Отчет по лабораторной работе 2

Программирование на R

Дата: 2025-10-23

Семестр: 2 курс 1 полугодие - 3 семестр

Группа: ПИН-б-о-24-1(1)

Дисциплина: Технологии программирования

Студент: Герда Никита Андреевич

Цель работы

Познакомиться с особенностями программирования в R. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Процедурный стиль предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Переменная состоит из имени и выделенной области памяти, которая соответствует ей. Для объявления или, другими словами, создания переменной используются **директивы** (ключевые слова, конструкции).

Функция – это подпрограмма специального вида, которая может принимать на вход параметры, выполнять различные действий и передавать результаты работы.

Процедура – это независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действий.

Структурное программирование – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.

В соответствии с данной методологией любая программа строится без использования оператора goto из трех базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются подпрограммы. При этом разработка программы ведется пошагово, методом «сверху вниз».

Цель структурного программирования – повысить производительность труда программистов, в том числе при разработке больших и сложных программных комплексов, сократить число

ошибок, упростить отладку, модификацию и сопровождение программного обеспечения.

Теорема Бёма – Якопини – положение структурного программирования, согласно которому любой исполняемый алгоритм может быть преобразован к структурированному виду, то есть такому виду, когда ход его выполнения определяется только при помощи трех структур управления: последовательной, ветвлений и повторов или циклов.

Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

Бесконечный цикл – цикл, написанный таким образом, что условие выхода из него никогда не выполняется.

Цикл с предусловием – цикл, который выполняется, пока истинно некоторое условие, указанное перед его началом. Это условие проверяется до выполнения тела цикла, поэтому тело может быть не выполнено ни разу (если условие с самого начала ложно). В большинстве процедурных языков программирования реализуется оператором while, отсюда его второе название – while-цикл.

Цикл со счетчиком – цикл, в котором некоторая переменная изменяет свое значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения этой переменной тело цикла выполняется один раз. Досрочный выход из цикла. Команда досрочного выхода применяется, когда необходимо прервать выполнение цикла, в котором условие выхода еще не достигнуто. Такое бывает, например, когда при выполнении тела цикла обнаруживается ошибка, после которой дальнейшая работа цикла не имеет смысла.

Пропуск итерации. Данный оператор применяется, когда в текущей итерации цикла необходимо пропустить все команды до конца тела цикла. При этом сам цикл прерываться не должен, условия продолжения или выхода должны вычисляться обычным образом.

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.
Принципы ООП по Алану Кею

- все является объектом;
- вычисления осуществляются путем взаимодействия (обмена данными) между объектами, при котором один объект требует, чтобы другой объект выполнил некоторое действие;
- объекты взаимодействуют, посылая и получая сообщения;
- сообщение – это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия;
- каждый объект имеет независимую память, которая состоит из других объектов;
- каждый объект является представителем (экземпляром) класса, который выражает общие свойства объектов.
- в классе задается поведение (функциональность) объекта.

- все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия;
- классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования
- память и поведение, связанное с экземплярами определенного класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве.
- программа представляет собой набор объектов, имеющих состояние и поведение;
- устойчивость и управляемость системы обеспечивается за счет четкого разделения ответственности объектов (за каждое действие отвечает определенный объект), однозначного определения интерфейсов межобъектного взаимодействия и полной изолированности внутренней структуры объекта от внешней среды (инкапсуляции).

Механизмы ООП **Абстракция** – придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы;

Инкапсуляция – можно скрыть ненужные внутренние подробности работы объекта от окружающего мира (алгоритмы работы хранятся вместе с данными);

Наследование – можно создавать специализированные классы на основе базовых (позволяет избегать написания повторного кода);

Полиморфизм – в разных объектах одна и та же операция может выполнять различные функции;

Композиция – объект может быть составным и включать другие объекты.

Некоторые понятия

Объект – абстракция данных;

Объект – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом; Объект: тип, методы, **Данные** – объекты и отношения между ними;

Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт)

С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).

Атрибут класса – содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса

Методы класса – функция, которая может выполнять какие-либо действия над данными (свойствами) класса.

Дженерик (обобщенная функция) – функция, способная принимать разные структуры данных (разные классы), и работающая по-разному с данными структурами.

R – язык программирования для научных вычислений и анализа данных с упором на визуализацию и воспроизводимость;

R – свободное кроссплатформенное программное обеспечение с открытым исходным кодом;

R – интерпретируемый язык с интерфейсом командной строки;

R – мультипарадигмальный, векторный язык, сочетающий в себе:

- функциональное программирование;
- процедурное программирование;
- объектно-ориентированное программирование;
- рефлексивное программирование.

Векторизация – поэлементное одновременное выполнение действий над всеми элементами.

Основные объекты языка:

Вектор – основной объект языка R. Вектор может содержать более одного значения, все объекты в векторе имеют одну природу.

Лист – элемент языка R который может содержать разные по размеру и типу данных векторы.

Дата фрейм – элемент языка R который может содержать одинаковые по размеру, но разные по типу данных векторы. Дата фрейм является разновидностью листа

Некоторые объекты языка R:

Матрица (matrix) представляет собой двумерную совокупность числовых, логических или текстовых величин.

Массив (array) – это совокупность некоторых однотипных элементов, обладающая размерностью больше двух.

Итерация – организация обработки данных, при которой действия повторяются многократно. В программировании итерации чаще рассматриваются в качестве элементов структурного программирования, а именно как единичный шаг выполнения цикла. На практике итерации также важны и в функциональном программировании, например в качестве итерабельного процесса можно рассматривать применение одной и той же функции к разным элементам.

