第二章 高级语言及其语法描述

授课人：高珍

gaozhen@tongji.edu.cn

概述

- 要学习和构造编译程序，理解和定义高级语言是必不可少的
- 本章概述高级语言的结构和主要共同特征，重点介绍程序设计语言的语法描述方法

内容线索

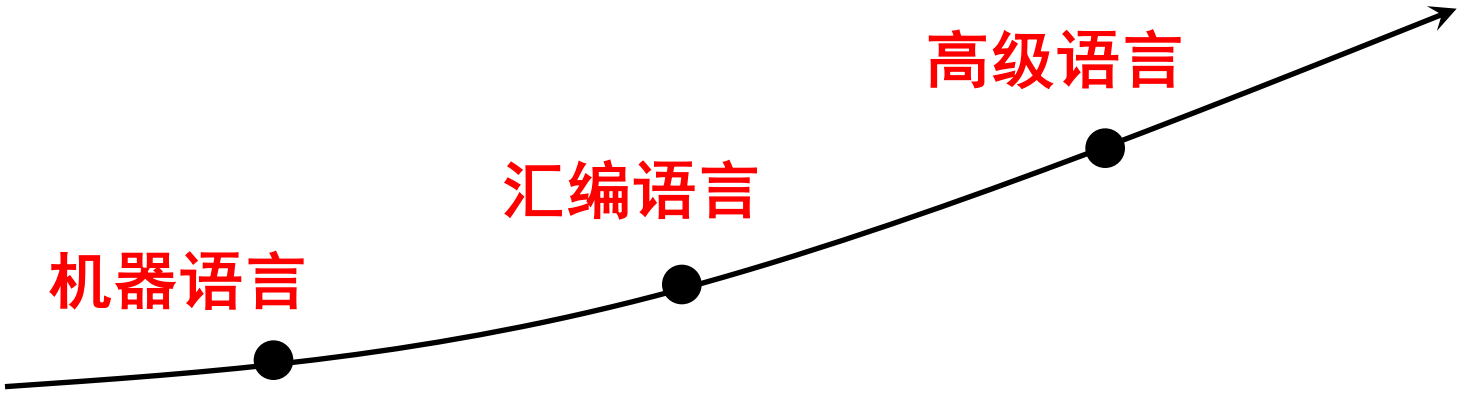
- 程序设计语言的定义
- 高级语言的一般特性
- 程序语言的语法描述（PPT-P47）

程序设计语言的定义

- **程序设计语言的定义**
- **程序语言的内涵**
- **程序语言的功能**

程序设计语言的定义

- 程序设计语言是为书写计算机程序而人为设计的符号语言。



各级语言的比较

比较	机器语言	汇编语言	高级语言
硬件识别	是唯一可以识别的语言	不可识别	不可识别
是否可直接执行	可直接执行	不可，需汇编、连接	不可，需编译/解释、连接
特点	✓面向机器 ✓占用内存少 ✓执行速度快 ✓使用不方便	➢面向机器 ➢占用内存少 ➢执行速度快 ➢较为直观 ➢与机器语言一一对应	■面向问题/对象 ■占用内存大 ■执行速度相对慢 ■标准化程度高 ■便于程序交换，使用方便
定位	低级语言，极少使用	低级语言，很少使用	高级语言，种类多，常用

程序设计语言的定义

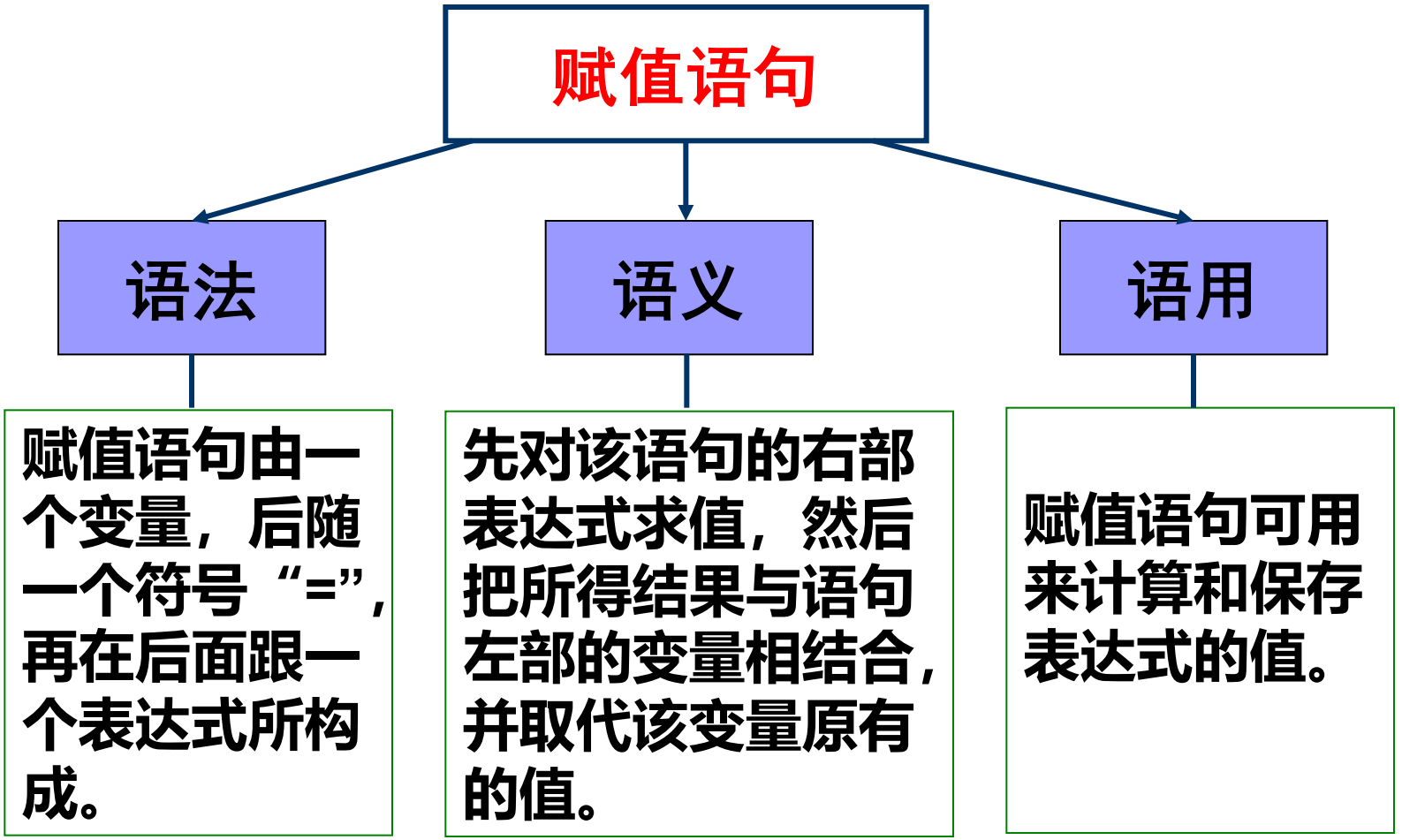
- ✓ 程序设计语言的定义
- ■ **程序语言的内涵**
- ■ 程序语言的功能

程序语言的内涵

语法	表示构成语言句子的各个单词符号之间的组合规律(构成规则)。
语义	表示按照各种表示方法所表示的各个单词符号的特定含义（各个单词符号和单词符号所表示的对象之间的关系）。
语用	表示各个单词符号或语言词句与其使用之间的关系。

程序语言的内涵

举例：C语言的赋值语句



1 语法

- 语言的语法是指可以形成和产生程序的一组规则。它包括**词法规则**和**语法规则**。
  - 词法规则是指程序中单词符号的形成规则。单词符号一般包括：标识符、基本字、常量、算符和界符。
  - 语法规则是指程序中语法单位的形成规则。程序语言的语法单位有：表达式、语句、分程序、过程、函数、程序。
- 描述方式：词法规则和语法规则都可以用自然语言、语法图、BNF(巴科斯范式)范式、或文法等描述。

