Министерство образования и науки Российской Федерации

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный университет
промышленных технологий и дизайна»

ИНФОРМАТИКА

Методические указания и задания к контрольной работе № 2
для заочной формы обучения
по всем направлениям подготовки

Составители:
М. А. Ермина
Д. А. Ермин
Ф. Л. Хватова

Санкт-Петербург
2015

Введение

Настоящие методические указания предназначены для дальнейшего освоения современных компьютерных технологий в дисциплине «Информатика».

Во втором семестре рабочей программой предусматривается изучение современных технологий обработки информации и выполнения необходимых заданий по этой теме в контрольной работе.

Вопросы к экзамену по курсу «Информатика»

(Ответы на все вопросы, должны сопровождаться примерами).

1. Основные понятия программирования. Этапы решения задач на ПК.
2. Алгоритмизация вычислительных процессов. Виды описания алгоритмов, их типы и свойства.
3. VBA. Базовые элементы языка.
4. VBA. Интегрированная среда (Редактор VBA).
5. VBA. Структура программы.
6. VBA. Типы данных.
7. VBA. Типы данных. Совместимость типов и значений, функции преобразования типов.
8. VBA. Тип данных String. Функции для обработки строк.
9. VBA. Организация ввода-вывода в программе (Инструкции).
10. VBA. Организация линейной структуры в программе (Инструкции).
11. VBA. Построение условных конструкций в программе (Инструкции).
12. VBA. Организация циклов в программе (Инструкции).
13. VBA. Структурированные типы данных. Массивы. Типичные операции над массивами.
14. VBA. Процедуры и функции. Параметры. Глобальные и локальные описания подпрограмм.
15. VBA. Процедуры, определяемые пользователем. Обращение к процедурам.
16. VBA. Функции, определяемые пользователем. Обращение к функциям.

Таблица соответствия последней цифры зачетной книжки и варианта задания:

	Последняя цифра зачетной книжки									
	1	2	3	4	5	6	7	8	9	0
Номер варианта	1	2	3	4	5	6	7	8	9	10

1. Контрольная работа №2

Тема - «Автоматизация обработки информации в приложениях Windows»

Цель работы – освоение новых информационных технологий для решения практических задач, актуальных в современной промышленности.

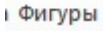
Контрольная работа предусматривает задания для 10 – и вариантов.

Для каждого варианта заданий создать именную папку для размещения четырех файлов (для четырех заданий варианта).

Для каждого задания:

1. разработать блок-схему алгоритма и создать ее средствами

   Microsoft Excel (кнопка 

   для выполнения команды вставки фигуры на вкладке «Вставка» в области «Иллюстрации»);

2. составить программный код на языке программирования Visual Basic for Applications, с помощью Редактора Visual Basic (Visual Basic Editor) в среде Microsoft Excel записать программный код в модуль;

3. отладить и выполнить программу на компьютере средствами Редактора Visual Basic;

4. оформить выполненное на компьютере задание на листе рабочей книги Microsoft Excel (представить текст задания, таблицу обозначений, блок-схему алгоритма, текст программного кода с результатами работы программы в виде рисунков).

Вариант 1

1. Даны числа **X** и **Y**. Вычислить **Z=f(T,P) (Линейный алгоритм).**

$$\mathrm{T}=\frac{\ln(x+y)}{3\cdot x-\cos\frac{x}{y}};\quad \mathrm{P}=\frac{\sqrt{x+y^3}}{3\cdot x\cdot y};\qquad \mathrm{Z}=\frac{P}{T}$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (операторов if).

$$F=\begin{cases} z+t|z|;\ z<-1{,}2 \\ \sin(tz);\ -1{,}2\le z\le 2{,}4 \\ e^{-2\sin(t+z)};\ z>2{,}4 \end{cases}$$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла **(do ..)**.

$$\mathrm{F(x)}=\frac{(2\cdot \mathrm{a}+\mathrm{x})}{(3\cdot \mathrm{a}-\mathrm{x})},\quad -1\le x\le 1,\quad \mathrm{a}=10,\quad \mathrm{h}=0.2$$

(Циклический алгоритм)

4. Заполнить массивы **X(10)** и **Y(10**) произвольными величинами. Если число неположительных элементов массива **X** больше числа положительных элементов массива **Y**, то все отрицательные элементы массива **X** заменить единицами и вывести массив **X**, если нет, то перед всеми положительными элементами массива **Y** поставить знак минус и вывести полученный массив **Y. (Обработка массива).**

Вариант 2

1. Даны величины **X** и **Y**. Вычислить **Z=f(T,P) (Линейный алгоритм).**

$$\mathrm{T}=\frac{x+y}{x^2-y^2};\quad \mathrm{P}=\frac{5\cdot Sin(x-y)}{x+y};\quad \mathrm{Z}=\mathrm{P}\times\mathrm{T};\qquad \text{для } \mathrm{x}\neq \mathrm{y}$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if**).

$$F=\begin{cases} 1+2zt;\ z\le -1 \\ -1+tz;\ -1<z<2 \\ 3t\cos z+2z;\ z\ge 2 \end{cases}$$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла **(do ..).**

$$\mathrm{F(x)}=\sqrt{(1+x^5)};\qquad \text{для } \mathrm{x}=0.5,0.6,\ldots,2$$

(Циклический алгоритм).

4. Даны массивы **A(5) и B(5),** состоящие из положительных элементов. Найти сумму среднего геометрического значения элементов массива **A** и среднего арифметического значения элементов массива **B**. **(Обработка массива)**.

Вариант 3

1. Даны числа **X** и **Y**. Вычислить **Z=f(T,P). (Линейный алгоритм).**

$$T = \frac{\ln x + 10}{8 \cdot x + \sin(x * y)} \qquad P = \frac{\sqrt{x + 3 \cdot y}}{7xy}; \qquad Z = P - T;$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if**). $F = \begin{cases} e^t + e^z; z < 1 \\ \sin(t) + z\cos(t); 1 \le z \le 2 \\ \sqrt{t^2 + z^3}; z > 2 \end{cases}$

(Разветвляющийся алгоритм).

3. Получить значения функции Y=F(x) с использованием цикла **(do ..).**

$$F(x) = 7 \cdot x + 10; \qquad \text{для } x = 1.5, 1.6, \ldots, 2.5$$

(Циклический алгоритм).

4. Дан массив **A(10)**. Найти сумму неположительных элементов массива **A**, а также их количество **(Обработка массива).**

Вариант 4

1. Даны числа **X** и **Y**. Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$T = \frac{e^{x+y}}{2 \cdot x + y}; \quad P = \frac{x + 3 \cdot y}{e^{x-y}}; \quad Z = T - P;$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if**). $F = \begin{cases} z^3 t - 1; z < -1{,}1 \\ z + tz + (z - t)^2; -1{,}1 \le z \le 2{,}3 \\ \cos(z + t) + 1{,}5; z > 2{,}3 \end{cases}$

(**Разветвляющийся алгоритм).**

3. Получить значения функции Y=F(x) с использованием цикла **(do ..).**

$$F(x) = 9 \cdot x + 1/x; \quad \text{для } x = 1.5, 1.6, \ldots, 2.5$$

(Циклический алгоритм).

4. Дан массив **B(10)**. Найти сумму отрицательных элементов массива **B**, количество отрицательных элементов массива **B** и их частное. **(Обработка массива).**

Вариант 5

1. Даны числа **X** и **Y**. Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$T = \mathrm{Sin}^3(x+y); \quad P = 6 \cdot x \cdot \sqrt{x+y}; \quad Z = P * T$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if**).

$$F = \begin{cases} |z|t; z < 0{,}6 \\ 2z + tz^2; -0{,}6 \le z \le 2{,}8 \\ \sin(t+z); z > 2{,}8 \end{cases}$$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла **(do ..).**

$$F(x) = 4 \cdot x + 1/x^2; \quad \text{для } x = 1.5, 1.6, \ldots, 2.5$$

(Циклический алгоритм).

4. Дан массив **B(10)**. Найти индексы наибольшего и наименьшего элементов массива **B** и сумму их значений **(Обработка массива)**.

Вариант 6

1. Даны числа **X** и **Y**. Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$T = \sqrt[3]{(x^4 + y^2)}; \quad P = x \times y + \sqrt{x+y}; \quad Z = \frac{p}{T}$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if).**

$$F = \begin{cases} t + tz^3; z < -1{,}2 \\ \ln|z| + t; -1{,}2 \le z \le -0{,}1 \\ \lg(1 + \sqrt{|z| + t}); z > -0{,}1 \end{cases}$$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла **(do ..).**

$$F(x) = \sin(x^3 + 1/x); \quad \text{для } x = 1.2, 1.3, \ldots, 2.5$$

(Циклический алгоритм).

4. Дан массив **C(10)**. Найти сумму и среднее арифметическое элементов массива **C**. (**Обработка массива).**

Вариант 7

1. Даны числа **X** и **Y**. Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$T = x^5 \cdot \ln y; \quad P = x^3 \cdot \sqrt{x + y^2}; \quad Z = \frac{p}{T}$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно

меньше сравнений (**операторов if).** $F = \begin{cases} \cos(t)z^2 - t; z < 0,6 \\ t + z; 0,6 \le z \le 2,4 \\ arctg(tz); z > 2,4 \end{cases}$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла(**do ..).**

$$F(x) = \cos(x + 1/x - 1); \quad \text{для } x = 2.5,2.6,\ldots,3.5$$

(Циклический алгоритм).

4. Дан массив **D(10)**. Найти сумму элементов массива **D** и произведение элементов массива **D (Обработка массива).**

Вариант 8

1. Даны числа **X** и **Y**.Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$T = \frac{x}{7} + \sqrt[3]{x^2 + y^2}; \quad P = 5 \cdot e^x + 3 \cdot e^y; \quad Z = \frac{P}{T}$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if).** $F = \begin{cases} 2t + z; z < 5 \\ t \lg z; 5 \le z \le 10 \\ 4t + \sqrt{z}; z > 10 \end{cases}$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла (**do ..**).

$$F(x) = \cos(x^2 + 1/x - 1); \quad \text{для } x = 1.5,1.6,\ldots,2.5$$

(Циклический алгоритм).

4. Дан массив **F(10)**. Найти сумму и произведение отрицательных элементов массива **F (Обработка массива).**

Вариант 9

1. Даны числа **X** и **Y**. Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$T = \frac{x + y}{x^3 - y^2}; \quad P = \frac{Sin(x - y)}{x - y}; \quad Z = \cos(\frac{T}{P}); \quad \text{для } x \ne y$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно

меньше сравнений (**операторов if).** $F = \begin{cases} tz + \sqrt{|z|}; z < 2,7 \\ t + z; 2,7 \le z \le 4,1 \\ tz^2 + \ln\sqrt{z}; z > 4,1 \end{cases}$

(Разветвляющийся алгоритм).

3. Получить значения функции **Y=F(x)** с использованием цикла**(do ..).**

$$\mathrm{F(x)} = arctg(x + 1/x - 1); \quad \text{для } \mathrm{x} = 2.2, 2.3, \ldots, 3.5$$

(Циклический алгоритм).

4. Дан массив **R(10)**. Найти сумму индексов и сумму значений максимального и минимального элементов массива **R**. **(Обработка массива).**

Вариант 10

1. Даны числа **X**, **Y** и **A**. Вычислить **Z= f(T,P) (Линейный алгоритм).**

$$\mathrm{T} = \frac{x - y}{2x - a}; \qquad \mathrm{P} = \mathrm{x} + 2 \cdot \mathrm{y} + 5; \quad \mathrm{Z} = \cos(T \cdot P);$$

2. Даны числа t и z. Вычислить значение функции F для трех значений z из заданных промежутков. Составить алгоритм, используя как можно меньше сравнений (**операторов if).** $F = \begin{cases} \sqrt{|zt + 1|}; z < -2 \\ z + z^2 + tz^3; -2 \le z \le 3 \\ \sqrt{\ln(2z) + t}; z > 3 \end{cases}$

(Разветвляющийся алгоритм).

3. Получить значения функции**Y=F(x)** с использованием цикла**(do ..).**

$$\mathrm{F(x)} = arctg(x + 1/x + 10); \quad \text{для } \mathrm{x} = 1.2, 1.3, \ldots, 2.5$$

(Циклический алгоритм).

4. Дан массив **R(10**). Найти сумму элементов массива **R** со значениями из диапазона **[5,25] (Обработка массива).**

2. Методические материалы

2.1 Автоматизация решения практических задач с использованием языка программирования высокого уровня VBA в приложениях WINDOWS

Этапы решения задач на компьютере

Решение задачи на компьютере – это автоматический процесс преобразования информации в соответствии с целью, поставленной в задаче.

Процесс решения задачи на ЭВМ – совместная деятельность человека и компьютера. Этот процесс является достаточно сложным и трудоемким, поэтому представляется в виде нескольких последовательных этапов. При этом на долю человека приходится творческая деятельность, а на долю компьютера – автоматическая обработка информации в соответствии с заданным алгоритмом.

Решение задач на ЭВМ состоит из следующих этапов:

1. постановка задачи;
2. построение математической модели;
3. разработка алгоритма;
4. запись алгоритма на языке программирования;
5. создание программного кода в интегрированной среде программирования;
6. отладка и тестирование программы;
7. получение и анализ результатов.

Постановка задачи

Постановку задачи выполняет человек, хорошо представляющий предметную область задачи. Он должен:

- определить цель решения задачи;
- дать точную формулировку задачи;
- предложить идею решения задачи;
- определить необходимый объем информации;
- описать исходные данные и указать способы их хранения;
- определить точность вычислений и форму выдачи результатов.

Построение математической модели

Чтобы решить задачу, связанную с исследованием реального объекта, необходимо описать этот объект в математических терминах, связанных определенными соотношениями (формулами), то есть построить его математическую модель. Такая модель всегда идеализирует реальный объект, но она позволяет математически строго решить задачу исследования объекта. Математическое описание поставленной задачи выполняет человек, при этом он должен выполнить:

- анализ похожих решённых задач;
- анализ условий существования решения;
- анализ технических и программных возможностей.

Разработка алгоритма

Алгоритм решения задачи разрабатывается на основе построенной математической модели и представляет конечную последовательность предписаний (правил), которая определяет процесс преобразования исходных данных в результаты решения задачи. Алгоритм разрабатывается одним из существующих способов, чаще всего в виде блок-схемы. Разработку алгоритма выполняет человек, имеющий знания в области программирования. Уровень его квалификации определяет эффективность разработанного алгоритма.

Запись алгоритма на языке программирования

Этот этап выполняет человек, умеющий программировать, так как он должен выбрать язык программирования, наиболее удобный для решения данной задачи. Программа – это один из способов представления алгоритма с использованием выбранного языка программирования для реализации его в компьютере.

Создание программного кода в интегрированной среде программирования

Программный код создается при помощи редактора кода интегрированной среды программирования. Внешний вид и функциональные возможности интегрированной среды программирования зависят от выбранного языка программирования высокого уровня.

Отладка и тестирование программы

Отладка программы – это проверка программы на наличие ошибок компиляции, ошибок выполнения и логических ошибок. Ошибки компиляции возникают, например, при наличии грамматических ошибок. Ошибки выполнения появляются после успешного завершения компиляции программного кода на стадии выполнения, например из-за некорректного ввода данных.

Логические ошибки возникают в результате неверно разработанного алгоритма. В ходе отладки происходит совершенствование программного кода. Во время отладки программы логические ошибки исправляют путем контрольного тестирования. Тесты (контрольные примеры) составляются так, чтобы проверить все возможные варианты работы алгоритма. Проверка осуществляется путем сравнения заранее известных результатов тестов с результатами, полученными компьютером.

Получение и анализ результатов

После устранения всех видов ошибок, выявленных отладкой и тестированием, получают результаты решения поставленной задачи. Получение результатов может быть многократным в зависимости от смены исходных данных, поскольку решение должно быть универсальным для задач подобного класса.

Алгоритмизация вычислительных процессов

Основные понятия об алгоритмизации задач

Алгоритм и его свойства

Алгоритм (алгорифм) – любая конечная последовательность основных математических и логических действий, однозначно определяющих процесс преобразования исходных данных в конечные результаты решения задачи.

Само слово «алгоритм» происходит от имени хорезмского учёного Абу Абдуллах Мухаммеда ибн Муса аль-Хорезми (от латинской формы арабского имени – Algorithmi). Считается, что он первым решил квадратное уравнение.

Алгоритм – одно из основных понятий математики и информатики. Исходными данными и результатами алгоритма могут служить самые разнообразные объекты. Это открывает возможность широкого применения понятия алгоритма. Например, можно говорить об алгоритмах перевода с одного языка на другой, об алгоритмах управления (диспетчеризация поездов, самолетов, городского транспорта, функционирования предприятий и т. д.).

Основные свойства алгоритмов

1. Дискретность – прерывность, раздельность. Алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов (этапов).
2. Определенность – каждое правило алгоритма должно быть четким и однозначным. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.
3. Понятность для исполнителя – исполнитель алгоритма должен знать, как его выполнять.
4. Результативность (конечность). Это свойство состоит в том, что алгоритм должен приводить к решению задачи за конечное число шагов.
5. Массовость – означает, что алгоритм решения задачи разрабатывается в общем виде, т. е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными.

Формы представления алгоритмов

На практике наиболее распространены следующие формы представления алгоритмов:

- словесная – запись на естественном языке;
- в псевдокодах – полуформализованное описание алгоритма на условном алгоритмическом языке, включающее в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и т. д.;
- табличная;
- графическая – с помощью графических символов;

- программная – запись на искусственном языке (языке программирования).

Словесный способ не имеет широкого применения по следующим причинам:

- описания не строго формализуемы;
- страдает многословностью записей;
- допускает неоднозначность толкования отдельных предписаний.

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых конструкций).

Графическое представление алгоритма является наиболее компактным и наглядным по сравнению с остальными формами представления алгоритмов. При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т. п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями потоков, определяющими очередность выполнения действий.

Ниже приведены наиболее часто используемые блочные символы (*табл. 1)*, которые соответствуют действующему ГОСТу.

Стандарт на условные графические обозначения в схемах алгоритмов – ГОСТ 19.003-80

Единая система программной документации

СХЕМЫ АЛГОРИТМОВ И ПРОГРАММ.
ОБОЗНАЧЕНИЕ УСЛОВНЫЕ ГРАФИЧЕСКИЕ

ГОСТ 19.003-80

Взамен
ГОСТ 19428-74

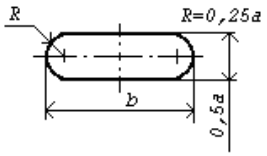
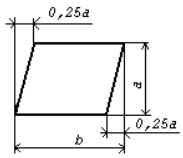
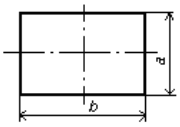
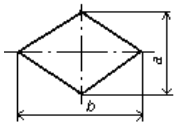
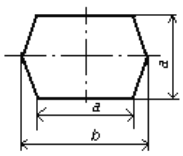
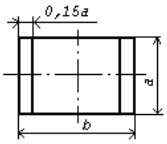
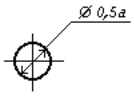
United system for program documentation.
Graphical flowchart symbols.

Настоящий стандарт распространяется на условные графические обозначения (символы) в схемах алгоритмов и программ, отображающие основные операции процесса обработки данных и программирования для систем программного обеспечения вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Стандарт устанавливает перечень, наименование, форму, размеры символов и отображаемые символами функции.

Перечень, наименование, обозначение символов и отображаемые ими функции

Таблица 1

Наименование	Обозначение и размеры в мм	Функция
29. Пуск - останов		Начало, конец, прерывание процесса обработки данных или выполнения программы
12. Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод)
1. Процесс		Выполнение операций или группы операций, в результате которых изменяется значение, форма представления или расположение данных
2. Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
3. Модификация		Выполнение операций, меняющих команды или группу команд, изменяющих программу
4. Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов или программ
26. Линия потока		Указание последовательности между символами
28. Соединитель		Указание связи между прерванными линиями потока, связывающими символами

Соотношение геометрических элементов символов

Размер a должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер a на число, кратное 5. Размер b равен 1,5a.

Примечание. При ручном выполнении схем алгоритмов и программ для обязательных символов 1-3, 12, 29 и рекомендуемых символов 3 и 4 допускается устанавливать b равным 2a.

Блок «процесс» применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Иногда для улучшения наглядности схемы несколько отдельных блоков объединяют в один блок.

Блок « решение» используется для обозначения переходов управления по условию. В каждом таком блоке должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок «модификация» – видоизменение, преобразование используется для организации циклических структур. Внутри блока записывается параметр цикла, для которого указывается его начальное значение, граничное условие и шаг изменения параметра цикла для каждого повторения.

Блок «предопределенный процесс» служит для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращения к библиотечным подпрограммам.

Разновидности структур алгоритмов

По своей структуре алгоритмы разделяют:

- линейные;
- разветвляющиеся;
- циклические;
- предопределенный процесс.

Перечисленные алгоритмы могут быть составной частью алгоритмического процесса решения задачи. Алгоритмы линейной структуры, как правило, являются составной частью любого алгоритмического процесса.

Алгоритм линейной структуры

В алгоритмах линейной структуры все блоки имеют последовательное соединение логической связью передачи информационных потоков (*рис. 1)*. В них могут использоваться все блоки, за исключением блоков проверки условия и модификации. В основном алгоритм линейной структуры – это объединение нескольких, следующих друг за другом блоков « процесс» и блоков « ввода/ вывода», в котором каждое последующее действие – операция выполняется строго за предыдущим.

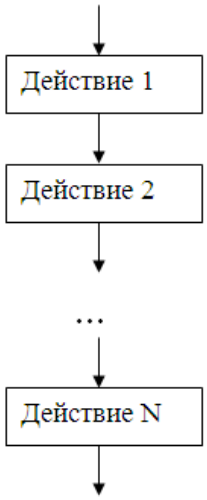


Рис. 1. Алгоритм линейной структуры

Алгоритм разветвляющейся структуры

Алгоритмы, в которых в зависимости от выполнения некоторого логического условия происходит разветвление вычислений по одному из нескольких возможных направлений, называют разветвляющимися. Такие алгоритмы предусматривают выбор одного из альтернативных путей продолжения вычислений. Каждое возможное направление вычислений называется ветвью. Логическое условие называют простым, если разветвляющийся процесс имеет две ветви, и сложным, – если процесс разветвляется на три и более ветви. Любое сложное логическое условие может быть представлено в виде простых условий.

В алгоритм простой разветвляющейся структуры заложены два варианта выполнения алгоритма. В этом случае можно выделить два вида структуры:

1. в зависимости от результата проверки записанного условия выполняются только действия ветви «да» (действия 1 и 2) или ветви «нет» (действия 3 и 4). Такая структура разветвления носит название полный выбор – полная альтернатива (*рис. 2.1*);
2. в зависимости от результата проверки условия либо выполняются действия ветви «да» (действия 1 и 2), либо пропускаются (*рис. 2.2*), образуя неполный выбор – неполную альтернативу.

