编译原理

田玲 教授、博导

lingtian@uestc.edu.cn

第三章 控制结构

主要讨论语言中描述算法的机制，即**控制结构**。主要讨论各种**语句级控制结构**和**单元级控制结构**。

对程序单元执行的控制。

第一节 语句级控制结构

❑语句级控制结构分为三种：
- ✓ 顺序（sequencing）
- ✓ 选择（selection）
- ✓ 重复（re

```
begin
  S1; S2; …; Sn
end
```

❑顺序
- ✓ 顺序运算符 “;”
- ✓ 在ALGOL和PASCAL中，用语句括号begin . . . end 把多个语句组成一个单独的语句，称为**复合语句**
- ✓ 在fortran中，以行结束符分隔语句

第一节 语句级控制结构

- 选择
  - ① if语句
  - ✓ 一般形式：
    if 条件 then 语句1 else 语句2
  - ✓ 选择结构引起二义性
    if x>0 then if x<10 then x:=0 else x:=1000

可省略

ALGOL 68中if语句的结束符号fi；Ada用end if；例如：
if x>0 **then if** x<10 **then** x:=0 **else** x:=1000 **fi fi**
if x>0 **then if** x<10 **then** x:=0 **fi else** x:=1000 **fi**

第一节 语句级控制结构

② 多重

✓

S

例：PASCAL的case语句

```
var operator:char;
operand1,operand2,result:boolean;
……
case operator of
        '.'  : result:=operand1 and
operand2;
       '+'  : result:=operand1 or operand2;
       '='  : result:=operand1 = operand2;
end
```

✓ 多种语言的case语句：

- PASCAL
- ALGOL 68
- Ada

第一节 语句级控制结构

- ✓ 不同语言case语句的差异：
  - ALGOL 68中，case语句基于整表达式的值，表达式的值为i，则选择第i个分支执行
  - PASC
    的值相
    现的次序是无关紧要的。
  - Ada结合ALGOL68和PASCAL；
  - 表达式的值不在显式列出的值中时的处理：**out**和**otherwise**；

1.选择分支采用枚举类型或整数类型的表达式；
2.在选择中要提供所有判别表达式类型的所有可能值；

PASCAL语言的用法

ALGOL 68语言的用法

第一节 语句级控制结构

✓ Dijkstra选择结构：

if B1→S1

□B2→S2

□B3→S3

……

□BN→SN

fi

其中，Bi是布尔表达式，称为卫哨。若有多个卫哨为真时执行任一Si。

□Dijkstra选择结构的最大特性是对非确定性的抽象

例如计算A和B的最大值，可写成

if A≤B MAX:=B

□A≥B MAX:=A

fi

第一节 语句级控制结构

- 重复（循环）

①计数

当预先

上重复。

✓FORT

✓Pascal的for语句

- 计

```
type day=(sun,mon,tus,wed,thu,fri,sat);
var week_day:day;
   ……
for week_day=mon to fri do …
```

- 复的值可在任何有序集上

- 在

FORTRAN的DO循环中,

```
DO 7 I=1,  10
   A (I) =0
   B (I) =0
7 CONTINUE
```

升序，计数器增加

降序，计数器减少

第一节 语句级控制结构

- 重复（循环）

②条件制导

当预先不知道
条件通常是一
✓PASCAL的
- while
- repe

✓ALGOL 6

➢重复说明可以是：
while <条件>
或 for <计数变量> in <离散范围>
或 for <计数变量> in reverse <离散范围>
➢可由exit或exit when<条件>终止循环
➢Exit退出所有循环，continue退出当前循环。

