Московский государственный технический университет

имени Н.Э. Баумана

Факультет «Информатика и вычислительная техника»

Кафедра «Компьютерные системы и сети»

Т.Н. Ничушкина

**Разработка программ линейной, разветвленной и циклической структуры**

Учебно-методическое пособие

Москва

(С) 2023 МГТУ им. Н.Э. БАУМАНА

УДК 004.432

Рецензент: доцент кафедры ИУ7 к.т.н. Новик Н.В..,

Т.Н. Ничушкина

Разработка программ линейной, разветвленной и циклической структуры. Учебно-методическое пособие по выполнению Домашнего задания № 1 по дисциплине «Информатика» **-** М.: МГТУ имени Н.Э. Баумана, 2020. 18 с.

Учебно-методическое пособие содержат теоретические сведения об особенностях разработки программ линейной, ветвящейся и циклической структуры на языке С++. Издание предназначено для выполнения задания № 1 первого модуля по дисциплине «Информатика»

Для студентов МГТУ имени Н.Э. Баумана, обучающихся по программе бакалавриата направления «Математика и компьютерные науки»

*Рекомендовано Научно-методическим советом МГТУ им. Н.Э. Баумана*

Ничушкина Татьяна Николаевна

Разработка программ линейной, разветвленной и циклической структуры.

**Учебно-методическое пособие по выполнению домашнего задания № 1**

**по дисциплине «Информатика»**

© Т.Н. Ничушкина, 2023

© МГТУ имени Н.Э. Баумана, 2023

# 

Оглавление

[**Введение 4**](#_Toc142649423)

[**1. Теоретическая часть 6**](#_Toc142649424)

[**1.1 Понятие алгоритма. Основные алгоритмические структуры 6**](#_Toc142649425)

[**1.2 Схемы алгоритма и псевдокоды 7**](#_Toc142649426)

[**1.3 Метод пошаговой детализации 13**](#_Toc142649427)

[**2. Практическая часть 18**](#_Toc142649428)

[**2.1. Указания по выполнению домашнего задания 18**](#_Toc142649429)

[**2.2 Требования к оформлению 32**](#_Toc142649430)

[**2.3 Требования к защите 33**](#_Toc142649431)

[**2.4 Пример выполнения задания 34**](#_Toc142649432)

[**2.5 Примеры вариантов домашнего задания 39**](#_Toc142649433)

[**Заключение 42**](#_Toc142649434)

[**Список литературы 43**](#_Toc142649435)

[**ПРИЛОЖЕНИЕ А. Пример титульного листа 44**](#_Toc142649436)

# Введение

Изящное и технически грамотное написание компьютерных программ предполагает не только хорошее владение средствами их разработки, но и достаточно развитое алгоритмическое мышление. Сложившаяся на протяжении десятилетий практика обучения программированию говорит о том, что именно недостаток алгоритмического мышления – основная причина неудач студентов в процессе освоения программирования.

Алгоритмическое мышление как умение выстраивать логически безупречную последовательность действий на пути к решению задачи, можно и нужно развивать. Опыт показывает, что одним из наиболее действенных способов достижения этого является проработка алгоритмов ряда небольших, но полезных программ, приводящая к формированию базы приемов программирования. Таких приемов сравнительно немного, но их накопление, обобщение и умение осознанно применять на практике позволяет студентам научиться писать программы.

Цель пособия – освоение базовых приемов алгоритмизации задач, углубление знаний по использованию базовых типов данных и формирование навыков разработки программ с использованием этих алгоритмов и данных на примере языка С++.

В учебно-методическом пособии к домашнему заданию 1 по курсу «Информатика» представлены базовые алгоритмы, без изучения которых знания, умения и навыки в области программирования не будут отличаться полнотой.

Настоящее учебно-методическое пособие предназначено для студентов 1 курса, изучающих дисциплину «Информатика» на кафедре ИУ-6 МГТУ им. Н.Э. Баумана в соответствии с программой подготовки бакалавров направления «Математика и компьютерные науки». Основная задача, которую автор ставил перед собой, – помочь студентам в освоении базовых приемов алгоритмизации задач, которые непосредственно востребуются при выполнении лабораторных работ и домашних заданий по названной учебной дисциплине.

# Теоретическая часть

**Цель домашнего задания**: освоение базовых приемов алгоритмизации задач, углубление знаний по использованию базовых типов данных и формирование навыков разработки программ с использованием этих алгоритмов и данных на примере языка С++.

## 1.1 Понятие алгоритма. Основные алгоритмические структуры

*Алгоритмом* называют *формально описанную* последовательность действий, которые необходимо выполнить для получения требуемого результата.

Различают последовательности действий (вычислений) линейной, разветвленной и циклической структуры.

*Линейная* структура процесса вычислений предполагает, что для получения результата необходимо выполнить некоторые операции в определенной последовательности. Например, для определения площади треугольника по формуле Герона необходимо сначала определить полупериметр треугольника, а затем по формуле рассчитать его площадь.

*Разветвленная* структура процесса вычислений предполагает, что конкретная последовательность операций зависит от значений одного или нескольких параметров. Например, если дискриминант квадратного уравнения не отрицателен, то уравнение имеет два корня, а если отрицателен, то действительных корней оно не имеет.

*Циклическая* структура процесса вычислений предполагает, что для получения результата некоторые действия необходимо выполнить несколько раз. Например, для того, чтобы получить таблицу значений функции на заданном интервале изменения аргумента с заданным шагом, необходимо соответствующее количество раз определить следующее значение аргумента и посчитать для него значение функции.

В общем случае любой цикл можно разделить на 4 этапа:

* *подготовка цикла* – задание начальных значений переменным цикла перед первым его выполнением;
* *тело цикла* – действия, повторяемые в цикле для различных значений переменных цикла;
* *изменение значений переменных цикла* при каждом новом его выполнении;
* *управление циклом* – проверка условия продолжения работы цикла.

Процессы вычислений циклической структуры в свою очередь делятся на три группы:

* *счетные циклы* или *циклы с заданным количеством повторений* – циклические процессы, для которых количество повторений известно;
* *итерационные циклы* – циклические процессы, завершающиеся по достижении или нарушении некоторых условий;
* *поисковые* *циклы* – циклические процессы поиска некоторой информации в таблицах, выход из которых происходит при нахождении необходимой информации или по завершению процесса поиска, если необходимая информация не найдена.

Часто в программировании решаются задачи, в которых требуется использовать цикл внутри другого цикла. Такие циклы называют *вложенными*. Правила организации внешнего и внутреннего циклов такие же, как и для простого цикла. Однако, при построении вложенных циклов необходимо соблюдать следующее дополнительное условие:

- все операторы внутреннего цикла должны полностью лежать в теле внешнего цикла,

- циклы ни в коем случае не могут пересекаться.