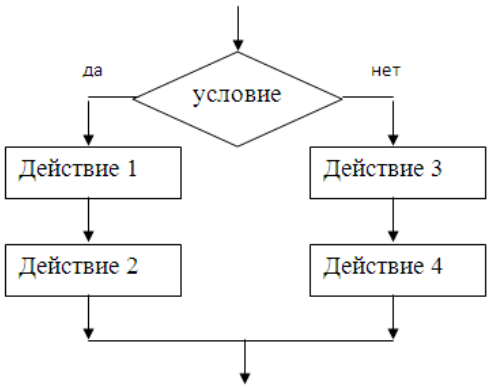


Рис. 2.1. Полная альтернатива

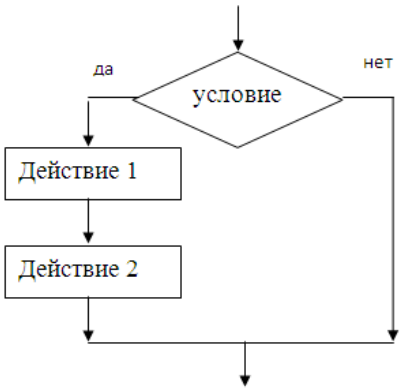


Рис. 2.2. Неполная альтернатива

Ниже представлена структура множественного выбора. В этом случае от значения параметра выбора, влияющего на выполнение условий, будет выполнено одно из перечисленных действий (*рис. 3)*.

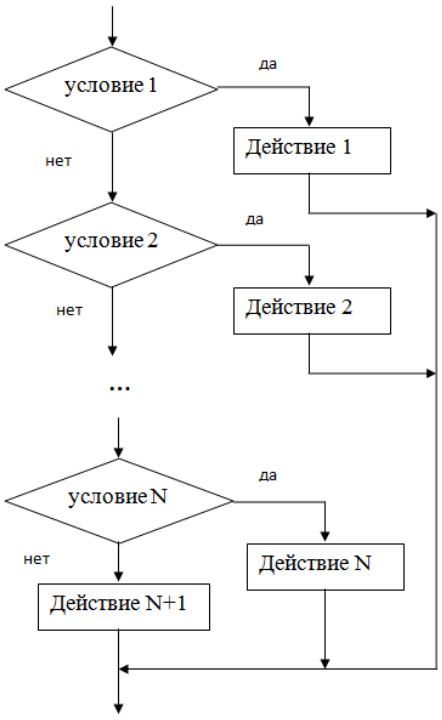


Рис. 3. Множественный выбор

Алгоритмы циклических структур

Алгоритмы циклических структур обеспечивают выполнение отдельных фрагментов алгоритма заданное или конечное, но неопределенное число раз, до получения результата при выполнении некоторого условия. Циклические алгоритмы позволяют существенно сократить объем программы за счет многократного выполнения группы повторяющихся вычислений, так называемого тела цикла. Специально изменяемый по заданному закону параметр, входящий в тело цикла, называется переменной цикла. Переменная цикла используется для подготовки очередного повторения цикла и отслеживания условий его окончания. В качестве переменной цикла

используют любые переменные, индексы массивов, аргументы вычисляемых функций и тому подобные величины. Во время выполнения тела цикла параметры переменной цикла изменяются в интервале от начального значения до конечного значения с заданным шагом (при организации циклических вычислений необходимо предусмотреть задание начального значения переменной цикла, закон ее изменения перед каждым новым повторением и ее конечное значение, при достижении которого произойдет завершение цикла).

Цикл называется простым, если в теле цикла нет разветвлений и циклических структур. В противном случае он называется сложным.

Циклические алгоритмы бывают двух видов: детерминированные и итерационные.

Циклы, в которых число повторений заранее известно из исходных данных или определено в ходе решения задачи, называют детерминированными. Для организации детерминированных циклов наиболее целесообразно использовать блок модификации, внутри которого указывается переменная цикла, ее начальное и конечное значения, а также шаг ее изменения.

Цикл с заданным числом повторений представлен на *рис. 4*,

где

P –параметр цикла (имя);

P1 – начальное значение параметра цикла (имя или константа);

P2 – конечное значение параметра цикла (имя или константа);

P3 – шаг изменения параметра цикла (имя или константа);

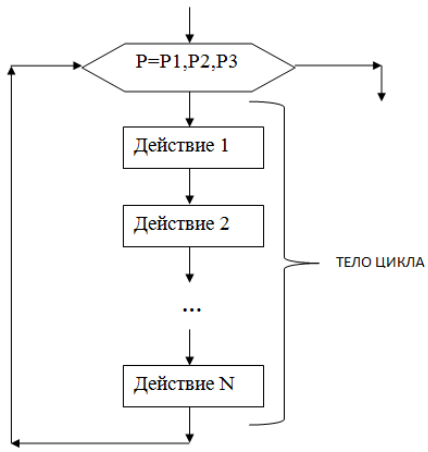


Рис. 4. Цикл с заданным числом повторений

Тело цикла выполняется столько раз, сколько разных значений примет параметр в заданных пределах.

Организовать подобный цикл возможно и при использовании блока проверки условия вместо блока модификации.

Циклы, в которых число повторений неизвестно из исходных данных и не определено по ходу решения задачи, называют итерационными. В

итерационных циклах для организации выхода из тела цикла предусматривается проверка некоторого заранее заданного условия, для чего используют блок проверки условия. В итерационных циклах невозможно использовать блоки модификации, так как при организации таких циклов заранее неизвестно количество изменений переменной цикла и ее конечное значение.

В зависимости от местонахождения блока проверки условия итерационные циклы могут быть организованы как циклы с предусловием (блок проверки условия размещен перед телом цикла) или с постусловием (блок проверки условия размещен после тела цикла). Если в цикле с предусловием входящие в тело цикла действия могут не выполняться ни разу (если начальное значение параметра цикла удовлетворяет условию выхода из цикла), то в цикле с постусловием они выполняются как минимум один раз (даже если начальное значение параметра цикла удовлетворяет условию выхода из него).

На *рис. 5* представлена блок-схема цикла с предусловием. Если условие выполняется, то выполняются действие 1, … , действие N (тело цикла) ветви алгоритма «да», иначе тело цикла не выполнится ни разу.

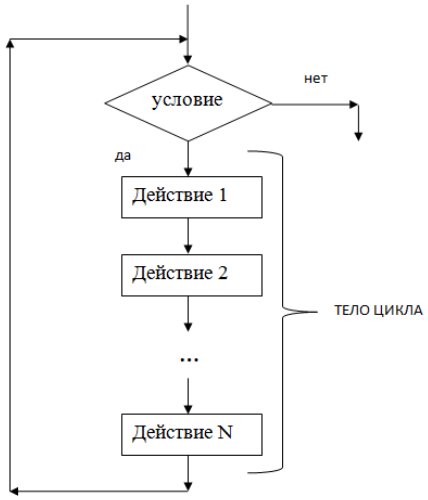


Рис. 5. Цикл с предусловием

На *рис. 6* представлена блок-схема цикла с постусловием. Тело цикла в данном случае выполнится хотя бы один раз. Если условие не выполняется, то выполняются, то вновь повторяются действие 1, … , действие N (тело цикла) ветви «нет», иначе тело цикла больше не выполнится ни разу.

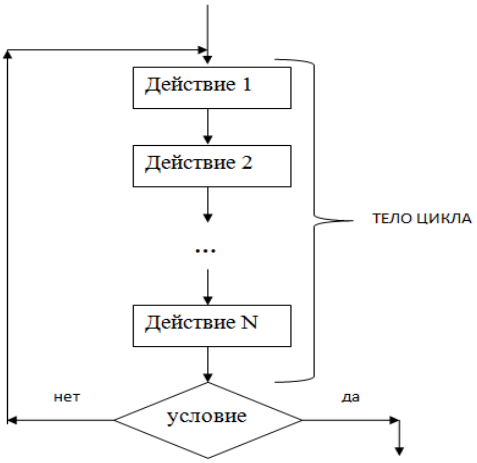


Рис. 6. Цикл с постусловием

На *рис. 7* представлен блок – предопределенный процесс обращения к ранее написанной подпрограмме.

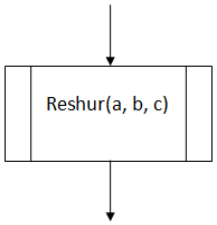


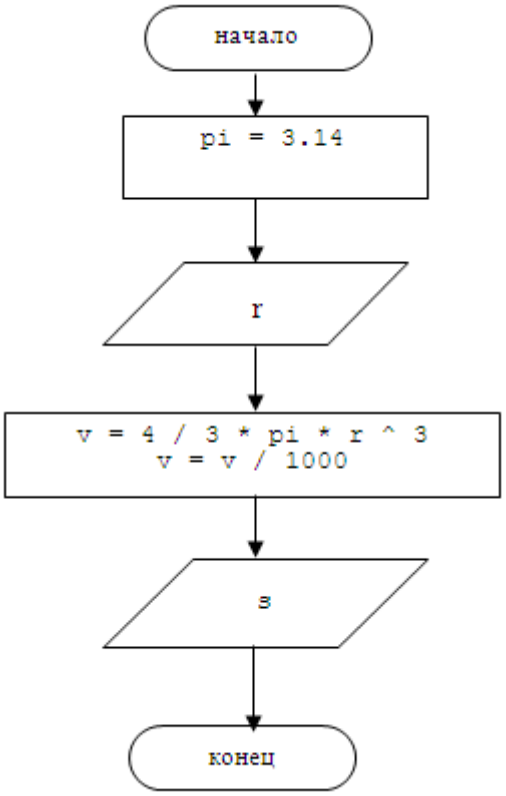
Рис. 7. Блок – предопределенный процесс

Примеры блок-схем алгоритмов

Конечным продуктом разработки алгоритма и реализации его на ПК является программа, записанная на языке программирования высокого уровня.

Пример 1. Алгоритм линейной структуры

Определить объем шара в литрах. Радиус шара равен 5 см.



$$v = \frac{4}{3}\pi \cdot r^3$$

Обозначения в блок-схеме:
v – объем шара (л);
r – радиус шара (см);
pi – константа (3,14).

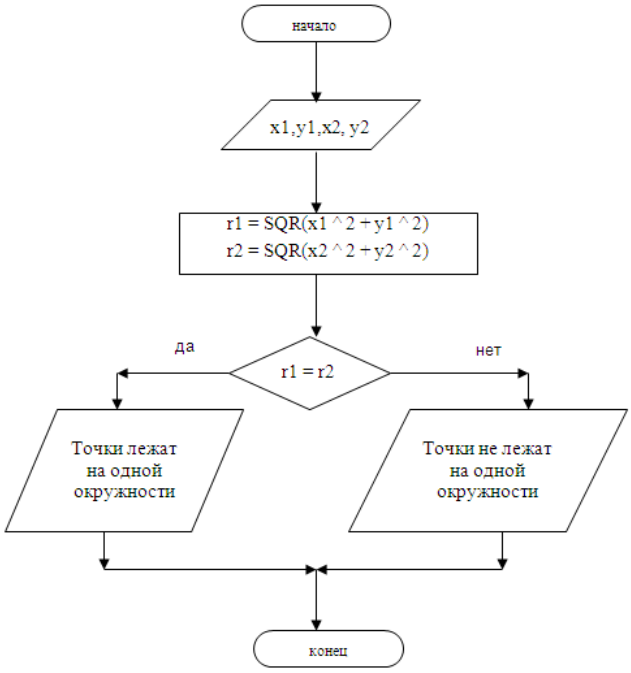
Рис. 8. Пример 1. Блок-схема алгоритма

```
Private Sub Lin()
'Определить обьем шара в литрах.Радиус шара равен 5 см.
Dim r As Single, v As Single
Const pi = 3.14
r = InputBox("введите радиус шара в см", , 5)
'Обьем шара в кубических сантиметрах
v = 4 / 3 * pi * r ^ 3
'Обьем шара в литрах (кубических дециметрах)
v = v / 1000
MsgBox "Обьем шара - " & Format(v, "##0.000") & " л", , "радиус шара - " & r & " см"
End Sub
```

Рис. 9. Пример 1. Программный код

Пример 2. Алгоритм разветвляющейся структуры

Определить принадлежность двух точек к одной окружности с центром в начале координат, если заданы координаты этих точек x1, y1, x2, y2.



Обозначения в блок-схеме:
x1, y1 – координаты первой точки;
x2, y2 – координаты второй точки;
r1 – расстояние от начала координат до первой точки;
r2 – расстояние от начала координат до второй точки.

Рис. 10. Пример 2. Блок-схема алгоритма

```
Private Sub vetv()
Dim x1 As Single, y1 As Single, r1 As Single, _
x2 As Single, y2 As Single, r2 As Single
'Заданы координаты двух точек.Определить лежат ли
'они на одной окружности с центром в начале координат.
MsgBox "введите координаты 1 точки "
x1 = InputBox("x=", " координаты 1 точки ")
y1 = InputBox("y=", " координаты 1 точки ")
MsgBox "введите координаты 2 точки "
x2 = InputBox("x=", " координаты 2 точки ")
y2 = InputBox("y=", " координаты 2 точки ")
r1 = Sqr(x1 ^ 2 + y1 ^ 2)
r2 = Sqr(x2 ^ 2 + y2 ^ 2)
If r1 = r2 Then
MsgBox "Точки лежат на одной окружности"
Else
MsgBox "Точки не лежат на одной окружности"
End If
End Sub
```

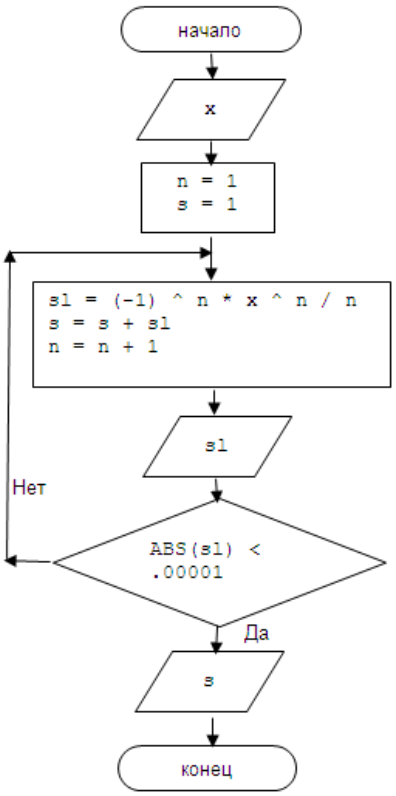
Рис. 11. Пример 2. Программный код

Пример 3. Алгоритм циклической структуры

Вычислить приближенное значение суммы сходящегося ряда (0 < x < 1)

$$1 - x + \frac{x^2}{2} - \frac{x^3}{3} + \frac{x^4}{4} - \ldots,$$

пренебрегая членами ряда, по абсолютной величине меньшими 0,00001.



Обозначения в блок-схеме:
x – параметр;
s – сумма ряда;
sl – текущий член ряда;
n – переменная-«счётчик».

Рис. 12. Пример 3. Блок-схема алгоритма

```
Private Sub summa()
Dim x As Single, s As Single, sl As Single, _
n As Integer, ssl As String
'Вычислить приближенное значение суммы сходящегося
'ряда
'           2     3     4
'          X     X     X
'1 - X + --- - --- + --- -
'          2     3     4
'
'0 < X < 1,
'пренебрегая членами ряда,по абсолютной величине
'меньшими 0,00001
x = InputBox("введите число X", "0 < X < 1")
n = 1
s = 1
ssl = "1"
Do
sl = (-1) ^ n * x ^ n / n
ssl = ssl & Chr(10) & Chr(13) & Format(sl, "0.0000000")
s = s + sl
n = n + 1
Loop Until Abs(sl) < 0.00001
MsgBox "X=" & x & Chr(10) & Chr(13) & "члены ряда:" & _
Chr(10) & Chr(13) & ssl, , "сумма ряда = " & s
End Sub
```

Рис. 13. Пример 3. Программный код

Язык программирования высокого уровня Visual Basic for Applications (VBA)

Visual Basic for Applications (VBA) – визуальная объектно-ориентированная система программирования, предназначенная для создания программ внутри офисных приложений Microsoft Office.

Языковой основой VBA является классический язык BASIC, а VBA образует ядро приложения Visual Basic.

С помощью VBA можно легко и быстро создавать пользовательские приложения, используя единую для всех офисных программ среду и язык. Язык VBA прост в освоении и позволяет быстро получать ощутимые результаты — конструировать профессиональные приложения, решающие практически все задачи, встречающиеся в среде Windows.

Будучи языком, встроенным в какое-либо офисное приложение, VBA улучшает возможности данного приложения. Рекомендуется начинать разрабатывать приложения для одной офисной программы, например Microsoft Excel, поскольку табличный процессор является наиболее популярной офисной программой и обладает достаточно простой объектной моделью. Однако можно создавать приложения и для других офисных программ.

Следующий этап после разработки алгоритма решения поставленной задачи – запись алгоритма на языке программирования. Язык программирования – искусственный язык. Он опирается на словарь служебных слов и команд, а также систему правил записи конструкций языка. С помощью языка программирования создается текст программы (программный код). Программа – полное и четкое описание алгоритма на языке программирования. Чтобы программа работала, ее нужно перевести в машинный код. Этим занимаются специальные программы – компиляторы и интерпретаторы.

Интерпретаторы просматривают и сразу выполняют инструкции языка, содержащиеся в тексте программы, подробно информируя разработчика программы о возникающих проблемах.

В систему программирования Visual Basic for Applications помимо других компонентов в качестве транслятора включен интерпретатор.

Алфавит VBA

Алфавит языка включает следующие символы:

- 53 буквы – прописные и строчные буквы латинского алфавита и знак подчеркивания(_);
- 10 цифр(0 - 9);
- 23 специальных символа (+ - * / . , : ; = < > ‘ “ ( ) [ ] & $ @ ^ \ пробел);
- составные символы (<> <= >=);

Неделимые последовательности знаков алфавита образуют слова, отделяемые друг от друга разделителями и несущие определенную смысловую нагрузку в программе. Разделителями могут служить пробел, двоеточие, точка

и другие специальные символы, и их комбинации.

Используемые в программном коде слова подразделяются на:

- ключевые;
- стандартные идентификаторы;
- идентификаторы пользователя.

Ключевые (зарезервированные) слова имеют фиксированное написание и однозначно определенный смысл, который не может быть изменен.

Например, Until, Goto, If, Loop, For, Do и т. д.

Идентификаторы пользователя применяются для обозначения имен констант, переменных, процедур и функций, типов данных, меток. Имена задаются программистом и должны отвечать следующим правилам:

- длина имени не должна превышать 255 символов;
- имя не может содержать знаков точки, пробела, % , & , ! , #, @, $;
- имя может состоять из любой комбинации букв, цифр и других символов, начинающейся с буквы;
- нельзя использовать имена, совпадающие с именами встроенных функций и процедур и ключевыми словами;
- имена должны быть уникальны внутри области, в которой они определены;
- регистр букв (верхний или нижний) не имеет значения, но для большей наглядности текста программы и облегчения ее понимания следует умело сочетать верхний и нижний регистры, например, вместо имени код лучше написать Код, или вместо имени сумма_вклада – написать Сумма_вклада.

Переменные, константы и стандартные функции

Переменная обозначается идентификатором, определяющим некоторую область в памяти, в которой хранится ее значение. Это значение может изменяться в процессе выполнения программы. Каждая переменная принадлежит к определенному типу данных, и поэтому должна быть объявлена в разделе описаний до выполнения каких- либо действий с нею.

Под описанием переменной подразумевается указание типа данных. В VBA переменные можно указывать явным и неявным образом. Для того чтоб в VBA включить обязательное объявление всех переменных необходимо в начале программы добавить строку: **Option Explicit**. После этой инструкции, на каждой не объявленной переменной будет происходить остановка программы, и отображаться ошибка до тех пор, пока всем переменным не будет присвоен тип в разделе **Dim.**

Примечание. *Для того чтоб в VBE (Visual Basic Editor) инструкция* ***Option Explicit*** *вставлялась автоматически в каждый новый модуль, необходимо в настройках редактора VBE активировать данную опцию:* ***Tools-Options...****, на вкладке* ***Editor*** *поставить галочку "Require Variable Declaration".*

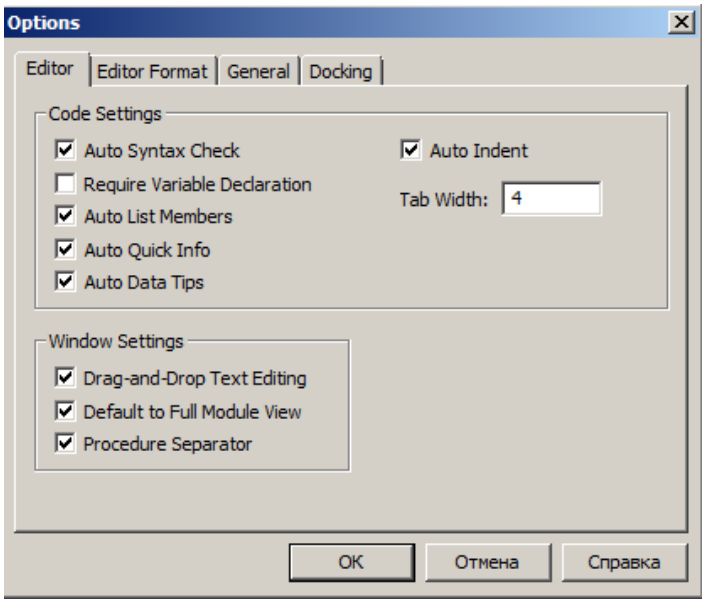


Рис. 14. Опция "Require Variable Declaration"

Описание переменных явным образом

Для описания типа переменных на уровне процедуры используется инструкция **Dim**, которая имеет следующий вид – после ключевого слова Dim следуют объявления переменных разделенных запятыми.

Dim <имя переменной1>[As Тип], <имя переменной2> [As Тип],…

где

Dim – ключевое слово;

имя переменной (идентификатор) – имя, удовлетворяющее стандартным правилам VBA;

Тип – один из допустимых типов данных VBA *(табл. 2)*.

Если тип переменной не задан, то она по умолчанию получает тип Variant и результат зависит от значения переменной.

Например, следующая инструкция описывает X и Y, как переменные целого типа(Integer), а Z – как переменную вещественного типа (Single).

Dim X AS Integer, Y AS Integer, Z As Single

В случае такого описания переменные X и Y будут хранить в памяти только числовые значения целого типа, Z будет хранить в памяти только числовые значения вещественного типа.

Типы данных в VBA

Тип данных определяет множество допустимых значений, которое может принимать указанная переменная.

Таблица 2

Тип данных	Диапазон хранимых значений
Byte(байт)	0..255
Boolean(логический)	True, False
Integer(целый)	от -32768 до 32767
Long(длинный целый)	от -2147483648 до 2147483647
Single(вещественный одинарной точности)	для отрицательных значений от $-3{,}402823*10^{38}$ до $-1{,}401298*10^{-45}$ для положительных значений от $1{,}401298*10^{-45}$ до $3{,}402823*10^{38}$
Double (вещественный двойной точности)	для отрицательных значений от $-1{,}79769313486232*10^{308}$ до $-4{,}94065645841247*10^{-324}$ для положительных значений от $4{,}94065645841247*10^{-324}$ до $1{,}79769313486232*10^{308}$
Currency(с фиксированной десятичной точкой)	от -922337203685477,5808 до 922337203685477,5807
Date(дата и время)	даты от 1.01.100 до 31.12.9999, время от 00:00:00 до 23:59:59
String(символьный)	длина строки до 2 млрд. символов
Variant	для хранения различных типов данных
Object	для хранения ссылок на объекты
Тип данных, определяемый пользователем с помощью ключевого слова Type	Позволяет хранить в переменной такого типа множество различных значений разного типа. Используется для описания структур данных.