例：
repeat
“从文件f中读一个项”；
“处理这个项”
until eof(f);

begin
“从文件f中读一个项”；
“处理这个项”
end

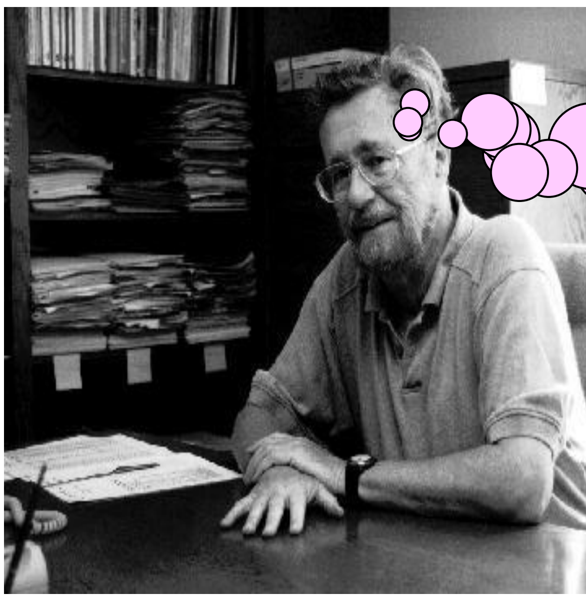
pp前加重复说明）

计算机科学先驱Edsger Wybe Dijkstra

- 1930年出生在荷兰鹿特丹市。
- Dijkstra曾是开发Algol的委员会成员。
- 编写了第一个Algol60编译器。
- 他发明或帮助定义的概念包括结构化编程、堆栈、向量、信号量和同步过程。

第一节 语句级控制结构

- 1968年，提出并写下了一篇著名的论文：《GoTo语句是有害的》。
- 1972年，Dijkstra荣获美国计算机协会的图灵奖。
- 2002年8月6日，去世，享年72岁。



简单是可靠的先决条件。

第二节 单元级控制结构

❑单元级控制结构:规定程序单元之间控制流程的机制

❑讨论四种单元

- ✓ 显式调用
- ✓ 异常处理
- ✓ 协同
- ✓ 并发

各单元组成一组并发或并行单元或进程，彼此之间不存在调用和返回，而是并行执行。

名字调用从属单元；从属单元执行完返回调用单元；比如子程序。

被调用单元是隐含的。非显式的调用。比如异常处理程序

第二节 单元级控制结构

1.显式调用从属单元：

- ✓ 调用方式

  由调用语句使用被调用单元的名字来进行调用；调用语句将控制转向被调用单元，被调用单元执行完后，将控制返回

- ✓

例：

subprogram S(F1,F2,…,FN);

……

end

- 位置绑定: **call** S(A1,A2,…,AN)

  call S(A1,,A3,,,,,A8,,A10)

- 关键字绑定:

  call S(A1=>F1,A3=>F3,A8=>F8,A10=>F10)

第二节 单元级控制结构

- ✓ **副作用**:对非局部环境的修改
- ✓ **副作用**可能导致的问题
  - ① 副作用降低
  - ② 副作用限制了数
  - 如:w:=x+f(x,y)+z
  - ③ 副作用影响目标代码的优化
  - 如:u:=x+z+f(x,y)+f(x,y)+x+z

如果对f(x,y)的调用，修改了x和y的值，则f(x,y)+x和x+f(x,y)两个表达式的值可能不同

如果对f(x,y)的调用，修改了x和y的值，则前面的x+z的值和后面的x+z的值可能不同，因此不能对它们提公因子（即只计值一次）

第二节 单元级控制结构

- ✓ **别名**:在单元激活期间,两个变量表示(共享)同一数据对象

例如:

① FORTRAN 的 EQUIVALENCE 语句, EQUIVALENCE(A,B)

② Pascal的变参使得形参和实参共享同一数据对象

注意:别名可能导致严重的程序问题。

例:考虑如下pascal过程

```
procedure swap(var x,y:integer);
begin
        x:=x+y;
        y:=x-y;
        x:=x-y;
end;
```

问题

在下列几种情况下,左边的程序是否正确执行:

swap(a,a);

swap(b[i],b[j]),其中i=j

swap(p^,q^),p和q指向同一个数据对象

第二节 单元级控制结构

- ✓ 变参和全局变量表示同一数据对象时，也会引起别名

```
procedure swap(var x:integer);/* a是全局变量
begin
        x:=x+a;
        a:=x-a;
        x:=x-a;
end;
```

