

В. В. Кашей,

Академия социального управления, г. Москва,

ИСПОЛЬЗОВАНИЕ СРЕДЫ АВТОМАТИЗИРОВАННОЙ РАЗРАБОТКИ АЛГОРИТМОВ И ПРОГРАММ В КУРСЕ ИНФОРМАТИКИ

Аннотация

Определяется оптимальный способ записи алгоритма при его разработке, основанный на требованиях структурного программирования (структурной теореме), удобства его записи и обработки в компьютере, возможности определения ошибок в структуре алгоритма, автоматизации записи алгоритма средствами среды разработки алгоритма и перевода его в программу. Описывается среда разработки алгоритма и перевода его в программу на алгоритмическом языке. Рассматриваются методические аспекты использования данной среды в курсе информатики основной и средней общеобразовательной школы.

Ключевые слова: алгоритм, структурное программирование, способ записи алгоритма, управляющая структура, псевдокод, среда автоматизированной разработки.

Задача компьютера состоит в облегчении интеллектуального труда человека в различных сферах деятельности, включая разработку программного обеспечения. RAD-системы предоставляют широкий спектр средств ускорения и облегчения разработки программ. В то же время при разработке алгоритма приходится выполнять достаточно большой объем нетворческой, рутинной работы для записи алгоритма в заданной форме. Оценим возможность автоматизации этой работы путем создания среды для быстрой разработки алгоритма.

Основой для разработки алгоритма является **структурная теорема** (более известна под названием **основной теоремы программирования**) [4], которая утверждает, что любой исполняемый алгоритм может быть реализован с помощью базовых управляющих структур трех видов: последовательностей (*sequence*), ветвлений (*selection*) и повторов (*repetition*).

Для описания алгоритмов введем понятие функционального блока. **Функциональный блок** — это отдельный фрагмент алгоритма, выполняющий

определенный набор действий и не зависящий при своей реализации от остальных фрагментов. Функциональные блоки могут соединяться между собой с использованием упомянутых выше базовых управляющих структур.

В процессе разработки структурированного алгоритма у каждого функционального блока должен быть только один вход и только один выход. Невыполнение этого правила может привести к нарушению структуры алгоритма и как следствие — к высокой вероятности появления ошибок при его разработке.

При разработке алгоритмов можно использовать различные способы их визуализации. **Основные способы визуализации алгоритмов** предполагают использование графических схем, текстового описания или их комбинации. При визуализации базовых управляющих структур сначала будем использовать графический способ записи, а затем запишем тот же фрагмент алгоритма, используя текстовый способ записи.

Рассмотрим, как реализуются три основных типа базовых управляющих структур.

Контактная информация

Кашей Владимир Васильевич, канд. пед. наук, доцент кафедры информационно-коммуникационных технологий Академии социального управления, г. Москва; адрес: 129281, г. Москва, Староватутинский проезд, д. 8; телефон: (495) 472-32-08, доб. 149; e-mail: ww54@mail.ru

V. V. Kaschei,

Academy of Public Administration, Moscow

USING ENVIRONMENT OF THE AUTOMATED DEVELOPMENT OF ALGORITHMS AND SOFTWARE IN THE INFORMATICS COURSE

Abstract

The article describes the optimal method of recording an algorithm during its development based on the requirements of structured programming (structural theorem), the convenience of its recording and processing in the computer, the ability to identify errors in the structure of the algorithm, automated recording algorithm by means of the IDE algorithm and translating it into a program. The environment of developing algorithm and translating it into a program in the algorithmic language is described. The methodological aspects of using this system in the informatics course in secondary school are considered.

Keywords: algorithm, structured programming, how to write algorithm, control structure, pseudocode, automated development environment.

Первая базовая управляющая структура — «следование» (sequence).

Базовая управляющая структура «следование» представляет собой цепочку функциональных блоков, последовательно выполняющихся друг за другом (рис. 1). Эта последовательность остается неизменной при любых обстоятельствах, поэтому здесь нет условий и не может быть управления при выполнении алгоритма. Следовательно, все функциональные блоки имеют тот же уровень вложенности (детализации), что и исходный функциональный блок.