Константы – элементы данных, значения которых, в процессе выполнения программы не меняются. Константы можно использовать как аргументы для процедур, в математических операциях, операциях сравнения и т. п. Константы в программе могут быть заданы явно своим значением (литеральные константы) или обозначены именем.

Существуют правила, которые необходимо соблюдать при написании литеральных констант.

Константы типа String

- строковые константы должны быть заключены в двойные кавычки (");

- пустая строковая константа (так называемая "нулевая строка") обозначается двумя двойными кавычками, между которыми нет даже пробела ("");
- строковая константа обязательно должна вся находиться на одной строке.

Числовые константы

- числовые константы могут содержать любой из числовых типов VBA;
- числовые константы должны состоять только из числовых символов от 0 до 9;
- числовая константа может начинаться со знака минус (-) и может содержать десятичную точку;
- для числовых констант можно использовать экспоненциальное представление.

Константы типа Date

- константы типа Date необходимо помещать между знаками фунта (#), при этом формат задания даты может быть различным:

  #3-5-99#

  #february 13, 2015 10:15am#

  #jun-20-2015#

  # 9 april 2015#

- VBA переформатирует дату к следующему виду: #4/9/2015#;
- если пропустить знак фунта (#) при записи литеральной константы даты, VBA не сможет правильно интерпретировать константу даты как дату;
- нельзя заключать литеральные константы даты в двойные кавычки, т. к. в этом случае VBA распознает такую константу как строковую.

Константы типа Boolean

Существуют только два вида констант типа Boolean: *True, False*. При этом их нельзя брать в кавычки или сокращать.

Именованные константы

Если константа обозначена именем, то она должна быть описана в разделе описаний.

Для описания типа именованных констант на уровне процедуры используется инструкция **Const,** которая имеет следующий вид – после ключевого слова Const следуют описания именованных констант (с указанием их значений) разделенные запятыми.

Const <имя константы > [AS Тип] = <Выражение>,…

где

имя константы – имя, удовлетворяющее стандартным правилам VBA;

Тип – один из допустимых типов данных VBA *(табл. 2)*;

Выражение – это любое значение или формула, возвращающая значение, которое должно использоваться в качестве константы.

Например, следующая инструкция описывает именованные константы с именами R и K.

Const R = "Результат ", K As Single = 0.2

Возможно описание без указания типа:

Const a = 4.55 (в этом случае описана константа типа **Variant)**.

Встроенные функции

Функция оперирует определенными данными и всегда возвращает некоторое результирующее значение. Функция – это встроенная формула, выполняющая действия над выражениями и генерирующая какое-то значение, которое вставляется в программу в том месте, где появляется имя этой функции. Некоторые из функций не требуют аргументов. Возвращаемое значение функции, как правило, используется в дальнейших вычислениях программы.

В VBA имеется большой набор встроенных функций и процедур, упрощающих программирование, которые можно разделить на следующие категории:

- математические;
- функции проверки типов;
- функции обработки строк;
- функции преобразования форматов;
- функции даты и времени.

Математические функции

Функция –	**возвращаемое значение**
Abs(x) –	модуль (абсолютная величина) числа x
Sqr(x) –	квадратный корень из числа x
Exp(x) –	возведение основания натурального логарифма в степень x
Log(x) –	логарифм натуральный аргумента x
Rnd –	случайное число из интервала [0,1]
Cos(x) –	косинус аргумента x
Sin(x) –	синус аргумента x
Atn(x) –	арктангенс от аргумента x
Tan(x) –	тангенс от аргумента x
Sgn(x) –	знак числа x
Fix(x) и Int(x) –	функции отбрасывают дробную часть числа x и возвращают целое значение

Примечание. Разница между функциями Fix(x) и Int(x) состоит в том, что для отрицательных значений аргумента Int(x) возвращает ближайшее отрицательное целое число, меньшее или равное числу x, а Fix(x) – ближайшее отрицательное целое число, большее или равное числу x.

Функции проверки типов

Функция –	**проверка**
IsArray(переменная) –	является ли переменная массивом
IsDate(переменная) –	является ли переменная датой
IsError(переменная) –	является ли переменная кодом ошибки
IsNull(переменная) –	является ли переменная пустым значением
IsNumeric(переменная) –	является ли переменная числовым значением
IsObject(переменная) –	является ли переменная объектом
IsEmpty (переменная) –	была ли переменная описана инструкцией Dim

Функции преобразования форматов

Val(строка) – возвращает числа, содержащиеся в строке, как числовое значение соответствующего типа.

Str(число) – возвращает значение типа Variant (String), являющееся строковым представлением числа (в качестве допустимого десятичного разделителя функция str воспринимает только точку).

Format(Выражение[,Формат [,Первый день недели[,Первая Неделя Года]]]) – возвращает значение типа Variant(String), содержащее выражение, оформленное согласно инструкциям, заданным в описании формата,

где

Выражение – обязательный аргумент (любое допустимое выражение – комбинация ключевых слов, операторов, переменных и констант, результатом которой является строка, число или объект);

Формат – необязательный параметр (любое допустимое именованное или определенное пользователем выражение формата).

Если к имени функции добавляется знак $ (Format$), то функция возвращает значение типа String.

При создании числового формата пользователя можно использовать следующие символы:

0 – резервирует позицию цифрового разряда. Отображает цифру или нуль. Если у форматируемого числа есть какая-нибудь цифра в этой позиции разряда, где в строке формата находится 0, функция отображает эту цифру, если нет, то в этой позиции отображается нуль;

– действие данного символа аналогично действию 0 с той лишь разницей, что незначащие нули не отображаются;

. – резервирует позицию десятичного разделителя, определяет, сколько разрядов необходимо отображать слева и справа от десятичной точки;

% – резервирует процентное отображение числа;

, – разделяет сотни от тысяч.

Если в формате стоит "FIXED", то число форматируется до двух знаков

после точки.

Например, функция Format(W, "###00.0") возвращает переменной W числовое значение в формате, указанном в кавычках.

Функции обработки строк

Рассмотрим наиболее часто употребляемые функции.

Chr(код) – преобразует ASCII-код в строку.

Например, Chr(10) осуществляет переход на новую строку, Chr(13) – возврат каретки.

Lcase(строка) – преобразует строку к нижнему регистру.

Ucase(строка) – преобразует строку к верхнему регистру.

Len(строка) – возвращает число символов строки.

Space(n) – возвращает строку, состоящую из n пробелов.

String(n,char) – возвращает строку, состоящую из n символов (char).

LTrim(строка) – возвращает копию строки без пробелов в ее начале.

RTrim(строка) – возвращает копию строки без пробелов в ее конце.

Trim(строка) – возвращает копию строки без пробелов в ее начале и конце.

Left(string, length) – возвращает подстроку, состоящую из заданного числа первых символов исходной строки.

Right(string, length) – возвращает подстроку, состоящую из заданного числа последних символов исходной строки.

Mid(string, pos[, length]) – возвращает подстроку строки, содержащую указанное число символов, где

string – строковое выражение, из которого извлекается подстрока;

pos – позиция символа в строке String, с которого начинается нужная подстрока;

length – число возвращаемых символов подстроки.

Функции времени и даты

Возвращают значение типа Variant.

Day(дата) – возвращает значение, содержащее целое число, которое представляет день в значении даты.

Month(дата) – возвращает значение, содержащее целое число, которое представляет месяц в значении даты.

Year(дата) – возвращает значение, содержащее целое число, которое представляет год в значении даты.

Аргумент дата — само значение даты или выражение, ее определяющее.

Date возвращает значение, содержащее системную дату.

Time возвращает значение, содержащее текущее время по системным часам компьютера.

Now возвращает значение, содержащее текущую дату и время по системному календарю и часам компьютера.

Выражения

При выполнении программы осуществляется обработка данных, в ходе которой с помощью выражений вычисляются и используются различные значения. Выражение представляет конструкцию, определяющую состав данных, операции и их порядок выполнения над данными.

Выражения могут включать:

- операнды;
- знаки операций;
- круглые скобки.

В простейшем случае выражение может состоять из одной константы (вырожденное выражение).

Операнды представляют собой данные, над которыми выполняются действия. В качестве операндов могут быть использованы константы, переменные, элементы массивов и функции.

Операции определяют действия, которые выполняются над операндами.

Круглые скобки используются, если необходимо нарушить приоритет выполнения операций.

В зависимости от типов операций и операндов выражения могут быть арифметическими, логическими, строковыми и выражениями типа даты.

Арифметические операции выполняются над числовыми данными и их результатом являются числа.

Арифметические операции – сложение (+), вычитание (-), умножение (*), деление (/), возведение в степень (^), целочисленное деление (\), остаток от деления по модулю (mod).

Операции сравнения или отношений. Результатом операций является значение типа Boolean, которое принимает одно из двух логических значений: True (истина) или False (ложь).

Операции отношения – равно (=), не равно (<>), меньше или равно (<=), больше или равно (>=), меньше (<), больше (>), ссылка на объект (Is), подобие (Like).

Логические операции используются в логических выражениях и результатом таких выражений являются логические значения **(True или False)**.

Логические операции – логическое и (And), логическое или (Or), логическое отрицание (Not), логическое исключающее или (Xor).

Операции над строками используются в выражениях над строками, и результатом вычисления такого выражения является строка символов.

Операции над строками – соединение двух строк (+) и конкатенация (&), однако предпочтительнее использовать знак &.

Приоритеты операций

VBA выполняет операции выражения в соответствии с их приоритетами (указаны в сторону понижения приоритета):

1. вызов функции и выражения в скобках;
2. ^ (возведение в степень);
3. - (смена знака числа);
4. * (умножение), / (деление), \ (целочисленное деление), Mod (деление по модулю);
5. + и – (сложение и вычитание);
6. <, >, >=, <=, <>, = (операции отношения);
7. Not (логическое отрицание);
8. And (логическое и);
9. Or (логическое или);
10. Xor (логическое исключающее или).

Структура программы на языке VBA

Исходный текст программы состоит из последовательности строк, каждая из которых может начинаться с любой позиции.

Структура программы включает:

- заголовок программы;
- раздел описаний (последовательность инструкций описаний – объявлений);
- исполняемую часть (последовательность исполняемых инструкций);
- Конец записи программы.

Заголовок программы находится в начале программы и имеет вид:

Sub < имя программы>()

Раздел описаний содержит описание элементов программы.

Исполняемая часть является основной частью («телом» программы), в которой над объявленными объектами производятся определенные действия, позволяющие получить требуемый результат.

Структура программы (с именем Prim1) на VBA с использованием констант, простых переменных выглядит следующим образом:

```
Sub Prim1()                                  ’заголовок программы
  {Инструкции описания:
   Const  <имя= выражение>                   ’ описание констант
   Dim<имя переменной  > As Тип,…}   ’ описание переменных
   {Исполняемая часть –
   <исполняемые инструкции>,
   <исполняемые функции>}
End Sub                                      ’конец записи программы
```

Примечание. При создании программ необходимо выполнять требования, перечисленные ниже.

1. Описания данных должны предшествовать описанию действий и содержать упоминание всех объектов, используемых в инструкциях.

2. В одной строке программы может быть записана одна или несколько инструкций, разделенных знаком двоеточия (:).

Например, Z=X+Y : W=X^2+Y^2

3. Возможно использование переноса строк: одна инструкция может быть записана в нескольких строках, при этом допускается не более семи продолжений одной и той же строки (признаком продолжения является расположение знаков «Пробел» и «Знак подчеркивания» в конце разбиваемой строки).

4. Нельзя разбивать переносом строковую константу.

5. Для пояснения текста программ можно использовать комментарии.

Комментарии – это фрагменты пояснительного текста в программе (любой набор допустимых в данном языке символов, которые не являются программным кодом и поэтому не компилируются). Комментарии могут быть расположены в любом месте программы. Признаком комментария является расположенный перед ним знак апострофа «‘» или для строки комментария – ключевое слово Rem.

Комментарии позволяют выполнить две функции: делают программу легко читаемой, поясняя смысл ее инструкций, и отключают фрагменты программы при ее отладке.

Например,

```
‘Заголовок процедуры общего вида с именем Prim1
SUB Prim1()
‘Объявление переменной C целого типа
Dim C As Integer
```

Программирование алгоритмов линейной структуры. Инструкции для реализации алгоритмов линейной структуры

Инструкции присваивания

Инструкция вычисляет значение выражения и присваивает это значение переменной или константе.

Формат записи инструкции представлен ниже.

[Let] <Имя переменной> (или <Имя константы>) = <Выражение>

где

<Имя переменной>, <Имя константы>- конструкции, построенные по правилам VBA.

Выражение – выражение того же типа, что и составляющие его операнды.

В частных случаях выражение может состоять только из константы, переменной или функции.

Например:

X=Sqr(S)

B=125.5

V=A^3

S=X^2+5

Ввод и вывод информации

Для ввода и вывода информации в VBA используются две разновидности диалоговых окон (ДО):

1. окна ввода (InputBox);
2. окна сообщений (MsgBox).

Ввод информации осуществляется с помощью диалогового окна ввода значений, которое реализуется встроенной функцией **InputBox.** В окне ввода отображается поле для ввода значения переменной, в которое пользователь должен ввести определенное значение и нажать кнопку «ОК».

Таким образом функция InputBox осуществляет следующие действия:

- выводит на экран диалоговое окно, содержащее заголовок, зону сообщения, поле ввода, значение по умолчанию;
- устанавливает режим ожидания ввода текста пользователем и нажатия кнопки;
- возвращает значение типа String, содержащее текст, введенный в поле.

Формат записи функции Inputbox:

Inputbox(сообщение[, заголовок окна][,поле ввода][,значение по умолчанию][,…])

где

сообщение – строковое выражение, отображаемое как текст сообщения в диалоговом окне. Сообщение может состоять из нескольких строк. Для их разделения допускается использование символа возврата каретки(Chr(13)), символа перевода строки(Chr(10)) или комбинации этих символов (Chr(13)) &

(Chr(10));

заголовок окна – строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот аргумент опущен, в строку заголовка помещается имя приложения;

поле ввода – вводимое строковое выражение (строка знаков).

Помимо указанных аргументов в формате функции InputBox могут использоваться параметры [,…], регулирующие положение диалогового окна на экране, и имя файла справки о данном окне.

Ниже приведен пример записи инструкции присваивания и задания переменной с значения, возвращаемого функцией InputBox, открывающей диалоговое окно для ввода *(рис. 15)*.

```
c = InputBox("Введите температуру в градусах по шкале Цельсия(C):", "конвертация градусов Цельсия")
```

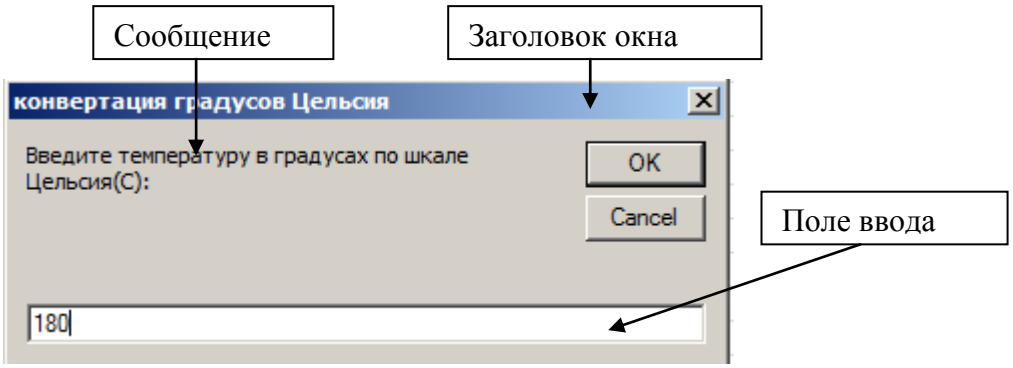


Рис. 15. Диалоговое окно ввода

На рисунке в окне текст «конвертация градусов Цельсия» расположен в строке заголовка окна, в зоне сообщения расположен текст «Введите температуру в градусах по шкале Цельсия(C):», а в поле ввода – число 180.

Вывод информации осуществляется в диалоговые окна сообщений, реализуемые инструкцией MsgBox или функцией MsgBox().

Формат записи инструкции MsgBox:

MsgBox строковое выражение_1 [, числовое выражение или имена констант][, строковое выражение_2][,…]

Формат записи функции MsgBox():

MsgBox (строковое выражение_1 [, числовое выражение или имена констант][, строковое выражение_2][,…])

где

строковое выражение_1 – сообщение, отображаемое в диалоговом окне;

числовое выражение – представляет тип используемого значка *(табл. 3)* или/и число и тип отображаемых кнопок *(табл. 4)*;

строковое выражение_2 – строковое выражение, отображаемое в строке заголовка диалогового окна.

Помимо указанных аргументов в формате MsgBox могут использоваться параметры, регулирующие положение диалогового кна на экране, и имя файла справки о данном окне [,…].

Таблица 3

Значения аргументов процедуры MsgBox, определяющих информационные значки в диалоговом окне сообщения		
Константа	Значение	Значок сообщения
VbCritical	16	
VbQuestion	32	
VbExclamation	48	
VbInformation	64	

По умолчанию инструкция MsgBox осуществляет вывод информации в диалоговое окно, имеющее одну кнопку «OK» (константа vbOKOnly) и устанавливает режим ожидания нажатия кнопки.

Таблица 4

Значения аргументов функции MsgBox, определяющих наличие кнопок в диалоговом окне сообщения		
Константа	Значение	Кнопки сообщения
vbOKOnly	0	кнопка «OK»
vbOKCancel	1	кнопки «OK» и «Отмена»
vbAbortRetryIgnore	2	кнопки «Прервать», «Повтор», «Пропустить»
vbYesNoCancel	3	кнопки «Да», «Нет», «Отмена»
vbYesNo	4	кнопки «Да», «Нет»
vbRetryCancel	5	кнопки «Повтор» и «Отмена»

Приведенная ниже инструкция вывода демонстрирует расположение параметров MsgBox в диалоговом окне вывода *(рис. 16)*. Знак операции & используется для слияния строк, а комбинация функций (Chr(10) & Chr(13)) – для перевода строки и возврата каретки, чтобы результаты вычислений были напечатаны в четыре строки. В диалоговом окне помимо кнопки «OK» расположен значок сообщения  (VbInformation).

```
MsgBox "x=" & Str(x) & Chr(10) & Chr(13) & _
"y=" & Str(y) & Chr(10) & Chr(13) & _
"r=" & Str(r) & Chr(10) & Chr(13) & _
"fi=" & Str(fi), vbInformation, "Координаты точки"
```

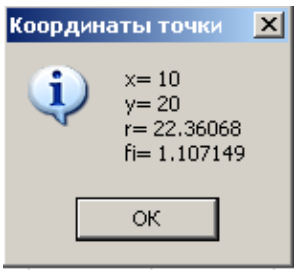


Рис. 16. Диалоговое окно вывода инструкции MsgBox

Если для вывода информации используется функция MsgBox, то в этом случае предполагается использовать в окне диалога несколько различных кнопок. Значение (тип – Integer), возвращаемое функцией MsgBox, зависит от того какая кнопка в окне сообщения была нажата *(табл. 5)*.

Ниже приведено обращение к функции MsgBox, которая реализует окно сообщения с вопросом о продолжении вычислений. В диалоговом окне *(рис. 17)* присутствует заголовок – «Ответьте на вопрос» и размещены две кнопки («Да», «Нет»). Значение, возвращаемое функцией MsgBox, будет присвоено переменной otv (тип – Integer).

```
Dim otv As Integer
otv = MsgBox("Будете продолжать вычисления?", vbInformation + vbYesNo, "Ответьте на вопрос")
```

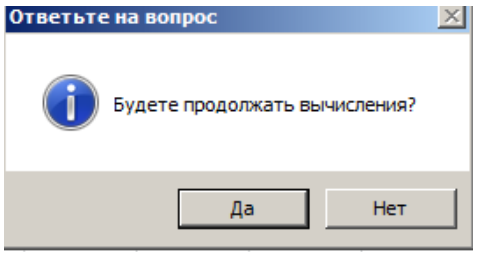


Рис. 17. Диалоговое окно вывода функции MsgBox

Если будет нажата кнопка «Да», то переменной otv будет присвоено значение 6 (константа vbYes), если кнопка «Нет», то переменной otv будет присвоено значение 7 (константа vbNo) согласно *табл. 5*.

Таблица 5

Перечень констант и значений, возвращаемых функцией MsgBox		
Константа	Значение	Нажатая кнопка
vbOK	1	«OK»
vbCancel	2	«Отмена»
vbAbort	3	«Прервать»
vbRetry	4	«Повтор»
vbIgnore	5	«Пропустить»
vbYes	6	«Да»
vbNo	7	«Нет»

Примеры программ с алгоритмом линейной структуры

1. Даны величины X и Y. Вычислить Z=f(T,P).

$T = \frac{x+y}{y}$; $P = Sin(x)$; Z = P * T, при y ≠ 0

Блок-схема алгоритма примера приведена ниже *(рис. 18)*.

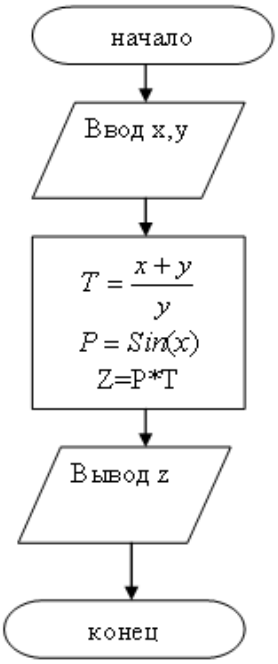


Рис. 18. Блок-схема алгоритма линейной структуры

Текст программы для примера приведен ниже *(рис. 19)*.

```
Option Explicit
Sub PRIM1()
Dim t As Single, p As Single, z As Single, X As Single, y As Single
'Ввод с клавиатуры значения переменной x
X = Val(InputBox("Введите значение x"))
'Ввод с клавиатуры значения переменной y
y = Val(InputBox("Введите значение y"))
'Вычисление значения переменной t
t = (X + y) / y
'Вычисление значения переменной p
p = Sin(X)
'Вычисление значения переменной z в формате с фиксированной точкой
z = Format(t * p, "FIXED")
'Вывод результата в диалоговое окно
MsgBox "Результат вычислений", vbInformation, "z=" & z
'Конец кода программы
End Sub
```

Рис. 19. Текст программы для примера 1

2. Рассчитать чистую потребность (материал / комплектующее) по формуле, приведенной ниже.

$$Ч = B - Ост_н + Ост_К$$

где

$Ч$ – чистая потребность;

B – валовая потребность;

$Ост_н$ – остаток в начале планово-учетного периода;

$Ост_К$ – остаток в конце планово-учетного периода.

Блок-схема алгоритма примера приведена ниже *(рис. 20)*.