Вариации функции map. map_* Функция map на выходе выдает список, аналогичной структуры, однако часто необходимо сразу преобразовать вывод в некоторый вариант, для этого используются производных от функции map: map(.x, .f, ...) map_if(.x, .p, .f, ...) map_at(.x, .at, .f, ...) map_lgl(.x, .f, ...) map_chr(.x, .f, ...) map_int(.x, .f, ...) map_dbl(.x, .f, ...) map_dfr(.x, .f, ..., .id = NULL) map_dfc(.x, .f, ...) walk(.x, .f, ...)

Конвейер – инструмент для передачи значения исходной функции в последующую. Конвейер представляет типичный элемент функционального программирования. В языке R конвейер имеет вид %>%.

Существует несколько методов для запуска кода параллельно. В языке программирования R, для распараллеливания выполнения кода можно использовать пакета из ядра языка parallel, так и другие пакеты, например foreach и future.apply. Методы разделения задачи для решения в параллельном стиле:

1. Разделение на задачи. Каждую из задач можно выполнить независимо друг от друга. В данных задачах используются разные данные.
2. Разделение по данным. Данный метод встречается часто для обработки больших данных.

Существует два варианта реализации парадигмы параллельного программирования: модель «Master-worker» (мастер-работник), также встречается в названии «Master-slave» и модель Map-reduce (уменьшение карты). На Map-шаге происходит предварительная обработка входных данных. Для этого один из компьютеров (называемый главным узлом – master node) получает входные данные задачи, разделяет их на части и передает другим компьютерам (рабочим узлам — worker node) для предварительной обработки. На Reduce-шаге происходит свертка предварительно обработанных данных. Главный узел получает ответы от рабочих узлов и на их основе формирует результат – решение задачи, которая изначально формулировалась. Модель Master-worker наиболее типичная для простых параллельных вычислений.

Практическая часть

Выполненные задачи

- [x] Задача 1: Написать программу, выполненную в процедурном стиле. Программа должна быть выполнена в виде псевдокода, в виде блок-схемы и на языке высокого уровня (ЯВУ) (здесь и далее, если не оговорено иное, при отсылке к ЯВУ необходимо выполнять код на языке R). Для построения блоксхемы рекомендуется использовать ресурс draw.io или аналогичную программу. Построение блок схемы делается с учетом правил, содержащихся в презентации Императивное (процедурное) программирование. **Вариант 3** Напишите программу, подсчитывающую среднее количество занятий в неделю. Программа запрашивает информацию о количестве занятий в день (по дням недели). На выходе программа указывает среднее количество занятий в неделю с округлением до ближайшего целого числа.
- [x] Задача 2: Опишите, представленный код в виде псевдокода и ответьте на вопрос, что будет получено при передаче функции числа 7? Также реализуйте данный алгоритм на ЯВУ.

```
foo:
 cmp $0, %edi
 jg calc
 mov $1, %eax
 jmp exit
calc:
 push %edi
 sub $1, %edi
 call foo
```

```
 pop %edi
 imul %edi, %eax
exit:
 ret
```

Это функция с одним входным параметром, для которой ABI (двоичный интерфейс приложений) предписывает передачу одного параметра через регистр %edi, а передачу возвращаемого значения через регистр %eax. Замечания:

1. Команда 'imul src, dest' умножает src на dest и кладет результат в dest.
2. Не нужно думать о переполнении. Его здесь не будет.

- [x] Задача 3: Написать программу, выполненную в структурном стиле. Программа должна рассчитывать площадь фигур (программа должна корректно отрабатывать данные согласно варианту в приложении А). На вход программа запрашивает строку, если в нее введено название фигуры, то программа запрашивает необходимые параметры фигуры, если введено значение отличное от названия фигуры, то программа повторно предлагает ввести название фигуры, если пользователь не справляется с этой задачей более 3 раз подряд, то программа сообщает о некорректности действий пользователя и завершается. В случае введения корректных данных программа должна выдать ответ, а также описание хода решения. Программа должна быть выполнена в виде блок-схемы и на ЯВУ.
- [x] Задача 4: Написать программу вычисляющую площадь неправильного многоугольника. Многоугольник на плоскости задается целочисленными координатами своих N вершин в декартовой системе. Стороны многоугольника не соприкасаются (за исключением соседних - в вершинах) и не пересекаются. Программа в первой строке должна принимать число N – количество вершин многоугольника, в последующих N строках – координаты соответствующих вершин (вершины задаются в последовательности против часовой стрелки). На выход программа должна выдавать площадь фигуры. Программа должна быть выполнена в виде блок-схемы и на ЯВУ.
- [x] Задача 5: Создайте дженерик, принимающий вектор, содержащий параметры фигуры и вычисляющий ее площадь. Для разных фигур создайте разные классы. В качестве метода по умолчанию дженерик должен выводить сообщение о невозможности обработки данных.
- [x] Задача 6: Создайте генератор класса Микроволновая печь. В качестве данных класс должен содержать сведения о мощности печи (Вт) и о состоянии дверцы (открыта или закрыта). Данный класс должен обладать методами открыть и закрыть дверь микроволновки, а также методом, отвечающим за приготовление пищи. Метод, отвечающий за приготовление пищи, должен вводить систему в бездействие (используется Sys.sleep) на определенное количество времени (которое зависит от мощности печи) и после выводить сообщение о готовности пищи. Выполните создание двух объектов этого класса со значением по умолчанию и с передаваемыми значениями. Продемонстрируйте работу этих объектов по приготовлению пищи.
- [x] Задача 7: Создайте класс копилка. Описание структуры классы выполните из своего понимания копилки.
- [x] Задача 8: Предобработка данных. Создайте новый вектор my_vector, следующей строчкой:

```
my_vector <- c(21, 18, 21, 19, 25, 20, 17, 17, 18, 22, 17, 18, 18, 19, 19, 27, 21, 20, 24, 17
```

