第八章　运行时数据区的管理

廖力
东南大学

引言

一、数据的存储单元

- 编译程序必须为目标程序的运行分配数据的存储单元。
  - 如：变量、常量单元，临时工作单元，返回地址。
  - 若无存放数据信息的单元，则目标程序将无法运行。

引言

二、存储单元分配策略

1、静态存储分配

- 在编译时就可以完全为数据项目分配存储单元，称为静态存储分配。
- 注：若一个程序设计语言不允许递归调用，而且不含有可变数组，则可使用静态存储分配策略。

引言

二、存储单元分配策略

2、动态存储分配

- 在运行时才能进行数据存储单元分配，称为动态存储分配。
- 注：1)若某程序设计语言允许过程递归调用，而且允许使用可变数组，那么在编译时就不可能完全为其数据项目分配存储单元，必须采取动态存储分配策略。
- 2)动态分配数据单元时一般使用：
  - 栈式存储分配
  - 堆式存储分配

引言

二、存储单元分配策略

2、动态存储分配

（1）栈式存储分配

- 运行时，每进入一个过程，就在栈顶为该过程分配一块数据区，一旦退出该过程，它所占的空间也退还给系统。

（2）堆式存储分配

- 有些语言允许用户随时动态地申请和释放存储空间，但申请和释放之间不遵守先申请后释放或后申请先释放原则，故不能使用栈式存储分配，而是更复杂的动态分配策略。
- 这种策略是：让运行程序持有一个大的存区（堆），在申请时从堆中取一块，释放时将一块存储区退还给堆。

第一节　静态存储管理

一、基本策略

- 采用相对存储分配策略
  - 在编译时，根据各类数据所需的存储空间大小以及存储方式规定，在符号表中建立“名字-地址”对应关系，然后根据这些对应关系进行变量名的地址分配。

第一节　静态存储管理

一、基本策略

- FORTRAN语言特点
  - FORTRAN语言是块结构语言，它由一个主程序段和若干子程序段组成，既不允许子程序嵌套也不允许过程递归调用。不提供可变长字符串和可变数组。数据名的类型在程序中应加以说明。
  - 因此，其所有数据名的类型和属性在编译时是完全确定的，从而，每个数据名的地址都可静态地进行分配。
- Fortran语言的存储分配策略
  - 1、一般初等数据类型的数据的存放
  - 2、公用语句COMMON的存储分配
  - 3、等价语句EQUIVALENCE的存储分配

第一节　静态存储管理

二、数据区

1、FORTRAN语言目标程序存储结构

- 各程序段独立编译，编译每段程序时，将各变量及其类型等属性填入符号表，再根据符号表计算每个数据占有的尺寸，并在符号表的地址栏分配其相对地址。
- 每段指令代码后面可安排此程序段的局部数据区。上面的相对地址都分配在局部数据区中。
- 若程序中有COMMON语句所定义的公用区，公用区安排在目标程序的最后。

主程序代码	主程序数据	子程序代码1	子程序数据1	子程序代码2	子程序数据2	…	无名区	有名区	…

第一节　静态存储管理

二、数据区

1、FORTRAN语言目标程序存储结构

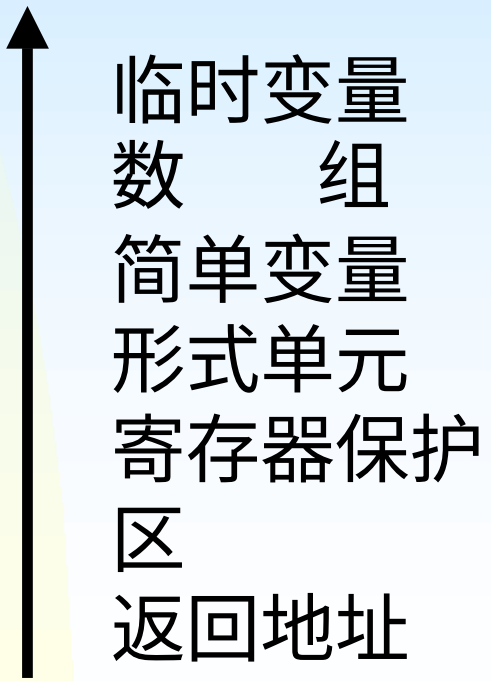
- 符号表中需要设两栏：区号和相对地址
  - 因为编译时只注意统计每个数据区的单元数，对于各区的首地址暂不分配，等到运行时再用“装入程序”将它们连成可运行的整体。
  - 在程序段编译时先填变量的相对地址。区号所对应地址等到运行时由“装入程序”进行处理。

第一节　静态存储管理

二、数据区

2、局部数据区的内容

– 返回地址、寄存器保护区、形式单元、简单变量、数组、临时变量。



第一节　静态存储管理

二、数据区

2、局部数据区的内容

- 1）返回地址：保存调用此程序段的返回地址。
- 2）寄存器保护区：保存调用段留在寄存器中的信息，使这些寄存器在过程体结束返回时重新被使用。

  注：此过程极费时间。
- 3）形式单元：存放实参的地址或值。

  传址：一个实参分配一个形式单元；

  传值：一个实参分配两个形式单元，一个放实参地址，一个放实参值。

第一节　静态存储管理

三、公用语句(COMMON语句)处理

1、COMMON语句的用途:

- 用于在不同程序段间建立共享数据区，以使不同程序段可以使用相同的数据，从而可以节省存储空间。

- 例如：在主程序中写：COMMON X,Y,I,Z(3)
- 　　在子程序中写：COMMON A,B,J,T(3)
  - 则，在无名公用区中X和A,Y和B,I和J,Z和T都分别分配在同一单元中。
- 注：占同一单元的变量在不同程序中可以叫不同名字。

第一节　静态存储管理

三、公用语句(COMMON语句)处理

2、COMMON语句的格式

- 由于只有一个无名公用区，为了实现不同程序使用不同的公用变量，可采用有名公用区。
- COMMON /区名1/变量名表 / /区名2/变量名表 ……
- 注：在同一个程序中出现多条COMMON语句，作用与一条相同。

第一节　静态存储管理

三、公用语句(COMMON语句)处理

3、公用语句的处理：

1）处理思想：

(1)每个公用区(包括无名公用区和有名公用区)中的变量名按先后出现的顺序拉成一条链(不同公用区有不同的链)，待到说明语句读完后，按链的顺序进行存储分配。

(2)所有公用链放在一起形成公用名表(COMLIST)。

第一节　静态存储管理

三、公用语句(COMMON语句)处理

3、公用语句的处理：

1）处理思想：

- 注：(1)COMMON语句是说明语句，所以对它的处理主要是填符号表。
- 　　(2)一个数据名可由若干个说明语句加以说明，对它的语序并没有严格规定，因此，只有读完全部说明语句后才能确定数据名的全部属性。

第一节　静态存储管理
三、公用语句(COMMON语句)处理

3、公用语句的处理：

1）处理思想：

- 注：(3)由于需要拉链，符号表中要加一栏CMP记录连接；又因为各公用元不一定属于同一个公用区，所以符号表中要加一栏DA表示所属公用区。
- 　　(4)建立一个公用名表COMLIST，用来记录公用区信息，如：公用区名、长度、链首位置、链尾位置等。公用名表(COMLIST)是整个程序一张表。

第一节　静态存储管理

三、公用语句(COMMON语句)处理

3、公用语句的处理：

2）语义子程序

- （1）读取公用语句中的变量名，查符号表是否有该变量，无则填入此变量名，将CMP栏填0，表示链尾；看COMLIST中有无此公用区名，无则填入；若有，就不填符号表；
- （2）根据公用区名查COMLIST表中对应区名的FT(链首）是否为空，若为空就将此变量名的符号表入口地址填入FT和LT(链尾)栏中，表示该变量名既处于链首又处于链尾；
- （3）若不空，则将该变量的符号表入口地址填入LT所指符号表哪一项的CMP中（拉链），同时也填入LT本身。

例如：COMMON X,Y

COMMON /B1/A,B,C//D,E,F(100)

符号表

	Name	…	DA	CMP
1	X			2
2	Y			6
3	A		B1	4
4	B		B1	5
5	C		B1	0
6	D			7
7	E			8
8	F			0

COMLIST

NAME	LENGTH	FT	LT
	…	1	8
B1	…	3	5

第一节　静态存储管理

四、等价语句(EQUIVALENCE)处理

1、等价语句(EQUIVALENCE)的作用

- 使同一程序单位中的两个或更多的变量共用同一存储单元。

2、 EQUIVALENCE语句的格式

  - EQUIVALENCE （变量表1），（变量表2），……
  - 例如：EQUIVALENCE (W,ST)
  - 注：括号内的变量表中至少包含两个变量名；可以用此语句定义共享的有简单变量名，数组名，数组元素等。