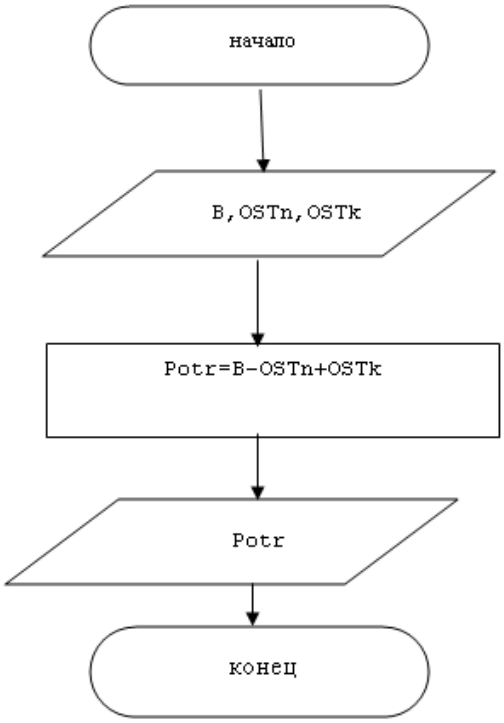


Рис. 20. Блок-схема алгоритма линейной структуры

Текст программы для примера приведен ниже *(рис. 21)*.

```
Private Sub MPOTR()
Dim B As Single, OSTn As Single, OSTk As Single, Potr As Single
B = Val(InputBox("Валовая потребность", "Материал / Комплектующее"))
OSTn = Val(InputBox("Остаток в начале планово-учетного периода", "Материал / Комплектующее"))
OSTk = Val(InputBox("Остаток в конце планово-учетного периода", "Материал / Комплектующее"))
Potr = B - OSTn + OSTk
MsgBox "Чистая потребность " & Str(Potr), , "Материал / Комплектующее"
End Sub
```

Рис. 21. Текст программы для примера 2

Программирование алгоритмов разветвляющейся структуры. Инструкции для реализации алгоритмов разветвляющейся структуры

Порядок выполнения инструкций в программе с разветвляющимся алгоритмом зависит от условий, которые проверяются по ходу выполнения программы. Вследствие этого программа реализуется по одному из нескольких заранее предусмотренных алгоритмом (возможных) направлений. Каждое отдельное направление называется ветвью вычислений.

Разветвляющиеся структуры можно реализовать в программах с помощью инструкций условного перехода (If ... Then ... Else … End If) или выбора по значению (Select Case ... EndSelect).

Порядок выполнения инструкций программы также можно изменить с помощью инструкции GOTO, называемой безусловным переходом. Инструкция безусловного перехода используется в тех случаях, когда необходимо выполнить не следующую по порядку инструкцию в программе, а помеченную меткой инструкцию.

Формат записи безусловного перехода:

Goto < метка>

Примечание. Чаще всего оператор Goto используется в инструкции If, если необходим безусловный переход к помеченной инструкции.

Инструкция If как способ реализации алгоритмов разветвляющейся структуры

В VBA имеются две формы записи инструкции условного перехода: полная и краткая.

If(если), Then(то), Else(иначе) – ключевые(служебные) слова.

Условие представляет собой выражение логического типа.

Формат записи краткой формы инструкции условного перехода в одну строку:

If <условие> Then <инструкции>

Формат записи краткой формы инструкции условного перехода в несколько строк:

If <условие> Then

...<инструкция1>

...<инструкция2>

…

End If

Когда условие истинно (т. е. результатом условия является True), то выполняется инструкция, или группа инструкций, следующая за ключевым словом «Then».

Полная форма инструкции условного перехода, который имеет две альтернативные ветви вычислений.

Формат записи полной формы инструкции условного перехода в одну строку:

If <условие> Then <инструкция1> Else <инструкция2>

Формат записи полной формы инструкции условного перехода в несколько строк:

If <условие> Then
...<инструкция1>
…
Else
...<инструкция2>
…
End If

Оператор If работает следующим образом: если условие истинно (имеет значение True), то выполняется инструкция1 (первая ветвь), в противном случае выполняется инструкция2 (вторая ветвь). Обе ветви могут содержать не одну, а несколько инструкций.

Для организации разветвлений по трем ветвям вычислений и более можно использовать несколько конструкций If, вложенных друг в друга. При этом каждое Else соответствует непосредственно предшествующему ему Then.

Формат записи инструкции условного перехода с вложенными структурами:

If <условие> Then
<инструкции>
ElseIf <условие> Then
<инструкции>
ElseIf <условие> Then
…..
…..
Else
<инструкции>
End If

Примечание. Следует избегать большой вложенности условных конструкций.

Инструкция выбора Select Case … End Select

Инструкция выбора позволяет выбрать ветвь вычислений из произвольного числа имеющихся вариантов, то есть организовать разветвления на произвольное число направлений. Select Case применяется в тех случаях, когда одна величина участвует в нескольких логических сравнениях и определяет, какой блок инструкций будет выполняться.

Формат записи инструкции выбора Select Case:

Select Case <выражение – селектор>
Case <список выражений 1> ‘вариант 1
<инструкция(и) 1>
…….
Case <список выражений n> ‘вариант n
<инструкция(и) n>

[Case Else
[<инструкция(и) Else>]]
End Select

Инструкция выбора состоит из выражения, называемого селектором, списков выражений-вариантов, начинающихся словом Case, и необязательной ветви Else, имеющей то же значение, что и в условной инструкции If. Каждый вариант представляет список выражений и инструкцию или группу инструкций, следующих за списком.

Список выражений может включать:

- константы;
- интервал значений;
- функцию Is.

Инструкции Else (необязательная часть) выполняются в том случае, если значение выражения (селектора) не совпадает ни с одним из предложений значений выражений, указанных после Case.

Инструкция выполняется следующим образом:

1. вычисляется значение выражения-селектора;
2. просматриваются последовательно выражения-варианты, константы и значения из диапазонов соответствующего списка на предмет совпадения значений со значением выражения-селектора;
3. если для очередного варианта получено совпадение, то выполняется инструкция или группа инструкций данного варианта, затем выполнение инструкции выбора заканчивается и осуществляется переход к следующей за End Select инструкции;
4. если не нашлось ни одного совпадения, то выполняется инструкция после слова Else (при его наличии), иначе осуществляется переход к инструкции, следующей за End Select.

Примеры программ с алгоритмом разветвляющейся структуры

1. Даны два числа A и B. Если A-B>0 вычислить $\sqrt{A-B}$, если нет вычислить $(A-B)^2$. Полученные результаты вывести.

Блок-схема алгоритма примера приведена ниже (*рис. 22*).

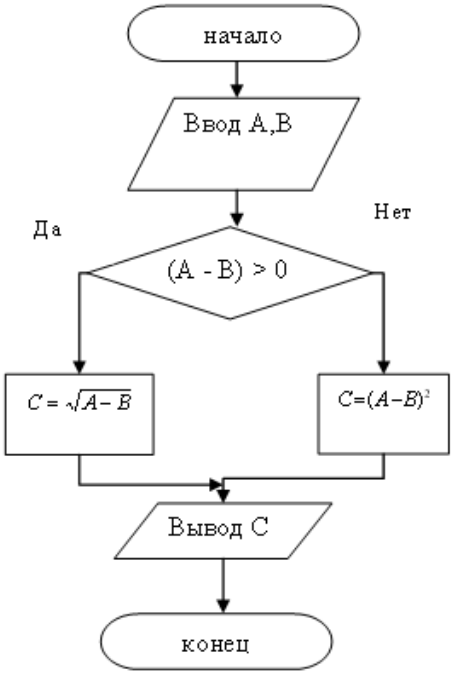


Рис. 22. Блок-схема алгоритма примера 1

Текст программы для примера приведен ниже *(рис. 23)*.

```
Option Explicit
Sub PRIM2()
'Объявление переменных A, B, C вещественного типа
Dim A As Single, B As Single, C As Single
'Ввод с клавиатуры значения переменной A
A = Val(InputBox("Введите значение A"))
'Ввод с клавиатуры значения переменной B
B = Val(InputBox("Введите значение B"))
'Вычисление значения переменной C в зависимости от выполнения условия (A - B) > 0
If (A - B) > 0 Then
C = Sqr(A - B)
Else
C = (A - B) ^ 2
End If
'Вывод результата в диалоговое окно
MsgBox "A= " & A & " " & "B= " & B, vbInformation, "C= " & Format(C, "FIXED")
'Конец кода программы
End Sub
```

Рис. 23. Текст программы для примера 1

В программе использована полная форма инструкции условного перехода с записью в несколько строк.

2. Определить является ли вводимое целое число четным.

Блок-схема алгоритма примера приведена ниже (*рис. 24*).

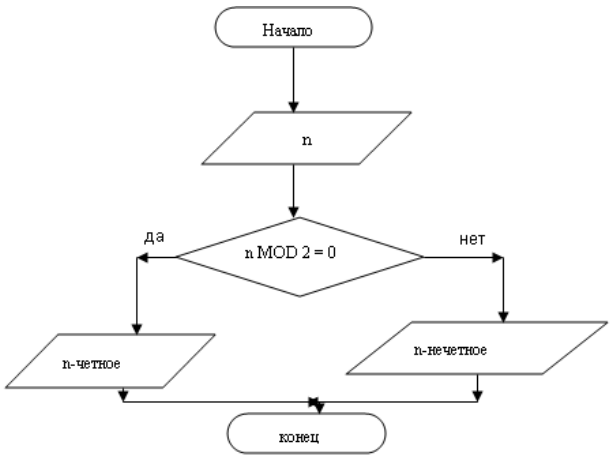


Рис. 24. Блок-схема алгоритма примера 2

Текст программы для примера приведен ниже *(рис. 25)*.

```
Private Sub Prov()
Dim n As Integer
n = InputBox("Введите число", "Определить является ли целое число четным")
If n Mod 2 = 0 Then MsgBox "четное число", , n Else MsgBox "нечетное число", , n
End Sub
```

Рис. 25. Текст программы для примера 2

В программе использована полная форма инструкции условного перехода с записью в одну строку.

3. Даны значения двух параметров c и t. Вычислить значение параметра F. Зависимость параметра F от параметров c и t приведена ниже.

$$F = \begin{cases} \frac{t^2 + 5}{6} + 4t; t = 1 \\ \frac{c + 7}{2t} \ln(t); t > 1 \\ \frac{t}{2} + \frac{c + t^2}{3}; t < 1 \end{cases}$$

Блок-схема алгоритма примера приведена ниже (*рис. 26*).

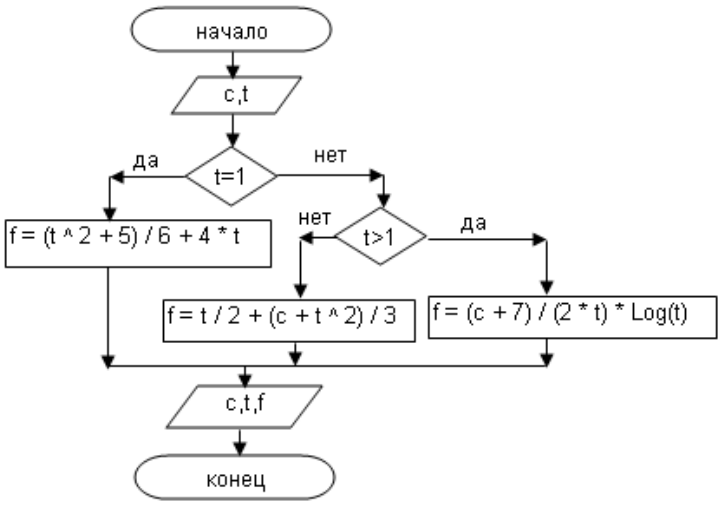


Рис. 26. Блок-схема алгоритма примера 3

Текст программы для примера приведен ниже *(рис. 27)*.

```
Sub fun()
Dim c As Single, t As Single, f As Single
c = InputBox("c=")
t = InputBox("t=")
If t = 1 Then
f = (t ^ 2 + 5) / 6 + 4 * t
ElseIf t > 1 Then
f = (c + 7) / (2 * t) * Log(t)
Else
f = t / 2 + (c + t ^ 2) / 3
End If
MsgBox "при" & Chr(10) & Chr(13) & "c=" & c _
& Chr(10) & Chr(13) & "t=" & t, , "f=" & Format(f, "fixed")
End Sub
```

Рис. 27. Текст программы для примера 3

В программе использована полная форма инструкции условного перехода с записью в несколько строк с вложенной структурой.

4. Вычислить площадь одной из трех фигур: квадрат, трапеция, круг.

Блок-схема алгоритма примера приведена ниже *(рис. 28)*.

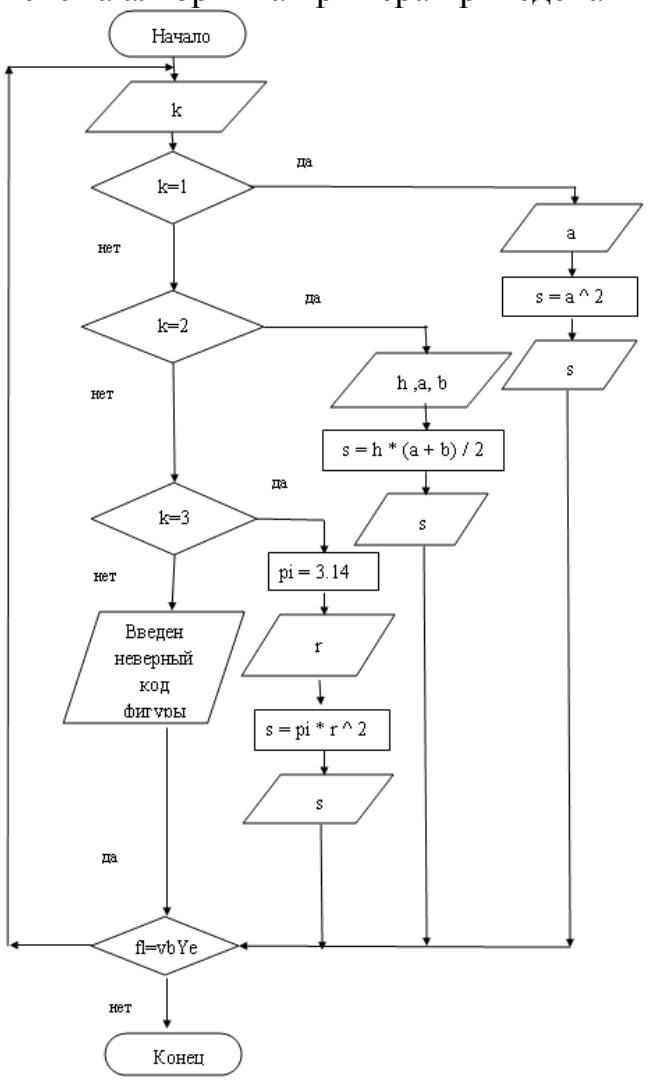


Рис. 28. Блок-схема алгоритма примера 4

Текст программы для примера приведен ниже *(рис. 29)*.

```
Private Sub P1F()
Dim k As Integer, fl, s As Single, _
a As Single, b As Single, h As Single, r As Single
'Написать программу для вычисления площади одной
'из трех фигур:квадрат, трапеция, круг.
'Расчет площадей фигур
1
k = InputBox("Расчет площадей фигур:" _
& Chr(10) & Chr(13) & "1-квадрат" _
& Chr(10) & Chr(13) & "2-трапеция" _
& Chr(10) & Chr(13) & "3-круг", "Введите код фигуры")
Select Case k
Case 1
'Определение площади квадрата
a = InputBox("Введите сторону квадрата ")
s = a ^ 2
MsgBox "Площадь квадрата: " & s
Case 2
'Определение площади трапеции
a = InputBox("Введите 1 основание трапеции ")
b = InputBox("Введите 2 основание трапеции ")
h = InputBox("Введите высоту трапеции ")
s = h * (a + b) / 2
MsgBox "Площадь трапеции: " & s
Case 3
'Определение площади круга
Const pi = 3.14
r = InputBox("Введите радиус окружности ")
s = pi * r ^ 2
MsgBox "площадь круга: " & s
Case Else
MsgBox "Введен неверный код фигуры", vbCritical
End Select
fl = MsgBox("Вы хотите продолжить работу ?", vbYesNo)
If fl = vbYes Then GoTo 1
End Sub
```

Рис. 29. Текст программы для примера 4

В программе использована инструкция выбора Select Case, в которой выражение-селектор представлено переменной k. В зависимости от значения этой переменной (для расчетов допустимы значения 1, 2, 3) площадь фигуры вычисляется по соответствующим формулам. Для возможности многократного выполнения программы использована функция MsgBox, которая реализует окно сообщения с вопросом о продолжении вычислений. В диалоговом окне присутствует вопрос – «Вы хотите продолжить работу?» и размещены две кнопки («Да», «Нет»). Значение, возвращаемое функцией MsgBox, будет присвоено переменной fl (тип – Integer). Для проверки значения переменной fl использована краткая форма инструкции условного перехода в одну строку, в которой после ключевого слова Then использована инструкция безусловного перехода GOTO. Если нажата кнопка «Да», то необходимо выполнить не следующую по порядку инструкцию в программе, а инструкцию, помеченную меткой 1 (ввод кода фигуры).

Программирование алгоритмов циклической структуры. Инструкции для реализации алгоритмов циклической структуры

Для организации циклов в VBA, т. е. многократного выполнения одной или нескольких инструкций, можно использовать две основные группы инструкций: For…Next (циклы с фиксированным числом повторений) и Do...Loop (циклы с условием и неизвестным числом повторений).

Существуют разновидности циклов с фиксированным числом повторений For...Next (For…Next и For Each…Next) и разновидности циклов с условием и неизвестным числом повторений While…Wend, Do...Loop (Do While...Loop, Do Until ...Loop, Do...Loop While, Do...Loop Until, Do...Exit Do…Loop), которые различаются типом проверяемого условия.

Инструкция цикла For…Next

В случаях, когда заранее известно число повторений некоторого процесса, пользуются инструкцией цикла с параметром.

Для этого типа цикла необходимо задать границы (начальное и конечное значения счетчика) в пределах которых будет изменяться переменная цикла.

Формат записи инструкции цикла For…Next:

For <параметр цикла> = <S1> **To** <S2> [**Step** <S3>]

<инструкции>

[Exit For]

<инструкции>

Next <параметр цикла>

где

For…Next – заголовок цикла;

<параметр цикла> – числовая переменная, определяемая в заголовке цикла;

<инструкции> – тело цикла;

S1 и S2 – выражения, определяющие начальное и конечное значения параметра цикла;

S3 – выражение, определяющее шаг приращения значения выражения S1 до значения S2.

Значение переменной увеличивается или уменьшается с каждым повторением цикла. Если в конструкции цикла отсутствует Step, то приращение равно 1 (по умолчанию).

При явном указании шага и для S3>0 должно выполняться условие – S1<=S2, в случае, если S3<0 должно выполняться условие – S1>=S2.

Тело цикла может состоять из одного или нескольких инструкций.

Работа инструкции For осуществляется таким образом, что тело цикла выполняется последовательно с каждым значением параметра цикла от начального до конечного значения. Когда значение параметра цикла превзойдет конечное значение <S2>, цикл завершится, и будет выполняться инструкция, следующая за конструкцией For.

Инструкция цикла For Each…Next

Для обработки группы однородных объектов или массивов применяется следующая конструкция цикла.

Формат записи инструкции цикла For Each…Next:

For Each элемент **In** группа (имя группы однородных объектов)

<инструкции>

[Exit For]

<инструкции>

Next элемент

Повторяет выполнение инструкций для каждого элемента семейства.

Для досрочного выхода (до достижения параметром цикла конечного значения) из инструкции цикла при выполнении некоторого дополнительного условия, в тело цикла надо добавить **Exit For**.

Инструкции цикла Do...Loop

В итерационных циклах с условием и неизвестным числом повторений необходимо задавать начальное значение параметра цикла до входа в цикл и изменять его значение в теле цикла.

В VBA применяются следующие способы организации итерационных циклов: «циклы с предусловием», «циклы с постусловием».

Циклы с предусловием

Необходимо отметить, что в этих циклах условие проверяется до того, как выполняется инструкция или группа инструкций. Тело цикла выполняется до тех пор, пока соблюдается (не соблюдается) какое-либо заданное условие. Минимальное число повторений этого цикла – 0.

Инструкция цикла Do While <условие>…Loop

Формат записи инструкции цикла:

Do While <условие>

<инструкции>

[Exit Do]

<инструкции>

Loop

Цикл выполняется до тех пор, пока <условие> истинно.

<условие> – логическое выражение, принимающее значение True (истина) или False (ложь).

Выполнение цикла с предусловием состоит из следующих шагов.

1. Проверяется <условие>. Если <условие> имеет значение True, то выполняются инструкции, составляющие тело цикла (до Loop) столько раз, пока<условие> не примет значение False.

2. Как только <условие> получит значение False, осуществится выход из цикла, т. е. переход к инструкции, следующей за Loop.

Необязательная инструкция **Exit Do** предназначена для прекращения цикла.

Инструкция цикла Do Until <условие>…Loop

Формат записи инструкции цикла:

Do Until <условие>

<инструкции>

[Exit Do]

<инструкции>

Loop

Ключевое слово Until указывает на то, что цикл выполняется до тех пор, пока условие не станет истинным.

<условие> – логическое выражение, принимающее значение True (истина) или False (ложь).

Выполнение цикла с предусловием состоит из следующих шагов.

1. Проверяется <условие>. Если <условие> имеет значение False, то выполняются инструкции, составляющие тело цикла (до Loop) столько раз, пока<условие> не примет значение True.

2. Как только <условие> получит значение True, осуществится выход из цикла, т. е. переход к инструкции, следующей за Loop.

Необязательная инструкция **Exit Do** предназначена для прекращения цикла.

Циклы с постусловием

Цикл выполняется хотя бы один раз, так как проверка <условия> выполняется в конце цикла. Тело цикла выполняется до тех пор, пока соблюдается (не соблюдается) какое-либо заданное условие.

Инструкция цикла Do…Loop While <условие>

Формат записи инструкции цикла:

Do

<инструкции>

[Exit Do]

<инструкции>

Loop While <условие>

Цикл выполняется до тех пор, пока <условие> истинно.

<условие> – логическое выражение, принимающее значение True (истина) или False (ложь).

Выполнение цикла с постусловием состоит из следующих шагов.

1. Выполняются инструкции, составляющие тело цикла (до Loop).

2. Проверяется <условие>. Если <условие> имеет значение True, то вновь выполняются инструкции, составляющие тело цикла (до Loop) столько раз, пока<условие> не примет значение False.

3. Как только <условие> получит значение False, осуществится выход из цикла, т. е. переход к инструкции, следующей за Loop.

Необязательная инструкция **Exit Do** предназначена для прекращения цикла.

Инструкция цикла Do…Loop Until <условие>

Формат записи инструкции цикла:

Do

<инструкции>

[Exit Do]

<инструкции>

Loop Until <Условие >

Ключевое слово Until указывает на то, что цикл выполняется до тех пор, пока условие не станет истинным.

<условие> – логическое выражение, принимающее значение True (истина) или False (ложь).

Выполнение цикла с предусловием состоит из следующих шагов.

1. Выполняются инструкции, составляющие тело цикла (до Loop).

2. Проверяется <условие>. Если <условие> имеет значение False, то вновь выполняются инструкции, составляющие тело цикла (до Loop) столько раз, пока<условие> не примет значение True.

3. Как только <условие> получит значение True, осуществится выход из цикла, т. е. переход к инструкции, следующей за Loop.

Необязательная инструкция **Exit Do** предназначена для прекращения цикла.

Инструкция цикла While…Wend.

Формат записи инструкции цикла:

While <условие>

<инструкции>

Wend

Цикл выполняется до тех пор, пока <условие> истинно.

<условие> – логическое выражение, принимающее значение True (истина) или False (ложь).

Выполнение цикла с предусловием состоит из следующих шагов.