语法描述方式（1）

（1）自然语言

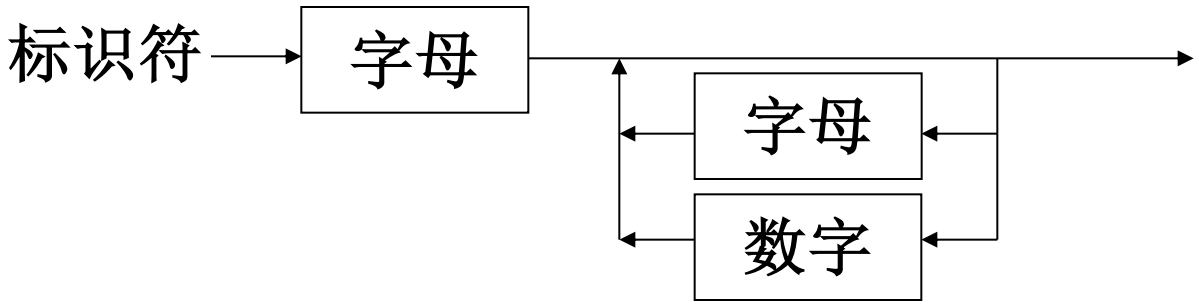
例1. 标识符是由字母后跟若干个（包括0个）字母或数字的符号串组成的。

例2. 赋值语句由一个变量，后随一个符号“=”，再在后面跟一个表达式所构成。

（2）语法图

- 是用图解形式描述程序设计语言语法规则的工具。

例1. 标识符的构成规则用语法图描述为：



语法描述方式（2）

（3）BNF范式

- 巴科斯范式(BNF: Backus-Naur Form 的缩写)是由 John Backus 和 Peter Naur 首先引入的用来描述计算机语言语法的符号集。
- 现在，新的编程语言几乎都使用巴科斯范式来定义编程语言的语法规则。

简单算术表达式的BNF范式

<简单表达式>:: = <简单表达式>+<简单表达式>

<简单表达式>:: = <简单表达式>*<简单表达式>

<简单表达式>:: = (<简单表达式>)

<简单表达式>:: = i

- 或者

<简单表达式>:: = <简单表达式> + <简单表达式>

| <简单表达式> * <简单表达式>

| (<简单表达式>) | i

巴科斯范式的内容

- 在双引号中的字("word")代表着这些字符本身。而double_quote用来代表双引号
- 在双引号外的字（有可能有下划线）代表着语法部分
- 尖括号(< >)内包含的为必选项
- 方括号([ ])内包含的为可选项
- 大括号({ })内包含的为可重复0至无数次的项
- 竖线(|)表示在其左右两边任选一项，相当于"OR"的意思
- ::= 是“被定义为”的意思。

2 语义

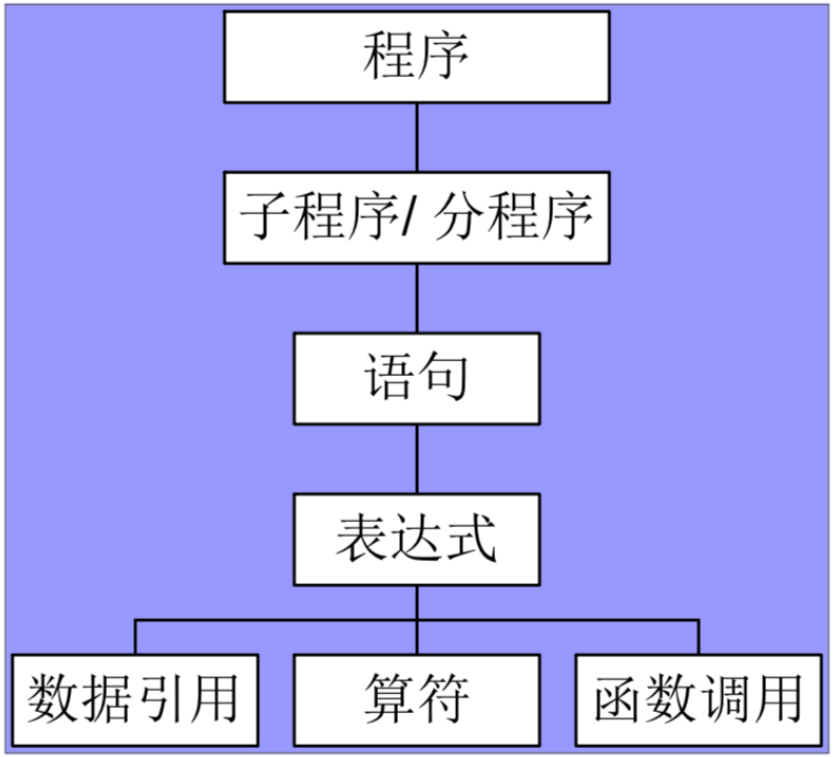
- 对于一个语言来说，不仅要给出它的词法、语法规则，而且要定义它的单词符号和语法单位的意义。这就是语义问题。
- 语义是指这样的一组规则，使用它可以定义一个程序的意义。
- 我们采用的方法为：基于属性文法的语法制导翻译方法。

程序设计语言的定义

- ✓ 程序设计语言的定义
- ✓ 程序语言的内涵
- ■ 程序语言的功能

程序语言（面向过程语言）的功能

- 一个程序语言的基本功能是描述**数据**和**对数据的运算**。
- 所谓程序，从本质上来说是描述一定数据的处理过程。
- 一个程序大体可以视为如图所示的层次结构



内容线索

- ✓ 程序设计语言的定义
- ■ 高级语言的一般特性
- ■ 程序语言的语法描述

高级语言的分类

(1) 强制式：Imperative Language(过程式语言)

- 形式：语句序列
- 举例：Fortran、C、Pascal

(2) 应用式：Applicative Language(函数式语言)

- 形式：func1(…func(n))
- 举例：Lisp

(3) 基于规则：Rule-Based Language(条件→动作)

- 形式：bird(x) → fly(x) & feather(x)
- 举例：Prolog

(4) 面向对象：Object-Oriented Language(封装/继承/多态)

- 形式：class,
- 举例：Smalltalk、C++、Java

几种有代表性的程序设计语言

- Fortran(FORmula TRANslation)
  - 世界上第一个被正式推广使用的计算机高级语言
  - 1954年提出
  - 科学计算领域首选的计算机高级语言。
- COBOL(Common Business Oriented Langauge)
  - 最早的高级编程语言之一，世界上第一个商用语言。
  - 世界上70%的数据都是用COBOL语言处理的，并且90%的ATM事务处理用的都是COBOL语言。
  - 只要大型机存在，COBOL就不会消失

几种有代表性的程序设计语言

- Ada语言
  - 1983年成为 ANSI 标准 ANSI/MIL-STD-1815A
  - 结构化程序设计语言的典范
  - 美国军方软件开发语言
- Pascal语言
  - 瑞士苏黎士理工学院的尼古拉斯-沃斯(Niklaus Wirth)教授在1971年设计
  - 强健数据类型概念、强制性的数据类型声明
  - 最优秀的结构化程序设计的教学语言
- Basic语言
  - 1964年Dartmouth大学的John G. Kemeny和Thomas E. Kurtz发明
  - 1975年，Bill Gates和Paul Allen编写了一个Basic解释器
  - 一种适合初学者的语言

几种有代表性的程序设计语言

- Prolog (Programming In Logic)
  - 1972年法国科莫劳埃小组为了提高归结法的执行效率，研制出一个定理证明程序的程序执行器，标志着第一个逻辑程序设计语言PROLOG的诞生。
  - 1974年及以后，R.科瓦尔斯基进一步从谓词逻辑的HORN子句的角度阐明PROLOG的理论基础，系统地提出逻辑程序设计的思想。
  - 八十年代日本第五代计算机的核心语言
- Lisp(LISt Processor)
  - 1958年，由约翰·麦卡锡（John McCarthy）创造的一种基于λ演算的函数式编程语言。
  - 从Lisp分支出来的Scheme、ML等语言在很多场合的火爆程度甚至超过了许多老牌明星。

几种有代表性的程序设计语言

- C语言和C++语言
  - 1969年，美国贝尔实验室的Ken Thompson为DEC PDP-7计算机设计了最早的UNIX；根据BCPL语言为UNIX设计了一种便于编写系统软件的B语言；
  - 1972—1973年，贝尔实验室的Denis Ritchie改造了B语言，为其添加了数据类型的概念，将其命名为C。
  - 1973年，Ken Thompson小组用C重新改写了UNIX的内核。与此同时，C语言的编译程序被移植到多种计算机上，迅速成为应用最广泛的系统程序设计语言
  - 1983年，贝尔实验室的Bjarne Stroustrup博士对C语言进行改进和扩充，C++语言。
  - 1998年正式发布了C++语言的国际标准ISO/IEC:98-14882。
  - C++支持面向对象的程序设计方法，特别适合于中型和大型的软件开发项目，同时，C++又是C语言的一个超集，这就使得许多C代码不经修改就可被C++编译通过。

几种有代表性的程序设计语言

- Java语言
  - 1990年代初， Sun Microsystems的James Gosling等人开发。最初被命名为Oak，作为一种小家用电器的编程语言，来解决诸如电视机、电话、闹钟、烤面包机等家用电器的控制和通讯问题。
  - 随着Internet的发展，Sun看到了Oak在计算机网络上的广阔应用前景，在1995年5月以“Java”的名称正式发布了。Java伴随着Internet的迅猛发展而发展，逐渐成为重要的Internet编程语言。
  - 是一种可以编写跨平台应用软件的面向对象的程序设计语言