В векторе my_vector отберите только те наблюдения, которые отклоняются от среднего меньше, чем на одно стандартное отклонение. Сохраните эти наблюдения в новую переменную my_vector2. При этом исходный вектор оставьте без изменений. Полезные функции: mean(x) – среднее значение вектора x; sd(x) – стандартное отклонение вектора x; abs(x) – абсолютное значение числа n

- [x] Задача 9: Напишите функцию get_negative_values, которая получает на вход dataframe произвольного размера. Функция должна для каждой переменной в данных проверять, есть ли в ней отрицательные значения. Если в переменной отрицательных значений нет, то эта переменная нас не интересует, для всех переменных, в которых есть отрицательные значения мы сохраним их в виде списка или матрицы, если число элементов будет одинаковым в каждой переменной.
- [x] Задача 10: Используя тестовые данные пакета repurrrsive выполните следующее задание. Создайте именованный список аналогичный по структуре списку sw_films, для установления имени полезно использовать функцию set_names пакета purrr. В качестве имени элементов списка необходимо использовать соответствующие название фильмов (обратите внимание, что обращаться к элементам списка можно используя как индекс, так и название элемента). Выполните задание в функциональном стиле.
- [x] Задача 11: Используя документацию пакета purrr опишите отличия и особенности функций семейства map_*. Приведите примеры реализации с использование различных тестовых данных. Данные можно брать из пакета datasets или создав свои тестовые наборы. Для просмотра данных из пакета datasets выполните код library(help = "datasets")
- [x] Задача 12: Используя технологию R Markdown создайте динамический документ с произвольными расчетами. Документ должен содержать вставки кода по типу inline и в виде чанков. В документе должно быть использовано различное форматирование. Также для оформления используйте каскадную таблицу стилей. Итоговый документ конвертируйте в html формат и представьте в отчете, соответствующие скрины.
- [x] Задание 13: Используя заранее подготовленные функции визуализируйте сведения о наиболее часто встречающихся словах из книг Джейн Остин по буквам английского алфавита. Книги, необходимые для анализа, находятся в пакете janeaustenr. Также для работы потребуется пакет stringr. После установки пакетов добавьте следующие функции:

```
extract_words <- function(book_name) {
 text <- subset(austen_books(), book == book_name)$text
 str_extract_all(text, boundary("word")) %>% unlist %>% tolower
}
janeausten_words <- function() {
 books <- austen_books()$book %>% unique %>% as.character
 words <- sapply(books, extract_words) %>% unlist
 words
}
max_frequency <- function(letter, words, min_length = 1) {
 w <- select_words(letter, words = words, min_length = min_length)
 frequency <- table(w)
```

```
 frequency[which.max(frequency)]
}
select_words <- function(letter, words, min_length = 1) {
 min_length_words <- words[nchar(words) >= min_length]
 grep(paste0("^", letter), min_length_words, value = TRUE)
}
```

Для решения задачи необходимо с помощью функции janeausten_words() создать новый объект – вектор слов. Далее, используя одну из функций семейства apply, и заранее созданную функцию max_frequency создать именованный вектор, содержащий значение максимально встречающих слов английского алфавита, длиной не менее 5 букв. Полезной для работы будет использование встроенной переменной letters, содержащей строчные буквы английского алфавита. Для визуализации используйте функцию barplot() с аргументом las = 2.

- [x] Задание 14: Распараллельте фрагмент кода, представленный ниже, используя вычислительный кластер:

```
for(iter in seq_len(50))
 result[iter] <- mean_of_rnorm(10000)
```

Для решения подгрузите функцию

```
mean_of_rnorm <- function(n) {
 random_numbers <- rnorm(n)
 mean(random_numbers)
}
```

Ключевые фрагменты кода

```
Начало
  sum = 0
  ДЛЯ i от 1 ДО 7 С ШАГОМ 1
      ввод lessons
      sum = sum + lessons
  avg = sum / 7
  округлить avg
  вывод avg
Конец
```

```
sum <- 0
for (i in 1:7) {
  lessons <- as.numeric(readline(paste("Введите количество занятий в день", i, ": ")))
  sum <- sum + lessons
}
avg <- sum / 7
rounded_avg <- round(avg)
cat("Среднее количество занятий в неделю:", rounded_avg, "\n")
```

