编译原理

绪论

语言分类：

按理论基础：过程式（命令式、强制式）--冯诺依曼体系结构；函数式（功能）；逻辑式（条件）--数理逻辑谓词运算；对象式--抽象数据结构

四代语言：机器（二进制）；汇编（机器语言符号化）；高级（告诉机器“做什么”和“怎么做”）；超高级（只告诉“做什么”，如SQL）；函数式、对象式和逻辑式

绑定：一个实体（对象）与某种属性建立某种联系的过程

变量的属性

- 作用域（大多是静态）
- 生存期，分配，C、C++是动态分配，可以隐式（进入变量的作用域时自动获得）或者显式（new）分配
- 值（动态）
- 类型

编译之前或编译时能确定的属性称为静态，运行时才能确定的属性称为动态

动态绑定的语言一般采用解释，静态一般采用编译

虚拟机：由实际机器加软件实现的机器，有自己的机器语言，比如字节码就可以看出是java虚拟机的机器语言

程序单元：在执行过程中独立的调用单位，被调用、运行的过程叫激活

运行时，单元=代码段+活动记录，是单元实例

活动记录要在运行过程中生成和释放

引用环境：一个程序单元可以引用的局部和非局部变量

语言设计过程：首先是适用于特殊功能的语言，然后是集成

数据类型

（用户自定义类型、概念、举例，5分简答题，后面的记概念，不用画图）

数据类型：对存储器中数据的抽象，定义了一组值的集合和一组操作

内部类型

用户定义类型（也是聚合）

- 笛卡尔积--记录、结构
- 有限映射--数组
- 序列（由任意多个类型相同的数据项构成）--字符串、顺序文件
- 递归--各类树、指针

- 判定或（二选一）--C语言 union，多个变量共用一个c存储空间，空间长度为最大变量的长度，修改一个变量就会影响到其他的；Pascal变体记录，含一个boolean变量，根据它判断后半部分的类型

```
struct{
  char name[20];
  int age;
  char sex;
  // 针对教室和学生选择填写成绩或者科目，二选一
  union{
      float score;
      char course[20];
  } sc;
} bodys[TOTAL];
```

- 幂集--总集合的任意子集

结构变量指的就是聚合类型，非结构化就是基本类型

枚举是非结构类型，是规定枚举类型可能的值，以及它们的顺序，然后具体的变量只能取一个值

抽象数据类型

内部类型和用户自定义类型是两个层次的抽象，抽象数据类型是以它们为基本表示的 更高层次的抽象，具有信息隐蔽、封装、继承等特性，基本表示不可见，比如C++、java的类

类型检查：数据对象的类型和使用的操作是否匹配

java、c++、python是强类型语言，类型检查全部在编译时完成（静态）

类型转换：收缩、拓展

两个变量能相互赋值则称为相容

两个变量的类型名相同为名字等价

两个变量的类型具有相同的结构称为结构等价

实现模型

描述符->存储区（数据对象）

Pascal：描述符记录变量名、类型、指向存储区的指针，数组还要记录数组名、成分类型、下标类型、上下界、单元个数

实现二维数组或者结构体中的数组就用描述符里的一个单元指向另一个描述数组的描述符

控制结构

（选填题）

注意流程图的Y和N不能漏写

语句级

顺序、选择、循环

单元级

- 显式调用

  实参与形参的绑定：位置绑定、关键词绑定

  副作用： 对绑定的非局部变量（绑定在别的程序单元的活动记录中）进行修改时，将产生副作用

  eg: w=f(x,y)+z，降低可读性；破坏运算律；影响编译优化

  别名：单元激活期间，两个变量共享同一数据对象，别名会造成swap等程序出问题
- 异常处理（即隐式调用）
- 协同程序：单元之间彼此显示地激活（通过命令），交错执行
- 并发单元：不存在调用和返回的概念

