МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Экономика и управление»

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ РЕШЕНИЯ ЗАДАЧ ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ

Методические рекомендации к практическим занятиям для студентов направления подготовки 27.03.05 «Инноватика» дневной формы обучения

Могилев 2022

УДК 004.4
ББК 32.973.202-018.2
П78

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Экономика и управление» «29» марта 2022 г., протокол № 8

Составитель ст. преподаватель Е. Г. Галкина

Рецензент канд. физ.-мат. наук, доц. В. А. Ливинская

Методические рекомендации к практическим занятиям предназначены для студентов направления подготовки 27.03.05 «Инноватика» дневной формы обучения, изучающих дисциплину «Программное обеспечение для решения задач профессиональной деятельности».

Учебно-методическое издание

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
ДЛЯ РЕШЕНИЯ ЗАДАЧ ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ

Ответственный за выпуск	И. В. Ивановская
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 36 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Перед выполнением индивидуального задания студент должен ознакомиться с соответствующей темой конспекта лекций, а в случае необходимости с рекомендуемой по дисциплине литературой.

В результате выполнения индивидуального задания студенту необходимо разработать алгоритм решения задания. Схема алгоритма должна быть изображена в соответствии с ГОСТ 19.701–90.

К защите индивидуального задания допускаются только студенты, выполнившие работу и оформившие отчет. Защита проходит в форме устного и письменного собеседования, когда студент отвечает на вопросы преподавателя, которые приведены в данных методических рекомендациях в списке контрольных вопросов к каждой теме, а также в случае необходимости дополняет свои ответы письменно примерами. При успешной защите преподаватель ставит на отчете свою подпись и дату защиты.

Список литературы

1 **Головин, И. Г.** Языки и методы программирования: учебник / И. Г. Головин, И. А. Волкова. – 2-е изд., стер. – Москва : Академия, 2016. – 304 с.

2 Информационные системы в экономике : учебное пособие / Под общ. ред. М. Н. Садовской. – Минск : БГЭУ, 2018. – 316 с.

3 **Стариченко, Б. Е.** Теоретические основы информатики : учебник / Б. Е. Стариченко. – 3-е изд., перераб. и доп. – Москва : Горячая линия–Телеком, 2017. – 400 с.

4 **Скитер, Н. Н.** Информационные технологии : учебное пособие / Н. Н. Скитер, А. В. Костикова. – Волгоград : ВолгГТУ, 2019. – 96 с.

5 **Щеглов, А. Ю.** Защита информации. Основы теории: учебник / А. Ю. Щеглов, К. А. Щеглов. – Москва : Юрайт, 2019. – 309 с.

1 *Подход*, *ориентированный на пользователя*. Основным содержанием этого подхода является ориентация на пользователя, т. е. в первую очередь необходимо узнать, что хочет пользователь получить от проектируемого интерфейса. Далее в процессе проектирования полученные требования реализуются в продукте. При сборе информации используются методы наблюдения за работой пользователя, проводятся интервью.

2 *Системный подход.* Пользователь рассматривается как маленькая интеллектуальная часть системы «человек – программный продукт».

3 *Деятельностный подход.* Изучается деятельность пользователя в целом, и постепенно оптимизируются её отдельные моменты.

4 *Итеративный подход* – метод последовательных приближений. Суть итеративного подхода заключается в создании изначально самого простейшего прототипа с целью показать заказчику и затем постепенно дорабатывать прототип, основываясь на реакции заказчика после каждого шага доработки.

5 *Экспертный подход.* Заключается в следующем: эксперт собирает важную, по его мнению, информацию, ведёт переговоры с заказчиком, задаёт нужные вопросы. На основе полученной информации создаётся интерфейс.

6 *Целеориентированный подход* проектирования. Разработка интерфейса ориентируется на цель, которая будет достигаться данным программным продуктом.

7 *Средоориентированный подход.* Разрабатывается среда интерфейса как место деятельности оператора.

При разработке интерфейса целесообразно гибко пользоваться указанными подходами, учитывая при выборе методов назначение разрабатываемого продукта, целевую аудиторию, время и бюджет разработки.