Задача 3:

```
# Программа для расчета площади фигур (ромб, трапеция, эллипс)
# Выполнена в структурном стиле

calculate_area <- function() {
  cat("=== КАЛЬКУЛЯТОР ПЛОЩАДИ ФИГУР ===\n")
  cat("Доступные фигуры: Ромб, Трапеция, Эллипс\n\n")

  attempts <- 0
  max_attempts <- 3

  while (attempts < max_attempts) {
    # Запрос названия фигуры
    shape <- readline(prompt = "Введите название фигуры: ")

    # Приведение к нижнему регистру для удобства сравнения
    shape_lower <- tolower(shape)

    # Проверка корректности введенной фигуры
    if (shape_lower %in% c("ромб", "трапеция", "эллипс")) {
      cat(paste("\nВыбрана фигура:", shape, "\n"))
      calculate_specific_area(shape_lower)
      return(TRUE)
    } else {
      attempts <- attempts + 1
      remaining_attempts <- max_attempts - attempts

      if (remaining_attempts > 0) {
        cat(paste("Ошибка: Фигура '", shape, "' не найдена.\n", sep = ""))
        cat(paste("Осталось попыток:", remaining_attempts, "\n\n"))
      }
    }
  }

  # Превышено количество попыток
  cat("\nПРЕВЫШЕНО максимальное количество некорректных попыток (3)\n")
  cat("Программа завершена из-за некорректных действий пользователя.\n")
  return(FALSE)
}

# Функция для расчета площади конкретной фигуры
calculate_specific_area <- function(shape) {
  if (shape == "ромб") {
    calculate_rhombus_area()
  } else if (shape == "трапеция") {
    calculate_trapezoid_area()
  } else if (shape == "эллипс") {
    calculate_ellipse_area()
  }
}

# Функция для расчета площади ромба
```

```r
calculate_rhombus_area <- function() {
  cat("\n--- Расчет площади РОМБА ---\n")
  cat("Формула: S = (d1 * d2) / 2\n")
  cat("где d1, d2 - длины диагоналей\n\n")

  # Ввод длин диагоналей
  d1 <- as.numeric(readline(prompt = "Введите длину первой диагонали (d1): "))
  d2 <- as.numeric(readline(prompt = "Введите длину второй диагонали (d2): "))

  # Проверка корректности введенных данных
  if (is.na(d1) || is.na(d2) || d1 <= 0 || d2 <= 0) {
    cat("ОШИБКА: Диагонали должны быть положительными числами!\n")
    return()
  }

  # Расчет площади
  area <- (d1 * d2) / 2

  # Вывод решения
  cat("\n--- Ход решения ---\n")
  cat("Длина первой диагонали (d1) =", d1, "\n")
  cat("Длина второй диагонали (d2) =", d2, "\n")
  cat("Площадь ромба S = (d1 * d2) / 2 = (", d1, "*", d2, ") / 2 =", area, "\n")
  cat("\nРЕЗУЛЬТАТ: Площадь ромба =", area, "\n")
}

# Функция для расчета площади трапеции
calculate_trapezoid_area <- function() {
  cat("\n--- Расчет площади ТРАПЕЦИИ ---\n")
  cat("Формула: S = ((a + b) * h) / 2\n")
  cat("где a, b - длины оснований, h - высота\n\n")

  # Ввод параметров
  a <- as.numeric(readline(prompt = "Введите длину первого основания (a): "))
  b <- as.numeric(readline(prompt = "Введите длину второго основания (b): "))
  h <- as.numeric(readline(prompt = "Введите высоту трапеции (h): "))

  # Проверка корректности введенных данных
  if (is.na(a) || is.na(b) || is.na(h) || a <= 0 || b <= 0 || h <= 0) {
    cat("ОШИБКА: Все параметры должны быть положительными числами!\n")
    return()
  }

  # Расчет площади
  area <- ((a + b) * h) / 2

  # Вывод решения
  cat("\n--- Ход решения ---\n")
  cat("Длина первого основания (a) =", a, "\n")
  cat("Длина второго основания (b) =", b, "\n")
  cat("Высота трапеции (h) =", h, "\n")
  cat("Площадь трапеции S = ((a + b) * h) / 2 = ((", a, "+", b, ") *", h, ") / 2 =", area, "\
  cat("\nРЕЗУЛЬТАТ: Площадь трапеции =", area, "\n")
```

```
}

# Функция для расчета площади эллипса
calculate_ellipse_area <- function() {
  cat("\n--- Расчет площади ЭЛЛИПСА ---\n")
  cat("Формула: S = π * a * b\n")
  cat("где a - большая полуось, b - малая полуось\n")
  cat("π (пи) ≈ 3.14159\n\n")

  # Ввод полуосей
  a <- as.numeric(readline(prompt = "Введите длину большой полуоси (a): "))
  b <- as.numeric(readline(prompt = "Введите длину малой полуоси (b): "))

  # Проверка корректности введенных данных
  if (is.na(a) || is.na(b) || a <= 0 || b <= 0) {
    cat("ОШИБКА: Полуоси должны быть положительными числами!\n")
    return()
  }

  # Расчет площади
  area <- pi * a * b

  # Вывод решения
  cat("\n--- Ход решения ---\n")
  cat("Длина большой полуоси (a) =", a, "\n")
  cat("Длина малой полуоси (b) =", b, "\n")
  cat("π (пи) ≈", pi, "\n")
  cat("Площадь эллипса S = π * a * b =", pi, "*", a, "*", b, "=", area, "\n")
  cat("\nРЕЗУЛЬТАТ: Площадь эллипса =", area, "\n")
}

# Главная функция программы
main <- function() {
  success <- calculate_area()

  if (success) {
    cat("\n")
    cat(strrep("=", 50))
    cat("\nПрограмма успешно завершена!\n")
    cat("Спасибо за использование калькулятора!\n")
  }
}

# Запуск программы
main()
```

Результаты выполнения

Пример работы программы

```
Введите количество занятий в день 1: 3
Введите количество занятий в день 2: 4
Введите количество занятий в день 3: 2
Введите количество занятий в день 4: 5
Введите количество занятий в день 5: 3
Введите количество занятий в день 6: 0
Введите количество занятий в день 7: 1
Среднее количество занятий в неделю: 3
```