程序语言的设计

语言

语言由语法和语义组成

语法包括词法规则（什么样的字符串可以构成有效符号，eg. 标识符规则）和各语法单位的形成规则

描述语言的方法：生成（文法）；识别（识别图）

识别图：圆形 终结符；方框 非终结符；串接并接；定义一个非终结符的语法图只能有一个出口和一个入口；回溯

下图不是很严谨，α表示非终结符，应该写成N更好，所以最后一定是会被换成γ的，最后生成的是$\gamma\beta^n$

语义规定语法单位的含义和正确性

还没有被普遍接受的工具，常用自然语言描述

GAM抽象机

文法

文法是语法的形式描述，根据文法产生语言

BNF：G=(N, T, S, P) N 非终结符 < > ； T 终结符；P 规则，产生式的非空有限集合；S 开始符号，属于非终结符（规则的左半边，加<>）

产生式规则 ->，推导过程 =>

大写 代替尖角符号，表示非终结符；小写 表示终结符；希腊字母表示两者的集合（或）；ε表示空符号串

左递归：A->Aβ；右递归：A->βA

0型文法（短语结构文法）

α->β，产生式左端至少有一个非终结符（产生空字符串除外）

表达的能力相当于图灵机，是最泛的

递归可枚举

1型文法（上下文相关文法）

|α|<=|β|

等价定义：αAβ->αωβ，其中A为非终结符

每次一定有一个非终结符产生一个以上的终结符

2型文法（上下文无关文法）

A->β，左边仅有一个非终结符，β∈N∪T，所以不受上下文限制

3型文法（正则文法）

右线性文法：A->αB或A->α，α∈T，B可以等于A，即可以表示递归

左线性即A->Bα，针对每个右线性文法，一定能找到与之等价的左线性文法

文法等价就是能产生相同的语言

推导 => 逆过程称为归约，由一步得到称为直接；=>+，推导一步以上，=>0，经过0步推导，即相等，=>* 经过任意步推导

最右推导（规范推导）：每次被替换的非终结符均是最右边的，逆过程称为最左归约

为编译器设置固定的推导规则：每次换哪个？怎么换？什么时候结束？

最右推导（规范推导）

句型：由开始符推导出来的符号串，开始本身也是一个句型，是由0步推导出的；句子：最终生成的，只含终结符；语言：由所有**句子**组成的集合

T：文法与语言

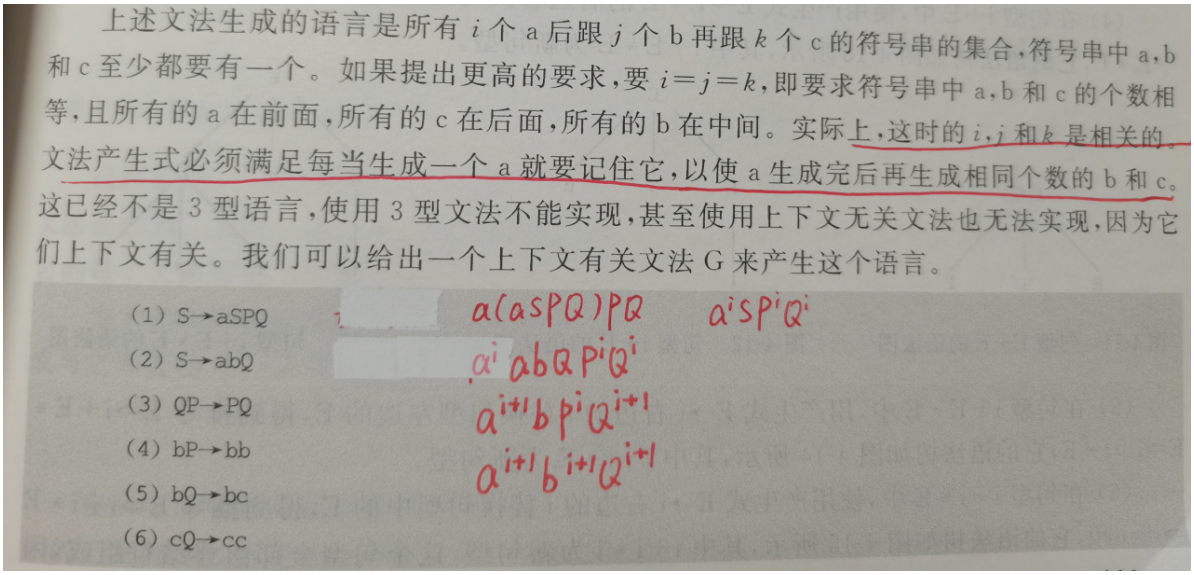
推导语言

多迭代几次找规律

逆推文法

次数一样的拆成一坨

把m>=k，把k代入进去检查一下


上述文法生成的语言是所有 i 个 a 后跟 j 个 b 再跟 k 个 c 的符号串的集合，符号串中 a，b 和 c 至少都要有一个。如果提出更高的要求，要 $i=j=k$，即要求符号串中 a，b 和 c 的个数相等，且所有的 a 在前面，所有的 c 在后面，所有的 b 在中间。实际上，这时的 i，j 和 k 是相关的。文法产生式必须满足每当生成一个 a 就要记住它，以使 a 生成完后再生成相同个数的 b 和 c。这已经不是 3 型语言，使用 3 型文法不能实现，甚至使用上下文无关文法也无法实现，因为它们上下文有关。我们可以给出一个上下文有关文法 G 来产生这个语言。

(1) S→aSPQ
(2) S→abQ
(3) QP→PQ
(4) bP→bb
(5) bQ→bc
(6) cQ→cc

$a(aSPQ)PQ$ $a^iSP^iQ^i$
$a^iabQP^iQ^i$
$a^{i+1}bP^iQ^{i+1}$
$a^{i+1}b^{i+1}Q^{i+1}$

语法树

二义文法：一个文法存在**某个句子**有多于一个的语法树

二义文法产生的语言不一定有二义性，一个有二义性的文法和一个没有的文法可能产生相同的语言。存在先天二义语言；即，每个产生它的文法都是二义的

T：（12分大题）

给出文法

求规范推导；

画出语法树：证明XX是该文法句型（看终结符）；判断和证明该文法有无二义性

求短语、直接短语、句柄、最左素短语；

！！语法树没有箭头，树的一个结点上只能画一个符号，即保持2型文法（不管运算律）

语言设计

表达式的设计：逻辑、关系、算术

设计算术表达式的时候要考虑运算顺序

*比＋优先，要放在+的下一层，从下往上的时候先被识别

<算术表达式>→<算术表达式>+<项>|<算术表达式>-<项>|-<项>|<项>
<项>→<因子>|<项>*<因子>|<项>/<因子>
<因子>→(<算术表达式>)|<常量>|<变量>
<变量>→<标识符>
<常量>（略）