1. Проверяется <условие>. Если <условие> имеет значение True, то выполняются инструкции, составляющие тело цикла (до **Wend**) столько раз, пока<условие> не примет значение False.

2. Как только <условие> получит значение False, осуществится выход из цикла, т. е. переход к инструкции, следующей за **Wend**.

Примеры программ с алгоритмом циклической структуры

1. Произвести табуляцию функции Y=F(x) для x=0;0.1;0.2;…;1.3;1.4

$F(x) = \sin(x^2 + 2x)$

Блок-схема алгоритма примера приведена ниже *(рис. 30)*.

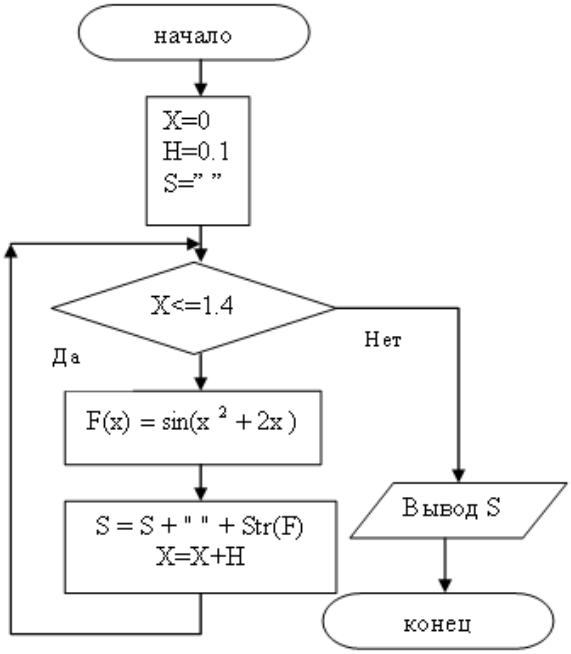


Рис. 30. Блок-схема алгоритма примера 1

Текст программы для примера приведен ниже *(рис. 31)*.

```
Option Explicit
Sub PRIM3()
'Вычисление значения функции F =Sin(X ^ 2 + 2 * X)
'Объявление типов переменных X, H, F,S
'Объявление переменных X, H, F вещественного типа
Dim X As Single, H As Single, F As Single
'Объявление переменной S строкового типа
Dim S As String
'Задание начальных значений
X = 0
H = 0.1
S = " "
'Начало цикла с предусловием
Do While X <= 1.4
'Вычисление значений функции в цикле в формате с фиксированной точкой
F = Format(Sin(X ^ 2 + 2 * X), "FIXED")
'Суммирование значений функции для вывода в одну строку
S = S + " " + Str(F)
'Изменение параметра цикла на величину шага
X = X + H
'Окончание цикла
Loop
'Вывод результата в диалоговое окно
MsgBox "S= " & S, vbInformation, "Значения функции"
'Конец кода программы
End Sub
```

Рис. 31. Текст программы для примера 1

В программе использован цикл с предусловием
Do While <условие>…Loop.

2. Вычислить приближенное значение числа pi с помощью $1-\frac{1}{3}+\frac{1}{5}-\ldots$ ряда, который стремится к pi/4, пренебрегая членами ряда по абсолютной величине меньшими t=0,0001 (точность вычислений).

Блок-схема алгоритма примера приведена ниже *(рис. 32).*

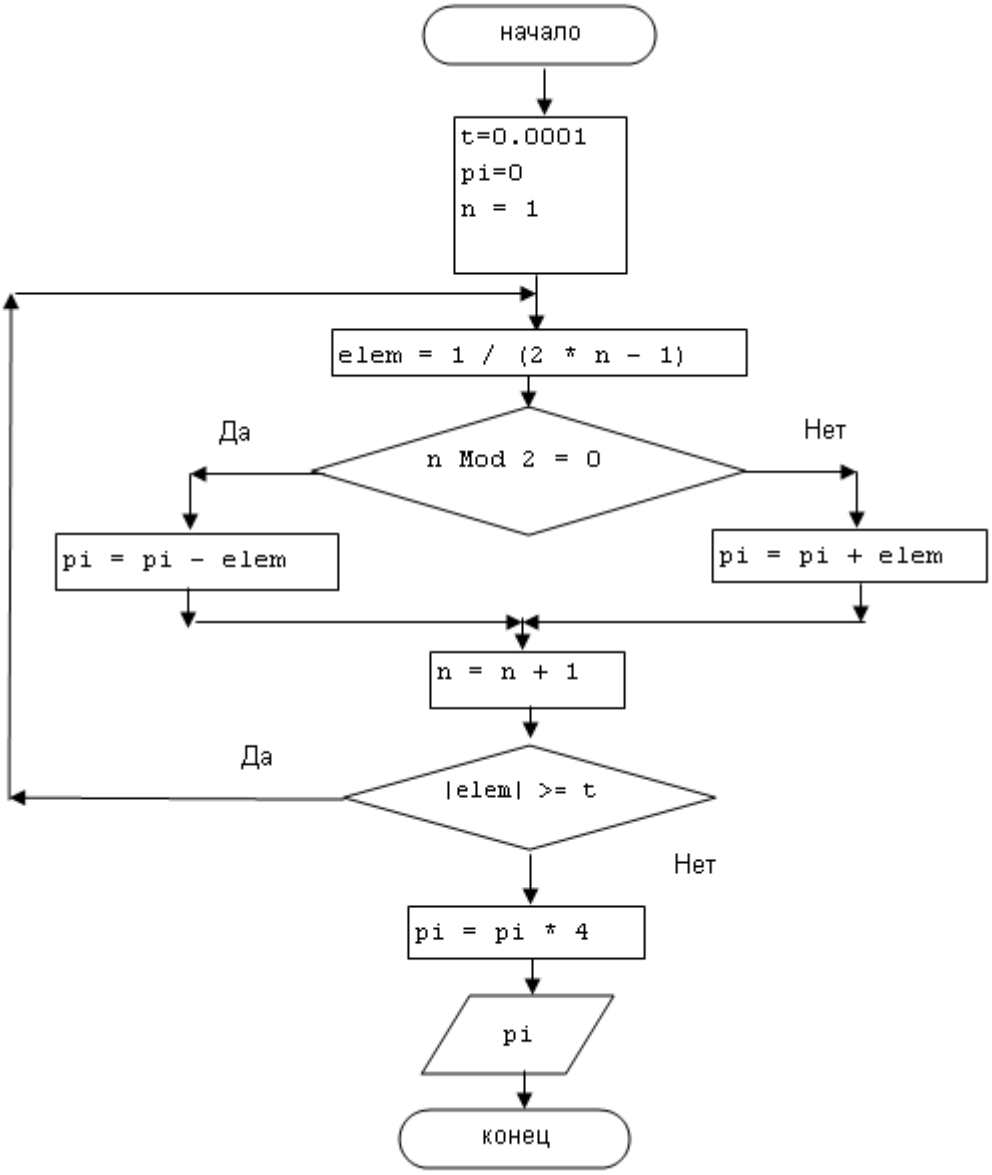


Рис. 32. Блок-схема алгоритма примера 2

Текст программы для примера приведен ниже *(рис. 33).*

```
Private Sub summa()
Dim t As Single, elem As Single, pi As Single, n As Integer
'Вычислить приближенное значение числа pi
'с помощью ряда
'      1     1
'1 - --- + --- - ...
'      3     5
'который стремится к pi/4
'пренебрегая членами ряда,по абсолютной величине
'меньшими t=0,0001 (точность вычислений)
t = 0.0001
pi = 0
n = 1
Do
elem = 1 / (2 * n - 1)
If n Mod 2 = 0 Then pi = pi - elem Else pi = pi + elem
n = n + 1
Loop While Abs(elem) >= t
pi = pi * 4
MsgBox "pi=" & pi, , "сумма " & n & " членов ряда"
End Sub
```

Рис. 33. Текст программы для примера 2

В программе использован цикл с постусловием

Do…Loop While <условие>.

3. Дана пирамида из n уровней шаров, основание которой представляет собой квадрат со стороной, состоящей из n шаров. Определить сколько шаров потребуется для строительства этой пирамиды.

Блок-схема алгоритма примера приведена ниже *(рис. 34)*.

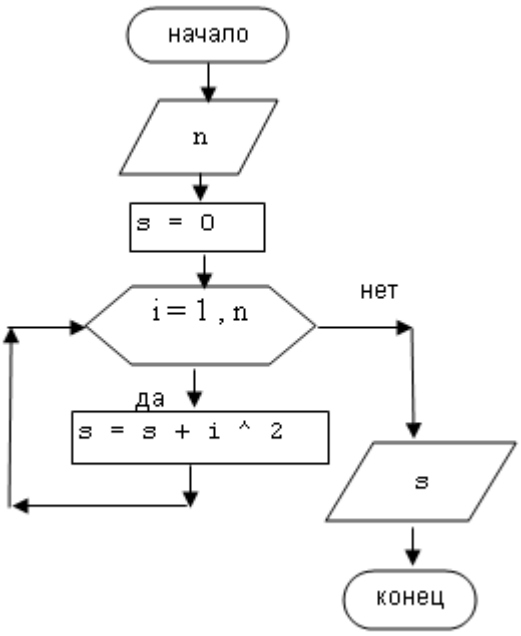


Рис. 34. Блок-схема алгоритма примера 3

Текст программы для примера приведен ниже *(рис. 35)*.

```
Private Sub Kol()
Dim n As Integer, i As Integer, s As Integer
'Дана пирамида из n уровней шаров,основание которой
'представляет собой квадрат со стороной,состоящей из
'n шаров.Определить сколько шаров потребуется для
'строительства этой пирамиды.
n = InputBox("Введите количество слоев шаров в пирамиде ")
s = 0
For i = 1 To n
s = s + i ^ 2
Next i
MsgBox "Для строительства пирамиды потребуется " & Str(s) & " шт."
End Sub
```

Рис. 35. Текст программы для примера 3

В программе использован цикл с фиксированным числом повторений For...Next. Значение переменной цикла увеличивается на 1 с каждым повторением цикла (в конструкции цикла отсутствует Step и приращение значения переменной равно 1 по умолчанию).

4. Оформить таблицу зависимости плотности от высоты для h=0-1000 м

Плотность воздуха убывает с высотой по закону:

$$p = p0 * e^{-h*z}$$

$$p0 = 1{,}29 \frac{кг}{м^3}$$

$$z = 1{,}25 * 10^{-4} \frac{1}{м}$$

Блок-схема алгоритма примера приведена ниже *(рис. 36)*.

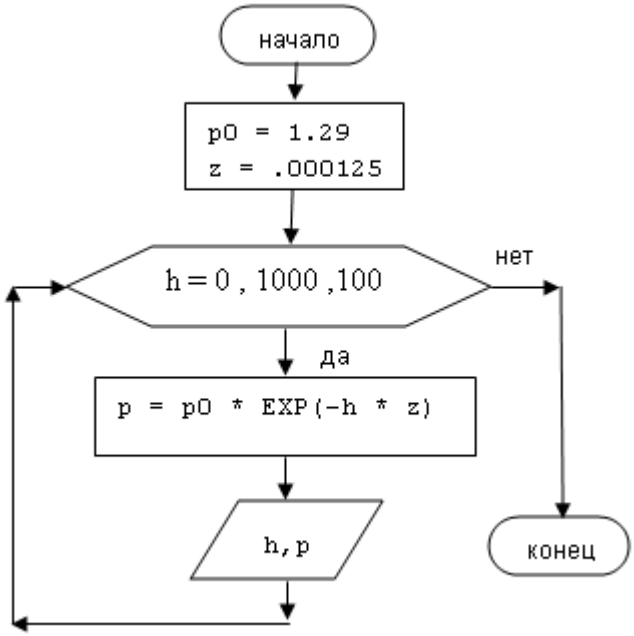


Рис. 36. Блок-схема алгоритма примера 4

Текст программы для примера приведен ниже *(рис. 37)*.

```
Private Sub P1()
Dim p As Single, h As Integer, t As String
'Плотность воздуха убывает с высотой по закону
'       -h*z
'p=p0*e
'             3
'p0=1,29 кг/м
'          -4
'z=1,25*10   1/м
'Оформить таблицу зависимости плотности от высоты для
'h=0-1000 м
Const p0 = 1.29, z = 0.000125
t = "h-ВЫСОТА(М), p-ПЛОТНОСТЬ ВОЗДУХА(КГ/М3)"
For h = 0 To 1000 Step 100
p = p0 * Exp(-h * z)
t = t & Chr(10) & Chr(13) & "h=" & Str(h) & "  p=" & Str(Format(p, "fixed"))
Next h
MsgBox t, , "ЗАВИСИМОСТЬ ПЛОТНОСТИ ВОЗДУХА ОТ ВЫСОТЫ "
End Sub
```

Рис. 37. Текст программы для примера 4

В программе использован цикл с фиксированным числом повторений For...Next. Значение переменной цикла увеличивается на 100 с каждым повторением цикла (в конструкции цикла присутствует ключевое слово Step и приращение значения переменной, равное 100).

Массивы

Понятие массива и его описание

Часто возникает необходимость решения задачи с большим, но конечным числом однотипных упорядоченных данных. В этом случае удобно описывать такой упорядоченный набор нумерованных компонент одним идентификатором (именем). Для обработки таких данных в языке VBA предусмотрен отдельный тип данных, называемый массивом.

Массив – это структурированный тип данных, представляющий собой набор однотипных переменных с одним именем, каждая из которых называется элементом массива и имеет свой номер (индекс/индексы).

Массивы могут быть: одномерные (для нумерации элементов используется один индекс), двумерные (для нумерации элементов используются два индекса: номер строки, номер столбца) и N-мерные. Число измерений массива может достигать 60.

Для указания компоненты массива можно использовать имя массива и порядковый номер нужной компоненты в этом массиве.

Массивы могут быть статические и динамические.

Статические массивы

Статическим называется массив с заранее известным количеством элементов.

Для объявления (описания) статического массива необходимо указать тип элементов массива и их количество. Тип элементов массива называется базовым.

Формат объявления (описания) одномерного статического массива представлен ниже.

Dim <идентификатор> ([<нижняя граница индекса ТО>] <верхняя граница индекса >) As <тип>

где

<идентификатор> – имя массива;

<нижняя граница индекса> – необязательный параметр, предназначенный для определения индекса первого элемента массива;

<верхняя граница индекса> – индекс последнего элемента массива;

<тип> – тип элементов массива.

При использовании в VBA массивов индексирование по умолчанию начинается с нуля, т.е. индекс 0 обозначает первый элемент массива, индекс 1 – второй и т. д.

Для задания по умолчанию нижней границы массива, равной 1, используется инструкция **Option Base 1**, которая ставится в начале модуля.

Формат объявления (описания) двумерного статического массива представлен ниже.

Dim <идентификатор> ([<нижняя граница индекса1 ТО>] <верхняя граница индекса1>,[<нижняя граница индекса 2 ТО>] <верхняя граница индекса 2>) As <тип>

где

<идентификатор> – имя массива;

<нижняя граница индекса1> – необязательный параметр, предназначенный для определения индекса1 первого элемента массива;

<нижняя граница индекса1> – необязательный параметр, предназначенный для определения индекса1 первого элемента массива;

<верхняя граница индекса1> – индекс1 последнего элемента массива;

<верхняя граница индекса2> – индекс2 последнего элемента массива;

<тип> – тип элементов массива.

Примеры описания статических массивов

Описание одномерного массива, состоящего из 11 вещественных чисел (одномерный массив с начальным значением индекса, равным 0):

```
Dim a(10) As Single
```

Описание одномерного символьного массива, состоящего из 3 элементов (интервал значений индекса указан явно):

```
Dim S(3 To 5) As String
```

Описание двумерного массива, состоящего из 15 элементов целых значений:

```
Dim kd(1 To 3, 1 To 5) As Byte
```

Описание двумерного массива, состоящего из 16*18 вещественных чисел (двумерный массив с начальными значениями индексов равными 0):

```
Dim R(15, 17) As Single
```

Динамические массивы

Динамическим называется массив, размерность которого определяется в ходе выполнения программы.

Формат объявления (описания) динамического массива представлен ниже.

Dim <идентификатор> () As <тип>

где

<идентификатор> – имя массива;

<тип> – тип элементов массива.

Размерность массива устанавливается и изменяется с помощью инструкции ReDim.

Формат инструкции ReDim для **одномерного** динамического массива представлен ниже.

ReDim <идентификатор> ([<нижняя граница индекса ТО>] <верхняя граница индекса >) As <тип>

где

<идентификатор> – имя массива;

<нижняя граница индекса> – необязательный параметр, предназначенный для определения индекса первого элемента массива;

<верхняя граница индекса> – индекс последнего элемента массива;

<тип> – тип элементов массива.

Формат инструкции ReDim для **двумерного** динамического массива представлен ниже.

ReDim <идентификатор> ([<нижняя граница индекса1 ТО>] <верхняя граница индекса1>,[<нижняя граница индекса 2 ТО>] <верхняя граница индекса 2>) As <тип>

где

<идентификатор> – имя массива;

<нижняя граница индекса1> – необязательный параметр, предназначенный для определения индекса1 первого элемента массива;

<нижняя граница индекса1> – необязательный параметр, предназначенный для определения индекса1 первого элемента массива;

<верхняя граница индекса1> – индекс1 последнего элемента массива;

<верхняя граница индекса2> – индекс2 последнего элемента массива;

<тип> – тип элементов массива.

Примеры описания динамических массивов

Например, объявим одномерный динамический массив целых чисел, размерность которого n определяется в ходе выполнения программы:

```
Dim a() As Integer
```

Вычислив необходимый размер одномерного массива, можно изменить размер массива с помощью **ReDim:**

```
ReDim a(1 To n) As Integer
```

Например, объявим двумерный динамический массив целых чисел, размерность которого n определяется в ходе выполнения программы:

```
Dim x() As Integer
```

Вычислив необходимые размеры двумерного массива, можно изменить размеры массива с помощью **ReDim:**

```
ReDim x(1 To n, 1 To m)
```

Для установки и изменения размерности массива без потери его содержимого применяется следующее описание массива.

Формат инструкции **ReDim Preserve** для **одномерного** динамического массива представлен ниже.

ReDim Preserve <идентификатор> ([<нижняя граница TO>] <верхняя граница>) As <тип>

Формат инструкции **ReDim Preserve** для **двумерного** динамического массива представлен ниже.

ReDim Preserve <идентификатор> ([<нижняя граница индекса1 TO>] <верхняя граница индекса1>,[<нижняя граница индекса 2 TO>] <верхняя граница индекса 2>) As <тип>

Для определения параметров динамического массива используются следующие функции.

Функция **Array(<список аргументов>) – с**оздаёт массив типа Variant.

Список аргументов представляет разделенный запятыми список значений, присваиваемых элементам массива.

Например, определим массив дней недели:

```
Dim День As Variant
День = Array("Пн", "Вт", "Ср", "Чт", "Пт", "Сб")
```

Функция **IsArray(<имя переменной>)** используется для проверки значений переменных типа **Variant**, содержащих массивы.

Она возвращает значение **True**, если переменная содержит массив, и значение **False**, если переменная не содержит массив (не является массивом).

Действия над элементами массивами

После объявления массивов в разделе описания можно обработать каждый элемент массива в исполняемой части программы, указав имя массива и индекс элемента в круглых скобках. Например, пятый элемент одномерного массива с именем A записывается как A(5).

Индексированные элементы массива являются переменными базового типа и могут быть использованы так же, как простые переменные. В частности, им можно присваивать любые значения, соответствующие их типу.

На практике обработку массивов целесообразнее выполнять с помощью специальных инструкций цикла таким образом, чтобы параметр цикла использовался для индексирования массива. При изменении параметра цикла от нижнего значения до верхнего, обрабатываются один за другим все элементы

массива.

Типовые алгоритмы обработки массивов

Рассмотрим основные действия над одномерными статическими массивами и их элементами на примере двух массивов.

Объявление массивов A, D и переменной для индексирования массивов.

Для задания по умолчанию нижней границы массива, равной 1, используется инструкцией Option Base 1:

```
Option Base 1

Dim A(10) As Single, D(10) As Single

Dim I As Integer
```

Ввод массива:

```
For I = 1 To 10
A(I) = InputBox("I=" + Str(I), " Ввод значений массива A")
Next I
```

Вывод массива:

```
S = ""
For I = 1 To 10
S = S & A(I) & " "
Next I
MsgBox S, , "Массив A из 10 элементов"
```

Копирование массива:

```
For I = 0 To 10
D(I) = A(I)
Next I
```

Обнуление массива:

```
For I = 1 To 10
A(I) = 0
Next I
```

Перестановка элементов массива

Перестановка значений массива осуществляется с помощью дополнительной переменной того же типа, что и базовый тип массива.

Пусть требуется поменять местами значения элементов k и n массива A, тогда соответствующий фрагмент программы с использованием вспомогательной переменной X выглядит следующим образом:

```
X = A(k): A(k) = A(n): A(n) = X
```

Поиск и обработка значений элементов массива

1. Пусть требуется определить положительность элементов массива A и вычислить сумму положительных и сумму неположительных элементов массива A. Фрагмент программы будет иметь следующий вид:

```
so1 = 0
so2 = 0
For i = 1 To k
If T(i) > 0 Then so1 = so1 + o(i) Else so2 = so2 + o(i)
Next i
```

2. Для получения полного перечня комбинаций цифр и номеров воспользуемся инструкцией цикла For Each…Next, которая предназначена для обработки группы однородных объектов или массивов. Массивы цифр и номеров представляют собой динамические массивы типа Variant, для определения их параметров используется функция **Array.** Данная функция создаёт массивы, представляя разделенный запятыми список значений, присваиваемых элементам массива.

Программа, записанная на VBA, будет иметь следующий вид:

```
Sub EF()
Dim s As String
Dim numbers As Variant
numbers = Array(1, 3, 7)
Dim letters As Variant
letters = Array("a", "b", "c")
s = ""
For Each letter In letters
For Each Number In numbers
 s = s & Number & letter & "  "
Next Number
Next letter
MsgBox s
End Sub
```

Будет получен следующий результат:

```
1a 3a 7a 1b 3b 7b 1c 3c 7c
```

Двумерные массивы

Ввод данных в двумерный массив осуществляется так же, как и одномерный. Отличие состоит только в том, что ссылка на элемент массива осуществляется указанием номера строки и номера столбца, на пересечении которых находится элемент.

При обработке двумерного массива из-за наличия двух индексов используется структура, называемая вложенным циклом, представляющая собой цикл в цикле: цикл по столбцам в цикле по строкам или наоборот.

Примеры программ с алгоритмом циклической структуры для обработки массивов

1. Дан массив A(1 TO 10). Найти индексы наибольшего и наименьшего элементов и их значения.