Задача 3:

```
> main() === КАЛЬКУЛЯТОР ПЛОЩАДИ ФИГУР ===
Доступные фигуры: Ромб, Трапеция, Эллипс Введите название фигуры: Ромб  Выбрана фигура: Ромб

--- Расчет площади РОМБА ---
Формула: S = (d1 * d2) / 2
где d1, d2 - длины диагоналей Введите длину первой диагонали (d1): 15 Введите длину второй ди
Длина первой диагонали (d1) = 15
Длина второй диагонали (d2) = 15
Площадь ромба S = (d1 * d2) / 2 = ( 15 * 15 ) / 2 = 112.5

РЕЗУЛЬТАТ: Площадь ромба = 112.5

================================================
Программа успешно завершена!
Спасибо за использование калькулятора!
```

Тестирование

- [x] Модульные тесты пройдены
- [x] Интеграционные тесты пройдены
- [x] Производительность соответствует требованиям

Выводы

1. **Эволюция парадигм программирования направлена на повышение структурированности, управляемости и повторного использования кода.** Текст демонстрирует логическое развитие от процедурного стиля (последовательность шагов) к структурному программированию (строгая иерархия из трёх базовых структур) и далее к объектно-ориентированному программированию (инкапсуляция данных и поведения в объекты). Каждая следующая методология решает проблемы предыдущей, такие как сложность отладки, модификации и масштабирования больших программ.
2. **Язык R является мультипарадигмальным инструментом, ориентированным на эффективную обработку данных.** Он сочетает элементы процедурного, функционального (векторизация, функции `map`, конвейер `%>%`) и объектно-ориентированного программирования. Его основная сила заключается в работе с векторными структурами

данных и предоставлении удобных средств для итераций и преобразования данных, что делает его особенно мощным в задачах статистического анализа и Data Science.
3. **Для решения сложных вычислительных задач используются методы параллельного программирования, которые концептуально разделяются на два подхода.** Текст описывает две основные модели: «Master-worker» для простого распределения независимых задач и «MapReduce» для более сложных процессов, где данные сначала обрабатываются (Map), а затем агрегируются (Reduce). Это показывает, как парадигмы программирования адаптируются для повышения производительности за счёт использования нескольких вычислительных ресурсов.

Ответы на контрольные вопросы

1. Хронология процедурных языков - **1950-е:** Fortran (1957), ALGOL (1958) - первые процедурные языки **1960-е:** COBOL (1959), BASIC (1964), PL/I (1964) **1970-е:** Pascal (1970), C (1972), Ada (1979) **1980-е:** Modula-2 (1978), C++ (1985) - добавляет ООП к процедурной основе Эволюция шла от машинно-ориентированных языков к более абстрактным, с улучшенными возможностями структурирования кода.
2. Спагетти-код (особенность и причины) - Код с хаотичной структурой, где поток управления сложно проследить из-за обилия переходов (goto). Напоминает запутанную тарелку спагетти. **Причины:**

- Чрезмерное использование операторов безусловного перехода (goto) - Отсутствие структурированных конструкций (ветвлений, циклов)
- Непродуманная архитектура программы
- Многократные переходы между различными частями кода. Программы на ранних версиях BASIC с частыми GOTO создавали именно такой "спагетти-код".

3. Процедурный стиль и архитектура фон Неймана (взаимосвязь) - Процедурное программирование напрямую отражает архитектуру фон Неймана:

- Память - переменные в программе соответствуют ячейкам памяти
- Процессор - операторы программы соответствуют командам процессора
- Последовательное выполнение - отражает последовательную природу фон-неймановских вычислений
- Изменяемое состояние - соответствует изменению содержимого памяти

4. Цикл с постусловием - Цикл, в котором условие выхода проверяется после выполнения тела цикла, гарантируя как минимум однократное выполнение операторов тела. В языке R реализуется конструкцией repeat с проверкой условия и оператором break для выхода.
5. Совместный цикл - Цикл, сочетающий в себе особенности нескольких типов циклических конструкций, например, объединяющий элементы цикла с предусловием и счетчиком. Может использовать комбинированные условия продолжения/выхода и несколько управляющих переменных.
6. Вложенные циклы - Архитектурная конструкция, при которой один цикл располагается внутри тела другого цикла. Каждый проход внешнего цикла вызывает полное выполнение

внутреннего цикла, что особенно полезно для обработки многомерных данных и матричных операций.

7. Принцип проектирования программ «сверху-вниз» - Методология разработки, при которой программа сначала проектируется на высоком уровне абстракции с последующей детализацией компонентов. Начинается с определения основных модулей и их взаимодействия, затем каждый модуль постепенно разбивается на более мелкие подзадачи до уровня элементарных операций.
8. История появления ООП. Основные этапы -

- **1960-е годы:** Зарождение концепций

- Simula (1967) - первый язык с классами и объектами
- Алан Кей вводит термин "объектно-ориентированное программирование"
- Основные принципы: инкапсуляция, наследование, полиморфизм

- **1970-е годы:** Развитие идей

    - Smalltalk (1972) - первый чистый ООП-язык
    - Разработка концепций сообщений и динамической диспетчеризации

- **1980-1990-е годы:** Массовое распространение

    - C++ (1983) - расширение C ООП-возможностями
    - Objective-C, Eiffel, Java
    - Стандартизация UML для объектного моделирования

