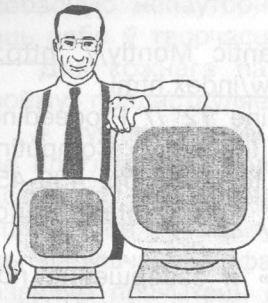


***А.И. Павловский**, к.ф-м.н., профессор, заведующий кафедрой прикладной математики и информатики БГПУ им. М. Танка,*
***А.Ф. Климович**, аспирантка кафедры прикладной математики и информатики БГПУ им. М. Танка*

Построение алгоритмов методом пошаговой детализации с использованием языка ИнтАл

В информатике и конструктивной математике основными являются проблемы алгоритмизации. Как отмечается в работах [1, 2], под алгоритмизацией понимается описание функционирования системы или процесса, в частности процесса решения задачи или класса задач путем построения и обоснования алгоритма. В ходе алгоритмизации можно выделить четыре этапа: разработка алгоритма, обоснование, представление и анализ.

В настоящее время значительное количество работ посвящено вопросам разработки алгоритмов. Так, фундаментальные принципы разработки алгоритмов излагаются в недавно вышедшем труде Кормена Т., Лейзерсона Ч., Ривеста Р. «Алгоритмы: построение и анализ» [3]. В учебном пособии [1] представлены теоретические принципы, технология практической разработки и анализ алгоритмов на уровне педвузовского образования. В статье Шевченко Т.Г. [4] об-

суждаются пути преподавания основ алгоритмизации в курсе информатики средней школы. Паронджанов В.Д. [5] предлагает методику разработки алгоритмов с помощью языка визуального программирования ДРАКОН, разработанного на основе формализации и структуризации блок-схем алгоритмов и программ.

В данной статье предлагается использование метода пошаговой детализации при разработке алгоритмов на УАЯ ИнтАл.

Структурное программирование: основы и стратегия

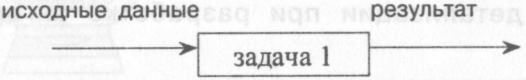
В 70-е годы в связи с известными трудностями в разработке алгоритмов (ухудшение «читабельности» программ, снижение их эффективности и надежности, усложнение тестирования) была разработана *методология структурного программирования (проектирования)*, которая рассматривает алгоритм (программу), как совокупность иерархических модулей, которые позволяют четко структурировать программу, что улучшает ее понимание разработчиками и позволяет выполнить математическое доказательство ее корректности, а, следовательно, повышает надежность функционирования программ и сокращает время ее разработки.

Основными принципами структурного программирования [6] являются *нисходящее проектирование* и *модульное программирование*. Нисходящее проектирование заключается в последовательном разбиении задачи на все более мелкие и мелкие участки, т. е. процесс программирования идет «сверху вниз». Модульное программирование предполагает создание для каждого такого участка отдельной автономной программы – модуля. Специально созданная программа объединяет все модули в единое целое и управляет их работой.

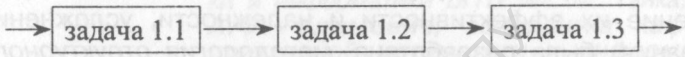
Для построения структурированных алгоритмов нужно иметь систематическую процедуру (технологию) [1], позволяющую строить алгоритмы по четким правилам. Такая процедура была разработана и называется она *пошаговой детализацией* (*последовательным уточнением, нисходящей разработкой*) – это технология построения структурированных алгоритмов, позволяющая последовательно, шаг за шагом детализировать, уточнять отдельные этапы алгоритма. Детали-

зация завершается тогда, когда в описании алгоритма не остается этапов, не реализованных конструкциями некоторого языка, например УАЯ ИнтАл.

