В.И.Шелехов, Н.С.Карнаухов

ДЕМОНСТРАЦИЯ ТЕХНОЛОГИИ ПРЕДИКАТНОГО ПРОГРАММИРОВАНИЯ НА ЗАДАЧЕ СОРТИРОВКИ ПРОСТЫМИ ВСТАВКАМИ

В данной работе демонстрируются типовые элементы технологии предикатного программирования на примере задачи **сортировки простыми вставками** [1]. Показано построение двух вариантов предикатной программы и реализация эффективной императивной программы. Читателю предварительно следует ознакомиться с работой [2] настоящего сборника.

Допустим, имеется массив a[low..hgh] с элементами некоторого типа T, где low — нижняя, а hgh — верхняя граница массива. Полагаем, что для любой пары значений типа T определены операции сравнения <, <=, > и >=. Необходимо реализовать сортировку массива методом простых вставок [1]. В этом методе реализуется итерация по элементам. Очередной i-ый элемент перемещается в подходящее место среди ранее упорядоченных элементов a[low] <= a[low+1] <=...<= a[i-1]. Массив, полученный на i-ом шаге итерации, оказывается упорядоченным для элементов с индексами от low до i.

Тип массива a имеет следующее описание:

```
type Ar(int l, h, type T) = array l..h of T;
```

Поскольку low, high и T являются постоянными параметрами программы сортировки, они определяются следующими глобальными описаниями:

```
int low,high;
type T;
type ArT = Ar(low, high, T);
```

Последнее описание определяет обозначение ArT, в котором параметры отсутствуют, но подразумеваются. Обозначение ArT используется для сокращения записи.

Задача сортировки массива a представляется предикатом sort(ArT a: ArT a'), где a' есть упорядоченный массив, являющийся перестановкой массива a. Предполагается, что результат сортировки должен быть получен в том же массиве a, поэтому в реализации массивы a и a' должны быть склеены. Напомним, что вместо sort(ArT a: ArT a') можно писать sort(ArT a*:).

Определение предиката sort реализуется через обобщающий предикат sorti(a, i: a'), сортирующий массив a при условии, что его элементы a[low],...,a[i-1] уже отсортированы.

```
sort(ArT a*: ) // a' — перестановка a, a' — упорядочен
{       if low >= high then
                a' = a
        else
                sorti(a, low+1: a')
        end
}
```

В определении предиката sorti используется вызов предиката pop_into, вставляющий элемент a[i] внутрь отсортированной части a[low],...,a[i-1].

```
sorti(ArT a*, int i: )
        // a[low..i-1]  — упорядочен, a' — перестановка a, a' — упорядочен
{   pop_into(a, i, a[i]: ArT c);
    if i = high then
        a' = c
    else
        sorti(c, i+1: a')
    end
}
```

В определении предиката pop_into элемент e=a[j] вставляется внутрь отсортированной части a[low],...,a[j-1]. Массив a' является перестановкой a с отсортированными элементами a'[low],...,a'[j].

```
pop_into(ArT a*, int j, T e: )  // a[low..j-1], a'[low..j]  — упорядочены, e=a[j],
                                // a'[j+1..high] = a[j+1..high]
{   if j = low or a[j-1] <= e then
      a' = a
    else
      ArT c = a(j-1: e, j: a[j-1]);
      pop_into(c, j-1, e: a')
    end
}
```

Массив c строится из массива a с помощью конструктора: c совпадает с a за исключением элементов a[j] и a[j-1], которые меняются местами, так как e=a[j]. Поэтому c является перестановкой a.

Определения предикатов sort, sorti и pop_into, а также описания типов и глобальных переменных представляют полную предикатную программу

сортировки. В этой программе элемент e = a[j] последовательно обменивается с соседними элементами a[j-1], a[j-2] и т.д., пока не окажется отсортированным. Понятно, что это не самая эффективная программа, поскольку элемент e можно сразу перенести в нужное место, сдвигая на одну позицию вверх все промежуточные элементы. Построим новую версию программы, реализующую подобный перенос элемента e.

