考虑一下如果一个程序有非常多的变量，但是其他部分都是正常数量时会发生什么事情。地址空间中分给符号表的块可能会被装满，但这时其他表中还有大量的空间。编译器当然可以简单地打印出一条信息说由于变量太多编译不能继续进行，但在其他表中还有空间时这样做似乎并不恰当。

另外一种可能的方法就是扮演侠盗罗宾汉，从拥有过量空间的表中拿出一些空间给拥有极少量空间的表。这种处理是可以做到的，但是它和管理自己的覆盖一样，在最好的情况下是一件令人讨厌的事，而最坏的情况则是一大堆单调且没有任何回报的工作。

我们真正需要的是一个能够把程序员从管理表的扩张和收缩的工作中解放出来的办法，就像虚拟内存使程序员不用再为怎样把程序划分成覆盖块担心一样。

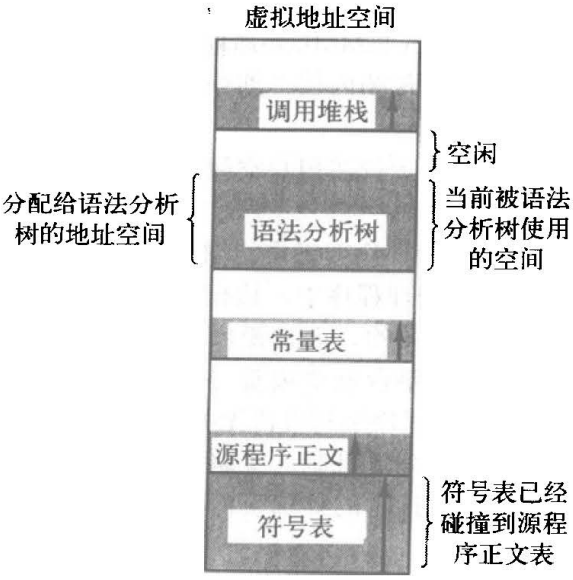


图3-31 在一维地址空间中，当有多个动态增加的表时，一个表可能会与另一个表发生碰撞

一个直观并且通用的方法是在机器上提供多个互相独立的称为段（segment）的地址空间。每个段由一个从0到最大的线性地址序列构成。各个段的长度可以是0到某个允许的最大值之间的任何一个值。不同的段的长度可以不同，并且通常情况下也都不相同。段的长度在运行期间可以动态改变，比如，堆栈段的长度在数据被压入时会增长，在数据被弹出时又会减小。

因为每个段都构成了一个独立的地址空间，所以它们可以独立地增长或减小而不会影响到其他的段。如果一个在某个段中的堆栈需要更多的空间，它就可以立刻得到所需要的空间，因为它的地址空间中没有任何其他东西阻挡它增长。段当然有可能会被装满，但通常情况下段都很大，因此这种情况发生的可能性很小。要在这种分段或二维的存储器中指示一个地址，程序必须提供两部分地址，一个段号和一个段内地址。图3-32给出了前面讨论过的编译表的分段内存，其中共有5个独立的段。

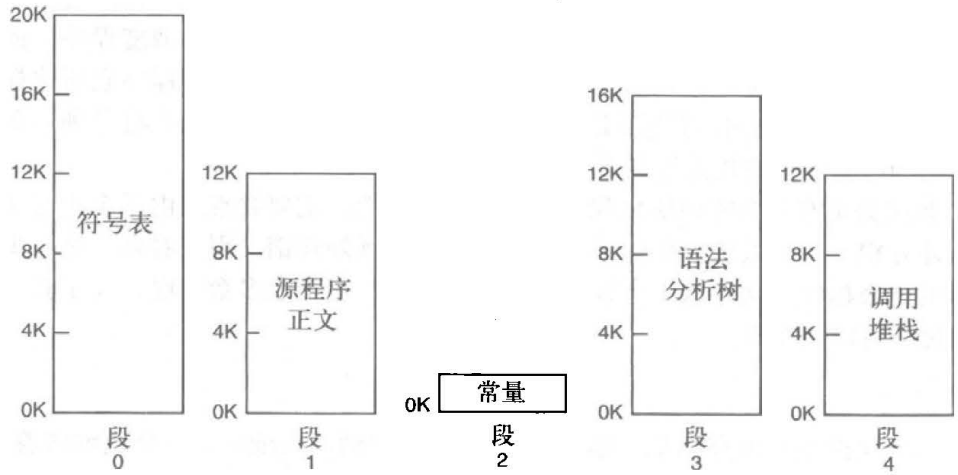


图3-32 分段存储管理，每一个段都可以独立地增大或减小而不会影响其他的段

需要强调的是，段是一个逻辑实体，程序员知道这一点并把它作为一个逻辑实体来使用。一个段可能包括一个过程、一个数组、一个堆栈、一组数值变量，但一般它不会同时包含多种不同类型的内容。

除了能简化对长度经常变动的数据结构的管理之外，分段存储管理还有其他一些优点。如果每个过程都位于一个独立的段中并且起始地址是0，那么把单独编译好的过程链接起来的操作就可以得到很大的简化。当组成一个程序的所有过程都被编译和链接好以后，一个对段$n$中过程的调用将使用由两个部分组成的地址（$n$，0）来寻址到字0（入口点）。

如果随后位于段$n$的过程被修改并被重新编译，即使新版本的程序比老的要大，也不需要对其他的过程进行修改（因为没有修改它们的起始地址）。在一维地址中，过程被一个挨一个紧紧地放在一起，