高级语言的一般特性

- **程序结构**
- **数据类型与操作**
- **语句与控制结构**

程序结构——单层结构

- Fortran程序结构　主程序＋若干个辅程序段（子程序、函数）
- 讨论：变量的作用域?

```
Program Main
      Read(I,J)
      Call max(I,J,K)
      Write(100, K)
100   Format(…)
end
```

```
subroutine max(x,y,z)
    integer x,y,z
    if x>y then
       z = x
    else
       z  = y
end
```

程序结构——多层结构

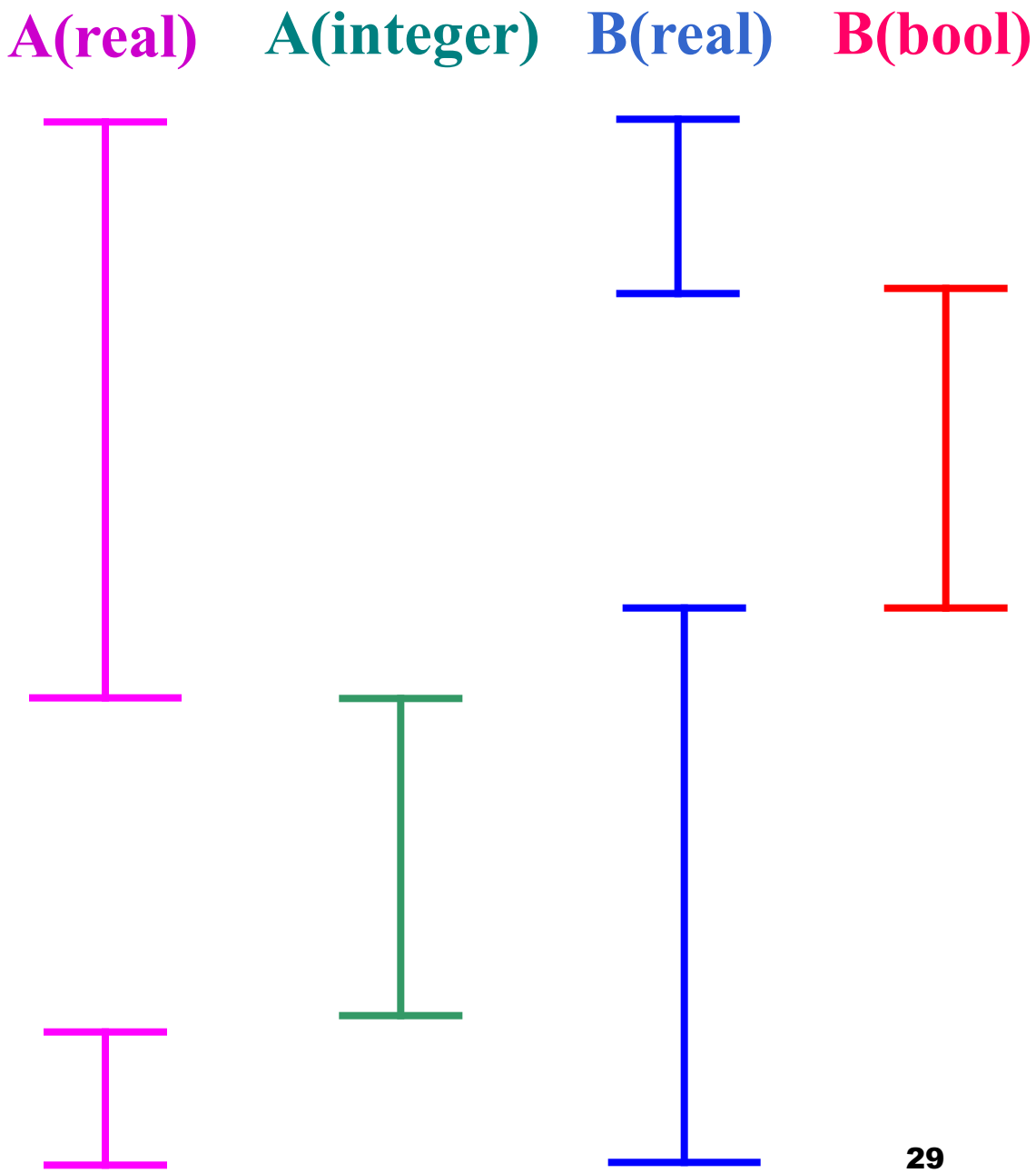
- Pascal程序结构 程序允许嵌套定义

```
Program P;
    var a,x:integer;
     procedure B1(b:integer);
          var i:integer;
          procedure B2(u:integer;Var v:integer);
               var c,d:integer;
               begin
               …
               end
          begin
            …
          end
    begin
          ….
    end
```

讨论：变量的作用域？-最近嵌套原则

- 作用域：一个名字能被使用的区域范围称作这个名字的作用域。
- 允许同一个标识符在不同的过程中代表不同的名字。
- 名字作用域规则--"最近嵌套原则"
  - 一个在子程序B1中说明的名字X只在B1中有效（局部于B1）；
  - 如果B2是B1的一个内层子程序且B2中对标识符X没有新的说明，则原来的名字X在B2中仍然有效。如果B2对X重新作了说明，那么，B2对X的任何引用都是指重新说明过的这个X。

```
program main
   var  A, B : real;
   ...
   procedure P1
    var B:boolean;
        ...
   begin
        ...
   end
   procedure P2
    var A:integer;
        ...
   begin
        ...
   end
begin
   ...
end
```



高级语言的一般特性

- ✓ 程序结构
- ■ **数据类型与操作**
- ■ 语句与控制结构

类型

- 每个被计算对象都带有自己的类型，以类型作为值的属性的概括，因此每个类型都意味着一个值的集合。
- 不同类型的值具有不同的操作运算
- 类型是一个值的集合和定义在这个值集上的一组操作的总称。
  - 如C语言中的整型变量(int)，其值集为某个区间上的整数，定义在其上的操作为+, -, *, /等

数据类型

- 数据类型通常包括以下三种要素：
  - a.用于区别这种类型的数据对象的属性
  - b.这种类型的数据对象可以具有的值
  - c.可以作用于这种类型数据对象的操作

初等类型、复合类型到抽象数据类型

- 类型本不存在
  - 内存里存储的内容，你认为它是什么，它就是什么
- 高级语言设计了**初等数据类型**：整型、浮点型、字符型等。不同的语言也会定义不同的初等类型等。
  - 初等数据类型并不能方便地解决所有问题
- **复合数据类型**是初等数据类型迭代派生而来
  - 典型的代表就是 “结构”，数组也可算作此类
- **抽象数据类型** (ADT)在复合数据类型的基础上增加了对数据的操作
  - 抽象数据类型进而进化为 “类(Class)”