Контрольные вопросы

1 Пользовательский интерфейс.
2 Этапы разработки пользовательского интерфейса.
3 Подходы к проектированию пользовательского интерфейса.

7 Порядок выполнения и защиты индивидуального задания

Вначале следует получить вариант индивидуального задания у преподавателя, проводящего практическое задание.

Отчет по индивидуальному заданию выполняется на листах формата А4. В состав отчета входят:

1) титульный лист;
2) цель работы;
3) текст варианта индивидуального задания;
4) краткое описание хода выполнения индивидуального задания.

Отчет оформляется индивидуально каждым студентом.

Содержание

1 Основы теории алгоритмов

Цель работы: программирование базовых конструкций алгоритмов; получение практических навыков по работе с линейными алгоритмами.

Задание

По данным индивидуального задания построить схему линейного алгоритма.

Методические указания

Процесс составления программы будет значительно упрощен при наличии алгоритма, детализированного вплоть до операторов языка. В этом случае программирование сведется к записи операторов с учетом правил выбранного языка программирования.

Происхождение понятия алгоритма связано с именем великого среднеазиатского ученого Аль Хорезми, жившего в IX веке н. э. Им были сформулированы впервые правила выполнения четырех арифметических действий.

Алгоритм – это точная инструкция, а инструкции встречаются во всех областях человеческой деятельности. Однако не всякую инструкцию можно назвать алгоритмом. Алгоритм применяется к искомому набору исходных величин, называемых аргументами.

Цель исполнения алгоритма – получение определенного результата, если в результате исполнения алгоритма не достигнута определенная цель, значит алгоритм либо неверен, либо не завершен.

Алгоритм – это метод (способ) решения задачи, записанный по определенным правилам, обеспечивающим однозначность его понимания и механического исполнения при всех значениях исходных данных за конечное число шагов.

Или более коротко: **алгоритм –** это строго определенная последовательность действий, необходимых для решения данной задачи.

Примером алгоритма может служить кулинарный рецепт приготовления блюда. Рассмотрим простейший алгоритм – алгоритм заварки чая:

1) подготовить исходные величины – чай, воду, чайник, стакан, ложку;
2) налить в чайник воду;
3) довести воду до кипения и снять с огня;
4) всыпать в чайник чай;
5) довести воду до кипения (но не кипятить), снять с огня;
6) чай готов. Процесс прекратить.

Основными свойствами алгоритмов являются.

1 **Универсальность (массовость)** – применимость алгоритма к различным наборам исходных данных.

2 **Дискретность** – процесс решения задачи по алгоритму разбит на отдельные действия.

Контрольные вопросы

1 Понятия и принципы тестирования.
2 Структурное тестирование (тестирование «белого ящика»).
3 Функциональное тестирование (тестирование «черного ящика»).

6 Проектирование пользовательского интерфейса

Цель работы: научиться разрабатывать простейший пользовательский интерфейс.

Задание

В соответствии с вариантом индивидуального задания, определяющим предметную область, разработать пользовательский интерфейс.

Методические указания

Пользовательский интерфейс – это интерфейсное приложение, с которым пользователь взаимодействует для использования программного обеспечения. Пользователь может манипулировать программным и аппаратным обеспечением и управлять им с помощью пользовательского интерфейса. Основа взаимодействия – диалоги.

Диалог – регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи: обмен информацией и координация действий. Каждый диалог состоит из отдельных процессов ввода-вывода, которые физически обеспечивают связь пользователя и компьютера.

Пользовательский интерфейс является частью программного обеспечения и спроектирован таким образом, чтобы обеспечить понимание пользователем программного обеспечения. Пользовательский интерфейс обеспечивает фундаментальную платформу для взаимодействия человека с компьютером.

Разработка пользовательского интерфейса включает те же основные этапы, что и разработка программного обеспечения:

– постановка задачи – определение типа интерфейса и общих требований к нему;

– анализ требований и определение спецификаций – определение сценариев использования и пользовательской модели интерфейса;

– проектирование – проектирование диалогов и их реализация в виде процессов ввода-вывода;

– реализация – программирование и тестирование интерфейсных процессов.

