# КОНТРОЛЬНАЯ РАБОТА № 1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЛИНЕЙНЫХ И ВЕТВЯЩИХСЯ АЛГОРИТМОВ

## Задание 1.1. Программирование формул

Целью работы является освоение программирования **алгоритмов с линейной структурой**, когда решение задачи является результатом выполнения цепи вычислений, в которой очередные вычислительные действия используют в качестве исходных данных результаты вычислений на предыдущих этапах.

**Задание** является заголовком столбцов таблицы вариантов заданий и формулами в соответствии с вариантом задания.

Действия по вычислениям промежуточных и окончательных результатов описываются операторами присваивания. Необходимо следить, чтобы порядок расположения операторов присваивания в программе от ее начала к концу соответствовал логической последовательности действий при решении поставленной задачи.

**Не следует** выражать одни переменные через другие для получения одной формулы из нескольких: в алгоритме и программе необходимо записать столько формул, сколько приводится в задании.

При выборе имен переменных и составлении арифметических выражений необходимо правильно устанавливать тип используемых величин (целые, вещественные и так далее). При использовании в формулах греческого алфавита можно использовать их латинские названия или буквы, сходные по начертанию. Например, символ *α* можно заменить на Alfa или A, *ω* на Omega или W.

К сожалению, в языке Паскаль имеется ограниченное количество математических функций, поэтому, при отсутствии стандартной функции ее выражают через другие, используя функциональные соотношения. Ниже приводятся основные математические функции, отсутствующие в языке Паскаль:

 ;

 ;

 ;

 ;

 ;

 ;

 .

Более того, любую функцию можно вычислить с помощью четырех арифметических операций итерационными методами или разложением в ряды.

Варианты задания, определенные в соответствии с табл. 1, приведены в табл. 2.

Таблица 2. Варианты заданий 1.1

|  |  |  |
| --- | --- | --- |
| № вар. | Вычислить выражение | При заданных значениях |
| 1 |  |  |

## Задание 1.2. Ветвящиеся алгоритмы

Логические выражения используются не только для решения задач булевой алгебры, но и для ветвления программы в логических и циклических операторах. Причем последний вариант использования логических выражений применяется наиболее часто.

Логические выражения состоят из логических констант, переменных и отношений, соединенных логическими операциями. В простейших случаях в операторах используют отношения: два выражения, соединенных знаком отношения или сравнения (<, >, >=, <=, =, <>), например *I* > 20. Но иногда возникают условия, требующие использования более сложных логических выражений.

**Пример задания**. На плоскости задана фигура (например, рис. 9, усеченный круг, в заданиях область фигуры заштрихована). Вводится точка с координатами X,Y. Определить, принадлежит введенная точка фигуре или нет. В результате анализа должно выводиться: «Введенная точка принадлежит фигуре» или «Введенная точка фигуре не принадлежит». Считать, что точка на границе фигуре принадлежит.



Рис. 9. Пример фигуры

Для определения вхождения точки в круг можно использовать формулу окружности

.

Соответственно изменив знак = на ≤ получим условие вхождения точки в круг c координатами центра (3,3) и радиусом 3:

.

Кроме этого область, занятая треугольником, так же не входит в закрашенную область, то есть полуплоскость над прямой *Y*= ‑*X +*7 фигуре не принадлежит. Условия нахождения точки внутри круга и под прямой должны выполняться одновременно. Для этого необходимо использовать логическую операцию *AND*.

Таким образом логическое выражение



примет истинное значение, если точка входит в закрашенную область, иначе ложное. Тогда в логическом операторе по прямой ветви Then выводится «Введенная точка принадлежит фигуре», а по ветви Else – «Введенная точка фигуре не принадлежит».

Но можно и поменять ветви местами, тогда при вхождении точки в фигуру логическое выражение должно принимать ложное значение. Тривиальный вариант: поставить перед предыдущим выражением знак отрицания *NOT*. Но более наглядным решением будет составление выражения с условием невхождения точки в фигуру. Здесь должно выполняться хотя бы одно из условий: точка не входит в круг или точка лежит над прямой, соответственно, логическое выражение примет вид:



При выполнении контрольной работы составить **два варианта программы** (без использования операции *NOT*) для фигуры, соответствующей варианту задания.