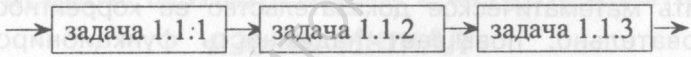
Непомнящий А.Б. раскрывает суть процесса пошаговой детализации схематически [7]. Автор представляет любой алгоритм в виде блока:



Сложная задача 1 может быть разбита на ряд более простых задач (подзадач):



При этом каждая из подзадач может быть разбита в свою очередь на более простые подзадачи. Например, задача 1.1 может быть разбита на три подзадачи:



Процесс деления на подзадачи может быть продолжен. Важно, чтобы для каждой подзадачи были четко определены исходные данные, результаты и содержание обработки исходных данных, т. е. указано назначение задачи и найден метод ее решения. В процессе поиска метода решения и происходит деление задачи на подзадачи.

Принципиальная основа структурного программирования выражена в *фундаментальной аксиоме замещения*, справедливой для структурированных алгоритмов. Согласно этой аксиоме, любой функциональный блок алгоритма можно заместить базовой структурой (*следование, ветвление, повторение*) из исходного базового множества.

Рассмотрим условный пример пошаговой детализации алгоритма, записанного в виде блок-схемы. Пусть существует задача с идеей **А1** (рис. 1а).

После первичного анализа задачи выясняем, что для ее реализации необходимо использовать, например, структуру

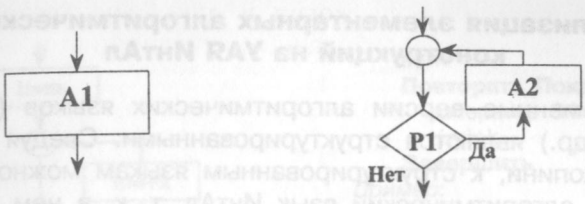


Рис. 1.

повторения. С учетом вышесказанного уточняем блок-схему так, как показано на рис. 1б.

Дальнейший анализ привел к необходимости замещения блока **A2** полной конструкцией ветвления (рис. 2), а затем блок **A3** был уточнен неполной конструкцией ветвления (рис. 3).

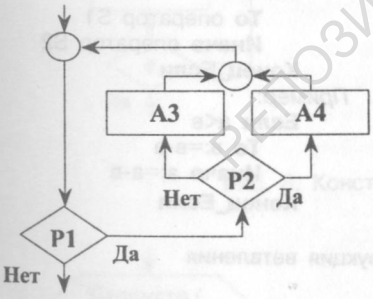


Рис. 2.

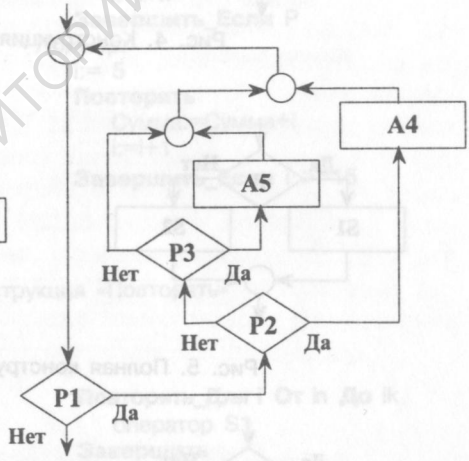
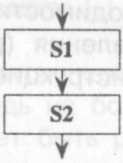


Рис. 3.

Таким образом, блок-схема, изображенная на рис. 3, является блок-схемой структурированного алгоритма, разработанного методом пошагового уточнения.

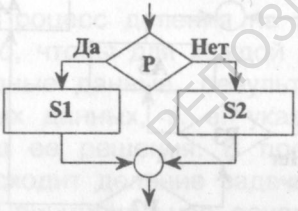
Реализация элементарных алгоритмических конструкций на УАЯ ИнтАл