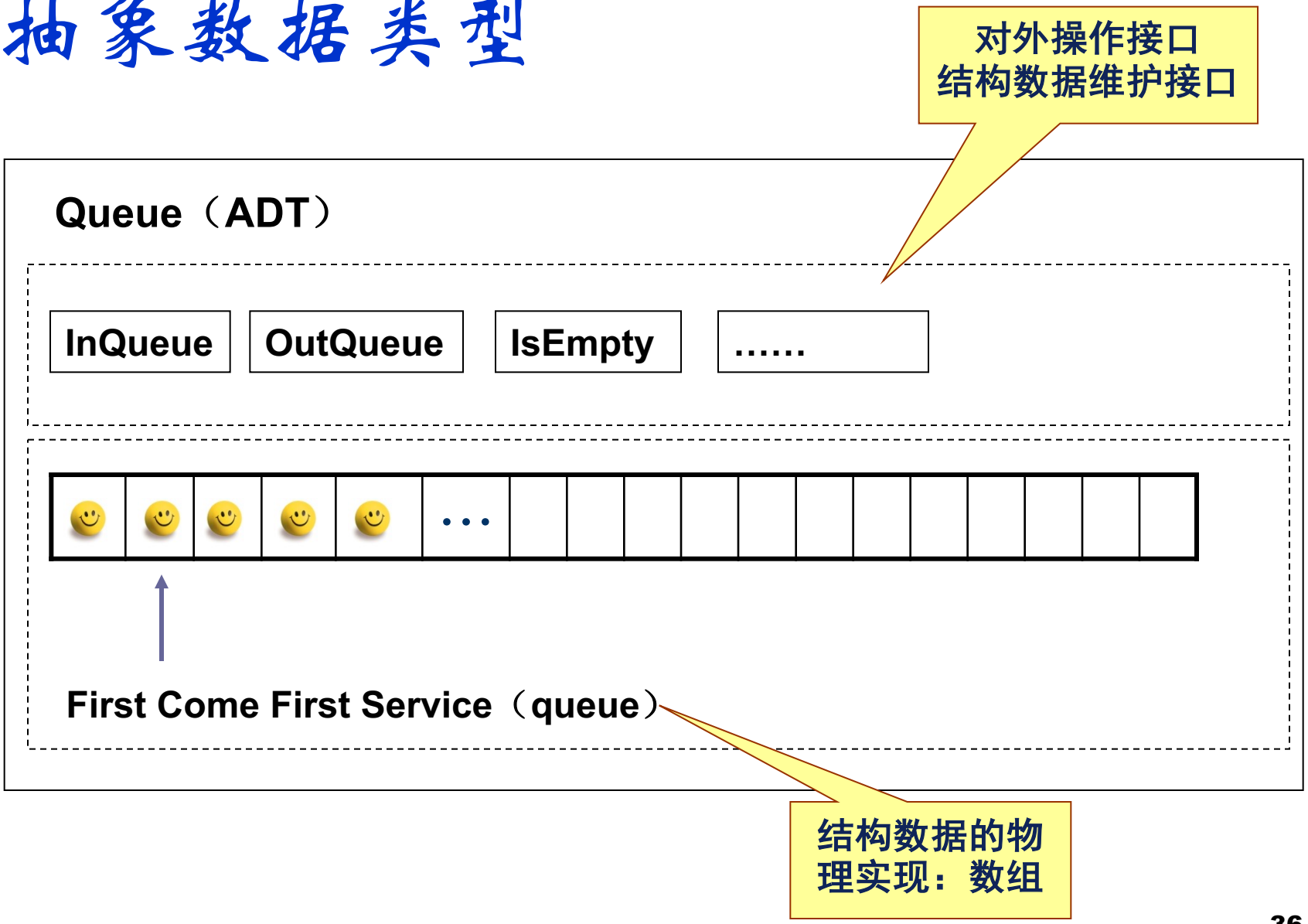
初等数据类型和复合数据类型

- 初等/基础数据类型
  - 数值数据:算术运算
  - 逻辑数据:逻辑运算
  - 字符数据:字符串操作
  - 指针类型:取地址/取值/加减地址等
- 复合数据类型
  - 数组
  - 记录(Record/Pascal)/结构体(Struct/C)
  - ……

抽象数据类型

- 一个抽象数据类型（Abstract Data Type，ADT）定义为：
  （1）一个数据对象集，数据对象由一个或多个类型定义；
  （2）一个作用于这些数据对象的抽象操作集；
  （3）完全封装，用户除了能使用该类型的操作来处理这类数据对象之外，不能作其他的处理。
- 抽象数据类型有两个重要特征：**信息隐蔽**和**数据封装**，使用与实现相分离

抽象数据类型

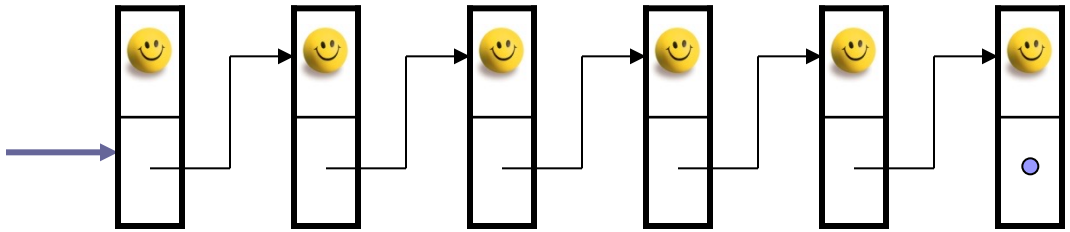

对外操作接口
结构数据维护接口
Queue（ADT）
InQueue
OutQueue
IsEmpty
……
First Come First Service（queue）
结构数据的物理实现：数组

抽象数据类型（续）

对外操作接口
结构数据维护接口

Queue（ADT）

InQueue | OutQueue | IsEmpty | ……



First Come First Service（queue）

结构数据的物理实现：链表

数据抽象的优点

- 可读性、可靠性、可维护性
  - 通过隐藏数据表示，用户代码不能直接访问该类型对象，不依赖于其表示，因此可以修改该类型对象的表示而不影响用户代码

高级语言的一般特性

- ✓ 程序结构
- ✓ 数据类型与操作
- ■ 语句与控制结构

语句与控制结构

- 表达式
- 语句
  - 简单语句：不含其它语句成分的基本句。
    - 说明语句
    - 赋值语句
    - 控制语句
    - 输入输出语句
    - 过程调用语句
  - 复合语句：句中有句的语句

表达式

- 表达式：一个表达式是由运算量（亦称操作数，即数据引用或函数调用）和算符组成的。
- 对于大多数程序语言来说，表达式的形成规则可概括为：
  （1）变量、常数是表达式；
  （2）若E1、E2为表达式，θ为二元算符，则 E1 θ E2为表达式；
  （3）若E为表达式， θ为一元算符，则θE为表达式；
  （4）若E为表达式，则（E ）是表达式。

语句

- 不同程序语言含有不同形式和功能的各种语句。
- 从功能上说语句大体可分**执行性语句**和**说明性语句**两大类：
  - 说明性语句旨在定义不同数据类型的变量或运算。
  - 执行性语句旨在描述程序的动作。
    - 执行性语句又可分为赋值语句、控制语句和输入/输出语句
- 从形式上说，语句可分为**简单句**、**复合句**和**分程序**等。

赋值语句

A:= B

- 意义是：“把B的值送入A所代表的单元”
  - 在赋值句中，赋值号‘：=’左右两边的变量名扮演着两种不同的角色。对赋值号右边的B我们需要的是它的值；对于左边的A我们需要的是它们的所代表的存储单元（的地址）。
- 为了区分一个名字的这两种特征，我们把一个名字所代表的那个存储单元（地址）称为该名字的左值；把一个名字的值称为该名字的右值。

控制语句

- 多数语言中所含的控制语句有：
  - 无条件转移语句： goto L
  - 条件语句： if B then S

    if B then S1 else S2
  - 循环语句： while B do S

    repeat S until B

    for i:=E1 step E2 until E3 do S
  - 过程调用语句： call P(X1,X2,…,Xn)
  - 返回语句： return(E)

输入输出语句

- Standard Input/output
  - getchar()
  - putchar()
  - scanf()
  - printf()
- File Access
  - fopen()
  - fclose()

高级语言的一般特性

- ✓ 程序结构
- ✓ 数据类型与操作
- ✓ 语句与控制结构

内容线索

- ✓ 程序设计语言的定义
- ✓ 高级语言的一般特性
- ■ 程序语言的语法描述

概述

- 对于高级程序语言及编译程序而言，语言的语法定义是非常重要的。本节将介绍语法结构的形式描述问题。

编译原理=形式语言理论+编译技术

形式语言

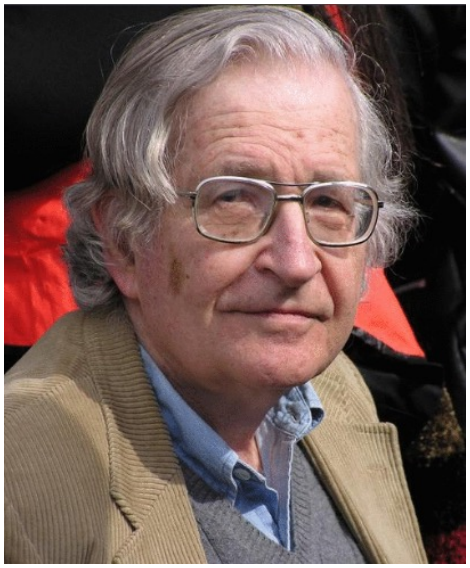
- **自然语言**
  - 人们平时说话时所使用的一种语言，不同的国家和民族有着不同的语言。
- **形式语言**
  - 通过人们公认的符号、表达方式所描述的一种语言，是一种通用语言，没有国籍之分。
  - 形式语言是某个**字母表上的字符串的集合**，有一定的描述范围。

为什么用形式语言

- 形式语言的最初起因： 语言学家乔姆斯基（Chomsky）想用一套形式化方法来描述语言。
- 形式语言在自然语言研究中起步，在计算机科学中得到广泛应用。
  - 最初的应用：编译
  - 现在已广泛应用在人工智能、图象处理、通信协议、通信软件等多个领域
  - 在计算机理论科学方面：
    - 是可计算理论（算法—在有限步骤内求得解、算法复杂性、停机问题）、定理自动证明、程序转换（程序自动生成）、模式识别等的基础。