Для получения эффективного результата разработки пользовательского интерфейса используют различные подходы к проектированию.

– свой набор исходных данных и условий для запуска программы;
– набор ожидаемых результатов работы программы.

Другое название теста – тестовый вариант. Полную проверку программы гарантирует *исчерпывающее тестирование.* Оно требует проверить все наборы исходных данных, все варианты их обработки и включает большое количество тестовых вариантов. Исчерпывающее тестирование во многих случаях невозможно – срабатывают ресурсные ограничения (прежде всего, ограничения по времени).

Целью проектирования тестовых вариантов является систематическое обнаружение различных классов ошибок при минимальных затратах времени и стоимости.

Тестирование обеспечивает:
– обнаружение ошибок;
– демонстрацию соответствия функций программы ее назначению;
– демонстрацию реализации требований к характеристикам программы;
– отображение надежности как индикатора качества программы.

Тестирование не может показать отсутствия дефектов (оно может выявить только присутствие дефектов).

Существуют два принципа тестирования программы:
1) функциональное тестирование (тестирование «черного ящика»);
2) структурное тестирование (тестирование «белого ящика»).

Тестирование «черного ящика».

Известны: функции программы.

Исследуется: работа каждой функции на всей области определения.

Эти тесты демонстрируют;
– как выполняются функции программ;
– как принимаются исходные данные;
– как вырабатываются результаты;
– как сохраняется целостность внешней информации.

При тестировании «черного ящика» рассматриваются системные характеристики программ, игнорируется их внутренняя логическая структура. Исчерпывающее тестирование, как правило, невозможно. Например, если в программе 10 входных величин и каждая принимает по 10 значений, то потребуется 10^{10} тестовых вариантов. Отметим также, что тестирование «черного ящика» не реагирует на многие особенности программных ошибок.

Тестирование «белого ящика».

Известна: внутренняя структура программы.

Исследуются: внутренние элементы программы и связи между ними.

Объектом тестирования здесь является не внешнее, а внутреннее поведение программы. Проверяется корректность построения всех элементов программы и правильность их взаимодействия друг с другом. Обычно анализируются управляющие связи элементов, реже – информационные. Тестирование по принципу «белого ящика» характеризуется степенью, в какой тесты выполняют или покрывают логику (исходный текст) программы.

3 **Однозначность** – правила и порядок выполнения действий алгоритма имеют единственное толкование.

4 **Конечность** – каждое из действий и весь алгоритм в целом обязательно завершаются.

5 **Результативность** – по завершении выполнения алгоритма обязательно получается конечный результат.

6 **Выполнимость** – результат алгоритма достигается за конечное число шагов.

Алгоритм считается правильным, если его выполнение дает правильный результат.

Выделяют **три** крупных **класса алгоритмов**:

1) **вычислительные** алгоритмы – работающие с числами и матрицами, хотя сам процесс вычисления может быть долгим и сложным;

2) **информационные** алгоритмы – работающие с большими объемами информации (алгоритмы баз данных);

3) **управляющие** алгоритмы – генерирующие различные управляющие воздействия на основе данных, полученных от внешних процессов, которыми алгоритмы управляют.

Для записи алгоритмов используют самые разнообразные средства. Выбор средства определяется типом исполняемого алгоритма. Выделяют следующие основные способы записи алгоритмов:

– **вербальный**, когда алгоритм описывается на человеческом языке представления, основанном на использовании так называемого псевдокода;

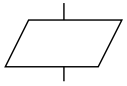
– **символьный**, когда алгоритм описывается с помощью набора символов;

– **графический**, когда алгоритм описывается с помощью набора графических изображений.

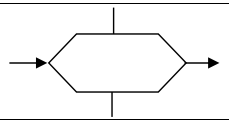
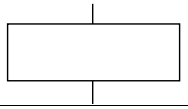
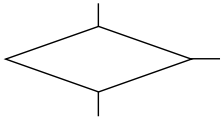
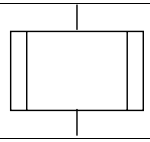
Общепринятыми способами записи являются графическая запись с помощью схем и символьная запись с помощью какого-либо алгоритмического языка.