## 1.2 Схемы алгоритма и псевдокоды

Формальное описание алгоритмов осуществляют с использованием схем алгоритмов и псевдокодов. На изображение *схем алгоритмов* существует ГОСТ 19.701–90, согласно которому каждой группе действий ставится в соответствие блок особой формы. Некоторые, часто используемые обозначения приведены в таблице 1.

**Таблица 1 – Основные блоки схем алгоритмов**

|  |  |  |
| --- | --- | --- |
| Название блока | Обозначение | Назначение блока |
| 1. Терминатор |  | Начало, завершение программы или подпрограммы |
| 2. Процесс |  | Обработка данных (вычисления, пересылки и т.п.) |
| 3. Данные |  | Операции ввода-вывода |
| 4. Решение |  | Ветвления, выбор, итерационные и поисковые циклы |
| 5. Подготовка |  | Счетные циклы |
| 6. Граница цикла |  | Любые циклы |
| 7. Предопределенный процесс |  | Вызов процедур |
| 8. Соединитель |  | Маркировка разрывов линий |
| 9. Комментарий |  | Пояснения к операциям |

При разработке алгоритма каждое действие обозначают соответствующим блоком, показывая их последовательность линиями, идущими слева направо и сверху вниз.

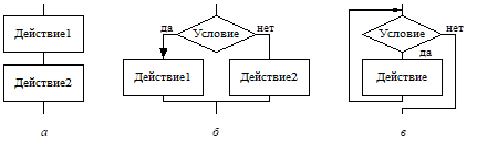
*Для удобства чтения схемы желательно, чтобы линия входила в блок сверху, а выходила снизу. Если направление линии отлично от стандартного (сверху-вниз и слева-направо), то в конце линии ставится стрелка, в противном случае ее можно не ставить.*

Если схема алгоритма не умещается на листе, то используют соединители. При переходе на другой лист или получении управления с другого листа в комментарии указывается номер листа, например, «с листа 3», «на лист 1».

Доказано, что для записи любого, сколь угодно сложного алгоритма достаточно трех базовых управляющих структур:

*следование* – обозначает последовательное выполнение действий (см. рисунок 1, *а*);

*ветвление* – соответствует выбору одного из двух вариантов действий (см. рисунок 1, *б*);

*цикл-пока* – определяет повторение действий, пока не будет нарушено некоторое условие, выполнение которого проверяется в начале цикла (см. рисунок 1, *в*).

**Рисунок 1 -** Базовые алгоритмические структуры: следование (*а*), ветвление (*б*) и цикл-пока (*в*)

Помимо базовых структур используют еще три дополнительные структуры, производные от базовых:

* *выбор* – обозначающий выбор одного варианта из нескольких в зависимости от значения некоторой величины (см. рисунок 2, *а*);
* *цикл-до* – обозначающий повторение некоторых действий до выполнения заданного условия, проверка которого осуществляется после выполнения действий в цикле (см. рисунок 2, *в*);
* *цикл с заданным числом повторений* (*счетный цикл*) – обозначающий повторение некоторых действий указанное количество раз (см. рисунок 2, *д*).

На рисунке 2 (*б*, *г* и *е*) показано, как каждая из дополнительных структур может быть реализована через базовые структуры.



**Рисунок 2 -** Дополнительные структуры и их реализация через базовые структуры: выбор (*а-б*), цикл-до (*в*-*г*) и цикл с заданным числом повторений (*д*-*е*)

Перечисленные структуры были положены в основу ***структурного* *программирования*** – технологии, которая представляет собой набор рекомендаций по уменьшению количества ошибок в программах. В том случае, если в схеме алгоритма отсутствуют другие варианты передачи управления, алгоритм называют *структурным*, подчеркивая, что он построен с учетом рекомендаций структурного программирования.

Схема алгоритма детально отображает особенности разработанного алгоритма. Иногда такой высокий уровень детализации не позволяет выделить основные подзадачи, составляющие суть алгоритма. В этих случаях для описания алгоритма используют псевдокод.

*Псевдокод* – формализованное текстовое описание алгоритма, он базируется на тех же основных структурах, что и структурные схемы алгоритма. Описать на псевдокоде неструктурный алгоритм нельзя.

Для каждой структуры используют свою форму описания. В специальной литературе были предложены несколько вариантов псевдокодов. Один из них приведен в таблице 2.

**Таблица 2 – Варианты псевдокодов для базовых структур**

|  |  |  |  |
| --- | --- | --- | --- |
| Структура | Псевдокод | Структура | Псевдокод |
| 1. Следование | <действие 1>  <действие 2> | 4. Выбор | **Выбор** <код>  <код1>: <действие 1>  <код2>: <действие 2>  ...  **Все-выбор** |
| 2. Ветвление | **Если** <условие>  **то** <действие 1>  **иначе <**действие 2>  **Все-если** | 5. Цикл с заданным количеством повторений | **Для** <индекс> = <n>, <k>, <h>  <действие>  **Все-цикл** |
| 3. Цикл-пока | **Цикл-пока** <условие>  <действие>  **Все-цикл** | 6. Цикл-до | **Выполнять**  <действие>  **До** <условие> |

**Пример 1.** Разработать алгоритм определения наибольшего общего делителя двух натуральных чисел.

Существует несколько способов определения наибольшего общего делителя двух натуральных чисел. Самым простым из них является так называемый «алгоритм Евклида». Суть этого метода заключается в последовательной замене большего из чисел на разность большего и меньшего. Вычисления заканчиваются, когда числа становятся равны.

Например:

|  |  |  |  |
| --- | --- | --- | --- |
| а) | **A** **B** | б) | **A** **B** |
|  | 225 125 |  | 13 4 |
|  | 225-125=100 125 |  | 13-4=9 4 |
|  | 100 125-100=25 |  | 9-4=5 4 |
|  | 100-25=75 25 |  | 5-4=1 4 |
|  | 75-25=50 25 |  | 1 4-1=3 |
|  | 50-25=25 = 25 |  | 1 3-1=2 |
|  |  |  | 1 = 2-1=1 |

Программа должна начинаться с ввода чисел. Заметим, что любой ввод данных пользователем должен сопровождаться *запросом на ввод*, чтобы пользователь знал, чего от него ждет программа после запуска. На схеме алгоритма и при записи псевдокодов этот запрос часто не указывают.

В основе алгоритма лежит циклический процесс, количество повторений которого заранее не известно (итерационный). Условие выхода из цикла – получение одинаковых чисел. Поскольку нельзя исключить, что пользователь введет равные числа, проверку будем осуществлять на входе в цикл, т. е. имеет смысл использовать цикл-пока. Если числа не равны, то при каждом проходе цикла одно из чисел (большее) должно заменяться разностью большего и меньшего. Для реализации этой замены потребуется описать оба варианта, т.е. использовать ветвление с проверкой, какое из чисел больше. После выхода из цикла можно выводить пользователю любое из двух полученных чисел, так как они равны между собой.