调用swap(a)?
当形参引用调用实参时，它与全局变量表示同一数据对象，或者有重叠的数据对象时，引起别名。

- ✓ 别名也影响编译器生成优化的代码

```
a:=(x-y*z)+w    /* 若a与x、y或z中任一个是别名
b:=(x-y*z)+u
```

- ✓ 别名的消除
  - 废除可能引起别名的结构
  - 限制使用指针、变参、全局变量、数组等

第二节 单元级控制结构

2.隐式调用单元：

✓ 调用方式

隐式地将控制从一个单元转向到另一个单元，通常用于异常处理。

定义 **异常是指导致程序正常执行中止的事件，要靠发信号来引发，用异常条件来表示,并发出相应的信号，引发相应的程序。**

? 异常处理要考虑的问题

5）处理异常之后，控制流程转向何处?

第二节 单元级控制结构

- PL/1语言的异常处理机制
  - ① 异常的说明
  - ② 异常的引
  - ③ 语言预定义
  - ④ 异常名可多
  - ⑤ 遇到一个C 立绑定关系；
  - ⑥ 当自动或由 行绑定于该异常
  - ⑦ 可用“NO

ON <条件> <异常处理程序>

SIGNAL <条件>

例如除零异常 ZERODIVIDE

(NOZERODIVIDE) BEGIN

……

END

使得ZERODIVIED异常在该语句、分程序或过程中失效

第二节 单元级控制结构

- PL/1异常处理的实现模型
  - ① 遇到ON时，就将条件（异常名）与指向相应处理程序的指针保留在当前活动记录的一个表项里；
  - ② 当单元U激活时，为U建立活动记录；
  - ③ 当引发一个异常时，检索ON语句的表项，从最新的表项开始，直到发现为该条件所设置的异常处理程序；
  - ④ 若未发现为该异常设置的处理程序，则执行默认的活动。

第二节 单元级控制结构

- CLU语言的异常处理机制
  - ① 处理方法
    - ✓ 当过程P引发一个异常时，只能将其信号传送给调用
  - ②
  - ③ 异常处
  - ④ 异常处理程序由except语句绑定于语句
  - ⑤ 当处理程序结束时，控制转移到紧跟在附加这个处理程序的语句之后
  - ⑥ 若未找到相应的处理程序，执行特殊异常**failure**

<语句> **except** <处理程序表> **end**
其中，<处理程序表>的形式是
when <异常表1>: <语句1>
……
when <异常表n>: <语句n>

当引发
而异常
成。

第二节 单元级控制结构

- CLU异常处理的实现模型
  - ① 当对一个异常发信号时，控制返回调用者
  - ② 对每一个过程附加一个处理程序表，有以下内容
    - ✓ **由处理程序处理的异常表**
    - ✓ **一对指针，指向处理程序的作用域**
    - ✓ **一个指向处理程序的指针**
  - ③ 在过程P中引发一个异常时，检索调用者的处理程序表，用来确定返回点

第二节 单元级控制结构

- Ada语言的异常处理机制
  - ① Ada预定义了一些异常；
  - ② 用户也可以自己定义，异常的说明类似于变量的类型说明
  - ③ 程序单元[illegible]引发异常
  - ④ 例如：[illegible]OR:**exception**;
  - ⑤ **raise** <异常名>;
  - ⑥ [illegible]异常

例如：

```
begin ...;
exception when HELP=>...;
          when DESPERATE=>...;
end;
```

