编程任务选题列表

- 【已选】词法分析器，根据正则表达式生成 DFA
  - 将输入的正则表达式转化为“简化正则表达式”
  - 将每一个“简化正则表达式”转化为 NFA
  - 将一组 NFA 转化为一个 DFA，使得能依据“匹配字符串长度第一优先”、“匹配规则先后次序第二优先”进行此法分析
  - 依据 DFA 将输入的字符串转化为分段结果和分类结果。

  其中，输入的正则表达式中可能包含的语法结构有：字符的集合（如 `[a-z0-9]` ）、可选的情形（即 $?r$）、多种循环（即 r* 与 r^+）、字符串（如 `"ab\n"` ）、单字符、并集与连接；“简化正则表达式”中可能包含的语法结构有字符的集合、空字符串、星号表示的循环（即 r*）、并集与连接。本任务中，用于存储正则表达式、NFA 与 DFA 的数据结构已经在 lang.h 中提供了（详见`regexp_NFA_DFA.zip`）。

- 【已选】语法分析器，根据上下文无关语法生成语法分析器
  - 依据输入的上下文无关语法计算 first 集合与 follow 集合
  - 依据输入的上下文无关语法生成生成状态转移表
  - 基于状态转移表，处理输入终结符序列，输出移入规约过程

  其中，本任务默认输入的上下文无关语法中，每一条产生式右侧的符号串都非空；状态转移表中的每一个节点是扫描线左侧结构 NFA 的节点集合；状态转移表应描述在每个节点上遇到每个不同终结符（这个符号为此时扫描线右侧的符号）时的动作，这个动作或为移入（此时应指明移入后的状态节点）或为规约（此时应指明规约所使用的产生式）。本任务中，用于的上下文无关语法以及状态转移表的数据结构已经在 cfg.h 中提供了（详见`shift_reduce.zip`）。

- 【已选】C 语言中 struct/union/enum 的定义与声明的词法分析与语法分析

  考虑 C 语言中 struct/union/enum 的定义与声明，基于 typedef 的类型定义，以及变量的定义。下面是它们的语法（不需要考虑一条语句定义多个变量的情形，也不需要考虑变量定义同时初始化的情形）：

```
STRUCT_DEFINITION ::= struct STRUCT_NAME { FIELD_LIST } ;
   STRUCT_DECLARATION ::= struct STRUCT_NAME ;
   UNION_DEFINITION ::= union UNION_NAME { FIELD_LIST } ;
   UNION_DECLARATION ::= union UNION_NAME ;
   ENUM_DEFINITION ::= enum ENUM_NAME { ENUM_ELE_LIST } ;
   ENUM_DECLARATION ::= enum ENUM_NAME ;
   TYPE_DEFINITION ::= typedef LEFT_TYPE NAMED_RIGHT_TYPE_EXPR ;
   VAR_DEFINITION ::= LEFT_TYPE NAMED_RIGHT_TYPE_EXPR ;
```

  本任务中，约定 struct 与 union 的域列表允许为空，但 enum 的元素列表不得为空。

```
FIELD ::= LEFT_TYPE NAMED_RIGHT_TYPE_EXPR ;
   FIELD_LIST ::= FIELD FIELD ... FIELD
   ENUM_ELE_LIST ::= ENUM_ELE, ENUM_ELE, ... , ENUM_ELE
```

这里提到的 `STRUCT_NAME` 、`UNION_NAME` 、`ENUM_NAME` 、`ENUM_ELE` 以及下面会提到的 `IDENT` （标识符）都表示以字母或下滑线开头且仅包含字母数码与下划线的名字。需要特别注意的是，C 语言的中变量定义与域定义中，变量类型与域类型都是通过两部分进行描述的：左半部分是基础类型，右半部分是包含变量名或域名的一个表达式。例如，`int * x` 这个定义可以分为 `int` 与 `* x` 两个部分，它表示 `* x` 的值（即存储在 `x` 地址的内容）为整数类型。这就是上面提到的：

`LEFT_TYPE` `NAMED_RIGHT_TYPE_EXPR`

本任务中需要考虑指针类型、数组类型、函数类型的情形，在基础类型方面只考虑 `int` 与 `char` 两个类型：

```
LEFT_TYPE ::= struct STRUCT_NAME { FIELD_LIST }
              | struct { FIELD_LIST }
              | struct STRUCT_NAME
              | union UNION_NAME { FIELD_LIST }
              | union { FIELD_LIST }
              | union UNION_NAME
              | enum ENUM_NAME { ENUM_ELE_LIST }
              | enum { ENUM_ELE_LIST }
              | enum ENUM_NAME
              | int | char | IDENT
```

```
NAMED_RIGHT_TYPE_EXPR ::= IDENT
                          | * NAMED_RIGHT_TYPE_EXPR
                          | NAMED_RIGHT_TYPE_EXPR [ NAT ]
                          | NAMED_RIGHT_TYPE_EXPR ( ARGUMENT_TYPE_LIST )
   ANNON_RIGHT_TYPE_EXPR ::= EMPTY
                          | * ANNON_RIGHT_TYPE_EXPR
                          | ANNON_RIGHT_TYPE_EXPR [ NAT ]
                          | ANNON_RIGHT_TYPE_EXPR ( ARGUMENT_TYPE_LIST )
   ARGUMENT_TYPE ::= LEFT_TYPE ANNON_RIGHT_TYPE_EXPR
   ARGUMENT_TYPE_LIST ::= ARGUMENT_TYPE, ..., ARGUMENT_TYPE
```

在 C 语言中，表达式（这里提到的 `NAMED_RIGHT_TYPE_EXPR` 与 `ANNON_RIGHT_TYPE_EXPR` ）后缀（数组与函数）的结合优先级高于前缀的结合优先级，并且允许使用圆括号改变优先级。例如，`int * x[10]` 表示 `int * (x[10])` ，定义了一个整数指针的数组；函数参数类型的语法也是类似的，例如 `int * [10]` 表示整数指针的数组类型，`int ( * )(int)` 表示整数一元函数的函数指针类型。本题约定，函数的参数类型列表可以为空。

本任务中，用于所有定义与声明的抽象语法树的 C 语言存储结构以及辅助构造函数、调试函数已经在 lang.h 与 lang.c 中提供了，main.c 程序也是固定的（详见 `struct_union_enum.zip`），请使用 flex 与 bison 实现词法分析与语法分析。

- 【已选】带结构化宏的程序语言的词法分析、语法分析与宏展开

  这个任务中，你需要在 WhileDB 语言中加入函数过程调用与结构化的宏。我们称一个宏是结构化的，如果宏参数的语法树在宏展开之后不会被破坏。一个结构化的宏可能是一个表达式宏也可能是一个程序语句宏。具体而言，你需要

  – 在不展开宏、将结构化的宏也当作特定语法结构的前提下，完成词法分析、语法分析，并能输出语法树用于调试

  - 在上述语法树上实现宏展开，并能够输出展开后的语法树。

- 【已选】带数组与字符串类型的程序语言的词法分析、语法分析与解释执行

  这个任务中，你需要在 WhileDB 语言中加入数组与字符串，并

  - 支持变量声明的同时初始化，包括对于数组的初始化
  - 支持字符串常量
  - 允许一条变量声明语句中同时声明多个变量
  - 基于小步语义实现解释执行。

- 支持函数调用的程序语言的词法分析、语法分析与解释执行

  这个任务中，你需要在 WhileDB 语言中加入函数过程调用，并基于小步语义实现解释执行。