实验 5 消除左递归（有替换）

实验难度：★★★★☆
建议学时：4 学时

一、 实验目的

- 了解在上下文无关文法中的左递归的概念。
- 掌握直接左递归、一般左递归的消除算法。

二、 预备知识

- 在这个实验中用到了单链表插入和删除操作。如果读者对这一部分知识有遗忘，可以复习一下数据结构中的相关内容。
- 理解指针的指针的概念和用法。在这个实验中，指针的指针被用来确定单链表插入和删除的位置，以及用来完成插入和删除的具体操作。
- 理解左递归和右递归的含义以及左递归的各种形式，如：简单直接左递归、普遍的直接左递归、一般的左递归。

三、 实验内容

3.1 阅读实验源代码

RemoveLeftRecursion.h 文件

主要定义了与文法相关的数据结构，这些数据结构定义了文法的单链表存储形式。其中 Rule 结构体用于定义文法的名称和文法链表，RuleSymbol 结构体用于定义文法产生式中的终结符和非终结符。具体内容可参见下面两个表格。

Rule 的域	说明
RuleName	文法的名称
pFirstSymbol	指向文法的第一个 Select 的第一个 Symbol
pNextRule	指向下一条文法

RuleSymbol 的域	说明
pNextSymbol	指向下一个 Symbol
pOther	指向下一个 Select
isToken	是否为终结符。1 表示终结符，0 表示非终结符
TokenName	终结符的名称。isToken 为 1 时这个域有效
pRule	指向 Symbol 对应的 Rule。isToken 为 0 时这个域有效

下面是一个简单文法，并使用图表说明了该文法的存储结构：

```
A -> Aa | aB
B -> bB
```

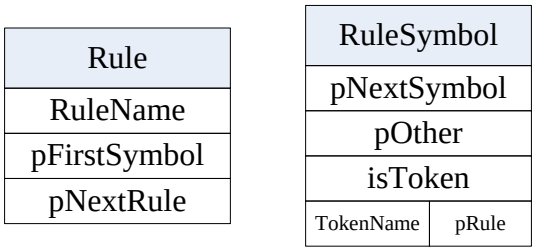

图 5-1：Rule 和 RuleSymbol 结构体图例

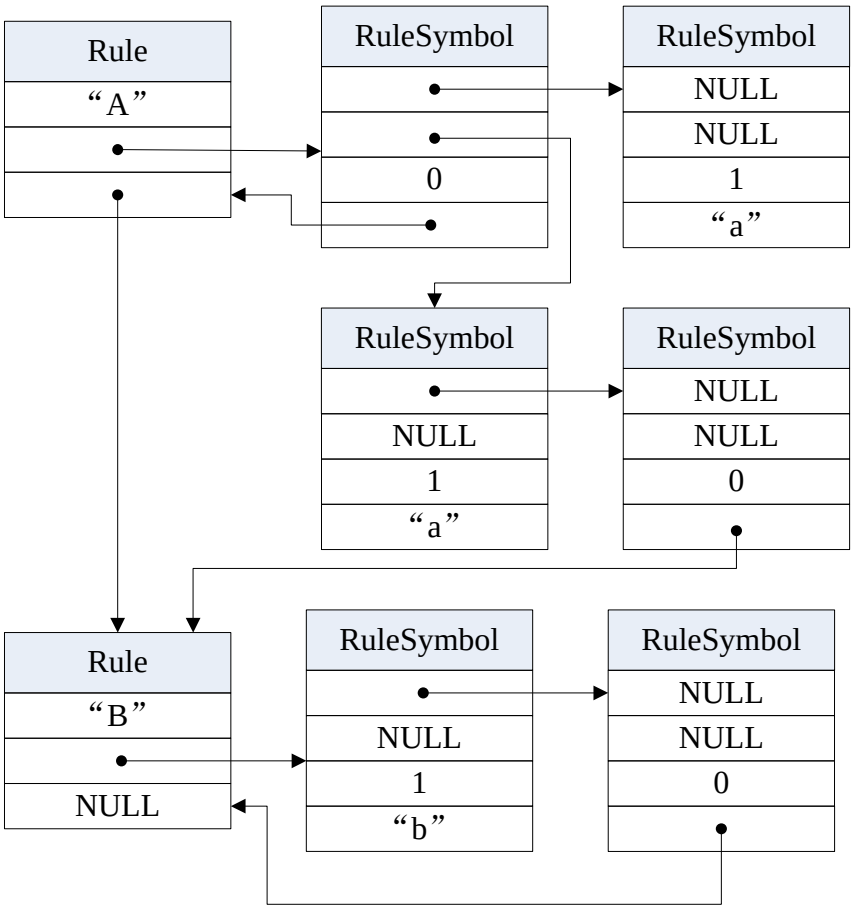

图 5-2：简单文法的存储结构

main.c 文件

定义了 main 函数。在 main 函数中首先调用 InitRules 函数初始化了文法，然后调用了 PrintRule 函数，打印消除左递归之前的文法，接着调用 RemoveLeftRecursion 函数对文法消除左递归，最后再次调用了 PrintRule 函数打印消除左递归之后的文法。

在 main 函数的后面，定义了一系列函数，有关这些函数的具体内容参见下面的表格。关于这些函数的参数和返回值，可以参见其注释。

函数名	功能说明
SymbolNeedReplace	判断当前 Rule 中的一个 Symbol 是否需要被替换。如果 Symbol 是一个非终结符，且 Symbol 对应的 Rule 在当前 Rule 之前，就需要被替换。此函数的函数体还不完整，留给读者完成。