语句的设计：说明、执行（复制、控制、复合/语句块）

程序单元的设计

程序的设计

注意区分<>表示内部是非终结符，其他的符号无特殊说明的话是终结符

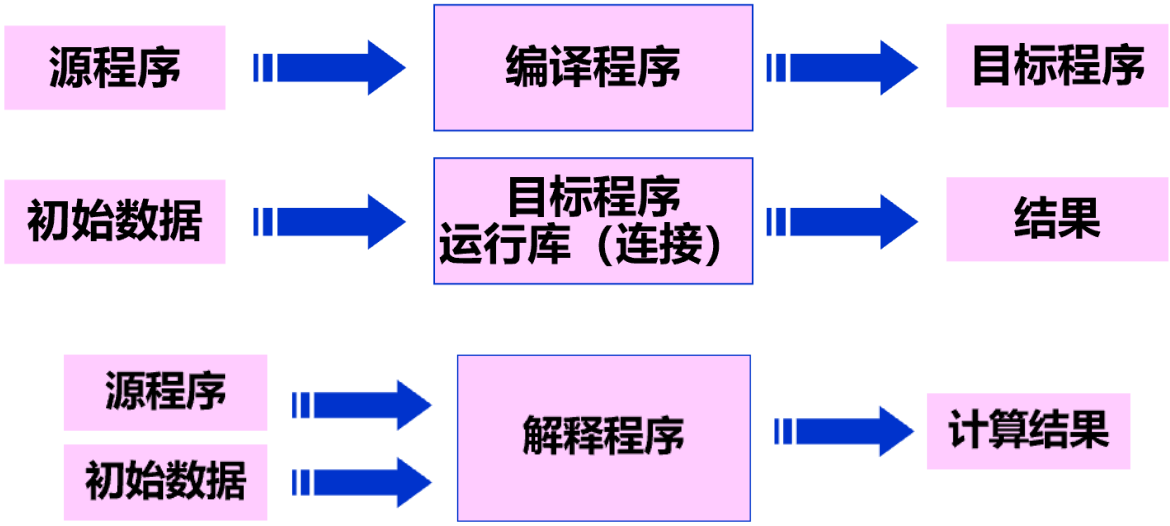
编译概述

翻译：把一种语言转换成完全等价的另一种语言

编译（compile）是把高级语言翻译成低级语言的过程，汇编是把低级语言（如汇编语言）翻译成机器码的过程

解释（interpret）：在执行期，动态将代码逐句解释为机器代码，或是已经预先编译为机器代码的子程序，之后再执行。缺点是重复执行需要重复翻译，花费的时间更多，优点是适合动态语言和交互式环境

java、python都是先编译后解释的语言，java是先编译成字节码（javac），然后在虚拟机上用java解释器执行（java），python是编译成PyCodeObject，对于有重用意义的代码（import模块），会将PyCodeObject写入.pyc文件保存，这样下次就可以根据.pyc直接建立PyCodeObject，不用再次编译