形式语言与自动机理论的发展

- 1956年，乔姆斯基（Chomsky）从产生语言的角度研究语言
  - 文法 (由文法产生的符号串构成语言)
- 1951-1956年间，克林（Kleene）从识别语言的角度研究语言
  - 自动机 (由自动机识别的符号串构成语言)
- 1959年，乔姆斯基不仅确定了文法和自动机分别从生成和识别的角度去表达语言，而且证明了文法与自动机的等价性。



形式语言真正诞生

文法（Grammar）

- 所谓**文法**是用来定义语言的一个数学模型。
- 表示语言的方法：
  - 若语言L是有限集合，可用列举法
  - 若L是无限集合（集合中的每个元素有限长度），用其他方法。
    - 方法一：文法产生系统，由定义的文法规则产生出语言的每个句子
    - 方法二：机器识别系统：当一个字符串能被一个语言的识别系统接受，则这个字符串是该语言的一个句子，否则不属于该语言。

文法的形式定义

- **文法G是一个四元组G=(V_N, V_T, S, £)，其中**

  V_N：非终结符的有限集合

  V_T：终结符的有限集合，$V_N \cap V_T = \Phi$

  S：开始符号且S$\in V_N$。

  £：形式为 $P \rightarrow \alpha$的产生式的有限集合，

  且$P \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$，$\alpha \in (V_N \cup V_T)^*$

文法示例

- 文法G_1 =({N}, {0, 1},N, {N→0N, N→1N, N→0, N→1})

  其中：V_N = {N}，[非终结符的有限集合]

  V_T = {0, 1}，[终结符的有限集合/字母表]

  S = N，[起始符]

  £ = {N→0N, N→1N, N→0, N→1} 。[产生式]

文法的解释（1）

- **终结符号V_T：** 组成语言字母表中的基本符号，属于个体记号
- **非终结符号 V_N：** 需要进一步定义的符号
  - 用来代表语法范畴。如“算术表达式”、“布尔表达式”、“过程”等。
  - 一个非终结符代表一个一定的语法概念，因此非终结符是一个类（或集合）记号，而不是个体记号。
- **开始符号S**属于非终结符号，至少在某个产生式的左部出现一次。

文法的解释（1）

- **产生式£（规则）：定义语法单位的一种书写规则**
  - 它的形式为：P→ α （或P::= α ）

    其中“→”读作“定义为”， P，α为符号串，箭头左边称为产生式左部，箭头右边称为产生式右部。

    例：S→0S1，0S→01
  - 若干个左部相同的产生式如 $P\rightarrow\alpha_1$, $P\rightarrow\alpha_2$, $P\rightarrow\alpha_3$, ... , $P\rightarrow\alpha_n$ 可合并成一个,缩写为

    $P\rightarrow\alpha_1|\alpha_2|\alpha_3|...|\alpha_n$,

    其中,每个α_i 称为是P的一个候选式.

习惯写法

- V_N:大写字母A、B、C、S等
- V_T: 小写字母，0~9，+、 - 等运算符，
- α、β、γ:文法符号串,$\in(V_T \cup V_N)^*$
- S:开始符号，第一个产生式中出现
- → :定义符号（推出/定义为）
- |: 或

文法的约定

- 为了简化，文法表示
  - 只写出产生式部分
  - 约定第一个产生式的左部符号为初始符号

    或

    在产生式前写上“G[A]”，其中G为文法名，A为初始符号.

例. 文法G[N]: N→0N, N→1N, N→0, N→1

文法G[E]: E→E+E | E*E | (E) | i

如何由文法产生语言的句子？

- **基本思想**：从识别符号开始，把当前产生的符号串中的**非终结符号**替换为相应产生式右部的**符号串**，直到最终全由**终结符号**组成。这种替换过程称为推导或产生句子的过程，每一步称为**直接推导**或**直接产生**。

直接推导与归约

- 直接推导
  - 如果A→γ是一个产生式，而α，β∈（$V_T \cup V_N$）*，则将产生式A→γ用于符号串αAβ得到符号串αγβ，记为αAβ ⇒ αγβ，称**αAβ直接推出αγβ**。
- 归约是推导的逆过程
  - 若存在αAβ ⇒ αγβ ，则称**αγβ能够直接归约成αAβ** 。

推导

- 推导：设$\alpha_1,\alpha_2,\ldots,\alpha_n\ (n>0)\in(V_T\cup V_N)^*$，且有

  $$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$$

  则称这个序列是从α_1到α_n的一个推导。
- 若存在一个α_1到α_n的一个推导，则称α_1可推导出α_n。

- $\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ (经一步或若干步推导)

- $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ (经0步或若干步推导)

最左（右）推导（规约）

- 若在推导关系中，每次最先替换最左（右）的非终结符，则称为**最左（右）推导**；
- 若在归约过程中，每次最先归约最左（右）的非终结符，则称为**最左（右）归约**。

- 例. 文法G[S]：S→AB　A→A0|1B　B→0|S1，请给出句子101001的最左和最右推导。

最左推导：

$S \Rightarrow AB$
$\Rightarrow 1BB$
$\Rightarrow 10B$
$\Rightarrow 10S1$
$\Rightarrow 10AB1$
$\Rightarrow 101BB1$
$\Rightarrow 1010B1$
$\Rightarrow 101001$

最右推导：

$S \Rightarrow AB$
$\Rightarrow AS1$
$\Rightarrow AAB1$
$\Rightarrow AA01$
$\Rightarrow A1B01$
$\Rightarrow A1001$
$\Rightarrow 1B1001$
$\Rightarrow 101001$

句型、句子和语言

文法G[E]: E→E+E | E*E | (E) | i

- **句型**：假定G是一个文法，E是它的开始符号，如果$E \overset{*}{\Rightarrow} \alpha$，则称α是文法G的一个句型。

  例 (E+E), (i+E), (i+i), E 都是G[E]的句型。

- **句子**：仅由终结符组成的句型称为句子。

  例 (i×i+i), (i+i) 都是G[E]的句子。

- **语言**：文法G所产生句子的全体，即：

  $L(G) = \{\alpha | S \overset{+}{\Rightarrow} \alpha，\ \alpha \in V_T^*\}$

例. 设有文法G1[S]: S →bA，A→aA | a

试求此文法所描述的语言。

解：因为从开始符号S出发可推出下列句子：

$S \Rightarrow bA \Rightarrow ba$

$S \Rightarrow bA \Rightarrow baA \Rightarrow baa$

$S \Rightarrow bA \Rightarrow baA \Rightarrow baaA \Rightarrow baaa$

…

$S \Rightarrow bA \Rightarrow baA \Rightarrow \ldots \Rightarrow baa\ldots a$

所以，$L(G1)=\{ ba^n \mid n>=1\}$

例.设文法G2[S]： $S \rightarrow AB$

$A \rightarrow aA|a$

$B \rightarrow bB|b$

该文法所描述的语言是什么?

解：从文法开始符号S出发，可推出下列句子：

$S \Rightarrow AB \Rightarrow ab$

$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaB \Rightarrow aab$

$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaB \Rightarrow aabB \Rightarrow aabbB \Rightarrow aabbb$

$S \Rightarrow AB \Rightarrow \underbrace{aa\ldots a}_{m}\underbrace{bb\ldots b}_{n}$

所以，$L(G2)=\{a^m b^n \mid m,n>=1\}$

例. 试对如下语言L(G3) = {$a^n b^n$ |n>=1}构造文法G3。

解： L(G3)由以下一些符号串x组成：

n=1，x=ab

n=2, x=aabb

n=3, x=aaabbb

…

L(G3)={ab,aabb,aaabbb,

…

由此可见，集合L（G3）有如下特点：

- 每个符号串呈对称形式，即a,b成对出现；
- L(G3)为无穷集合，描述它的规则中含递归定义；
- 文法中终结符只有a,b。

因此可用以下两条产生式定义语言L(G3)

S→ aSb|ab

即：G3=（{S}，{a,b}，S，{S→ aSb|ab})

同一句型有不同的推导序列

- 设有文法G[N_1]: $N_1 \rightarrow N$　　$N \rightarrow ND|D$
  $D \rightarrow 0|1|2$

  则句子12可由若干种不同的推导序列推导出来:

  (1) $N_1 \Rightarrow N \Rightarrow ND \Rightarrow N2 \Rightarrow D2 \Rightarrow 12$

  (2) $N_1 \Rightarrow N \Rightarrow ND \Rightarrow DD \Rightarrow 1D \Rightarrow 12$
- 可见，同一句型（句子）可以通过不同的推导序列推导出来。

随堂练习

- P36
  - 第6题
    - 令文法G_6为

      N→D|ND

      D →0|1|2|3|4|5|6|7|8|9

      (1) G_6的语言$L(G_6)$是什么?