Описание алгоритма с помощью блок-схем осуществляется рисованием последовательности геометрических фигур, каждая из которых подразумевает выполнение определенного действия алгоритма. Ниже приводятся изображения символов схемы программы по ГОСТ 19.701–90. Внешний вид основных блоков, применяемых при написании блок-схем, приведен в таблице 1.1.

Таблица 1.1 – Графические изображения элементов схем алгоритмов

Символ	Значение	Применение
	Завершение	Начало, конец обработки данных или выполнения программы
	Данные	Обозначает ввод, вывод данных

Окончание таблицы 1.1

Символ	Значение	Применение
	Подготовка	Описывается подготовка данных для выполнения повторяющихся действий
	Процесс	Обработка данных любого вида (выполнение операции или группы операций)
	Решение	Выбор направления выполнения программы в зависимости от некоторых переменных условий
	Типовой процесс	Одна или несколько операций, которые определены в другой программе, модуле
	Соединитель (узел)	Используется при разрыве линий схемы алгоритма
	Комментарий	Используется для пояснений

Основные алгоритмические конструкции:

- линейный алгоритм;
- разветвляющийся алгоритм;
- циклический алгоритм;
- вспомогательный алгоритм.

Линейный алгоритм – алгоритм или фрагмент алгоритма, в котором порядок исполнения инструкций соответствует порядку их записи.

Инструкции линейного алгоритма выполняются последовательно одна за другой в порядке их записи. Схему линейного алгоритма обычно представляют в виде символов, соединенных последовательно. В каждый символ входит не более одной линии потока информации. Из каждого символа выходит не более одной линии потока информации. Обычно схему размещают таким образом, что символы размещаются один под другим, и нет необходимости обозначать линии потока информации стрелками.

Допускается изображать линейный алгоритм горизонтально или в виде изгибающейся под прямыми углами цепочки блок-символов. В этом случае

схеме. Линии на схеме иерархии показывают только подчиненность модулей. Каждый модуль активизируется вышестоящим и, закончив работу, возвращает управление вызвавшему модулю. Таким образом, вызываемая подпрограмма подчинена вышестоящему модулю и подчиняет себе нижестоящие модули.

После того как вся программа разделена на отдельные составные части, заключенные в отдельные модули, приступают к их проектированию, т. е. к разработке алгоритмов этих модулей и их кодированию на алгоритмическом языке. Алгоритмы самостоятельных модулей разрабатываются независимо один от другого.

Порядок разработки алгоритмов модулей и их кодирования на алгоритмическом языке может быть любым. Например, сначала могут быть запрограммированы все модули одного уровня иерархии, а затем следующего уровня, либо программируют все модули одной ветви иерархии.

Контрольные вопросы

1 Для чего используют модульный принцип построения программ?
2 До каких пор происходит разделение программы на модули?
3 Каким образом модульный принцип построения программ обеспечивает возможность организации совместной работы?
4 Что не показывает структурная схема модульной структуры программы?

5 Отладка и тестирование программ

Цель работы: изучить общие понятия и принципы тестирования программ.

Задание

В соответствии с вариантом индивидуального задания составить тест создаваемой программы.

Методические указания

После набора исходного текста программы и ввода его в память ЭВМ начинается этап ее тестирования и отладки. На этом этапе выявляются ошибки, допущенные при составлении алгоритма и написании программы. В результате должна быть получена программа, обеспечивающая решение поставленной задачи. Отладка программы является одним из наиболее трудоемких и длительных процессов в программировании.

После окончания отладки программа, готовая к использованию, записывается в память ЭВМ на языке машинных команд.

Тестирование – процесс выполнения программы с целью обнаружения ошибок. Шаги процесса задаются тестами.

Каждый тест определяет:

сложность головного модуля. Его становится трудно отлаживать и изменять. Для устранения этого недостатка в структуре модули второго уровня разделяются на части, которые реализуются своими подчиненными модулями. Эти модули также могут быть разделены на подчиненные модули.