На рисунке 3 показана схема алгоритма, а ниже приведено его описание на псевдокоде.



**Рисунок 3 -** Схема алгоритма Евклида

**Алгоритм Евклида – псевдокод:**

Ввести A, B

**цикл-пока** A≠B

**если** A>B

**то** A:=A-B

**иначе** B:=B-A

**все-если**

**все-цикл**

Вывести А

**Конец алгоритма**

Алгоритмы более сложных программ разрабатывают, используя специальный метод.

## 1.3 Метод пошаговой детализации

С использованием *метода пошаговой детализации* разработку алгоритмов выполняют поэтапно. Этот метод еще часто называют ***декомпозицией*.** На первом этапе описывают решение поставленной задачи, выделяя подзадачи и считая их решенными. На следующем – аналогично описывают решение подзадач, формулируя уже подзадачи следующего уровня. Процесс продолжают, пока в процессе декомпозиции не доходят до подзадач, алгоритмы решения которых очевидны. При этом, описывая решение каждой задачи, желательно использовать не более одной-двух конструкций, таких как цикл или ветвление, чтобы четче представить себе структуру программы.

**Пример 2.** Разработать программу, которая с заданной точностью ε находит значение аргумента x по заданному значению функции y при известном значении n.

При n>1 данная функция является монотонно возрастающей. Следовательно, для нахождения значения x можно применить метод половинного деления. Суть метода заключается в следующем. Вначале определяют отрезок , такой что . Затем делят его пополам и определяют, в какой половине отрезка находится x, для чего сравнивают f(xt) и y. Полученный отрезок опять делят пополам и так до тех пор, пока разность x1 и x2 не станет меньше заданного значения ε.

Для разработки алгоритма используем метод пошаговой детализации.

*Шаг 1.*Определяем общую структуру программы:

**Программа:**

Ввести y, n, eps.

Определить x.

Вывести x, y.

**Конец.**

*Шаг 2.* Детализируем операцию определения x:

**Определить x:**

Определить x1, такое что f(x1)≤y.

Определить x2, такое что f(x2)≥y.

Определить x на интервале [x1,x2].

**Все.**

*Шаг 3.* Детализируем операцию определения x1. Значение x1 должно быть подобрано так, чтобы выполнялось условие f(x1) ≤ y. Известно, что x>0, следовательно, можно взять некоторое значение x, например, x1=1, и последовательно уменьшая его, например, в два раза определить значение x1, удовлетворяющее данному условию:

**Определить x1:**

x1:=1

**цикл-пока** f(x1)>y

x1:=x1/2

**все-цикл**

**Все.**

*Шаг 4.* Детализируем операцию определения x2. Значение x2 находим аналогично x1, но исходное значение будем увеличивать в два раза.

**Определить x2:**

x2:=1

**цикл-пока** f(x2)<y

x2:=x2\*2

**все-цикл**

**Все.**

*Шаг 5.* Детализируем операцию определения x. Определение x выполняется последовательным сокращением отрезка [x1, x2].

**Определить x:**

**цикл-пока** x2-x1>eps

Сократить отрезок [x1,x2].

**все-цикл**

**Все.**

*Шаг 6.* Детализируем операцию сокращения интервала определения x. Сокращение отрезка достигается делением пополам и отбрасыванием половины, не удовлетворяющей условию f(x1) ≤ y ≤ f(x2)

**Сократить интервал определения x.**

xt:=(x1+x2)/2

**если** f(xt)>y

**то** x2:=xt

**иначе** x1:=xt

**все-если**

**Все.**

Таким образом, за шесть шагов мы разработали весь алгоритм. Полностью он выглядит следующим образом.

**Программа:**

Ввести y,n,eps.

x1:=1

**цикл-пока** f(x1)>y

x1:=x1/2

**все-цикл**

x2:=1

**цикл-пока** f(x2)<y

x2:=x2\*2

**все-цикл**

**цикл-пока** x2-x1>eps

xt:=(x1+x2)/2

**если** f(xt)>y

**то** x2:=xt

**иначе** x1:=xt

**все-если**

**все-цикл**

Вывести xt,y.

**Конец.**

При разработке алгоритма методом пошаговой детализации мы использовали псевдокод, но можно было использовать и схемы алгоритма.

Достоинством метода пошаговой детализации является то, что в процессе разработки на каждом шаге мы решали одну достаточно простую задачу. Использование метода, таким образом, существенно облегчает разработку алгоритмов.

1.4. Контрольные вопросы

1. Что такое алгоритм? Как можно описать алгоритм?
2. Что такое псевдокод? Чем он отличается от схемы алгоритма?
3. Какой вычислительный процесс называют разветвляющимся?
4. Каковы правила построения ветвлений?
5. Какой вычислительный процесс называют циклическим?
6. Какие типы циклов вы знаете?
7. Какие основные этапы присутствуют в любом циклическом процессе?
8. Каков порядок проектирования циклического процесса?
9. Какой цикл называют вложенным? Какое правило соблюдается при построении вложенных циклов?
10. Что такое итерационный цикл и его основные особенности?
11. Что такое метод пошаговой детализации и где он применяется?

# 2. Практическая часть

## 2.1. Указания по выполнению домашнего задания

Домашнее задание №1 выдается на 1-ой неделе и включает разработку алгоритмов программ, кодирование и отладку программ домашнего задания 1 (части 1-3). При этом допускается использовать как схемы алгоритма, так и псевдокоды. Все алгоритмы должны быть структурными, т. е. должны использовать только допустимые управляющие структуры.

Проверка домашнего задания осуществляется поэтапно, по мере выполнения практических заданий. Окончательный срок завершения задания – 5 неделя 1-го семестра.

Домашнее задание состоит из трех частей, каждая из которых — это отдельная задача.

**Первая часть задания на вычисление выражения по заданной формуле.** При выполнении задания необходимо проанализировать выражение, выбрать тип данных всех переменных заданного выражения и операции и/или стандартные функции для его вычисления. При программировании выражений в этом задании *не использовать* функцию возведения в степень **pow**, так как у нее есть ряд особенностей, с которыми мы познакомимся позже в процессе обучения. Для возведения в квадрат и любую другую степень до 5 использовать операцию умножения.

Для проверки влияния выбранного типа данных на точность вычислений не составлять новую программу, а поменять при тестировании тип данных и форматы их ввода и вывода. Результат выводить нужно по вещественному формату **%20.16f** или **%20.16lf**. Это позволит более точно определить разницу результатов при использовании разных типов данных.

В отчет включить один вариант программы для наиболее точного типа данных, например, double. Количество тестов для каждого типа данных не менее трех. В таблице результатов для всех рассмотренных типов данных привести не менее трех тестов. Для оценки точности значения переменных по каждому типу данных, необходимо выполнить вычисление для конкретного набора данных на онлайн калькуляторе или Вольфраме. Все значения для разного типа данных при одном наборе данных нужно сравнивать не между собой, а с этим посчитанным. В выводе пояснить, как в вашем случае тип данных влияет на точность результата и почему.