Современные версии алгоритмических языков (Паскаль, Бейсик и др.) являются структурированными. Следуя теореме Бома и Якопини, к структурированным языкам можно отнести и учебный алгоритмический язык ИнтАл, т. к. в нем реализованы базовые структуры (следование, ветвление и цикл-пока) Кроме того, в учебных целях может использоваться более трех базовых конструкций. Так, для УАЯ ИнтАл [8] базовыми можно считать конструкции: следование, ветвление (полная и неполная команда «Если»), повторение (команды: «Повторять_Пока», «Повторять», «Повторять_Для»), выбор (полная и неполная команда «Выбрать_По») (рис. 4-11).




```
- - - - - - - -
оператор S1
оператор S2
- - - - - - - -
```

Рис. 4. Конструкция следования

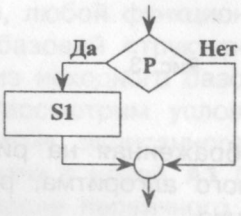



```
Если P
   То оператор S1
   Иначе оператор S2
Конец_Если
```

Пример:

```
Если a<в
   То в:=в-а
   Иначе а:=а-в
Конец_Если
```

Рис. 5. Полная конструкция ветвления

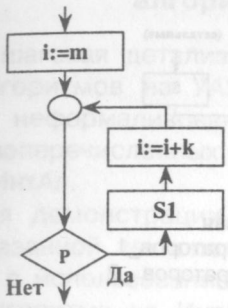



```
Если P
   То оператор S1
Конец_Если
```

Пример:

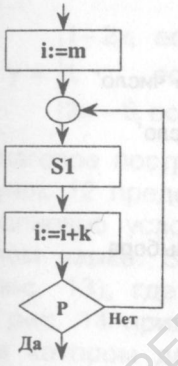
```
Если x>m
   То m:=x
      n:=n+1
Конец_Если
```

Рис. 6. Неполная конструкция ветвления



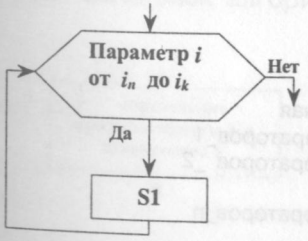

```
i:=m
Повторять_Пока P
    оператор S1
    i:=i+k
Завершить
Пример:
    i:= 5
Повторять_Пока i<=15
    Сумма:=Сумма+i
    i:=i+1
Завершить
```

Рис. 7. Конструкция «Повторять_Пока»



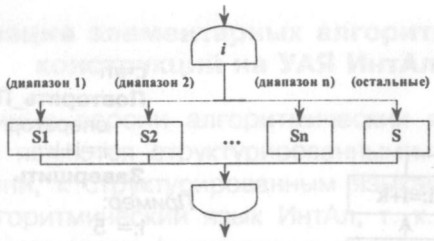

```
i:=m
Повторять
    оператор S1
    i:=i+k
Завершить_Если P
Пример:
    i:= 5
Повторять
    Сумма:=Сумма+i
    i:=i+1
Завершить_Если i >=15
```

Рис. 8. Конструкция «Повторять»




```
Повторять_Для i От in До ik
    оператор S1
Завершить
Пример:
Повторять_Для i От 5 До 15
    Сумма:=Сумма+i
Завершить
```

Рис. 9. Конструкция «Повторять_Для»

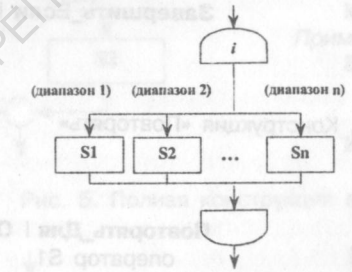


Выбрать_По селекторная_переменная
диапазон_значений_1 : блок_операторов_1
диапазон_значений_2 : блок_операторов _2
- - -
диапазон_значений_n : блок_операторов_n
Иначе блок_операторов_для_иных
Конец_Выбора

Пример:

Выбрать_По x
0 : Стр:=' x=0'
Флаг:=Истина
1..32767 : Стр:=' x – положительное число'
Флаг:=Истина
Иначе Стр:=' x – отрицательное число'
Флаг:=Ложь
Конец_Выбора

