编译原理课程设计

二零一六•春

目录

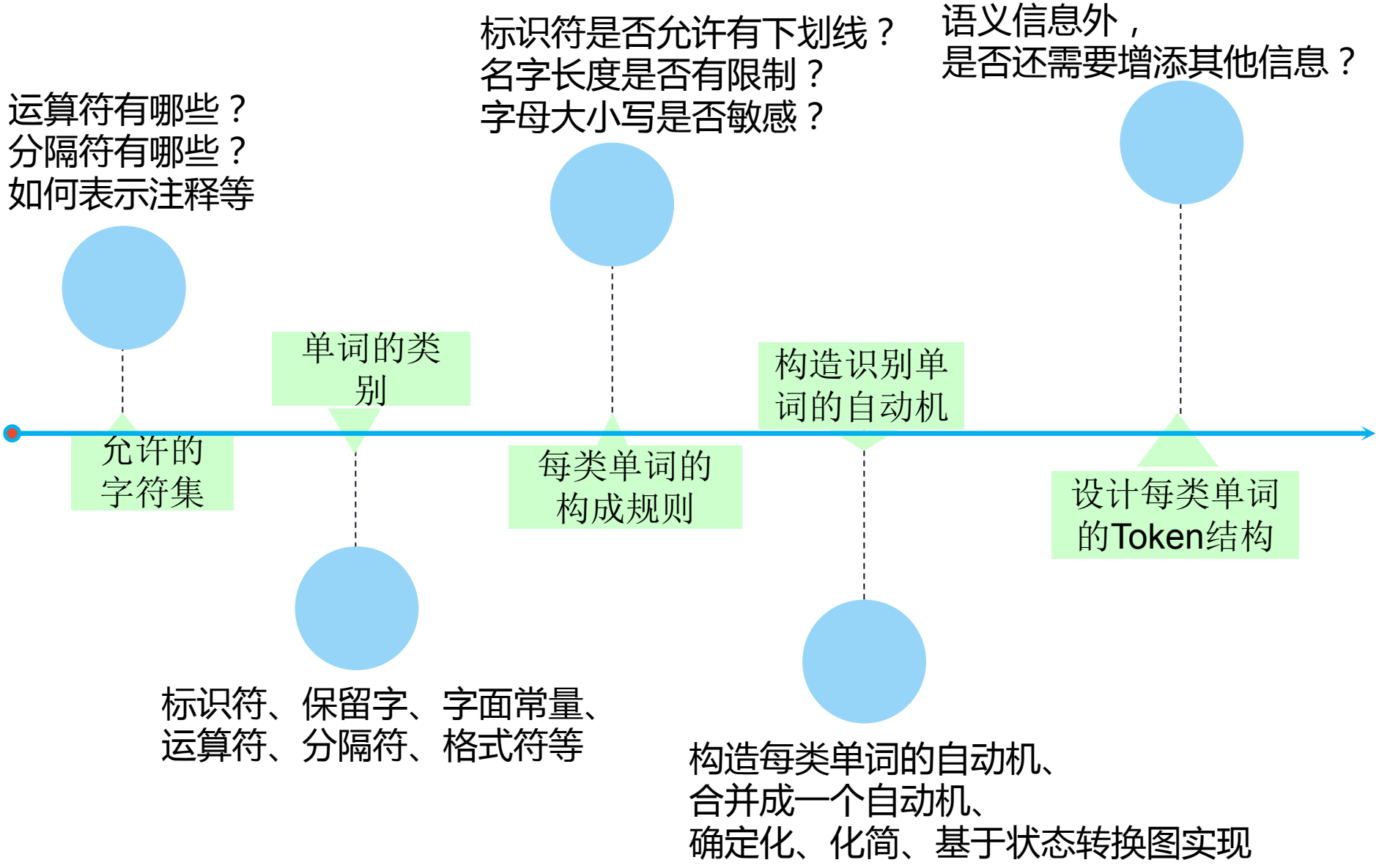
目的

- 通过实际设计和开发，更进一步地理解和掌握“编译程序的设计方法和实现技术”；
- 培养**大型软件**的程序设计方法；
  - 遵循基本的软件开发过程：设计、编码、集成和测试