第二节 单元级控制结构

3.SIMULA 67语言协同程序

定义 **实现两个或两个以上程序单元之间交错执行的程序称为协同程序。**

- SIMULA 67的协同程序是一个类实例
- 当遇到new时，建立类的一个新实例，并
-

例如：
设有程序单元C1和C2，C1先开始执行，
当执行到C2单元类的new C2"语句时，

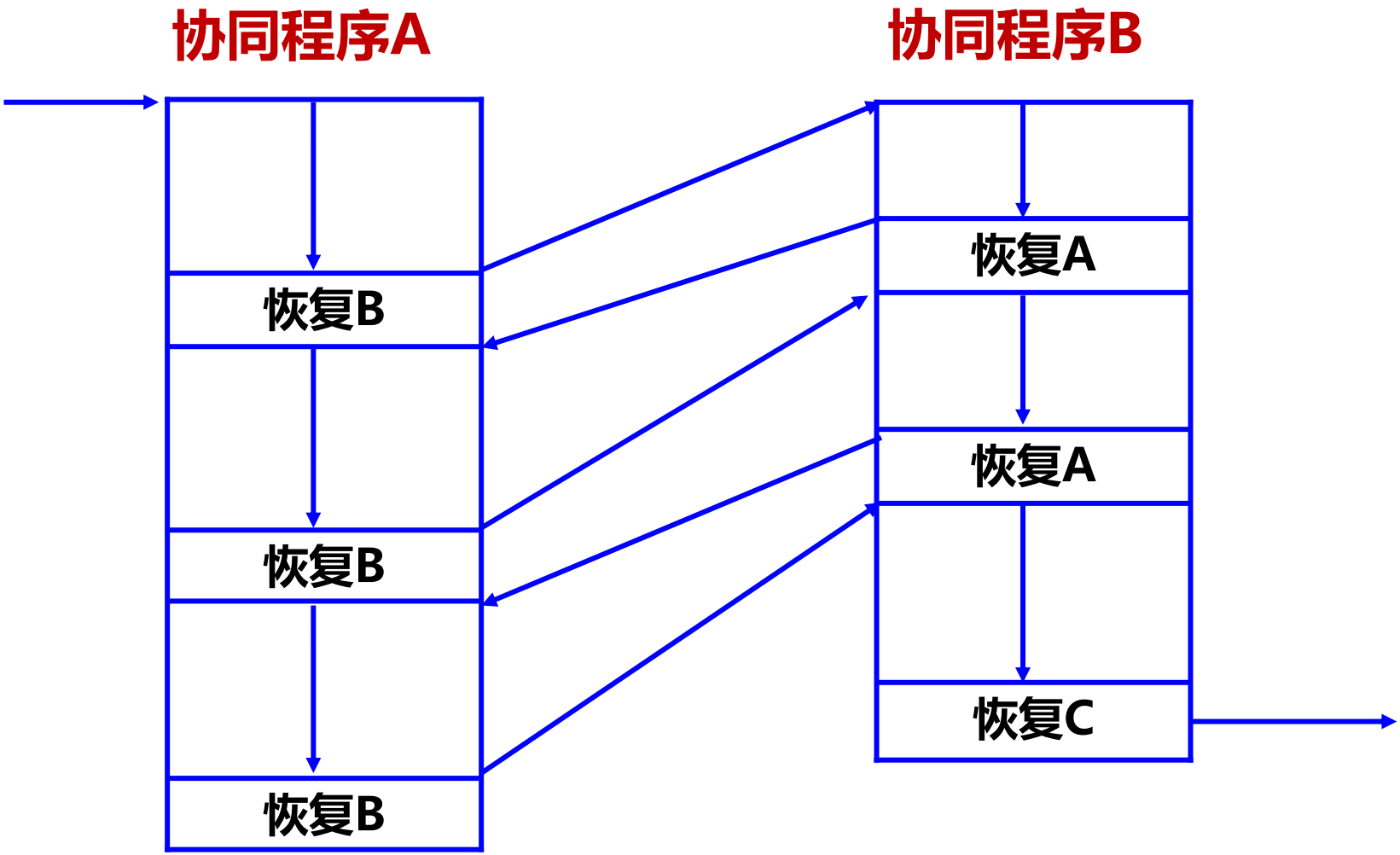
一般形式为：

```
class 类名(参数);
begin 说明;
    语句;
    detach;
    语言表2
end
```