      (2) 给出句子0127, 34和568的最左推导和最右推导
  - 第8题
    - 令文法为

      E→T|E+T|E-T

      T→F|T*F|T/F

      F →(E)/i

      (1) 给出i+i*i, i*(i+i)的最左推导和最右推导

      (2) 给出i+i+i, i+i*i和i-i-i的语法树

Chomsky文法体系(回顾)

- 乔姆斯基Chomsky文法体系中，任何一种文法必须包含
  - 两个不同的有限符号的集合
    - 非终结符集合V_N
    - 终结符集合V_T
  - 一个起始符S
  - 一个形式规则的有限集合£ （产生式集合）。
- 注意
  - £中的产生式是用来产生语言句子的规则，而句子则是仅由终结符组成的字符串。
  - 这些字符串必须从一个起始符S开始，不断使用£中的产生式而导出来。
  - 文法的核心是产生式的集合，它决定了语言中句子的产生。

Chomsky文法体系分类

- 文法G=(V_N, V_T, S，£)，£：P→α，其中

  $P \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$，$\alpha \in (V_N \cup V_T)^*$属于Chomsky文法体系

- 该体系对产生式的形式做了一些规定，分为四类，即0型、1型、2型、3型文法

0型文法

- 0型文法：无限制文法，短语文法
  - 对应的语言：递归可枚举语言
  - 与图灵机等价。
- 例.下列文法是0型文法

  $S \rightarrow aBC|aSBC$

  $CB \rightarrow BC$

  $aB \rightarrow ab$

  $bB \rightarrow bb$

  $bB \rightarrow b$

  $bC \rightarrow bc$

  $cC \rightarrow cc$

  $cC \rightarrow c$

1型文法

- 也称上下文有关文法（CSG：Context-sensitive Grammar)
- 产生式的形式为$P \rightarrow \alpha$, 其中

  $|P| \leq |\alpha|$， $\alpha \in (V_N \cup V_T)^*$ ， $P \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$
  - 对应的语言：上下文有关语言（CSL：Context-sensitive Language)
  - 若不考虑ε，与线性有界自动机（LBA, Linear Bounded Automaton）等价。
  - 举例：$\alpha A \beta \rightarrow \alpha \gamma \beta$

1型文法示例

- 例.下列文法是1型文法

  $S \rightarrow aBC|aSBC$

  $CB \rightarrow BC$

  $aB \rightarrow ab$

  $bB \rightarrow bd$

  $bC \rightarrow bc$

  $cC \rightarrow ce$

2型 文法

- 也称上下文无关文法（CFG：Context-free Grammar)
- 产生式的形式为$P \rightarrow \alpha$, 其中

  $P \in V_N$，且$\alpha \in (V_N \cup V_T)^*$

  - 对应的语言：上下文无关语言 （CFL： Context-free Language)
  - 对应的自动机：下推自动机（PDA： Pushdown Automaton)。

2型文法示例

- **例. 上下文无关文法:**

  $S \to 01$

  $S \to 0S1$

  产生的语言是 $L = \{ 0^n1^n \mid n \geq 1 \}$,

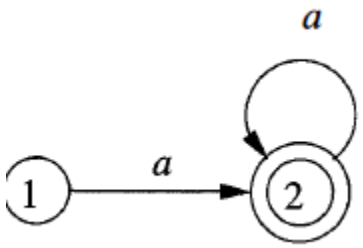
  如 $0011, 000111, 01 \in L$, **而** $10, 1001, \varepsilon, 010 \notin L$.

  但没有任何有限自动机能够接受语言L.

3型 文法

- 也称正规文法
  - 右线性文法（Right-linear Grammar）：任何产生式为 $A \rightarrow \omega B$ 或 $A \rightarrow \omega$，其中 $A、B \in V_N$, $\omega \in V_T^*$。
  - 左线性文法（Left-linear Grammar）：任何产生式为 $A \rightarrow B\omega$或 $A \rightarrow \omega$，其中 $A、B \in V_N$, $\omega \in V_T^*$。
  - 等价于正规式
  - 对应的语言：正规语言
  - 对应的自动机：有限自动机（Finite Automaton）。
- 例.文法 $S \rightarrow aS$，$S \rightarrow a$

  对应正规式：a^+，或者a^*a



思考：对于右线性文法，画出其有限自动机

例. 设有文法G1[S]: S →bA，A→aA | a

试求此文法所描述的语言。

解：因为从开始符号S出发可推出下列句子：

$S \Rightarrow bA \Rightarrow ba$

$S \Rightarrow bA \Rightarrow baA \Rightarrow baa$

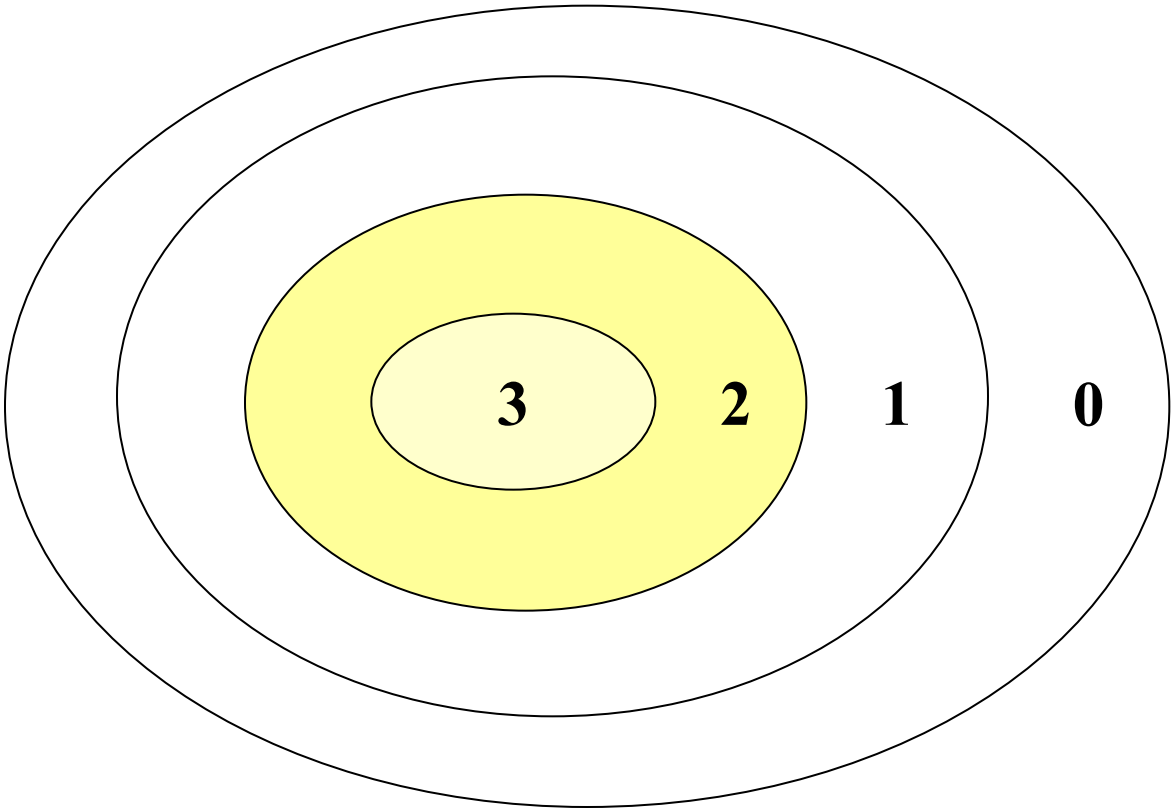
$S \Rightarrow bA \Rightarrow baA \Rightarrow baaA \Rightarrow baaa$