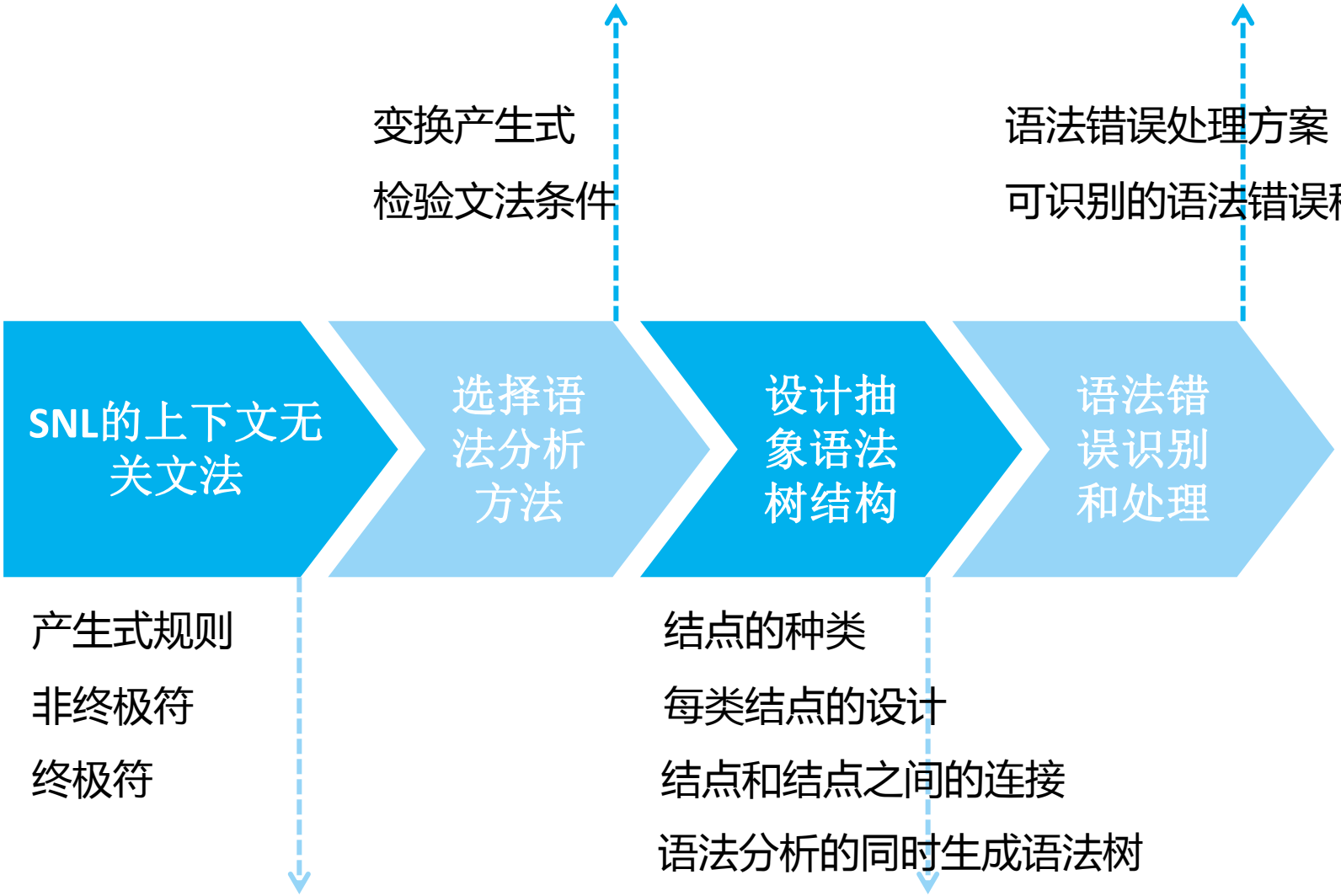
内容

- 总目标：
  - 设计并实现**SNL程序设计语言**的词法分析程序和语法分析(LL1或递归下降)程序；
- 具体任务：
  - 了解SNL语言的词法和语法
  - 熟悉编译程序中词法分析程序和语法分析程序之间的关系，并设计好词法分析和语法分析的接口
  - 设计并编写词法分析程序和语法分析程序
  - 设计Token和抽象语法树的数据结构
  - 设计并实现词法错误和语法错误的识别和处理算法

关于SNL语言的词法

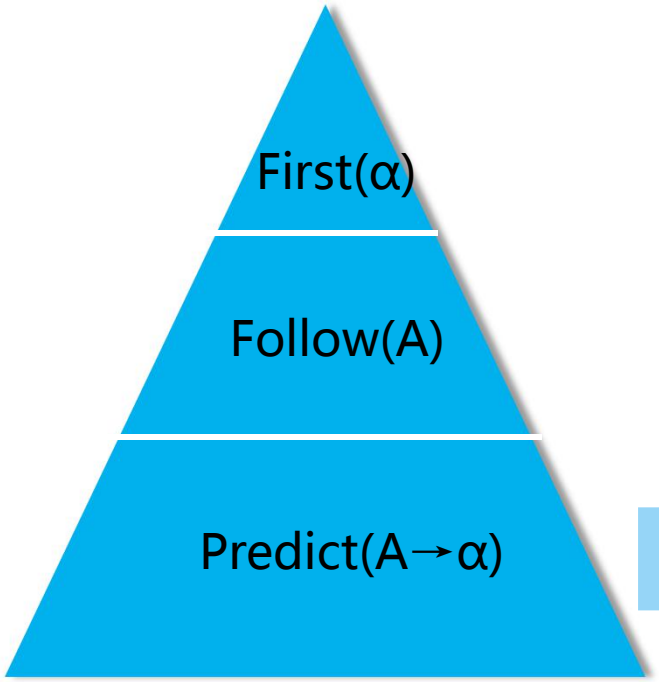

运算符有哪些？
分隔符有哪些？
如何表示注释等
允许的字符集
单词的类别
标识符、保留字、字面常量、
运算符、分隔符、格式符等
标识符是否允许有下划线？
名字长度是否有限制？
字母大小写是否敏感？
每类单词的构成规则
构造识别单词的自动机
构造每类单词的自动机、
合并成一个自动机、
确定化、化简、基于状态转换图实现
Token中除单词类别、
语义信息外，
是否还需要增添其他信息？
设计每类单词的Token结构

