ПРИМЕР ДОКАЗАТЕЛЬНОГО ПОСТРОЕНИЯ ПРОЦЕДУРЫ:
ВЫЧИСЛЕНИЕ НЕСМЕЩЁННОЙ (ИСПРАВЛЕННОЙ) ДИСПЕРСИИ

И.Е. Ермаков, ermakov@metasystems.ru
Проект OberonCore.ru

Пример разобран по просьбе участников форума DelphiKingdom.com.
Используется язык Компонентный Паскаль в реализации BlackBox Component Builder.
Текст набран в среде BlackBox.

Имеем задачу: написать процедуру, вычисляющую для входной последовательности исправленную выборочную дисперсию, за один проход.

Выборочная исправленная дисперсия расчитывается по формуле:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2$$

S2(n) = 1 / (n - 1) * SUM(i = 1..n; (Xi - AVG(i)) ^ 2),
где AVG(n) - выборочное среднее от n членов последовательности.

Исправленная дисперсия выражается через неисправленную следующим образом:

$$S^2 = \frac{n}{n-1}S_n^2,$$

$$S_n^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}X_i^2 - \left(\frac{1}{n}\sum_{i=1}^{n}X_i\right)^2.$$

Таким образом,

S2(n) = n / (n - 1) * (AVG(n, Xi^2) - AVG(n, Xi)^2),

где AVG расчитывается как среднее арифметическое.

Область определения:
D(S2): n > 1

Пусть входная последовательность имеет неизвестную длину, считывается из потока ввода.

Покажем ход мыслей при строгом решении задачи, с уровнем подробности, ориентированным, например, на объяснение студентам. Опытный программист будет менять этот уровень в зависимости от задачи, иногда большую часть прокручивая в уме (но обязательно прокручивая).

В преподавании же делать подобные построения просто необходимо, т.к. это самый прямой (если не единственный) способ добиться полного понимания смысла составляемых программ и организации процесса их составления (без использования пошагового отладчика и без воображения в уме того, как программа будет выполняться). Но заметим (по собственному опыту преподавания), что при обучении ярко выраженных "технарей" "чистый" подход может не работать; там, видимо, следует держаться поближе к тому, как реально программа исполняется (хотя метод пошагового уточнения и понятие инварианта являются обязательными).

При решении задачи используем метод пошаговой детализации с уточнением (см. Н. Вирт "Систематическое программирование" [1], В.Н. Касьянов, В.К. Сабельфельд "Сборник заданий по практикуму на ЭВМ" [4] и др.), базовую схему цикла "Полный проход" (см. [5], [6]) и инвариант цикла для рекуррентно вычисляемых переменных. См. также А.Г. Кушниренко, Г.В. Лебедев "Программирование для математиков" (курс Мехмата МГУ) [7].

Заметим, что на каждом шаге детализации имеем полностью компилирующуюся процедуру.

Комментарии выделены серым цветом и уменьшенным шрифтом, точки дальнейшей детализации выделены скобками (*< >*) и синим цветом. (Среда BlackBox/Component Pascal допускает произвольное шрифтовое оформление в исходном тексте).

ОПРЕДЕЛЕНИЕ ПРОЦЕДУРЫ

```
PROCEDURE CalcDisperse* (IN rd: TextMappers.Scanner; OUT d: REAL);
(* IN rd - курьер для входной последовательности; OUT d - несмещённая дисперсия для последовательности
   Предусловия:
      20    количество элементов последовательности > 1
*)
END CalcDisperse;
```

ДЕТАЛИЗАЦИЯ 1

```
PROCEDURE CalcDisperse* (IN rd: TextMappers.Scanner; OUT d: REAL);
(* IN rd - курьер для входной последовательности; OUT d - несмещённая дисперсия для последовательности
   Предусловия:
      20    количество элементов последовательности > 1
*)
   VAR n: INTEGER;    (* Количество элементов в последовательности *)
       avg_X2, avg2_X: REAL; (* AVG(n, Xi^2), AVG(n, Xi)^2 *)
BEGIN
   (* < Полный проход по последовательности и подсчёт n, avg_X2, avg2_X >*);
   ASSERT(n > 1, 20); (* Проверка предусловия для входной последовательности *)
   d := n / (n - 1) * (avg_X2 - avg2_X)
END CalcDisperse;
```

ДЕТАЛИЗАЦИЯ 2

Вводим вычисление AVG(n; xi). Для сокращения будем опускать у AVG и SUM вторую часть в скобках.

AVG(n) = SUM(n)/n

Хотя можно представить саму функцию AVG в рекуррентном виде:
AVG(n+1) = (AVG(n) * n + x)/(n + 1),
- это нерационально, лучше рекуррентно считать просто SUM(n), а AVG вычислять после окончания цикла.