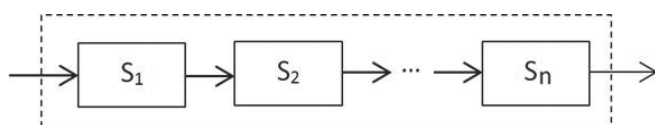


Рис. 1. Базовая управляющая структура «следование»

Текстовая визуализация такой структуры проста и состоит из нескольких строк, каждая из которых соответствует одному функциональному блоку. При записи описания действия, которое задает функциональный блок, длины строки может не хватить и текст придется переносить на новую строку, хотя описание еще не закончено. Чтобы понимать, где начинается новое действие, а где продолжается старое, рекомендуется каждую новую строку действий начинать со специального символа. В качестве такого символа будем использовать звездочку. Этот символ входит в число первых 128 символов кода ASCII, а значит, будет всегда верно отображаться на экране. Звездочка достаточно редко встречается в записи алгоритмов, например, для обозначения операции умножения. Однако в этом случае слева и справа от нее записываются сомножители. В нашем случае звездочка будет находиться в начале строки и слева от нее не предполагается наличия символов. Кроме того, звездочку легко набрать на дополнительной цифровой клавиатуре независимо от состояния переключателя NumLock. Количество звездочек будет определять уровень вложенности функционального блока.

Такой способ визуализации алгоритма будем называть **псевдокодом**. Термин «псевдокод» указывает, что данный текст не является строго формализованной записью. При записи алгоритмов на псевдокоде жестко определяются только формат записи, определяющий базовые управляющие конструкции путем использования нескольких ключевых слов для описания управления ходом исполнения алгоритма, и правила использования символов, определяющих уровни вложенности базовых управляющих конструкций. За исключением этих ограничений при описании алгоритма с помощью псевдокода допускается произвольная форма записи для остального текста.

Базовая управляющая структура типа «следование» записывается на псевдокоде в следующем виде:

```
* действие 1
* действие 2
...
* действие n
```

Вторая базовая управляющая структура — «ветвление» (selection).

Данный функциональный блок имеет только один выход (рис. 2). Поэтому выходы функциональных блоков S1 и S2 должны быть соединены до завершения функционального блока структуры «ветвление».

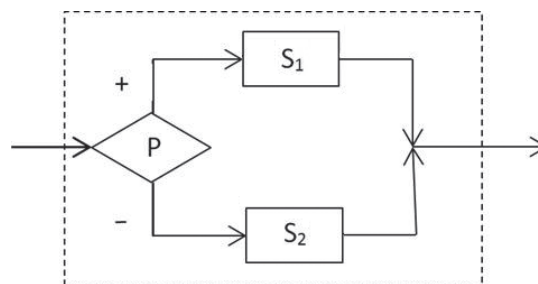


Рис. 2. Базовая управляющая структура «ветвление»

Внутри исходного функционального блока в данном случае имеется управляющая структура, отображаемая на рисунке в виде ромба-условия и соединительных линий. Функциональные блоки S1 и S2 входят как составляющая часть внутренней структуры функционального блока «ветвление» и используются в зависимости от выполнения условия, поэтому их уровень вложенности увеличивается на единицу, что отображается дополнительной звездочкой в начале строки при записи этих функциональных блоков.

С помощью псевдокодов эта структура записывается следующим образом:

```
* если <условие (P — предикат, логическое
  выражение)>
* * <действие1 (S1)>
* иначе
* * <действие2 (S2)>
* конец-если
```

Смысл этой записи в том, что в зависимости от условия P выполняется действие1 (S1) или действие2 (S2). Для этой управляющей структуры определены ключевые слова «если», «иначе» и «конец-если». Подчеркивание ключевых слов не обязательно и используется здесь для большей наглядности.

Третья базовая управляющая структура — «повторение», «цикл» (repetition).

Эта базовая управляющая структура, как и базовая управляющая структура «ветвление», имеет в своем составе условие, поэтому включенный в ее состав функциональный блок имеет больший (на единицу) уровень вложенности (рис. 3).

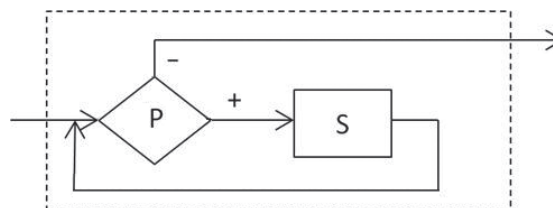


Рис. 3. Базовая управляющая структура «повторение», «цикл»

С помощью псевдокодов эта базовая управляющая структура записывается в следующем виде:

```
* цикл-пока <условие повторения цикла Р>
* * <действие S>
* конец-цикла
```

Каждая из трех базовых управляющих структур может быть представлена как один функциональный блок: если обратить внимание на прямоугольники, изображенные пунктирными линиями, можно заметить, что каждая из трех структур может быть вписана внутрь такого прямоугольника. Причем у такого прямоугольника (нарисованного штриховыми линиями) будет только одна входная линия и только одна выходная. Это значит, что любая из этих управляющих структур может рассматриваться как один функциональный блок и, наоборот, каждый функциональный блок может преобразоваться в одну из рассмотренных управляющих структур.