宿主语言：编写编译程序的语言；源语言、目标语言与宿主语言一般不同

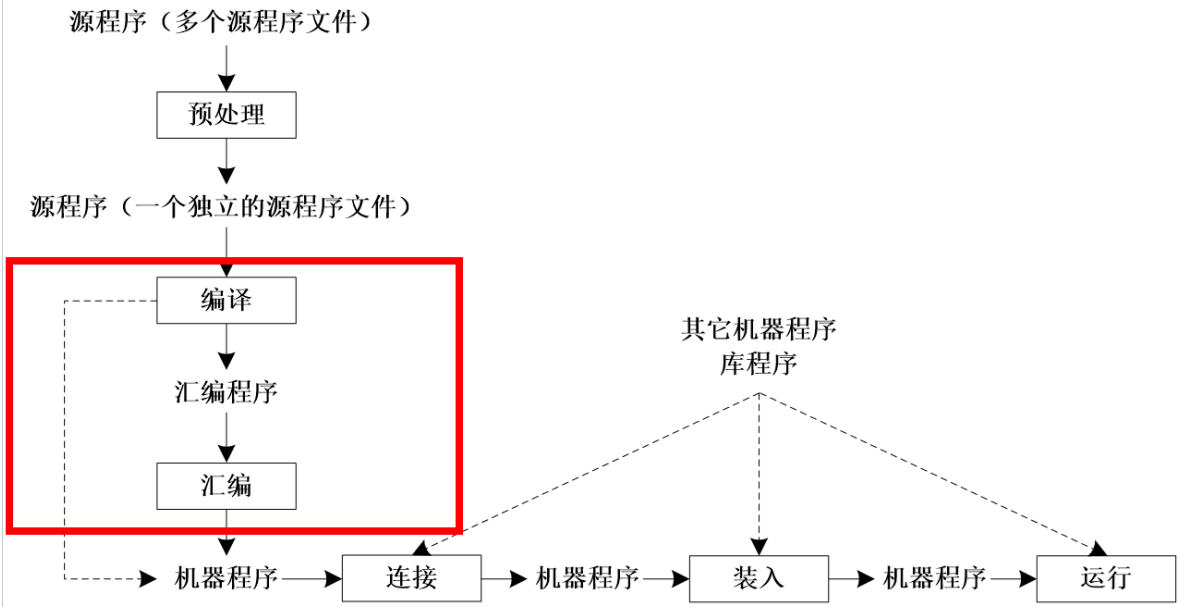
自驻留：编译程序能生成可供宿主机执行的机器代码

交叉编译：生成非宿主机的机器代码

自编译：编译程序是用源语言写的，如C语言编译器

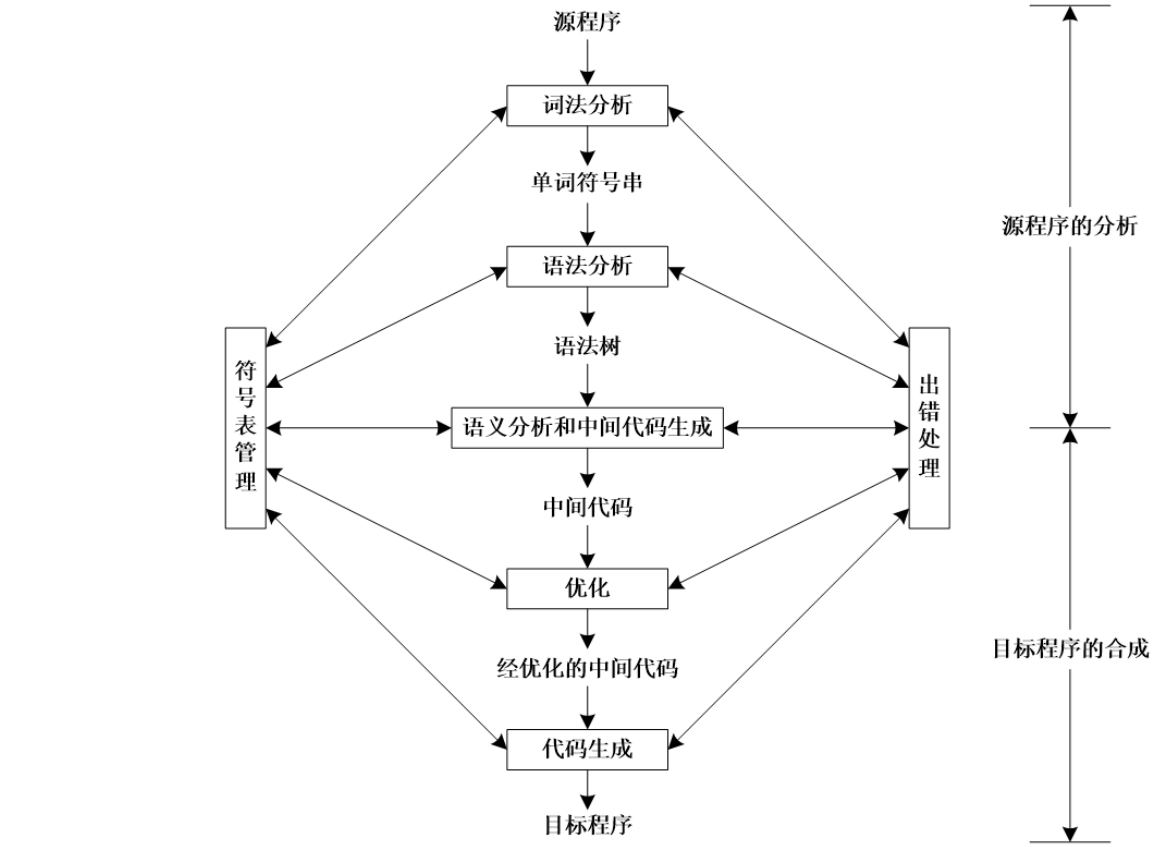
编译步骤（5+2 简答题）

完整的程序处理过程



注意是先汇编再连接（库程序也经过了编译和汇编）

编译步骤



词法分析

从左到右逐个扫描字符串，按照词法规则，识别出单词符号作为输出，如果出错则输出错误信息

语言的符号：基本（关键/保留）字、标识符、常数、运算符、界符

输出：（单词类别，单词的属性），两元都是编码

❑ **种别编码的一般方式**

- ✓**对于界符和运算符，一符一种，即一个符号对应一个编码；**
- ✓**标识符作为单独的一种，用自身的值区别不同的标识符；**
- ✓**对于常数，按它的类型来编码，如整型、实型、布尔型和字符型各为一种，按自身的值来区别不同的常数；**
- ✓**基本字可分为一种，也可一符一种**