Рассмотрим порядок разработки структуры программы на примере простейшей вычислительной задачи, которая выполняет как минимум три функции: ввод исходных данных, осуществление вычислений и представление результатов. Каждый из этих подчиненных модулей может быть простым и достаточно сложным в зависимости от решаемой задачи и требований, предъявляемых к программе. Предположим, что к программе предъявляются требования обеспечения ввода исходных данных как с клавиатуры, так и из внешней памяти, а вывод результатов должен осуществляться как на экран дисплея, так и в файл данных, хранящийся во внешней памяти ЭВМ. Сама задача может быть достаточно сложной и решаться в два этапа: сначала вычисляются некоторые промежуточные результаты, а затем – окончательные. В этом случае функции программы, выделенные в отдельные модули второго уровня иерархии, легко разделяются на подфункции, которые будут представлены модулями следующего уровня иерархии. Полученная схема иерархии модулей показана на рисунке 4.1. При необходимости разделение модулей может быть продолжено. Разделение модулей на составляющие осуществляется до тех пор, пока модуль будет выполнять только одну простейшую функцию.

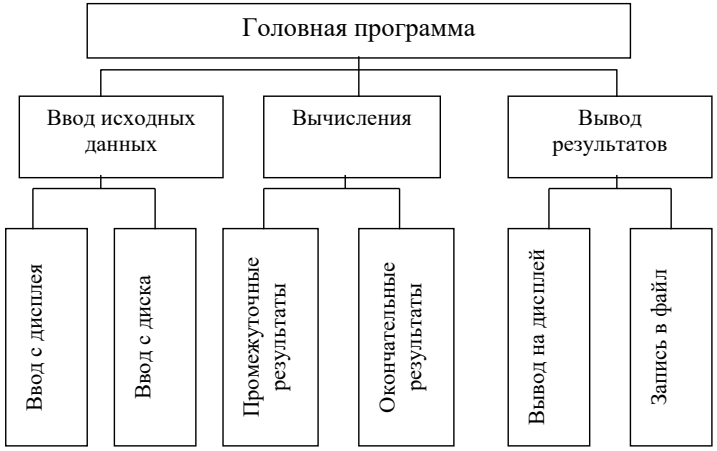


Рисунок 4.1 – Модульная структура программы

Структурная схема не показывает потока данных, порядка выполнения или моментов и частоты активизации каждого модуля. Расположение модулей на заданном уровне не определяет порядок их исполнения. Управление частотой и порядком выполнения скрыто внутри прямоугольников и не показывается на

линии потока информации, идущие снизу вверх и справа налево, следует изображать в виде стрелок.

Линейный фрагмент имеет любой алгоритм. В алгоритме «Кипячение воды» можно выделить два линейных фрагмента (рисунок 1.1).

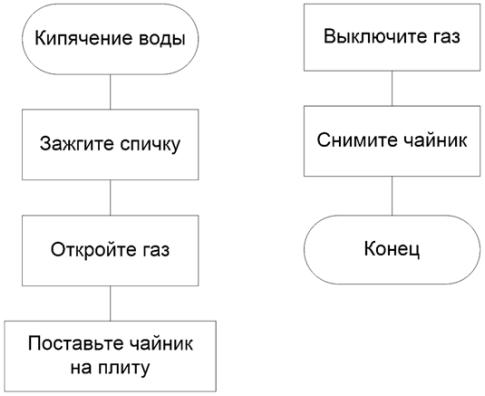


Рисунок 1.1 – Линейные фрагменты алгоритма «Кипячение воды»

После завершения этапа разработки алгоритма выбирается один из алгоритмических языков, позволяющий реализовать решение задачи, и осуществляется кодирование алгоритма на этом языке.

Перед написанием программы следует детализировать организацию данных, предусмотренную разработанным алгоритмом решения задачи. Следует продумать какие константы, переменные и более сложные структуры данных будут необходимы для реализации алгоритма. Для этого рекомендуется составить таблицу соответствия обозначений (имен) в алгоритме и программе.