Сумма рекуррентно представляется очевидным образом:

SUM(n+1) = SUM(n) + X_(n+1)

Таким образом, в цикле расчитываются рекуррентные последовательности n, SUM(n, X) и SUM(n, X^2) для n = 1 .. <количество_элементов_во_входном_потоке>.

```
PROCEDURE CalcDisperse* (IN rd: TextMappers.Scanner; OUT d: REAL);
(* IN rd - курьер для входной последовательности; OUT d - несмещённая дисперсия для последовательности
   Предусловия:
      20    количество элементов последовательности > 1
*)
   VAR n: INTEGER; sum, sum2: REAL; (* Рекуррентные переменные *)
      (* INVAR (n, sum, sum2):
         FOR ANY n :
            n - количество элементов в обработанной последовательности
            & sum = SUM(i = 1..n; xi)
            & sum2 = SUM(i = 1..n; xi^2)
      *)
         avg_X2, avg2_X: REAL; (* AVG(n, Xi^2), AVG(n, Xi)^2 *)
BEGIN
```

```
    (* < Полный проход по последовательности и подсчёт n, sum, sum2 >*);
    avg_X2 := sum2 / n; avg2_X := Math.IntPower(sum / n, 2);
    ASSERT(n > 1, 20); (* Проверка предусловия для входной последовательности *)
    d := n / (n - 1) * (avg_X2 - avg2_X)
END CalcDisperse;
```

ДЕТАЛИЗАЦИЯ 3

Строим цикл рекуррентного расчёта n, sum, sum2.

Условие и продвижение цикла по последовательности строим шаблонно, по базовой схеме "Полный проход", (см. [3], [4]). Остаётся только корректно добавить обработку текущего элемента и изменения рекуррентных переменных n, sum и sum2 с сохранением INVAR(n, sum, sum2).

```
PROCEDURE CalcDisperse* (IN rd: TextMappers.Scanner; OUT d: REAL);
(* IN rd - курьер для входной последовательности; OUT d - несмещённая дисперсия для последовательности
    Предусловия:
        20    количество элементов последовательности > 1
*)
    VAR n: INTEGER; sum, sum2: REAL; (* Рекуррентные переменные *)
        (* INVAR (n, sum, sum2):
            FOR ANY n :
                n - количество элементов в обработанной последовательности
                & sum = SUM(i = 1..n; xi)
                & sum2 = SUM(i = 1..n; xi^2)
        *)
            avg_X2, avg2_X: REAL; (* AVG(n, Xi^2), AVG(n, Xi)^2 *)
BEGIN
    (* Устанавливаем начальные значения *)
    n := 0; sum := 0; sum2 := 0;
    (* ASSERT( INVAR (n, sum, sum2) ) *)
    rd.Scan;
    WHILE rd.type = TextMappers.real DO
        (*< Обработка rd.real и восстановление INVAR(n, sum, sum2) >*);
        rd.Scan
    END;
    (* ASSERT( INVAR(n, sum, sum2) & (n = количество_всех_элементов) )
        => sum и sum2 содержат сумму элементов и сумму квадратов элементов последовательности соответственно *)
    avg_X2 := sum2 / n; avg2_X := Math.IntPower(sum / n, 2);
    ASSERT(n > 1, 20); (* Проверка предусловия для входной последовательности *)
    d := n / (n - 1) * (avg_X2 - avg2_X)
END CalcDisperse;
```

ДЕТАЛИЗАЦИЯ 4
Получение окончательного варианта

```
PROCEDURE CalcDisperse* (IN rd: TextMappers.Scanner; OUT d: REAL);
(* IN rd - курьер для входной последовательности; OUT d - несмещённая дисперсия для последовательности
    Предусловия:
        20    количество элементов последовательности > 1
*)
    VAR n: INTEGER; sum, sum2: REAL; (* Рекуррентные переменные *)
        (* INVAR (n, sum, sum2):
            FOR ANY n :
                n - количество элементов в обработанной последовательности
                & sum = SUM(i = 1..n; xi)
                & sum2 = SUM(i = 1..n; xi^2)
```

```
      *)
         avg_X2, avg2_X: REAL; (* AVG(n, Xi^2), AVG(n, Xi)^2 *)
BEGIN
   (* Устанавливаем начальные значения *)
   n := 0; sum := 0; sum2 := 0;
   (* ASSERT( INVAR (n, sum, sum2) ) *)
   rd.Scan;
   WHILE rd.type = TextMappers.real DO
      sum := sum + rd.real;
      sum2 := sum2 + Math.IntPower(rd.real, 2);
      (* Продвинулись вперёд в обработке, но нарушили инвариант... *)
      n := n + 1;
      (* Восстановили инвариант -  ASSERT( INVAR (n, sum, sum2) ) *)
      rd.Scan
   END;
   (* ASSERT( INVAR(n, sum, sum2) & (n = количество_всех_элементов) )
      => sum и sum2 содержат сумму элементов и сумму квадратов элементов последовательности соответственно *)
   avg_X2 := sum2 / n; avg2_X := Math.IntPower(sum / n, 2);
   ASSERT(n > 1, 20); (* Проверка предусловия для входной последовательности *)
   d := n / (n - 1) * (avg_X2 - avg2_X)
END CalcDisperse;
```