第一节　静态存储管理

四、等价语句(EQUIVALENCE)处理

3、等价语句内存分配的特点：

- 1）等价语句定义了若干个等价片，每个等价片是由括号括起来的若干个变量名组成的。这些变量名称为等价元。
  - 注：等价元可以是简单变量或下标为常数的下标变量。
- 2）等价片中等价元共享存储单元。
- 3）不同等价片中若有相同变量名，则为等价相关，应将其合并，让更多等价元共享存储单元。

第一节　静态存储管理

四、等价语句(EQUIVALENCE)处理

4、编译EQUIVALENCE的主要工作

- 找出那些变量存在等价关系，将它们构成等价环；同时指出各等价元的首地址关系，以便说明语句处理完后进行内存分配。

5、实现方式

在符号表中再增设两栏：

- EQ：等价环形链，若为null，表示该变量不属于等价元；否则指向下一个等价元的入口，构成环形链。
  - 注：若只有一个等价元，则指向自身。
- OFFSET：相对位移量，用来指出各等价元存储首地址间的地址相对关系。

例如：说明语句如下：

- INTEGER I,J,K,X,A(10,10)
- EQUIVALENCE(X,A(2,3)),(I,J,A(1,2),K)
- 经编译程序对说明语句处理后获得如下符号表，建立等价环并填写相对位移量。

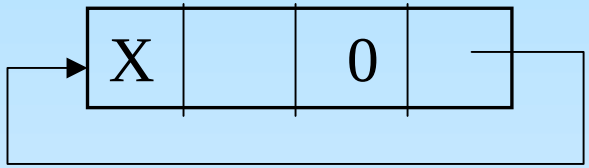
	NAME	…	OFFSET	EQ
1	I		-11	3
2	J		-11	4
3	K		-11	2
4	X		0	5
5	A		-21	1

6、等价环、相对位移量建立的过程：

- EQUIVALENCE(X,A(2,3)),(I,J,A(1,2),K)

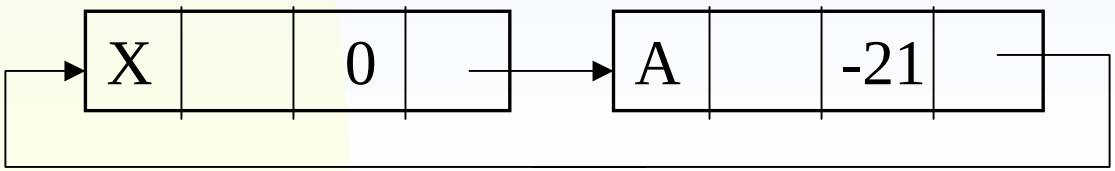
1)扫描X，构造

- X.OFFSET=0

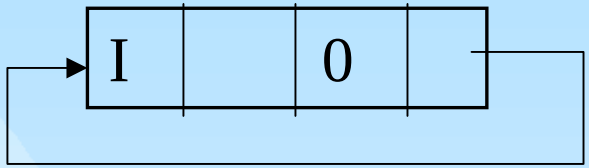


2)扫描A(2,3)，由于它和X是同一片内的，所以变量X和数组A可以构成等价环，因为

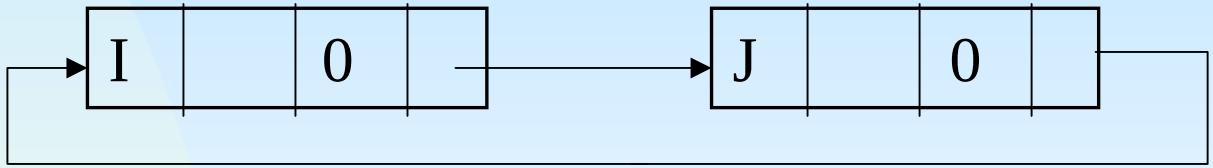
- A(2,3) .OFFSET=X.OFFSET=0;又因为
- A(2,3)＝A(1,1)+[(2-1)+(3-1)*10];
- 故：A的首地址为－21。



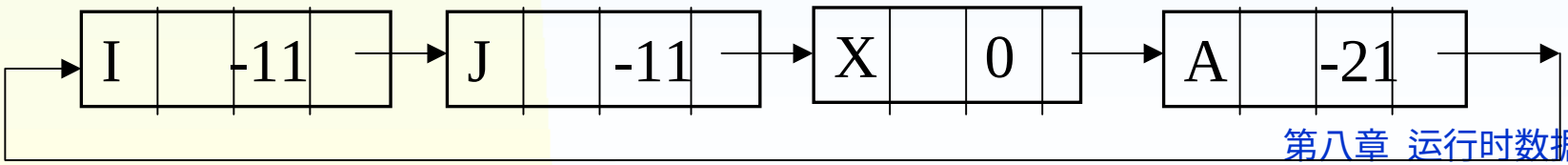
- 接着向后扫描，扫到I得：



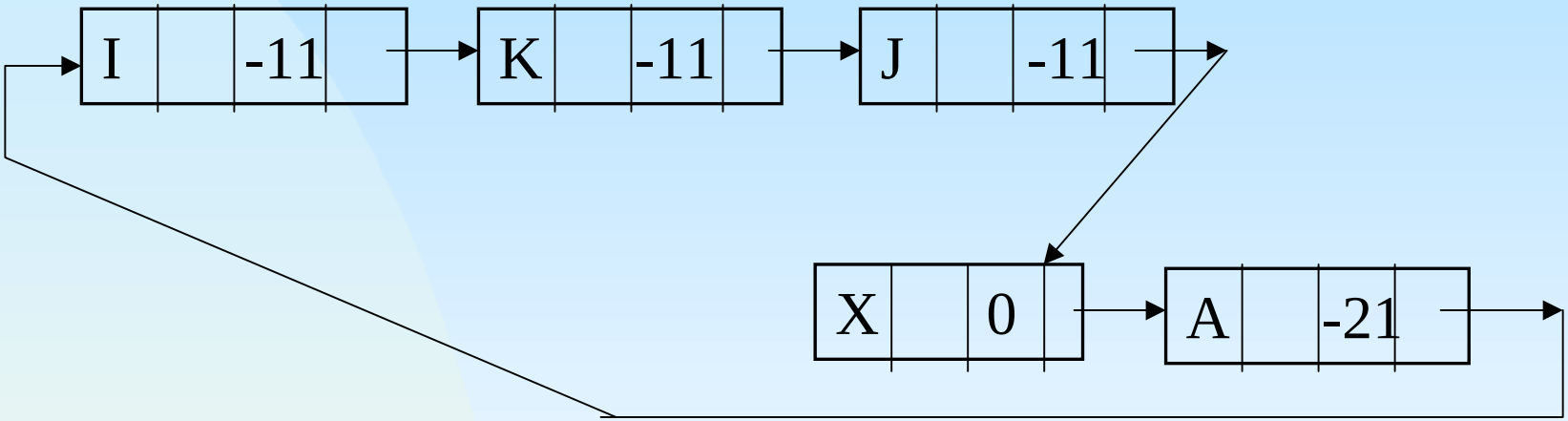
•扫描到J，构成等价环：



•扫描到A(1,2)，发现A.EQ<>null，即它在某等价环内，且该等价环和当前等价环相关，需要合并等价环，且以前一个等价环的位移量为准;由于A(1,2),I,J同享一个单元，所以其OFFSET相同。



