Санкт-Петербургское государственное бюджетное профессиональное образовательное учреждение
«Академия управления городской средой, градостроительства и печати»

УТВЕРЖДАЮ
Заместитель директора по УМР
_____ О.В.Фомичева
«20» 12 2023г

Методические указания по выполнению практических работ

«ОП.06 ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»

для специальности

08.02.15 Информационное моделирование в строительстве

Форма обучение –очная

Санкт-Петербург

2023г.

Разработчики: Ипатова С.В., Оболенская Е.Г. методисты СПБ ГБПОУ «АУГСГиП »

Одобрена на заседании цикловой комиссии
Математики и информационных технологий

Протокол №.3.....

«__24__» __11.....__2023.

Председатель цикловой комиссии

_____И.А. Минько

Методические рекомендации к практическим занятиям предназначены в качестве методического пособия при проведении практических занятий по ОП.03 «Прикладные компьютерные программы в профессиональной деятельности» для специальности 08.02.15 «Информационное моделирование в строительстве»

Практические занятия проводятся после изучения соответствующих разделов и тем дисциплины. Выполнение обучающимися практических заданий позволяет им понять, где и когда изучаемые практические умения могут быть использованы в будущей практической деятельности.

Целью практических занятий является приобретение практических умений и навыков.

Методические рекомендации по каждому практическому занятию имеют теоретическую часть с необходимыми для выполнения работы формулами, пояснениями, таблицами; алгоритм выполнения заданий.

В рамках программы дисциплины обучающимися осваиваются умения и знания

Код ПК, ОК	Умения	Знания
ОК 01 -02 ОК 09 ПК 1.1 ПК 1.5 ПК 2.2- 2.3 ПК 3.1- 3.3 ЛР1-4, ЛР10, ЛР13-17	– работать в среде программирования; – использовать языки программирования	– типы данных; – базовые конструкции изучаемых языков программирования; – интегрированные среды программирования на изучаемых языках.

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности

ОК 09 Пользоваться профессиональной документацией на государственном и иностранном языках

ПК 1.1. Адаптировать программные средства в соответствии со стандартами применения технологий информационного моделирования зданий

ПК 1.5. Автоматизировать решение задач формирования, анализа и передачи данных о здании средствами программ информационного моделирования

ПК 2.2 Проектировать строительные конструкции с использованием технологии информационного моделирования

ПК 2.3 Проектировать инженерные сети и оборудование с использованием технологии информационного моделирования

ПК 3.1. Формировать данные структурных элементов информационной модели при решении профильных задач на этапе разработки архитектурной, конструктивной частей, инженерных систем и оборудования проекта

ПК 3.2. Обрабатывать данные структурных элементов информационной модели при решении профильных задач на этапе разработки архитектурной, конструктивной частей, инженерных систем и оборудования проекта

ПК 3.3. Актуализировать данные структурных элементов информационной модели при решении профильных задач на этапе разработки архитектурной, конструктивной частей, инженерных систем и оборудования проекта

Критерии оценки работы к практическим работам

Оценка «5» (отлично) ставится, если:

- правильно выполнил графическое изображение чертежа и графики, сопутствующие ответу;
- показал умение иллюстрировать теоретические положения конкретными примерами, применять их в новой ситуации при выполнении практического задания;
- продемонстрировал усвоение ранее изученных сопутствующих вопросов, сформированность и устойчивость используемых при ответе умений и навыков;
- отвечал самостоятельно без наводящих вопросов преподавателя

Оценка «4» (хорошо) ставится, если:

- в изложении допущены небольшие пробелы, не исказившие логического и информационного содержания ответа;
- нет определенной логической последовательности, неточно используется математическая и специализированная терминология и символика;
- допущены один-два недочета при освещении основного содержания ответа, исправленные по замечанию преподавателя;
- допущены ошибка или более двух недочетов при освещении второстепенных вопросов или в выкладках, легко исправленные по замечанию или вопросу преподавателя

Оценка «3» (удовлетворительно) ставится, если:

- неполно или непоследовательно раскрыто содержание материала, но показано общее понимание вопроса, имелись затруднения или допущены ошибки в определении понятий, использовании терминологии, чертежах, исправленные после нескольких наводящих вопросов преподавателя;
- не справился с применением теории в новой ситуации при выполнении практического задания, но выполнил задания обязательного уровня сложности по данной теме,
- при знании теоретического материала выявлена недостаточная сформированность основных умений и навыков.

Оценка «2» (неудовлетворительно) ставится, если:

- не раскрыто основное содержание учебного материала
- обнаружено незнание или непонимание большей или наиболее важной части учебного материала
- допущены ошибки в определении понятий, при использовании терминологии, которые не исправлены после нескольких наводящих вопросов
- не сформированы компетенции, умения и навыки

Практические занятия

тема	практические занятия	часы
Тема 1.2 Принципы разработки алгоритмов	**Практические занятия** Разработка линейных алгоритмов и алгоритмов ветвления. Разработка циклических алгоритмов.	2
Тема 2.2 Элементы языка. Простые типы данных	**Практические занятия** Знакомство с инструментальной средой программирования	2
Тема 2.3 Базовые конструкции структурного программирования	**Практические занятия** Разработка программ разветвляющейся структуры. Разработка программ с использованием цикла с предусловием. Разработка программ с использованием цикла с постусловием	3
Тема 2.4 Работа с массивами и указателями. Структурные типы данных	**Практические занятия** Разработка программ с использованием одномерных массивов и указателей. Сортировка одномерных массивов. Разработка программ с использованием двумерных массивов. Сортировка двумерных массивов. Разработка программ с использованием структур. Разработка программ с использованием строк.	3
Тема 2.5 Процедуры и функции. Работа с файлами	**Практические занятия** Разработка программ с использованием функций. Разработка программ с использованием рекурсивных функций. Разработка программ работы со структурированными файлами. Разработка программ работы с текстовыми файлами. Разработка программ работы с неструктурированными файлами.	3
Тема 3.1 Класс - как механизм создания объектов	**Практические занятия** Организация классов и принцип инкапсуляции. Разработка приложений с использованием классов.	3
Тема 3.2 Принципы наследования и полиморфизма Понятия деструктора и конструктора	**Практические занятия** Программная реализация принципов наследования. Программная реализация принципов полиморфизма. Разработка конструкторов и деструкторов.	2
Тема 4.2 Разработка приложений	**Практические занятия** Разработка многомодульных приложений.	2
	дифференцированный зачет (практическая работа)	2
		22

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
РАЗРАБОТКА ЛИНЕЙНЫХ АЛГОРИТМОВ И АЛГОРИТМОВ ВЕТВЛЕНИЯ

Цель работы: научиться составлять программы линейной структуры, реализовывать в программе оператор присваивания, процедуры ввода/вывода; строить блок-схемы линейной конструкции.

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

Приступая к решению задач этого раздела, следует вспомнить, что:

- программы с линейной структурой являются простейшими и используются, как правило, для реализации обычных вычислений по формулам;
- в программах с линейной структурой инструкции выполняются последовательно, одна за другой;
- алгоритм программы с линейной структурой может быть представлен следующим образом

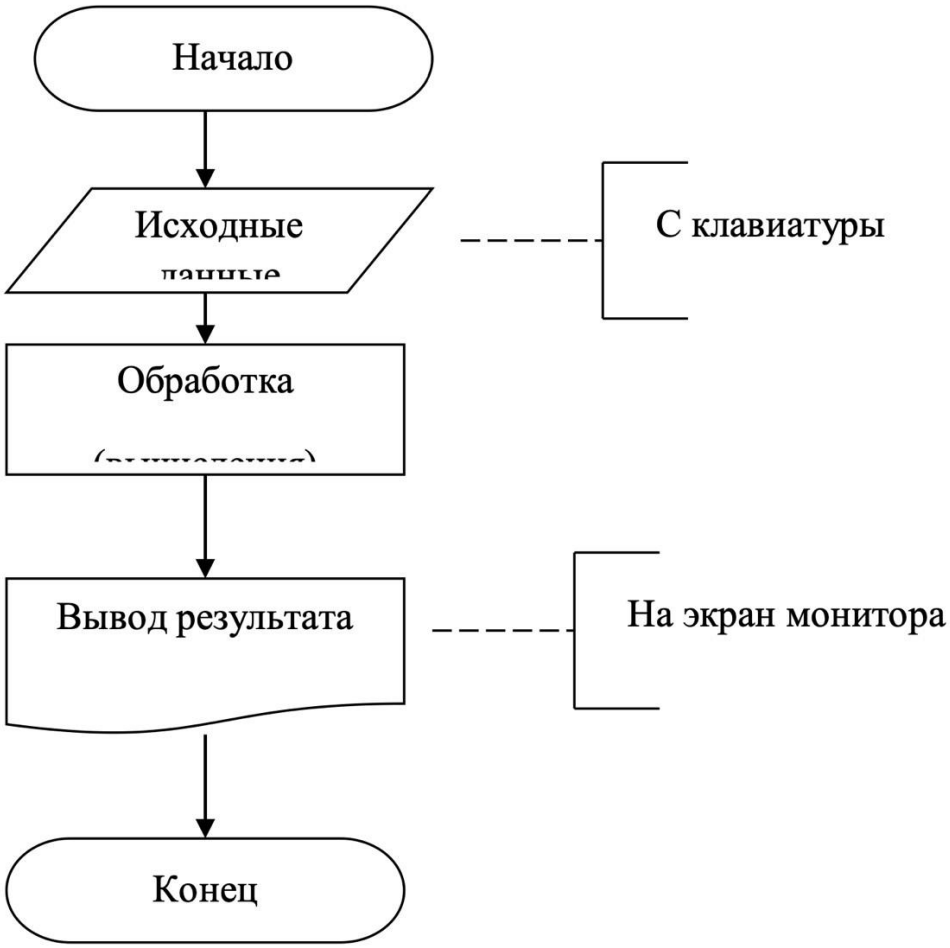
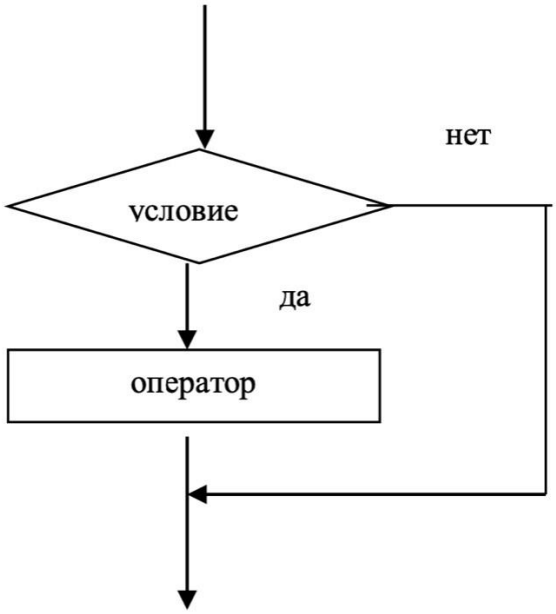


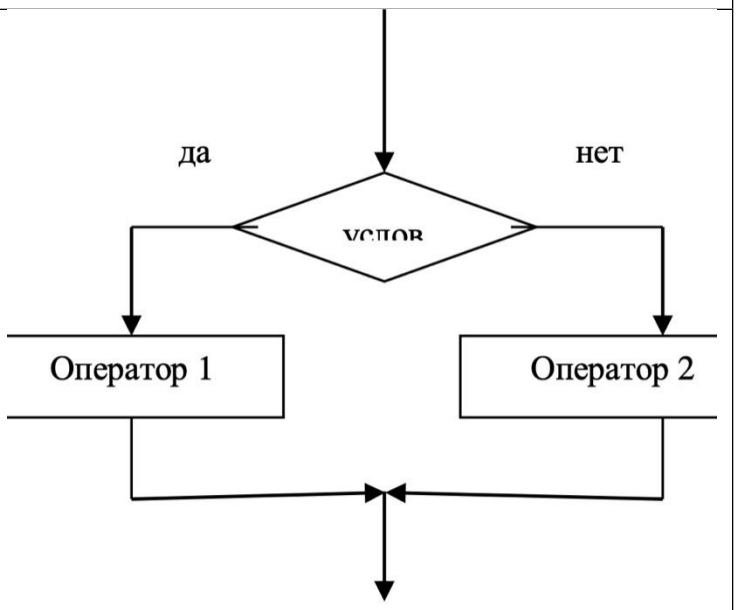
Рис. 1 – Блок-схема линейной конструкции

Ветвление — алгоритм, в котором предусмотрены разветвления, указанные в последовательности действий на два направления в зависимости от итогов проверки заданного условия. То есть такой алгоритм, обязательно содержит условие и в зависимости от результата выполнения условия происходит выбор действия.

Например: Если рабочему дали задание на разработку конструкторской документации, то выполняем это задание, иначе будем заниматься своими делами. Таких примеров можем привести много из обычной жизни и наук. К примеру, физика: **Если** удар упругий, **то** масса тела сохраняется, **иначе** масса изменяется.

Алгоритмическая конструкция ветвление программируется с помощью **условного оператора If**, который может быть представлен двумя вариантами (Таблица 1).

Конструкция	Графическое представление (блок-схема)
1 Вариант — неполное ветвление	
If <условие> **Then** <оператор> Неполное ветвление — в зависимости от результата проверки условия либо выполняются действия одной ветви «да» (оператор), либо эти действия не выполняются.	

2 Вариант — полное ветвление	
If <условие> **Then** <оператор 1> **Else** <оператор 2> Полное ветвление — в зависимости от результата проверки условия выполняются только оператор 1 ветви «да» или только оператор 2 ветви «нет»	

Условие — это логическое выражение, которое может принимать одно из двух значений: true (истина — условие выполняется) и false (ложь — условие не выполняется).

В условии используются операции отношения (=, <> , >,<, >=, <=) и логические операции (and (И), or (ИЛИ), xor (исключающее ИЛИ), not (отрицание)). Если требуется проверить несколько условий, их объединяют с помощью логических операций.

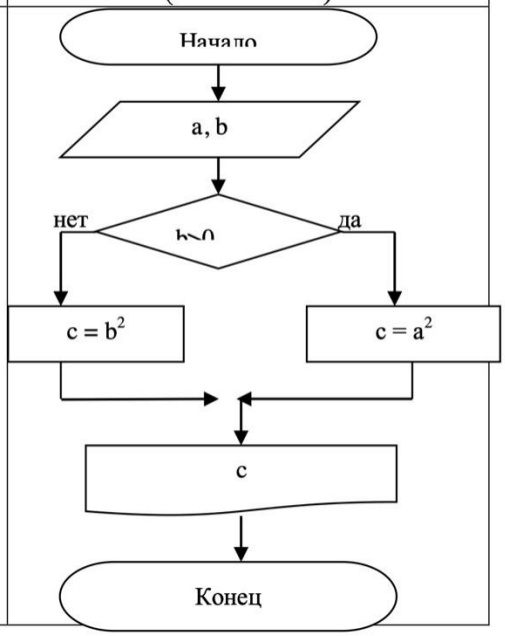
Примеры логических выражений: A<2

(x<>0) and (y<>0)

Если между служебными словами стоят несколько операторов, то они заключаются в операторные скобки Begin...End

Рассмотрим пример:

Даны 2 вещественных числа. Если числа положительные, то возвести в квадрат первое число, иначе возвести в квадрат второе число.

Листинг программы	Графическое представление (блок -схема)
Program Primer_1; Uses crt; {Обозначим 1- ое число через переменную a, 2-ое через переменную b, результат — c} Var a, b, c: real; Begin Clrscr; Writeln('Введите первое число '); Readln(a); Writeln('Введите второе число '); Readln(b); **If** (a>0) and (b>0) **Then** c:=sqr(a) **Else** c:=sqr(b); Writeln('Результат = ', c:5:2); End.	

Задание:

1. Написать программу вычисления объема цилиндра. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление объема цилиндра

Введите исходные данные:

Радиус основания (см) —> 5

Высота цилиндра (см) —> 10

Объем цилиндра 1570.80 куб. см.

Для завершения работы программы нажмите <Enter>.

2. Написать программу вычисления объема цилиндра. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление объема цилиндра

Введите исходные данные: Радиус основания (см) —> 5

Высота цилиндра (см) —> 10

Объем цилиндра 1570.80 куб. см.

Для завершения работы программы нажмите <Enter>.

3. Написать программу вычисления двух выражений:

$$z1 = \frac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4 + b + 2}}$$

$$z_2 = \sqrt{\frac{a + b}{a - 3}}$$

4. Для каждой программы составить блок-схему

5. Выполните задание по варианту, назначенному преподавателем.

Вариант 1

Задание 1

Даны три действительные числа. Возвести в квадрат те из них, значения которых положительны, и в четвертую степень — отрицательные.

Задание 2

Написать программу, которая вычисляет частное от деления двух чисел. Программа должна проверять правильность введенных пользователем данных и, если они неверные (делитель равен нулю), выдавать сообщение об ошибке. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление частного.
Введите в одной строке делимое и делитель,затем нажмите <Enter> -> ***12 0***
Вы ошиблись. Делитель не должен быть равен нулю.

Вариант 2

Задание 1

Даны два действительные числа. Если числа положительны найти их сумму, если отрицательны — произведение

Задание 2

Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 10% предоставляется, если сумма покупки больше 1000 руб. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление стоимости покупки с учетом скидки. Введите сумму покупки и нажмите <Enter>

*-> **1200***

Вам предоставляется скидка 10%

Сумма покупки с учетом скидки: 1080.00 руб.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 3

Задание 1

Даны действительные числа x и y, не равные друг другу. Меньшее из этих чисел заменить половиной их суммы, а большее — их удвоенным произведением

Задание 2

Написать программу проверки знания даты основания Санкт-Петербурга. В случае неверного ответа пользователя программа должна выводить правильный ответ. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

В каком году был основан Санкт-Петербург?

Введите число и нажмите <Enter>

*-> **1705***

Вы ошиблись, Санкт-Петербург был основан в 1703 году.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 4

Задание 1

Данные два вещественных числа. Если первое число больше второго, то возвести его в третью степень, если равно второму — прибавить к нему второе число

Задание 2

Написать программу определения стоимости разговора по телефону с учетом скидки 20%, предоставляемой по воскресеньям. Ниже представлен рекомендуемый вид экрана программы во время ее работы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление стоимости разговора по телефону.

Введите исходные данные:

Длительность разговора (целое количество минут) —> ***3*** *День недели (1 - понедельник, ... 7 — воскресенье) —>* ***6*** *Предоставляется скидка 20%.*

Стоимость разговора: 5.52 руб.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 5

Задание 1

Даны три действительные числа. Если первое число больше второго, умножить данное число на 5, если первое число больше третьего — разделить на два.

Задание 2

Написать программу— модель анализа пожарного датчика в помещении, которая выводит сообщение «Пожароопасная ситуация», если температура в комнате превысила 60 ºС

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 6

Задание 1

Даны действительные числа a, b, c. Удвоить эти числа, если a □ b □ c , иначе оставить без изменения.

Задание 2

Написать программу, которая анализирует данные о возрасте и относит человека к одной из четырех групп: дошкольник, ученик, работник, пенсионер. Возраст вводится с клавиатуры.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 7

Задание 1

Даны три числа a, b, c. Определить какое из них равно d. Если ни одно не равно d, то найти сумму чисел a, b, c.

Задание 2

Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется в том случае, если сумма покупки больше 500 руб., в 5% — если сумма больше 1000 руб. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление стоимости покупки с учетом скидки. Введите сумму покупки и нажмите <Enter>

*-> **640***

Вам предоставляется скидка 3%

Сумма покупки с учетом скидки: 620.80 руб.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 8

Задание 1

Даны три действительные числа. Найти минимальное и максимальное число.

Задание 2

Написать программу проверки знания истории архитектуры. Программа должна вывести вопрос и три варианта ответа. Пользователь должен выбрать правильный ответ и ввести его номер. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Архитектор Исаакиевского собора: 1. Доменико Трезини

2. Огюст Монферран

3. Карл Росси

Введите номер правильного ответа и нажмите <Enter>

-> ***3***

Вы ошиблись.

Архитектор Исаакиевского собора — Огюст Монферран.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 9

Задание 1

Даны три действительные числа. Если все числа положительны, найти среднее арифметическое, иначе произведение.

Задание 2

Написать программу проверки знания истории архитектуры. Программа должна вывести вопрос и три варианта ответа, а пользователь — выбрать правильный ответ и ввести его номер. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Невский проспект получил свое название:

1. По имени реки, на берегах которой расположен Санкт-Петербург

2. По имени близко расположенного монастыря Александро-Невской лавры 3. В память о знаменитом полководце Александре Невском

Введите номер правильного ответа и нажмите <Enter>

*-> **3***

Вы ошиблись.

Правильный ответ: 2.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 10

Задание 1

Даны два вещественных числа, если числа не равны нулю, возвести из в третью степень, иначе во вторую степень.

Задание 2

Написать программу, которая сравнивает два числа, введенных с клавиатуры. Программа должна указать, какое число больше, или, если числа равны, вывести соответствующее сообщение. Ниже представлен рекомендуемый вид экрана во время работы программы.

Введите в одной строке два целых числа

*-> **34 67***

34 меньше 67.

Задание 3

Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Контрольные вопросы

1. Какие типы данные вы знаете?
2. Что такое алгоритм?
3. Приведите пример алгоритма из реальной жизни.
4. Какими свойствами обладает алгоритм?
5. Что такое линейная конструкция?
6. Какие операторы используются для реализации линейной конструкции в программе?
7. Назовите процедуры ввода/вывода данных.
8. Что такое формат вывода данных?
9. Перечислите основные разделы программы.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
РАЗРАБОТКА ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цель работы: научиться использовать операторы циклов при составлении программ на языке Паскаль; составлять блок-схему циклической структуры

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

Операторы цикла используются для многократного повторения аналогичных вычислений.

Для организации цикла в Паскале имеются три различных оператора.

1. Регулярный оператор For

For <параметр цикла>:=<начальное значение> **to** <конечное значение> **do** S;

S- простой или составной оператор.

инструкция for используется для организации циклов с фиксированным, определяемым во время разработки программы, числом повторений;

количество повторений цикла определяется начальным и конечным значениями переменной-счетчика;

переменная-счетчик должна быть целого типа (integer).

При каждом прохождении цикла < параметр цикла >, начиная с <начального значения>, увеличивается на единицу. Цикл выполняется, пока <параметр цикла> не станет больше <конечного значения>.

Другой вариант записи оператора For:

For <параметр цикла >:=< начальное значение> **downto** <конечное значение> **do** S;

В этом случае при каждом прохождении цикла <параметр цикла> уменьшается на единицу от <начального значения> до <конечного значения>.