若设类x, 变量y1和y2是对x的引用, 那么可写成:

```
y1:-new x(...);
y2:-new x(...);
```

第二节 单元级控制结构



协同程序间的控制转移关系

补充知识

- **进程**：一个具有一定独立功能的程序在一个数据集合上的一次动态执行过程。

- **进程的特点**：
  - ✓ 动态性：进程具有动态的地址空间
  - ✓ 独立性：各进程的地址空间相互独立，除非采用进程间通信手段，
  - ✓ 并发性、异步性
  - ✓ 结构化

❑ 进程与程序的区别

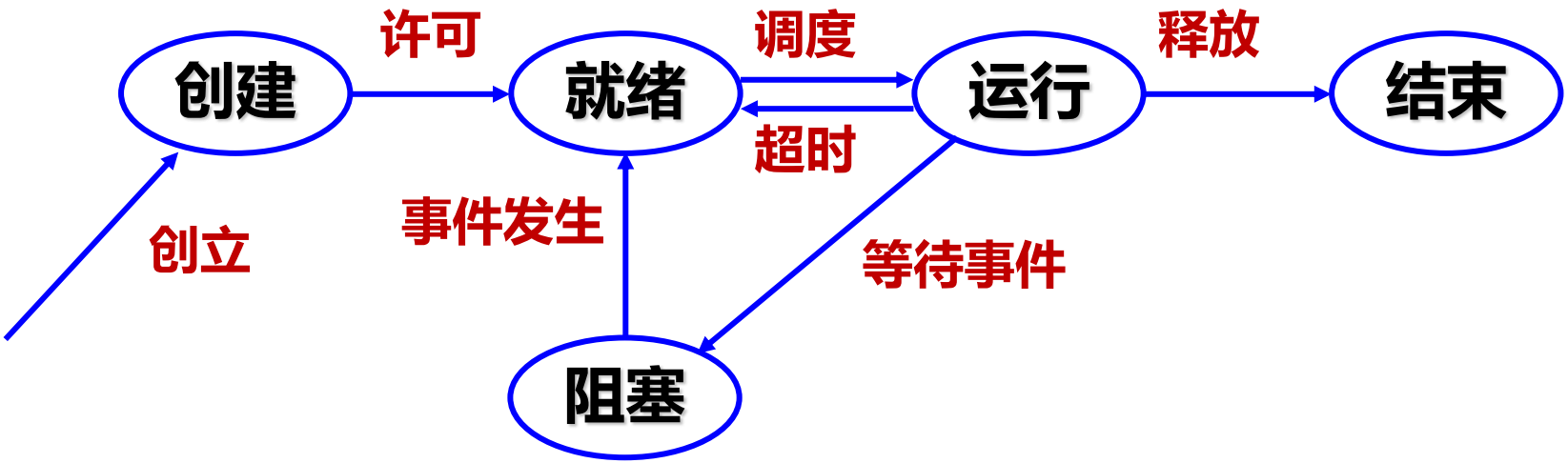
- ✓ 进程是动态的，程序是静态的：程序是有序代码的集合；进程是程序的执行。
- ✓ 进程是暂时的，程序的永久的：进程是一个状态变化的过程，程序可长久保存。
- ✓ 进程与程序的组成不同：进程的组成包括程序、数据和进程控制块（即进程状态信息）。
- ✓ 进程与程序的对应关系：通过多次执行，一个程序可对应多个进程；通过调用关系，一个进程可包括多个程序。