- 扫描到K，构成等价环：



注：在上述过程中还应该检查是否有等价冲突等问题。

第一节　静态存储管理

五、地址分配

1、公用区地址分配

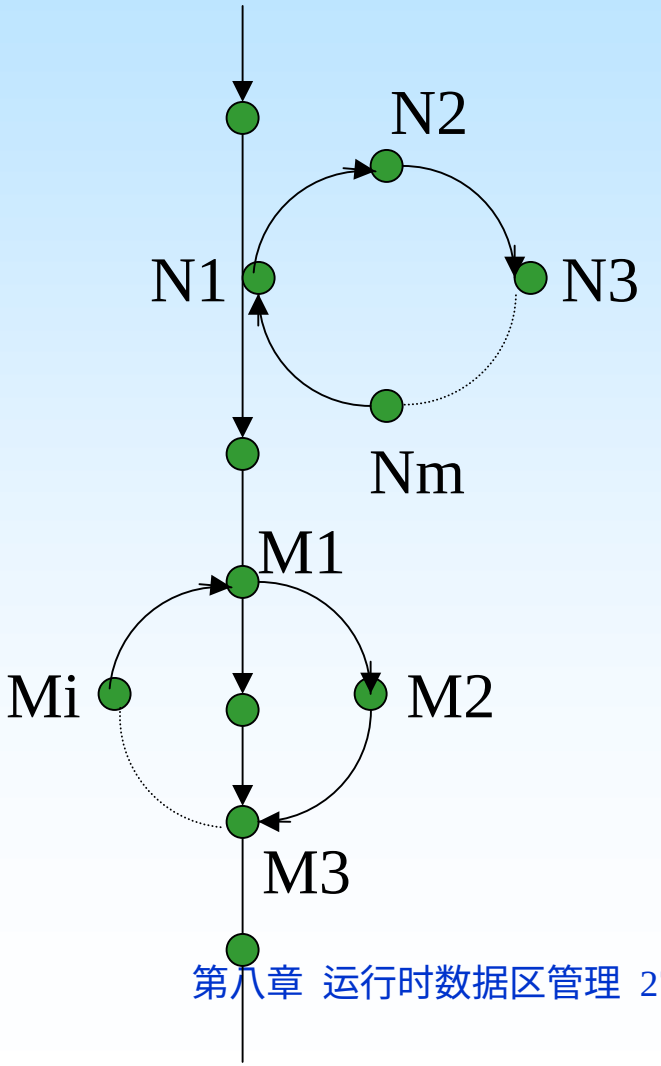
- 过程：
  - 从公用区的链首变量开始，沿着公用链逐个为公用元分配内存地址。每次分配地址时查看该公用元的EQ栏是否为空。
    - 若不空，则按EQ指出的等价环，为环中所有等价元分配存储地址，直到等价环处理完，返回公用链继续向下分配。
    - 若为空，继续分配后继公用元。
  - 此过程制导公用链处理完为止。

第一节　静态存储管理

五、地址分配

1、公用区地址分配

- 具体操作：
  - 1）在分配到变量N1时发现N1.EQ<>null,从EQ中找出等价环N1,N2,…,Nm,其相对位移量分别为f1,f2,f…,fm，为它们分配地址为ADDR(Ni)=a+(fi－f1),i=2,3,…,m。
  - 其中a为N1分到的地址。



- 具体操作：
- 2）若a+fi-f1<0,则公用区越界，为避免这种等价结果，等价环最好不要处于公用链链首位置。
- 3）若某变量在等价环中已分配了地址，回到公用链又为其分配地址，这表示分配出现冲突。
- 4）公用区长度计算：每处理完一个等价环，公用区长度len=MAX(len,a+MAX(fi-f1+SIZE(Ni))),
  - 其中i=1..m；
  - 等式右边的len表示分配到这个等价环之前公用区的长度。
  - A作为分配公用元地址计数器，初值假定为0；
  - SIZE (Ni)表示第Ni个等价元占有存储单元数。

注：1）一般假定从127区开始进行存储分配；

2）等到该公用区处理完，len值就表示该程序段中使用该数据区的最大长度。即：公用区可以用等价语句扩大。

第一节　静态存储管理

五、地址分配

2、局部区变量地址的分配

- 1)一般变量的分配
- 2)局部区出现等价环，处理策略：

(1)选择环中相对位移量最小的等价元开始分配地址，以解决冒头问题；

(2)为了使等价环中变量与非该等价环中的变量不占有相同存储单元，对地址分配计数器做如下处理：

- a:＝a+$MAX_{i=1}^{m}$ (fi-f+SIZE(Ni))
- 其中： Ni是等价环中m个元素N1,N2,…,Nm的任意一个，fi是它们相应的相对位移量，并设其中最小的相对位移量为f.

第一节　静态存储管理

五、地址分配

2、局部区变量地址的分配

- 2)局部区出现等价环，处理策略：

注：除了等价环元素之外，局部区变量的内存分配严格按照变量出现的顺序进行分配。

- 3)局部区等价环分配算法：

- BEGIN
- N_1:=N;　　f:=OFFSET[N];
- WHILE EQ[N_1]<>N DO
- {N1:=EQ[N1];　f:=MIN(f,OFFSET[N_1])};
- len:=负无穷
- N_1:=N;
- REPEAT
- DA[N_1]:=现行程序段序号；
- ADDR[N_1]:=a+(OFFSET[N_1]-f);
- len:=MAX(len,(OFFSET[N_1]-f)+SIZE(N_1));
- N_1:=EQ[N_1];
- UNTIL N_1=N;
- A:=a+len
- END;

第一节　静态存储管理

- 例如：根据下面的FORTRAN程序，为变量分配内存地址。
  - 假定整型量占1字，实型占2字编址。
- INTEGER A,B　　/*MAIN*/
- DIMENGSION A(2,3),C(5),D(8)
- COMMON I,J,X,A /R/E,F
- EQUIVALENCE (A(2,2),C(1)),(D(4),B)
- ……
- V=B*5
- U=1.5*V
- ……
- END

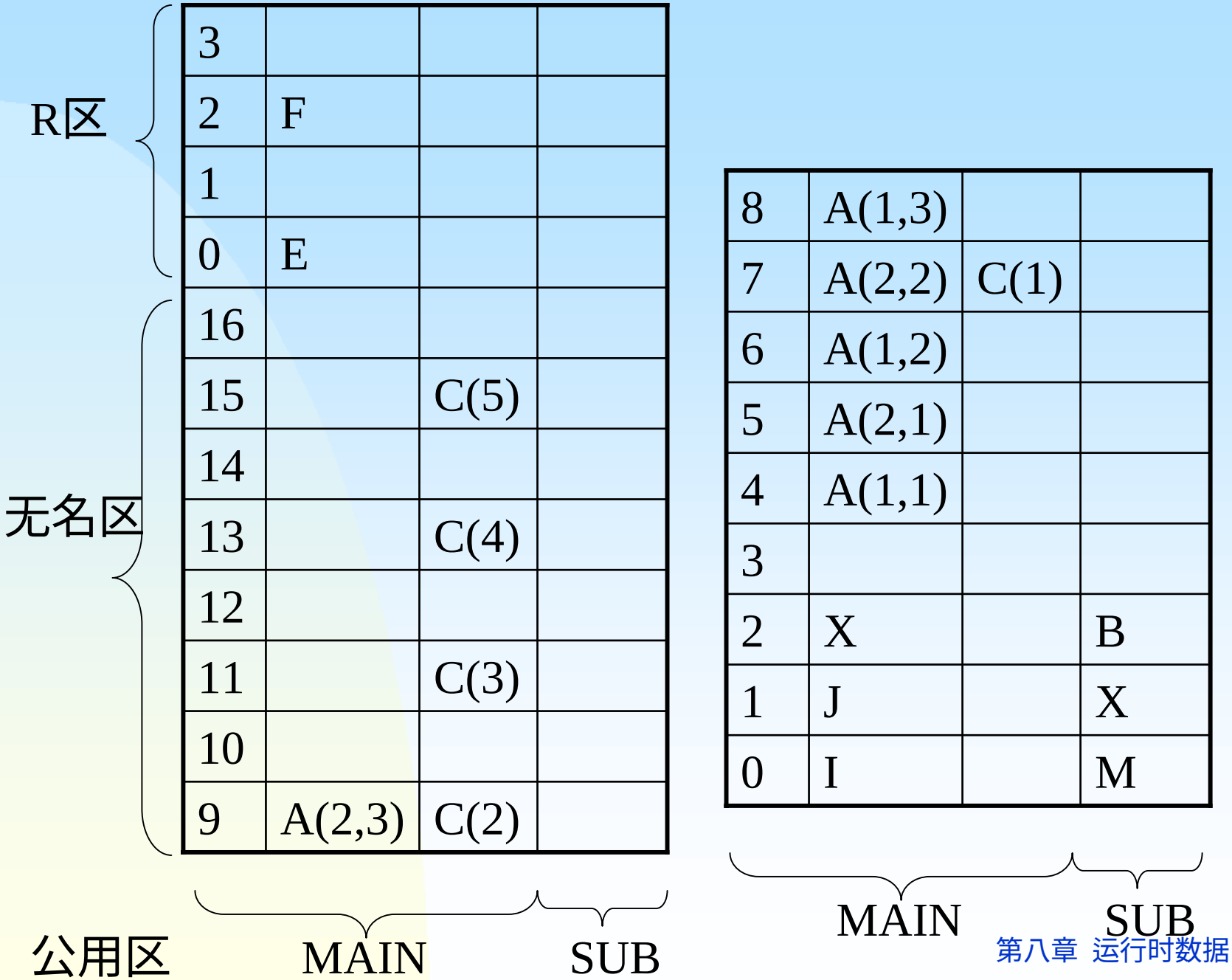
- SUBROUTINE SR1　　/*子程序*/
- 　　COMMON M,X,B
- 　……
- END
- FUNCTION FN1　　/*子函数*/
- 　REAL W,Y
- ……
- END