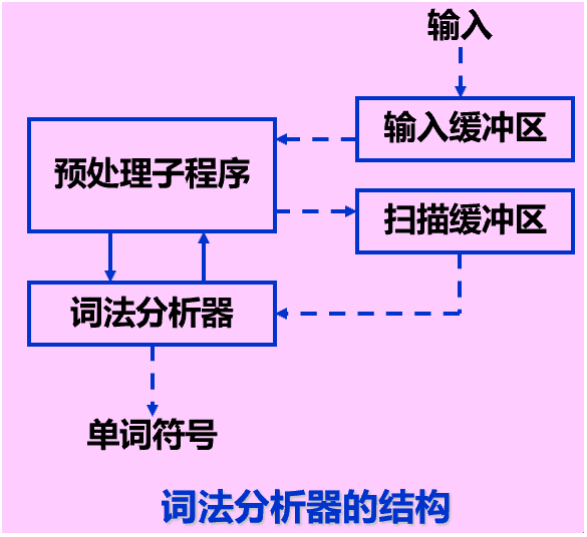
一符一种或者一类一种，将一类作为一种的话就通过属性来区分

词法分析器设计

扫描缓冲区的结构

双指针：起始指针+搜索指针，搜索指针识别到空白符等不属于当前符号的字符就回退一格，输出两个指针之间的内容

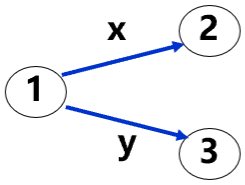
双缓冲区



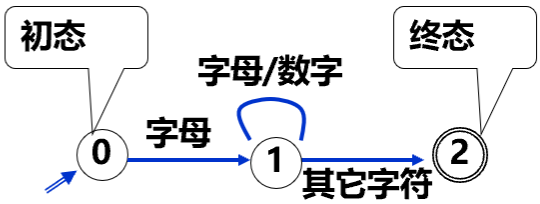
❑ **状态转换图**（state transition diagram），简称转换图,是一张有限方向图，是设计词法分析器的有效工具；它由如下成分构成：

- ✓**结点**(node)：圆圈表示结点，代表状态（state）
- ✓**有向边**（弧）:连接结点，边上的标记字符表示该状态下可能接收或识别的字符；

例子



状态转换图



识别标识符的转换图

词法分析的出错处理只能识别数字开头的标识符、字母表以外的非法字符等，对于错把if写成fi这样的错误要结合语法和语义才能分析出来

词法分析器可以作为单独的处理，也可以作为语法分析器的子程序，当语法分析需要下一个新单词时就调用该函数

符号表的管理：一条记录包含名字域和信息域（对应二元式吗？），由于域的长度不固定，可采用间接表的技术。词法分析阶段当识别到新的名字时就会登入符号表，信息会在后面的阶段完善

语法分析

输入是终结符组成的串（二元式），输出是正确句子的语法树或报错

自上而下：从S出发，最右推导，直到ω

自底向上：从ω出发，最左归约，直到S

自上而下

回溯分析法

不确定方法

穷举所有的语法树，直到找到匹配的那一课，不匹配就回溯（类似DFS）

穷举规则：使用待匹配指针指向输入串中的符号，一个一个地尝试当前非终结符的候选产生式，匹配到了就指针后移，没匹配上又没终结就回溯

因为待匹配指针是从左往右移动的，所以左递归无法判断何时终止，公共左因子无法选择合适的候选式，而造成回溯

消除回溯：

- 将左递归转换成右递归

  消除直接左递归的方法

```
// 左
S->Sa|b
// 右
S->bS'
S'->aS'|ϵ
//a可以是终结符和非终结符组成的串
//比如S->S+T，就改成S'->+TS'|ϵ，！！！别忘了ϵ
```

消除推导产生的间接左递归：A1->f(A2); A2->f(A3); ... ;An->f(A1)，一般是这种循环推导产生了间接递归，我们按这个顺序就能推导出直接递归，然后再改写

notice：推导的时候一般会保留根；要把整个链条上的规则都用上才能推导出完整的规则，别漏掉了产生非终结符串的情况，最后检查一下是否等价

- 提取公共左因子

  有左公因子会导致试探策略无法找到改选哪个规则，所以一般提取左公因子，把右边的字符串再写成一个从非终结符到终结符的产生式

- 不管当前的待匹配符是什么，ε都是可能的选项，无法判断应该选哪条规则，除非目前输入串已经扫描结束，只能使用产生ε的规则，所以ε也可能导致回溯

只有自上而下需要消除左递归，自下而上不用

递归下降分析法

我们想构造一个不带回溯的（确定的）自上而下的分析程序，所以先对左递归和公共左因子进行改写（但要注意改写后也不一定能保证无回溯）

对每个非终结符构造函数，在函数中匹配它的产生式，每个候选式写成一个if，如果产生终结符，匹配上了则待匹配指针后移，如果产生非终结符，则调用它的函数，如果一个候选式都匹配不上则报错（所以如果候选式有ε的话是一定不会报错的）

BNF：->, <, >, |

扩充的BNF：{α}：α的任意多次重复，相当于α*；[α]：α可有可无，相当于α|ε；可以画出对应的状态转换图

改写成扩充的BNF，已经相当于做了几步推导，写出了简化形式，所以可以降低递归下降分析树的高度

预测分析法

表驱动

下推栈+预测分析表+控制程序（12分大题：求集合、表，求某个文法的分析过程，分析它是不是LL(1)）

所有自上而下的方法都要提取公共左因子和消除左递归（**要注意一下有没有间接的，把每个最左的非终结符都代入检查一下**）（改写的时候注意别把根改变了）

1. 控制程序

❑预测分析器的执行算法：
预测分析程序总是按照栈顶符号和当前输入符号a行事。分析开始时，栈底先放一个#，然后放进文法的开始符号。对任何(X,a)，总控程序执行下述动作之一：