Блок-схема алгоритма примера приведена ниже *(рис. 38).*

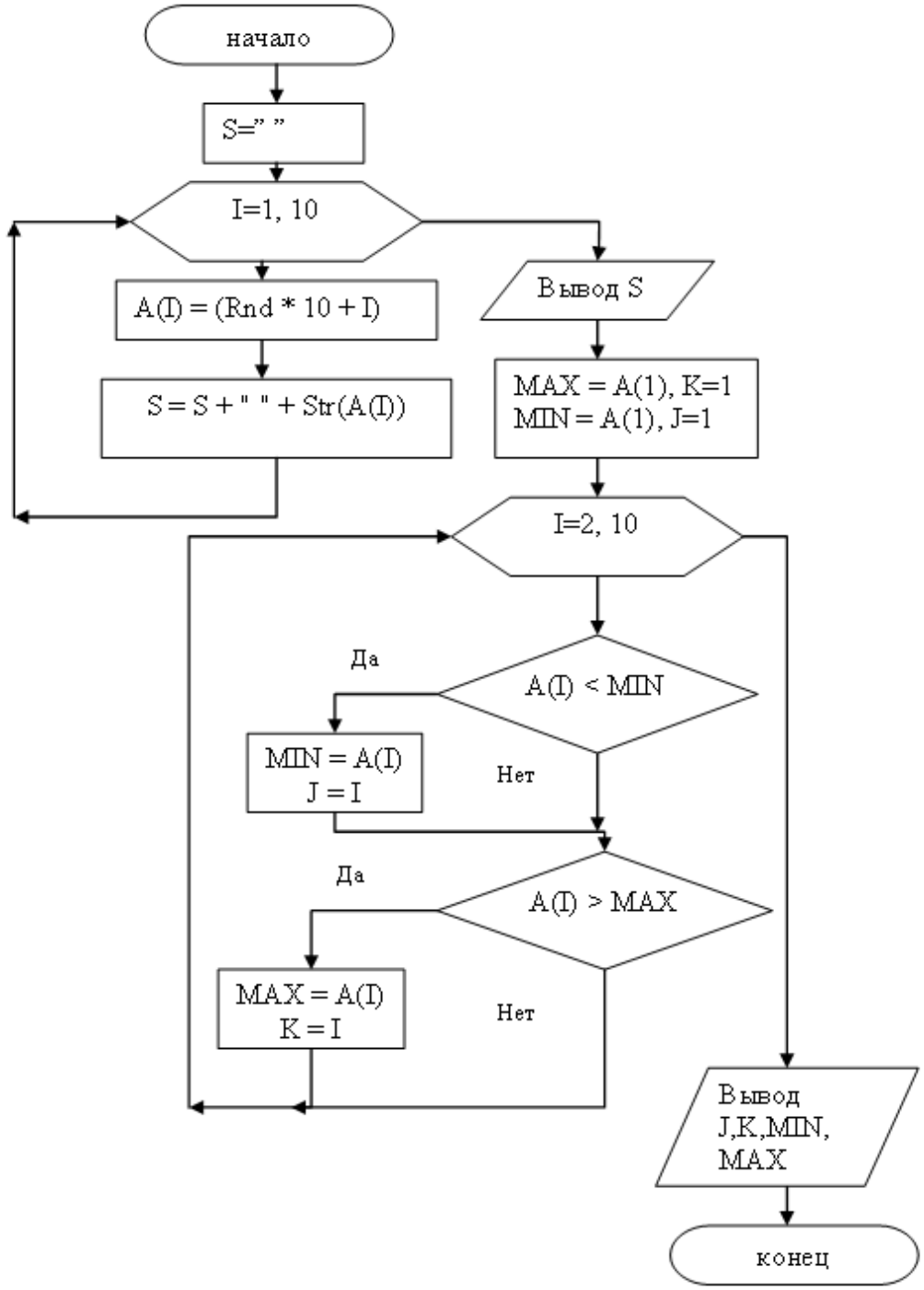


Рис. 38. Блок-схема алгоритма примера 1

Текст программы для примера приведен ниже *(рис. 39).*

```
Option Explicit
Sub PRIM4_1()
'Объявление переменных I, J, K целого типа
Dim i As Integer, J As Integer, K As Integer
'Объявление переменных MIN, MAX и массива A(1 To 10) вещественного типа
Dim A(1 To 10) As Single, MIN As Single, MAX As Single
'Объявление переменной S строкового типа
Dim s As String
'Задание начального значения
s = " "
'Начало цикла, где I- параметр цикла
For i = 1 To 10
'Заполнение элементов массива - A(I)случайными величинами
A(i) = (Rnd * 10 + i)
'Суммирование значений элементов в строку для вывода в окно
s = s + " " + Str(Format(A(i), "FIXED"))
'Продолжение цикла
Next i
'Вывод значений элементов массива в диалоговое окно
MsgBox "Вывод элементов массива", vbInformation, "S= " & s
'Задание начальных значений переменным MIN и MAX
MAX = A(1): K = 1
MIN = A(1): J = 1
'Начало цикла
For i = 2 To 10
'Поиск MIN путем сравнения MIN с последующими элементами
If A(i) < MIN Then MIN = A(i): J = i
'Поиск Max путем сравнения Max с последующими элементами
If A(i) > MAX Then MAX = A(i): K = i
'Продолжение цикла
Next i
'Вывод результата вычислений
MsgBox "Индексы  мин. и макс.  элементов " & (Chr(10) & Chr(13)) _
& "j= " & J & " " & " k= " & K, vbInformation, _
" MIN= " & Str(Format(MIN, "fixed")) & " " & " MAX= " & Str(Format(MAX, "fixed"))
'Конец кода программы
End Sub
```

Рис. 39. Текст программы для примера 1

2. Дан массив целых чисел B(10). Найти сумму значений элементов массива из диапазона [-5,35].

Блок-схема алгоритма примера приведена ниже *(рис. 40)*.

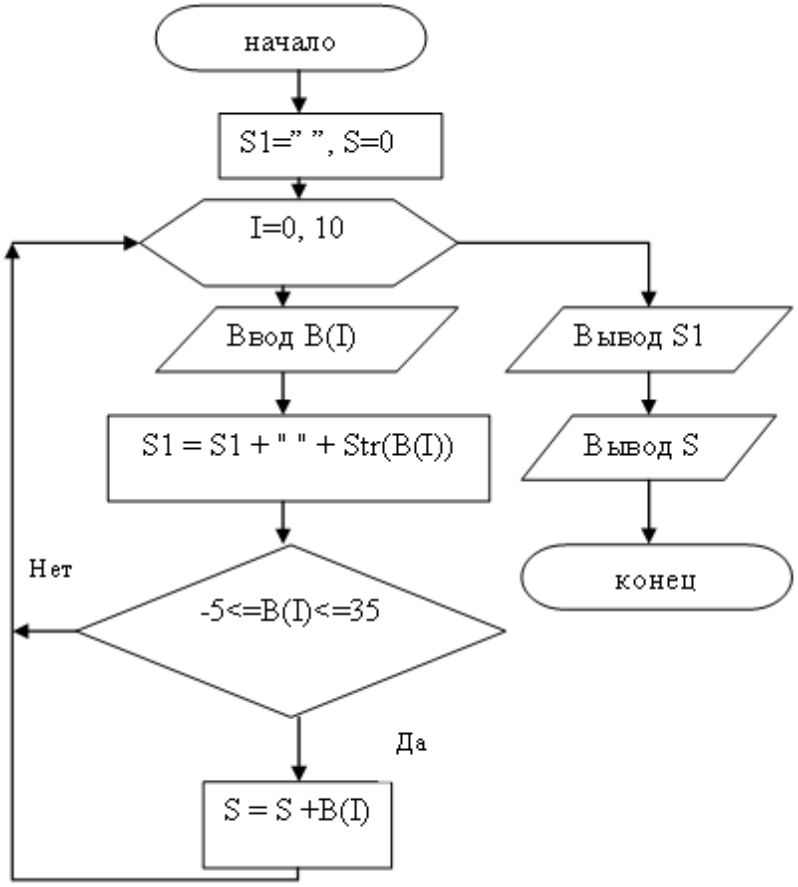


Рис. 40. Блок-схема алгоритма примера 2

Текст программы для примера приведен ниже *(рис. 41)*.

```
Option Explicit
Sub PRIM4_2()
'Объявление переменных I, S, массива B(10) целого типа
Dim B(10) As Integer, i As Integer, s As Integer
'Объявление переменной S1 строкового типа
Dim S1 As String
'Задание начальных значений переменным S1,S
S1 = " ": s = 0
'Начало цикла, где I- параметр цикла
For i = 0 To 10
'Заполнение элементов массива B(I) введенными с клавиатуры величинами
B(i) = Val(InputBox("Введите значение элемента массива"))
'Суммирование значений элементов в строку для вывода в окно
S1 = S1 + " " + Str(B(i))
'Вычисление суммы после сравнения значения элемента массива с границами заданного 'диапазона
If (B(i) >= -5) And (B(i) <= 35) Then s = s + B(i)
'Продолжение цикла
Next i
'Вывод значений элементов массива в диалоговое окно
MsgBox S1, vbInformation, "Вывод массива в строку"
'Вывод результата вычислений
MsgBox " s= " & s, vbInformation, "Сумма значений из указанного диапазона [-5,35]"
'Конец кода программы
End Sub
```

Рис. 41. Текст программы для примера 2

3. Дан массив D(10). Найти количество отрицательных элементов массива.

Блок-схема алгоритма примера приведена ниже *(рис. 42)*.

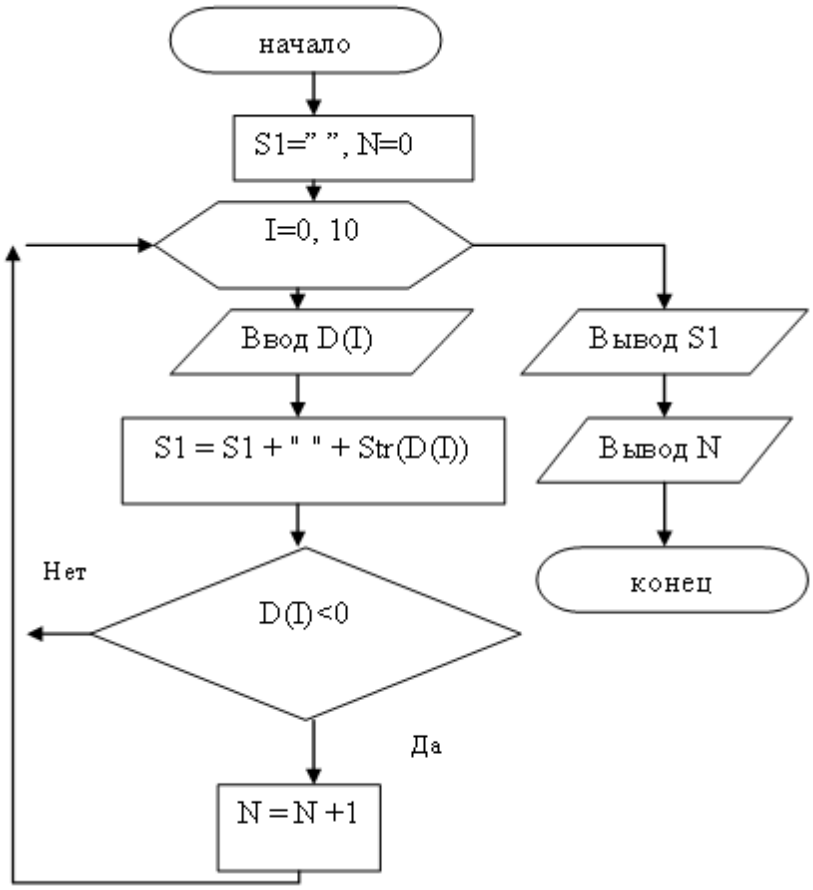


Рис. 42. Блок-схема алгоритма примера 3

Текст программы для примера приведен ниже *(рис. 43)*.

```
Option Explicit
Sub PRIM4_3()
'Объявление переменных I, N целого типа, массива D(10) вещественного типа
Dim D(10) As Single, i As Integer, n As Integer
'Объявление переменной S1 строкового типа
Dim S1 As String
'Задание начальных значений переменным S1,N
S1 = " ": n = 0
'Начало цикла, где I- параметр цикла
For i = 0 To 10
'Заполнение элементов массива D(I) введенными с клавиатуры величинами
D(i) = Val(InputBox("Введите значение элемента массива"))
'Суммирование значений элементов в строку для вывода в окно
S1 = S1 + " " + Str(D(i))
'Вычисление суммы после сравнения значения элемента массива с 0
If D(i) < 0 Then n = n + 1
'Продолжение цикла
Next i
'Вывод значений элементов массива в диалоговое окно
MsgBox S1, vbInformation, "Вывод массива в строку"
'Вывод результата вычислений
MsgBox " N= " & n, vbInformation, "Количество отрицательных элементов массива"
'Конец кода программы
End Sub
```

Рис. 43. Текст программы для примера 3

4. Сформировать целочисленную матрицу, используя датчик случайных чисел. В полученной матрице найти столбцы, в которых число четных элементов превышает число нечетных элементов.

Блок-схема алгоритма примера приведена ниже *(рис. 44)*.

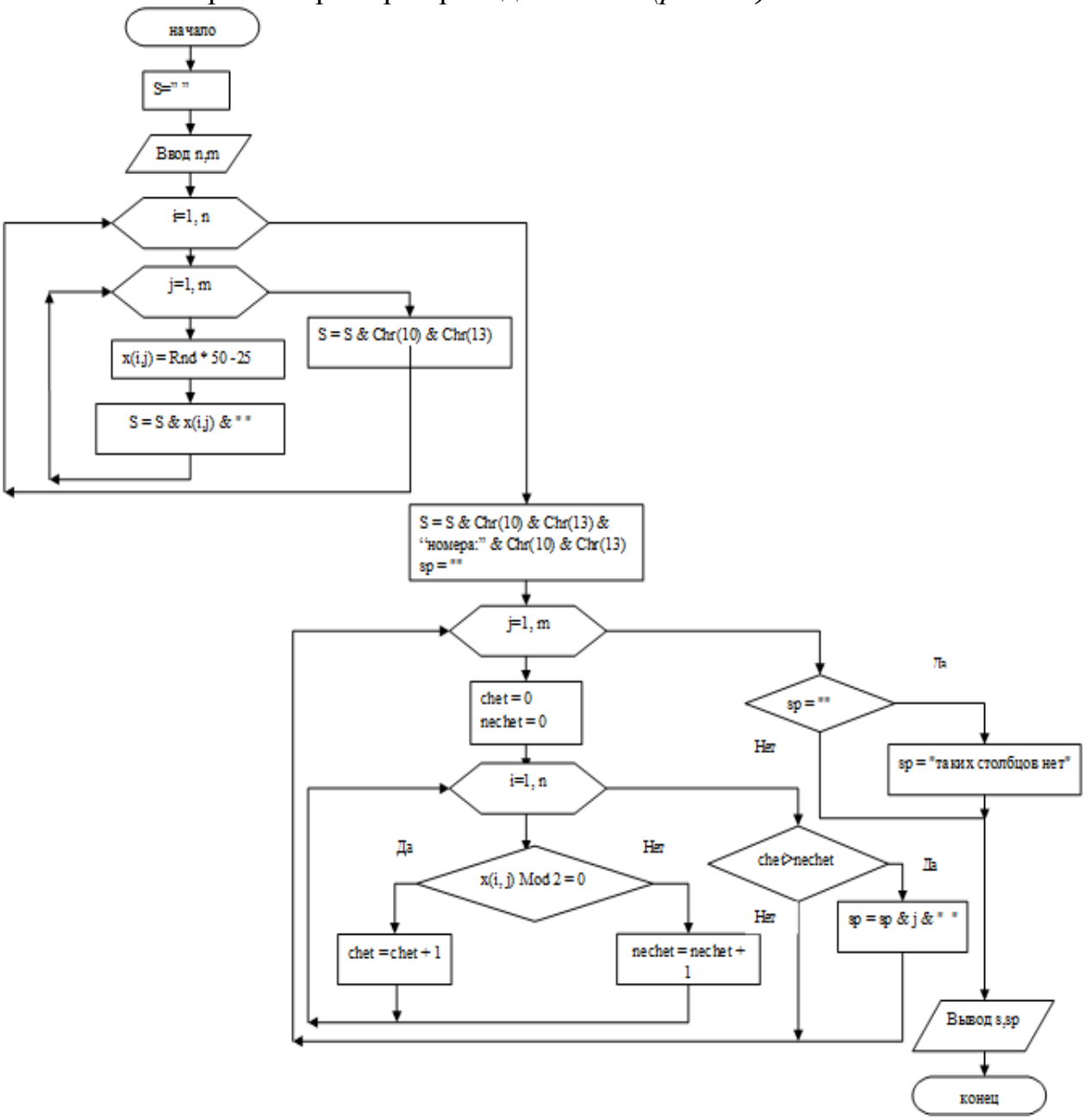


Рис. 44. Блок-схема алгоритма примера 4

Текст программы для примера приведен ниже *(рис. 45)*.

```
Sub matr()
'Используя датчик случайных чисел сформировать целочисленную матрицу.
'Найти столбцы, в котором число четных элементов превышает число нечетных.
Dim x() As Integer
Dim s As String, sp As String, chet As Byte, nechet As Byte
Dim n As Byte, m As Byte, i As Byte, j As Byte
s = ""
'Определим размерность массива
n = InputBox("число строк: ", "матрица", 1)
m = InputBox("число столбцов: ", "матрица", 1)
'зарезервируем память под элементы массива
ReDim x(1 To n, 1 To m)
' заполним матрицу случайными целыми числами
Randomize
For i = 1 To n
For j = 1 To m
x(i, j) = 50 * Rnd - 25
s = s & x(i, j) & "  "
Next j
s = s & Chr(10) & Chr(13)
Next i
s = s & Chr(10) & Chr(13)
s = s & " столбцы, в которых число четных превышает число нечетных, имеют номера:" & Chr(10) & Chr(13)
' обработка элементов массива
sp = ""
For j = 1 To m
chet = 0: nechet = 0
For i = 1 To n
If x(i, j) Mod 2 = 0 Then chet = chet + 1 Else nechet = nechet + 1
Next i
If chet > nechet Then sp = sp & j & "  "
Next j
If sp = "" Then sp = "таких столбцов нет"
MsgBox s & Chr(10) & Chr(13) & sp, _
vbInformation, " поиск столбцов, в которых число четных превышает число нечетных"
End Sub
```

Рис. 45. Текст программы для примера 4

Работа с пользовательскими функциями и процедурами общего вида

В VBA процедура является частью кода, выполняющего определенный алгоритм, представленный набором инструкций. Процедура может быть двух видов:

- процедура общего вида (**Sub**);
- процедура-функция (**Function**).

Процедуры **Sub** и **Function** являются самостоятельными процедурами, позволяющими выполнять последовательность инструкций (действий) над заданными аргументами.

Запись процедуры общего вида имеет синтаксис, представленный ниже:

[Private/Public] [Static] Sub Имя[(Список аргументов)]
<Инструкции>
<[Exit Sub]>
<Инструкции>
End Sub

Запись процедуры-функции имеет синтаксис, представленный ниже:

[Private/Public] [Static] Function Имя[(Список аргументов)] [As Тип]
<Инструкции>
<[Exit Function]>
<Инструкции>
End Function

Примечание. По правилам VBA допускается отсутствие некоторых параметров в их заголовках, поэтому в представленном выше синтаксисе они заключены в квадратные скобки.

Первая строка – заголовок процедуры или процедуры-функции, признаком которой является ключевое слово **Sub** или соответственно **Function.** Вслед за именем в скобках располагается список формальных аргументов (они могут отсутствовать).

Ниже перечислены обозначения параметров заголовка процедур и их назначение.

Private – процедура общего вида **Sub** (процедура-функция **Function***)* доступна для всех других процедур только того модуля, в котором она написана.

Public – процедура общего вида **Sub** (процедура-функция **Function***)* доступна для всех процедур во всех модулях.

[Static] – локальные переменные процедуры общего вида **Sub** (процедуры-функции **Function***)* сохраняются между вызовами этой процедуры.

Имя – имя процедуры общего вида **Sub** (процедуры-функции **Function***)*, удовлетворяющее правилам построения имен в VBA.

Список аргументов – список имен переменных, представляющий аргументы, которые передаются в процедуру общего вида **Sub** (процедуру-

функцию **Function***)* при ее вызове и возвращаются в вызывающую процедуру. Имена переменных разделяются запятой.

Инструкции – последовательность инструкций соответственно алгоритму.

Наличие (возможно также отсутствие) инструкций **Exit Sub** и **Exit Function** приводит к немедленному выходу из процедур.

Согласно синтаксису записи процедур общего вида и процедур-функций запись заголовка и окончания процедур различны для этих типов процедур. Также процедура-функция может иметь тип, так как вычисленное значение в самой процедуре Function присваивается одноименной переменной с типом, указанным в ее заголовке.

Обращение к процедурам и возврат вычисленных в них значений тоже имеют различия.

Вызов процедур общего вида (Sub)

Вызов процедуры общего вида **Sub** осуществляется из главной (вызывающей) процедуры по ее имени со списком фактических аргументов.

Инструкция обращения к процедуре имеет следующий формат:

Call <Имя процедуры>(Список фактических аргументов)

Возврат из процедуры происходит к инструкции, следующей за обращением.

Примеры использования процедур общего вида

1. Даны программные коды двух процедур OBRACHENIE и prim1. Обе процедуры хранятся в одном модуле *(рис. 46)*.

```
Sub OBRACHENIE()
Dim n As Byte, m As Byte
Dim res As Long
n = Val(InputBox("", , 10))
m = Val(InputBox("", , 2))
Call prim1(n, m, res)
MsgBox "res=" & res
End Sub
Private Sub prim1(n, m, prog1)
Dim i As Byte
prog1 = 1
For i = 1 To n Step m
prog1 = prog1 * i ^ 2
Next i
End Sub
```

Рис. 46. Программные коды двух процедур OBRACHENIE и prim1

В приведенном выше программном коде процедуры OBRACHENIE строка **Call prim1(n, m, res)** представляет собой обращение к процедуре prim1, где **prim1** – имя процедуры, а **n, m, res –** аргументы, называемые фактическими.

Аргументы **n, m** – аргументы, передающие значения из вызывающей процедуры в вызываемую процедуру (над этими значениями в процедуре производятся вычисления), **res** – аргумент, передающий результат вычислений

из вызываемой процедуры prim1 в вызывающую процедуру OBRACHENIE. Фактические аргументы, представляющие собой исходные данные для вычислений, должны быть определены до вызова процедуры.

Блок-схема алгоритма для вызывающей процедуры OBRACHENIE примера приведена ниже *(рис. 47)*.

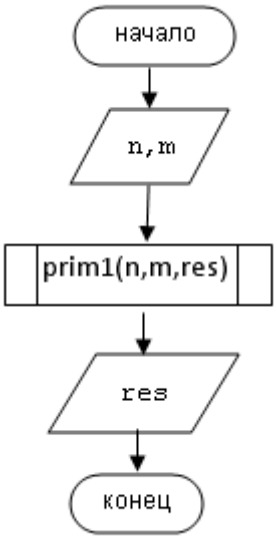


Рис. 47. Блок-схема алгоритма для вызывающей процедуры OBRACHENIE примера 1

Блок-схема алгоритма для вызываемой процедуры prim1 примера приведена ниже *(рис. 48)*.