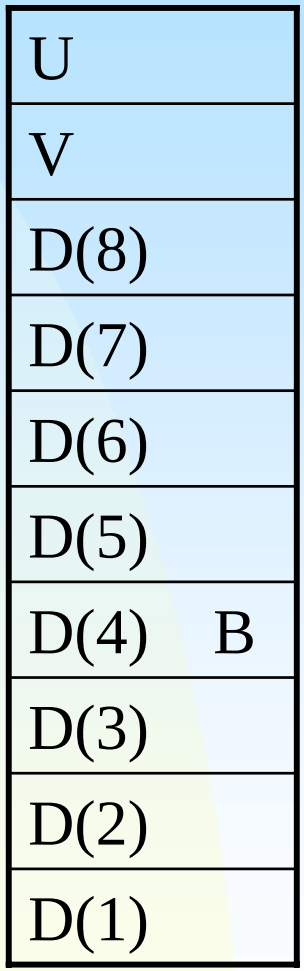
COMLIST表

区名	length	FT	LT
	17	K+4	K+0
R	4	K+7	K+8

•假定公用区区号从127区开始进行存储分配，则其符号表为：

	name	type	CAT	DIM	offset	EQ	CMP	DA	ADDR
K+0	A	Int	数组	2	0	K+2	0	127	4
K+1	B	Int	简变		6	K+3		1	a+6
K+2	C	Real	数组	1	3	K+0		127	7
K+3	D	Real	数组	1	0	K+1		1	a+0
K+4	I	Int	简变				K+5	127	0
K+5	J	Int	简变				K+6	127	1
K+6	X	Real	简变				K+0	127	2
K+7	E	Real	简变				K+8	128	0
K+8	F	Real	简变				0	128	2
K+9	V	Real	简变					1	a+16
K+10	U	Real	简变					1	a+18


R区
无名区
公用区
3
2 F
1
0 E
16
15 C(5)
14
13 C(4)
12
11 C(3)
10
9 A(2,3) C(2)
MAIN
SUB
8 A(1,3)
7 A(2,2) C(1)
6 A(1,2)
5 A(2,1)
4 A(1,1)
3
2 X B
1 J X
0 I M
MAIN
SUB



MAIN局部区

第一节　静态存储管理

六、临时变量地址分配

1、临时变量的类型

- 由于它暂存某些运算的中间结果，所以只可能是整型、实型、布尔型、双精度型等。

2、临时变量的作用域

- 一个临时变量从它被定值（赋值）开始到最后一次被引用的地方为止，其间程序可达到的四元式全体称为这个临时变量的作用域。
- 注：两个临时变量的作用域若不相交，则它们可分配在同一单元中。

第一节　静态存储管理

六、临时变量地址分配

3、临时变量存储分配算法：

- 令临时变量名均分配在局部数据区中，若某一单元已分配给某个临时变量名，则把该名的作用域作为此单元的分配信息记录下来。
- 每当要对一个新临时变量名进行分配时，首先求出此名的作用域，然后按序检查每个已分配单元，一旦发现新求出的作用域与某单元所记录的作用域不相交时就把这个单元分配给这个新名。同时把它的作用域信息也添加到该单元的分配信息中。
- 若新临时变量名的作用域和所有已分配单元的作用域有冲突，就分配给它一个新单元，同时把新名的作用域作为此单元的分配信息保存起来。

- 例如：赋值语句：
  - X:=A+B-((C+D)+(E+F)*(G+H))翻译成四元式为：
  - 四元式　　临时变量名　　地址
  - 1)(+,A,B,T_1)　　T_1　　a
  - 2)(+,C,D,T_2)　　T_2　　a+1
  - 3)(+,E,F,T_3)　　T_3　　a+2
  - 4)(+,G,H,T_4)　　T_4　　a+3
  - 5)(*, T_3, T_4 ,T_5)　　T_5　　a+2
  - 6)(+, T_2, T_5,T_6)　　T_6　　a+1
  - 7)(-, T_1, T_6 ,T_7)　　T_7　　a
  - 8)(:=, T_7, _,X)

第一节　静态存储管理

六、临时变量地址分配

3、临时变量存储分配算法：

注：1）在简单表达式中使用的临时变量的特点是单赋值单引用。这些临时变量的作用域是嵌套的或不相交的。

- 2）可以采用一个栈来存放这类临时变量。需要的临时变量的单元数等于最大的嵌套层数。
- 3）对于具有多寄存器的机器来说，临时变量不宜使用存储器，用寄存器来存放可以节省访问内存的时间。

第二节 栈式存储管理

一、允许过程(函数)递归调用的数据存储管理

- 1、语言特点

  允许过程（函数）的递归调用，但不允许定义嵌套的过程（函数），也不许使用可变数组。如C语言。

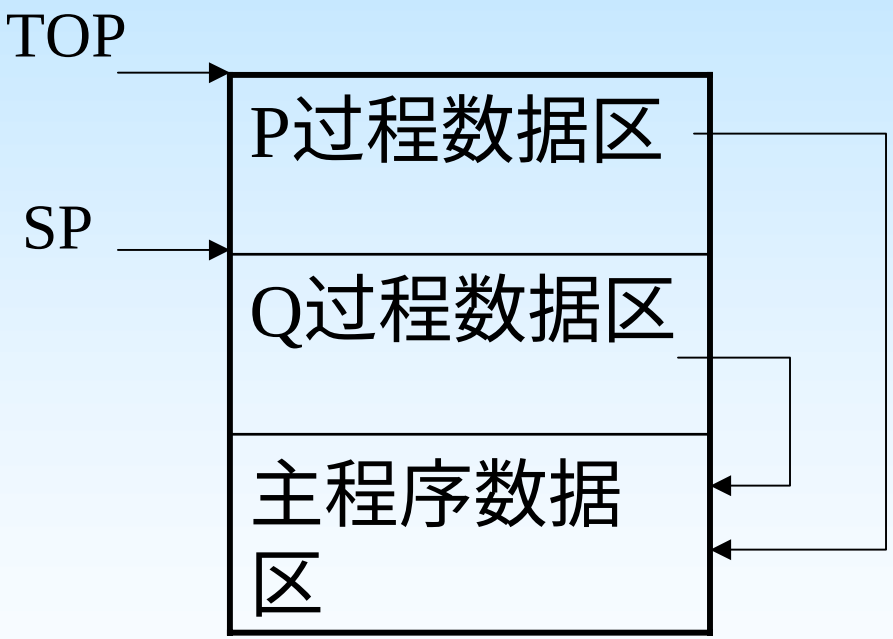
- 2、栈式存储分配：

- 每进入一个过程，就有相应的数据区建立在栈顶。当程序开始运行前，用于建造数据区的栈是空栈。当开始进入主程序执行语句前，便在栈中建立全局变量和主程序数据区。在主程序中若有调用过程的语句时，便在栈顶累筑该过程的活动记录。在进入过程后执行过程的可执行语句前再把局部数组累筑于栈顶，若还有调用过程的语句就重复上述过程。

第二节 栈式存储管理

- 例如有如下程序：
- MAIN ()
- {……
- Q( );
- }
- P( )
- {……}
- Q( )
- {……
- P( );
- }

- 当P过程进入运行后，栈顶数据区有两个指针：
  - SP指向现行过程数据区起点；
  - TOP指向顶点
- 注：从数据区中引出指向主程序数据区的箭头表示外部变量引用关系。