ДОКАЗАТЕЛЬСТВО ПРАВИЛЬНОСТИ АЛГОРИТМА

Теорема: Процедура CalcDisperse решает поставленную задачу.

[ ]
Утверждение теоремы верно по приведённому выше построению (конструктивное доказательство).
[x]

В особо ответственных случаях (криптография, встроенные системы и т.п.) полное доказательство с преобразованиями предикатов может быть развёрнуто из приведённого построения. Основным является построение программы "под доказательство".

Литература

1. Вирт Н. Систематическое программирование. Введение. - М.: Мир, 1977.
2. Дейкстра Э. Дисциплина программирования. - М.: Мир, 1978.
3. Грис Д. Наука программирования. - М.: Мир, 1981.
4. Касьянов В.Н., Сабельфельд В.К. Сборник заданий по практикуму на ЭВМ. - М.: Наука, 1985.
5. *http://oberoncore.ru/wiki/паттерны_циклов* - О базовых схемах циклов (в формулировках курса [4], изложение - И.Е. Ермаков).
6. Ткачев Ф.В. Специальный курс "Введение в современное программирование", физический факультет МГУ, читается с 2001 г. Раздаточный материал к лекции о циклах: *http://www.inr.ac.ru/~info21/08.pdf*
7. Кушниренко А.Г., Лебедев Г.Н. Программирование для математиков. - М.: Наука, 1988.
8. Кушниренко А.Г., Лебедев Г.Н., Сворень Р.А. Основы информатики и вычислительной техники. - М.: Просвещение, 1990.

** Электронные версии некоторых книг можно найти на EuroProg.ru.*

ПРИЛОЖЕНИЕ:
Исходный текст модуля и тестовой процедуры,
для запуска в среде BlackBox Component Builder

```
MODULE MyStat;

   IMPORT Math, TextMappers, DevCommanders, Log := StdLog;

   PROCEDURE CalcDisperse* (IN rd: TextMappers.Scanner; OUT d: REAL);
   (* IN rd - курьер для входной последовательности; OUT d - несмещённая дисперсия для последовательности
      Предусловия:
         20    количество элементов последовательности > 1
   *)
      VAR n: INTEGER; sum, sum2: REAL; (* Рекуррентные переменные *)
         (* INVAR (n, sum, sum2):
            FOR ANY n :
               n - количество элементов в обработанной последовательности
               & sum = SUM(i = 1..n; xi)
               & sum2 = SUM(i = 1..n; xi^2)
         *)
            avg_X2, avg2_X: REAL; (* AVG(n, Xi^2), AVG(n, Xi)^2 *)
   BEGIN
      (* Устанавливаем начальные значения *)
      n := 0; sum := 0; sum2 := 0;
      (* ASSERT( INVAR (n, sum, sum2) ) *)
      rd.Scan;
      WHILE rd.type = TextMappers.real DO
         sum := sum + rd.real;
         sum2 := sum2 + Math.IntPower(rd.real, 2);
         (* Продвинулись вперёд в обработке, но нарушили инвариант... *)
         n := n + 1;
         (* Восстановили инвариант -  ASSERT( INVAR (n, sum, sum2) ) *)
         rd.Scan
      END;
      (* ASSERT( INVAR(n, sum, sum2) & (n = количество_всех_элементов) )
         => sum и sum2 содержат сумму элементов и сумму квадратов элементов последовательности соответственно *)
      avg_X2 := sum2 / n; avg2_X := Math.IntPower(sum / n, 2);
      ASSERT(n > 1, 20); (* Проверка предусловия для входной последовательности *)
      d := n / (n - 1) * (avg_X2 - avg2_X)
   END CalcDisperse;

   PROCEDURE Test*;
      VAR sc: TextMappers.Scanner;
            d: REAL;
   BEGIN
      sc.ConnectTo(DevCommanders.par.text);
      sc.SetPos(DevCommanders.par.beg);
      CalcDisperse(sc, d);
      Log.Real(d)
   END Test;

END MyStat.
```

❗MyStat.Test 2.0 10.0 5.0 8.0❗

Не забываем ставить .0, иначе TextMappers.Scanner трактует как INTEGER, а усложнять цикл двумя вариантами считывания мы не стали.