В зависимости от уровня детализации алгоритма мы можем или «сворачивать» алгоритм до более крупных блоков с меньшим уровнем детализации, вплоть до одного блока, или «раскрывать» функциональные блоки, записывая одну из трех структур вместо одного функционального блока. Если разработку алгоритма организовывать таким образом, что, начиная с одного функционального блока, действия в каждом блоке постепенно детализируются, разделяя эти блоки на более мелкие с более детализированным описанием, то в итоге получим алгоритм, состоящий из «элементарных» блоков, реализация которых на любом языке программирования будет предельно ясной и однозначно понимаемой.

Такой подход называется **проектированием «сверху вниз» или методом пошаговой детализации**. В этом случае в процессе детализации одного блока не надо обращать внимание на остальные блоки, в том числе одного уровня с разрабатываемым блоком, и можно сосредоточиться на разработке только одного блока.

Для удобства записи алгоритмов введем **три дополнительные управляющие структуры**, которые могут быть сведены к вышеописанным базовым управляющим конструкциям.

Сокращенная версия «если»:

```
* если <условие>
* * <действие>
* конец-если
```

Цикл с постусловием:

```
* цикл с постусловием
* * <действие>
* конец-цикла, когда <условие прекращения цикла>
```

Цикл с параметром:

```
* цикл по <параметр цикла> от <начальное значение параметра> до <конечное значение параметра>
* * <действие>
* конец-цикла
```

Ключевые слова псевдокодов выделены подчеркиванием.

Сравним рассмотренные выше способы записи алгоритма.

1. Запись алгоритма в графической форме более наглядна и понятна человеку, чем запись в текстовой форме, поэтому этот способ более распространен. Для описания алгоритмов в графической форме (схем алгоритмов) был разработан особый ГОСТ [1].
2. Представление алгоритма в графической форме наглядно и просто для понимания до тех пор, пока алгоритм не очень велик. Если схема алгоритма занимает несколько страниц и линии управления переходят с одного листа на другой, то разобраться в такой записи становится весьма непросто.
3. Формализовать запись алгоритма в графической форме гораздо сложнее, чем в текстовой. Использование псевдокодов позволяет формализовать эту запись и как следствие облегчает контроль правильности записи алгоритма и перевод его в программу на алгоритмическом языке.
4. Алгоритм, записанный на псевдокоде, проще сохранять на машинных носителях, чем графическую схему.
5. Текстовую запись легче обрабатывать средствами компьютера. Она легко преобразуется в программу на языке программирования. Такое преобразование в большей части является чисто механической работой, поэтому ее можно поручить компьютеру.

Использование псевдокодов для описания алгоритма в определенной степени реализовано в **школьном алгоритмическом языке** [2, 3], однако в такой реализации не определены:

- жесткость задания ключевых слов и базовых управляющих конструкций;
- наличие символов, определяющих уровень вложенности, позволяющих обеспечить контроль структуры алгоритма.

Уровни вложенности в школьном алгоритмическом языке задаются с использованием отступов (пробелов или табуляции) — неотображаемых символов, что недостаточно наглядно отображает структуру алгоритма. Используя символы, определяющие уровень вложенности при записи алгоритма, мы реализуем требования структурного программирования и обеспечиваем соблюдение принципа модульности, т. е. «автоматически» избегаем многих ошибок в структуре алгоритма уже на этапе его разработки.

Использование жестко формализованных шаблонов для ключевых слов псевдокода позволяет формализовать, а значит, и автоматизировать процесс перевода алгоритма, записанного на псевдокоде, в программу на языке программирования.

Для создания среды разработки алгоритма и преобразования его в программу на языке программирования необходимо программное обеспечение, которое не просто позволяет редактировать текст, но и может выполнять программируемые действия с ним. Этим требованиям удовлетворяет текстовый процессор. Наиболее распространенным и освоенным является текстовый процессор Word, имеющий достаточно мощный инструмент — макро-

команды, позволяющие программировать действия текстового процессора.

В процессе автоматизированного создания алгоритма и последующего его преобразования в программу на языке Паскаль используются следующие макрокоманды: устанавливающая признак начала алгоритма, вставки строки для описания нового действия, вставки шаблонов для структур ветвления и повторений, шаблонов действий для ввода и вывода, а также макрокоманда замены псевдокодов на операторы программы.