Рис. 10. Полная конструкция выбора



Выбрать_По селекторная_переменная
диапазон_значений_1 : блок_операторов_1
диапазон_значений_2 : блок_операторов _2
- - -
диапазон_значений_n : блок_операторов_n
Конец_Выбора

Рис. 11. Неполная конструкция выбора

Пошаговая детализация при разработке алгоритмов на УАЯ ИнтАл

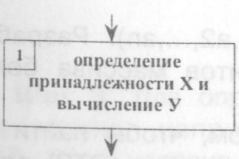
Пошаговая детализация при разработке структурированных алгоритмов на УАЯ ИнтАл состоит в систематической замене неформализованного этапа (части) алгоритма одной из вышеперечисленных конструкций (команд) учебного алгоязыка ИнтАл.

Для демонстрации разработки алгоритмов с помощью вышеуказанной технологии приведем два примера, реализованных с использованием блок-схем и языка ИнтАл. При записи алгоритма на ИнтАл неформализованные функциональные блоки будем заключать в широкие круглые скобки.

Пример 1. Построить структурированный алгоритм для вычисления функции:

$$y = \begin{cases} 1 - 2x, & \text{если } x < 0 \\ 1, & \text{если } 0 \le x < 1 \\ 2x - 5, & \text{если } x \ge 1 \end{cases}$$

Пошаговое построение алгоритма приведено на рис. 12-14. Рисунок 12 представляет задачу в виде алгоритма, в котором записано условие задачи в словесной форме на естественном языке. Затем эта запись преобразована в развилку (рис. 13), где выделено вычисление значения y при x<0. На рис. 14 приведен окончательный алгоритм на языке ИнтАл, в котором действия предыдущего алгоритма, записанные в естественной форме, заменены развилкой. В итоге выполнения двух шагов мы получили структурированный алгоритм решения задачи. В окончательном варианте решения введен заголовок алгоритма и операции ввода/вывода.

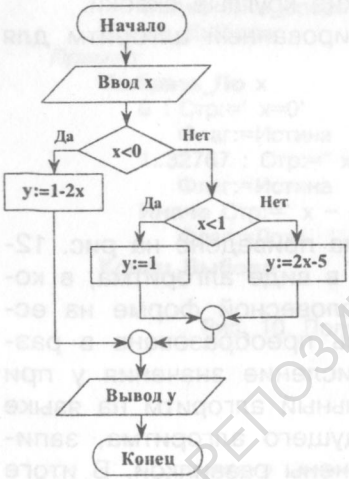


Программа Функция
Определение принадлежности
X и вычислениеY
Конец_Программы

Рис. 12.


```
Если x<0
  То y:=1-2*x
  Иначе продолжение
        исследования X
        и определения Y
Конец_Если
```

Рис. 13.




```
Программа Функция
Описание
  x, y: Вещественный
Конец_Описания
Ввод(x,'x=')
Если x<0
  То y:=1-2*x
  Иначе
  Если x<1
    То y:=1
    Иначе y:=2*x-5
  Конец_Если
Конец_Если
Вывод('y=',y)
Конец_Программы
```

Рис. 14.

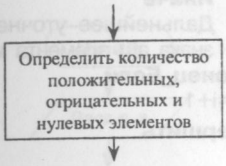
Пример 2. Задан массив A=(a1,a2,...,an). Разработать алгоритм подсчета количества элементов массива больших нуля, меньших нуля и равных нулю.

Идея алгоритма заключается в том, чтобы найти количество положительных (*POL*) и отрицательных (*OTR*) элементов массива, а затем, зная значения *n*, *POL* и *OTR* найти количество нулевых элементов по формуле:

NUL=n-OTR-POL.

Последовательное построение блок-схемы и программы на языке ИнтАл показано на рис. 15-20.

Шаг 1. Представим задачу в виде крупного блока, который содержит ее текстовое условие (рис. 15).