2. Оператор цикла While с проверкой предусловия: While

<условие> **do** S; {Пока выполняется условие, делать} Цикл

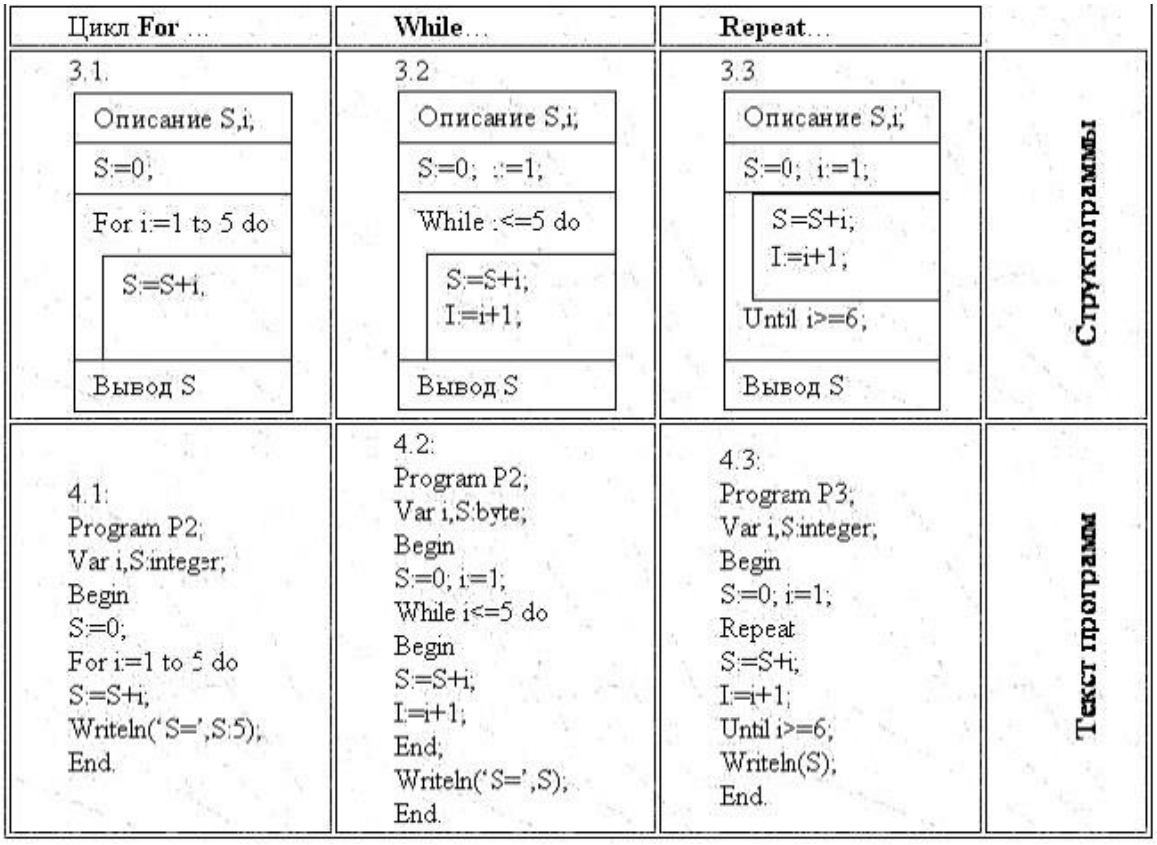
выполняется, пока условие истинно (true).

3. Оператор цикла Repeat с проверкой постусловия:

Repeat S **until** <условие>; {Выполнять до тех пор, пока не будет выполнено условие} Цикл выполняется, пока условие ложно (false).

Пример

Постановка задачи. Найти сумму 5 целых чисел от 1 до 5. Написать программы для определения суммы с помощью трех рассмотренных операторов цикла.



Задание 1. Составьте программы, используя регулярный оператор цикла согласно своему

варианту. Номер варианта соответствует номеру вашего рабочего ПК.

Вариант 1

1. Написать программу, которая выводит таблицу квадратов первых десяти чисел. Ниже представлен рекомендуемый вид экрана во время работы программы.

Таблица квадратов

Число	Квадрат
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 2

1. Написатьпрограмму,котораявыводиттаблицуквадратовцелыхположительныхчисел, введенных пользователем с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы.

Таблица квадратов нечетных чисел

Число	Квадрат
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 3

1. Написать программу, которая вводит с клавиатуры 5 дробных чисел и вычисляет их среднее арифметическое. Рекомендуемый вид экрана во время работы программы приведен ниже:

Вычисление среднего арифметического последовательности дробных чисел. После ввода каждого числа нажимайте <Enter>

-> 5.4

-> 7.8

-> 3.0

-> 1.5

-> 2.3

Среднее арифметическое введенной последовательности: 4.00

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 4

1. Написать программу, которая выводит на экран таблицу стоимости, например, яблок в диапазоне от 100 г до 1 кг с шагом 100. Ниже представлен рекомендуемый вид экрана программы во время ее работы (данные, введенные пользователем, выделены полужирным шрифтом).

Введите цену одного килограмма и нажмите <Enter> (копейки от рублей отделяйте точкой) -> 16.50

Вес (гр)	*Стоимость (руб.)*
100	1.65
200	3.30
300	4.95
400	6.60
500	8.25
600	9.90
700	11.55
800	13.20
900	14.85
1000	16.50

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 5

1. Написать программу, которая выводит на экран таблицу перевода из градусов Цельсия (C) в градусы по Фаренгейту (F) для значений от 15 до 30 с шагом 1 градус. Перевод осуществляется по формуле F =C*1.8+32

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 6

1. Написать программу, которая выводит таблицу значений функции $y=-2,4x^2+5x-3$ в диапазоне от – 2 до 2 с шагом 0,5. Ниже представлен рекомендуемый вид экрана во время работы программы:

X	У
-2	-22,60
-1,5	-15,90
-1	-10,40
-0,5	-6,10
0	-3,00
0,5	-1,10
1	-0,40
1,5	-0,90
2	-2,60

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 7

1. Написать программу, которая вводит с клавиатуры 7 дробных чисел и вычисляет сумму положительных чисел и произведение отрицательных чисел. Рекомендуемый вид экрана во время работы программы приведен ниже:

Вычисление среднего арифметического последовательности дробных чисел. После ввода каждого числа нажимайте <Enter>

-> 1.4

-> 7.8

-> 3.0

-> -7,6

-> -9,2

-> 2.3

Сумма положительных чисел равна =

Произведение отрицательных чисел равно =

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 8

1. Написать программу, которая вычисляет среднее арифметическое последовательности дробных чисел, вводимых с клавиатуры. После того, как будет введено последнее число, программа должна вывести минимальное и максимальное число последовательности.

Количество чисел должно задаваться во время работы программы. Рекомендуемый вид экрана приведен ниже. Данные, введенные пользователем, выделены полужирным шрифтом.

Обработка последовательности дробных чисел. Введите количество чисел последовательности -> 5

Вводите последовательность. После ввода каждого числа нажимайте <Enter> -> ***5.4*** *->* ***7.8*** *->* ***3.0*** *->* ***1.5*** *->* ***2.3***

Количество чисел: 5

Среднее арифметическое: 4.00

Минимальное число:

Максимальное число:

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 9

1. Написать программу, которая выводит таблицу значений функции $y=-9x^2+2x$ в диапазоне от – 3 до 3 с шагом 1. Ниже представлен рекомендуемый вид экрана во время работы программы:

X	У
-3	-87
-2	-40
-1	-11
0	0
1	-7
2	-32
3	-75

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Вариант 10

1. Написать программу, которая вычисляет произведение последовательности целых чисел, вводимых с клавиатуры. После того, как будет введено последнее число, программа должна вывести минимальное и максимальное число последовательности. Количество чисел должно задаваться во время работы программы. Рекомендуемый вид экрана приведен ниже. Данные, введенные пользователем, выделены полужирным шрифтом.

Обработка последовательности дробных чисел. Введите количество чисел последовательности -> 5

*Вводите последовательность. После ввода каждого числа нажимайте <Enter> -> **5** -> **7**- > **3** -> **1** -> **2***

*Количество чисел: **5***

Произведение: 210

Минимальное число:

Максимальное число:

2. Оформить отчет. Отчет должен содержать коды программ и блок-схемы

Задание 2. Составьте программу, используя оператор Repeat по варианту, предложенному преподавателем.

Вариант 1

Написать программу, вычисляющую произведение положительных чисел, которые вводятся с клавиатуры, используя оператор Repeat. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление произведения положительных чисел.

Вводите после стрелки числа. Для завершения ввода введите ноль.

-> ***45*** -> ***23*** -> ***15***

Введено чисел: 3

Произведение чисел =

Вариант 2

Написать программу, которая определяет клавиатуры последовательности положительных ограничена). Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение максимального числа последовательности
положительных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> ***56***
-> ***75***
-> ***43***
-> *О*
Максимальное число: 75.

Вариант 3

Написать программу, которая определяет минимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение максимального числа последовательности положительных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> ***6***
-> ***75***
-> ***3***
-> *О*
Максимальное число:

Вариант 4

Написать программу, вычисляющую произведение положительных чисел, которые вводятся с клавиатуры, используя оператор Repeat. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление произведения положительных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> ***45*** -> ***23*** -> ***15***
Введено чисел: 3
Произведение чисел =

Вариант 5

Используя цикл Repeat, напишите программу, которая требует ввод пароля, например, числа 111, и если пароль правильный, то выдает на экран сообщение «Вы правильно ввели пароль». Пароль можно вводить три раза.

Вариант 6

Используя цикл Repeat, напишите программу определения идеального веса для взрослых людей по формуле: Ид. Вес = рост – 100.Выход из цикла значение роста = 250

Вариант 7

Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление среднего арифметического последовательности положительных чисел. Вводите после стрелки числа. Для завершения ввода введите ноль.

-> **4 -> 230-> 15**

Введено чисел:

Сумма чисел:

Среднее арифметическое:

Вариант 8

Используя цикл Repeat, напишите программу, которая требует ввод пароля, например, числа 834, и если пароль правильный, то выдает на экран сообщение «Вы правильно ввели пароль», если пароль не верный на экране вывести сообщение «Вы ввели не верный пароль у вас осталось 3 попытки», при каждом неверном вводе пароля в сообщении количество попыток уменьшается. Пароль можно вводить три раза.

Вариант 9

Написать программу, которая определяет минимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Ниже представлен рекомендуемый вид экрана во время работы программы

(данные, введенные пользователем, выделены полужирным шрифтом).

Определение максимального числа последовательности положительных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
*-> **61***
*-> **75***
*-> **33***
-> 0
Максимальное число:

Вариант 10

Написать программу, вычисляющую произведение положительных чисел, которые вводятся с клавиатуры, используя оператор Repeat. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление произведения положительных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> 45 -> 23 -> 15
Введено чисел: 3
Произведение чисел =

Задание 3. Составьте программы, с использованием оператора While по варианту, предложенному преподавателем.

Программа, выводит таблицу значений функции для аргумента, изменяющегося в заданных пределах с заданным шагом.

Вариант	Функция	
1.	$y = \frac{\sin(3x)}{x^2}$,	0<x<10
2.	$y = 1 - x + \frac{x^2}{2} + 5x^4$,	-5<x<6
3.	$y = \cos^2 x + \sin 5x$	-3<x<3
4.	$y = -4\sin(x+5) * \cos x$	-6<x<6
5.	$y = \frac{\sin(\frac{\pi}{2} - 5x)}{x^3}$	-5<x<5
6.	$y = \frac{\sin 2x + \sin 6x}{x^4}$	-3<x<6
7.	$y = \cos^3 x + \sin 10x$	-2<x<5
8.	$y = 2\sqrt{\cos x} - tgx$	-5<x<2
9.	$y = \frac{\sin 10x + x^2 - \cos(6x-2)}{x^4}$	-4<x<2
10.	$y = 2\sqrt{\cos 5x} - \sin x$	-6<x<6

Контрольные вопросы

1. В каких случаях в программе необходимо использовать итерационный цикл?
2. Назовите отличия итерационных циклов и цикла с параметром.
3. Какова структура оператора цикла с параметром? Как выполняется цикл с парметром?
4. Какого типа должны быть пареметр цикла, его начальное и конечное значения в цикле с параметром в языке Pascal?
5. Могут ли параметр цикла, его начальное и конечное значения в цикле с параметром в языке Pascal быть разных типов? Обоснуйте ответ.
6. Чем отличается цикл «До» от цикла «Пока»?
7. Сколько раз повторится итерационный цикл?
8. Какова структура цикла с постропроверкой условия?
9. Какова структура цикла с предпроверкой условия?
10. В каких случая в программе необходимо использовать регулярный цикл?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
СОСТАВЛЕНИЕ ПРОГРАММ УСЛОЖНЁННОЙ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ

Цель работы: научиться составлять программы с использованием вложенных циклов

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

Для решения задачи достаточно часто требуется использовать две и более циклические конструкции, одна из которых помещается в тело цикла другой. Такие конструкции называют вложенными циклами. Правила организации внешнего и внутреннего циклов такие же, как и соответствующих простых операторов. Однако при использовании вложенных циклов необходимо соблюдать следующее условие: внутренний цикл должен полностью укладываться в циклическую часть внешнего цикла.

Цикл, в тело которого мы вкладывали команды, называется **внешним циклом**. А цикл, который мы вложили в тело первого, называется **внутренним или вложенным** циклом.

Принцип работы вложенного цикла таков: на первом проходе, внешний цикл вызывает внутренний, который исполняется до своего завершения, после чего управление передается в тело внешнего цикла. На втором проходе внешний цикл опять вызывает внутренний. И так до тех пор, пока не завершится внешний цикл.

Пример 1. Вывести на экран таблицу умножения (от 1 до 9).

Алгоритм решения

Перебрать во внешнем цикле числа от 1 до 9. Для каждого из них перебрать внутреннем цикле числа от 1 до 9. Во внутреннем цикле выполнять переменных-счетчиков внешнего и внутреннего циклов. Таким образом на одну итерацию внешнего цикла произойдет девять итераций внутреннего, и сформируется одна строка таблицы умножения. После каждой строки надо перейти на новую. Это делается во цикле, после того как закончится выполняться внутренний.

Для построения таблицы необходимо использовать форматированный вывод, т.е. задавать ширину столбцов, иначе произойдет сдвиг, т.к. количество цифр в каждой строке различно.

```
var i,j: byte; begin
for i:=1 to 9 do begin for j:=1 to 9 do
write(i*j:4);
writeln; end;
end.
```

Результат работы:

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Задание

Вариант 1

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
a:=1; b:=1;
For i:=0 to n Do
Begin
For j:=1 To b Do
Write ('*') ;
WriteLn;
c:=a+b; a:=b; b:=c;
End;
```

если a=6? Решение какой задачи выражает этот фрагмент программы?

Вариант 2

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
b: = 0 ;
While a<>0 Do
Begin
b:=b*10+a Mod 10; a:=a Div 10;
End;
Write (b) ;
```

если a=13305? Решение какой задачи выражает этот фрагмент программы?

Вариант 3

Выведите на экран числа в следующем виде:

7 6 5 4 3 2

6 5 4 3 2

5 4 3 2

4 3 2

3 2

2

Вариант 4

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
a:=67; b:=31;
For i:=0 to n Do
Begin
For j:=1 To b Do
Write ('*') ;
WriteLn;
c:=a+b; a:=b; b:=c;
End;
```

если a=20? Решение какой задачи выражает этот фрагмент программы?

Вариант 5

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
b: = 0 ;
While a<>0 Do
Begin
b:=b*10+a Div 10; a:=a Mod 10;
End;
Write (b) ;
```

если a=13305? Решение какой задачи выражает этот фрагмент программы?

Вариант 6

Выведите на экран числа в следующем виде:

1 2 3 4 5 6 7 8 9 0

2 3 4 5 6 7 8 9 0

3 4 5 6 7 8 9 0

4 5 6 7 8 9 0

4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 0

Вариант 7

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
a:=1; b:=1;
For i:=0 to n Do
Begin
For j:=1 To b Do
Write ('*') ;
WriteLn;
c:=a*b; a:=c; b:=a;
End;
```

если a=22? Решение какой задачи выражает этот фрагмент программы?

Вариант 8

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
b: = 0 ;
While a<>0 Do
Begin
b:=b*10-a Div 10; a:=a Mod 10;
End;
Write (b) ;
```

если a=56112? Решение какой задачи выражает этот фрагмент программы?

Вариант 9

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
a:=87; b:=25;
For i:=0 to n Do
Begin
For j:=1 To b Do
Write ('*') ;
WriteLn;
c:=a-b; a:=b; b:=c;
End;
```

если a=86? Решение какой задачи выражает этот фрагмент программы?

Вариант 10

Что будет выведено на экране монитора после выполнения следующего фрагмента программы:

```
b: = 2 ;
While a<>0 Do
Begin
b:=b-10+a Mod 10; a:=a Div 10;
End;
Write (b) ;
```

если a=8? Решение какой задачи выражает этот фрагмент программы?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

Цель работы: изучить принципы работы с одномерными массивами на языке программирования Pascal. Получение навыков применения основных алгоритмов для решения задач с использованием массивов.

Необходимые материалы и оборудование: ПК, Pascal ABC

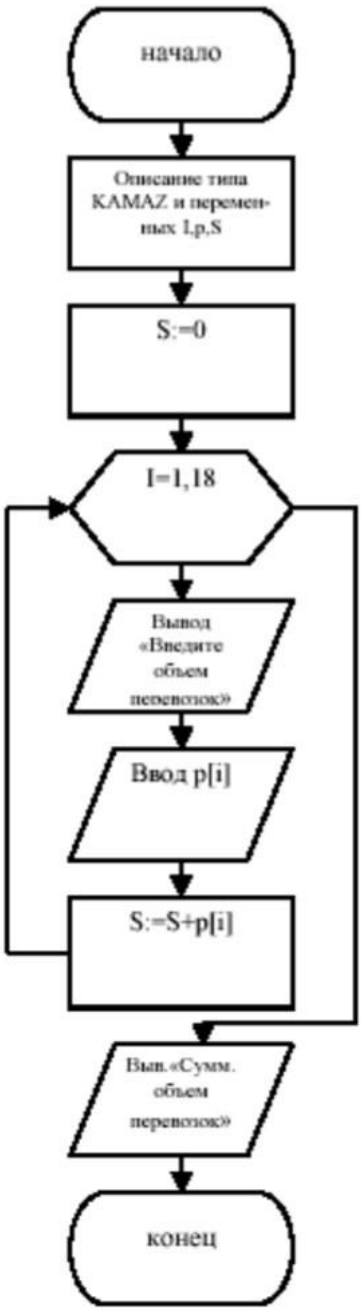
Пояснения к работе:

Нахождение суммы элементов массива

Задача 1. В автопарке, имеющем 18 машин марки КАМАЗ, каждый из КАМАЗов перевез за день определенный объем груза. Определить суммарный объем перевозок грузов за день. При решении задачи будем использовать тип массива KAMAZ для описания всех КАМАЗов автопарка; переменную P[i] для описания объема груза, перевезенного i-той машиной за день (I меняется от 1 до 18). Блок-схема алгоритма для решения данной задачи будет выглядеть следующим образом (см. Рис. 1): Текст следующий вид:

```
Program Pr1;
Uses wincrt;
Type KAMAZ=array [1..18] of real;
Var
i:integer;
p:K AM AZ;
S:real;
Begin
S:=0;
For i:=1 to 18 do
Begin
Writeln('Введите объем перевозок',I,'-ой машины, т');
Readln(p[i]);
S:=S+p[i]; End;
Writeln('Суммарный объем перевозок S=',S:8:2,'т');
End.
```

Накопление суммы в данном примере будет проводиться по шагам, при вводе значения объема перевозок для очередной машины сумма будет увеличиваться на данную величину. Аналогично реализуется и алгоритм нахождения произведения элементов массива (с заменой начального значения суммы S:=0 на начальное значение произведения S:=1 и с заменой операции сложения элементов массива «+» на операцию умножения «*»).



Блок-схема программы

Нахождение количества элементов массива, удовлетворяющих заданному условию

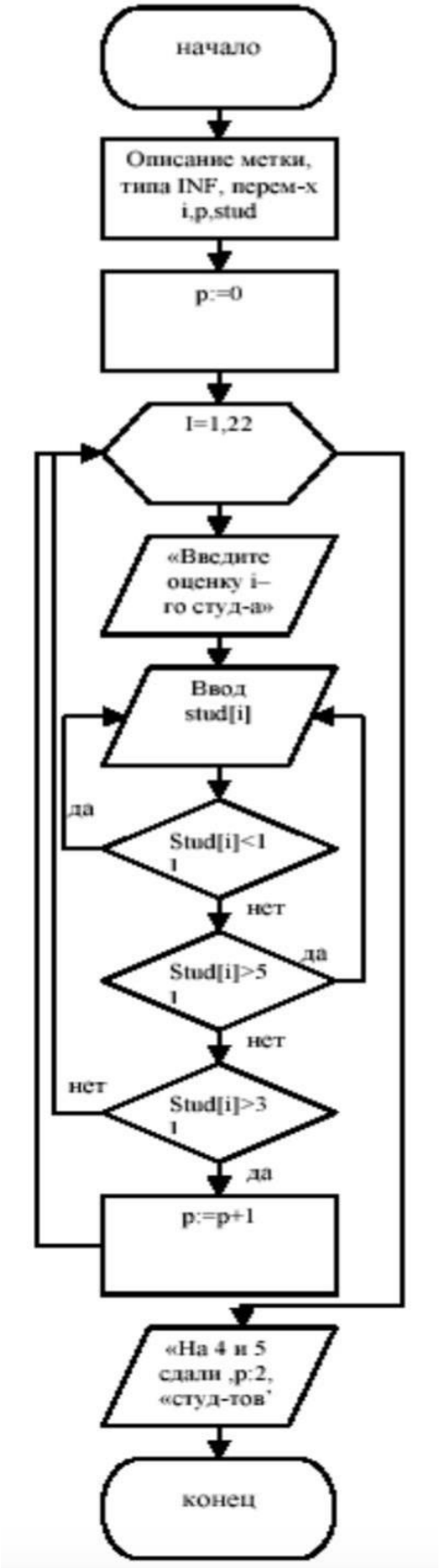
Задача 2. Известны результаты экзамена по информатике одной группы из 22 студентов. Определить, сколько студентов сдали экзамен на 4 и 5. Один из вариантов решения этой задачи следующий:

На Рис. 2 представлена блок-схема алгоритма поставленной задачи.

Текст программы:

```
Program pr3;
Uses wincrt;
Label 1;
Type INF=array[1..22] of integer;
Var
stud:INF;
i,p:integer;
begin
p:=0;
for i:=1 to 22 do
begin
1: writeln('Введите оценку ',i,'-го студента'); readln(stud[i]);
if (stud[i]<1) or (stud[i]>5) then goto 1;
if stud[i]>3 then p:=p+1;
end;
writeln('На 4 и 5 сдали экзамен ',p:2,' студентов'); end.
```

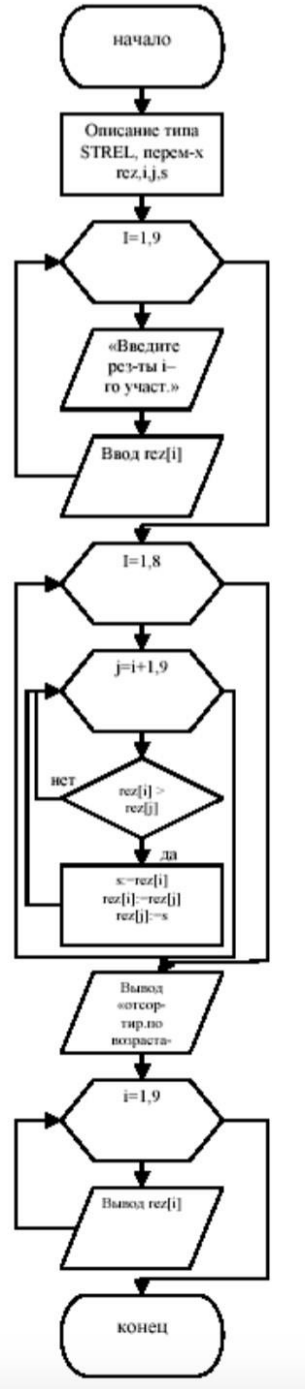
В данной программе для обозначения списка оценок по информатике использовался тип массива INF, для обозначения оценок конкретных студентов – переменная stud. Программа предусматривает проверку
корректности вводимых данных: при попытке ввода несуществующей по пятибалльной системе оценки, программа повторяет ее ввод. Для этого используется оператор перехода GOTO, где имя метки, к которой осуществляется переход (в данном случае 1), описывается в разделе описания меток LABEL.



Блок-схема программы

Сортировка массива по возрастанию

Задача 3. Предположим, известны результаты соревнований по стрельбе, в которых принимали участие 9 человек. Расположить данные результаты в порядке возрастания набранных при стрельбе очков. Алгоритм решения данной задачи является наиболее сложным из приведенных выше примеров и требует использования вложенных циклов.



Блок-схема

Один из способов сортировки массивов заключается в следующем. Сначала первый элемент массива в цикле сравнивается по очереди со всеми оставшимися элементами. Если очередной элемент массива меньше по величине, чем первый, то эти элементы переставляются местами.

Сравнение продолжается далее уже для обновленного первого элемента. В результате окончания данного цикла будет найден и установлен на первое место самый наименьший элемент массива. Далее продолжается аналогичный процесс уже для оставшихся элементов массива, т.е. второй элемент сравнивается со всеми остальными и, при необходимости, переставляется с ними местами.

После определения и установки второго элемента массива, данный процесс продолжается для третьего элемента, четвертого элемента и т.д. Алгоритм завершается, когда сравниваются и упорядочиваются предпоследний и последний из оставшихся элементов массива.

Программа реализации изложенного алгоритма может иметь следующий вид:

```
Program pr4;
Uses crt;
Type STREL=array[1..9]of integer;
Var
rez:strel;
i,j,s:integer;
Begin
For i:=1 to 9 do
begin
writeln('Введите результаты ',i,'-го участника');
readln(rez[i]);
end;
for i:=1 to 8 do
for j:=i+1 to 9 do
if rez[i]>rez[j] then
begin
s:=rez[j];
```

```
rez[j]:=rez[i];
rez[i]:=s;
end;
writeln('Отсортированные по возрастанию результаты:');
for i:=1 to 9 do write (rez[i]:5,' ');
end.
```

Здесь STREL – тип массива результатов стрельбы участников, rez[i] – переменная для описания результатов i-го участника (i меняется от 1 до 9).

ВАРИАНТЫ ЗАДАНИЙ

Исходные данные необходимо оформить в виде массива. При выполнении задания ввод исходных данных и вывод результатов сопровождать комментариями (какие данные нужно ввести и что получается в результате). Составить блок-схему программы. Оформить отчет

1. Подсчитать среднемесячную зарплату сотрудника предприятия.
2. Дан объем продукции, выпущенной заводом за год (помесячно). Найти наименьший объем. В качестве результата вывести номер месяца и объем выпущенной продукции.
3. Курс доллара в течение года менялся в диапазоне от 28 руб. до 30 руб. Найти наибольшее значение курса доллара. В качестве результата вывести номер месяца и значение курса доллара.
4. Известен месячный план выпуска некоторой продукции и объемы выпущенной этой продукции заводом за год (помесячно). Определить, когда завод не выполнил план. Результат получить в виде: номер месяца и объем выпущенной продукции.
5. Даны результаты сдачи экзамена по группе студентов (в группе 20 студентов). Подсчитать количество студентов, не сдавших экзамен.
6. Известна среднемесячная зарплата 10 сотрудников отдела. Расположить данные в порядке убывания.
7. Известен годовой процент на вклад с капитализацией (начисление процентов ежемесячно). Определить, сколько денег получит вкладчик в конце года, если на 1 января сумма вклада составляла 700 000 руб. в качестве результата вывести сумму вклада на конец каждого месяца.

8. Известны данные по продаже компьютеров в течение недели. Найти общее количество проданных компьютеров.
9. Известны данные по продаже компьютеров в течение недели. Расположить эти данные в порядке возрастания.
10. Известен месячный план выпуска некоторой продукции и объемы выпущенной продукции заводом за год (помесячно). Определить месяц, в котором было максимальное отклонение от плана. В качестве результата вывести номер месяца и отклонение.
11. Известен месячный план выпуска некоторой продукции и объемы выпущенной продукции заводом за год (помесячно). Определить, был ли выполнен годовой план.
12. Даны результаты сдачи экзамена по группе студентов (в группе 20 студентов). Подсчитать количество студентов, сдавших экзамен без троек.
13. Известен месячный план выпуска некоторой продукции и объемы выпущенной этой продукции заводом за год (помесячно). Определить, когда завод перевыполнил план. Результат получить в виде: номер месяца и объем продукции, выпущенной сверх плана.
14. Подсчитать среднемесячную зарплату сотрудника предприятия и найти зарплату, которая наиболее близка к средней. В качестве результата вывести среднюю зарплату, наиболее близкую и ее номер в массиве.
15. Даны результаты сдачи экзамена по группе из 15 студентов. Подсчитать количество студентов, не сдавших экзамен, в численном и в процентном соотношении.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
ОБРАБОТКА МНОГОМЕРНЫХ МАССИВОВ

Цель работы: изучить принципы работы с многомерными массивами на языке программирования Pascal. Получение навыков применения основных алгоритмов для решения задач с использованием массивов.

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

Исходные данные для решения многих задач удобно представить в виде таблицы. Например, результат производственной деятельности заводов некоторой фирмы можно представить в виде таблицы (см.Таблица 1)

Колонки и строки таблицы, как правило, содержат однородную информацию, если использовать терминологию Pascal – данные одинакового типа. Поэтому в программе для хранения и обработки табличных данных можно использовать совокупность одномерных массивов. Так, приведенная выше таблица может быть представлена как совокупность одномерных массивов следующим образом:

Zavod1:array[1..4] of integer; Zavod2:array[1..4] of integer; Zavod3:array[1..4] of integer;

Каждый из приведенных массивов может хранить информацию о количестве продукции, выпущенной одним заводом.

Возможно и такое представление:

product1:array[1..3] of integer; product2:array[1..3] of integer; product3:array[1..3] of integer; product4:array[1..3] of integer;

Таблица 1

	Продукт 1	Продукт 2	Продукт 3	Продукт 4
Производство 1	1500	14000	15	125
Производство 2	1380	15600	25	140
Производство 3	2500	13000	8	165

В этом случае массив предназначен для хранения информации о количестве продукции одного наименования, произведенной каждым заводом. Помимо совокупности одномерных массивов, таблица может быть представлена как двумерный массив. В общем виде описание двумерного массива выглядит следующим образом:

Имя:**array**[нижняя_граница_индекса1..верхняя_граница_индекса1,нижняя_граница_индекса 2..верхняя_граница_индекса2] **of** тип
где Имя – имя массива;
array – ключевое слово, показывающее, что объявляемый элемент данных является массивом;

нижняя_граница_индекса1, верхняя_граница_индекса1, нижняя_граница_индекса2, верхняя_граница_индекса2 – целые константы, определяющие диапазоны изменения индексов и, следовательно, число элементов массива;
тип – тип элементов массива.

Приведенная выше таблица (см. Таблица 1) может быть представлена в виде двумерного массива следующим образом:
Product:**array**[1..3,1..4] **of** integer;

Чтобы использовать элемент массива, нужно указать имя массива и индексы элемента. Первый индекс обычно соответствует номеру строки таблицы, второй – номеру колонки. Так, элемент product[2,3] содержит число продуктов третьего наименования, выпущенных вторым заводом.

Двумерные массивы, в которых диапазоны индексов начинаются с 1, также называются иногда матрицами. Размерность матрицы определяется как M*N, где M – число строк в матрице, N – число столбцов. Если число строк матрицы равняется числу столбцов, то матрицы такого вида называются квадратными.

Элементы квадратной матрицы вида B[1,1], B[2,2]< B[3,3] составляют главную диагональ матрицы. Иногда вводят понятие побочной диагонали квадратной матрицы, которую составляют элементы B[1,N], B[2,N-1], B[3,N-2], .. B[N,1], где N – число строк (столбцов) матрицы.

Приведем еще примеры описания двумерных массивов в Pascal-программах: Type
MATR=**array**[1..4,1..5] **of** integer;
Type B= **array**[2..9,0..6] **of** real;
Type C= **array**[-1..4,-1..4] **of** char.

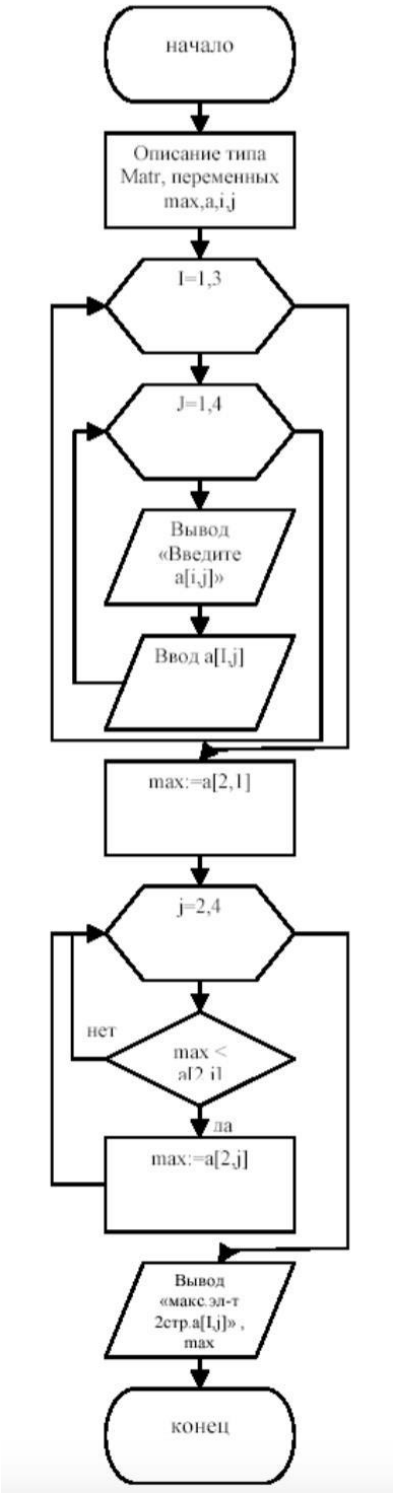
Также допускается указание имени другого типа массива в качестве типа элементов массива, например:
Type VEC= **array**[1..4] **of** real;
MAS= **array**[1..5] **of** vec.

В результате приведенного выше описания тип массива MAS будет объявлен как тип двумерного массива, первый индекс которого будет меняться от 1 до 5, а второй индекс – от 1 до 4, т.е. размерность массива составит 5*4 элементов.

При вводе и выводе значений элементов двумерных массивов используются вложенные циклы, в которых внешний оператор цикла, как правило, задает изменение строк массива, внутренний оператор цикла – изменение столбцов.

ПРИМЕРЫ ЗАДАЧ

1. Нахождение наибольшего элемента в заданной строке.



Блок-схема программы

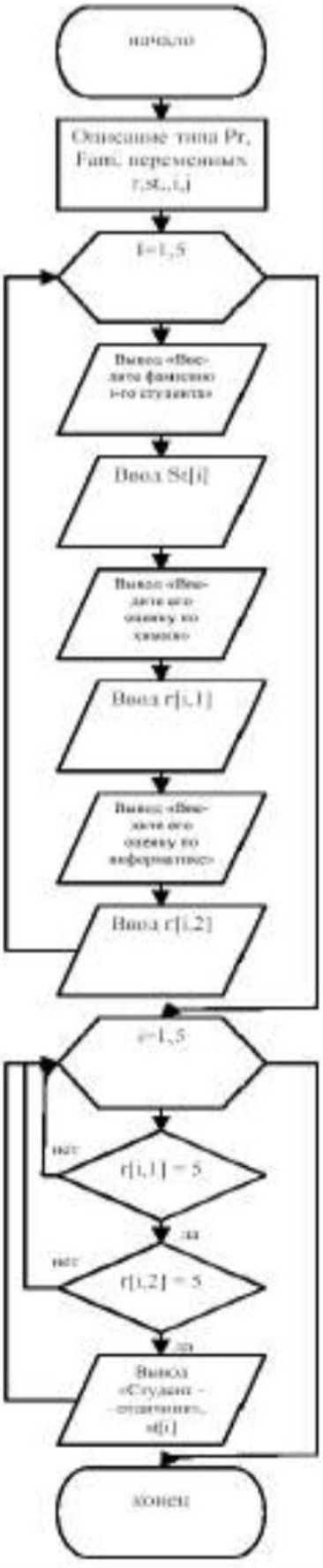
Пусть задана матрица A из действительных чисел размера 3x4. Найти наибольший элемент во второй строке данной матрицы. Блок-схема алгоритма

```
Program Max_str;
Uses crt;
Type Matr=array[1..3,1..4] of real;
Var max:real;
a:Matr;
i,j:integer;
begin
for i:=1 to 3 do
for j:=1 to 4 do
begin
writeln('Введите элемент a[',i,',',j,']'); readln(a[i,j]);
end;
max:=a[2,1];
for j:=2 to 4 do
if max<a[2,j] then max:=a[2,j]; writeln('Наибольший элемент второй строки=',max:8:2);
end.
```

Данная программа представляет собой реализацию алгоритма нахождения наибольшего элемента вектора, полученного путем фиксирования одного из индексов двумерного массива.

2. Нахождение элементов массива, удовлетворяющих заданному условию.

Известны результаты 5 студентов по итогам экзаменов по химии и информатике. Найти фамилии студентов, сдавших оба экзамена на отлично. Для решения поставленной задачи может быть использована следующая программа (ее блок-схема представлена на рис. 2.



Блок-схема программы

```
Program Sessia;
type PR=array [1..5,1..2]of integer; Fam=array[1..5]of string[10];
var r:pr;
st:fam;
i,j:integer;
begin
for i:=1 to 5 do
begin
writeln('Введите фамилию ',i,'-го студента'); readln(st[i]);
writeln('Введите оценку данного студента по химии (от 2 до 5)');
readln(r[i,1]);
writeln('Введите оценку данного студента по информатике (от 2 до 5)');
readln(r[i,2]);
end;
for i:=1 to 5 do
if (r[i,1]=5) and (r[i,2]=5) then writeln('Студент-отличник - ',st[i]);
end.
```

В данной программе для хранения фамилий студентов используется одномерный строковый массив st типа Fam, для хранения оценок студентов – двумерный целочисленный массив r типа PR, причем первый столбец матрицы r используется для хранения результатов экзамена по химии, второй столбец – экзамена по информатике. Если у некоторого студента оценки за оба экзамена составили 5 баллов, то его фамилия будет выведена на экран с сообщением «Студент- отличник».

3. Нахождение сумм элементов строк матриц

Рассмотрим задачу нахождения сумм элементов строк матрицы на примере задачи подсчета итогов футбольного чемпионата. Пусть задана таблица результатов игр 5 команд футбольного чемпионата размером 5x5. На диагонали таблицы стоят значения 0, другие элементы таблицы равны 0, 1 или 2, где 0 баллов соответствует проигрышу команды в игре, 1 балл – ничьей, 2 балла – выигрышу. Определить сумму баллов каждой команды по результатам чемпионата.

Легко заметить, что для построения матрицы R результатов игр достаточно ввести лишь стоящую выше (или ниже) главной диагонали половину матрицы, т.к. результаты остальных игр могут быть рассчитаны из известного соотношения: если, например, первая команда обыграла вторую, то элемент R[1,2]=2, а элемент R[2,1]=2-R[1,2]=0;

аналогично, если вторая команда сыграла в ничью с третьей, то R[2,3]=1, R[3,2]=2-R[2,3]=1. Таким образом, нетрудно получить вид взаимосвязи элементов матрицы: R[i,j]+R[j,i]=2, где i и j меняются от 1 до 5 (кроме элементов главной диагонали). На главной диагонали матрицы R по условию задачи всегда стоят числа 0.

```
Program foot;
Type tab=array[1..5,1..5] of integer;
Var r:tab; i,j,s:integer;
begin
{ввод стоящих выше диагонали элементов матрицы}
for i:=1 to 4 do
for j:=i+1 to 5 do
begin
    writeln (‘Введите результат игры ‘,i,’–й команды с ’,j,’ –й: 0, 1 или 2 балла’);
    readln(r[i,j]); end;
{заполнение стоящих на диагонали элементов нулями}
for i:=1 to 5 do r[i,i]:=0;
{вычисление стоящих ниже диагонали элементов матрицы}
for i:=2 to 5 do
for j:=1 to i-1 do r[i,j]:=2-r[j,i];
{вывод на экран матрицы результатов игр} writeln(‘Таблица чемпионата’);
for i:=1 to 5 do
begin
    for j:=1 to 5 do write(r[i,j]:4);
    writeln; end;
{вычисление сумм элементов строк матрицы}
for i:=1 to 5 do begin
    s:=0;
    for j:=1 to 5 do s:=s+r[i,j];
    writeln(i,‘-ая команда набрала ’,s:3,’ очков’);
end;
end.
```

ЗАДАНИЯ

Исходные данные необходимо оформить в виде двумерного массива, в части заданий использовать дополнительно и одномерные массивы. При выполнении задания ввод исходных данных и вывод результатов сопровождать комментариями (какие данные нужно ввести и что получается в результате).

1. Известен план выпуска компьютеров и количество выпущенных компьютеров тремя фирмами за шесть месяцев. Определить для каждой фирмы, был ли выполнен план по итогам шести месяцев.
2. Известно количество сделанных проектов тремя фирмами за два квартала. Определить максимальное количество выпущенных проектов. В качестве результата вывести месяц, в котором это было, и название фирмы.
3. Известен план выпуска информационной модели и количество выпущенных информационных моделей тремя фирмами за три месяца. Определить, в каком месяце не был выполнен план третьей фирмой.
4. Известен план выпуска компьютеров и количество выпущенных компьютеров тремя фирмами за шесть месяцев. Определить для каждой фирмы количество месяцев, когда план был перевыполнен.
5. Известна заработная плата, полученная 5 сотрудниками отдела в течение года. Определить максимальную заработную плату. В качестве результата вывести фамилию и размер заработной платы.
6. Известно количество выпущенной продукции тремякомпаниями за первый квартал (помесячно). Найти среднемесячное количество выпущенной продукции для каждой компании.
7. Известно количество выпущенной продукции тремя заводами за первый квартал (помесячно). Найти среднемесячное количество выпущенной продукции по всем заводам.
8. Известны результаты сдачи трех экзаменов пятью студентами. Найти фамилии студентов, не сдавших оба экзамена.
9. Известно количество сделанных столов тремя фабриками за два квартала. Определить, какая фабрика выпустила максимальное количество столов по итогам шести месяцев.
10. Известна заработная плата, полученная 10 сотрудниками отдела в течение года. Определить среднемесячную зарплату по отделу.

11. Известны результаты сдачи двух экзаменов семью студентами. Найти фамилии студентов, не сдавших хотя бы один экзамен.

12. Известно количество сделанных столов тремя фабриками за два квартала. Определить, какая фабрика выпустила минимальное количество столов по итогам первых трех месяцев.

13. Известны результаты сдачи трех экзаменов десятью студентами. Найти средний балл каждого студента и общий средний балл. Точность среднего балла – два знака после запятой.

14. Известно количество сделанных столов тремя фабриками за два квартала. Определить, какая фабрика выпустила максимальное количество столов по итогам второго квартала.

15. Известны результаты сдачи двух экзаменов десятью студентами. Определить фамилии студентов, сдавших экзамены без троек.

Контрольные вопросы

1. Что понимают под массивом данных?
2. Что называют размерностью массива?
3. Что понимают под индексом элемента массива?
4. Какой массив называется одномерным?
5. Приведите примеры одномерных массивов.
6. Как описываются одномерные массивы на языке PASCAL?
7. Как задается диапазон изменения индексов массива?
8. Как обозначаются индексы массивов на языке PASCAL?
9. Какие стандартные алгоритмы по работе с одномерными массивами Вы знаете?
10. Поясните понятия двумерного массива, матрицы.
11. Что обозначают индексы матрицы?
12. Сколько элементов в матрице из 7 строк и 9 столбцов?
13. Дайте понятие квадратной матрицы, диагоналей квадратной матрицы.
14. Приведите пример описания двумерных массивов на языке PASCAL.
15. Поясните порядок использования вложенных циклов при вводе элементов двумерного массива.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
СОСТАВЛЕНИЕ ПРОГРАММ СО СТРУКТУРИРОВАННЫМ ТИПОМ ДАННЫХ «МНОЖЕСТВО»

Цель работы: организовывать программы и использованием структурированного типа данных «множество».

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

В Pascal отсутствуют средства ввода-вывода элементов множества, поэтому работу программы необходимо проверять, выполняя ее в пошаговом режиме и отслеживая изменения значений переменных в окне просмотра:

1. С помощью команды **Debug/Add watch** задать переменные, значения которых необходимо наблюдать.
2. Открыть окно наблюдаемых значений, выбрав команду **Debug/Watch**, а окно редактора уменьшить так, чтобы окна не перекрывали друг друга.
3. Установить курсор на точку программы, до которой она выполняется правильно
4. Запустить программу до этой точки, выбрав команду **Run/Go to cursor (F4),** а затем выполнять ее по шагам с помощью команды **Run/Step over (F8),** наблюдая в окне значения переменных.
5. Для прерывания процесса отладки выбрать команду **Run/Program reset (Ctrl+F2)**

Пример