Контрольные вопросы

1 Кто и когда впервые ввел понятие алгоритма?
2 Каковы способы записи алгоритмов?
3 В чем заключаются основные свойства алгоритма?
4 Перечислите основные алгоритмические структуры и опишите их.
5 Каковы основные принципы разработки алгоритмов?
6 Чем объясняется разнообразие форм записи алгоритмов?
7 Что такое исходные данные?
8 Что такое результат выполнения алгоритма?
9 Что представляет собой графическая форма записи алгоритма?

2 Разветвляющиеся структуры схемы алгоритма

Цель работы: программирование базовых конструкций алгоритмов; получение практических навыков по работе с разветвляющимися алгоритмами.

Задание

По данным индивидуального задания построить схему разветвляющегося алгоритма.

Методические указания

Разветвляющийся алгоритм – алгоритм или фрагмент алгоритма, в котором порядок исполнения инструкций не определен изначально, имеет не менее двух вероятных направлений, каждое из которых соответствует одному из возможных исходов проверки логического условия.

Своим ветвлением разветвляющийся алгоритм обязан логическому условию, проверка которого на истинность обязательно приводит к возможности реализации одного из двух взаимоисключающих исходов. Линии потока информации, выходящие из блока «Решение», могут быть отмечены парами противоположных по смыслу знаками (таблица 2.1).

Таблица 2.1 – Знаки и символы, используемые в разветвляющихся алгоритмах, для обозначения исходов проверки логического условия

Вариант обозначения	Значение
«Да», «Нет» «+», «–» «1», «0»	Условие истинно: «Да», «+», «1» Условие ложно: «Нет», «–», «0»

В то же время, комбинируя проверку нескольких логических условий, можно получить многоуровневое ветвление с тремя и более вероятными направлениями.

Обычно ветвление является фрагментом более сложного алгоритма. На рисунке 2.1 представлен фрагмент алгоритма «Кипячение воды».

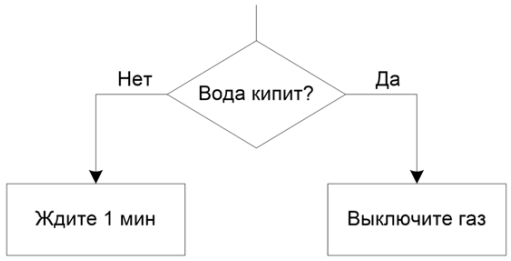


Рисунок 2.1 – Фрагмент разветвляющегося алгоритма

Программа, предназначенная для решения простейших задач, состоит, как правило, из нескольких десятков операторов алгоритмического языка. В этом случае она представляет собой единое целое.

При проектировании более сложных программ, у которых имеется несколько взаимосвязанных самостоятельных функций, более целесообразно использовать модульный принцип построения программ, обладающий существенными преимуществами. Модульный принцип заключается в том, что программа строится из множества отдельных программных модулей, взаимосвязанных между собой определенными связями. Такой подход в качестве следующего шага проектирования программы требует разработки ее структуры с использованием вспомогательных алгоритмов.

Вспомогательный алгоритм – функционально независимый алгоритм, к которому алгоритм более высокого уровня может обращаться произвольное количество раз.

В программировании вспомогательный алгоритм именуется «подпрограмма». В результате более чем полувековой мировой практики программирования созданы разнообразные вспомогательные алгоритмы, которые позволяют решать большинство стандартных задач программирования. Практически каждый язык программирования имеет обширную библиотеку подпрограмм, с помощью которой можно решать сложные алгоритмические задачи намного быстрее и проще.

Таким образом, процесс разработки структуры программы состоит в разбиении всей решаемой задачи на отдельные части, которые реализуются соответствующими модулями программы, и в установлении взаимосвязей между этими модулями.

Преимущества использования модульного принципа программирования состоит в следующем. Упрощается отладка программ, т. к. ограниченный доступ к модулю и однозначность его внешнего поведения исключает влияние ошибок в других, связанных с ним модулях на его функционирование. Это дает возможность поэтапной разработки программы, постепенно присоединяя написанные модули к ранее отлаженным. После каждого такого присоединения неверная работа программы сигнализирует о присутствии ошибок в новом модуле, а не в уже отлаженных.

Обеспечивается возможность организации совместной работы больших коллективов разработчиков, т. к. каждый программист имеет дело с независимой от других частью программы (модулем или группой модулей).