…

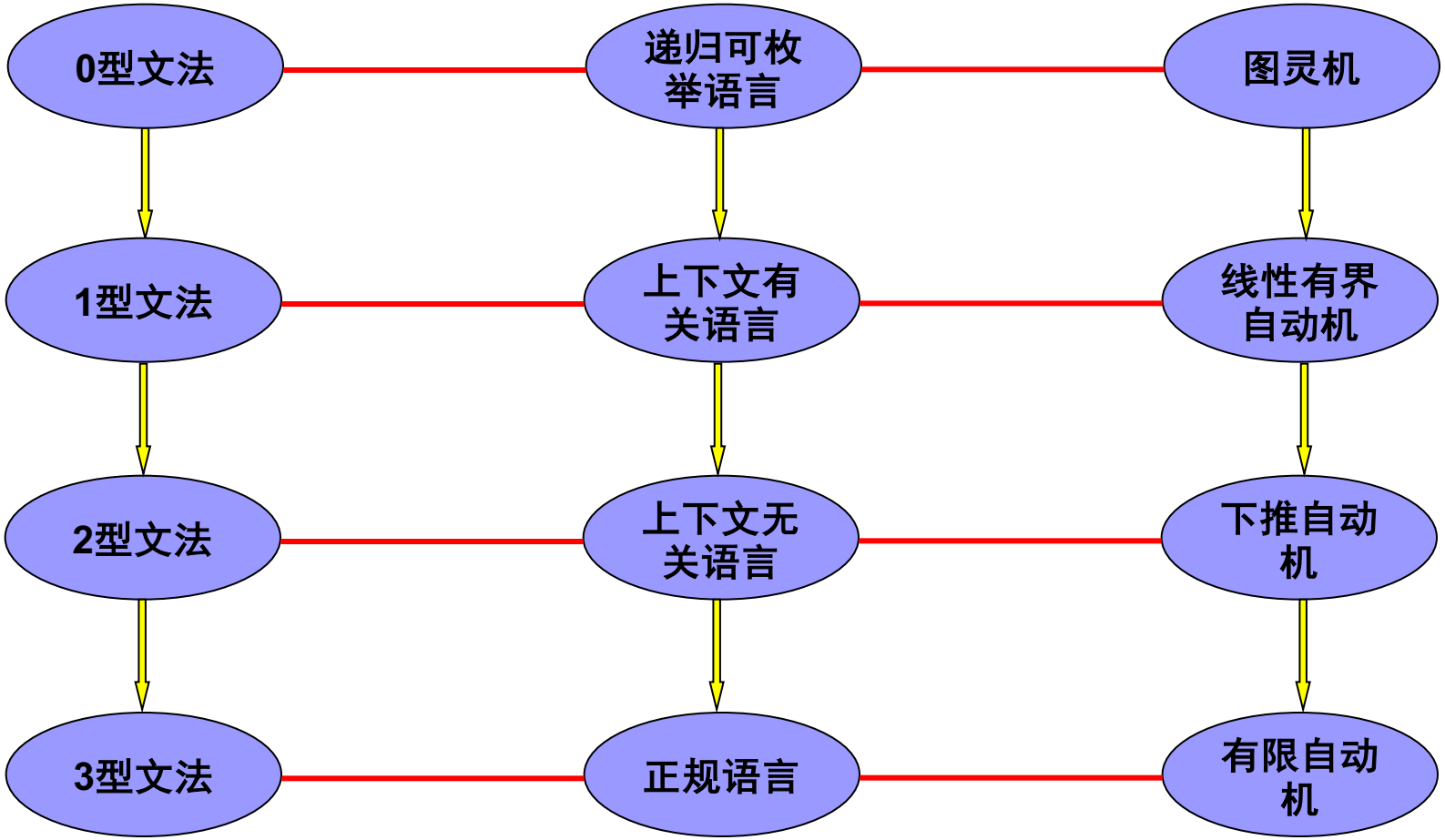
$S \Rightarrow bA \Rightarrow baA \Rightarrow \ldots \Rightarrow baa\ldots a$

所以，$L(G1)=\{ ba^n \mid n>=1\}$

Chomsky文法体系



文法、形式语言和自动机的对应关系



总结（1）

- 自然语言是上下文有关的。
- 但是上下文无关文法有足够的能力描述现今多数程序设计语言的语法结构
  - 算术表达式
  - 语句
    - 赋值语句
    - 条件语句
    - ……

上下文无关文法描述语言语法结构

- **算术表达式 的上下文无关文法表示**

  文法G=({E}, {+, *, i, (,)}, P, E}

  P: E → i　　E → E+E

  E → E*E　　E → (E)

- **条件语句的上下文无关文法表示**

  <条件语句>→if<条件表达式>then<语句>

  |if<条件表达式>then<语句>else<语句>

  ……

总结（2）

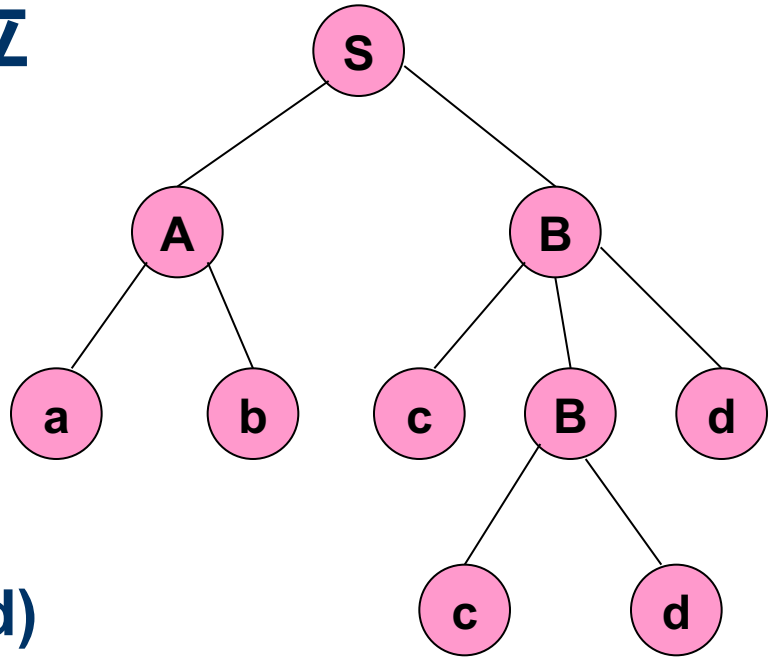
- 基于**正规文法**讨论词法分析问题
- 基于**上下文无关文法**讨论语法分析问题
- 用上下文无关文法和正规文法描述语言语法
- 而其中上下文有关的问题，可通过表格处理解决

语法分析树和文法的二义性

- 语法分析树，简称语法树
- 二义性

什么是语法树？

- 语法树是表示一个**句型推导过程**的图，是一棵倒立的树。
  - 结点
  - 边
  - 根结点
  - 末端结点
  - 末端分支：A(a,b)和B(c,d)
  - 兄弟结点



从推导构造语法树

- 方法：把开始符号S做为语法树的根结点，对每一个直接推导画一次结点的扩展，该结点是直接推导中被替换的非终结符号，其所有儿子结点所组成的符号串是推导中所用产生式右部。直到推出句型或句子或无法再推导时结束。

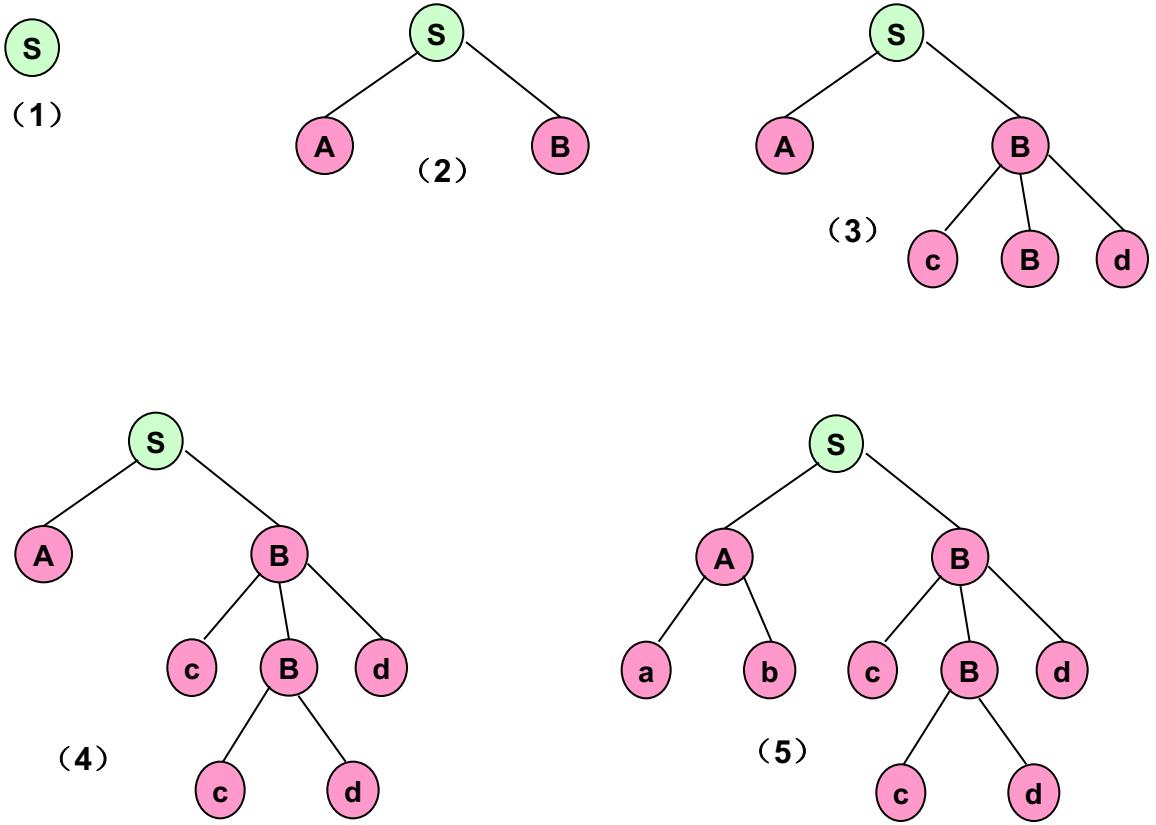
语法树的构造过程

$S \Rightarrow AB$

$\Rightarrow AcBd$

$\Rightarrow Accdd$

$\Rightarrow abccdd$



语法树的特征

- 给定文法G，G=(V_N,V_T, S，£)，对于G的任何句型都能构造与之关联的语法树（推导树）。这棵树具有下列特征：

  1、根结点的标记是开始符号S；

  2、每个结点的标记都是V中的一个符号；

  3、若一棵子树的根结点为A，且其所有儿子结点的标记从左向右的排列为$A_1A_2…A_R$，那么 $A \rightarrow A_1A_2..A_R$一定是£中的一条产生式（规则）；