**Требование данной работы**: описать всю фигуру одним логическим выражением.

Алгоритм этой задачи представляет типичную ветвящуюся структуру с одним блоком ввода, одним блоком решения (анализа вхождения точки) и двумя блоками вывода. Так как размеры блоков должны быть одинаковыми (ромб блока решения не должен превышать блоков ввода-вывода), то при необходимости можно использовать фигуру комментария, представленную на рис. 10.



Рис. 10. Комментарий в блок-схемах алгоритмов

### Варианты заданий

1. 

# КОНТРОЛЬНАЯ РАБОТА № 2. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

## Задание 2.1. Циклы с известным числом повторений

Целью работы является освоение программирования алгоритмов с циклической структурой, когда какой-либо участок программы выполняется определенное количество раз.

Типичный пример циклического процесса – вычисление конечных сумм. При определении сумм многократно вычисляется выражение, стоящее под знаком суммы и складывается с предыдущей частичной суммой. Вычисления производятся до тех пор, пока не будут сложены выражения под знаком суммы для всех значений изменяющейся переменной.

**Пример задания.** Составить программу, вычисляющую значение суммы

 .

Прежде чем вычислять выражение под знаком суммы и очередную частичную сумму, необходимо определить начальное значение параметра цикла (в данном случае *i*, которое изменяется от 1 до 10 с шагом 1, то есть *i* будет принимать последовательно значения 1, 2, 3, 4, ..., 9, 10), и начальную частичную сумму *S*. Так как вычисления еще не производились, то *S* = 0.

Затем вычисляется выражение под знаком суммы для *i* = 1, затем *i* = 2, 3, ... до 10 и каждый раз складывается с предыдущей частичной суммой *Si*-1. При этом получается новая частичная сумма *Si*. После этого *i* увеличивается на единицу и проверяется, не стало ли *i* > 10. Если еще меньше или равно 10, то вычисляется новая частичная сумма, в противном случае вычисление суммы будет закончено, и это значение выводится на печать.

Воспользуемся стандартной схемой циклического процесса, представленной на рис. 11.

**Блок 2** – блок подготовки к вычислению суммы, в котором задаются начальные значения параметра цикла и частичной суммы.

**Блок 3** – блок проверки окончания цикла. Необходимо проверить, стало ли i больше 10. Если стало, то цикл закончен, следующим должен выполняться блок печати. Если нет, то вычисление частичной суммы продолжается дальше, то есть выполняются блоки 4 и 5.

**В блоке 4** производится вычисление выражения, стоящего под знаком суммы и сложение с предыдущей частичной суммой Si‑1. В итоге получается новая частичная сумма Si.

**В блоке 5** происходит изменение параметра цикла (увеличение i на 1). Это блок подготовки к новому циклу.

Проверка может осуществляться условным оператором IF, но для организации циклов в языке Паскаль специально предусмотрены три оператора цикла. Если количество повторений заранее известно и параметр является целым числом, то целесообразно использовать оператор FOR, включающий в себя блоки 2, 3, 5. В этом случае в алгоритме можно применить блок «Модификация».

Алгоритм для примера с использованием оператора FOR приведен на рис. 12. Варианты заданий – в табл. 3.



Рис. 11. Блок-схема алгоритма циклического процесса



Рис. 12. Блок-схема алгоритма примера.

Таблица 3. Варианты заданий 2.1

|  |  |
| --- | --- |
| № вар. | Вычислить сумму |
| 1 |  |
|  |  |

## Задание 2.2. Двойные и кратные циклы

Целью работы является освоение программирования алгоритмов с вложенными циклами. Пример такой задачи – вычисление двойной суммы.

**Пример задания.**

Вычислить сумму  с точностью до 10-4.