Для удобства вызова макросов можно создать на ленте процессора Word настраиваемую вкладку и разместить на ней пиктограммы вызова макрокоманд (рис. 4).

Для вставки очередной управляющей структуры курсор в предыдущей строке устанавливается на одну позицию правее самой правой звездочки в строке, чтобы программа могла «прочитать» и скопировать уровень структуры (все звездочки) в данной строке.

С целью облегчения создания раздела описания данных можно использовать макрокоманды создания шаблонов для ключевых слов и типов данных.

В процессе работы со средой создания алгоритма могут образоваться «пустые» строки, которые содержат только символы, определяющие уровень вложенности, — звездочки. Для их удаления используется макрокоманда удаления пустых строк, которая удаляет все такие строки. Рекомендуются использовать ее непосредственно перед командой преобразования алгоритма в программу, чтобы в тексте программы не оказалось пустых строк.

После применения команды преобразования алгоритма в текст программы получим текст программы. В этом тексте необходимо добавить в соответствующие места символ «точка с запятой» (;). Макрокоманда этого не делает, так как это требует семантического анализа текста программы. Эта операция оставляется учащемуся как возможность проверить его знание синтаксиса языка. В принципе, операцию расстановки символа «;» можно переложить на компьютер, если макрокоманды будут расставлять его уже при создании алгоритма в соответствующих строках. Однако в этом случае в описании алгоритма появятся символы «точка с запятой» (;), не несущие смысловой нагрузки на данной стадии разработки алгоритма.

Инсталляция рассмотренной среды разработки алгоритма состоит в импорте на компьютер двух файлов: файла с набором макрокоманд и файла настроек ленты текстового процессора Word. При использовании вкладки для вызова макрокоманд версия Word должна быть не ниже 2010.

Среда разработки алгоритма является открытой системой и может быть дополнена и модифицирована как в части используемых макрокоманд, так и в части отображения их на ленте Word. Процесс

инсталляции системы на компьютере с установленным текстовым процессором Word занимает не более пяти минут.

При использовании вышеописанной среды разработки алгоритмов и программ в курсе информатики возникает **ряд проблем**.

Во-первых, считается, что использование графических схем при объяснении создания алгоритма проще для восприятия учащимися за счет большей наглядности графического метода и лучшего понимания человеком информации в графической форме. Это верно при первоначальном обучении созданию простых алгоритмов. Однако, как только алгоритм становится достаточно сложным, особенно если он не уместится на одной странице, простота и наглядность графической схемы пропадают и псевдокод становится проще и надежнее за счет жесткого следования требованиям структуры (звездочки) и возможности использования формальных действий в соответствии с заданными правилами, что позволяет перевести часть творческой работы в чисто формальную, механически выполняемую.

Во-вторых, при использовании псевдокода и «ручной» записи, когда ученик сам пишет ключевые слова псевдокода, сам проставляет структуру алгоритма (звездочки), он осознает и автоматически запоминает за счет моторной памяти правила записи ключевых слов и структуры. Переход на автоматизированную систему, когда часть работы за него выполняет компьютер, в значительно меньшей степени формирует навык создания алгоритмов и особенно программ. В этом случае можно предложить начинать использование среды разработки алгоритма учащимися не сразу, а после того как у них выработается необходимый навык. Можно ограничить использование среды только занятиями в классе, когда время создания программы критично и требуется рассмотреть максимальное число задач за ограниченное время, оставив «ручную» запись для выполнения домашних заданий, самостоятельных и контрольных работ.

В-третьих, использование автоматического перевода псевдокода в текст программы так же, как и при создании алгоритма, снижает выработку навыка написания программ и формирует знание синтаксиса языка не как разработчика программ, а как эксперта, оценивающего текст программы, написанный другим. Появляется противоречие между необходимостью получить устойчивые знания синтаксиса языка и навыками написания программ и использованием среды разработки, которая автоматически преобразует алгоритм в программу. Разрешить его можно, если ограничить использование среды на начальных этапах создания программ. Использование среды разработки алгоритма позволяет избавиться от рутинной работы, но не освобождает от необходимости знать и понимать правила записи алгоритмов и программ, так как в любой момент может потребоваться