Программа Знаки
Определить количество положительных, отрицательных и нулевых элементов массива
Конец_Программы

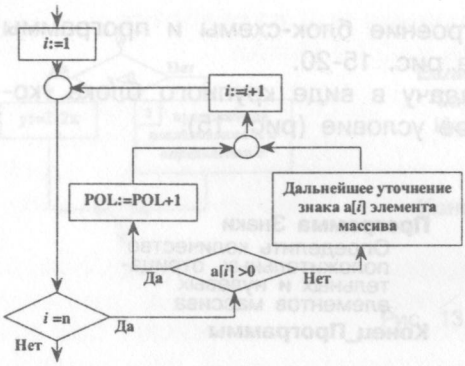
Рис. 15.

Шаг 2. Для определения знака каждого элемента необходимо поочередно перебрать все элементы массива. Для организации перебора воспользуемся, например, конструкцией цикла «Пока», который организуем по промежуточной переменной *i* (рис. 16).

i:=1
Повторять_Пока i<=n
Определение знака a[i] элемента массива
i:=i+1
Завершить

Рис. 16.

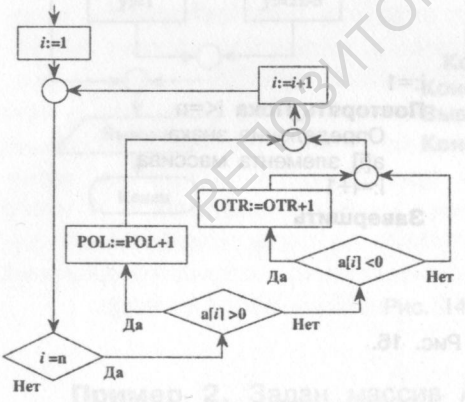
Шаг 3. Для определения знака элемента массива сначала можно, например, проверить следующее условие: a[i]>0. Если условие истинно, то POL:=POL+1. Если условие ложно, то необходимо далее уточнить знак a[i] элемента. Для реализации шага воспользуемся полной конструкцией ветвления (рис. 17).




```
i:=1
Повторять_Пока i<=n
   Если a[i] >0
      То POL:=POL+1
      Иначе
      Дальнейшее уточнение
      знака a[i] элемента массива
   Конец_Если
   i:=i+1
Завершить
```

Рис. 17.

Шаг 4. Дальнейшее уточнение знака a[i] элемента осуществляется с помощью неполной конструкции ветвления, в которой проверяется условие a[i]<*0*. Если условие истинно, то OTR:=OTR+1. При невыполнении этого условия необходимо переходить к следующему элементу массива (рис. 18).




```
i:=1
Повторять_Пока i<=n
   Если a[i] >0
      То POL:=POL+1
      Иначе
         Если a[i] <0
            То OTR:=OTR+1
         Конец_Если
   Конец_Если
   i:=i+1
Завершить
```

Рис. 18.

Шаг 5. Для завершения разработки алгоритма решения задачи остается на входе в цикл добавить операцию ввода количества элементов массива, сформировать массив и ус-

тановить начальные значения переменных POL и OTR, а на выходе из цикла вычислить значение переменной NUL и вывести полученные результаты (рис. 19-20).