Здесь внешней суммой является сумма по *i*, а внутренней – сумма по *j*. Можно рассматривать вычисление этих сумм отдельно, учитывая что вычисление внутренней суммы является частью вычисления внешней суммы, то есть телом внешнего цикла.

Сами суммы в задании никак не обозначены, и им необходимо присвоить имена, которые могут быть относительно произвольные: SUM\_IN (внутренняя) и SUM\_OUT (внешняя); SUMI и SUMO; S1 и S2 и другие. Так же обозначается и внутреннее слагаемое: SLAG, S и так далее.

Внешняя сумма по *i* вычисляется так же, как и в задании 2.1. Так как используется счетчик циклов, и количество слагаемых определено, то для ее вычисления справедлив алгоритм, представленный на рис. 12, кроме блока 4. Блок 4 – это не обычное сложение, а внутренний цикл по вычислению суммы по *j*. Таким образом, блок 4 замещается несколькими блоками, производящими вычисление внутренней суммы. Говорят, что цикл (суммирование) по *j* вложен в цикл по *i*.

Внутренняя сумма по *j* является примером суммы с бесконечным верхним пределом. Проверка окончания цикла осуществляется следующим образом. Так как выражение под знаком суммы постепенно убывает с ростом слагаемых в сумме, то наступает момент, когда очередное слагаемое станет меньше наперед заданного числа *ε* (грубо говоря, точности вычисления сумм), и остальные слагаемые будут мало влиять на конечный результат. Поэтому, когда выражение под знаком суммы | *f*(*j*) | будет меньше *ε*, то вычисления прекращаются и предполагается, что сумма найдена с заданной точностью. Если возможны отрицательные значения под знаком суммы, то обязательно надо использовать **функцию вычисления модуля ABS**, так как, например, число -10 меньше 10-4 (0.0001), но заданная точность еще не достигнута.

Так как количество слагаемых заранее неизвестно, то циклом FOR пользоваться нельзя. Для этих целей предназначаются циклические операторы WHILE (соответствует блок-схеме, приведенной на рис. 11) и REPEAT. Необходимо помнить, что у них параметр цикла автоматически не изменяется и его надо менять принудительно, а при составлении блок-схемы алгоритма вычисления внутренней суммы (вместо блока 4, рис. 12.) блок «Модификация» не используется.

При вычислении суммы в некоторых вариантах должен вычисляться факториал, который определяется формулой

 .

Где **П** – знак произведения (аналогично знаку суммы), то есть 5! = 1 2 3 4 5 = 120. Факториал можно вычислить отдельным циклом, третьим, а можно и прямо в цикле вычисления внутренней суммы. Для этого вводится дополнительная переменная, например *f* = *j* !, и затем в цикле *f* умножается на текущее значение *j*.

В языке Турбо Паскаль под переменные типа INTEGER выделяется два байта, и допустимые для них значения находятся в диапазоне только от -32768 до 32767. Поэтому число 10!, реально равное 3628800, в этом случае будет представлено как 24320. Таким образом выражение под знаком суммы может никогда и не стать меньше заданной точности. Для работы с большими целыми числами (факториалом) во избежание переполнения рекомендуется использовать вещественный тип Real с диапазоном представления от 2.910-39 до 1.71038, или целый тип LongInt с диапазоном от -2.147.483.648 до 2.147.483.647.

Кроме значения суммы при отладке полезно вывести значение счетчика циклов, то есть узнать, из скольких слагаемых состоит каждая сумма. Например, если все или некоторые внутренние суммы состоят из одного слагаемого, то в программе явно логическая ошибка.

**Требование данной работы**: при составлении программы точность обязательно должна задаваться в форме вещественного числа с плавающей точкой, а вывод суммы выполняться с фиксированной точкой и количеством знаков дробной части, соответствующим заданной точности.

Варианты заданий контрольной работы приведены в табл. 4.

Таблица 4. Варианты заданий 2.2

|  |  |  |
| --- | --- | --- |
| № вар. | Вычислить | Точность вычислений *ε* |
| 1 |  | 10-3 |