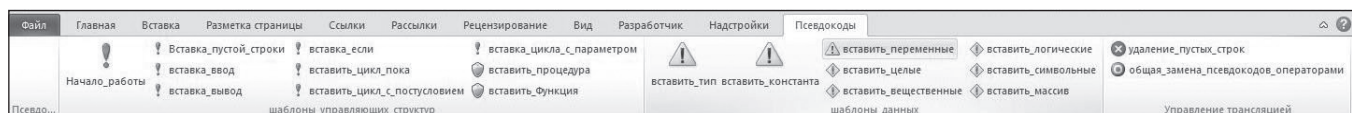


Рис. 4. Настраиваемая вкладка на ленте текстового процессора Word

вмешательство учащегося для исправления ошибок и корректировки полученной записи алгоритма или программы. Это требует от учащихся выработки навыков анализа алгоритмов и программ на предмет обнаружения ошибок и хорошего знания языка программирования. Умение оценивать работу чужой программы проверяется при выполнении некоторых заданий ЕГЭ по информатике.

В-четвертых, высокая скорость работы макрокоманд среды может затруднять процесс объяснения создания алгоритма и особенно программы при применении системы на лекционных и практических занятиях в случае, если преподаватель использует компьютер с выводом результатов на мультимедийный проектор или интерактивную доску и учащиеся не работают на компьютере в момент объяснений преподавателя. Представленная среда разработки алгоритма может использоваться, например, когда требуется интерактивно показать разработку алгоритма и преобразовать его в программу при выполнении просьбы рассмотреть какую-либо задачу, если для нее заранее не подготовлен иллюстративный материал.

Часто при разработке программ стадии разработки алгоритма не уделяется достаточного внимания. Более того, многие разработчики программ, а вслед за ними и учащиеся, сразу приступают к написанию программы, без разработки алгоритма, поэтому у них часто возникают ошибки в алгоритме, особенно в его структуре. Предлагаемая среда переносит акцент на разработку алгоритма за счет автоматизации разработки программы по записанному на псевдокоде алгоритму.

В организациях, профессионально занимающихся разработкой программного обеспечения, алгоритм является обязательным документом, и все изменения в программах обязательно отображаются в описании алгоритма. Если алгоритм первичен, а программа вторична, то изменения в алгоритме будут автоматически отображены в программе и требования документирования программного обеспечения будут соблюдены. Останется только приучить обучающихся сохранять алгоритм вместе с программой.

Среда разработки алгоритмов может оказаться полезной при разработке методических материалов, в которых рассматривается разработка алгоритмов и программ, так как позволяет оперативно автоматически получать поэтапно разработанные тексты алгоритмов и итоговую программу. Таким образом, рассмотренная среда может использоваться как профессиональное средство разработки алгоритмов и программ.

Выводы:

1. Учитывая возможность автоматизации разработки алгоритма и программы, следует использовать запись алгоритма, максимально формализованную с точки зрения управляющих структур. Такой формой является запись на псевдокоде.
2. Для записи управляющих структур используются макрокоманды, которые экономят время и внимание человека, обеспечивая строго формальную запись управляющих конструкций. При этом обеспечивается однозначность записи ключевых слов и автоматически соблюдается правильность структуры алгоритма.
3. Благодаря этому имеется возможность автоматического преобразования псевдокода в программу при минимальной доработке.
4. Разработанный комплекс программного обеспечения среды разработки алгоритмов и программ является открытой системой, допускающей внесение изменений и дополнений. Комплект инсталляции состоит из двух файлов, импортируемых на компьютер с установленным Word версии не ниже 2010.
5. Среда может использоваться на уроках информатики в процессе объяснений и при решении задач на компьютере.
6. Использование среды не означает полного отказа от изучения учащимися языка программирования, а, наоборот, требует от них достаточно глубоких знаний в этой области для проведения анализа автоматически разработанных программ и поиска возможных ошибок.
7. Среда может быть использована преподавателями как инструмент для ускорения создания алгоритмов и программ при подготовке методических материалов к занятиям.

Литературные и интернет-источники

1. ГОСТ 19.701–90 (ИСО 5807–85) Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. http://www.docload.ru/standart/Pages_gost/28346.htm
2. Семакин И. Г., Залогова Л. А., Русаков С. В., Шестакова Л. В. Информатика: учебник для 9 класса. М.: БИНОМ. Лаборатория знаний, 2016.
3. Семакин И. Г., Хеннер Е. К., Шеина Т. Ю. Информатика и ИКТ. Базовый уровень: учебник для 10 класса. М.: БИНОМ. Лаборатория знаний, 2016.
4. Bohm C., Jacopini G. Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules // Communications of the ACM. V. 9, I. 5, May, 1966.