Во-первых, вместо обмена элементов a[j-1] и e=a[j] в конструкторе ArT c = a(j-1: e, j: a[j-1]) следует просто перемещать элемент a[j-1] с помощью ArT c = a(j: a[j-1]). Новый массив c не является перестановкой для a. Перестановкой, соответствующей перемещению a[j-1], является (c[low],...,c[j-2], e, c[j],...,c[high]). Во-вторых, в качестве исходного упорядочиваемого массива следует рассматривать: d = (a[low],...,a[j-1], e,a[j+1],..., a[high]), поскольку условие e=a[j] уже не выполняется. Будем использовать новый предикат shift(a, j, e: a'), который перемещает элемент e массива a в соответствующее место, так что a'[low..j] становится упорядоченным.

```
pop_into(ArT a*, int j, T e: )  // a[low..j-1], a'[low..j]  — упорядочены, e=a[j],
                                // a'[j+1..high] = a[j+1..high], j > low
{     if a[j-1] <= e then
        a' = a
      else
        shift(a, j, e: a')
      end
}
```

Предикат shift вставляет элемент e внутрь отсортированной части a[low],...,a[j-1]. Массив a' является перестановкой массива (a[low],...,a[j-1], e, a[j+1],...,a[high]). Элементы a'[low],...,a'[j] являются упорядоченными.

```
shift(ArT a*, int j, T e: ) // a[low..j-1], a'[low..j]  — упорядочены, a[j-1] > e,
                            // a'[j+1..high] = a[j+1..high], j > low
                            // a' — перестановка (a[low..j-1], e, a[j+1..high])
  {   ArT c = a(j: a[j-1]);
      if j-1 = low or a[j-2] <= e then
          a' = c(j-1: e)
      else
          shift(c, j-1, e: a')
      end
  }
```

Доказательство того, что предикатная программа является правильной, т.е. что определение каждого предиката удовлетворяет его спецификации, не вызывает принципиальных затруднений. В случае записи спецификаций на формальном языке предикатная программа в принципе может быть автоматически верифицирована.

Для предикатной программы, составленной из определений предикатов sort, sorti, pop_into и shift, построим соответствующую императивную программу.

Сначала склеим переменные a, a' и c во всех определениях и проведем очевидные упрощения. При этом конструктор ArT c = a(**j:** a[j-1]) заменяется оператором присваивания. a[j] := a[j-1]. Далее, заменим хвостовую рекурсию циклом. Получим:

```
sort(ArT a*: )
{     if  low < high then
          sorti(a, low+1: a)
      end
}
sorti(ArT a*, int i: )
{     loop
          pop_into(a, i, a[i]: a);
          if i = high then
              exit
          else
              i:= i+1
          end
end
}
pop_into(ArT a*, int j, T e: )
{     if a[j-1] > e then
        shift(a, j, e: a)
      end
}
```

```
shift(ArT a*, int j, T e: )
    loop
        a[j] := a[j-1];
        if j-1 = low or a[j-2] <= e then
          a[j-1] := e; exit
        else
          j := j-1
        end
    end
}
```

В полученной программе проведем подстановку определения предиката shift в тело pop_into. Полученное определение pop_into подставляется в тело sorti, а затем sorti — в тело sort. В результате подстановки параметры i, j и e становятся локалами.

```
sort(ArT a*: )
{     if low < high then
          int i := low+1;
          loop
              int j := i; T e := a[i];
              if a[j-1] > e then
                  loop
                    a[j] := a[j-1];
                    if j-1 = low or a[j-2] <= e then
                         a[j-1] := e; exit
                    else j := j-1
                    end
                  end
              end
          end ;
          if i = high then  exit
          else  i:= i+1
          end
      end
}
```

Заменяя объемлющий цикл **loop** на цикл **for**, получаем итоговую императивную программу:

```
sort(ArT a*: )
{     if low < high then
          for int i := low+1.. high do
               int j := i; T e := a[i];
               if a[j-1] > e then
                    loop
                         a[j] := a[j-1];
                         if j-1 = low or a[j-2] <= e then
                              a[j-1] := e; exit
                         else j := j-1
                         end
                    end
               end
          end
     end
}
```

Последнее преобразование не меняет процесса исполнения программы. Оно относится к **оформлению**, улучшающему структуру программы.

СПИСОК ЛИТЕРАТУРЫ

1. **Кнут Д.** Искусство программирования для ЭВМ. Т.3: Сортировка и поиск. Пер. с англ. — М.: Мир, 1978. — 843с.
2. **Шелехов В.И.** Предикатное программирование: основы, язык, технология // Наст. сб. — С. 7–15.