**Пример 1.** Вычислить выражение:

Определить, как влияет на точность вычисления выбранный тип данных.

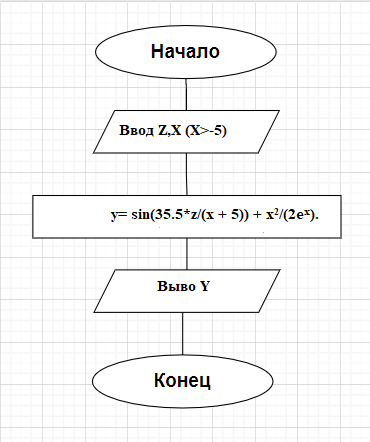
Анализ выражения показал, что в выражении есть операции сложения, умножения, деления, возведение в квадрат, синус и экспонента. Для синуса и экспоненты можно использовать стандартные функции. А вот операции возведения в квадрат стандартной функции нет. Так как нам запрещено использовать стандартную функцию **pow**, используем умножение х на самого себя.

**Схема алгоритма**

Как видно из выражения, при х = -5 в операции деления будет ситуация деления на 0. Кроме того, функция экспоненты становится близкой к 0, что тоже может вызвать в выражение деление на 0. Поэтому, при вводе исходных данных лучше ограничить x. Оно должно быть больше -5. При проектировании алгоритма есть 2 варианта, как это сделать.

1. Необходимо при вводе предусмотреть проверку ввода с помощью конструкции ветвления или цикла с постусловием.
2. До ввода выдавать подсказку на ограничения в исходных данных.

Так как это задание на вычисление выражений, и мы еще не проходили циклов и ветвлений, выбираем вариант два и при вводе будем предупреждать пользователя об ограничении. При этом, конечно, пользователь все равно может ввести некорректные данные. В этом случае, на экране будет выведен результат работы, но это будет ошибка формата, форма вывода которой зависит от среды программирования. Для вывода сообщений об ошибках вещественной арифметики необходимо подключение соответствующих средств компилятора, которые по умолчанию не задаются.



Подсчет выражения для трех тестов при:

**z = 3, x = 5**

**z = 12, x = -2**

**z = 10, x = 0**

**Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Вычисление выражения y= sin(35.5\*z/(x + 5)) + x^2/(2\*e^x).

int main()

{  double y, z, x;

   puts("Input z, x (x > -5)");

   scanf("%lf %lf", &z, &x);

   printf("Inputed z = %6.2lf   x = %6.2lf\n", z, x);

    y = sin(35.5\*z/(x+5)) + x\*x/(2\*exp(x));

   puts("\_\_\_\_\_\_\_ Result value \_\_\_\_\_\_\_\_");

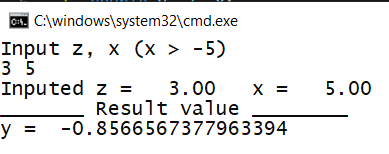
   printf("y = %20.16lf\n", y);  // вывод чисел на экран

   printf("\n");

   return 0;

}

**Пример работы программы**

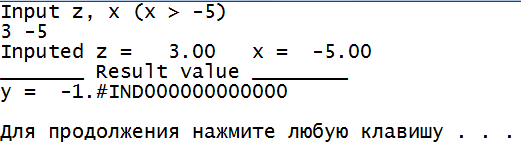
****

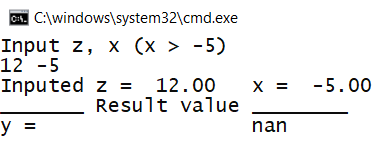
**Тесты**

Для типа **double**

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| x=5, z=3 | -0.856656738 | -0.85665673779663394 |
| x=-2, z=12 | 14.1903172 | 14.190317190693893 |
| x=0, z=10 | 0.95105465 | 0.9510546532543747 |

Аналогичную таблицу составить для других типов исходных данных, но на тех же исходных данных.

Ниже приведены скриншоты работы программы при вводе x=-5, не входящем в разрешенную область значений в разных средах программирования:



**Вторая часть задания на применение операторов ветвления с тестированием всех ветвей алгоритма.** При выполнении задания необходимо проанализировать его и выбрать нужные структурные операторы ветвления. *В целях освоения процесса тестирования ветвящихся алгоритмов не объединять несколько условий проверок в одно.* Каждое условие должно проверяться отдельно. При невыполнении условия в альтернативной ветке **выводить сообщение** о том, почему оно не выполнилось. Например, если x должно быть больше 20, а введенное значение , то необходимо выдать сообщение о том, что условие не выполнилось, так как х ему не удовлетворяет. Это позволит при отладке четко отслеживать, по какой ветке алгоритма прошла программа. В таблицу результатов обязательно включить результаты тестирования **ВСЕХ** ветвей алгоритма с указанием, какая это ветвь. В выводе пояснить результаты и указать, были ли обнаружены ошибки при тестировании.

**Пример 2.**

Даны действительные числа X, Y и W.

Определить .

Протестировать все ветви алгоритма.

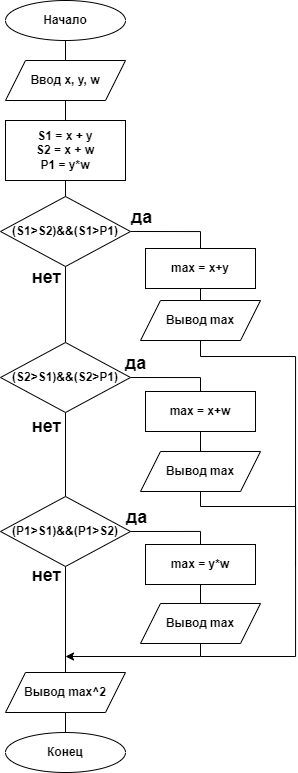
Анализ задания показал, что введенные данные образуют выражения, из которых нужно определить максимальное по значению. Для этого следует использовать несколько полных операторов ветвления, каждый из которых проверяет на максимум одно из выражений. Чтобы при проверках и вычислениях избежать лишних расчетов, целесообразно ввести вспомогательные переменные:

.

Кроме того, в альтернативных ветвях алгоритма следует предусмотреть вывод того выражения, которое не прошло сравнения.

Для тестирования алгоритма посчитаем три варианта проверок. Приведенный алгоритм не единственный и возможны другие приемы нахождения максимального значения из трех величин.