C数据区结构

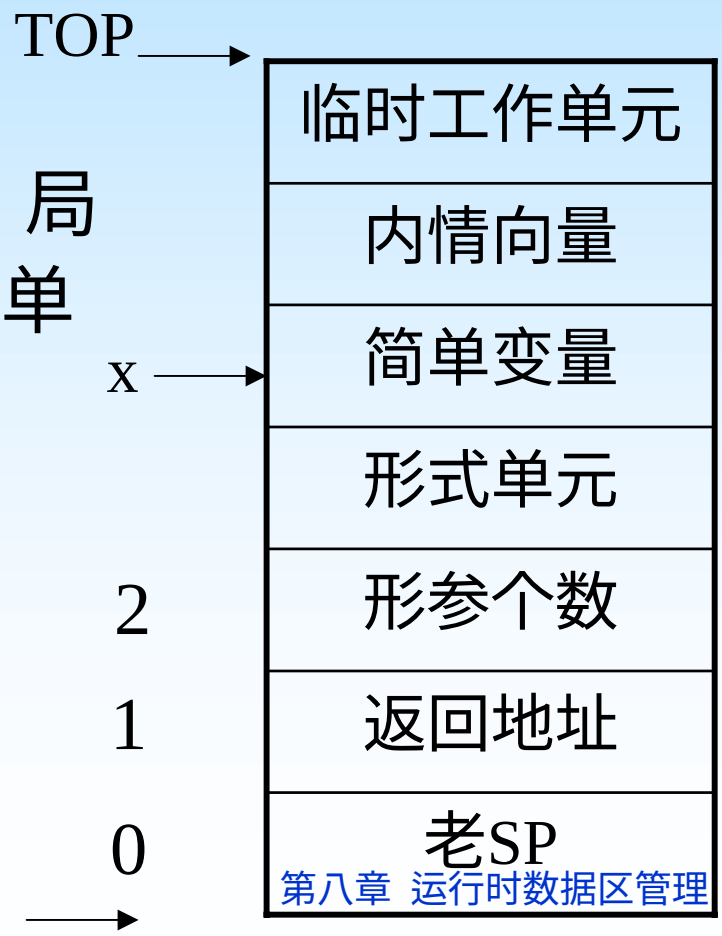
第二节 栈式存储管理

一、允许过程(函数)递归调用的数据存储管理

- 3、活动记录：

包含连接数据、形式单元、局部变量、内情向量和临时工作单元等。

C语言的活动记录



第二节 栈式存储管理

注：1)活动记录的建立是按照调用次序而累筑，而非排列次序；

2)栈顶活动记录数据区有两个指针SP和TOP，SP指向现行数据区起点，TOP指向顶点；

3)从数据区中引出指向主程序数据区的箭头表示外部变量引用关系；

4)C语言的活动记录所含区段是：连接数据(包含老SP值(即前一活动记录的首地址；或称施调过程的数据区首地址)和返回地址(即调用语句的下一条指令的地址))、参数(形参)个数、形式单元(存放实参值或地址)、过程的局部变量(简单变量)、数组内情变量和临时工作单元。

第二节 栈式存储管理

- 注：5)过程中的局部变量(简单变量)在内存地址可表示为变址形式x[SP]，其中SP为当前数据区首地址，用作变址值，x称为相对位移量。连接数据：
  - 老SP值：前一活动记录的首地址（施调过程的数据区首址）；
  - 返回地址；
- 简单变量X在数据区内相对地址设为x，则，X的内存地址表示为变址形式x[SP],SP为当前数据区首址，x为相对位移量。

第二节 栈式存储管理

一、允许过程(函数)递归调用的数据存储管理

4、C语言的数据区建立与撤销

- 1)过程调用段：
  - Call语句翻译成中间代码后所作操作有两个：将参数传递到过程或函数的数据区的形参单元中；跳转到过程或函数。
  - 例如：call Q((T1,T2,…,Tn)语句的中间代码为：
    - Par T1
    - …
    - Par Tn
    - Jsr n,Q

传值解释为：(i+3)[TOP]:=Ti
传址解释为：(i+3)[TOP]:=addr(Ti)

解释为：1[TOP]:=SP;
3[TOP]:=n;
jsr Q;

注：C语言的数据区是由过程调用引起。

一、允许过程(函数)递归调用的数据存储管理

4、C语言的数据区建立与撤销

- 2)过程进入段
- 工作： (1)定义新活动记录的SP，保护返回地址和定义这个活动记录的TOP,即：
  - SP:=TOP+1
  - 1[SP]:=返回地址
  - TOP:=TOP+L　　/*L是过程Q的活动记录所需单元数*/
- (2)若过程中包含可变数组，且数组的空间分配在活动记录的 顶上，则接下来执行对数组进行存储分配指令，这段指令是编译数组说明时产生的运行子程序，具体工作是：
  - 计算各维上下界；
  - 调用数组空间分配程序，构造内情向量表，计算数组尺寸预留数组空间。

第二节 栈式存储管理

一、允许过程(函数)递归调用的数据存储管理

4、C语言的数据区建立与撤销

3)过程返回段

- (1) 假定此时E的值已经计算出来并已放在某临时单元T中，那么就将T值传送到某个特定的寄存器中，以备调用段获取(调用段将从此寄存器获得被调用过程的结果值)；
- (2)恢复SP和TOP，以便回到调用段的数据区。
  - TOP:=SP-1；SP:=0[SP] 或SP:=1[TOP]
- (3)按返回地址回到调用语句的下一语句去继续运行。
  - X:=2[TOP]；Jmp 0[X]
- 注：若是过程调用，则不必回送结果，其他相同。

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

1、语言特点

既允许过程嵌套，也允许过程递归调用。

2、存储管理方式

1)根据嵌套过程语言的规定，由变量的最小作用域原则，一个过程可以引用包围它的任意外层过程所定义的变量和数组。所以，运行时过程必须知道它所有直系外层过程的最新活动记录的地址。

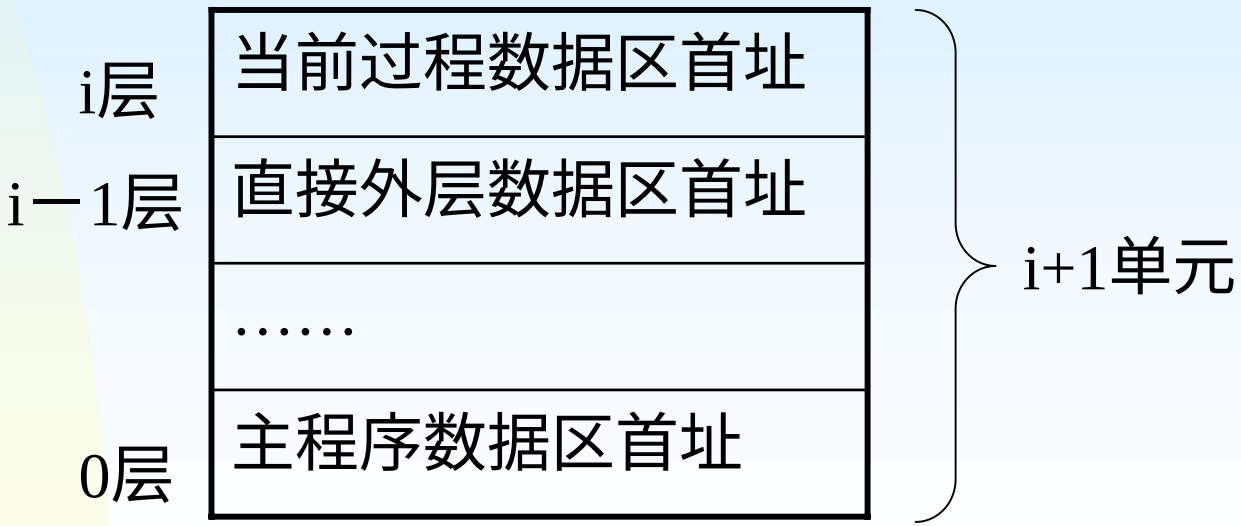
2)由于允许递归，过程活动记录的位置是动态变化的。因此，每个活动记录中必须设法记住直系外层的最新活动记录的位置，以处理递归。

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

3、解决方案 使用层次显示表display

- 1）内容：每进入一个过程，在建立其活动记录区的同时，为它建立一张层次显示表，以登记它所有直系外层最新活动记录的首址和本过程活动记录的首址。



第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

3、层次显示表display

1)内容

注：(1)直系外层是指从0层开始直到当前过程，即当前过程的所有直系祖先；

(2)可将本过程display表放在其活动记录中的形式单元之上

(3)也可将全局display存储在本过程的活动记录中(在返回地址之上)。

(4)因为当前过程的层数是确定的，所以它的display表也是确定的。

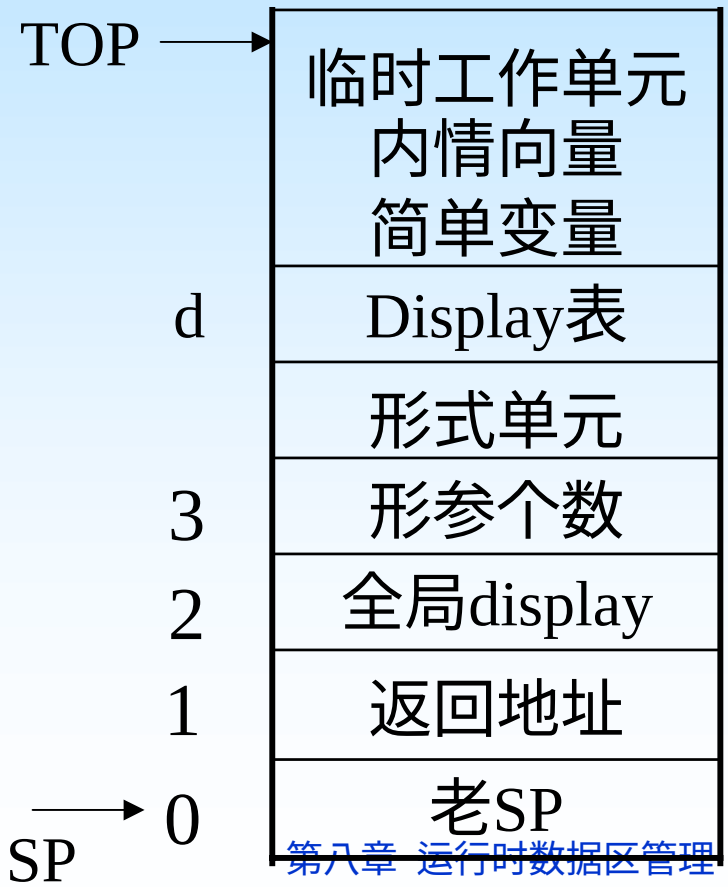
(5)由于每个过程的形参单元的数目在编译时是完全确定的,所以display表的相对地址d在编译时也是完全确定的。