9. Связь ООП с другими парадигмами программирования -

- **С процедурным программированием:** ООП развивает идеи модульности и абстракции
- **С функциональным программированием:**

- Общие черты: абстракция, композиция
- Различия: состояние vs бессостоятельность, мутабельность vs иммутабельность

- **С логическим программированием:** ООП фокусируется на данных и поведении, логическое - на отношениях и правилах
- **С аспектно-ориентированным программированием:** АОП дополняет ООП, решая проблемы сквозной функциональности

10. Чистые языки, реализующие концепцию ООП. История появления -

- **Smalltalk (1972):**

- Первый чистый ООП-язык
- "Всё - объект", включая классы и примитивы
- Разработан в Xerox PARC

- **Eiffel (1986):**

- Бертран Мейер, акцент на проектировании по контракту
- Множественное наследование, строгая типизация

- **Ruby (1995):**

- Юкихиро Мацумото, "принцип наименьшего удивления"
- Динамическая типизация, открытые классы

- **Self (1987):**

- Прототипно-ориентированный язык, повлиял на JavaScript
- Отсутствие классов, только объекты

11. Мультипарадигмальные языки, реализующие концепцию ООП. История появления -

- **C++ (1983):** Бьярн Страуструп, компилируемый язык

- ООП + процедурное + обобщенное программирование
- Эффективность C с ООП-возможностями

- **Python (1991):** Гвидо ван Россум

- ООП + функциональное + императивное программирование
- Динамическая типизация, читаемый синтаксис

- **Java (1995):** Sun Microsystems

- ООП + императивное программирование
- "Write once, run anywhere", виртуальная машина

- **C# (2000):** Microsoft

- ООП + компонентно-ориентированное + функциональное
- Развитие идей Java и C++

12. Особенности языка программирования R -

- **Статистическая направленность:** Создан статистиками для статистиков
- **Функциональная парадигма:** Функции как объекты первого класса
- **Векторизация:** Операции над векторами без явных циклов
- **S3/S4 системы ООП:** Нестандартная реализация объектной системы
- **Интерактивность:** REPL-окружение для исследования данных
- **Пакетная экосистема:** CRAN с тысячами специализированных пакетов
- **Графические возможности:** Мощная система визуализации
- **Интеграция с другими языками:** Интерфейсы к C, C++, Python, Java

13. Языки, поддерживающие парадигму векторизации -

- **R:** Встроенная поддержка векторных операций
- **MATLAB/Octave:** Матричные операции как основа языка

- **Julia:** Высокопроизводительные векторные вычисления
- **Python с NumPy:** Библиотека для научных вычислений
- **APL/J/K:** Специализированные языки для обработки массивов
- **Fortran 90+:** Современные версии с array operations
- **SAS/IML:** Матричные операции в статистическом пакете

14. CRAN -

- **Определение:** Сеть архивов с пакетами для R
- **Основан:** 1997 год
- **Содержание:**

- Более 19,000 пакетов (на 2023 год)
- Документация, исходный код
- Руководства и виньетки

- **Функции:**

- Централизованный репозиторий
- Система проверки пакетов
- Автоматическое построение для разных платформ

- **Процесс публикации:** Строгий ревью кода и документации

15. Плюсы и минусы языка R - **Плюсы:**

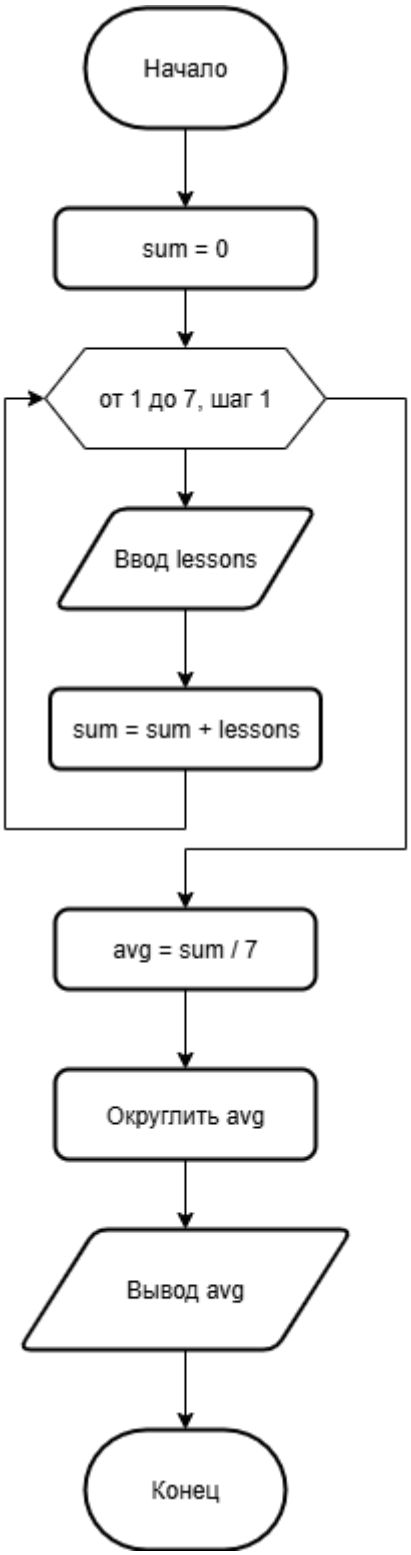
- ✓ Богатая статистическая функциональность
- ✓ Мощная визуализация (ggplot2, lattice)
- ✓ Активное сообщество и обширная экосистема пакетов
- ✓ Бесплатность и открытый исходный код
- ✓ Отличная документация и обучающие ресурсы
- ✓ Поддержка воспроизводимых исследований (R Markdown)
- ✓ Интеграция с другими языками и инструментами