  4、若一标记为A的结点至少有一个儿子，则$A \in V_N$

  5、树的叶结点符号所组成的符号串W就是所给句型；若w中仅含终结符号，则w为文法G所产生的句子。

■例. 文法 G[E]：

E→E+T | T

T→T*F | F

F→(E) | i

句型E+F*i推导：

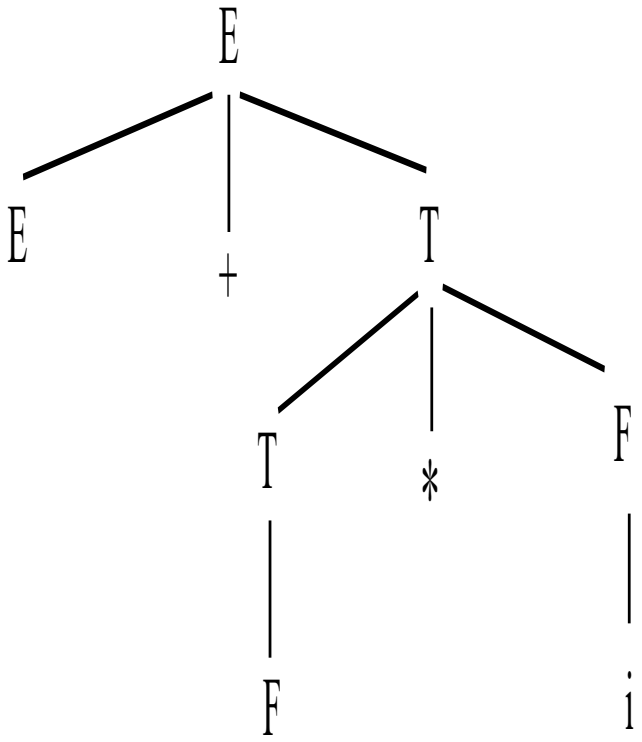
E⇒ E+T

⇒ E+T*F

⇒ E+F*F

⇒ E+F*i

对应的语法树如右图



语法树解释

- 语法树表明了在推导过程中使用了哪条规则和使用在哪个非终结符号上，但它并没有表明**使用规则（产生式）的顺序**。
- 一个句型是否只对应唯一的一棵语法树呢？
  - 不一定
- 一个句型是否只有唯一的一个最左(最右)推导呢？
  - 不一定

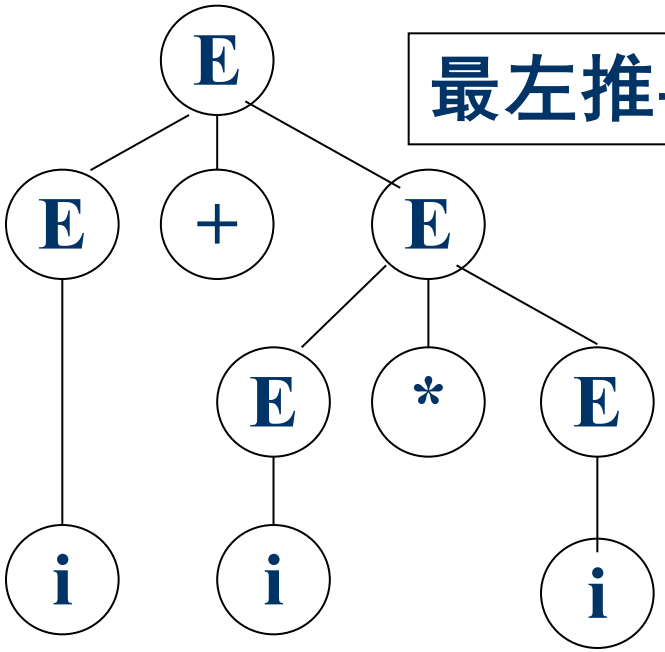
二义性

- 如果一个文法的句子（句型）存在两棵语法树,那么,该**句子是二义性的**。
- 如果一个文法包含二义性的句子,则称这个**文法是二义性的**; 否则，该文法是无二义性的。

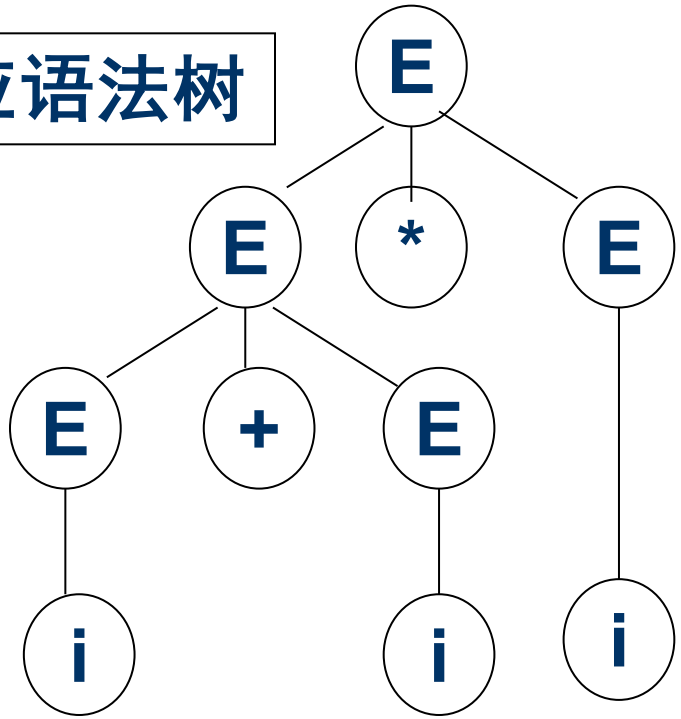
文法G[E]：E→E+E｜E*E｜(E)｜i

E⇒E+E ⇒i+E
⇒i+E*E ⇒i+i*E
⇒i+i*i

E ⇒E*E⇒E+E*E
⇒i+E*E⇒i+i*E
⇒i+i*i



最左推导及相应语法树



同理，该句子的最右推导及相应语法树也不同

二义文法改造为无二义文法

- 文法G[E]：E→E+E｜E×E｜(E)｜i 是二义性的，如果规定“×”和“+”的优先性，并服从左结合，上式就可以构造出无二义性文法。

文法G[E]:

$$E \rightarrow T \mid E+T$$

$$T \rightarrow F \mid T \times F$$

$$F \rightarrow (E) \mid i$$

句子的推导过程唯一：如 (i×i+i)

规定

- 作为描述程序语言的上下文无关文法，对其如下限制：

(1) 文法不含产生式 $P \rightarrow P$；

(2) 每个非终结符P都有用，即每一$P \in V_N$，必须都有用处：

1) $S \overset{*}{\Rightarrow} \alpha P \beta$，P在句型中出现

2) $\gamma \in V_T^*$， $P \overset{+}{\Rightarrow} \gamma$，即对P不存在不终结的回路。

随堂练习

- P36 第9题
  - 证明下面的文法是二义的
    - S → **iSeS | iS | i**

课外自学

- 第一章 引论
  - 1.3 编译程序的结构
  - 1.4 编译程序与程序设计环境
  - 1.5 编译程序的生成
- 第二章 高级语言及其语法描述
  - 2.1 程序语言的定义
  - 2.2 高级语言的一般特性

Canvas测试

- **乔姆斯基文法**

Dank u
Dutch

Merci
French

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

감사합니다
Korean

Tack så mycket
Swedish

धन्यवाद
Hindi

תודה רבה
Hebrew

Obrigado
Brazilian Portuguese

谢谢
Chinese

Thank You !

Dankon
Esperanto

ありがとうございます
Japanese

Trugarez
Breton

Tak
Danish

Danke
German

Grazie
Italian

நன்றி
Tamil

děkuji
Czech

ขอบคุณ
Thai

go raibh maith agat
Gaelic