第二节 栈式存储管理

3、层次显示表display

2)Pascal语言的活动记录

PASCAL的活动记录



第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

4、直系外层中变量的引用

- 假定现行过程中引用了某一直系外层(k)的变量X，则可以用下面两条变址指令获得X的值，送到R2寄存器。
  - LD R_1,(d+k)[SP];
  - LD R_2,x[R_1];
- 注： 第一句作用是从display表中取出k层过程的最新活动记录首址送给R_1 ；
- x是变量X在第k层活动记录的相对地址。

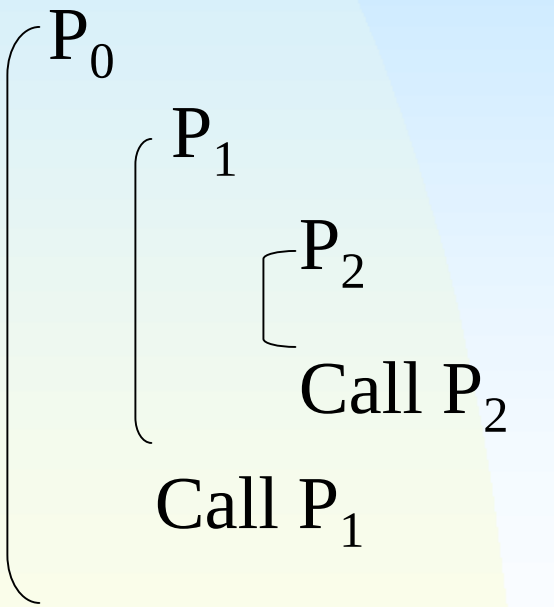
第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

5、层次显示表display的建立

1)若被调用过程是真实过程

- 例如：



若当前处于P_1过程，它的display表包含：

P_0活动记录首地址和P_1过程最新活动记录首址。

当执行Call P_2后，应进入过程 P_2，在建立P_2活动记录的同时应建立P_2的display表。此表的内容为：

P_0活动记录首地址和P_1过程最新活动记录首址，及P_2活动记录首址。

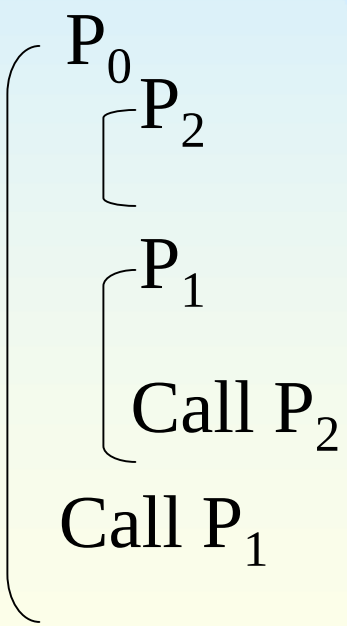
第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

5、层次显示表display的建立

1)被调用过程是真实过程

例如：同层调用



若当前处于P_1过程，当执行Call P_2后，应进入过程 P_2， P_2的display表包含两项：

P_0活动记录首地址和P_2的SP。

P_0活动记录首地址在P_2的display表中有过，可从其中抄得。

第二节 栈式存储管理

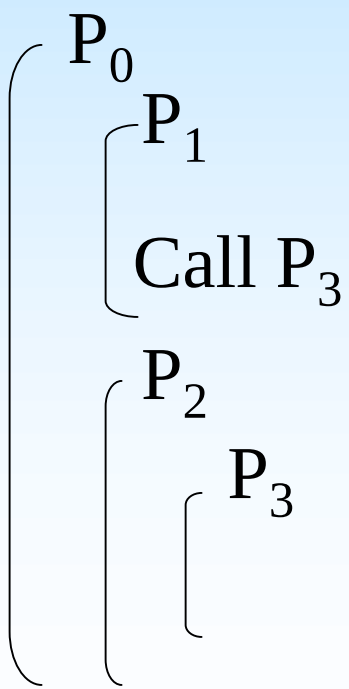
二、嵌套过程语言的栈式存储管理

5、层次显示表display的建立

1)被调用过程是真实过程

故：当进入第i层过程时，其display 表可从施调过程得display表中抄录i项，i代表当前数据区的静态层次，然后加上 自身活动记录首址SP组成。

- 注：要完成上面的过程，将施调过程的display表首址作为连接数据送到被调用过程的全局display单元。
- 不允许隔层调用，是因为造不出被调过程的display表。

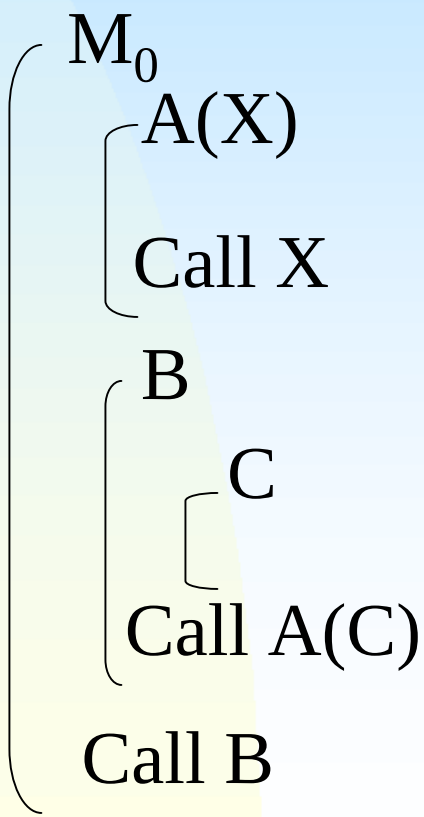


第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

5、层次显示表display的建立

2)被调用过程是形式过程



注：在过程B中调用A时，把过程名C作为实参传递给A，而A中又有CALL X语句，X是形参，，这就是调用形式过程。

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

5、层次显示表display的建立

2)被调用过程是形式过程

注：(1)若被调用过程是形式过程，根据形实替换原则，应调作为实参的那个过程C，建立该过程的活动记录和display表，显然，此时C.display表不能从施调过程A中抄得，而应从把该过程名当作实参传递的那个过程中(即过程B)抄得，即应把B.display表首地址传递给C的全局display表。

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

5、层次显示表display的建立

2)被调用过程是形式过程

注：2)总之，要构造某被调过程的display表，它或者从施调过程的display表中抄若干项，或者从把实在过程名当作实参传递的那个过程的display表中抄若干项，然后再加上本身的SP组成。

3)此时，连接数据要变为三项：老SP、返回地址和全局display地址。

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

- 1）过程调用段
- （1）传递简单变量（包括临时变量和常量）
  - Par Ti解释为: (i+4)[TOP]:=Ti /*传值*/
  - 或： (i+4)[TOP]:=addr(Ti) /*传地址*/
- （2）传递数组：一般传递内情向量表首址。
  - (i+4)[TOP]:= Ti的内情向量表首地址