CopySymbol	拷贝一个 Symbol。此函数的函数体还不完整，留给读者完成。
CopySelect	拷贝一个Select。在此函数中可以调用CopySymbol函数，将Select中的Symbol逐个进行拷贝。此函数的函数体还不完整，留给读者完成。
ReplaceSelect	替换一个 Select 的第一个 Symbol。 **提示：** ● 在调用此函数之前，已经调用了SymbolNeedReplace函数，确保Select的第一个Symbol需要被替换。 ● Select的第一个Symbol是一个非终结符，该Symbol对应的Rule就是用于替换该Symbol模板。 ● 当需要拷贝Select时，可以调用CopySelect函数。 此函数的函数体还不完整，留给读者完成。
FreeSelect	释放一个 Select 的内存。此函数的函数体还不完整，留给读者完成。
RuleHasLeftRecursion	判断一条 Rule 是否存在左递归。 **提示：** ● 只有一条Rule中的一个Select存在左递归，那么这条Rule就存在左递归。 ● 如果一个Select的第一个符号（非终结符）对应的就是此条文法，这个Select就存在左递归。 此函数的函数体还不完整，留给读者完成。
AddSymbolToSelect	将一个 Symbol 添加到 Select 的末尾。此函数的函数体还不完整，留给读者完成。
AddSelectToRule	将一个 Select 加入到文法末尾，当 Select 为 NULL 时就将一个ε终结符加入到文法末尾。在本程序中ε可以用 $ 来代替。此函数的函数体还不完整，留给读者完成。
RemoveLeftRecursion	对文法消除左递归。 **提示：** ● 在本函数中包含了替换算法，从而可以处理间接左递归的情况。而且由于替换后还可能需要替换，所以设置了一个标识isChange，并初始化为0，每当发生替换之后就将该标识赋值为1，并将此标识作为循环条件，直到没有替换发生时，才能结束替换。 ● 在本函数中使用指针的指针pSelectPtr来确定符号在单链表中插入和删除的位置，在进入下一次循环之前应为pSelectPtr设置正确的值。 ● 在每处理完一条文法后，会将文法链表的游标指向新文法，这样在继续执行for循环的移动游标操作后，就会跳过这条新文法，从而继续对后面的文法进行消除左递归操作（新文法不包含左递归，所以并不需要对这条文法进行处理）。 此函数的函数体还不完整，留给读者完成。
InitRules	使用给定的数据初始化文法链表。
CreateRule	创建一个新的 Rule。
CreateSymbol	创建一个新的 Symbol。

FindRule	根据 RuleName 在文法链表中查找名字相同的文法。
PrintRule	输出文法。此函数的函数体还不完整，留给读者完成。

3.2 为函数 InitRules 添加注释

在 InitRules 函数之前定义了两个结构体,这两个结构体用来定义初始化文法数据的存储形式。具体内容可参见下面两个表格。

SYMBOL 的域	说明
isToken	是否为终结符。1 表示终结符，0 表示非终结符。
Name	终结符和非终结符的名称。

RULE_ENTRY 的域	说明
RuleName	文法的名称。
Selects	SYMBOL 结构体的二维数组，其中每一行表示一个 Select，一行中的每个元素分别表示一个终结符或非终结符。

为 InitRules 函数添加注释。(注意，在本程序中使用了指针的指针，体会其在单链表的插入和删除操作中的作用)。

3.3 为 PrintRule 函数编写源代码

为 PrintRule 函数编写源代码，同时理解在本程序中文法的链式存储结构，编写完源代码后，选择“Run”菜单中的“Run Without Debugging”，会在“TERMINAL”窗口输出文法的产生式。由于还没有为 RemoveLeftRecursion 函数和其他未完成的函数编写源代码，所以前后两次输出的文法是一样的。

3.4 编写源代码并通过验证

按照下面的步骤继续实验：

1. 为 RemoveLeftRecursion 函数和其他未完成的函数编写源代码。注意尽量使用已定义的局部变量。
2. 按 Ctrl+Shift+B,在弹出的下拉列表中选择“生成项目”。如果生成失败，根据“TERMINAL”窗口中的提示信息修改源代码中的语法错误。
3. 按 Ctrl+Shift+B，然后在下拉列表中选择“测试”，启动验证。在“TERMINAL”窗口中会显示验证的结果。如果验证失败，可以在“EXPLORER”窗口中，右击“result_comparation.html”文件，在弹出的菜单中选择“Open Preview”，可以查看用于答案结果文件与读者编写程序产生的结果文件的不同之处,从而准确定位导致验证失败的原因。

四、思考与练习

1. 请读者仔细检查自己编写的源代码,查看在消除左递归的过程中,是否调用了 free 函数释放了从文法链表中移除的 Symbol,是否调用了 FreeSelect 函数释放了从文法链表中移除的 Select，否则会造成内存泄露。
2. 编写一个 FreeRule 函数，当在 main 函数的最后调用此函数时，可以将整个文法的内存释放掉，从而避免内存泄露。
3. 使用自己编写的代码对下面两个例子进行验证,确保程序可以为所有形式的文法消除左递归，并验证通过（文法中的终结符用粗体表示）。

   例 1： A -> B**a** | A**a** | **c**

   B -> B**b** | A**b** | D

D -> Ad

例 2： exp -> exp addop term| term

addop -> + | -

term -> term mulop factor | factor

mulop -> *

factor -> (exp) | number