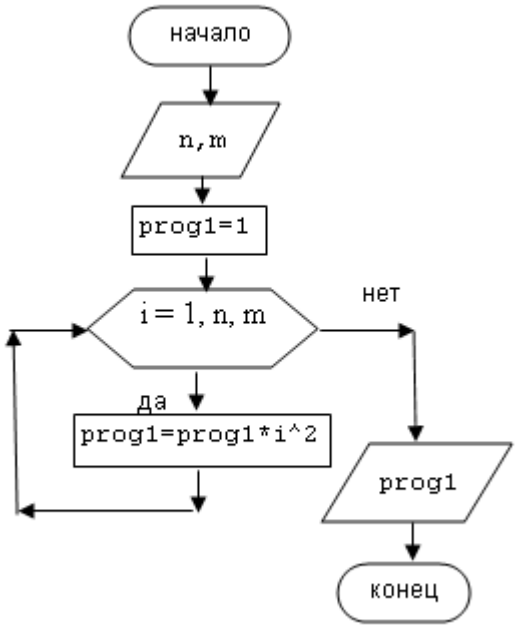


Рис. 48. Блок-схема алгоритма для вызываемой процедуры prim1 примера 1

Ввод и вывод данных в блок-схеме процедуры prim1 соответствует передаче данных через аргументы процедуры, **n, m, prog1** представляют собой формальные аргументы процедуры.

Для n=10, m=2 процедура prim1 находит произведение квадратов нечетных чисел из интервала от 1 до 10.

2. Ниже приведен пример обращения из процедуры **DEMO()** к процедуре общего вида **Obr(f, i, o, FIO)** с фактическими аргументами **f, i, o, FIO.** Обе процедуры хранятся в одном модуле *(рис. 49)*.

В процедуру **Obr** передаются фамилия, имя, отчество пользователя, процедура возвращает ФИО пользователя.

```
Public Sub Obr(f As String, i As String, o As String, FIO As String)
FIO = f + " " & Left(i, 1) & "." & Left(o, 1) & "."
End Sub
```

```
Private Sub DEMO()
Dim FIO As String, f As String, i As String, o As String
f = InputBox("Ваша фамилия!", "Знакомство")
i = InputBox("Ваше имя!", "Знакомство")
o = InputBox("Ваше отчество!", "Знакомство")
Call Obr(f, i, o, FIO)
MsgBox "Пользователь - " & FIO, , Date
End Sub
```

Рис. 49. Программные коды двух процедур **DEMO** и **Obr**

Блок-схема алгоритма для вызывающей процедуры **DEMO** примера приведена ниже *(рис. 50)*.

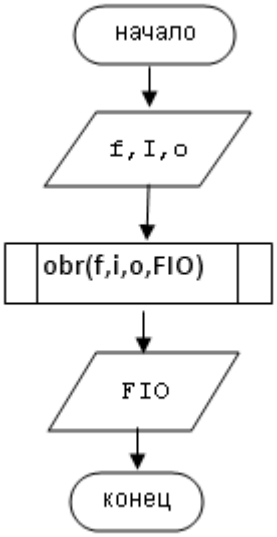


Рис. 50. Блок-схема алгоритма для вызывающей процедуры **DEMO** примера 2

Блок-схема алгоритма для вызываемой процедуры **Obr** примера приведена ниже *(рис. 51)*.

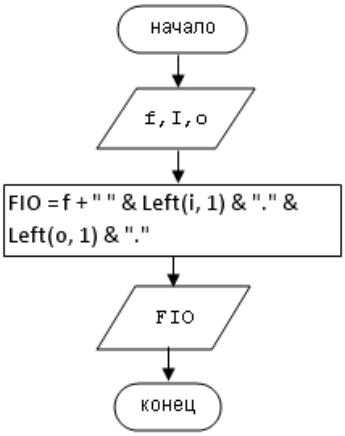


Рис. 51. Блок-схема алгоритма для вызываемой процедуры **Obr** примера 2

В данном примере имена фактических и формальных аргументов полностью совпадают. Ввод и вывод данных в блок-схеме процедуры **Obr** соответствует передаче данных через аргументы процедуры.

Вызов процедур-функций (Function)

В отличие от обращения к процедуре общего вида (**Sub)** обращение к процедуре-функции (**Function)** производится по ее имени со списком фактических аргументов.

Имя функции со списком фактических аргументов может находиться в правой части выражения инструкции присваивания в вызывающей программе (например, **Z= F(x,y)**, где **F** – имя процедуры-функции, а **x** и **y –** аргументы) или может быть аргументом функции Format, параметром инструкции MsgBox**.**

Примеры использования процедур-функций

1. Даны программные коды вызывающей процедуры общего вида fun и вызываемой процедуры-функции f. Обе процедуры хранятся в одном модуле *(рис. 52)*. Процедура fun вводит параметры c и t, в инструкции MsgBox обращается к функции f с аргументами c и t – f(c,t) и выводит вычисленное и отформатированное с помощью Format значение функции.

```
Sub fun()
Dim c As Single, t As Single
c = InputBox("c=")
t = InputBox("t=")
MsgBox "при" & Chr(10) & Chr(13) & "c=" & c _
& Chr(10) & Chr(13) & "t=" & t, , "f=" & Format(f(c, t), "fixed")
End Sub
```

```
Public Function f(c As Single, t As Single) As Single
If t = 1 Then
f = (t ^ 2 + 5) / 6 + 4 * t
ElseIf t > 1 Then
f = (c + 7) / (2 * t) * Log(t)
Else
f = t / 2 + (c + t ^ 2) / 3
End If
End Function
```

Рис. 52. Программные коды процедуры общего вида fun и процедуры-функции *f*

Блок-схема алгоритма для вызывающей процедуры fun примера приведена ниже *(рис. 53)*.

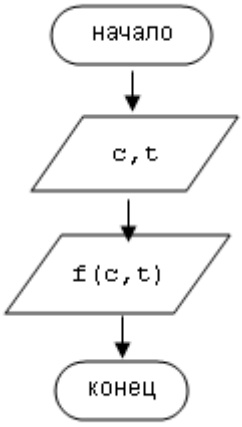


Рис. 53. Блок-схема алгоритма для вызывающей процедуры fun примера 1

Блок-схема алгоритма для вызываемой процедуры-функции *f* примера1 приведена ранее на *рис. 26*.

Ключевое слово *Public* в заголовке процедуры-функции *f* показывает, что она доступна всем другим процедурам во всех модулях, созданных пользователем. Эта функция также доступна для вычислений в таблицах рабочего листа Excel, она относится к категории функций «**Определенные пользователем»,** как и все процедуры-функции, созданные пользователем в данной рабочей книге.

Обращение к процедуре-функции из листа рабочей книги Excel

При использовании в вычислениях процедур-функций, созданных пользователем, проще обратиться к «Мастеру функций», нажав кнопку 

панели инструментов на вкладке «Формат» в области «Библиотека функций» или нажав кнопку  слева от строки формул и выбрать категорию «Определенные пользователем». Окно «Мастера функций» будет содержать обзор функций данной категории, для помеченной функции в окно будет выводиться вид обращения и необходимые аргументы для ее работы, а также пояснения и справку.

После выбора определенной функции появляется окно «Аргументы функции», в котором выполняется задание определенных для данной функции аргументов. Аргументы функции могут быть представлены константами, ссылками на ячейки, диапазонами областей и другими функциями. После задания определенных для данной функции аргументов выполняется вычисление и возврат вычисленного значения в выделенную ячейку таблицы.

Приведенный выше программный код процедуры-функции f *(рис. 52)* с аргументами c и t (f(c,t)) предназначен для поиска значений функции:

$$F=\begin{cases}\dfrac{t^2+5}{6}+4t; t=1\\ \dfrac{c+7}{2t}\ln(t); t>1\\ \dfrac{t}{2}+\dfrac{c+t^2}{3}; t<1\end{cases}$$

Сформируем таблицу с исходными данными для расчетов на листе рабочей книги *(рис. 54)*.

c	t	f
6	-7	
	-6	
	-5	
	-4	
	-3	
	-2	
	-1	
	0	
	1	
	2	
	3	
	4	
	5	
	6	
	7	

Рис. 54. Таблица с исходными данными

Обратимся к «Мастеру функций», выберем категорию «Определенные пользователем» *(рис. 55)*.

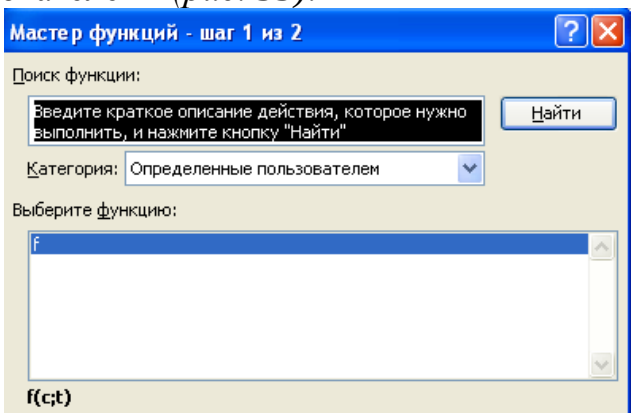


Рис. 55. «Мастер функций» категория «Определенные пользователем»

Выберем функцию f с аргументами c и t (f(c,t)) *(рис. 56)*.

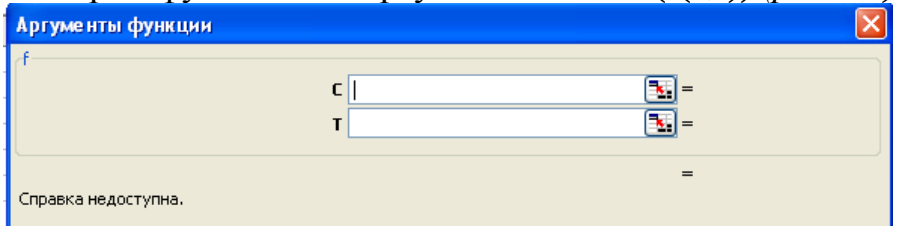


Рис. 56. «Мастер функций», окно «Аргументы функции»

Определим исходные данные для верхней ячейки столбца f *(рис. 57)*.

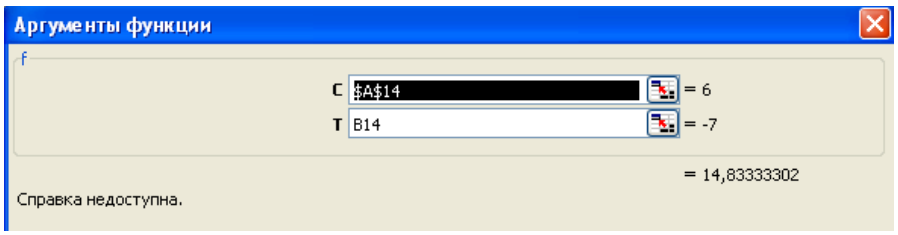


Рис. 57. Вычисленное значение функции

Скопируем вычисления в другие ячейки столбца f *(рис. 58)*.

	c	t	f
13			
14	6	-7	14,83
15		-6	11,00
16		-5	7,83
17		-4	5,33
18		-3	3,50
19		-2	2,33
20		-1	1,83
21		0	2,00
22		1	5,00
23		2	2,25
24		3	2,38
25		4	2,25
26		5	2,09
27		6	1,94
28		7	1,81

=f(A14;B14)

Рис. 58. Таблица значений функции f(c,t)

2. Рассмотрим пример применения процедуры-функции для решения квадратного уравнения.

Сформируем таблицу с исходными данными для расчетов на листе рабочей книги *(рис. 59)*.

	a	b	c	Решение кв. уравнения
4	1	2	3	
5	6	5	4	
6	2	11	3	
7	2	4	2	
8	2	9	1	

Рис. 59. Таблица с исходными данными

Ниже представлен программный код процедуры-функции Reshur, которая формирует текстовое сообщение о решении квадратного уравнения с

параметрами a,b,c (исходные данные – формальные аргументы). Сформированный ответ хранится в переменной Reshur.

Ключевое слово *Public* в заголовке процедуры-функции Reshur показывает, что она доступна не только всем другим процедурам во всех модулях, созданных пользователем, но и для вычислений в таблицах рабочего листа Excel, и относится к категории функций «**Определенные пользователем»,** как и все процедуры-функции, созданные пользователем в данной рабочей книге.

```
Public Function Reshur(a As Single, b As Single, c As Single) As String
Dim d As Single, s As String
s = "при " & " a=" & a & " b=" & b & " c=" & c & Chr(10)
d = b ^ 2 - 4 * a * c
If d = 0 Then
s = s & "ур-ние имеет корни " & "x1=x2=" & -b / 2 / a
ElseIf d > 0 Then
s = s & "ур-ние имеет корни " & _
" x1=" & Format((-b + Sqr(d)) / 2 / a, "fixed") & _
" x2=" & Format((-b - Sqr(d)) / 2 / a, "fixed")
Else
s = s & "ур-ние имеет компл. корни"
End If
Reshur = s
End Function
```

Рис. 60. Программный код процедуры-функции Reshur

Блок-схема алгоритма для вызываемой процедуры-функции Reshur примера приведена на *рис. 61*.

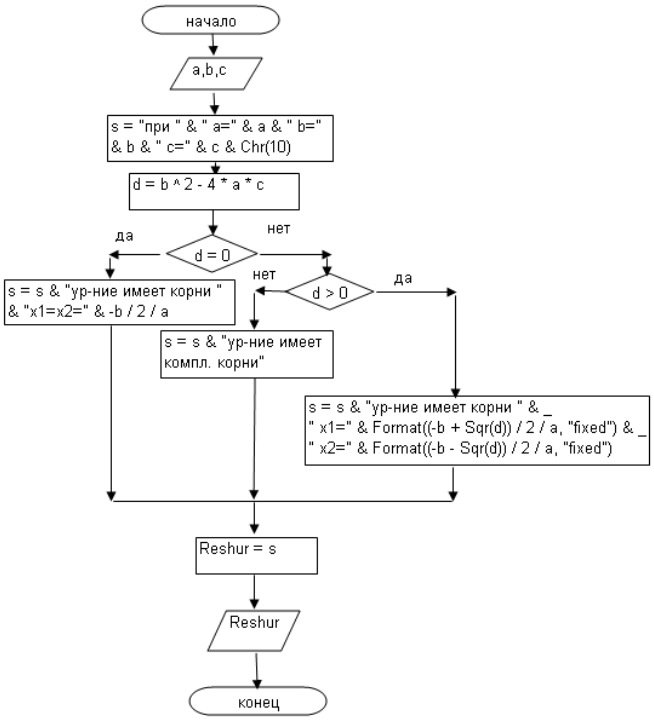


Рис. 61. Блок-схема алгоритма для вызываемой процедуры Reshur примера 2

В ячейку E4 таблицы, сформированной на рабочем листе Excel *(рис. 59)* введем обращение к процедуре-функции Reshur =Reshur(B4;C4;D4). Фактическими аргументами для процедуры-функции Reshur в данном случае являются адреса ячеек: B4, C4, D4.

Скопируем вычисления в другие ячейки столбца E. Получим таблицу следующего вида *(рис. 62)*.

a	b	c	Решение кв. уравнения
1	2	3	при a=1 b=2 c=3 ур-ние имеет компл. корни
6	5	4	при a=6 b=5 c=4 ур-ние имеет компл. корни
2	11	3	при a=2 b=11 c=3 ур-ние имеет корни x1=-0,29 x2=-5,21
2	4	2	при a=2 b=4 c=2 ур-ние имеет корни x1=x2=-1
2	9	1	при a=2 b=9 c=1 ур-ние имеет корни x1=-0,11 x2=-4,39

Рис. 62. Таблица значений функции Reshur

2.2. Система программирования Visual Basic for Applications

Система программирования – это пакет программных средств, включающий не только транслятора, но и редактор для ввода кода, средства автоматизации создания и отладки программ, библиотеки с готовыми блоками кода, удобный справочник и другие специальные средства.

Из любого приложения Microsoft Office в операционной среде Windows можно запустить систему программирования Visual Basic for Applications – коротко (VBA). Данная система программирования является визуальной системой программирования, так как объекты в этой системе создаются при помощи мыши. В ОС Windows большое количество стандартных элементов: окон, меню, кнопок и т. д. Для них в системе программирования заготовлены стандартные блоки машинного кода.

Visual Basic for Applications представляет собой единую среду редактирования, схожую со средой автономной версии Visual Basic. Среда редактирования Visual Basic включает редактор кода, иерархическое средство просмотра объектов, многооконный отладчик, окно отображения свойств и средство просмотра проектов для управления кодом и объектами проекта.

Visual Basic for Applications, как и все современные системы программирования имеет хороший инструментарий для корректного написания программы, ее форматирования, редактирования и отладки, встроенный в удобный и интуитивно понятный интерфейс, способствующий всесторонней поддержке программирования. Вся разработка и отладка программ ведется с помощью Редактора Visual Basic (Visual Basic Editor, VBE).

Обычно в окне редактора используются основные три отдельных окна:

- **Project** (Окно проекта);
- **Properties (**Окно свойств**);**
- С**ode (**Окно кода**).**

В окне проекта отображается иерархическое дерево проектов приложения и модулей этих проектов с их элементами.

- **Microsoft Outlook Objects (**папка с объектом приложения ThisOutlookSession);
- **Forms** (папка содержит пользовательские модули форм);
- **Modules (**папка содержит стандартные модули пользователя**);**
- **Class Modules (**папка содержит пользовательские модули классов);
- **References** (папка содержит ссылки на объекты из внешних библиотек и список модулей этих библиотек).

Для того чтобы начать этап написания программы на языке программирования VBA, необходимо создать пользовательский модуль, поскольку все процедуры и функции могут быть созданы только в теле модуля.

Создание программных модулей на VBA

Настройка MS Office 2007 и выше для работы с редактором VBA
Вывод вкладки «Разработчик»

1. В меню «**Файл**» выбрать пункт «**Параметры**» *(рис. 63)*.

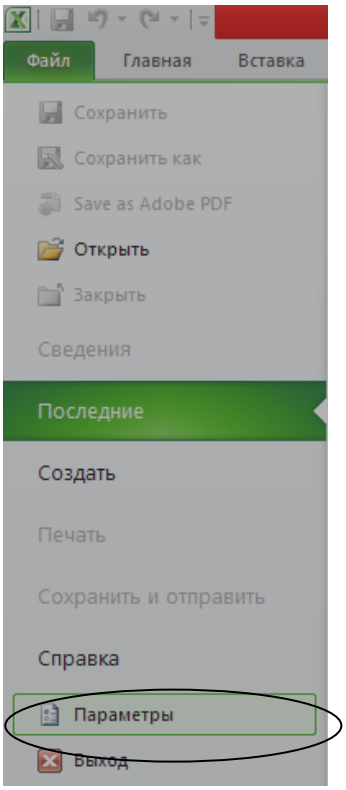


Рис. 63. Пункт «**Параметры**» меню «**Файл**»

2. В подменю «**Настройка ленты**» следует выбрать (отметить) пункт «**Разработчик**» *(рис. 64)*. После применения данной настройки на ленте появится вкладка «**Разработчик**».

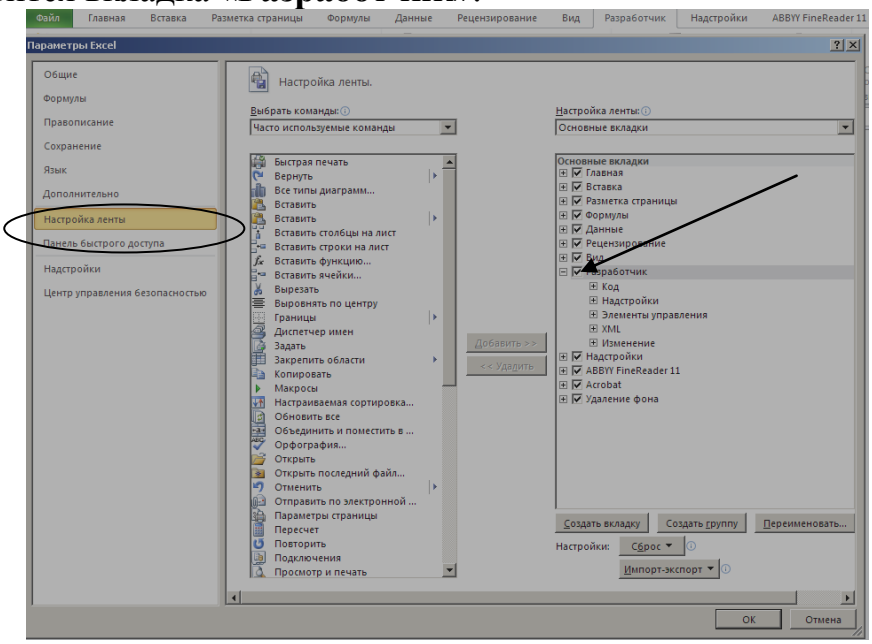


Рис. 64. Пункт «Разработчик» подменю «Настройка ленты»

Запуск редактора Visual Basic for Applications

Для начала работы в редакторе **Visual Basic for Applications** необходимо нажать на пиктограмму, показанную на рисунке ниже *(рис. 65)*.

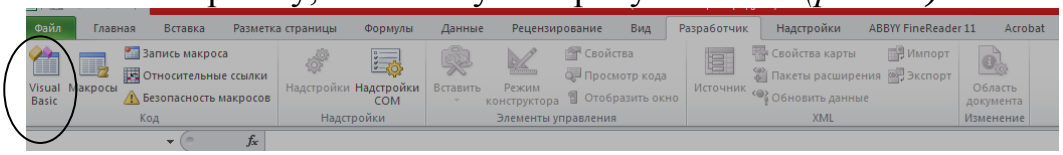


Рис. 65. Кнопка запуска редактораVisual Basic for Applications

Редактор Visual Basic for Applications

Меню данного редактора представлено на рисунке ниже. Команды для работы с VBA сгруппированы в нескольких раскрывающихся пунктах меню *(рис. 66)*.

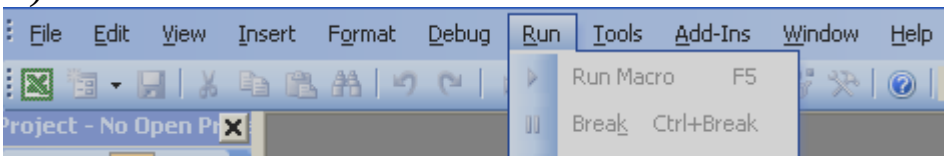


Рис. 66. РедакторVisual Basic for Applications

Сохранение проекта в MS Office Excel 2007 и выше

Важно!!!

При сохранении файла следует обратить внимание на тип файла для сохранения программных модулей в рабочей книге (для правильного последующего исполнения созданных программ следует указать Тип файла: **Книга Excel с поддержкой макросов**) *(рис. 67)*.

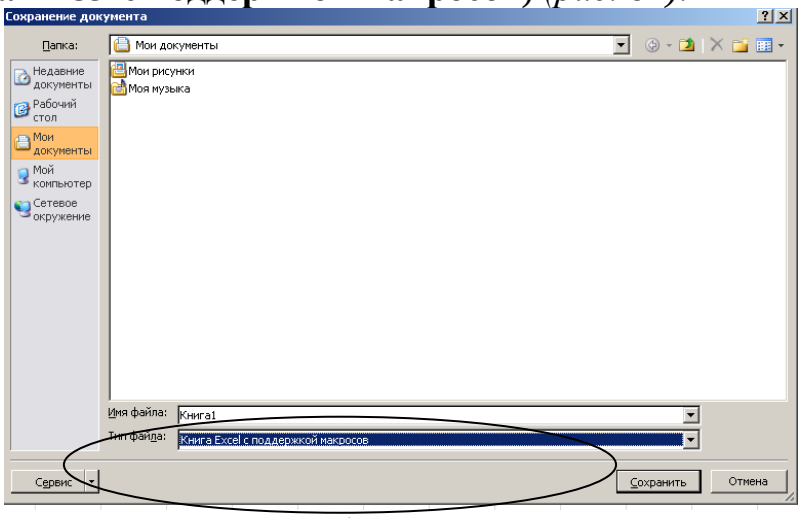


Рис. 67. Сохранение файла

Запуск редактора Visual Basic for Applications в MS Office Excel 2003

После запуска приложения (Excel, Word) необходимо войти в меню «Сервис», выбрать команду «Макрос» и из предложенного меню выбрать команду «Редактор Visual Basic» *(рис. 68)*.