**Схема алгоритма**



**Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Вычисление выражения max^2(x+y, x+w, y\*w).

int main()

{  double y, w, x, s1, s2, p1, max, max2;

   puts("Input x, y, w");

   scanf("%lf %lf %lf", &x, &y, &w);

   printf("Inputed x=%6.2lf   y=%6.2lf   w=%6.2lf\n", x, y, w);

   s1 = x + y;

   s2 = x + w;

   p1 = y \* w;

   if ((s1>s2) && (s1>p1)) {

         max = s1;

         // вывод максимума на экран

         printf("max: x + y = %7.3lf\n", max);

   }

   else {

      //вывод сообщения о невыполнении условия

      puts("s1<=s2 or s1<=p1");

      if ((s2>s1) && (s2>p1)) {

         max = s2;

         // вывод максимума на экран

         printf("max: x + w = %7.3lf\n", max);

      }

      else {

         //вывод сообщения о невыполнении условия

         puts("s2<=s1 or s2<=p1");

         if ((p1>s1) && (p1>s2)) {

            max = p1;

            // вывод максимума на экран

            printf("max: y \* w = %7.3lf\n", max);

         }

         else {

            //вывод сообщения о невыполнении условия

            puts("p1<=s1 or p1<=s2");

            max = 0;

            puts("There is no absolute maximum");

         }

      }

   }

   max2 = max\*max;

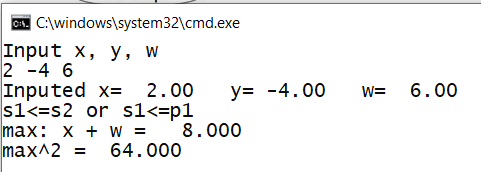
   // вывод квадрата максимума на экран

   printf("max^2 = %7.3lf\n", max2);

   printf("\n");

}

**Пример работы программы**

****

**Тесты**

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| x=-4, y=12, w=3 | Ветвь y\*w (36) max^2=1296 | Ветвь y\*w max^2=1296 |
| x=-2, y=12, w=-5 | Ветвь x+y (10) max^2=100 | Ветвь x+y max^2=100 |
| x=2, y=-4, w=6 | Ветвь x+w (8) max^2=64 | Ветвь x+w max^2=64 |

**Третья часть задания на использование циклов.**

Задание предполагает использование *итерационных циклов* для решения задач численными методами с заданной точностью. Для этого необходимо проанализировать задачу и определить тот численный метод, который будет лежать в основе ее решения. Разработать алгоритм решения с применением структурных приемов программирования. Если есть возможность и навык, просчитать решение вручную. Если нет, в интернете найти решение этой задачи на указанных данных. После разработки программы ее необходимо протестировать как на разных данных, так на одинаковых данных, но для разных значений точности. При этом при выводе на экран результата вычисления, нужно выводить и точное значение, полученное решением задачи вручную или взятое из интернета. Это позволит быстрее обнаружить ошибку, как в алгоритме, так и в его реализации в виде программы. В отчете привести результаты ручного расчета или данные из интернета. В таблице результатов указать не менее трех значений точности, и, если это можно – рассмотреть разные исходные данные. Сделать вывод о влиянии точности на количество разбиений интервала.

**Пример 3.** Решить задачу, организовав итерационный цикл.

Вычислить сумму ряда c точностью ξ:

Проверить программу при значениях . Определить, как изменяется число итераций при изменении точности. Точное значение равно .

**Решение**

Задача заключается в нахождении суммы ряда с определенной точностью, так как на компьютере невозможно найти точное аналитическое решение. Как видно из формулы, для решения понадобится операция возведения числа в степень (при этом без использования функции **pow**) и операция вычисления факториала, реализации которой нет в стандартных библиотеках. Таким образом, реализовывать обе операции придется вручную. Для более простого вычисления степени и факториала будем использовать рекуррентное вычисление, учитывая ту особенность этих величин, что новое значение и степени и факториала можно найти умножением предыдущего значения на текущее.

Вычислять новые члены ряда будем до тех пор, пока значение очередного члена ряда не будет меньше заданной точности ξ, то есть .

Обобщенный неформальный алгоритм решения достаточно прост:

Начало

*Шаг 1.* Ввести ξ

*Шаг 2.* Задать изначальное значение k=1

*Шаг 3.* Задать изначальное значение S=1

*Шаг 4.* Определить новый член ряда r

*Шаг 5.* Прибавить член ряда r к сумме ряда S

*Шаг 6.* Если , перейти к шагу 4

*Шаг 7.* Вывести S

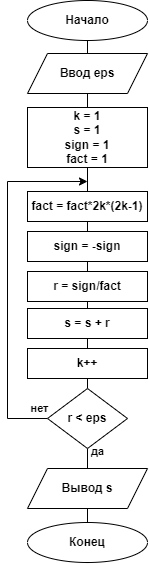
Конец

Также проверим точное значение, указанное в условии задачи, в Вольфраме:



Берем это значение в качестве эталонного.

**Структурная схема**



**Текст программы**

#include <stdio.h>

#include <math.h>

int main(int argc, char\* argv[])

{

    int i, k;

    unsigned long fact;

    double sign, r, eps, s, st = cos(1);

    puts("Input eps:");

    scanf("%lf", &eps);

    // Начальные настройки цикла

    k = 1;

    s = 1;

    sign = 1.; //знак (-1)^k

    fact = 1; //подсчёт факториала

    do

    {

        // расчет (2k)!

        fact = fact\*2\*k\*(2\*k-1); //рекуррентное вычисление факториала

        sign = -sign; //изменение знака на противоположный

        r = sign/fact;

        s = s + r;

        k++;

    } while(fabs(r)>eps);

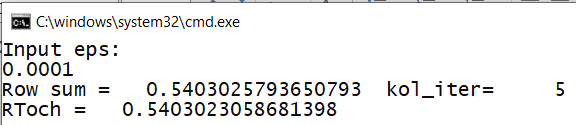
    printf("Row sum = %20.16lf  kol\_iter=%6d\n", s, k);

    printf("RToch = %20.16lf\n", st);// Точное значение суммы ряда

    return 0;

}

**Пример работы программы**

****

**Тесты**

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| eps=0.01 | 0.5403023058681398 | 0.5402777777777777 |
| eps=0.001 | 0.5403023058681398 | 0.5403025793650793 |
| eps=0.0001 | 0.5403023058681398 | 0.5403025793650793 |
| eps=0.000001 | 0.5403023058681398 | 0.5403023037918870 |

**Пример 4.** Решить задачу, организовав итерационный цикл с точностью ξ.

Написать программу вычисления площади, ограниченной функцией и осью x на заданном интервале [0,1] по формуле , где: f(x) - заданная функция, [a, b] – заданный интервал, n - число отрезков разбиения интервала и количество итераций цикла.

Проверить программу при . Определить, как изменяется число отрезков деления при изменении точности. Точное значение площади равно 0,2761423749154.

**Решение**

Анализ показал, что поставленная задача – нахождение площади криволинейной трапеции, ограниченной осью х, прямыми y=0 и y=1, заданной функцией. Такие задачи не имеют точного решения на компьютере, так как эта площадь, по сути, значение определенного интеграла на заданном отрезке. А для вычисления интегралов в программировании используются специальные приближенные численные методы. В соответствие с указанной формулой эту задачу нужно решать с помощью численного метода прямоугольников, который иногда называют методом последовательного приближения (мы постепенно приближаемся к нужной точности вычисления).