① **若X=a=‘#’,分析成功，且分析过程终止；**
② **若X=a≠‘#’,把X从栈顶上托，并让a指向下一个输入符号；**
③ **若X是非终结符，则查看分析表M，若M[X,a]中存放着X的一个产生式，则上托X，并把产生式右部符号按逆序推进栈；若M[X,a]是“出错”标志（也就是空白），则调用出错处理程序error;**

2. 求FIRST和FOLLOW集

FIRST(α)，α是包含N和T的符号串，包含α的所有可能推导的开头终结符及可能的ε

- α->a...|ε，FIRST(α)={a, **ε**}
- α->B...，FIRST(B)-{ε} 加入FIRST(α) （B不可以为空）
- α->$A_1 A_2 \dots A_k \dots A_n$，A1到k-1可以为ε，Ak不空，ε和FIRST(Ak)-{ε}加入FIRST(α)

Notice：只有α能直接产生ε时，对于第三种情况，就是只有A1到An都能为ε时，才将ε加入FIRST(α)，如果是α->Bab...，B->cd|ε，α其实是不会产生ε的，因为后面一定会有ab...，在构造预测分析表时，不能直接产生ε，就不能轮到FOLLOW集

FOLLOW(A)，A的follow集不是S的，包含该文法S的所有句型中紧跟在A后的终结符即可能的#

- #加入FOLLOW(**S 起始文法**)
- S->...Aa...，a加入FOLLOW(**A**)，时刻理解好follow的含义
- S->...AB...，FIRST(B)-{ε}加入FOLLOW(A)，**注意减ε**
- A->...αB，将FOLLOW(A)加入FOLLOW(B)（前加入后）
- A->...BDF..., B后只有非终结符，且都可能为ε，将FOLLOW(A)加入FOLLOW(B)

加入的（子集）可以记下来

一直按这几条规则迭代直到集合不再增大，所以**一定要重复几遍**

3. 构造分析表，所谓预测分析表就是能根据终结符（FIRST集和FOLLOW集）预测产生式（分析表的第一栏中没有空）

设有文法G，构造FIRST(α)和FOLLOW(A)，然后执行如下算法
①对文法G的每个产生式执行(2)和(3)；
②对每个终结符a∈FIRST(α)，则把A→α放入M[A,a];
③若ε∈FIRST(α)，则对所有b∈FOLLOW(A)，把A→α放入M[A,b];
④把所有无定义（空白）的M[A,a]标上“出错”标志

2）是看α的FIRST，不是看A的，因为可能A->α|β，a只属于α，所以就只将A->α放入M[A,a]，如果没有或，则FIRST(A)=FIRST(α)，只看FIRST(A)即可

3）FOLLOW集对应产生ε的产生式，即自己无法处理就让后面的非终结符来产生，**只看FIRST集中有ε的非终结符的产生式**（是不是放入A->ε就可以了？本质上应该是，但有的可能没有直接写A->ε，而是A->BC，B->ε，C->ε），所以FOLLOW集不一定能全部用上

预测分析表中#对应的一栏可能没有产生式，或者对应A->ε，就是最后是直接遇到#和最后空几个之后遇到#的区别

LL(1)文法 仅利用当前非终结符和向前查看1个输入符号（即输入串的带匹配符号）就能唯一决定采取什么动作。

> ❑ **LL(1)文法：**
> **设有文法G，若它的任一产生式$A\to\alpha_1|\alpha_2|...|\alpha_n$,均满足:**
> **①$FIRST(\alpha_i)\cap FIRST(\alpha_j)=\varnothing$,其中$i\neq j,i,j=1,2,...,n$**
> **②若$\alpha_i \overset{+}{\Rightarrow}\varepsilon$,则$FIRST(\alpha_j)\cap FOLLOW(A) = \varnothing$,$j=1,2,...,n$且$j\neq i$**
> **则文法G称为LL(1)文法。**

LL(1)文法可以保证自上而下匹配的唯一性，不是LL(1)文法就会出现分析表中一项可以填两个产生式的情况，有多重入口，不是唯一确定

对于非LL(1)文法，可以设置最近匹配原则来使得入口唯一确定

是不是所有递归都能变成右递归？单递归（线性）应该都可以，中间递归的改写：S->aSb，可改成S->AB，A->aA，B->bB

总得来说，递归形式和LL(1)文法没有确定的关联（右线性文法（即单右递归）是不是一定LL(1)？），改写的意义是什么？消除左递归和公共左因子两种可能产生不确定性的因素，但可能还有别的产生不确定性的因素，（消除左递归和公共左因子也不意味着就变成了右线性），比如直接或间接的中间递归等

LL(1)文法的第一条规则是对公共左因子说法的拓展，第二条规则是对非右递归的扩展？

中间递归也可能会导致非LL(1)文法，比如S->aSa|ε，但可以改写成S->aaS|ε

自下而上

移进-归约分析法

> ❑ **短语：设有文法G，S是开始符号，设$\alpha\beta\delta$是G的一个句型，若有**
> **$S\overset{+}{\Rightarrow}\alpha A\delta$且$A\overset{+}{\Rightarrow}\beta$**
> **则称β是句型$\alpha\beta\delta$关于A的短语；**