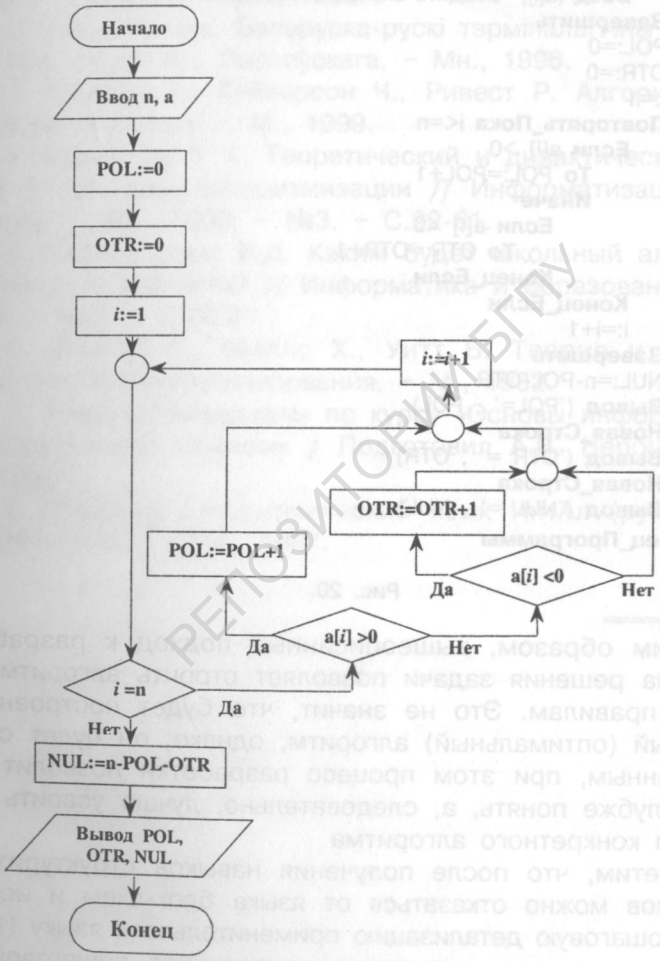


Рис. 19.

```
Программа Знаки
Описание
    a: Массив[10] Целый
    n, i, POL, OTR, NUL: Целый
Конец_Описания
Ввод (n, 'Введите количество элементов массива')
Повторять_Для i От 1 До n
    Ввод (a[i], 'Введите значение элемента массива')
Завершить
POL:=0
OTR:=0
i:=1
Повторять_Пока i<=n
    Если a[i] >0
        То POL:=POL+1
        Иначе
            Если a[i] <0
                То OTR:=OTR+1
            Конец_Если
    Конец_Если
    i:=i+1
Завершить
NUL:=n-POL-OTR
Вывод ('POL=' , POL)
Новая_Строка
Вывод ('OTR =' , OTR)
Новая_Строка
Вывод ('NUL =' , NUL)
Конец_Программы
```

Рис. 20.

Таким образом, вышеописанный подход к разработке алгоритма решения задачи позволяет строить алгоритмы по единым правилам. Это не значит, что будет построен эффективный (оптимальный) алгоритм, однако, он будет структурированным, при этом процесс разработки позволит учащимся глубже понять, а, следовательно, лучше усвоить особенности конкретного алгоритма.

Отметим, что после получения навыков структуризации алгоритмов можно отказаться от языка блок-схем и использовать пошаговую детализацию применительно к языку ИнтАл.

На наш взгляд, использование принципа пошаговой детализации при разработке алгоритмов в базовом курсе информатики средней школы, кроме того, будет способствовать

более эффективному формированию логико-алгоритмического мышления и обучения основам алгоритмизации в целом.

Литература

1. Павловский А. И. Алгоритмы: разработка и анализ: Учебное пособие. – Мн., 1999.
2. Інфарматыка: Беларуска-рускі тэрміналагічны слоўнік / Пад рэд. праф. А.І. Паўлоўскага. – Мн., 1996.
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М., 1999.
4. Шевченко Т. Г. Теоретический и дидактический материал по основам алгоритмизации // Информатизация образования. – Мн., 1999. – №3. – С.22-61.
5. Паронджанов В.Д. Каким будет школьный алгоритмический язык XXI века? // Информатика и образование. – М., 1994. – №3. – С.76-91.
6. Лингер Р., Миллс Х., Уитт Б. Теория и практика структурного программирования. – М., 1982.
7. Учебные материалы по курсу «Основы информатики и вычислительной техники» / Подготовил А.Б. Непомнящий – М., 1990.
8. Учебный алгоритмический язык ИнтАл (руководство программиста). – Мн., 1997.