Суть метода в делении площади на n прямоугольников шириной d и высотой f(xi).

;

где: .

Увеличивая **n**, получаем приближения площади: **S1, S2, S3 ...**

Останавливаемся, когда .

Обобщенный неформальный алгоритм решения:

Начало

*Шаг 1.* Ввести a, b, d.

*Шаг 2*. Задать число прямоугольников n:=10.

*Шаг 3*. Определить шаг d:=(b-a)/n.

*Шаг 4*. Определить площадь фигуры S1.

*Шаг 5*. Увеличить число прямоугольников вдвое n:=n\*2.

*Шаг 6*. Уменьшить шаг вдвое d:=d/2.

*Шаг 7*. Определить площадь фигуры S2.

*Шаг 8*. Если Разность площадей меньше ξ, то перейти к шагу 11

*Шаг 9*. Запомнить новое значение площади S1:=S2.

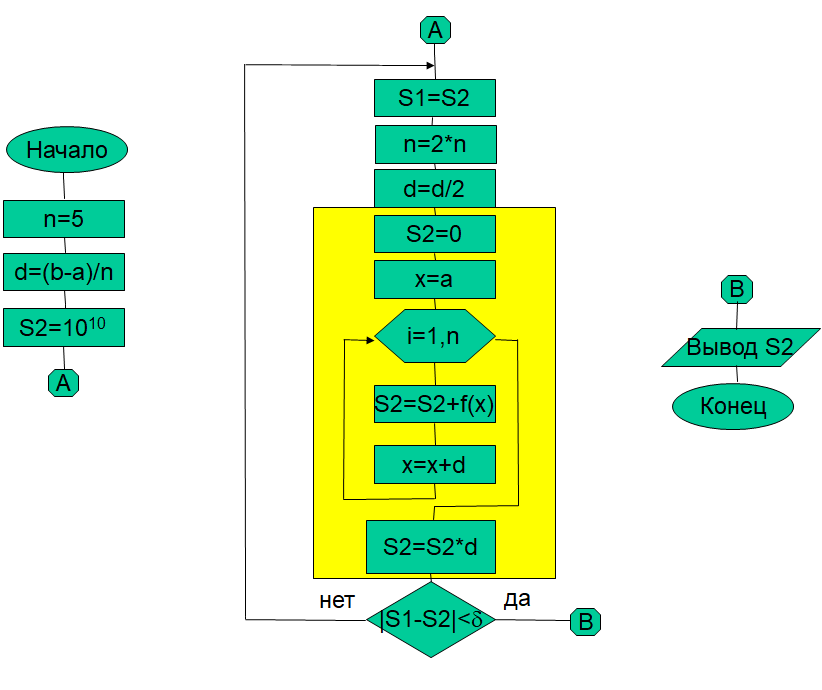
*Шаг 10.* Перейти к шагу 5.

*Шаг 11.* Вывести S2.

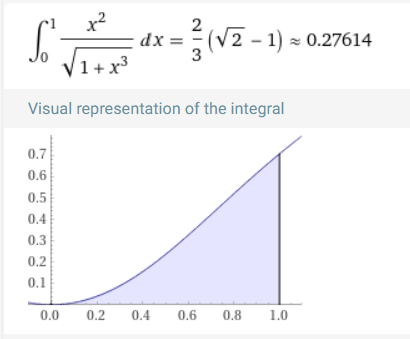
Конец.

Этот алгоритм не совсем структурный. Приведение его к структурному осуществляется с помощью определения первой площади очень большой, что позволит не вычислять предварительно первую начальную площадь.

**Структурная схема**



Перед разработкой программ рассчитаем интеграл с помощью Вольфрама:



Это значение совпадает с точным значением, указанным в программе. Его и возьмем в качестве эталона для сравнения точности результатов.

**Текст программы**

#include <stdio.h>

#include <math.h>

int main(int argc, char\* argv[])

{int i, n, k;

double s1, s2, x, a, b, eps, d, st=0.2761423749154;

puts("input a, b, eps");

scanf("%lf %lf %lf", &a, &b, &eps);

// Начальные настройки внешнего цикла

n=5;

d=(b-a)/n;

s2=1.0e+10;

k=0;

do

{// Начальные настройки внутреннего цикла

s1=s2;

s2=0;n=n\*2;

d=d/2;

x=a;k++;

for(int i=1;i<=n;i++)

{ s2=s2+x\*x/sqrt(1+x\*x\*x);

x=x+d;

}

s2=s2\*d;

} while(fabs(s2-s1)>eps);

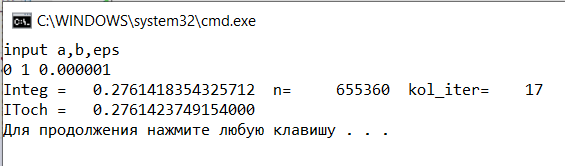
printf("Integ = %20.16lf n= %10d kol\_iter=%6d\n",s2,n,k);

printf("IToch = %20.16lf\n",st);// Точное значение площади

return 0;

}

**Пример работы программы**

****

**Тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат | n |
| a=0, b=1, eps=0.01 | 0.2761423749154 | 0.2674495761621426 | 40 |
| a=0, b=1, eps=0.001 | 0.2761423749154 | 0.2755901275691579 | 640 |
| a=0, b=1, eps=0.0001 | 0.2761423749154 | 0.2760733243286015 | 5120 |
| a=0, b=1, eps=0.000001 | 0.2761423749154 | 0. 0.2761418354325712 | 655360 |

***Рекомендации к выполнению всех частей задания.***

Для выполнения каждой из частей задания необходимо:

1. Прочитать и проанализировать задание в соответствии со своим вариантом.

2. Декомпозировать его на подзадачи, если это необходимо.

3. Разработать алгоритм программы и, если необходимо, подпрограмм.

4. Разработать тесты для поиска ошибок (тестирования) и отладки программы в соответствии с ее особенностями.

5. В среде программирования Visual Studio или Qt Creator создать новый проект и закодировать программу в соответствии с разработанным алгоритмом.

6. Протестировать и отладить программу.

7. Составить отчет по домашнему заданию.

8. Продемонстрировать работу программы преподавателю.

9. Защитить задание преподавателю.

## 2.2 Требования к оформлению

Отчет должен оформляться в соответствии с требованиями кафедры к отчетам. В отчете по домашнему заданию должны быть сделаны выводы о том, какие достоинства и недостатки имеют выбранные методы и разработанные алгоритмы, какие знания и умения студент приобрел в процессе работы над заданием. Кроме того, в начале отчета, после указания цели домашнего задания или его части, или в конце отчета, после выводов, должны быть даны ответы на соответствующие контрольные вопросы.

Отчет можно оформлять как для всего задания сразу, так и отдельно для каждой части.