```
Program Dem_Mno;        {Демонстрация операций над множествами}
type
  Digits=set of 0..9;
  var
  d1, d2,d3,d: digits; begin
d1:=[2, 4,6,8];           {Заполнение множеств}
  d2:=[0..3,5];
  d3:=[1,3,5,7,9];
  d:=d1+d2;               {Объединение множеств d1и d2}
  d:=d+d3;                {Объединение множеств d и d3}
  d:=d-d2;                {Разность множеств d и d2}
  d:=d*d1;                {Пересечение множеств d и d1}
  end.
```

ЗАДАНИЯ

1.Опишите множество Pr(1..20) и поместите в него все простые числа в диапазоне 1..20. Составьте блок-схему.

2. Опишите множество Alf(‘a’..’я’) и поместите в него гласные буквы. Составьте блок-схему. 3.Опишите множества M1(1,2) и M2(2,1). Сравните множества M1 и M2 на равенство. Составьте блок-схему.

4.Опишите множества M1(‘a’, ‘b’) M2(‘b’, ’a’, ‘c’). Сравните два этих множества на неравенство. Составьте блок-схему.

5.Опишите множества M1(‘a’, ‘b’,’c’) M2(‘a’, ‘c’). Сравните два этих множества с использованием операции >=. Составьте блок-схему.

6.Опишите множества M1(1, 2, 3) M2(1, 2, 3, 4). Сравните два этих множества с использованием операции <=. Составьте блок-схему.

7. Опишите множества M1(1, 2) M2(5, 6). Получите результирующее множество M3=M1-M2. Определите содержится в M3 элемент 7. Составьте блок-схему.

8. Опишите множества M1(1, 2, 3,4) M2(3, 4, 1). Получите результирующее множество M3=M1- M2. Определите содержится в M3 элемент 7. Составьте блок-схему.

9.Опишите множества M1(1, 2, 3) M2(1, 4, 2, 5). Получите результирующее множество M3=M1*M2. Определите содержится в M3 элемент 2. Составьте блок-схему.

10. Опишите множества M1(1, 2) M2(5, 6). Получите результирующее множество M3=M1+M2. Определите содержится в M3 элемент 2. Составьте блок-схему.

Контрольные вопросы

1. Что такое множество?
2. Как в Паскале организовывается работа с множеством?
3. Перечислите операции сравнения множества
4. Как описать множество в программе?
5. С помощью какой функции определить содержится ли элемент в множестве?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
ОРГАНИЗАЦИЯ ПРОГРАММ С ИСПОЛЬЗОВАНИЕ ПРОЦЕДУР. ОРГАНИЗАЦИЯ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ

Цель работы:

- Научиться описывать процедуры и функции в программе;

- Научиться задавать фактические и формальные параметры;

- Научиться организовывать вызов процедуры и функции в программе

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

Подпрограмма — это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем.

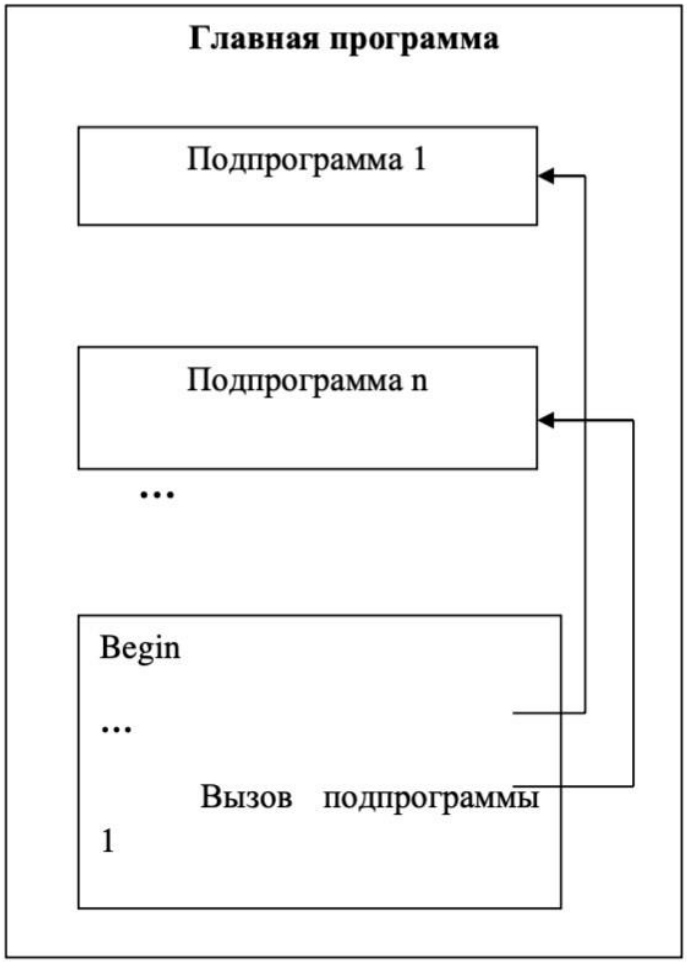


Рис. 7.1 Структура программы с подпрограммами

Различают два вида подпрограмм — это процедуры и функции.

Процедура и функция — это именованная последовательность описаний и операторов. При использовании процедур или функций Pascal— программа должна содержать текст процедуры или функции и обращение к процедуре или функции. Тексты процедур и функций помещаются в раздел описаний процедур и функций.

Процедура — это независимая именованная часть программы, которую можно вызвать по имени для выполнения определѐнной в ней последовательности действий. Функция отличается от процедуры тем, что возвращает результат указанного при еѐ описании типа. Вызов функции может осуществляться из выражения, где имя функции используется в качестве операнда.

Заголовок процедуры имеет вид: PROCEDURE <имя> [(<сп. ф. п .>)] ; Заголовок функции: FUNCTION <имя> [(<сп.ф.п.>)] : <тип>;

Здесь <имя> - имя подпрограммы (правильный идентификатор); <сп.ф.п.> - список формальных параметров;

<тип> - тип возвращаемого функцией результата.

Описание, определение и вызов процедур

Описание процедуры производится в разделе описаний основной программы. Любая процедура оформляется аналогично программе, может содержать заголовок, разделы описаний и операторов. Синтаксис заголовка процедуры:

Procedure <Name>(<Список формальных параметров>);

{Раздел описаний}

Begin ...{Раздел операторов процедуры}End;

где

Procedure - служебное слово;

Name - произвольный идентификатор, определяющий имя процедуры.

Procedure MyProc (A,B,C: Real; var X1,X2: Real); Begin

WriteLn('A=',A, ' B=', B, 'C=', C); X1:=A+B;

X2:=A*B-C

End;

Разделы описаний процедуры подобно основной программе могут содержать разделы описания меток (Label), констант (Const), типов (Type), переменных (Var) и раздел процедур и функций. Раздел операторов помещается после служебного слова Begin и заканчивается служебным словом End, после End ставится " ; ".

В основной программе процедуры располагают перед разделом операторов (телом программы) основной программы.

Формальные параметры — это переменные, посредством которых передаются данные из места вызова процедуры в еѐ тело, либо из процедуры в места вызова. Список формальных параметров может отсутствовать, при этом символ " ; " ставится сразу за именем процедуры и данные из места вызова процедуры в еѐ тело не передаются.

Для вызова процедуры на исполнение к ней необходимо обратится.

Вызов процедуры производится указанием имени процедуры и списком фактических параметров:

Name(<Список фактических параметров>);

MyProc(K, L+M, 12, Y1, Y2);

Выполнение оператора вызова процедуры состоит в том, что все формальные параметры заменяются соответствующими фактическими. После выполнения процедуры происходит передача управления в основную программу, т.е. начинает выполняться оператор, следующий за оператором вызова процедуры.

Фактические параметры — это переменные (или значения заданные явно), которые передаются в процедуры на место формальных параметров. Если в вызываемой процедуре отсутствует список формальных параметров, то список фактических параметров тоже отсутствует.

Количество фактических параметров должно соответствовать количеству формальных параметров; соответствующие фактические и формальные параметры должны совпадать по порядку записи и по типу данных.

Описание, определение и вызов функции

Оформляется функция аналогично процедуре. Отличительной особенностью функции является то, что она возвращает только один результат выполнения. Этот результат обозначается именем функции и возвращается (передается) в основную программу (место вызова). Функция состоит из заголовка, раздела описаний и раздела операторов.

Function <Name>(<Список формальных параметров>):<Type>; ... {Раздел описаний}

Begin

...{Раздел операторов процедуры}

Name:=<выражение соответствующего типа;

...

End;

где Function - служебное слово; Name - произвольный идентификатор,

определяющий имя функции. В отличии от процедур в разделе операторов тела функции обязательно должен быть хотя бы один оператор присвоения имени функции выражения или значения соответствующего типа. После работы функции результат присваивается имени функции.

Таким образом, алгоритм можно оформить в виде функции в том случае, если в качестве результата получается одно единственное значение. Для вызова функции достаточно указать ее имя (с фактическими параметрами) в любом выражении, где тип результата функции будет приемлем. Имя функции можно использовать в арифметических выражениях и других командах.

Пример1. Разработать функцию, определяющую по двум катетам гипотенузу прямоугольного треугольника.

```
Function Gepoten(a,b:real):real; Begin
Gepoten:=Sqrt(Sqr(a)+Sqr(b))
End;
```

Вызов функции из основной программы может выглядеть следующим

```
z:=Gepoten(x, y); {z присваивается значение гипотенузы} или WriteLn('Значение гипотенузы', Gepoten(x, y));
```

Передача параметров в подпрограммы

Передача параметров в подпрограмму может осуществляться несколькими способами. Параметры процедур и функций могут быть следующих видов: параметры-значения, **параметры-переменные**, параметры-константы и не типизированные параметры.

Группа параметров без предшествующего ключевого слова является списком параметров-значений

Группа параметров, перед которыми следует ключевое слово Const и за которыми следует тип, является списком **параметров-констант**.

Группа параметров, перед которыми стоит ключевое слово Var и за которыми следует тип, является списком типизированных параметров-переменных.

Группа параметров, перед которыми стоит ключевое слово Var или Const за которыми не следует тип, является списком не типизированных параметров-переменных.

Передача параметров по значению

При передаче параметров по значению в формальный параметр передаѐтся копия значения соответствующего фактического параметра. Примерами параметров-значений служат параметры A, B и C в процедуре с заголовком MyProc. В этом случае фактическим параметром, соответствующим A, либо B, либо C, может быть любое выражение или переменная типа Real. Для параметров-значений компилятор при вызове процедуры

выделяет место в сегменте стека (специальная область оперативной памяти) для каждого формального параметра, вычисляет значение фактического параметра и передаѐт его в ячейку, соответствующую формальному параметру, выполняет тело процедуры. После завершения работы процедуры, формальные параметры уничтожаются, а фактические остаются неизменными (по значению).

Формальный параметр-значение обрабатывается, как локальная по отношению к процедуре или функции переменная, за исключением того, что он получает свое начальное значение из соответствующего фактического параметра при активизации процедуры или функции.

Соответствующее фактическое значение параметра-значения должно быть выражением и его значение не должно иметь файловый тип или какой-либо структурный тип, содержащий в себе файловый тип. Фактический параметр должен иметь тип, совместимый по присваиванию с типом формального параметра-значения. Если параметр имеет строковый тип, то формальный параметр будет иметь атрибут размера, равный 255.

Приступая к решению задач этого раздела, следует вспомнить, что:

- для передачи информации в процедуру следует использовать параметры, а не глобальные переменные, т. е. объявленные вне процедуры;
- тип каждого фактического параметра (константы или переменной) в инструкции вызова процедуры должен соответствовать типу соответствующего формального параметра, указанного при объявлении функции;
- если в инструкции объявления процедуры перед именем формального параметра нет слова var, то в качестве формального параметра в инструкции вызова процедуры можно использовать константу или переменную соответствующего типа. Если слово var присутствует в инструкции, то формальным параметром можно назначить только переменную;
- если аргумент процедуры применяется для возврата результата в программу, вызвавшую эту процедуру, то перед именем аргумента нужно поставить слово var.

Пример 1

Программа вычисления степени по вводимым пользователем значением числа и показателя степени.

```
program Func;
uses crt;
var
a,z,r: integer;
m:integer;
{Функция вычисления степени, N, X – формальные параметры}
function Stepen(n:integer; x:integer):integer;
var i:integer;
y: integer;

begin
y:=1;
for I:=1 to n do y:=y*x; stepen:=y
end;

{Цикл вычисления n-ой степени числа x}
{Присваивание функции результата вычисления степени}

{Конец функции}

Begin
clrscr;
writeln('vvedite znachenie chisla a i pokazatel stepeni m'); readln(a);
readln(m);
z:=Stepen(m,a);
Writeln('z=', z:3);
end.
```

Пример № 2

Даны два натуральных числа a и b. Требуется определить наибольший общий делитель трех величин. Определить наибольший общий делитель трех величин a+b, |a-b|, a*b
Идея решения состоит в следующем математическом факте: НОД(x, y, z) = НОД(НОД(x,y),z)

```
Program NOD; Uses crt;
var
a,b,c: integer;
Procedure Evklid (M,N: integer; Var k: integer); {m,n – формальные параметры
(параметры- аргументы, к – параметр-результат)}
Begin
while m<>n Do
If m>n
then m:=m-n else n:=n-m; k:=m
End;
Begin
clrscr;
write('a=');
readln(a);
Write('b=');
readln(b);
Evklid(a+b, ABC(a-b), a*b); Evklid(c, a*b, c); Writeln('NOD=',c)
End.
```

ЗАДАНИЕ

Напишите программу на языке программирования, используя процедуры и функции, по варианту, предложенному преподавателем.

Вариант 1

Дана целочисленная квадратная матрица 4х4. Определить:

1. произведение элементов во второй строке (оформить в виде функции)
2. сумму элементов главной диагонали (оформить в виде процедуры)
3. Составить блок-схемы

Вариант 2

Дана целочисленная матрица размером 4х3. Определить:

1. Количество нулевых элементов в 4 строке. (Оформить в виде функции).
2. Максимальный элемент матрицы (оформить в виде процедуры)
3. Составить блок-схемы

Вариант 3

Дана целочисленная прямоугольная матрица. Определить:

1. Сумму положительных элементов матрицы (оформить в виде функции)
2. Минимальный элемент матрицы (оформить в виде процедуры)
3. Составить блок-схемы

Вариант 4

Дана целочисленная квадратная матрица 3х3. Определить:

1. Количество отрицательных элементов матрицы (оформить в виде функции)
2. Количество положительных элементов в 3 строке (оформить в виде процедуры)
3. Составить блок-схемы

Вариант 5

1. Сформировать одномерный массив B с помощью генератора случайных чисел в интервале от 0 до 50 (оформить в виде процедуры).

2. Подсчитать для сформированного массива B сумму положительных элементов (оформить в виде функции).

3. Составить блок-схемы

Вариант 6

1. Сформировать многомерный массив размерностью 3х3, ввод элементов осуществлять с клавиатуры (оформить в виде процедуры).

2. Подсчитать для сформированного массива произведение элементов главной диагонали (оформить в виде функции).

3. Составить блок-схемы

Вариант 7

1. Сформировать многомерный массив T размерностью 4х2 с помощью генератора случайных чисел в интервале от 0 до 100 (оформить в виде процедуры)

2. Посчитать для сформированного массива T сумму элементов первой строки (оформить в виде функции)

3. Составить блок-схемы

Вариант 8

1. Сформировать одномерный массив D, состоящий из 10 элементов с помощью генератора случайных чисел в интервале от 0 до 200 (оформить в виде процедуры)

2. Подсчитать для сформированного массива D среднее арифметическое (оформить в виде функции)

3. Составить блок-схемы

Вариант 9

Дана целочисленная квадратная матрица 4х4. Определить:

1) произведение положительных элементов матрицы (оформить в виде функции)
2) минимальный элемент во второй строке (оформить в виде процедуры)
3) составить блок-схемы

Вариант 10

Дана целочисленная прямоугольная матрица. Определить:

1) сумму отрицательных элементов (оформить в виде функции)
2) максимальный элемент в первой строке
3) составить блок-схемы

Контрольные вопросы

1. Что такое подпрограмма?
2. Что такое процедура?
3. Что такое функция?
4. Что такое фактический параметр?
5. Что такое формальный параметр?
6. Как фактические параметры должны соответствовать формальным параметрам?
7. Чем описание процедуры отличается от описания функции?
8. Как осуществляется вызов процедуры?
9. Как осуществляется вызов функции?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
ОРГАНИЗАЦИЯ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ТЕКСТОВЫХ ФАЙЛОВ

Цель работы: научиться объявлять текстовые файлы в программе, научиться создавать и обрабатывать текстовые файлы с помощью языка программирования Паскаль.

Необходимые материалы и оборудование: ПК, Pascal ABC

Пояснения к работе:

В Паскале понятие **файла** употребляется в двух смыслах:

1. как поименованная информация на внешнем носителе (внешний файл);
2. как переменная файлового типа (внутренний файл)

Файловый тип переменной— это структурированный тип, представляющий собой совокупность однотипных элементов

Файл можно представить как последовательную цепочку элементов, пронумерованных от 0, заканчивающуюся специальным кодом, который называется **маркер конца**

Эл. 0	Эл. 1	…	Эл. N	Маркер конца

Var <имя переменной>: **File of** <тип элемента> Например:

Var

F1: **file of** Integer;

F2: **file of** Real;

F3: **file of** Char;

Файловая переменная связывается с именем файла в результате обращения к стандартной процедуре ASSIGN:

ASSIGN (<ф.п.>, <имя файла или л.у.>);

Здесь <ф.п.> - файловая переменная (правильный идентификатор, объявленный в программе как переменная файлового типа);

<имя файла> - текстовое выражение, содержащее имя файла

Инициализация файла

Инициировать файл означает указать для этого файла направление передачи данных. В Турбо Паскале можно открыть файл для чтения, для записи информации, а также для чтения и записи одновременно.

Процедура APPEND

инициирует запись в ранее существовавший текстовый файл для его расширения, при этом указатель файла устанавливается в его конец. Процедура APPEND применима только к текстовым файлам, т.е. их файловая переменная должна иметь тип TEXT . Процедурой APPEND нельзя инициировать запись в типизированный или нетипизированный файл. Если текстовый файл ранее уже был открыт с помощью RESET или REWRITE, использование процедуры APPEND приведет к закрытию этого файла и открытию его вновь, но уже для добавления записей.

Процедура CLOSE

Закрывает файл, однако связь файловой переменной с именем файла, установленная ранее процедурой ASSIGN, сохраняется. Формат обращения:

CLOSE (<ф.п.>)

При создании нового или расширении старого файла процедура обеспечивает сохранение в файле всех новых записей и регистрацию файла в каталоге. Функции процедуры CLOSE выполняются автоматически по отношению ко всем открытым файлам при нормальном завершении программы. Поскольку связь файла с файловой переменной сохраняется, файл можно повторно открыть без дополнительного использования процедуры ASSIGN.

Типы файлов

В Паскале различают текстовые, типизированные и нетипизированные файлы

Текстовые файлы предназначены для хранения текстовой информации. Для доступа к записям применяются процедуры READ, READLN, WRITE, WRITELN.

В этом случае осуществляется обращение к дисковому файлу или логическому устройству, связанному с переменной процедурой ASSIGN.

Приступая к решению задач этого раздела, следует вспомнить, что:

1. в программе, которая выводит результаты в файл или читает исходные данные из файла, должна быть объявлена файловая переменная типа text;
2. для доступа к конкретному файлу файловую переменную нужно связать с этим файлом (делается это при помощи инструкции assign);
3. для того, чтобы файл был доступен, его надо открыть (для ЧТЕНИЯ С ПОМОЩЬЮ ИНСТРУКЦИИ reset, ДЛЯ ЗАПИСИ — rewrite, для добавления — append);
4. при работе с файлами возможны ошибки, например, из-за того, что программа пытается открыть файл, которого нет, поэтому после каждой инструкции, которая может привести к возникновению ошибки, желательно, используя функцию iOResuit, проверять код завершения операции с файлом: чтобы программа могла контролировать результат выполнения операции с файлом, в ее текст надо поместить директиву

5. запись в файл выполняют инструкции write и writein, чтение — read и readin, причем в качестве первого параметра этих инструкций следует указывать файловую переменную;
6. по завершении работы с файлом его нужно обязательно закрыть инструкцией close;
файл, созданный программой, в которой тип файловой переменной объявлен как text, можно просмотреть при помощи редактора текста.

Пример 1. Написать программу, которая на сменном диске компьютера (A:) создает файл numbers.txt и записывает в него 5 введенных пользователем целых чисел.

```
{ Создает на диске A: файл и записывает в него5 целых чисел, введенных пользователем }
var f: text; { текстовый файл }
n: integer; { число }
i: integer; { счетчик чисел }
begin
writeln('Создание файла');
writeln('Введите пять целых чисел.1);
writeln('После ввода каждого числа нажимайте <Enter>');
Assign(f,'a:\numbers.txt');
Rewrite(f); { открыть в режиме перезаписи } for i:=l to 5 do begin write('->');
readln(n);
writeln(f,n); end;
close(f); { закрыть файл }
writeln('Введенные числа записаны в файл ','a:\numbers.txt'); readln;
end.
```

Пример 2. Написать программу, которая выводит на экран содержимое файла a:\numbers.txt.