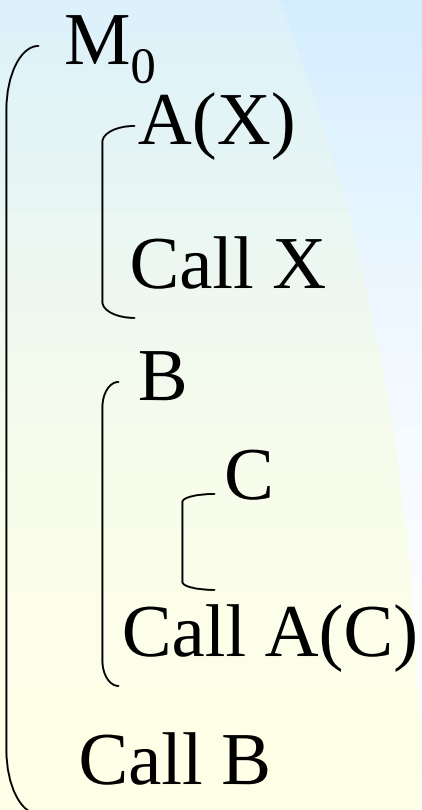
第三节 栈式存储管理

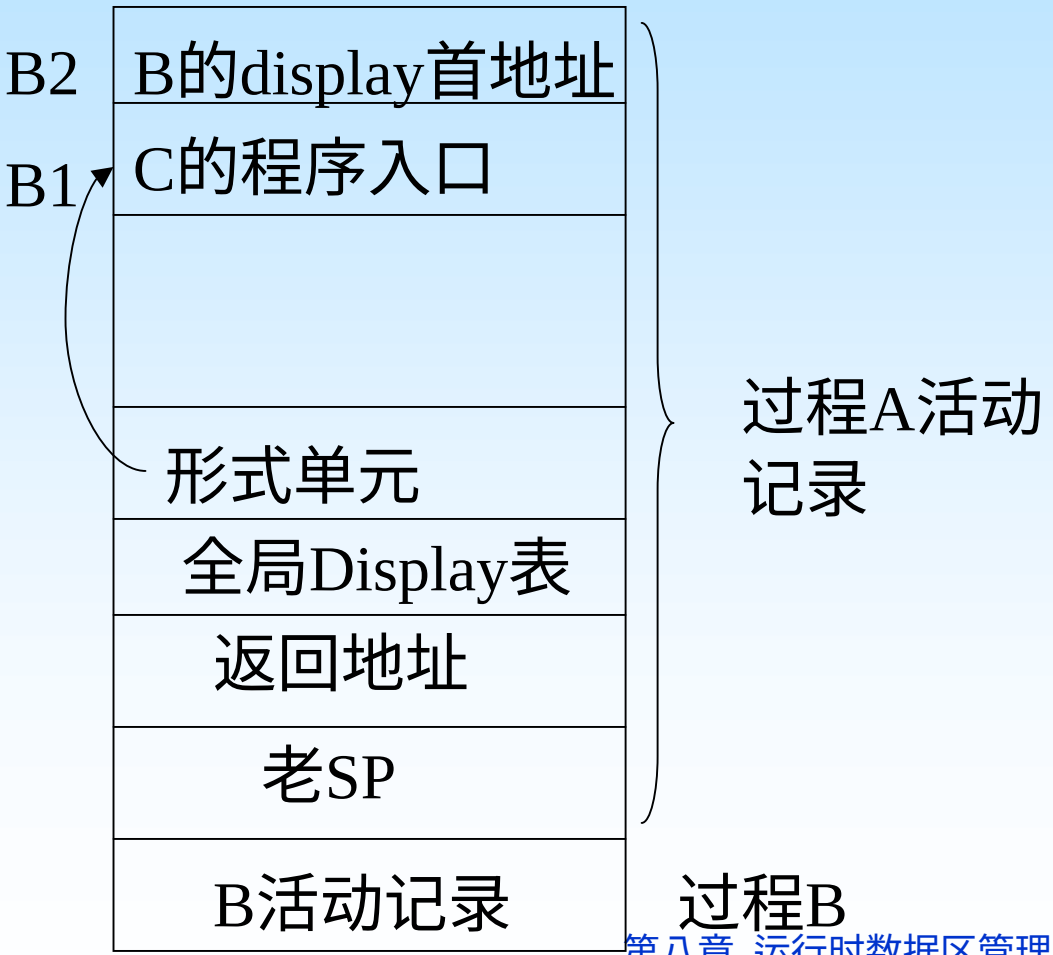
二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

1)过程调用段

(3)传递过程名





第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

1)过程调用段

（3）传递过程名：

- 传递两个参数：实在过程C的程序入口地址；过程B的display表首址
- 解决方法：
  - 在过程A的活动记录的临时变量区中建立两个相继单元B1,B2。B1保留实在过程C的入口地址；B2保留把过程名当作实参传递的那个过程的display表首址。
  - 然后： (i+4)[TOP] :=addr(B1);
  - 这样，就可以间接引用(i+4)[TOP]形参单元得到C的程序入口地址，由B2单元得到C的全局display.

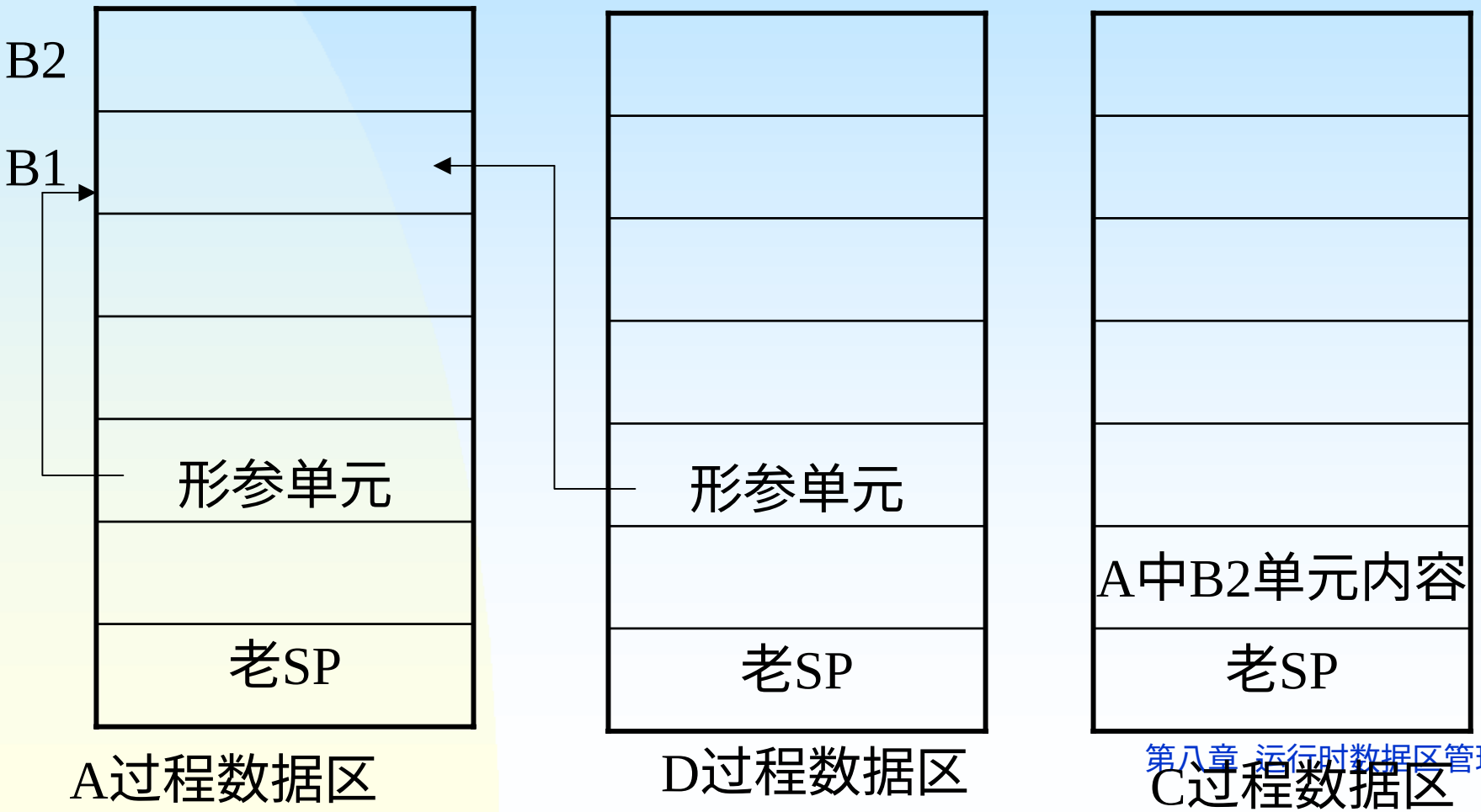
第二节 栈式存储管理

(4)传递形式参数（包括形式过程名）

- 例如：下面的程序段：

PROC B;	PROC A(X);	PROC D(P)
proc C;		
……	……	……
end;	call D(X)	call P
……	……	……
call (C);	end	end
end		