Повышается надежность программы, т. к. относительно малый размер модулей, и, как следствие, небольшая сложность их позволяет провести более полную проверку программы.

При проектировании структуры программы следует стремиться к получению модульно-иерархической структуры.

Чтобы создать схему иерархии, следует начинать от вершины и идти вниз, т. е. от цели программы к ее отдельным функциям. Нужно, чтобы один модуль в программе управлял выполнением остальных модулей, расположенных на более низшем уровне иерархии. С ростом сложности программы растет и

Контрольные вопросы

1 Для чего используют структуру «цикл»?
2 Какие виды циклов вы знаете?
3 Что такое тело цикла?

4 Классические методы проектирования модульных программ

Цель работы: получение практических навыков по проектированию модульных программ.

Задание

По данным индивидуального задания спроектировать схему алгоритма, используя модульный принцип построения программ.

Методические указания

Проектирование программы начинается с постановки задачи, которая выражается в разработке технического задания. Задание определяет общий подход к решению задачи. В нем формулируются условия задачи, исходные данные для решения, конечные цели решения и форма выдачи результатов. В задании могут оговариваться область применения программы и накладываемые при этом ограничения.

Вторым этапом является выбор метода решения поставленной задачи. Вначале формулируется общее математическое описание задачи, которое заключается в составлении математической модели, приемлемой для ее решения. Под математической моделью понимается совокупность математических зависимостей, отображающая реальные исследуемые процессы. Математическая модель всегда основывается на некотором упрощении и является идеализированным отображением реального объекта.

Для ряда задач разработка математической модели осуществляется достаточно просто и не вызывает трудностей. Для других задач этот этап является весьма сложным, требует специальных знаний в предметной области, в области математического моделирования технических объектов и значительных затрат времени.

Выбранный метод решения должен обеспечивать представление вычислительного процесса в виде последовательности элементарных арифметических и логических операций. Если ни один из известных методов не подходит для решения поставленной задачи, возникает необходимость разработки нового метода решения.

Пример разветвляющего алгоритма для решения квадратного уравнения $ax^2 + bx + c = 0$ представлен на рисунке 2.2.

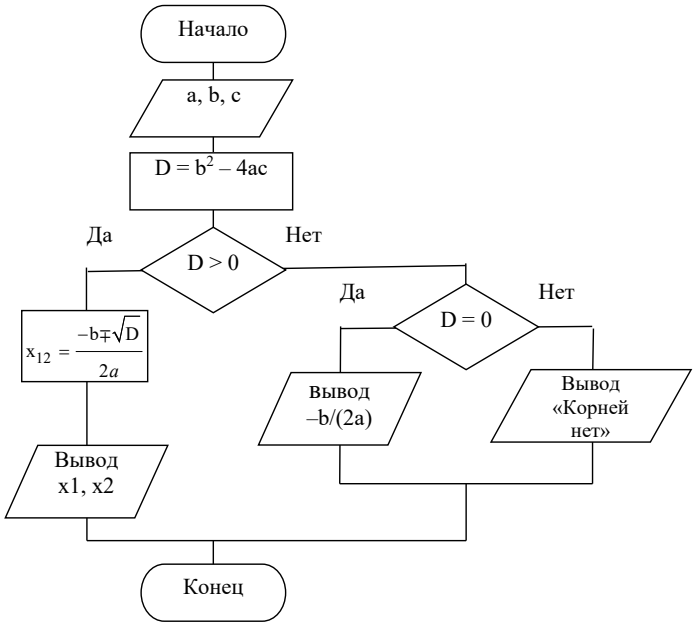


Рисунок 2.2 – Пример составления разветвляющего алгоритма

Перед составлением алгоритма программы надо четко определить, *что* в нее *требуется ввести* и *что* должны *получить в результате*.

В данном случае:

– в качестве исходных данных выступают три вещественных числа a, b и c (коэффициенты квадратного уравнения);

– в качестве результата – корни квадратного уравнения, вещественные числа x1 и x2.