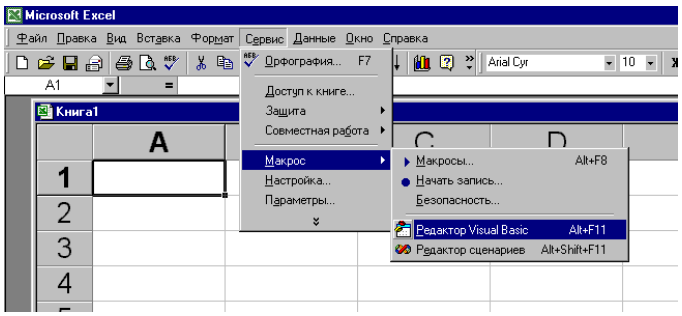


Рис. 68. Запуск редактора в MS Office Excel 2003

Редактор Visual Basic (VBE)

Вставка модуля

Для вставки модуля необходимо выбрать меню «Вставка»/Insert, а затем выбрать команду «Модуль»/Module. Будет создан пустой модуль *(рис. 69)*. Наличие нового модуля отразится в окне проекта (VBA Project). Если оно не активно, то выберите команду «Вид»/View, «Проект»/Project Explorer либо нажмите кнопку  панели инструментов **VBE.**

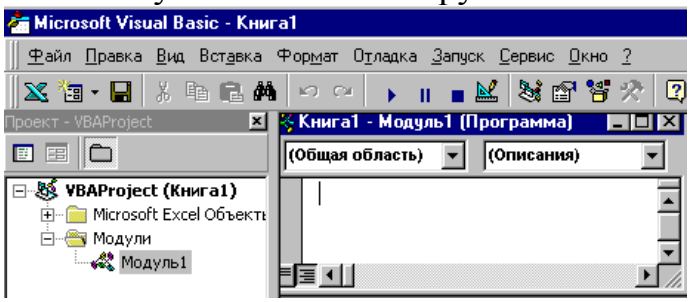


Рис. 69. Вставка модуля в редакторе VBA

Самыми первыми словами в модуле должны быть слова Option Explicit. Эта инструкция проверяет, описана ли переменная или константа, которая участвует в программе в разделе описания процедуры, или нет. Если пользователь забыл это сделать, то с помощью данной инструкции компилятор языка выдаст сообщение об ошибке. Применение инструкции Option Explicit позволит исключить достаточно много ошибок при программировании, особенно у начинающих программистов. Далее записывается программный код.

Начало работы с модулем

При работе с модулем, содержащим программный код, важно знать назначение основных кнопок на панели инструментов. После создания текста программы ее необходимо откомпилировать для проверки синтаксиса и семантики написанного текста. Для этого необходимо выбрать меню «Отладка»/Debug , а затем выбрать команду «Компилировать проект»/Compile Project.

На панели инструментов для этого служит специальный значок  .

Иногда при запуске редактора VBA этот значок отсутствует. Его необходимо поместить на панель инструментов. Для этого в меню «Вид»/View нужно выбрать команду «Панели инструментов»/Toolbars и «Настройка». В

открывшемся диалоговом окне выбрать вкладку «Команды», найти категорию «Отладка»/Debug, выделить значок «Компилировать проект»/Compile Project и, не отпуская нажатую клавишу мыши, поместить этот значок на панель инструментов.

При успешной компиляции можно выполнить команды меню «Запуск»/Run:

- ✓ команду «Запуск программы»/Run Macro,
- ✓ команду «Прервать программу»/Break,
- ✓ команду «Сброс программы»/Reset.

Найти их можно на панели инструментов, как показано на *рис. 70*.

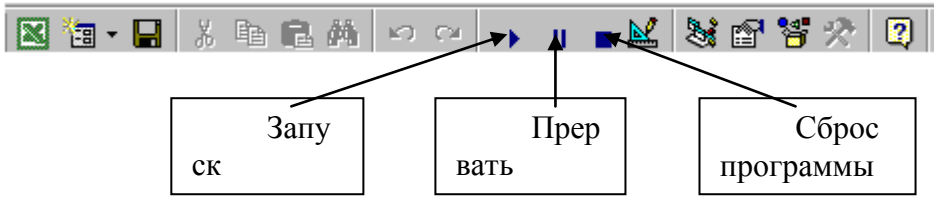


Рис. 70. Начало работы с модулем

Отладка программ

При написании и создании программы неизбежно появляются ошибки:

1. компиляции;
2. выполнения;
3. логические.

Ошибки компиляции возникают при некорректном вводе числа скобок, неправильном имени, неполном вводе инструкции и т. д. Некоторые ошибки появляются при завершении набора строки и нажатия клавиши Enter. Строка, в которой обнаружена ошибка, выделяется красным цветом, и на экране появляется диалоговое окно с сообщением о возможной причине ошибки (*рис. 71.1)*.

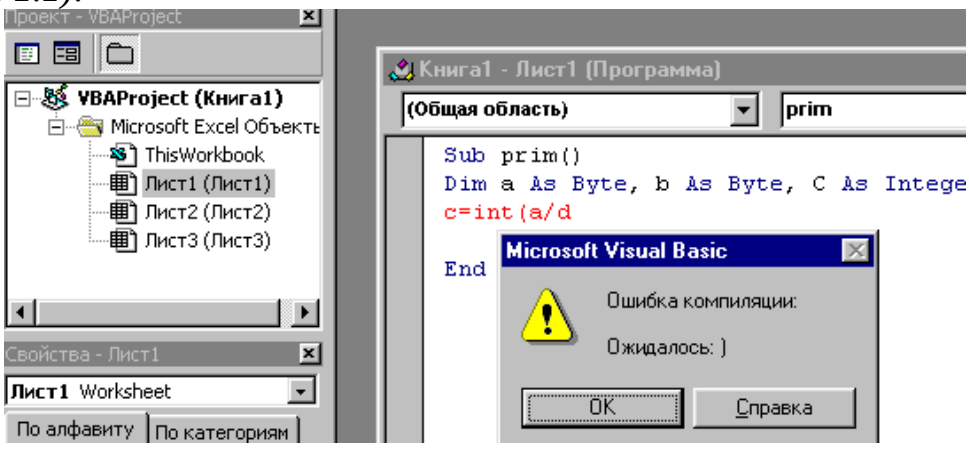


Рис. 71.1. Сообщение об ошибке при компиляции программы

Как упоминалось ранее, инструкция Option Explicit , с которой должен начинаться модуль, позволяет достаточно легко отслеживать ошибки, связанные с использованием переменных, не объявленных явно, и своевременно их устранять.

Ошибки выполнения. Появляются после успешного завершения компиляции программы уже на стадии выполнения. Причиной этого может быть некорректный ввод данных, например, вместо числа вводится строка знаков. Ошибка может возникнуть также из-за некорректных данных при выполнении операции деления, например деление на ноль (*рис. 71.2)*.

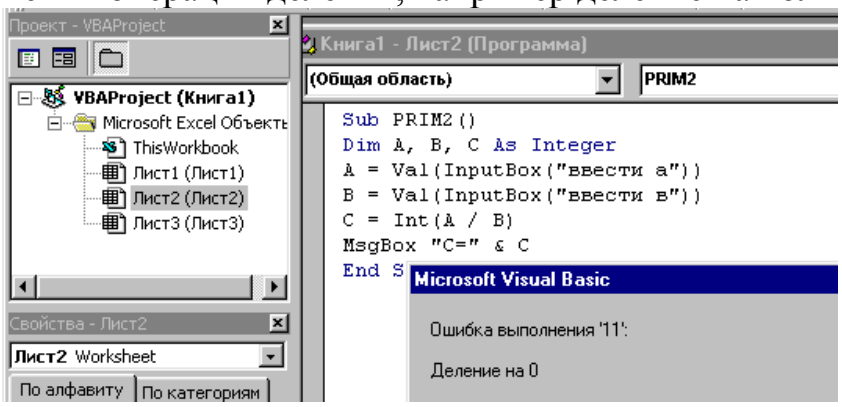


Рис. 71.2. Сообщение об ошибке при выполнении программы

Логические ошибки. Распознаются наиболее сложно, так как они не приводят к прерыванию программы, но при этом при выполнении программы выдаются неверные результаты. Причиной тому чаще всего является некорректный алгоритм.

Отключение макросов

Макросы могут быть отключены из-за установленного высокого уровня безопасности.

Чтобы установить другой уровень безопасности в **MS Office 2007** и выше надо выбрать команду «**Безопасность макросов**» вкладки **«Разработчик»** в области **«Код»** *(рис. 72.1)*.

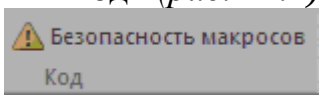


Рис. 72.1. Установка уровня безопасности в **MS Office 2007** и выше

Чтобы установить другой уровень безопасности в **MS Office Excel 2003** пользователь должен выбрать меню «Сервис» команду «**Безопасность**» *(рис. 72.2)*.

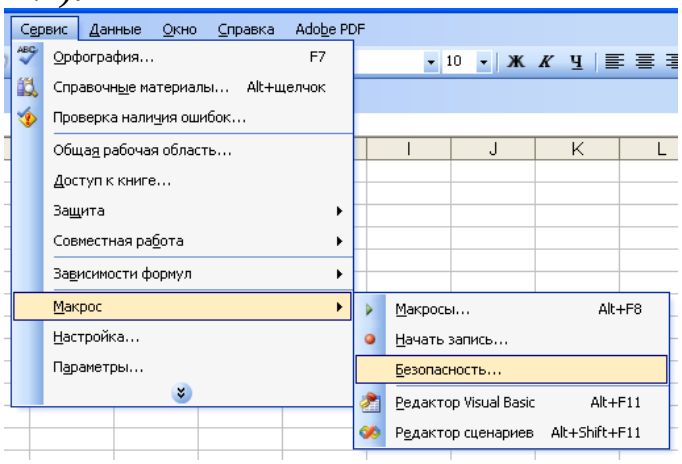


Рис. 72.2. Установка уровня безопасности в **MS Office 2003**

Диалоговое окно «Безопасность» позволит установить средний или низкий уровень безопасности *(рис. 73)*.

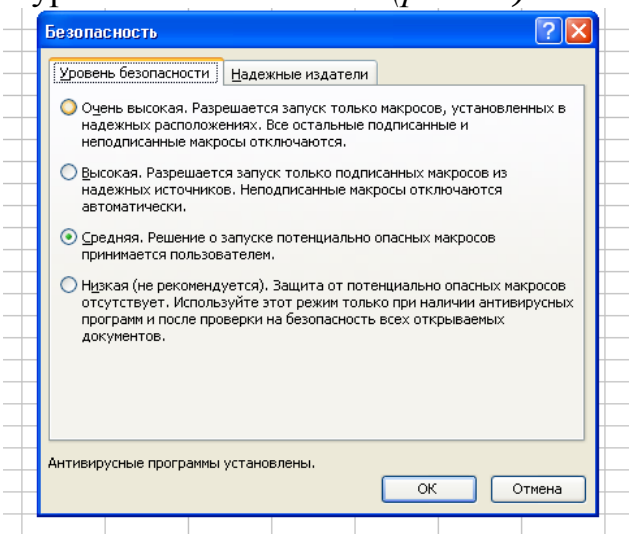


Рис. 73. Диалоговое окно «Безопасность»

При открытии файла возникает запрос на отключение макросов (если установить средний уровень безопасности) *(рис. 74)*.

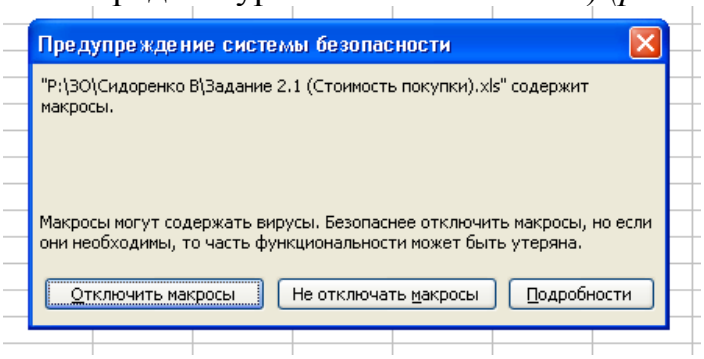


Рис. 74. Запрос на отключение макросов

При выборе **«Не отключать макросы»** содержащиеся в открытом файле коды программ могут запускаться на выполнение.

3. Правила оформления работы

1. Задания контрольной работы в соответствии с вариантом следует выполнить на ПК в рабочей книге Microsoft Excel;
2. Каждое выполненное задание подлежит оформлению. Требования к оформлению представлены ниже.

Оформление задания в рабочей книге Microsoft Excel

Вид листа рабочей книги после оформления задания

На листе рабочей книги должны быть представлены: задание, таблица обозначений, блок-схема алгоритма, текст программного кода с результатами расчета в виде рисунков *(рис. 75)*.

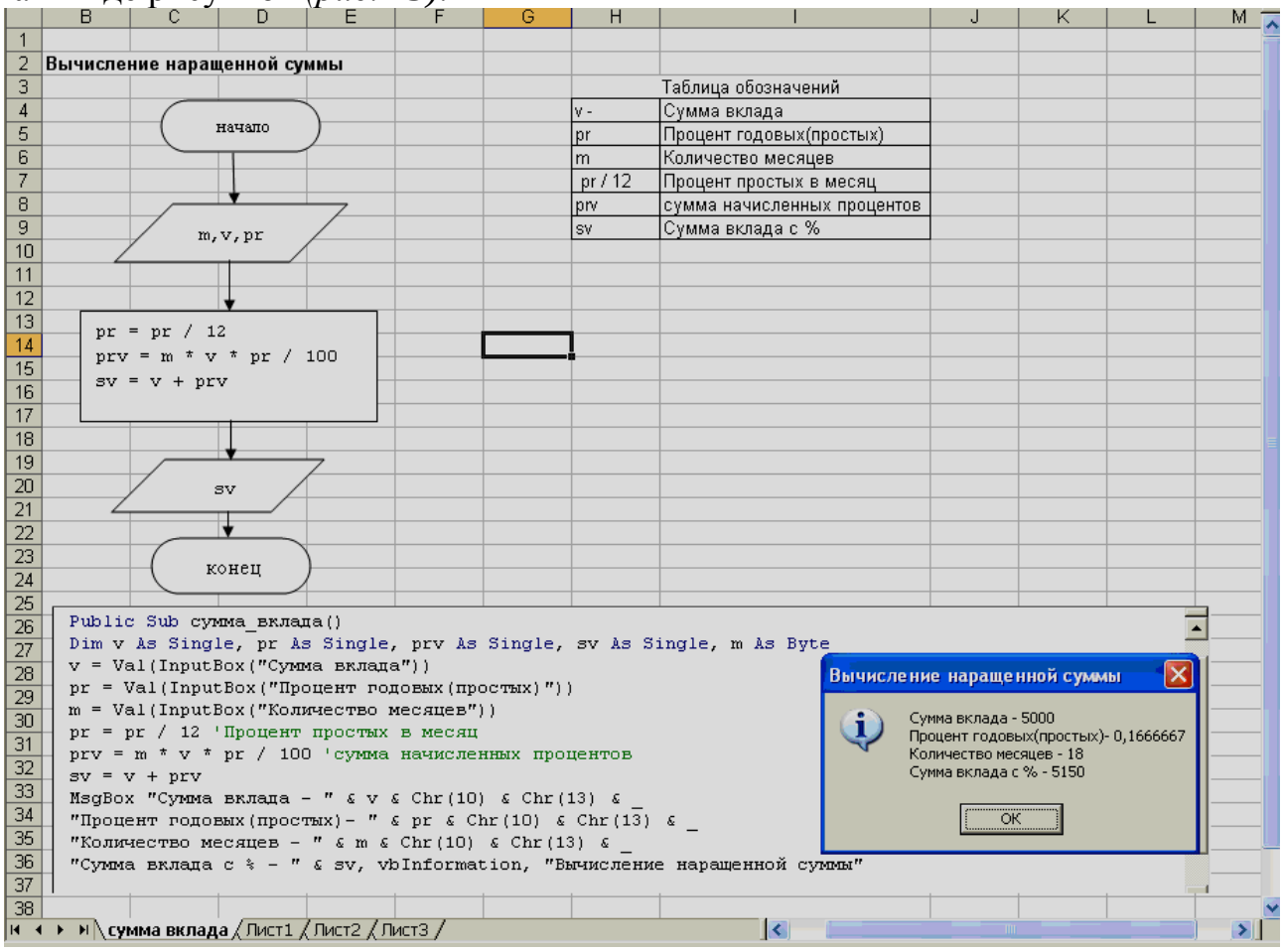


Рис. 75. Оформление задания

Вид программного модуля в редакторе VBA

Ниже представлен вид программного кода, записанного в Module1 и окна проекта (*рис. 76)*.

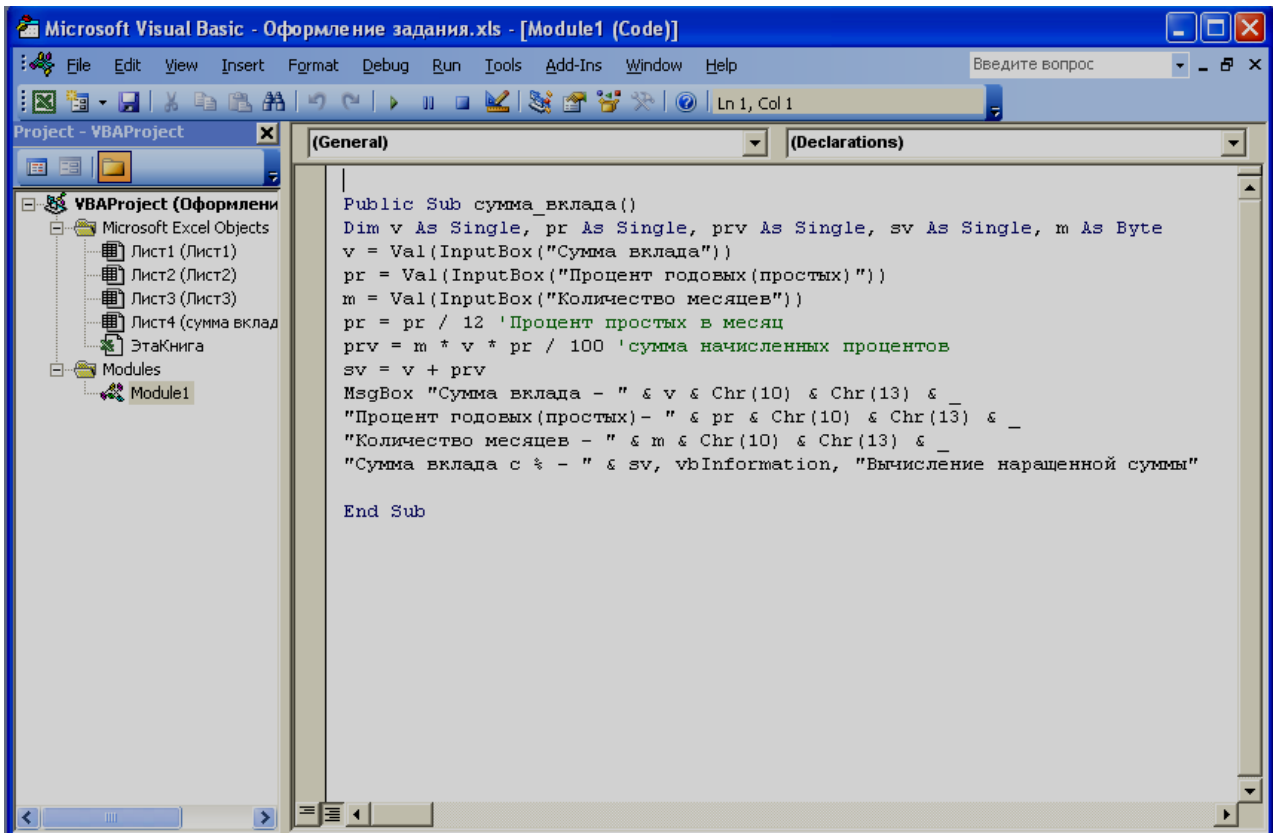


Рис. 76. Программный код

В **MS Office Excel 2007 и выше** создать блок-схему алгоритма можно средствами Microsoft Excel (кнопка 

для выполнения команды вставки фигуры (категория «Блок-схема») на вкладке «Вставка» в области «Иллюстрации»)

Примечание. Для создания блок-схемы алгоритма в **MS Office Excel 2003** используется панель рисования *(рис. 77)*, которая содержит автофигуры для отрисовки блок-схем.

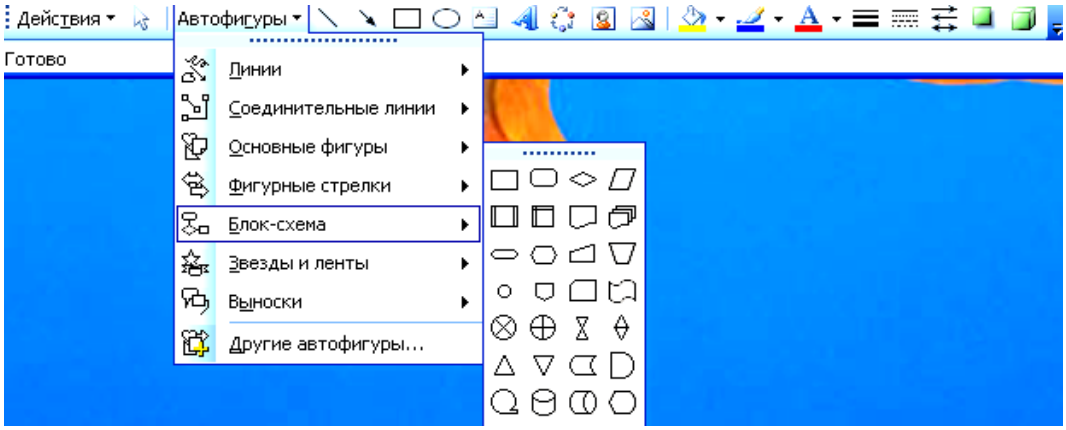


Рис. 77. Панель рисования в **MS Office Excel 2003**

4. Список литературы

3. Уокенбах, Д. Microsoft Office Excel 2007 Библия пользователя / Д. Уокенбах. – М.:ООО ”И.Д. Вильямс”, 2008. – 816 с.
4. Гарбер, Г.З. Основы программирования на Visual Basic и VBA в Excel 2007/ Г.З. Гарбер. – М.: СОЛОН – ПРЕСС,2008. – 192 с.
5. Гарнаев, А. Ю. Excel, VBA, Internet в экономике и финансах / А. Ю. Гарнаев. – СПб.: БХВ – Петербург, 2005. – 816 с.
6. Гарнаев, А. Ю. Самоучитель VBA. Технология создания пользовательских приложений на примерах / А. Ю. Гарнаев. – 2-е изд., перераб. и доп. – СПб.: БХВ – Петербург, 2004. – 500 с.