第二节 单元级控制结构

❑ 进程的状态

第二节 单元级控制结构

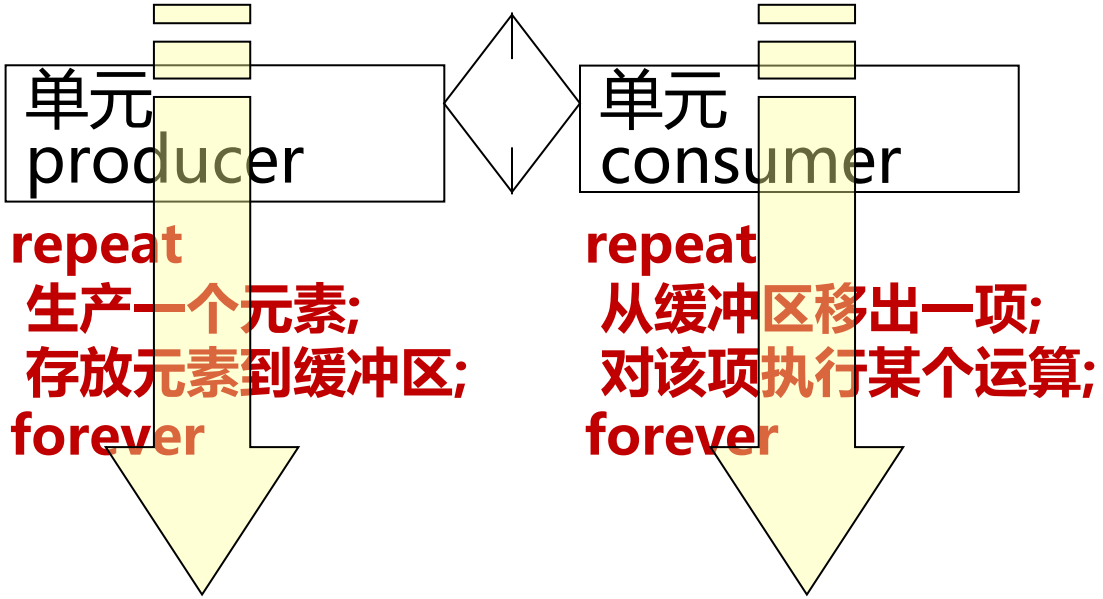
❑ 进程的五状态模型



第二节 单元级控制结构

4. 并发单元

例：**生产者-消费者问题**
生产者生产一系列的“值”，并依次将它们存放在某个缓冲区中；消费者以生产者相同的次序从缓冲区中移出这些值。



? 同步问题

如何正确访问缓冲区？即：不会向已满的缓冲区写数据,不会从空缓冲区读数据。

第二节 单元级控制结构

❑ 动作的“不可分”与“可分”

设t表示所存项目总数,append是生产者向缓冲区存数的操作, remove是消费者从缓冲区取数的操作,这两个操作都要修改t的值,而(1)t:=t+1和(2)t:=t-1来实现

- ✓ 读t到
- ✓ 更新
- ✓ 将专

结论：（1）和（2）的执行不能被打断，就像不可分的操作一样，即（1）和（2）必须互斥执行

+1或m-1

若以上的操作是可分的（即可被打断执行），那么如果t的值为m,那么执行一个append和一个remove之后t的值可能是多少?

? 问题

第二节 单元级控制结构

- 进程的**并发性**:诸进程的执行概念上是可重叠的（即正在执行的进程尚未终止，另一个进程可能开始执行）
- 进程**竞争**：进程之间能过竞争以得到共享的资源
- 进程的**合作**：进程之间通过合作以达到共同的目标

注意：语言为了实现进程之间的同步，需要提供同步语句（或称原语）以实现进程之间的通信，基本的同步机制有信号灯、管程和会合

第二节 单元级控制结构

❑ 信号灯

定义：信号灯是一个数据对象,该数据对象采用一个整数值,并可用**原语P**和**V**对它进行操作。信号灯在说明时,以某个整数值对它初始化。

P、V操作的定义：

P(s): if s>0 then s:=s-1
else 挂起当前进程

V(s): if 信号灯上有挂起的进程
then 唤配进程
else s:=s+1

原语P、V是不可再分的原子操作

第二节 单元级控制结构

注意：在访问共享资源前,不能忘记执行一次P操作;释放资源时不要忽略执行一次V操作

❑ 信号灯的弱点

- ✓ 简单低级，难设计，难理解
- ✓ 不能作静态检查
- ✓ 容易出错，容易导致死锁

第三章习题

思考题:

3-1、3-2、3-3、3-4