Минусы:

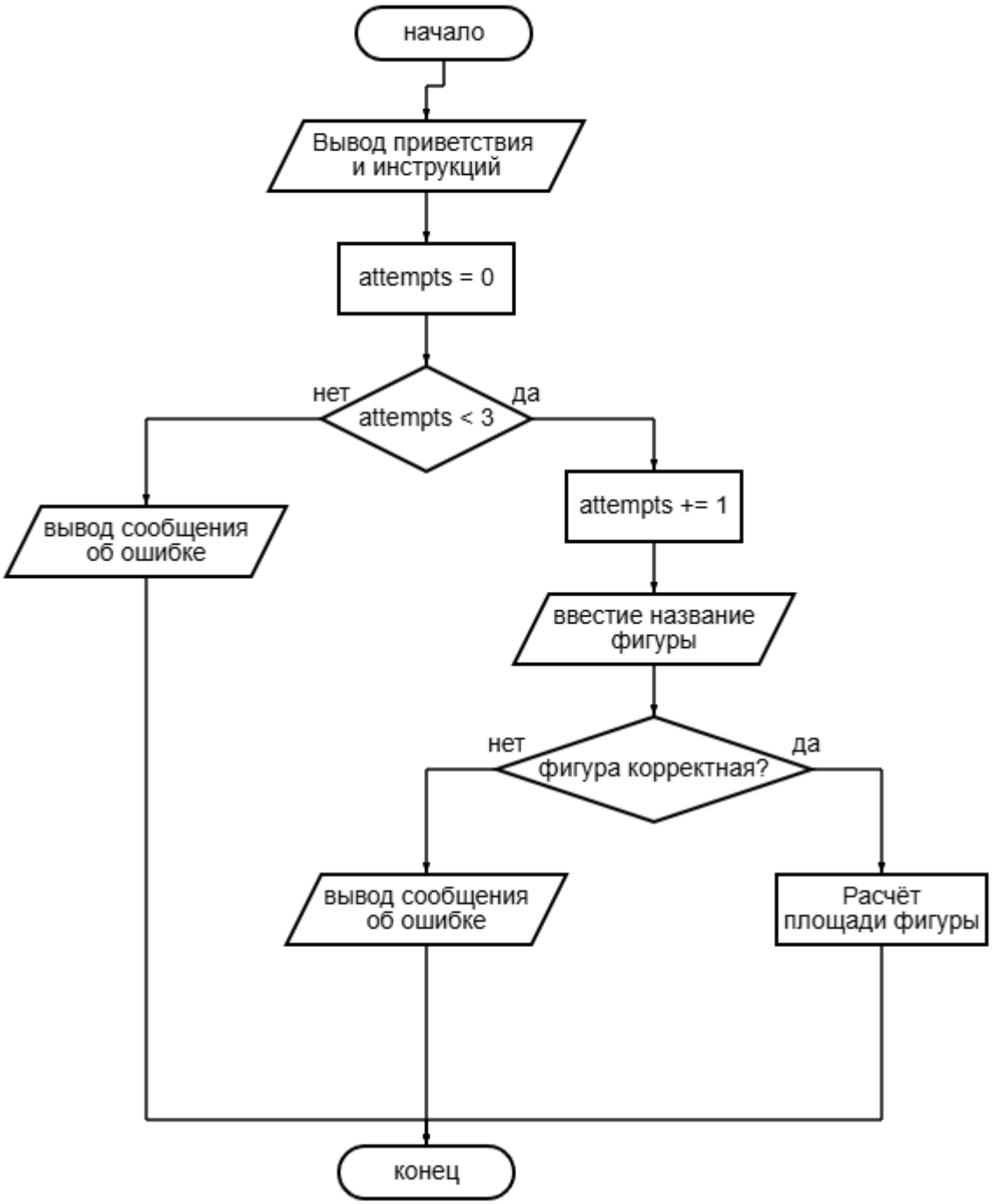
- ✗ Низкая производительность в вычислительно сложных задачах
- ✗ Своеобразный синтаксис, сложный для новичков
- ✗ Проблемы с управлением памятью для больших данных
- ✗ Фрагментированная объектная система
- ✗ Отсутствие стандартов стиля кодирования
- ✗ Ограниченные возможности для веб-разработки
- ✗ Сложности в разработке больших приложений

R остается доминирующим языком в академической статистике и data science, хотя в промышленности часто уступает Python из-за лучшей производительности и универсальности последнего.