```
{ Выводит на экран содержимое файла a:\numbers.txt } var
f: text; { текстовый файл } n: integer; { число } begin
writeln('Содержимое файла a:\nunibers.txt1);
writeln ('----------------------------') ;
Assign(f,'a:\numbers.txt'); Reset(f); { открыть файл для чтения } While not EOF(f) do { пока не достигнут конец файла } begin
readln(f,n); { прочитать число из файла } writeln(n); { вывести прочитанное число на экран } end; Close(f); writeln ('-readln;
закрыть файл
end.
```

Пример 3. Вычисляет среднее арифметическое чисел, находящихся в файле a:\numbers.txt

```
var
f: text; { текстовый файл } n: integer; { число, прочитанное из файла } kol: integer; { кол-во прочитанных чисел } sum: integer; { сумма прочитанных чисел ) sa: real; { среднее арифметическое }
begin
writeln('Вычисление среднего арифметического чисел, writeln('находящихся в файле a:\numbers.txt'); writeln('Чтение из файла. Подождите.'); sum:=0; kol:-0;
Assign(f,'a:\numbers.txt'); Reset (f); { открыть файл для чтения } While not EOF(f) do { пока не достигнут конец begin
readln(f,n); { прочитать число из файла }
sum:=sum+n;
kol:=kol+l; end;
Close(f); { закрыть файл } sa:=sum/kol;
writeln('Прочитано чисел: ',kol); writeln('Сумма чисел: ',sum) ; writeln('Среднее арифметическое: ',sa:9:2); readln;
end.
```

Задание

1. Создать файл, состоящий из вещественных чисел. Определить количество нулевых значений в этом файле
2. Дан текстовый файл, содержащий произвольный текст. Определить чего в нем больше: русских букв или цифр.
3. Дан файл, содержащий текст на русском языке. Определить, входит ли заданное слово в указанный текст.

Контрольные вопросы

1. Что такое файловая переменная?
2. Какова структура файла?
3. Как описывается файловая переменная в программе?
4. Что такое типизированный файл?
5. Что такое нетипизированный файл?
6. Что такое текстовый файл?
7. Как объявляется текстовый файл в программе?
8. Назовите алгоритм работы с текстовым файлом

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
СОЗДАНИЕ ПРОГРАММ РАБОТЫ С ОДНОМЕРНЫМИ МАССИВАМИ НА ЯЗЫКЕ С++

Цель работы: Научиться создавать программы обработки одномерных массивов С++

Необходимые материалы и оборудование: ПК, среда программирования CodeBlocks

Пояснения к работе:

Массив – это совокупность однородных элементов. расположенных последовательно в памяти. Элемент массива – переменная с именем и индексом, заключенным в квадратные скобки.

Объявление массива должно включать: тип каждого элемента; название массива; в квадратных скобках – число элементов в массиве.

Примеры объявления массивов:

int sales[60]; /* массив из 60 значений типа int */

float data[10];/* массив из 10 значений типа float */

Доступ к отдельному элементу массива осуществляется с помощью *индекса*. Индекс описывает позицию элемента внутри массива.

Индексы элементов массивов в С начинаются с нуля

Поэтому, например, **sales[12]** – это 13-ый по порядку элемент массива, а не 12-ый. Индексы могут быть целыми числами или целочисленными выражениями.

Размерность массива определяется количеством индексов, необходимых для доступа к элементу массива. У элементов одномерного массива один индекс, у двумерного – два и т.д. Двухмерный массив представляет собой список одномерных массивов. Пример объявления двумерного массива (5 строк, 3 столбца): **int a[5][3];** Каждый элемент массива должен иметь столько индексов, сколько их в объявлении массива (например, в трехмерном массиве элемент должен иметь 3 индекса).

Фактически это объявление массива из 5 одномерных массивов по 3 элемента в каждом.

Имя массива одновременно является указателем на первый элемент массива.

Пример: Ввод одномерного массива и поиск минимального элемента

Написать программу, которая выполняет ввод значений одномерного массива и вывод введенного массива. Пример программы:

```
// ex44_01 - ввести массив и найти минимальный элемент

#include <iostream>
#include <stdio.h>
#define SIZE 10
    using namespace std;
    int main()
    {
      setlocale(LC_ALL,"Russian");
float m[SIZE], min;
int i, mini; // счетчик
cout << "Ввод элементов массива" << endl ; for (i=0; i<10; i++)
    {
      cout << "m[" << i << "] = "; cin >> m[i];
    }
     // для контроля: вывод массива for (i=0; i<10; i++)
    cout << "m[" << i << "]=" << m[i] <<" "; cout << endl;
// определение минимального элемента
min = m[0]; mini = 0;
    for(i=1; i<10; i++) if (min > m[i])
    {
      min = m[i]; mini = i;
    }
cout << "минимальный элемент массива: " << min <<endl; cout << "его индекс: " <<
mini << endl;
      getchar(); getchar(); return 0;
    }
```

Задание

Во всех программах предусмотреть ввод массива и контрольный вывод введенного массива.

Задание 1. Создание простейших программ

Создать программу согласно вариантам:

1. Напишите программу, которая вводит с клавиатуры 10 действительных чисел, и организовывает их хранение в массиве. После этого определить сумму элементов, значение которых больше среднего арифметического элементов массива. Указание. Сначала найти сумму элементов массива и вычислить среднее значение. Затем еще раз просмотреть массив и найти требуемую сумму.

2. Напишите программу, которая сначала вводит десять чисел в одномерный массив, а затем складывает отдельно все положительные элементы этого массива, отдельно отрицательные элементы. Указание. Объявить две переменных для сумм. Просмотреть в цикле массив, выполняя суммирование в зависимости от знака элемента.

3. Разработайте программу, которая будет вводить с клавиатуры 10 чисел float и сохранять их в некотором одномерном массиве. Затем у пользователя запрашивается ввод ещё одного "контрольного" числа. Проверить, находится ли введенное с клавиатуры число среди элементов массива.

4. Напишите программу, которая вводит с клавиатуры 10 реальных чисел, организовывает их хранение в одномерном массиве, а затем определяет сумму элементов с чётными индексами. Указание. Использовать конструкцию i+=2 в части модификации оператора цикла.

5. Напишите программу, которая вводит с клавиатуры 10 чисел float, организовывает их хранение в одномерном массиве, а затем определяет количество положительных и отрицательных элементов массива.

6. Напишите программу, которая вводит с клавиатуры 10 чисел float, организовывает их хранение в одномерном массиве, а затем меняет местами максимальный элемент с последним. Вывести измененный массив на экран.

7. Напишите программу, которая вводит с клавиатуры 10 чисел, организовывает их хранение в одномерном массиве, а затем выводит содержимое массива в обратном порядке с указанием индекса (номера) каждого элемента.

8. Напишите программу, которая вводит с клавиатуры 10 целых чисел, организовывает их хранение в одномерном массиве, а затем определяет сумму первых пяти и вторых пяти элементов массива.

9. Напишите программу, которая вводит с клавиатуры 10 действительных чисел, организовывает их хранение в массиве и определяет разность между максимальным и минимальным элементом массива.

10. Напишите программу, которая вводит с клавиатуры 10 целых чисел, организовывает их хранение в массиве и определяет количество чётных и количество нечётных элементов в массиве.

11. Напишите программу, которая вводит с клавиатуры 10 целых чисел, организовывает их хранение в массиве и заменяет нулями максимальный и минимальный эдементы. Указание. Просмотреть массив, одновременно с поиском максимального и минимального элементов запоминать их индексы. Затем присвоить нулевое значение элементам с указанными индексами.

12. Напишите программу, которая вводит с клавиатуры 10 действительных чисел, организовывает их хранение в массиве и определяет количество элементов, принадлежащих вводимому с клавиатуры диапазону.

13. Напишите программу, которая вводит с клавиатуры 10 действительных чисел, организовывает их хранение в массиве и определяет, являются ли элементы массива членами арифметической прогрессии. Указание. Последовательно сравнивать три элемента массива: m[i], m[i+1] и m[i+2] для значений i от 0 до N-3

14. Ввести массив из 10 положительных целых чисел. Отобразить на экране элементы массива, а рядом – строки из звездочек, в которых число звездочек равно значению элемента массива.

15. Написать программу, которая вычисляет среднеарифметическое значение без учета максимального и минимального элементов. Указание. Найти и запомнить индексы максимального и минимального элементов, затем выполнить суммирование с проверкой условия (индекс!=maxi И индекс!=mini, где maxi и mini – индексы).

16. Написать программу, которая определяет, сколько раз введенное с клавиатуры число встречается в массиве

17. С клавиатуры вводится массив из 10 вещественных чисел. Записать во второй массив на те же места нуль, если элемент отрицателен, и единицу– если положителен

Задания 2

1. Ввести массив из 10 положительных целых чисел (от 0 до 15). Отобразить на экране строку с элементами массива, а под каждым из них – столбик из звездочек, в котором число звездочек равно значению элемента массива.

2. Дан одномерный массив целых чисел размерностью 10. Определите количество положительных групп в этом массиве. (Группой называется последовательность, состоящая из 2-х и более положительных чисел находящихся рядом.).

Указание: Вначале просматриваем массив и ищем положительный элемент, у которого «сосед справа» тоже положителен (другими словами, ищем пару положительных элементов). Если такая пара обнаружена, увеличиваем счетчик групп n на единицу и продолжаем перемещение по массиву, пока очередной элемент положителен. Достигнув последнего элемента в группе (т.е. очередной элемент оказался не положительным), снова начинаем поиск пары положительных элементов.

3. Преобразовать свою программу из задания 1, используя указатели на элементы массива.

4. С клавиатуры вводится массив из 10 вещественных чисел. Выполнить два прохода сортировки методом вставок: найти минимальный элемент и поменять его местами с первым элементом, затем найти минимальный элемент в оставшейся части массива и поменять его местами со вторым элементом.

5. С клавиатуры вводится массив из 10 вещественных чисел. Выполнить один проход сортировки методом пузырька: сравнить элемент со следующим, если следующий элемент меньше, поменять их местами.

6. Ввести массив из 10 целых чисел. Проверить, образуют ли элементы массива неубывающую последовательность. (Указание: в цикле с постусловием сравнивать соседние элементы, пока условие a[i]<a[i+1] не будет нарушено или будет достигнут конец массива).

Контрольные вопросы

1. Что такое массивы?

2. Объявление массива в языке C++.

3. Как индексируются массивы в языках C/C++ ?

4. Сколько индексов должно быть у элемента массива?

5. Какая связь между указателями и массивами?

6. Какие типы данных допустимы для индексов массива? 7. Как размещаются в памяти элементы массива?

8. Как обратиться к элементу массива?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
ЛОГИЧЕСКИЙ ТИП ДАННЫХ. ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ, ОПЕРАЦИИ ОТНОШЕНИЯ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ НА ЯЗЫКЕ PYTHON

Цель работы: Изучение логического типа данных и логических выражений на языке Python.

Необходимые материалы и оборудование: ПК, среда программирования PyCharm

Пояснения к работе:

Операции сравнения в Python

Ниже приведена таблица, в которой указаны основные операции сравнения, которые можно применять на языке Python (см. табл. 1).

Таблица 1. Операции сравнения в языке Python

Операция сравнения	**Функция**
==	Возвращает **True**, если оба операнда равны. Иначе возвращает **False**
!=	Возвращает **True**, если оба операнда НЕ равны. Иначе возвращает **False**
>	Возвращает **True**, если первый операнд больше второго
<	Возвращает **True**, если первый операнд меньше второго
>=	Возвращает **True**, если первый операнд больше или равен второму
<=	Возвращает **True**, если первый операнд меньше или равен второму

Рассмотрим примеры использования каждого оператора сравнения (рис. 1, рис. 2, рис. 3, рис. 4, рис. 5, рис. 6):

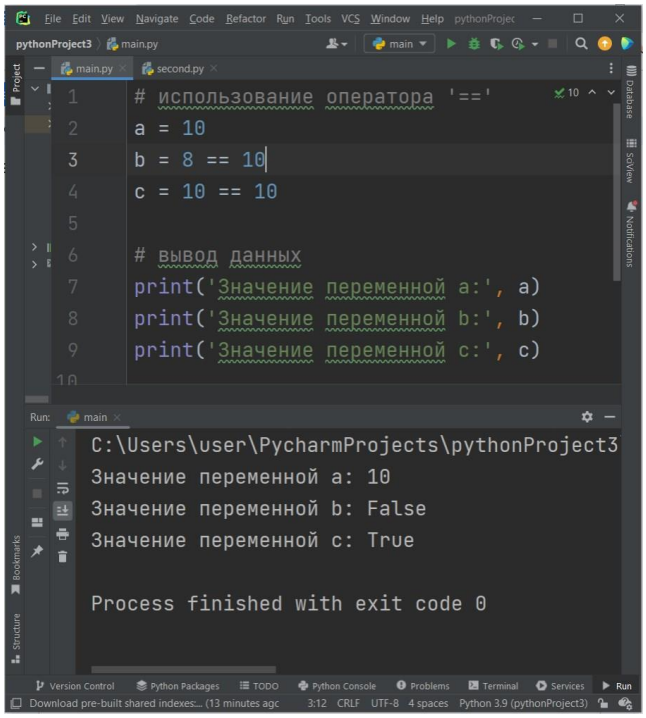


Рисунок 1 - Пример работы программы с использованием оператора сравнения ‘==’

Произведем **анализ работы** данной программы: в первой строке указан однострочный комментарий. В строке 2 вводится переменная с именем **‘a’**, в которую записывается знач ение**10**. В строке 3 вводится переменная с именем **‘b’**, в которую записывается значение **8**, после чего применяется оператор **‘==’**, справа от которого пишется значение **10**. Значение данной переменной, в таком случае, будет **False**. Если значение после знака ‘=’ будет точн о таким же, как и значение после оператора присваивания ‘==’, то значение переменнойбу дет **True** (как можно заметить при выводе переменной с именем **‘c’**). В строке 4 вводится переменная с именем **‘c’**.

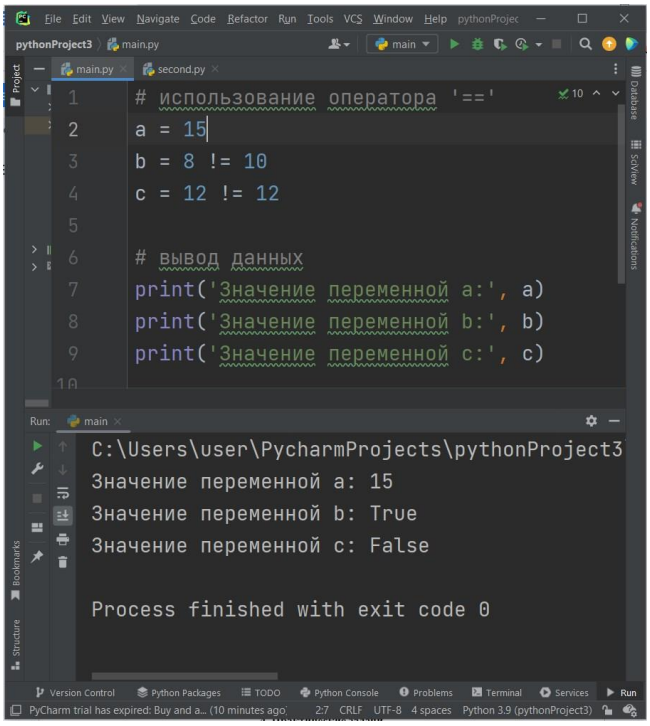


Рисунок 2 - Пример работы программы с использованием оператора сравнения ‘!=’

В последнем примере наоборот - если значение после знака ‘=’ будет точно таким же, к ак и значение после оператора присваивания ‘!=’, то значение переменной будет **False** (ка кможно заметить при выводе переменной с именем **‘c’**). Принцип работы такой же, у нас е сть три переменных со значениями **15**, **8** и **12**.

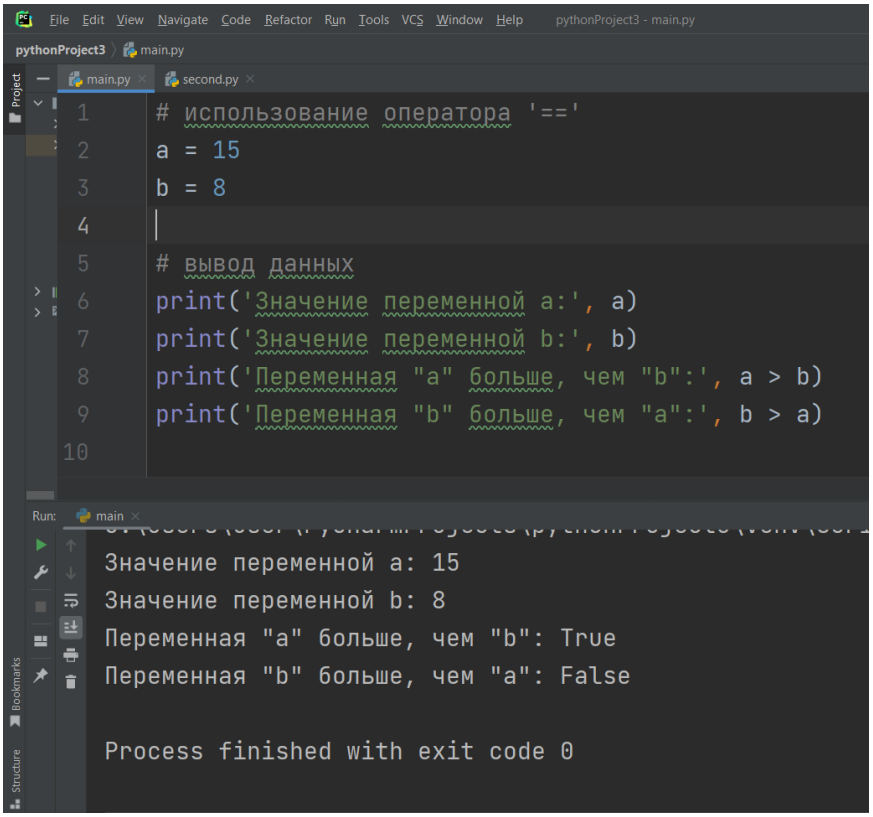


Рисунок 3 - Пример работы программы с использованием оператора сравнения ‘>’

В данном примере используются две переменные: **‘a’** со значением **15** и **‘b’** со значение м **8**. В строках 6 и 7 выводятся значения переменных **‘a’** и **‘b’**. В строке 8 мы задаем вопро сконсоли - значение переменной **‘a’** больше, чем значение переменной **‘b’**. Это правда, по этому в выводе данной строки можно заметить значение **True**. Обратный вопрос задается встроке 9, результатом является, соответственно, значение **False**.

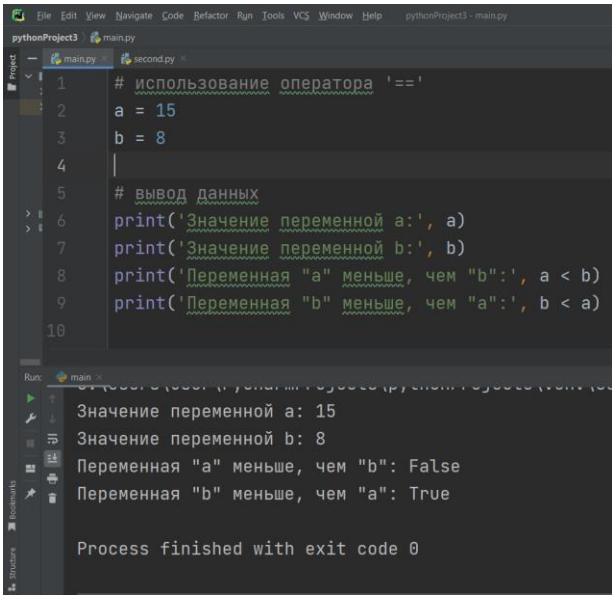


Рисунок 4 - Пример работы программы с использованием оператора сравнения ‘<’

В данном примере обратная ситуация: в строках 8 и 9 применяется оператор сравнения **‘<’ (меньше)**. Соответственно, когда результат операции является правдой (истиной), как встроке 9, в консоли можно заметить значение **True**. Соответственно, в строке 8 можно за метить значение **False**.

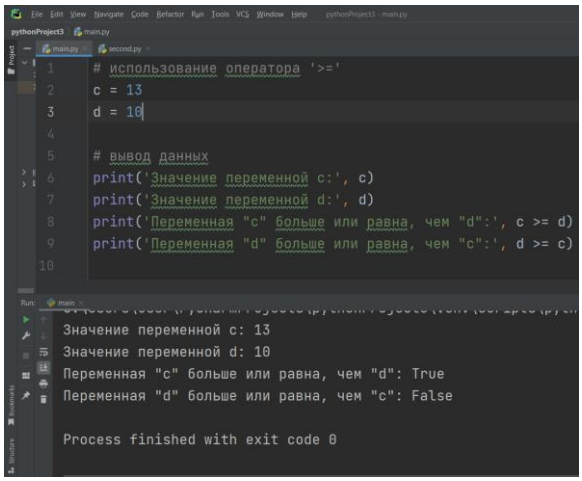


Рисунок 5 - Пример работы программы с использованием оператора сравнения ‘>=’

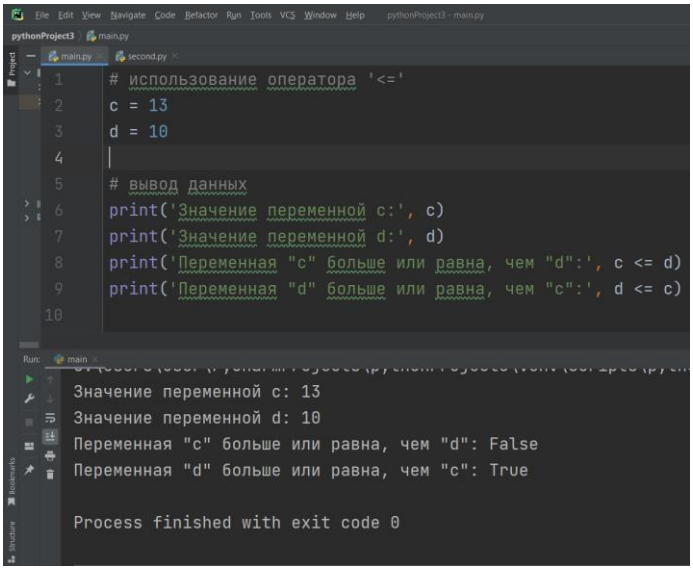


Рисунок 6 - Пример работы программы с использованием оператора сравнения ‘<=’

В последних четырех примерах алгоритм примерно одинаковый: вводится две перемен ные; выводятся их значения, после чего применяются определенные операторы присваива ния. Врезультате применения операторов присваивания, на консоли можно заметить либо значение **True**, либо **False**. Операции сравнения могут сравнивать самые различные объек ты - строки, числа, логические значения, однако, **оба операнда операции должны предст авлять один и тот же тип данных.**

Приведем примеры использования операторов сравнения, которые применяются к с трокам (рис. 7, рис. 8):

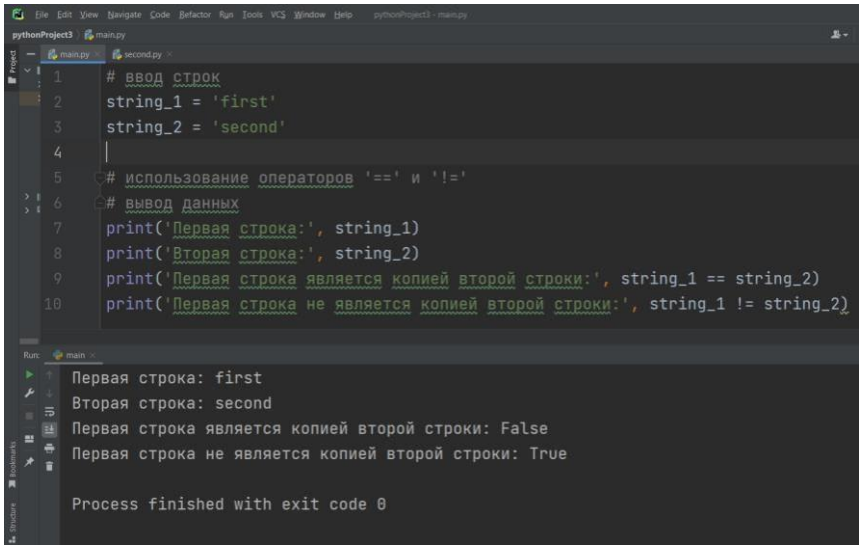


Рисунок 7 - Пример работы программы
с использованием оператора сравнения и строк

Как можно заметить, в строках 2 и 3 вводятся две строки с именами **‘string_1’** и **‘string _2’**. Затем, в строках 7 и 8 выводятся содержимое обеих строк. Кроме того, в строках 9 и 1 0 происходит следующее: в строке 9 задается вопрос - содержимое строки **‘string_2’** являе

тся копией содержимого строки **‘string_1’**? Это ложь, поэтому на консоли можно заметить значение **False**. В строке 10 задается обратный вопрос, поэтому, соответственно, на консо ли можно заметить значение **True**.

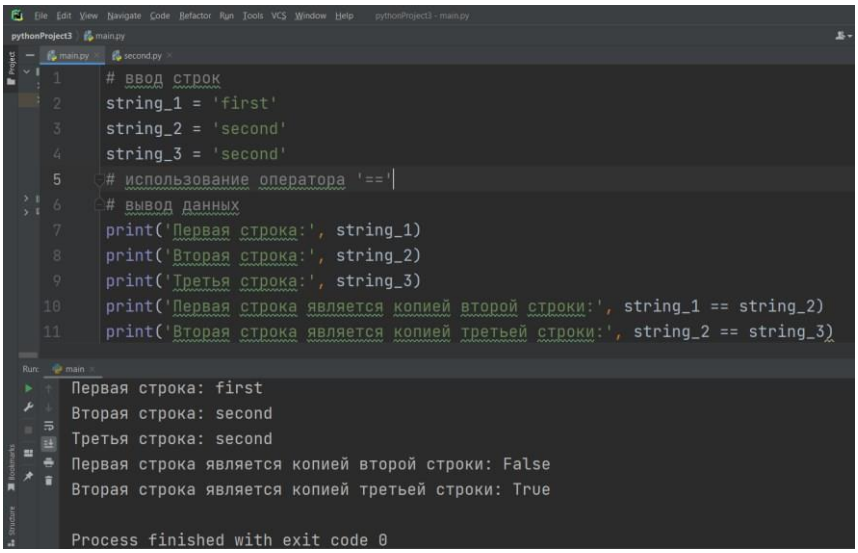


Рисунок 8 - Пример работы программы
с использованием оператора сравнения и строк

В данном примере вводится еще одна строка с именем **‘string_3’** и содержимым, идент ичным содержимому строки с именем **‘string_2’**. Соответственно, когда консоли задаетсяв опрос, одинаковы ли содержимые переменных **‘string_2’** и **‘string_3’**, в последней строке консоли можно заметить значение **True**. Соответственно, когда спрашивается ободнознач ности содержимого строк **‘string_1’** и **‘string_2’**, на консоли можно заметить значение **Fal se**.

Язык Python обладает всеми возможностями, которых следует ожидать от современ ного языка программирования. В тоже время, язык Python является объектно-ориентированным языком программирования (ООП). ООП является современным подход ом к решению задач с помощью вычислительных машин. В рамках ООП собственнаяинфо рмация программы и команды, которые она передает компьютеру, записываются интуити вно понятным образом. Это не является единственным способом разработки программ, но в больших проектах, как правило, предпочтительный.

Логические операции в Python

Для создания составных условных выражений любой сложности применяются спец иальные операции, которые называются *логическими операциями*. Рассмотрим основныел огические операции, которые можно использовать на языке программирования Python:

- оператор **and** (логическое умножение) применяется, как правило, исключительно д ля двух операндов (рис. 9):

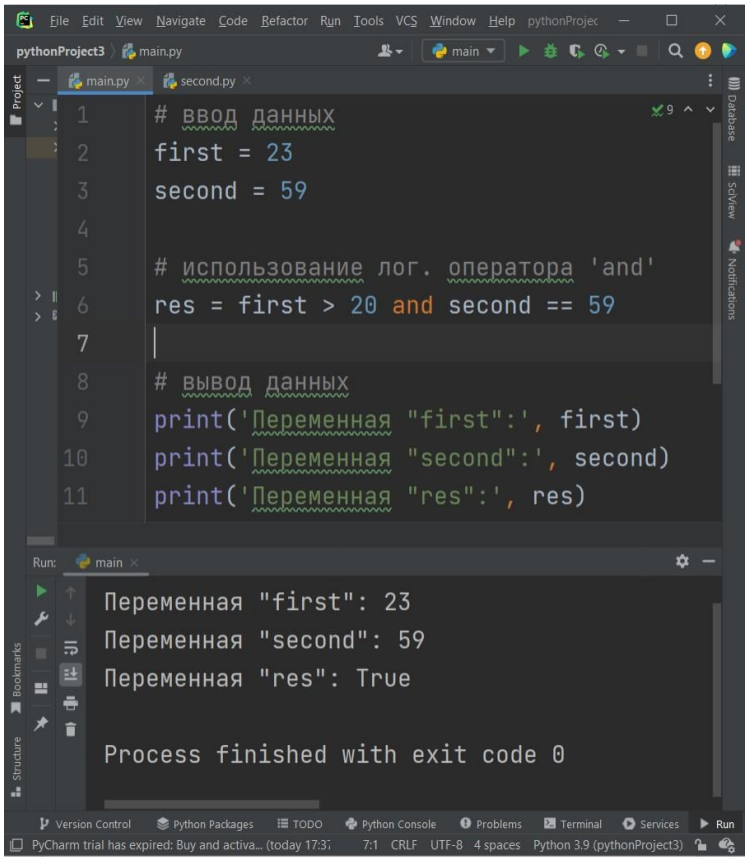


Рисунок 9 - Пример работы программы с использованием оператора *and*

В данном случае, оператор **and** сравнивает результаты двух выражений: **first > 20** и **second == 59**. Если оба этих выражений возвращают значение **True**, то оператор **and** возвращаетзначение **True** (как это можно заметить в третьей строке консоли). Рассмотрим еще один пример, где в консоли можно заметить значение **False** (рис. 10):

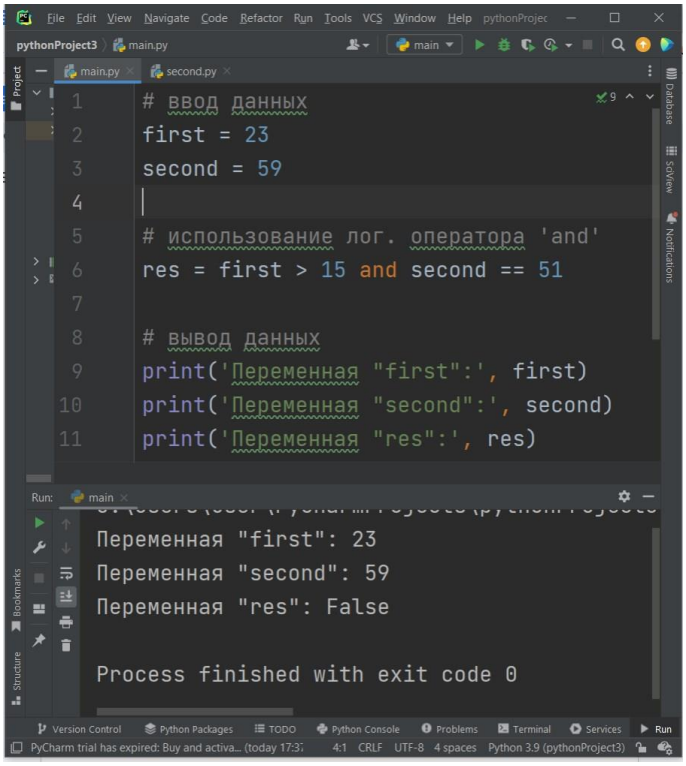


Рисунок 10 - Пример работы программы с использованием оператора *and*

- оператор **or** (логическое сложение), также может применяться к двум операндам с одинаковым типом (рис. 11):

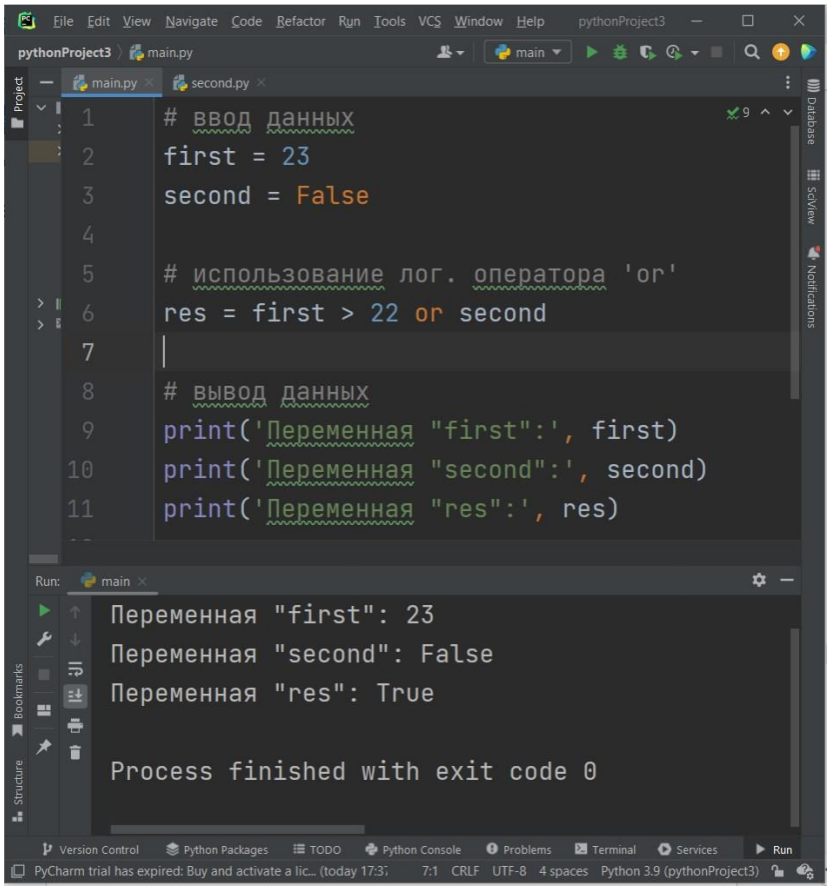


Рисунок 11 - Пример работы программы с использованием оператора *or*

Проведем анализ работы данной программы: в строке 2 вводится переменная с именем **‘first’** и значением **23**. В строке 3 вводится переменная с именем **‘second’** со значением **False**. Встроке 6 используется логический оператор **or**. Если значение переменной **‘first’** больше, чем **22**, выводится значение **True** (как в данном примере), в противном случае вывод итсязначение **False**.

Рассмотрим еще один пример использования оператора **or**, когда на консоли выводится значение **False** (рис. 12):

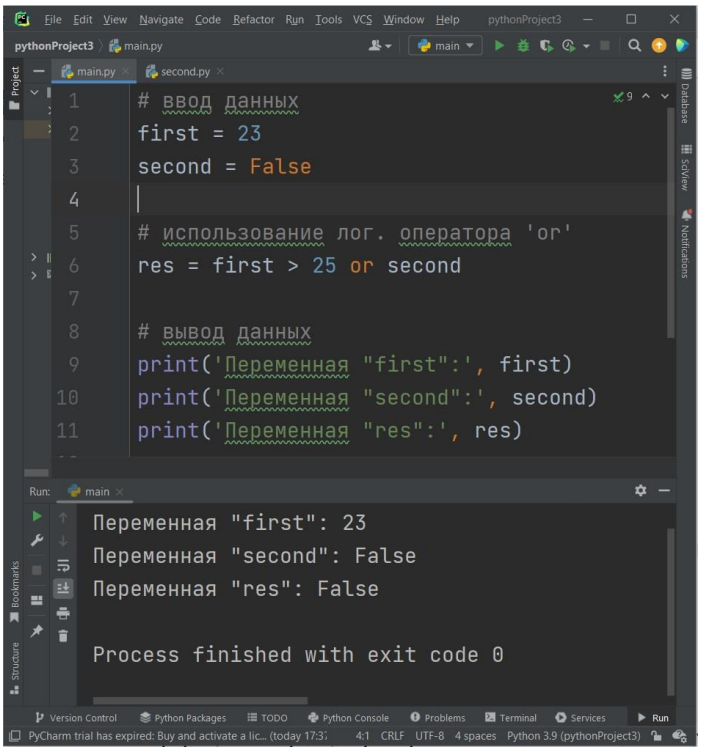


Рисунок 12 - Пример работы программы с использованием оператора *or*

В данном случае, наоборот, значение переменной **‘first’** меньше, чем **25**, поэтому в третьей строке консоли выводится значение **False**. В таком случае, если выражение, котороена ходится слева от оператора **or** истинно, тогда на консоли можно заметить значение **True**. В противном случае, можно заметить значение **False**.

- оператор **not** (логическое отрицание), который возвращает значение **True**, если выражение равно **False** (рис. 13):

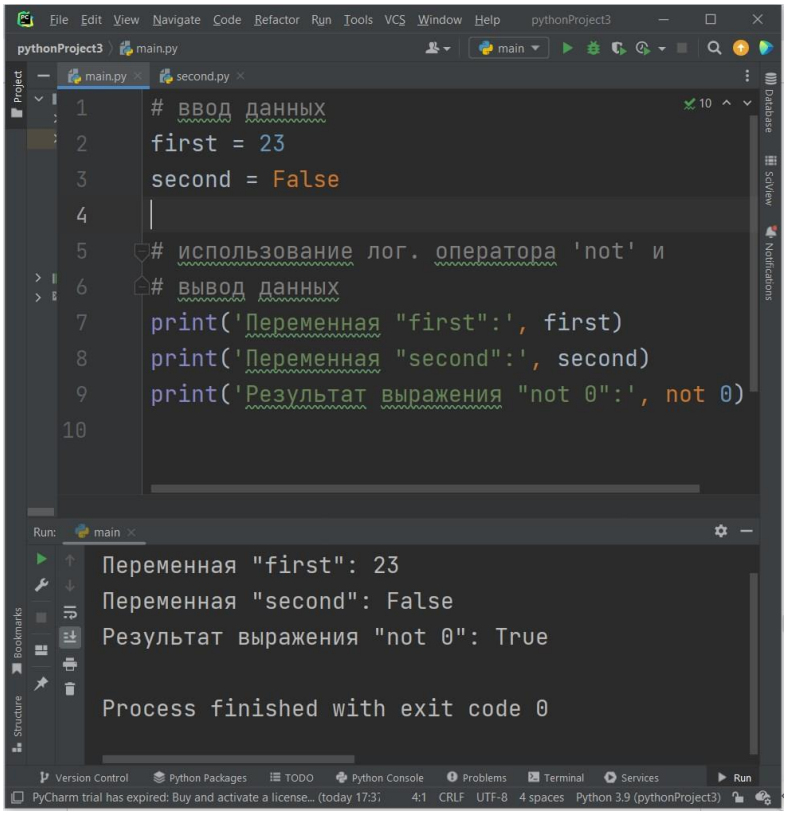


Рисунок 13 - Пример работы программы с использованием оператора *not*

Рассмотрим еще один пример использования оператора **or**, когда на консоли выводится значение **False** (рис. 14):

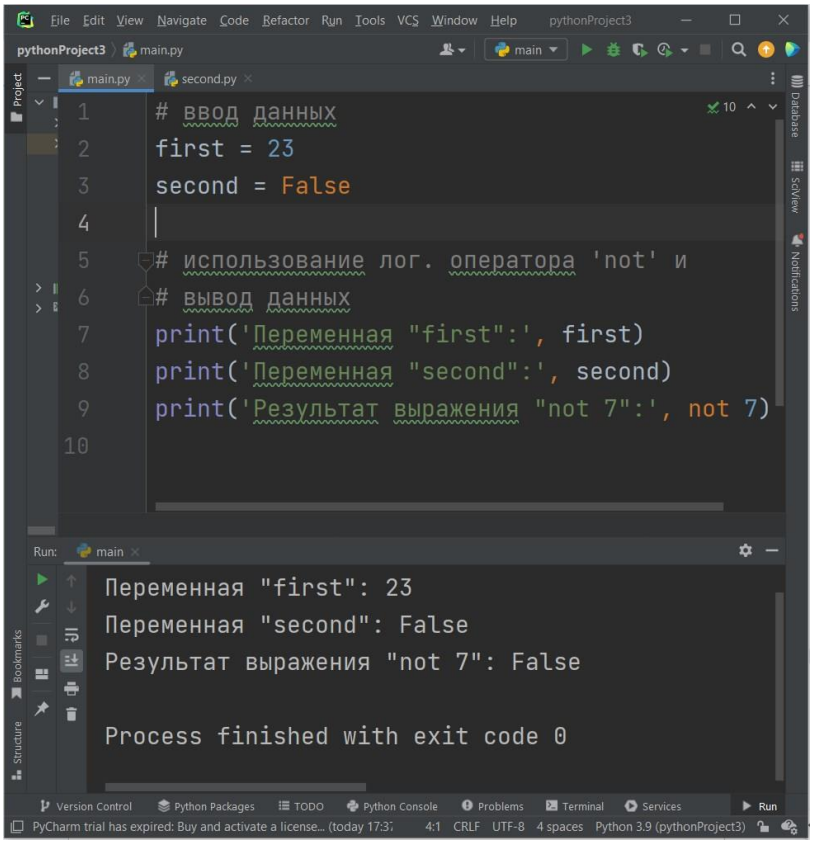


Рисунок 14 - Пример работы программы с использованием оператора *not*

Применение оператора in

Данный оператор возвращает значение **True**, если в некотором наборе значений ест ь определенное значение. Рассмотрим синтаксис применения оператора **in** (рис. 12):

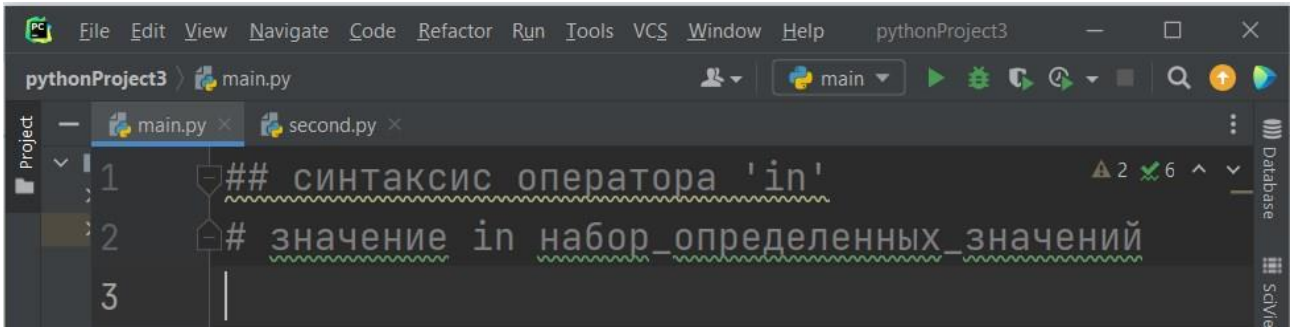


Рисунок 12 - Синтаксис применения оператора *in*

Можно, например, задать строку при помощи набора символов. С помощью оператора **i n** возможно проверить - есть ли в строке подстрока (набор из меньшего числа символов, ч емвся строка) (рис. 13, рис. 14):

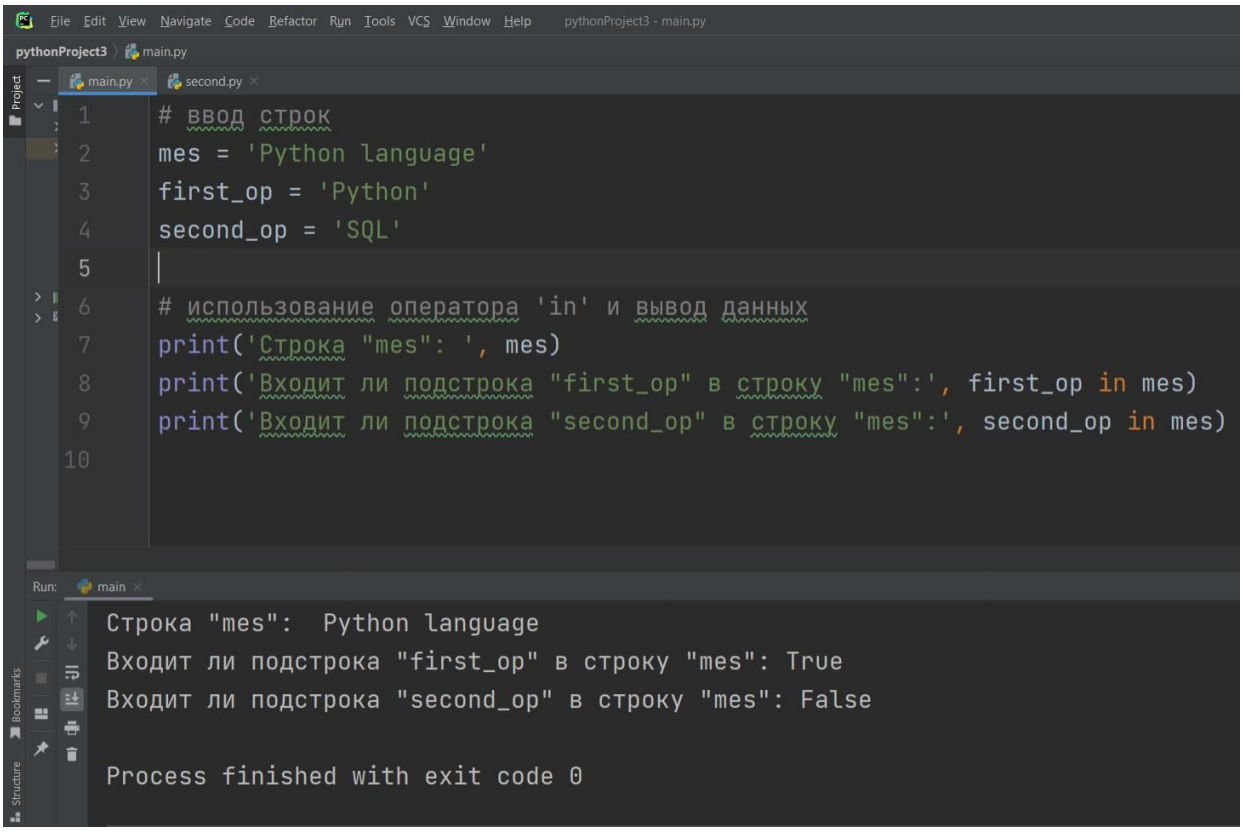


Рисунок 13 - Пример работы программы с использованием оператора *in*

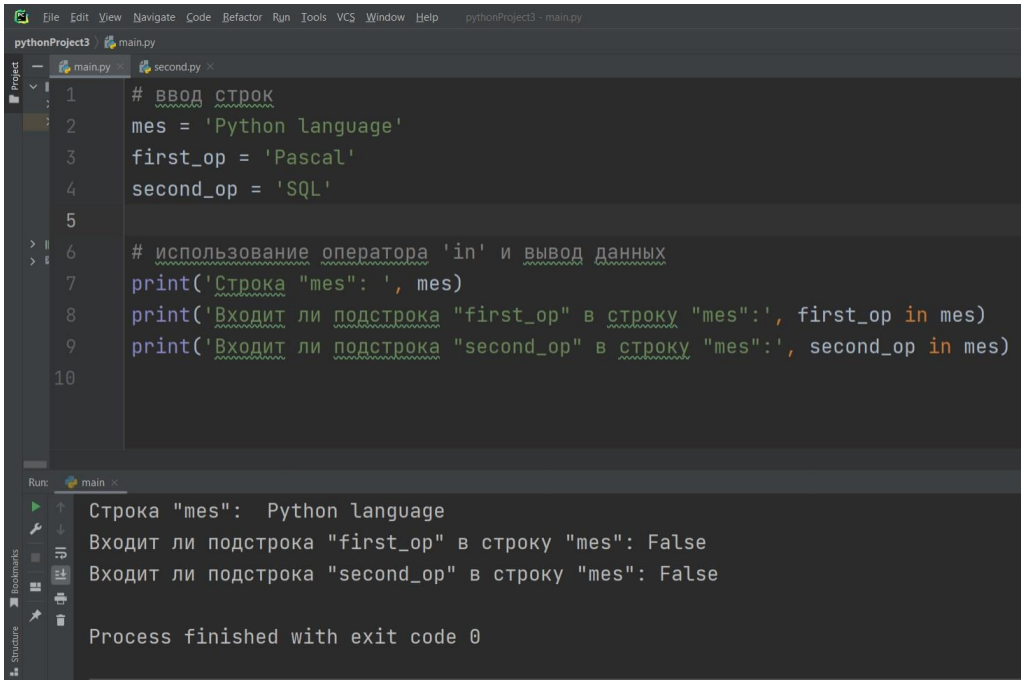


Рисунок 14 - Пример работы программы с использованием оператора *in*

Цикл while

Цикл **while** проверяет истинность некоторого условия, и если условие истинно, то выполняются инструкции цикла. Он имеет следующее формальное стандартное определение (рис. 15):

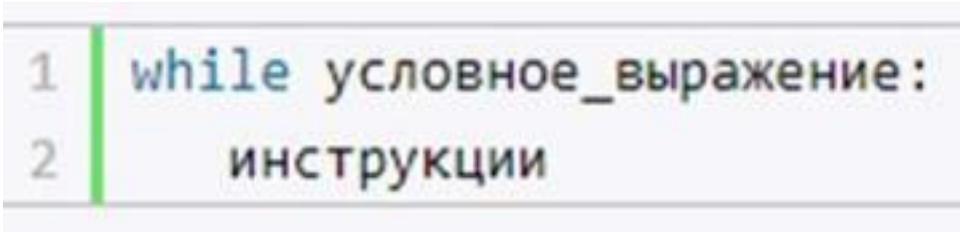


Рисунок 15 - Конструкция цикла while

После ключевого слова **while** указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу **while**, располагаются на последующих строках и должны иметь отступ от начала ключевого слова **while** (рис. 16):