短语：由非终结符推导出的句型或句子，只包含叶子结点中的符号

不能单独推出的不是，S->A+B，S不能单独推出A，A不是短语，A+B才是

XX句型的短语，画出从开始符推出XX句型的语法树，根据这棵语法树找，不是从这个句型开始推语法树

直接短语：A一步推导出β

句柄：最左直接短语

规范归约：找句柄归约，即最左归约，是最右推导的逆过程

算符优先分析法

算符文法（形式有点像四则运算的文法）：不含两个或两个以上直接相连的非终结符，也且不含ε产生式

算符优先文法：如果算符文法G的任何两个终结符之间至多只有一个优先关系（优先关系不是数学关系，不对称，不能传递，也未必存在）

> ❑ **算法之间的优先关系：**
> **对算符文法G, $a,b\in V_T$，$P,Q,R\in V_N$，定义**
> **① a=b:G中有P→…ab…或P→…aQb…**
> **② a<b:G中有P→…aQ…且$Q\overset{+}{\Rightarrow}$b…或$Q\overset{+}{\Rightarrow}$Rb…**
> **③ a>b:G中有P→…Qb…且$Q\overset{+}{\Rightarrow}$…a或$Q\overset{+}{\Rightarrow}$…aR**

素短语：至少含一个终结符，且不包含除它自身之外的更小素短语（比如`i1*i2`和i1都是短语，那么i1是素短语，i1*i2不是，因为它包含i1）

控制流程

- 初始压入#
- 读入字符b，a=最顶部的终结符（由于是算符文法，只可能是栈顶或栈顶下一个）
  - a=b=# 结束
  - a<=b b入栈
  - a>b 归约最左素短语（根据优先关系的定义可以证明，最左素短语中的终结符都相等，大于之前和之后的终结符，就是一直往下找，直到出现一个终结符小于上一个终结符），将其出栈，b入栈
  - 空白 出错

算符优先分析法并不属于规范归约，属于结构归约，处于栈顶的最左素短语与对应的产生式**在结构上一致，长度一致，对应的终结符相同，而对应的非终结符可以不同。**比如`F+F*F`可以用`T->T*F|F`归约

优先关系表

构造集合

FIRSTVT(P)：由P产生的第一个终结符

首先找直接的，然后找P->Q…，后加入前

LASTVT(P)：由P产生的最后一个终结符

先找直接的，然后找P->…Q，后加入前，重复直至不再扩大（每次谁加入谁就用括号标记出来，一旦更改就同时改）

构造表（构造算法要会写，可能会考默写算法？）

对文法G的每个产生式$P \to X_1X_2...X_n$，构造其优先关系表的算法如下：

for i:=1 **to** n-1 **do**
  begin
    if X_i和X_{i+1}均为终结符 **then** 置$X_i=X_{i+1}$;
    if $i \le n-2$且X_i和X_{i+2}都为终结符，而X_{i+1}为非终结符
    then 置$X_i=X_{i+2}$;（两个连续的或者间隔一个的终结符置为相等）
    if X_i为终结符而X_{i+1}为非终结符
    then for FIRSTVT(X_{i+1})中的每个a **do** 置$X_i<a$;
    if X_i为非终结符而X_{i+1}为终结符
    then for LASTVT(X_{i+1})中的每个a **do** 置$a>X_{i+1}$;
  end

PS：理解优先关系和控制流程?

归约是自底向上的，所以需要更多步推导出来的优先级更高，被更早归约，所以控制流程中我们也是归约优先级高的

PS：理解从LASTVT和FIRSTVT集合构造表

前两个if对应优先级的第一个定义，第三个if对应第二个定义，第四个if对应第三个定义（记住FIRSTVT、LASTVT都是由非终结符多推了一步出来的，所以都是更优先的、大于的）

？找句柄

？后面再理解一下最左素短语中的终结符大于之前和之后的终结符

？结构规约

PS：算符优先法和四则运算的双栈法、表达式树法

分析规则比较难记，要在理解的基础上结合推理来记忆

LR分析法

控制流程

action表 [状态，输入字符]　　goto表 [状态 非终结符]

画双栈归约过程会清楚一些，第一步把a推入符号栈，不用双栈的话就想着状态和符号对应，上托|β|个状态

驱动程序控制过程（双栈，状态栈和符号栈对应）：

初始时，状态栈中置初始状态0，输入指针指向输入串第一个符号。以后根据状态栈栈顶状态s和输入指针所指当前输入符a查分析表action[s,a]

① **若action[s,a]= s_j,则将状态j推入栈顶，输入指针指向下一输入符号。**

② **若action[s,a]= r_j,则按第j个产生式A→β规约，设|β| =t,应上托t个状态出栈，再根据当前的栈顶状态si及规约后的非终结符A，查goto表，若goto[si,A]=j,则将状态j推入栈顶（指针不动）。**

