (4,8)
272              (  0   (8,9)
282           void  0   (9,13)
273              )  0   (13,14)
...
# 语法测试
$ ./build/parser ./tests/parser/normal/local-decl.cminus
>--+ program
|  >--+ declaration-list
|  |  >--+ declaration
...
```

- 验证
可以使用 `diff` 与标准输出进行比较。

```
$ cd 2022fall-Compiler_CMinus
$ export PATH="$(realpath ./build):$PATH"
$ cd tests/parser
$ mkdir output.easy
$ parser easy/expr.cminus > output.easy/expr.cminus
$ diff output.easy/expr.cminus syntree_easy_std/expr.syntax_tree
[输出为空，代表没有区别，该测试通过]
```

我们提供了 `test_syntax.sh` 脚本进行快速批量测试。该脚本的第一个参数是 `easy` `normal` `hard` 等难度，并且有第二个可选参数，用于进行批量 `diff` 比较。脚本运行后会产生名如 `syntree_easy` 的文件夹。

```
$ ./test_syntax.sh easy
[info] Analyzing FAIL_id.cminus
error at line 1 column 6: syntax error
...
[info] Analyzing id.cminus

$ ./test_syntax.sh easy yes
...
[info] Comparing...
[info] No difference! Congratulations!
```

3.3 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` 和 `src/parser/syntax_analyzer.y`。
    - 需要在 `Reports/1-parser/README.md` 中撰写实验报告。
      - 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
  - git 提交规范
    - 不破坏目录结构（实验报告所需图片放在目录中）
    - 不上传临时文件（凡是可以自动生成的都不要上传，如 `build` 目录、测试时自动生成的输出、`.DS_Store` 等）
    - git log 言之有物
- 提交方式：
  - 代码提交：本次实验需要在希冀课程平台上发布的作业Lab1-代码提交提交自己仓库的 gitlab 链接（注：由于平台限制，请提交http协议格式的仓库链接。例：学号为 PB011001 的同学，Lab1 的实验仓库地址为`http://202.38.79.174/PB011001/2022fall-compiler_cminus.git`），我们会收集最后一次提交的评测分数，作为最终代码得分。
  - 实验评测
    - 除已提供的 easy, normal, hard 数据集之外，平台会使用额外的隐藏测试用例进行测试。
  - 报告提交：将 `Reports/1-parser/README.md` 导出成 pdf 文件单独提交到Lab1-报告提交。
  - 提交异常：如果遇到在平台上提交异常的问题，请通过邮件联系助教，助教将收取截止日期之前，学生在 gitlab 仓库最近一次 commit 内容进行评测。
- 迟交规定
  - Soft Deadline：2022-09-30 23:59:59 (UTC+8)
  - Hard Deadline：2022-10-07 23:59:59 (UTC+8)
  - 补交请邮件提醒 TA：
    - 邮箱：`zhenghy22@mail.ustc.edu.cn` 抄送 `chen16614@mail.ustc.edu.cn`
    - 邮件主题：lab1迟交-学号
    - 内容：迟交原因、最后版本commitID、迟交时间
  - 迟交分数
    - x 为相对 Soft Deadline 迟交天数，grade 满分 100

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

- 评分标准：
  实验一最终分数组成如下：
  - 平台测试得分：(70分)
  - 实验报告得分：(30分)
    注：禁止执行恶意代码，违者0分处理。
- 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。

如有任何问题，欢迎在论坛提意见进行批判指正。