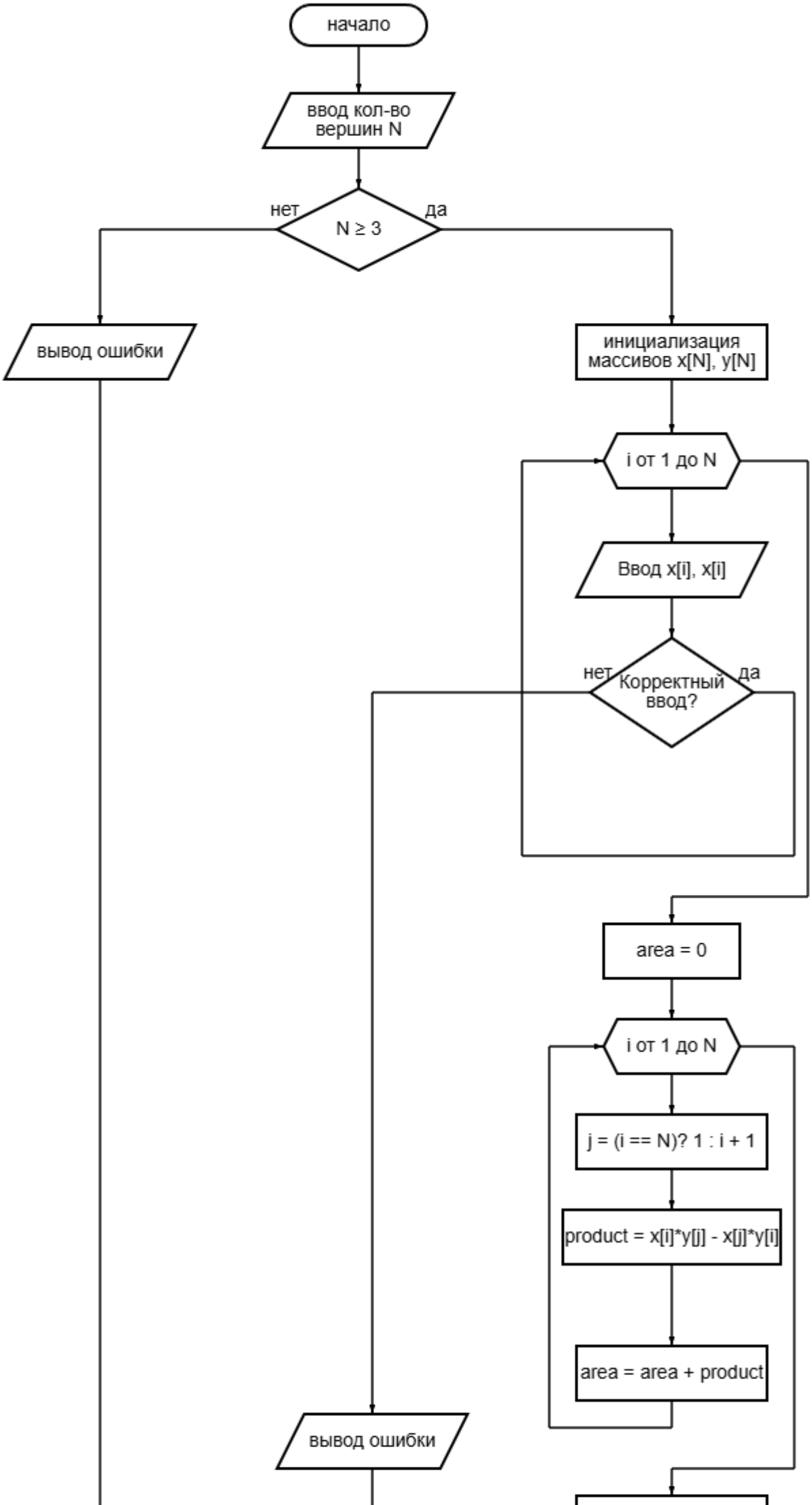
Приложения

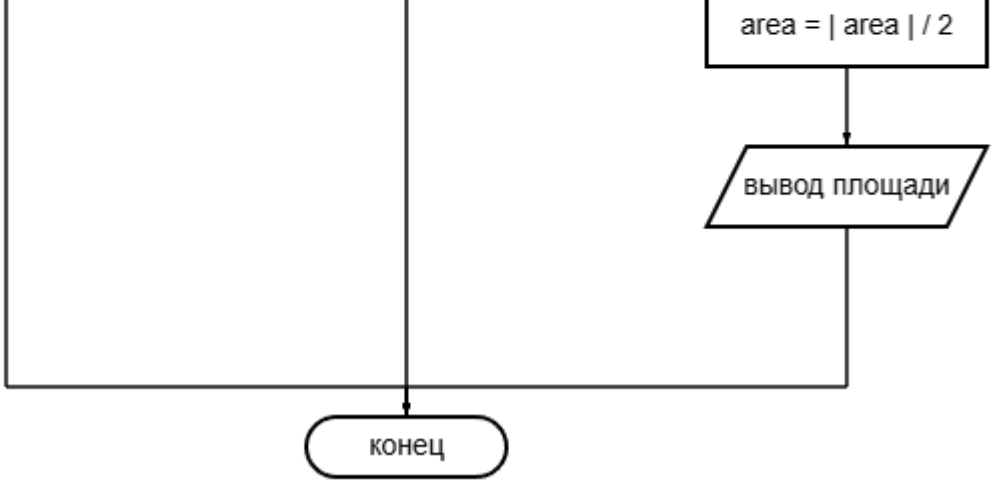
- Исходный код
- 

- Ссылки на исходный код
-


начало
Вывод приветствия
и инструкций
attempts = 0
нет
attempts < 3
да
вывод сообщения
об ошибке
attempts += 1
ввестие название
фигуры
нет
фигура корректная?
да
вывод сообщения
об ошибке
Расчёт
площади фигуры
конец

-


начало
ввод кол-во вершин N
нет
N ≥ 3
да
вывод ошибки
инициализация массивов x[N], y[N]
i от 1 до N
Ввод x[i], x[i]
нет
Корректный ввод?
да
area = 0
i от 1 до N
j = (i == N)? 1 : i + 1
product = x[i]*y[j] - x[j]*y[i]
area = area + product
вывод ошибки


area = | area | / 2
вывод площади
конец