```
1 number = 1
2
3 while number < 5:
4     print(f"number = {number}")
5     number += 1
6 print("Работа программы завершена")
```

Рисунок 16 - Пример работы цикла while

В данном случае цикл while будет выполняться, пока переменная number меньше 5. Сам блок цикла состоит из двух инструкций (рис. 17):

```
1 print(f"number = {number}")
2 number += 1
```

Рисунок 17 - Конструкции цикла

Здесь стоит обратить внимание на то, что они имеют отступы от начала оператора **while** - в данном случае от начала строки. Благодаря этому Python может определить, что они принадлежат циклу. В самом цикле сначала выводится значение переменной number, а потом ей присваивается новое значение. Также стоит не забывать о том, что последняя инструкция **print("Работа программы завершена")** не имеет отступов от начала строки, поэтому она не входит в цикл while.

Весь процесс цикла можно представить следующим образом:

1. Сначала проверяется значение переменной **number** - больше ли оно 5. И поскольку вначале переменная равна 1, то это условие возвращает True, и поэтому выполняются инструкции цикла;
2. Инструкции цикла выводят на консоль строку number = 1. И далее значение переменной number увеличивается на единицу - теперь она равна 2. Однократное выполнение блока инструкций цикла называется **итерацией**. То есть таким образом, в цикле выполняется первая **итерация**;
3. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 2, поэтому выполняются инструкции цикла;

4. Инструкции цикла выводят на консоль строку number = 2. И далее значение переменной number опять увеличивается на единицу - теперь она равна 3. Таким образом, выполняется вторая итерация;

5. Опять проверяется условие number < 5. Оно по прежнему равно True, так как number = 3, поэтому выполняются инструкции цикла;

6. Инструкции цикла выводят на консоль строку number = 3. И далее значение переменной number опять увеличивается на единицу - теперь она равна 4. То есть выполняется третья итерация;

7. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 4, поэтому выполняются инструкции цикла;

8. Инструкции цикла выводят на консоль строку number = 4. И далее значение переменной number опять увеличивается на единицу - теперь она равна 5. То есть выполняется четвертая итерация;

9. И вновь проверяется условие number < 5. Но теперь оно равно False, так как number = 5, поэтому выполняются выход из цикла. Все цикл - завершился. Дальше уже выполняются действия, которые определены после цикла. Таким образом, данный цикл произведет четыре прохода или четыре итерации.

В итоге при выполнении кода мы получим следующий консольный вывод (рис. 18):

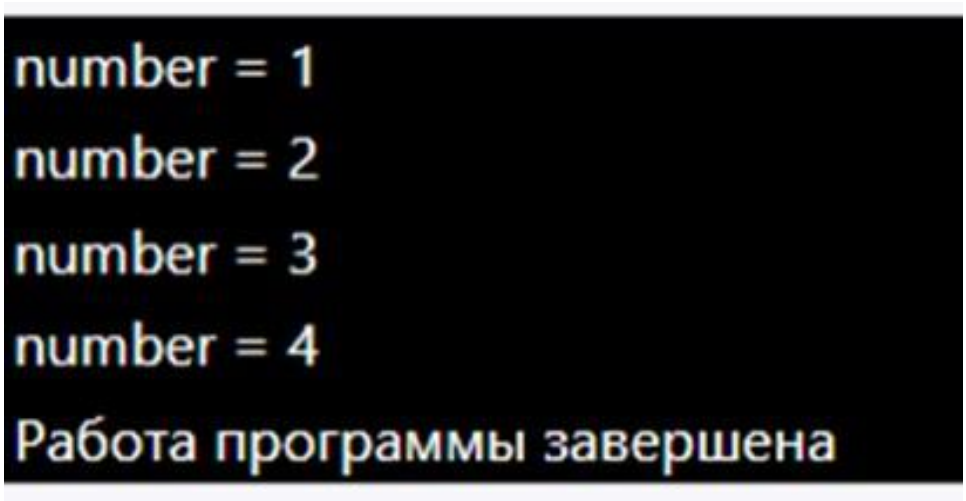


Рисунок 18 - Консольный вывод при использовании цикла while

Для цикла **while** также можно определить дополнительный блок **else**, инструкции которого выполняются, когда условие равно False (рис. 19):

```
number = 1

while number < 5:
    print(f"number = {number}")
    number += 1
else:
    print(f"number = {number}. Работа цикла завершена")
print("Работа программы завершена")
```

Рисунок 19 - Пример программы с циклом while

Задание

Вариант 1

Создать программу, в которой используется следующая строка: **‘mes’ = ‘Python’**. Кроме того, в программе используется подстрока: **‘first_op’**. В данной программе необходимо реализовать оператор **in** и проверить - входит ли подстрока с именем **‘first_op’** в строку **‘mes’**. Результаты продемонстрировать на консоли.

Вариант

Создать программу, в которой используются переменные с именами **‘a’**, **‘c’** и **‘e’**. Необходимо использовать специальный оператор **‘!=’**, относительно двух пар переменных - **‘a’** и **‘c’**, а затем **‘a’** и **‘e’**. Результаты продемонстрировать на консоли.

Вариант 3

Создать программу, в которой пользователь вводит три строки из символов. Необходимо вывести данные строки отдельно на консоль.

Вариант 4

Создать программу с использованием цикла **while** и оператора **continue**. Цикл **while** работает с переменной **‘z’**, значение которой изначально равно 3. Данный цикл выводит значение переменной **‘z’** во второй степени. При каждой итерации значение переменной **‘z’** увеличивается на единицу. Когда значение переменной становится равным 6, оператор **continue**приостанавливает работу цикла **while**.

Вариант 5

Создать программу, в которой используются переменные с именами **‘c’** и **‘d’**. Необходимо использовать специальный оператор **‘>=’**, относительно данных переменных. Результаты продемонстрировать на консоли.

Вариант 6.

Создать программу, в которой используются: переменная **‘q’** с типом **str**, переменная **‘w’** с типом **float** и переменная **‘e’** с типом **list.** Необходимо при помощи функции **type()** вывести на консоль типы данных переменных.

Вариант 7

Создать программу с использованием цикла **for**. Данный цикл перебирает значения переменной **‘x’** от 1 до 5 и переменной **‘y’** от 2 до 5 и выводит соответствующие значения на консоль.

Вариант 8

Создать программу в среде PyCharm, в которой вводятся с клавиатуры три переменные с типом данных **int**: **‘a’**, **‘b’** и **‘c’**. Затем в переменную **‘z’** записывается сумма значений данных переменных. Вывести на консоль значения введенных переменных и сумму.

Вариант 9

Создать программу, в которой пользователь вводит 5 строк из символов. Необходимо вывести данные строки отдельно на консоль.

Вариант 10

Создать программу, в которой используются переменные с именами **‘x’** и **‘y’**. Необходимо использовать специальный оператор **‘>=’**, относительно данных переменных. Результаты продемонстрировать на консоли.

Контрольные вопросы

1. Какую роль выполняет команда float в языке Python?
2. Что такое логическая операция?
3. Как можно передавать аргументы программе?
4. Какую роль выполняет функция boolean() в языке Python?
5. Какое минимальное количество аргументов можно передать программе на языке Python за один раз?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ
ИТЕРАТИВНЫЕ КОНСТРУКЦИИ. ЦИКЛ ДЛЯ ОБХОДА ПОСЛЕДОВАТЕЛЬНОСТЕЙ. БЕЗУСЛОВНЫЕ ОПЕРАТОРЫ ПЕРЕХОДА К СЛЕДУЮЩЕЙ ИТЕРАЦИИ.

Цель работы: Изучение итеративных конструкций и цикла для обхода последовательностей на языке Python

Необходимые материалы и оборудование: ПК, среда программирования PyCharm

Пояснения к работе:

Выход из цикла. Операторы break и continue

Для управления циклом мы можем использовать специальные операторы **break** и **continue**. Оператор **break** осуществляет выход из цикла. А оператор **continue** выполняет переход к следующей итерации цикла. Оператор **break** может использоваться, если в цикле образуются условия, которые несовместимы с его дальнейшим выполнением. Рассмотрим следующий пример (рис. 1):

```
number = 0
while number < 5:
    number += 1
    if number == 3 :    # если number = 3, выходим из цикла
        break
    print(f"number = {number}")
```

Рисунок 1 - Пример программы с использованием оператора break

Здесь цикл **while** проверяет условие **number < 5**. И пока **number** не равно 5, предполагается, что значение number будет выводиться на консоль. Однако внутри цикла также проверяется другое условие: **if number == 3**. То есть, если значение **number** равно 3, то с помощью оператора **break** выходим из цикла. И в итоге мы получим следующий консольный вывод (рис. 2):

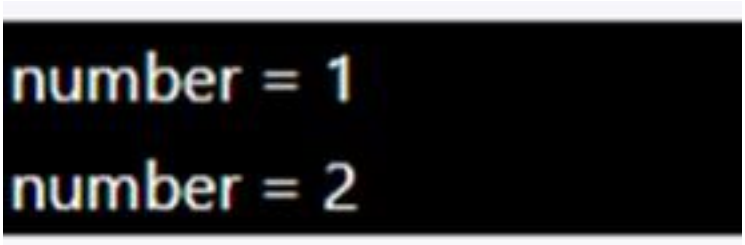


Рисунок 2 - Консольный вывод при работе с оператором break

В отличие от оператора **break** оператор **continue** выполняет переход к следующей итерации цикла без его завершения. Например, в предыдущем примере заменим **break** на **continue** (рис. 3):

```
number = 0
while number < 5:
    number += 1
    if number == 3 :    # если number = 3, переходим к новой итерации цикла
        continue
    print(f"number = {number}")
```

Рисунок 3 - Пример программы с использованием оператора continue

И в этом случае если значение переменной **number** равно 3, последующие инструкции после оператора **continue** не будут выполняться (рис. 4):

```
number = 1
number = 2
number = 4
number = 5
```

Рисунок 4 - Консольный вывод с использованием оператора continue

Вложенные циклы

Кроме того, одни циклы внутри себя могут содержать другие циклы. Рассмотрим п ример вывода простой таблицы умножения (рис. 5, рис. 6):

```
# ввод переменных для циклов
a = 1
b = 1
# конструкция внешнего цикла
while a < 5:
    # конструкция вложенного цикла
    while b < 5:
        # вывод произведений переменных
        print(a * b, end = '\t')
        # прибавление единицы к 'b' после очередной итерации
        b += 1
    # переход на новую строку
    print('\n')
    b = 1
    # прибавление единицы к 'a' после очередной итерации
    a += 1
```

Рисунок 5 - Листинг кода при работе с вложенными циклами

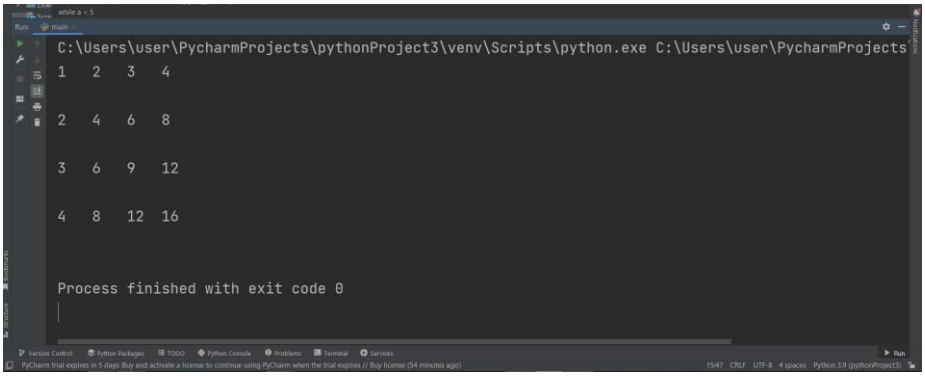


Рисунок 6 - Вывод консоли при работе с вложенными циклами

Произведем анализ данной программы: внешний цикл с конструкцией **‘while a < 5’** срабатывает столько раз, пока переменная **‘a’** не станет равна 10. Внутри данного цикла такжесрабатывает внутренний цикл **‘while b < 5’**. Внутренний цикл срабатывает до тех пор, пока значение переменной **‘b’** не станет равн о 5. Здесь стоит отметить, что все итерациивнутреннего цикла срабатывают в рамках одно й лишь итерации внешнего цикла. В каждой итерации внутреннего цикла на консоль (рис. 10) выводится произведение значений **‘a’** и**‘b’**. Когда внутренний цикл заканчивает свою р аботу, значение переменной **‘b’** сбрасывается в единицу, а значение переменной **‘a’** увели чивается на единицу. После этого происходитпереход к следующей итерации внешнего ци кла.

Приведем следующий пример вложенного цикла **for**, когда на консоли появляются всевозможные комбинации двух символов (рис. 7):

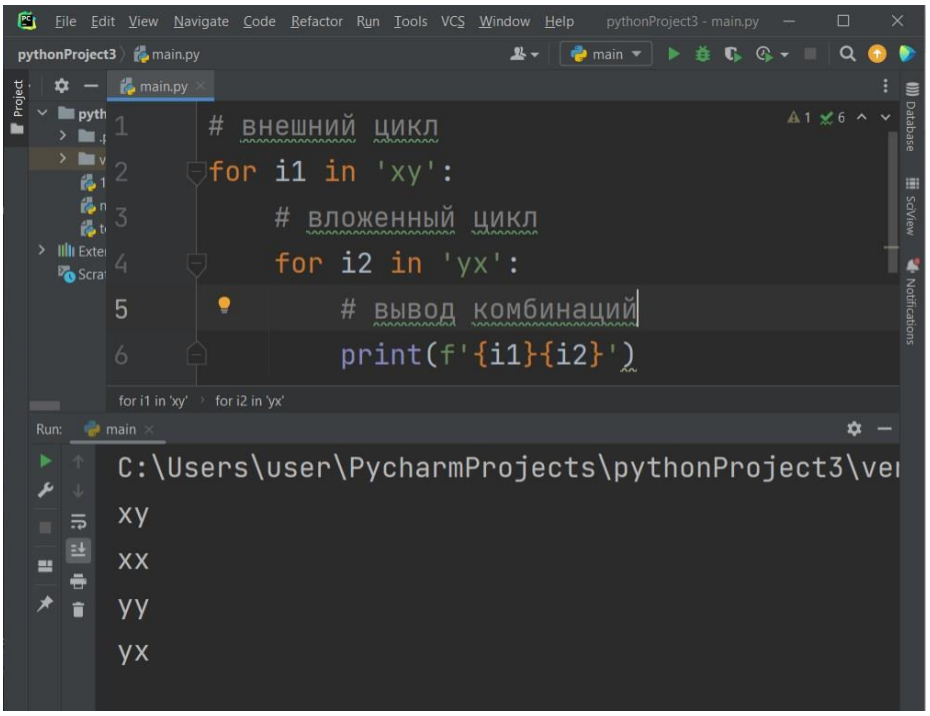


Рисунок 7 - Работа программы вложенного цикла for

Цикл while (повторение)

Цикл **while** проверяет истинность некоторого условия, и если условие истинно, то выполняются инструкции цикла. Он имеет следующее формальное стандартное определение (рис. 8):

```
while условное_выражение:
    инструкции
```

Рисунок 8 - Конструкция цикла while

После ключевого слова **while** указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу **while**, располагаются на последующих строках и должны иметь отступ от начала ключевого слова **while** (рис. 9):

```
number = 1

while number < 5:
    print(f"number = {number}")
    number += 1
print("Работа программы завершена")
```

Рисунок 9 - Пример работы цикла while

В данном случае цикл while будет выполняться, пока переменная number меньше 5.

Сам блок цикла состоит из двух инструкций (рис. 10):

```
print(f"number = {number}")
number += 1
```

Рисунок 10 - Конструкции цикла

Здесь стоит обратить внимание на то, что они имеют отступы от начала оператора **while** - в данном случае от начала строки. Благодаря этому Python может определить, что они принадлежат циклу. В самом цикле сначала выводится значение переменной number, а потом ей присваивается новое значение. Также стоит не забывать о том, что последняя инструкция **print("Работа программы завершена")** не имеет отступов от начала строки, поэтому она не входит в цикл while.

Весь процесс цикла можно представить следующим образом:

1. Сначала проверяется значение переменной **number** - больше ли оно 5. И поскольку вначале переменная равна 1, то это условие возвращает True, и поэтому выполняются инструкции цикла;
2. Инструкции цикла выводят на консоль строку number = 1. И далее значение переменной number увеличивается на единицу - теперь она равна 2. Однократное выполнение блока инструкций цикла называется **итерацией**. То есть таким образом, в цикле выполняется первая **итерация**;
3. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 2, поэтому выполняются инструкции цикла;
4. Инструкции цикла выводят на консоль строку number = 2. И далее значение переменной number опять увеличивается на единицу - теперь она равна 3. Таким образом, выполняется вторая итерация;
5. Опять проверяется условие number < 5. Оно по прежнему равно True, так как number = 3, поэтому выполняются инструкции цикла;
6. Инструкции цикла выводят на консоль строку number = 3. И далее значение переменной number опять увеличивается на единицу - теперь она равна 4;
7. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 4, поэтому выполняются инструкции цикла;
8. Инструкции цикла выводят на консоль строку number = 4. И далее значение переменной number опять увеличивается на единицу - теперь она равна 5. То есть выполняется четвертая итерация;
9. И вновь проверяется условие number < 5. Но теперь оно равно False, так как number = 5, поэтому выполняются выход из цикла. Все цикл - завершился. Дальше уже выполняются действия, которые определены после цикла. Таким образом, данный цикл произведет четыре прохода или четыре итерации.

В итоге при выполнении кода мы получим следующий консольный вывод (рис. 11):

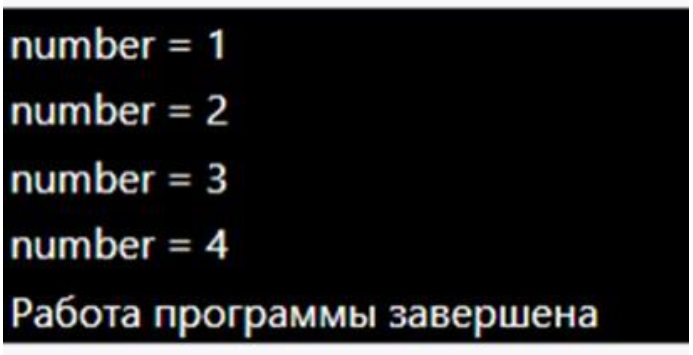


Рисунок 11 - Консольный вывод при использовании цикла while

Для цикла **while** также можно определить дополнительный блок **else**, инструкции которого выполняются, когда условие равно False (рис. 12):

