# Exercise Sheet

2022.10.08

1. 请采用下面两种方法对包含加法、乘法、括号、自然数常数与变量名表达式进行词法分析与语法分析，其中变量名为长度大于 0 不超过 20、首字母为字母或下划线且只包含字母、数码与下划线的字符串，自然数常数是没有额外前导零的 0 到 99999 之间（含）的整数。

   - 采用以下无歧义的上下文无关语法。

     ```
     E -> F      E -> E + F      F -> F * G
     F -> G      G -> ( E )      G -> ID      G -> NAT
     ```

   - 采用以下上下文无关语法，并引入优先级与结合性规定解决移入/规约冲突。

     ```
     E -> E + E      E -> ID      E -> ( E )
     E -> E * E      E -> NAT
     ```

   提交时请创建两个文件夹 simple_arith_a 与 simple_arith_b，每个文件夹包含 lang.l 与 lang.y 两个文件。编程时可以使用附件文件夹 simple_arith 中提供的 lang.h, lang.c, main.c 与 Makefile 进行编译测试。在 Canvas 系统上提交时，请将所有需要提交的文件夹用 zip 压缩后提交。

2. 请使用 Flex 词法分析器与 Bison 语法分析器写一个带断言标注的简单 C 语言语法分析器。提交时请在 SAC 文件夹中放置你的所有源代码、Makefile 以及必要的说明文件。在 Canvas 系统上提交时，请将所有需要提交的文件夹用 zip 压缩后提交。这个编程任务的具体要求如下（也可以参考附件中的 SAC/lang.h）：

   - 输入的 C 程序中可以包含 struct、union 与 enum 的声明与定义，函数的声明与定义，全局变量的定义，以及断言语言谓词的定义标注。简单 C 语言程序中不会包含 typedef，也不会包含 `#define`，`#include`，`#ifdef` 等宏。
   - 简单 C 语言程序中的 struct、union、enum 定义只会出现一些简单的情形。
     - 简单 C 语言中，struct 与 union 的每个域都必须单独定义并且以分号结束。例如，下面的 struct 都定义不会出现在简单 C 语言中：

       ```
       struct slist2 {
         int x, y;
         struct slist2 * link;
       };
       ```

       ```
       struct slist2 {
         int x;
         int y;
         struct slist2 * link
       };
       ```

       而下面 struct 定义可以出现在简单 C 语言程序中：

       ```
       struct slist2 { int x; int y; struct slist2 * link; };
       ```

  - 简单 C 语言程序中 enum 定义中不会指明值与整数之间的对应关系，例如下面 enum 定义不会出现：

```
enum kind { T_EXPR = 0, T_CMD, T_IDENT };
```

  - 简单 C 语言程序中没有 bitfield。
  - 简单 C 语言程序中可能既有 struct、union 与 enum 的声明也有它们的定义。
- 除了上述关于 struct/union/enum 的约定之外，我们额外假定在简单 C 语言中只会如下使用 C 程序类型：
  - 简单 C 语言程序中，只有确定长度的数组，没有不定长度的数组。
  - 简单 C 语言程序中，在声明变量类型、函数返回值类型、函数参数类型以及 struct/union 的域类型时，不能省略 `struct`，`union` 以及 `enum` 这三个保留字。
  - 简单 C 语言程序中，类型中不会出现 `const`，`volatile`。
  - 简单 C 语言程序中，没有浮点数类型 float, single 与 double。
  - 简单 C 语言程序中，没有函数类型或函数指针类型（但是有函数）。
- 我们假设简单 C 语言程序中的变量定义（包括全局变量、局部变量）与 struct 或 union 的域不会出现下面复杂情况：
  - 简单 C 语言程序中，不能在定义 struct、union 或 enum 的同时定义相关类型的变量，也不能在定义 struct、union 或 enum 的同时将 struct 或 union 的域定义为相关类型。例如，下面 struct 定义不会出现在简单 C 语言程序中：

```
struct expr {
  int kind;
  union { int a; char * b; } value;
};
```

```
struct pair { int a; int b; } p1;
```

  - 简单 C 语言程序中，变量定义的同时不能进行初始化，即不会出现 `int i = 0;` 这样的语句。
  - 简单 C 语言程序中，不会同时定义多个变量，例如 `int x, y;` 会被写为 `int x; int y;`；类似的，struct 与 union 的定义中也不会同时定义多个域。
  - 简单 C 语言中，struct 与 union 的域以及变量定义都不会有 attribute。
- 简单 C 语言的程序语句中可以出现局部变量声明，赋值语句、条件分支语句、循环语句、控制流语句与函数调用。
  - 赋值语句中不允许出现多于一个写入操作（函数调用除外），即，在简单 C 语言中只能出现形如 `E = E`，`E += E`，`E -= E`，...，`E ++`，`E --`，`++ E`，`-- E` 的赋值语句，其中 `E` 表示不包含写入操作（函数调用除外）的表达式，表达式中的强制类型转换一律使用语法 `( T ) E`，其中 `T` 是 C 类型、`E` 是表达式。因此，以下都不是简单 C 语言中的语句：`x = (++ y) + (++ y)`，`a[i] = (b[i] += c[i])`，`a[i] = b[i] = c[i]`。以下都是简单 C 语言中的语句：`a[i] = b[i]`，`x += y`，`++ z`，`d = ( char * ) f(x,g(y),z)`，`*f(a) = g(b)`，`f(x) = g(x) + h(x)`。
  - 表达式中可能出现十进制整数常量，不会出现浮点数常量、字符常量或字符串常量。
  - 表达式中可以使用 sizeof，但是我们约定 sizeof 只能用于计算类型的大小，不能用于计算变量（或某个左值）占据的空间大小，并且 sizeof 只能用于计算基础类型（各种整数类型）、struct/union/enum 类型以及他们的指针类型（包括高阶指针类型）占用的空间大小。