③ **若action[s,a]=acc,分析成功，输入串被接受。**

④ **若action[s,a]或goto[s,A]为空白（error），则转出错处理程序。**

活前缀：规范句型（由规范推导推出的句型）中不含句柄之后任何符号的一个前缀

LR是规范归约，所以要识别句柄，活前缀有可能已经含有句柄或者遇到之后的终结符会形成句柄，所以我们可以先识别活前缀（活前缀也可以认为是已进栈的部分）

LR(0)项目：在一个产生式右部添加一个圆点，称为一个LR(0)项目，用以表示**活前缀和句柄的关系**

活前缀与句柄的三种关系

注意：LR(0)项目中的小圆点，表示识别的位置。圆点左边是已识别部分，右边是期待识别的部分；

❑ **LR(0)项目的分类：（选择题）**

① **归约项目**:形如A→α•

② **移进项目**:形如A→α•aβ，$a \in V_T$

③ **待约项目**:形如A→α•Bβ，$B \in V_N$

④ **接受项目**:形如S→α•， S为开始符号

也即·左边是已进栈的部分，右边是期待进栈的部分

待约的意思是我们期待B入栈，但非终结符不能直接入栈，只能等待其某个候选式的全部符号入栈后归约形成B

文法的拓广：对于S->A|B的情况接受项目不唯一，不确定什么时候结束，所以在文法G中增加产生式S'→S，从而使S'→S•成为唯一的接受项目

识别活前缀的不确定状态转换图

❑ **构造识别所有活前缀的转换图：**

① **每个状态是一个LR(0)项目；**

② **S′→•S是唯一的初态；**

③ **所有的其他项目是终态，是某个活前缀的识别态；**

④ **若状态i为$X \to X_1 \ldots X_{i-1} \bullet X_i \ldots X_n$，状态j为$X \to X_1 \ldots X_i \bullet X_{i+1} \ldots X_n$，则从状态i画一条有向边到状态j，标记为$X_i$；**

⑤ **如果X_i为一非终结符，并有$X_i \to \alpha_1 | \alpha_2 | \ldots | \alpha_m$ 则从状态i画ε有向边到所有状态$X_i \to \bullet \alpha_i$。** （往前多看一步，看Xi产生什么）

消除不确定（ε）->构建LR(0)项目集规范族（有效项目的闭包）

一般的，如果A→α•Bβ对活前缀δα有效，即有$S\Rightarrow\delta A\omega\Rightarrow\delta\alpha B\beta\omega$；令$\beta\omega\Rightarrow\omega'$，$B\in V_N$，关于B的产生式为$B\rightarrow\eta$，则有$S\Rightarrow\delta A\omega\Rightarrow\delta\alpha B\beta\omega\Rightarrow\delta\alpha B\omega'\Rightarrow\delta\alpha\eta\omega'$这表明$B\rightarrow\bullet\eta$也对活前缀δα有效。（也相当于是往前看了一步）

① **有效项目：对于项目A→α•β，如果有**

$$S\overset{*}{\Rightarrow}\delta A\omega\Rightarrow\delta\alpha\beta\omega$$

则称$A\xrightarrow{R}\alpha\bullet\beta$对活前缀δα有效（αβ是句柄， δα 活前缀遇到β就能激活，可以把有效项目当作能产生形成句柄需要的符号的产生式（点打在活前缀之后））

② **有效项目集：对于一个活前缀有效的项目可能不止一个，对活前缀δα有效的项目集合称为δα的有效项目集。**

③ **LR(0)项目集规范族：文法G的所有有效项目集组成的集合称为文法G的LR(0)项目集规范族；**

·后面如果是非终结符，就要往前多看一步

每个状态是一个LR(0)项目，求这个项目的I闭包closure(I)，也就是**求能产生 形成句柄需要的符号的 产生式**（需要的符号前面加上点就是项目）。goto的意思就是读入·后面的符号（N或T），然后发生状态转移。每次转移到新的状态的时候要再求这个状态的闭包，也就是加入点号之后的非终结符的生成式（A->B·B，B->CD, C->g，这时要加入B->·CD和C->·g吗？）

抓住一个核心：

❑ LR(0)项目集规范族的构造

① **closure(I)：设I是一个LR(0)项目集，closure(I)按如下规则构造：**

- ✓ **对任何i∈I，都有i∈closure(I)；**
- ✓ **若项目A→α•Bβ∈closure(I)，且B→η是文法的一个产生式，则B→•η∈closure(I)；**
- ✓ **重复前两个规则，直到closure(I)不再增大；**

可以验证，如果项目A→α•Xβ对δα有效,则A→αX•β对δαX有效；即如果I是对δα有效的项目集，则go(I,X)则是对δαX的效的项目集

② **go(I,X)：设I是一个LR(0)项目集，X是文法符号，状态转换函数go(I,X)定义为：**

$$go(I,X)=closure(\{A\rightarrow\alpha X\bullet\beta\mid A\rightarrow\alpha\bullet X\beta\in I\})$$

状态是什么？为什么会有归约几个字符就弹几个状态出栈

下周一半期考试 考到预测分析法

实验报告

代码放在报告的最后

考试前一周交报告

语法分析器，当面检查

词法分析器

三种报错（要有报错截图）

不匹配，比如Pascal中只有:=出现了：，只有单独的:就不对

非法字符

标识符长度溢出

识别回车是为了报出出错的行数，要详细写出哪一个词出错了

7f 词法分析不会报错

不能直观感受，要根据规则