Отчет по ДЗ должен содержать:

1. Титульный лист (Пример приведен в приложении А). В названии ДЗ указывается либо название части, для которой оформляется отчет, либо общее название задания, если оно имеется. Если общего названия нет, то указываются названия всех частей.
2. Цель домашнего задания или его части.

Для каждой из частей задания приводится

1. Текст задания.
2. Схема алгоритма.
3. Текст программы.
4. Пример работы программы.
5. Таблица тестовых примеров или скриншоты результатов выполнения программы на тестовых данных.
6. Выводы.
7. Ответы на контрольные вопросы, касающиеся всего задания или его части

## 2.3 Требования к защите

Студент считается выполнившим домашнее задание, если у него к сроку окончания работы над заданием завершены и проверены преподавателем отчеты по выполнению указанных заданий.

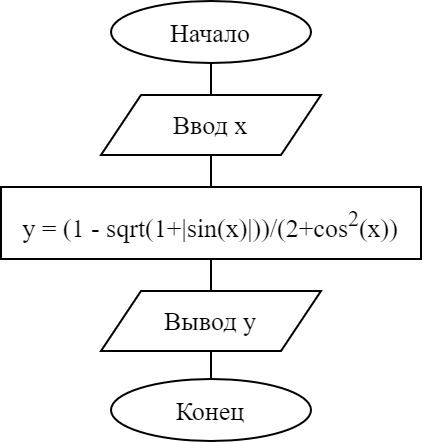
Для получения зачета по домашнему заданию, студент должен представить отчет, оформленный в соответствии с требованиями, продемонстрировать работающую программу и ответить на вопросы преподавателя по программе.

Только при наличии защиты по всем частям домашнего задания проставляется модуль, в котором выполняется это задание.

## 2.4 Пример выполнения задания

**Задание 1*.*** Вычислить: . Определить, как влияет на точность вычисления выбранный тип данных.

Анализ выражения показал, что в выражении есть операции сложения, деления, возведение в квадрат, синус, косинус, взятие модуля и квадратного корня. Для почти всех функций можно использовать стандартные функции, для возведения в квадрат будем умножать косинус на себя. В данном примере областью определения является все множество действительных чисел, следовательно, нет необходимости ограничивать ввод для пользователя. Схема алгоритма и текст программы представлены ниже.



**Текст программы**:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Вычисление выражения y = (1 - sqrt(1+|sin(x)|))/(2+cos^2(x)).

int main() {

 double y, x;

   puts("Input x");

   scanf("%lf", &x);

   printf("Inputed x = %6.2lf\n", x);

   y = (1 - sqrt(1 + abs(sin(x)))) / (2 + cos(x)\*cos(x));

   puts("\_\_\_\_\_\_\_ Result value \_\_\_\_\_\_\_\_");

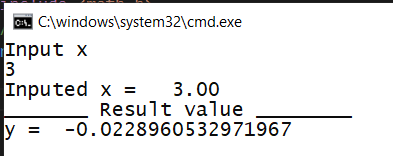
   printf("y = %20.16lf\n", y);  // вывод чисел на экран

   printf("\n");

   return 0;

}

**Пример работы программы**:



Аналитически посчитаем выражения для тестов:

x = 4.2

x = 0.53

x = -15

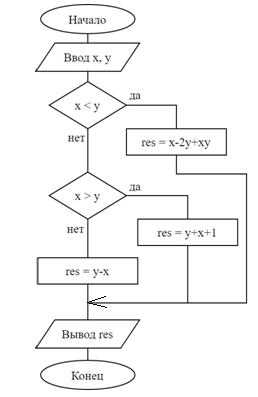
**Тесты**:

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат (float) | Полученный результат (double) |
| x = 4.2 | -0.1642843927 | -0.1642843782901764 | -0.1642843927390613 |
| x = 0.53 | -0.0827134485 | -0.0827134549617767 | -0.0827134484627702 |
| x = -15 | -0.110446802 | -0.1104468032717705 | -0.1104468019758611 |

Из таблицы видно, что значения float и double начинают расходиться после 7-го знака после запятой, что подтверждает тот факт, что тип данных float имеет точность 7 знаков.

**Задание 2**. Даны действительные числа . Вычислить :

Протестировать все ветви алгоритма.

Проанализировав задание, можно понять, что для решения понадобятся несколько полных операторов ветвления, каждый из которых будет проверять положение x относительно y, при этом при невыполнении того или иного условия необходимо выводить соответствующее сообщение. Схема алгоритма и текст программы приведены ниже.

**Текст программы**:

#include <stdio.h>

int main()

{

    double y, x, res;

    puts("Input x, y");

    scanf("%lf %lf", &x, &y);

    printf("Inputed x = %6.2lf y = %6.2lf\n", x, y);

    if (x < y) res = x - 2\*y + x\*y;

    else {

        puts("x >= y");

        if (x > y) res = y + x + 1;

        else {

            puts("x = y");

            res = y - x;

        }

    }

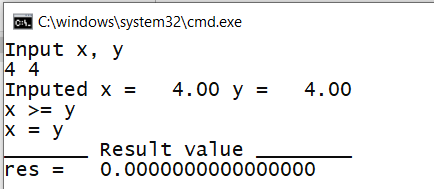
    puts("\_\_\_\_\_\_\_ Result value \_\_\_\_\_\_\_\_");

    printf("res = %20.16lf\n", res);

    return 0;

}

**Пример работы программы**:



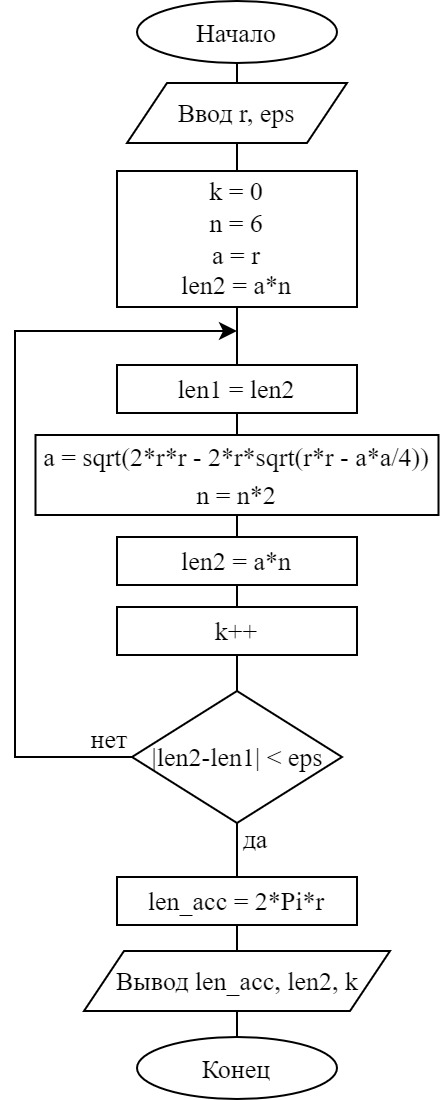
**Тесты**:

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| x = 1, y = 20 | Ветвь x < y res = -19 | Ветвь x < y res = -19 |
| x = 0, y = -12 | Ветвь x > y res = -11 | Ветвь x > y res = -11 |
| x = 6, y = 6 | Ветвь x = y res = 0 | Ветвь x = y res = 0 |

**Задание 3**. Решить задачу, организовав итерационный цикл. Вычислить длину окружности с точностью как предел последовательности периметров вписанных правильных многоугольников с удваивающимся числом сторон (начать с ). Использовать формулу удвоения стороны -угольника:

Проверить программу при . Определить влияние точности на количество итераций цикла.