关于SNL语言的语法


变换产生式
检验文法条件
语法错误处理方案
可识别的语法错误种类
SNL的上下文无关文法
选择语法分析方法
设计抽象语法树结构
语法错误识别和处理
产生式规则
非终极符
终极符
结点的种类
每类结点的设计
结点和结点之间的连接
语法分析的同时生成语法树

语法分析条件的检查

针对文法的每条产生式A→α，计算：



针对每个非终极符，检查：

$$Predict(A \rightarrow \alpha) \cap Predict(A \rightarrow \beta) = \Phi$$

抽象语法树的数据结构

```
struct {symbol t;            // 结点对应的符号
        int n;               //儿子结点的个数
        treeNode* son        //儿子结点链表
       } treeNode
```

操作：

treeNode* ***buildNode***(symbol: V_N);

treeNode* ***addNode***(father:*treeNode, son:*treeNode)

递归下降法-抽象语法树的生成

文法如下：S → a E
E → S d | d

递归下降程序如下：

```
S( )
{
    match(a);
    E();      }
E()
{
    switch(token) {
    case 'a': S();
              match(d);
              break;
    case 'd': match(d);
              break;
   default: error();
 }
main()
{
    read();
    S();
}
```

增加语法树生成的递归下降程序如下：

```
treeNode* S( )
{
    match(a); A = buildNode('a');addNode(root, a)
    e = E();addNode(root,e);
    return root;       }
treeNode* E()
{
    e = BuildNode('E');
    switch(token) {
    case 'a': s = S(); addNode(e,s);
              match(d);D = buildNode('d');addNode(e,D);
              break;
    case 'd': match(d);D = buildNode('d');addNode(e,D);
              break;
   default: error(); return e;
 }
void main()
{
     treeNode* root = buildNode('S');
     read();
     root = S();
}
```

LL1分析-抽象语法树的生成

文法如下：S → a E
E → S d | d

LL1分析表如下：

	a	d	#
S	[1]		
E	[2]	[3]	

```
main()
{   push(S);    //将开始符压入符号栈
    read(ch);  //从输入流读入一个符号
    while (stack!=Null && ch!='#')   //符号栈不空，且未到达输入串尾部
    {
        X = pop(1);       //从符号栈弹出一个符号
        if (X∈VT)&& (X=a) {pop(1); read(ch);}      //匹配
        else if (X∈VT)&& (X!=a){error();break;}    //不匹配
        else if (X∈VN) {   i = LL1(X,ch);     //查找LL1分析表找到替换产生式号
                           pop(1);             //弹出非终极符，准备替换
                           push(Yn, ..., Y1); // Y1.....Yn是产生式i的右部,执行替换}
    }
}
```

LL1分析-抽象语法树的生成

LL1分析的同时生成语法树：在处理产生式左部符号时，其右部符号尚未处理，无法获得其语法树信息，所以额外需要一个语法树栈，用来保留尚未完成处理的子结点的语法树结点地址

```
treeNode* main()
{
    treeNode* root = buildNode('S');
    push(S)；pusht(root);  //将开始符压入符号栈,将根结点压入语法树栈
    read(ch)；//从输入流读入一个符号
    while (stack!=Null && ch!='#')   //符号栈不空，且未到达输入串尾部
    {
        X = pop(1);       //从符号栈弹出一个符号
        if (X∈VT)&& (X=ch) { pop(1); read(ch); popt(1); }
                                  //匹配,从语法树栈弹出终极符对应的语法树结点；
        else if (X∈VT)&& (X!=ch){error();break; }    //不匹配
        else if (X∈VN) {   i = LL1(X,ch);     //查找LL1分析表找到替换产生式号
                           pop(1);            //弹出非终极符，准备替换
                           push(Yn, ..., Y1); // Y1.....Yn是产生式i的右部,执行替换
                          tn = buildNode('Yn'); ....;t1 = buildNode('Y1');
                          addNode(X, Yn);.....addNode(X,Y1);
                          popt(1);    //弹出非终极符对应的语法树结点
                          pusht(tn); pusht(tn-1);.....pusht(t1); //压入各子结点   }
    }
    return root;
}
```

要求

- 平时：
  - 保证出勤率
  - 无故缺席两次以上者成绩为不及格
- 验收：
  - 原则上最后一次课验收并答辩，可提前不可延后
  - 提交代码和课程设计报告，演示程序并回答提问
- 成绩：
  - 分优秀(约15%)、良好(约40%)、中等(约30%)、及格(约15%)和不及格五档