- 这里par Ti 可以理解为传递形参内容，即(i+4)[TOP]＝(i+3)[SP].
- 每次调用后数据区的建立情况如下：



第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

1)过程调用段

(4)传递形式参数

- 注：这时C过程数据区中全局display是存B2的内容，即B的display的首址，所以数据区的活动记录中应设置全局display。
- 若高级语言不允许把过程名作为实参传递，就没有必要保留全局display单元，甚至无需建立display表，而改用静态链指向它直系外层的最新活动记录首址即可。

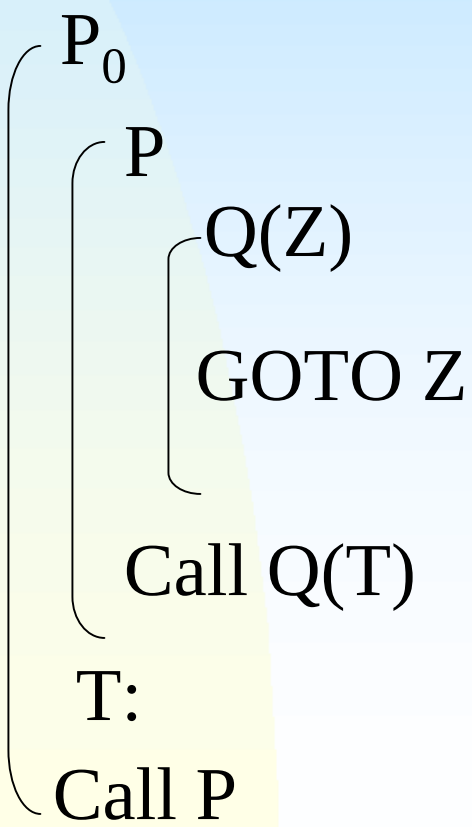
第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

1)过程调用段

- （5）传递标号



Par T可以理解为在Q活动记录的临时变量区建立两个相继单元B1,B2;B1用于存放标号T的对应地址，B2用于存放P_0活动记录首址。然后把B1地址传给Q过程的相应形参单元。

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

1)过程调用段

（5）传递标号

- JSR n,Q指令可解释为：
- 1[TOP]=SP;
- 3[TOP]=SP+d　　或　3[TOP]=B2
- 4[TOP]=n
- JSR Q

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

2)过程进入段

- SP=TOP+1;
- 1[SP]=返回地址
- TOP=TOP+L

- 构造display表，,即按全局display指出位置抄I项并加上本身的SP,其中i为该过程的静态层次号LN
- 若有数组说明，根据内情向量表，建立数组存储区，并修改栈顶指针TOP

第二节 栈式存储管理

二、嵌套过程语言的栈式存储管理

6、Pascal语言的数据区建立和撤销

- 3)过程返回段
  - TOP=SP-1;
  - SP=0[SP];
  - X=2[TOP];
  - JMP 0[X];
- 若 return语句含有返回值或Q是函数，则需先把返回值送到某特定寄存器。

- 例如：对下面求阶乘的程序，第二次调用F函数后，给出栈式数据区的内容。假定每个过程有存储函数F的结果单元。

```
Program M;
  var A,B:integer
  function F(N:integer):integer;
        if N<=2 then teturn(N)
          else return (N*F(N-1));
 begin
   read(A);          /*设A=5*/
   B:=F(A);
  write (B);
 end.
```

+21	从F返回结果	
+20	D表	K+14
+19		K
+18	形参N	4
+17	参数个数	1
+16	全D地址	K+11
+15	返回地址	
+14	K+6(SP_1)	
+13	从F返回结果	
+12	D表	K+6
+11		K

+10	形参N	4
+9	参数个数	1
+8	全D地址	K+2
+7	返回地址	
+6	K(SP_0)	
+5	从F返回结果	
+4	B	
+3	A	5
+2	D表	K
+1		
K		

第三节 堆式存储管理

- 对象：
  - 在高级语言中有些数据存储空间的请求与释放不再遵循后进先出的原则，而且是全局性的。
- 对策：
  - 让运行程序持有一块专用的全局存储空间来满足这些数据的存储要求。这种存储空间就叫“堆”(heap)。
  - 堆通常是一片连续的足够大的存储区，当需要时，就从堆中分配一小块存储区；用完就及时退还给堆。

第三节 堆式存储管理

一、堆式存储管理技术

- 1、固定长块管理
- 方法：
  - 把堆空间分成许多固定长的块，每块的第一个字用作指示器，将所有未用块链接起来，形成一张可利用表。
  - 当堆管理程序收到请求分配空间时，就查询该表，并将一块空白块的首指示器送给申请者，然后修改此表。

第三节 堆式存储管理

一、堆式存储管理技术

2、可变长块管理

- 申请分配空间时，堆管理程序从可利用表中查到一个未用块，若该块大于所需空间，就将它分成两部分，等于申请空间的一部分和剩余部分；这样下去留在可利用表中的块越来越小，最后形成无法用的碎片。
- 若查到的块只比申请空间大一点，就不再分开，整个给申请者使用，此块中用不到的那部分也无法给别人用，就是内部碎片。
- 注：这两种碎片很多时，碎片的总和大于申请空间，但无法分配给申请者。故可变长块管理主要考虑如何减少碎片的影响。

第三节 堆式存储管理

一、堆式存储管理技术

2、可变长块管理分配策略

- (1)首次匹配式堆管理策略
  - 给表中每块首字中加入块大小的信息，分配时沿可利用表顺序查找，选择第一个大于所需空间的未用块，此块划出所需空间后剩余的部分必须大于某规定长度。
  - 注：改进算法，可采用循环链表，释放存储区时合并相邻未用块。
- (2)最优匹配式堆管理策略
  - 分配时对可利用表从头查到尾，找到一个容量正好等于或稍大于申请空间的未用块。
  - 注：此方法看似合理，但太花时间。

第三节 堆式存储管理

一、堆式存储管理技术

2、可变长块管理分配策略

- (3)最差匹配式堆管理策略
  - 分配时找到一个容量大于申请空间且是可利用表中最大的的未用块，把它分开划给申请者，剩余部分留在表中。
  - 注：为了节省查找最大未用块的时间，可以将可利用表的未用块从大到小排好。这样分配的剩余部分要插到表的适当位置上。

第三节 堆式存储管理

一、堆式存储管理技术

3、按块长不同分为若干集合

- 方法：把整个堆空间分为若干集合，每个集合中块长是相等的，并把它们链接在一起。当申请m个长度为n的块时，就到块长为n或稍大于n的集合中去查找可利用块。若该集合中由m个这样的块，则满足申请；若没有m个，则从块长较大的集合中取一块或多块，把它们分成相等的两块然后再分配。
- 注：优点：减少搜索时间，容易实现块合并；
- 缺点：会使内部碎片增加。

第三节 堆式存储管理

二、堆空间的释放与无用单元收集

1、堆空间的释放

- 对应语句:
  - Dispose(P);　free(p);
- 处理方法：
  - 将释放块作为新块插入到可利用表的链首位置。
  - 由于这样做一段时间后，可利用表中将有大量小块，影响分配程序的操作；此时可改进一下:
  - 将地址连续的小块合并成一大块。
- 注：这样做可利用表必须按块的地址顺序组织。

二、堆空间的释放与无用单元收集

2、无用单元的收集

- 1）执行时机：
  - 在堆的可利用空间几乎耗尽，以至于不能满足用户申请要求，或发现可利用空间已降至某个危险点时执行。
- 2）收集过程：
  - 标记阶段：查看已分配的块近期有无访问，访问过就做一标记；
  - 收集阶段：把所有没有加标记的存储块加入可利用表中，然后消除标记。
- 注：最好在可利用空间降到某个数值时就调用收集程序，以避免收集时间过长。
- 执行收集程序必须中断用户程序的执行，所以选择执行收集程序的时刻必须合理。

第三节 堆式存储管理

二、堆空间的释放与无用单元收集

2、无用单元的收集

- 注：PASCAL语言运行时的数据区存储管理将栈和堆安排在存区两端，各自无关的向中间靠拢。当碰到一起时存储空间用完。此时调用无用单元收集程序。
- 数据区是否够用时无法估计的，堆和栈碰头也是难免的，所以设置堆管理是很重要的。

小结

1)Fortran的公用语句和等价语句的处理

所有公用变量单独存放在一起(放在公用区)

等价语句形成等价链，注意存储空间分配的冒头问题

2)C语言程序的数据存储管理

允许递归调用

使用活动记录，调用过程中生成

3)PASCAL语言程序的数据存储管理

允许递归调用和过程嵌套定义

使用带有DISPLAY表的活动记录，全局DISPLAY表首地址是调用过程的DISPLAY表首地址