Контрольные вопросы

1 Для чего используют разветвляющиеся алгоритмы?
2 Назовите варианты обозначения исходов проверки логического условия.
3 Приведите примеры использования разветвляющегося алгоритма для организации нескольких вычислительных ветвей.
4 Нарисуйте алгоритмическую схему разветвляющегося алгоритма для n вероятностных направлений.

3 Циклические структуры

Цель работы: программирование базовых конструкций алгоритмов; получение практических навыков по работе с циклическими алгоритмами.

Задание

По данным индивидуального задания построить схему алгоритма с циклом.

Методические указания

Циклический алгоритм – фрагмент алгоритма с ветвлением, в котором инструкция или группа инструкций исполняются более одного раза, т. е. повторяются.

Тело цикла – группа повторяющихся инструкций.

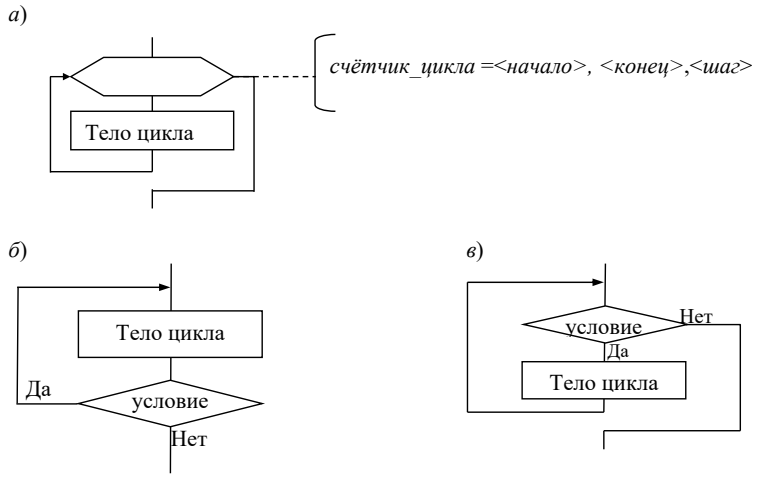
Основой циклического алгоритма является повторение инструкции или группы инструкций – тела цикла. Начало и завершение повторения тела цикла определяется итогом проверки логического условия.

Как правило, циклический алгоритм является фрагментом более сложного алгоритма.

Различают два основных типа циклов: циклы со счетчиком (циклы с заданным количеством повторений) и циклы с условием (условные циклы), которые позволяют повторить группу операторов или один оператор заданное количество раз. Условные циклы имеют две базовые структуры – **цикл с постусловием** и **цикл с предусловием**.

Циклический алгоритм с постусловием (рисунок 3.1, *б*) характеризуется тем, что проверка условия расположена после тела цикла (от лат. *post* – после). Циклический алгоритм с предусловием характеризуется тем, что проверка условия расположена перед телом цикла (рисунок 3.1, *в*).

Пример циклического алгоритма для вычисления n-го члена последовательности, заданной формулой $a_n = a_{n-1} + a_{n-2}$, если $a_1 = 1$, $a_2 = 1$ представлен на рисунке 3.2.



а – циклический алгоритм с параметром; *б* – циклический алгоритм с постусловием; *в* – циклический алгоритм с предусловием

Рисунок 3.1 – Базовые структуры циклических алгоритмов

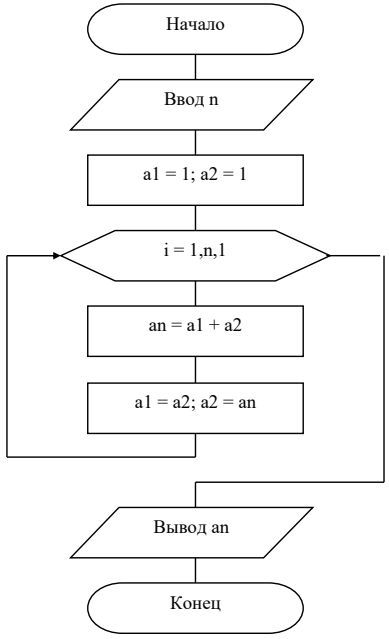


Рисунок 3.2 – Пример составления циклического алгоритма