Для решения задачи на каждой итерации цикла будем находить периметр многоугольника путем умножения количества сторон *n* на длину стороны *a*. При этом *n* каждую итерацию удваивается, а *а* – рассчитывается по формуле из условия. Схема алгоритма и текст программы приведены ниже.



**Текст программы**:

#include <stdio.h>

#include <math.h>

int main()

{

    double eps, a, r, len\_acc, len1, len2;

    const double PI = 3.14159265358979; //число Пи

    int n, k = 0;

    puts("Input radius and eps:");

    scanf("%lf %lf", &r, &eps);

    printf("Inputed radius = %lf eps = %20.16lf\n", r, eps);

    n = 6;

    a = r;

    len2 = a\*n;

    do {

        len1 = len2;

        a = sqrt(2\*r\*r - 2\*r\*sqrt(r\*r - a\*a/4));

        n \*= 2;

        len2 = a\*n;

        k++;

    } while (fabs(len2-len1) > eps);

    len\_acc = 2\*PI\*r; //точное значение длины

    printf("Accurate circle length   = %20.16lf\n", len\_acc);

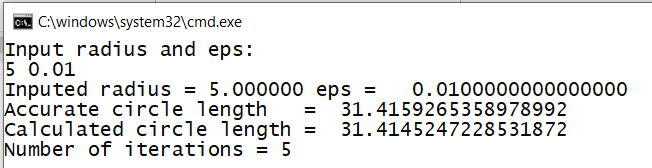
    printf("Calculated circle length = %20.16lf\n", len2);

    printf("Number of iterations = %d\n", k);

    return 0;

}

**Пример работы программы:**

****

**Тесты**:

Посчитаем точное значение площади для некоторых значений радиуса:

R = 5

R = 21

R = 0.2

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат | k |
| R = 5, eps=0.01 | 31.4159265359 | 31.4145247228531872 | 5 |
| R = 5, eps=0.001 | 31.4159265359 | 31.4158389215477243 | 7 |
| R = 5, eps=0.0001 | 31.4159265359 | 31.4159046323509443 | 8 |
| R = 21, eps=0.001 | 131.9468914508 | 131.9467994549236209 | 8 |
| R = 0.2, eps=0.0001 | 1.2566370614 | 1.2566230431645511 | 6 |

## 2.5 Примеры вариантов домашнего задания

**Домашнее задание 1. Вариант 28**

**Часть 1.** **Вычисление выражений.**

Вычислить: . Определить, как влияет на точность вычисления выбранный тип данных.

**Часть 2.** **Программирование разветвляющегося вычислительного процесса.**

Дано действительное число x. Вычислить f(x):

Протестировать все ветви алгоритма.

**Часть 3.** **Программирование циклического процесса. Типы циклов.**

Решить задачу, организовав итерационный цикл. Вычислить длину окружности с точностью ξ как предел последовательности периметров вписанных правильных многоугольников с удваивающимся числом сторон n (начать с n=6). Использовать формулу удвоения стороны n-угольника:

.

Проверить программу при (ξ должна вводиться с клавиатуры). Определить влияние точности ξ на количество итераций цикла.

**Домашнее задание 1. Вариант 26**

**Часть 1.** **Вычисление выражений.**

Вычислить: . Определить, как влияет на точность вычисления выбранный тип данных.

**Часть 2.** **Программирование разветвляющегося вычислительного процесса.**

Даны действительные числа X, Y и W.

Определить .

Протестировать все ветви алгоритма.

**Часть 3.** **Программирование циклического процесса. Типы циклов.**

Решить задачу, организовав итерационный цикл с точностью ξ.

Написать программу вычисления площади, ограниченной функцией

и осью x на заданном интервале [0,1] по формуле:

где: f(x) - заданная функция, [a, b] – заданный интервал, n - число отрезков разбиения интервала.

Проверить программу при . Определить, как изменяется число отрезков деления при изменении точности. Точное значение площади равно .

# Заключение

В результате выполнения домашнего задания 1 студенты получат навыки декомпозиции программы на более простые фрагменты, для которых легко найдут базовые структуры для реализации всего алгоритма решения поставленной задачи. Научатся записывать эти базовые конструкции на языке программирования С++, ставя в соответствие базовым структурам алгоритма соответствующие конструкции языка.

В процессе кодирования и отладки студенты получат знания и умения работать в среде программирования, использовать сообщения встроенного редактора с целью раннего обнаружения синтаксических ошибок, а также исправлять эти ошибки. Кроме того, наличие встроенного отладчика даст возможность студентам научиться составлять тесты для обнаружения ошибок выполнения и локализовать обнаруженные при тестировании программы ошибки, используя встроенные средства отладки.

# Список литературы

Г.С. Иванова, Т.Н. Ничушкина, Р.С. Самарев. Средства процедурного программирования Microsoft Visual С++ 2008. Учебное пособие. – М.: Издательство МГТУ им. Н.Э. Баумана, 2011.

Г.С. Иванова. Программирование. Учебник для ВУЗов. – М.: Кнорус, 2013.- 432 с.

Г.С. Иванова. Технология программирования. Учебник для ВУЗов. – М.: Кнорус, 2013. – 336 с.

Т.Н. Ничушкина, В.В. Гуренко. Разработка алгоритмов простейших программ. Электронное учебное издание. -М.: МГТУ им. Н.Э. Баумана, 2014.

В.В.Подбельский. Язык С++: Учеб. пособие. – М.: Финансы и статистика, 2006.

# ПРИЛОЖЕНИЕ А. Пример титульного листа

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **ФУНДАМЕНТАЛЬНЫЕ НАУКИ**

КАФЕДРА **ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА (фн11)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **02.03.01 МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ**

**Отчет**

|  |  |
| --- | --- |
| **по домашнему заданию №** | 1 |

**Название:**

***Часть 1.1.* Вычисление выражений.**

***Часть 1.2* Программирование ветвлений.**

***Часть 1.3.* Программирование циклических процессов.**

**Дисциплина: Информатика**

**Вариант 28**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ФН11-23Б |  |  | В.П.Малышев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Т.Н.Ничушкина |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023