  - 条件分支语句包括 if 语句（可能有也可能没有 else 分支）与 switch 语句（有 case 分支与 default 分支）。
  - 循环语句包括 for 语句、while 语句与 do-while 语句。
  - 我们假设简单 C 语言程序中，for 语句的初始化步骤中不能同时声明变量，例如，下面 C 语句

```
for (int i = 0; i < n; ++ i) { s += a[i]; }
```

    在简单 C 语言程序中应当写为：

```
int i;
for (i = 0; i < n; ++ i) { s += a[i]; }
```

  - 控制流语句包括 break、continue 与 return 语句，return 语句可以有返回值也可以没有返回值。
  - 函数调用可以在表达式中出现，没有返回值的函数调用也是单独的程序语句。
  - 我们假设程序中不会包含 goto 语句，也不会出现供 goto 语句使用的 label。

- 除了 C 程序代码之外，源程序中可能包含一些关键性的标注，这些关键性标注会使用 `//@ ...` 表示单行标注，或使用 `/*@ ... */` 表示多行标注（也可能只有一行）。这些关键标注包括：函数规范标注、断言标注与断言语言的谓词定义标注。你编写的词法分析器与语法分析器应当处理这些关键标注。在这个编程任务中，你可以假设程序源代码中不会出现其他注释。
- 我们约定 `With`，`Require`，`Ensure`，`__return`，`Inv`，`forall`，`exists`，`Let` 都是额外的保留字，不能用作变量名、函数名或类型名。
- 断言标注是出现在程序语句中用于表达断言的标注。关于断言标注可以出现的位置以及断言的语法我们做如下约定：
  - 断言标注只能出现在语句之间，不能出现在赋值语句或函数调用的内部。
  - 如果要在 if 条件分支的分支中或循环语句的循环体中添加断言标注，应当保证所在分支由一组大括号约束，或应当保证所在的循环体由一组大括号约束。例如，下述断言标注的位置是合法的：

```
for (i = 0; i < n; ++ i) {
  //@ s >= 0
  s += a[i];
}
```

    但是下述断言标注是非法的：

```
for (i = 0; i < n; ++ i)
  //@ s >= 0
  s += a[i];
```

    而下述断言标注虽然是合法的，但是其中的断言会被视为整个循环之后的断言，而非循环体内的断言：

```
for (i = 0; i < n; ++ i)
  s += a[i];
  //@ s >= 0
```

  - 在循环前，可以添加循环不变量型断言标注（也可以不添加），需要用关键字 `Inv` 将这类断言标注与普通断言标注相区分，例如：

```
//@ Inv s >= 0
for (i = 0; i < n; ++ i)
  s += a[i];
```

- 断言的语法与 C 表达式（特别是布尔类型表达式）的语法基本相同，但有两点区别：(1) 断言中可以出现 `forall` 与 `exists` 这两个逻辑量词表示『任意』与『存在』，其语法如下面例子所示：

```
//@ Inv forall j, j < 0 || a[j] == 0 || i <= j
for (i = 0; i < n; ++ i)
  s += a[i];
```

`forall` 与 `exists` 的优先级低于所有逻辑连接词。(2) 断言中不能出现 sizeof。

- 断言语言的谓词定义标注。我们约定，C 程序中可以包含断言语言的谓词定义标注，但是这些标注不能出现在 struct、union、enum 或函数的定义或声明的内部。下面是一个谓词定义标注的例子：

```
/*@
  Let listrep(l) = l == 0 || exists t, (l -> tail == t) && listrep(t)
*/
```

- 在 C 语言程序中，可能出现函数声明与函数定义，我们约定：
  - 函数的返回值与参数不能是 struct 类型、union 类型或数组类型，但可以是它们的指针类型。
  - 函数声明中不能省略参数名称而只罗列参数类型。
  - 函数声明与函数定义都可以附带函数规约标准，函数规约标注出现的位置如下面例子所示：

```
struct list * reverse(struct list * p)
/*@ Require listrep(p)
    Ensure  listrep(__return)
*/;
```

```
struct list * reverse(struct list * p)
/*@ Require listrep(p)
    Ensure  listrep(__return)
*/
{
  struct list * w;
  struct list * t;
  struct list * v;
  w = 0;
  v = p;
  // Inv listrep(v) && listrep(w)
  while (v) {
    t = v -> tail;
    v -> tail = w;
    w = v;
    v = t;
  }
  return w;
}
```

– 函数规约标注可以由 Require-Ensure 两部分构成，也可以由 With-Require-Ensure 三部分构成。Require 与 Ensure 之后各应该有一个断言，With 之后是一个非空的辅助数学变量列表。
– 函数规约标注可以由 Require-Ensure 两部分构成，也可以由 With-Require-Ensure 三部分构成。Require 与 Ensure 之后各应该有一个断言，With 之后是一个非空的辅助数学变量列表 (With 有『任意』的意思)，例如

```
void swap(int * px, int * py)
/*@ With x y
    Require (px != py) && store_i32(px, x) && store_i32(py, y)
    Ensure  store_i32(px, y) && store_i32(py, x)
*/;
```