


代码区别于程序数据的一个重要属性是在运行时它是不能被修改的。当程序正在执行时，CPU 只从内存中读出它的指令。CPU 很少会重写或修改这些指令。

6.2.3 局部性小结

在这一节中，我们介绍了局部性的基本思想，还给出了量化评价程序中局部性的一些简单原则：


- 重复引用相同变量的程序有良好的时间局部性。
- 对于具有步长为 k 的引用模式的程序，步长越小，空间局部性越好。具有步长为 1 的引用模式的程序有很好的空间局部性。在内存中以大步长跳来跳去的程序空间局部性会很差。
- 对于取指令来说，循环有好的时间和空间局部性。循环体越小，循环迭代次数越多，局部性越好。

在本章后面，在我们学习了高速缓存存储器以及它们是如何工作的之后，我们会介绍如何用高速缓存命中率和不命中率来量化局部性的概念。你还会弄明白为什么有良好局部性的程序通常比局部性差的程序运行得更快。尽管如此，了解如何看一眼源代码就能获得对程序中局部性的高层次的认识，是程序员要掌握的一项有用而且重要的技能。

 **练习题 6.7** 改变下面函数中循环的顺序，使得它以步长为 1 的引用模式扫描三维数组 a：

```

1  int sumarray3d(int a[N][N][N])
2  {
3      int i, j, k, sum = 0;
4
5      for (i = 0; i < N; i++) {
6          for (j = 0; j < N; j++) {
7              for (k = 0; k < N; k++) {
8                  sum += a[k][i][j];
9              }
10         }
11     }
12     return sum;
13 }
```

 **练习题 6.8** 图 6-20 中的三个函数，以不同的空间局部性程度，执行相同的操作。请对这些函数就空间局部性进行排序。解释你是如何得到排序结果的。

```

1  #define N 1000
2
3  typedef struct {
4      int vel[3];
5      int acc[3];
6  } point;
7
8  point p[N];
```

a) structs 数组

```

1  void clear1(point *p, int n)
2  {
3      int i, j;
4
5      for (i = 0; i < n; i++) {
6          for (j = 0; j < 3; j++)
7              p[i].vel[j] = 0;
8          for (j = 0; j < 3; j++)
9              p[i].acc[j] = 0;
10     }
11 }
```

b) clear1 函数

图 6-20 练习题 6.8 的代码示例