```
number = 1

while number < 5:
    print(f"number = {number}")
    number += 1
else:
    print(f"number = {number}. Работа цикла завершена")
print("Работа программы завершена")
```

Рисунок 12 - Пример программы с циклом while

Задание

Вариант 1

Создать программу с использованием цикла **for**. Данный цикл перебирает значения переменной **‘a’** от 3 до 7 и переменной **‘b’** от 10 до 13 и выводит соответствующие значения на консоль.

Вариант 2

Создать программу, в которой происходит перебор элементов списка с именем **‘words’** при помощи цикла **while**.

Вариант 3

Создать программу, в которой реализован оператор выхода из цикла **break.** Изначально вводится переменная **‘num’** со значением **0**. При каждой итерации цикла значение переменной **‘num’** увеличивается на единицу. Когда значение переменной **‘num’** становится равным **3**, оператор **break** завершает работу цикла (на консоли должны быть выведены значения **1** и **2**).

Вариант 4

Создать программу с использованием цикла **while** и оператора **continue**. Цикл **while** работает с переменной **‘z’**, значение которой изначально равно 3. Данный цикл выводит значение переменной **‘z’** во второй степени. При каждой итерации значение переменной **‘z’** увеличивается на единицу. Когда значение переменной становится равным 6, оператор **continue**приостанавливает работу цикла **while**.

Вариант 5

Создать программу, в которой реализован оператор выхода из цикла **break.** Изначально вводится переменная **‘s’** со значением **0**. При каждой итерации цикла значение переменной **‘s’**увеличивается на единицу. Когда значение переменной **‘s’** становится равным **5**, оператор **break** завершает работу цикла (на консоли должны быть выведены значения с **1** по **4** включительно).

Вариант 6

Создать программу, в которой используются: переменная **‘q’** с типом **str**, переменная **‘w’** с типом **float** и переменная **‘e’** с типом **list.** Необходимо при помощи функции **type()** вывести на консоль типы данных переменных.

Вариант 7

Создать программу с использованием цикла **for**. Данный цикл перебирает значения переменной **‘x’** от 1 до 5 и переменной **‘y’** от 2 до 5 и выводит соответствующие значения на консоль.

Вариант 8

Создать программу с использованием цикла **while** и переменной **‘y’**, значение которой, изначально, равно 2. Данный цикл выводит значения переменной **‘y’** в третьей степени до тех пор, пока значение переменной **‘y’** не станет равным 10. При каждой итерации значение переменной **‘y’** увеличивается на 2.

Вариант 9

Создать программу с использованием цикла **while** и оператора **continue**. Цикл **while** работает с переменной **‘q’**, значение которой изначально равно 32. Данный цикл выводит значение переменной **‘q’** во второй степени. При каждой итерации значение переменной **‘q’** увеличивается на четыре. Когда значение переменной становится равным 1024, оператор **continue** приостанавливает работу цикла **while**.

Контрольные вопросы

1. Какую роль выполняет цикл for в языке Python?
2. Что такое оператор выхода из цикла?
3. Как можно передавать аргументы оператору?
4. Какую роль выполняет функция reverse(x) в языке Python?
5. Как можно использовать оператор not в языке Python?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ ИТЕРАТИВНЫЕ КОНСТРУКЦИИ. ЦИКЛ С УСЛОВИЕМ. НАПИСАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ЦИКЛА С УСЛОВИЕМ.

Цель работы: Изучение итеративных конструкций и цикла с условием на языке Python.

Необходимые материалы и оборудование: ПК, среда программирования PyCharm

Пояснения к работе:

Цикл for в языке Python

Следующий цикл, который мы разберем - цикл **for**. Данный цикл производит перебор н екого набора значений и помещает каждое значение в переменную, а затем в цикле можно производить действия (математические, логические и т.п.) с данной переменной. Посмотр им на форму конструкции цикла **for** (рис. 1):

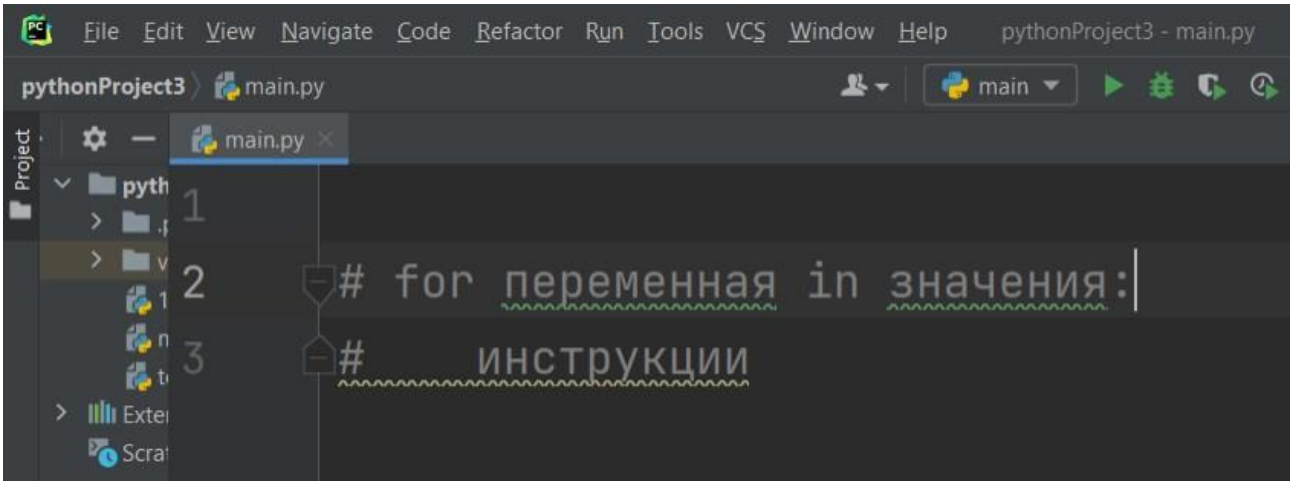


Рисунок 1 - Конструкция цикла for

При выполнении цикла на языке Python происходит последовательное получение всех з начений из набора и передача их переменной. Приведем один из вариантов применения ци кла**for** (рис. 2):

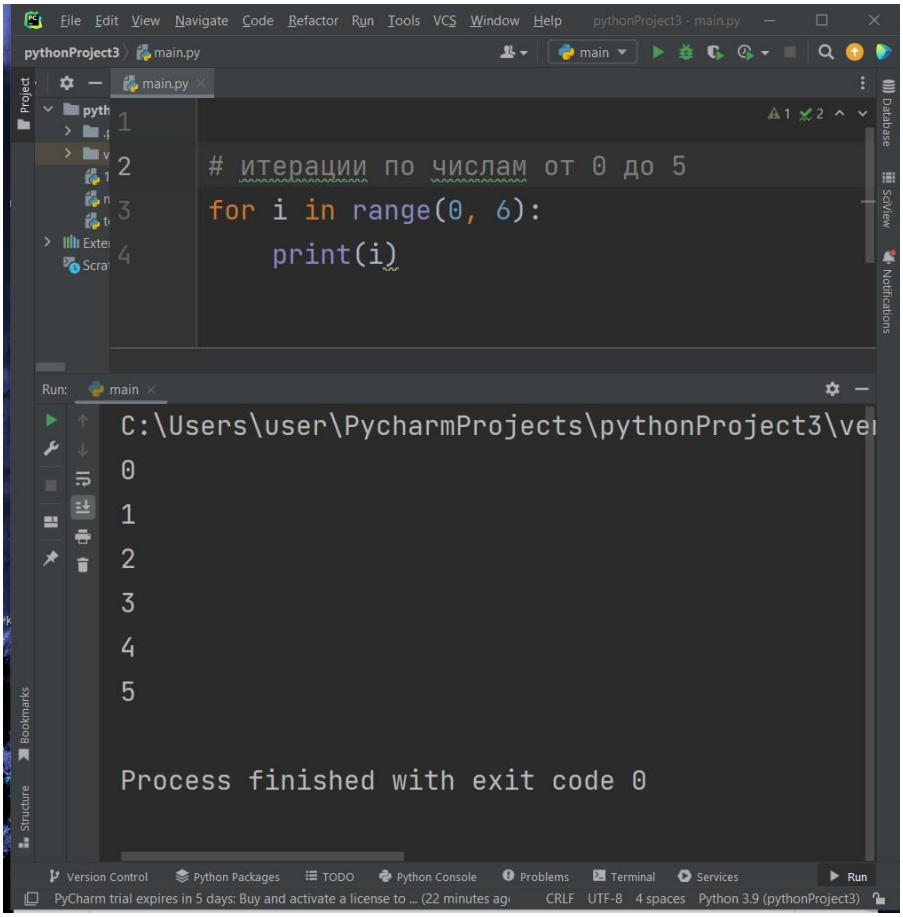


Рисунок 2 - Работа программы с циклом for при передаче значений

Произведем **анализ данной программы**: в строке 2 вводится однострочный комментарий; в строках 3 - 4 реализован цикл **'for'**, который перебирает значения мнимой переменной **i** вопределенном диапазоне. Здесь стоит отметить, что в конструкции **range (0, 6)** не выводится значение **6**, поэтому, если нужно реализован программу с выводом от как ого-то числа почисло **6** включительно, необходимо написать **range (0, 7)**.

Данному циклу можно передавать не только цифры, но и слова. В таком случае, цикл **for** будет перебирать не значения, а символы в слово, которое будет передано циклу (рис. 3):

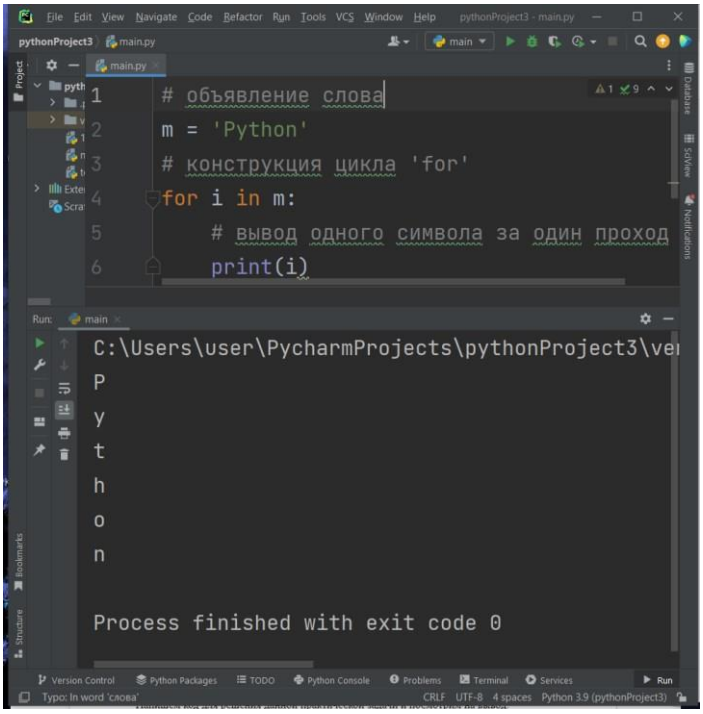


Рисунок 3 - Работа программы с циклом for при передаче буквенных символов

Соответственно, если бы в слове, которое записано в переменную **‘m’**, было бы больше символов, итераций цикла было бы больше.

Использование оператора if в Python

Оператор **‘if’** записывается с условным выражением, а далее могут следовать одна или несколько частей **‘elif’** и необязательная часть **‘else’**. Общая форма записи оператора **‘ if’**выглядит так (рис. 4):

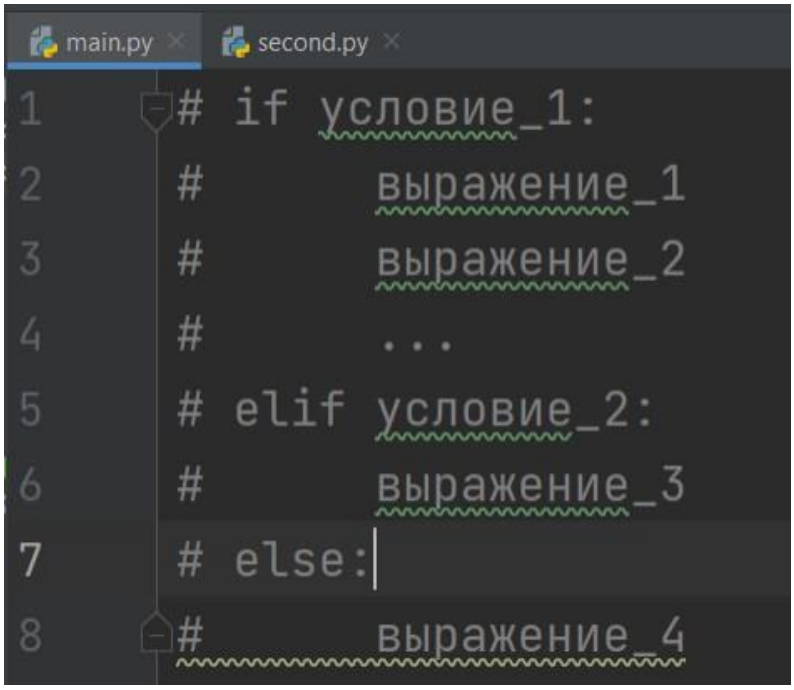


Рисунок 4 - Синтаксис использования оператора ‘if’

Рассмотрим варианты использования оператора **‘if’** (рис. 5, рис. 6, рис. 7, рис. 8):

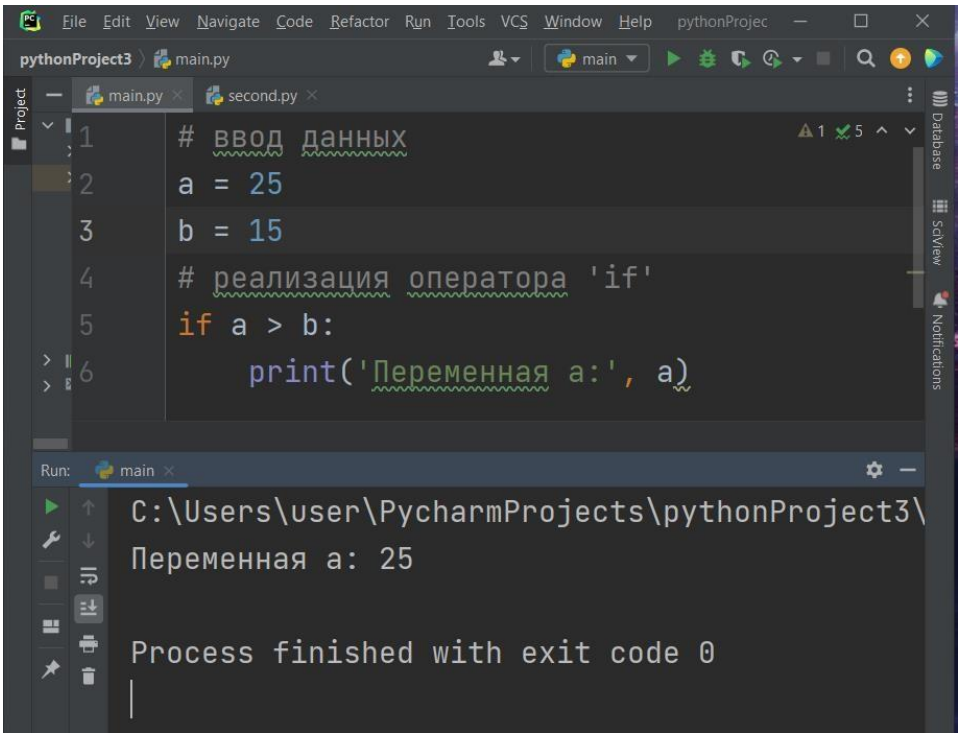


Рисунок 5 - Пример работы программы с использованием оператора ‘if’

В данном случае, если значение переменной **‘a’** больше, значение переменной **‘b’** (что я вляется истинной), выводится значение переменной **‘a’**.

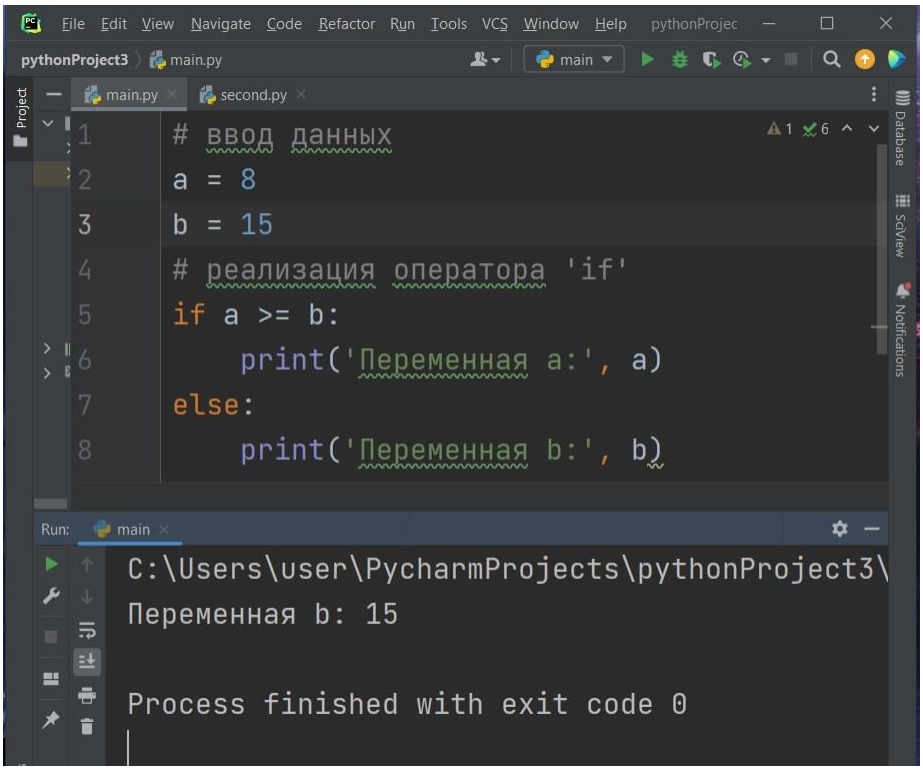


Рисунок 6 - Пример работы программы с использование оператора ‘if’ и ответвления ‘else’

Поясним работу программы последнего примера: в строках 2 и 3 вводятся переменные **‘a’** и **‘b’**. В строках 5 -

8 реализованы оператор **‘if’** и ответвление **‘else’**. Условие следующее: если значение переменной **‘a’** больше или равно значению переменной **‘b’**, то на консоль выводится переменная **‘a’**. В противном случае, на консоль выводится переменная **‘b’**. Данноеусловие не выполняется, поэтому на консоль выводится переменная **‘b’**.

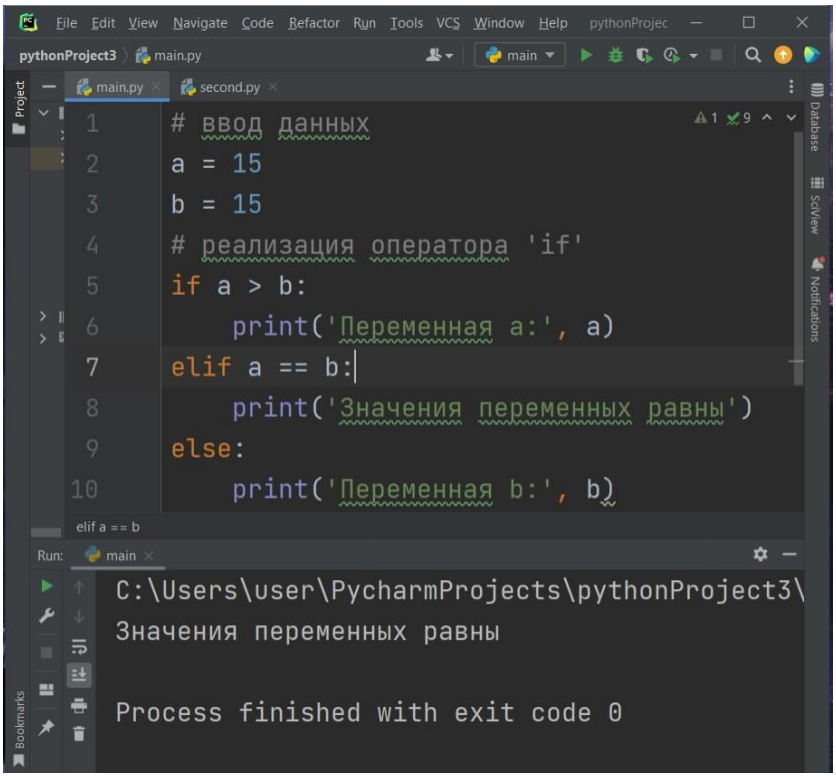


Рисунок 7 - Пример работы программы с использование оператора ‘if’, одного ответвления ‘elif’ и ‘else’

В данной ситуации, срабатывает условие в строке 7, так как значения переменных равны. Поэтому, на консоль выводится сообщение о том, что значения данных переменных равны.

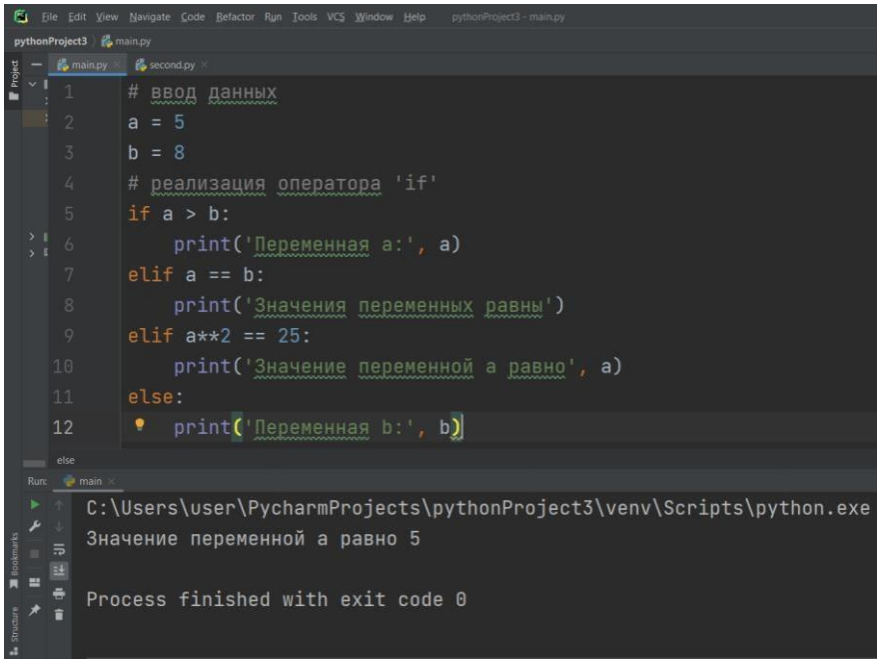


Рисунок 8 - Пример работы программы с использование оператора ‘if’, нескольких ответвлений ‘elif’ и ‘else’

Программа идет последовательно: идет с верхней строки до последней (при возможности) и последовательно проверяет выполнение всех условий. В данной ситуации, условие в строке5 условие не выполняется, так как значение переменной **‘a’** не превосходит значение переменной **‘b’**, поэтому команда **print()** в строке 6 пропускается. Условие в 7-ой строке также невыполняется, так как число **5** не равно числу **8**. Затем, условие в строке 9 выполняется, так как при возведении числа **5** во вторую степень, получается число **25**. Если условие в 9-ойстроке выполняется, то компилятор пропускает все последующие условия.

Арифметические операции в Python

Ниже представлена таблица с основными арифметическими операциями, которые можно использовать в язык Python (см. табл. 1).

Табл. 1. Арифметические операции в языке *Python*

Арифметические операции

Операция	Обозначение	Пример
Сложение	+	3 + 4 = 7
Вычитание	-	7 - 2 = 5
Умножение	*	2 * 2 = 4
Деление	/	8 / 2 = 4
Целочисленное деление	//	9 // 2 = 4
Остаток от деления	%	9 % 2 = 1
Возведение в степень	**	2 ** 3 = 8

Оператор “+”. Данный оператор объединяет строки, которая состоит из операндов. Рассмотрим пример работы данного оператора (рис. 9):

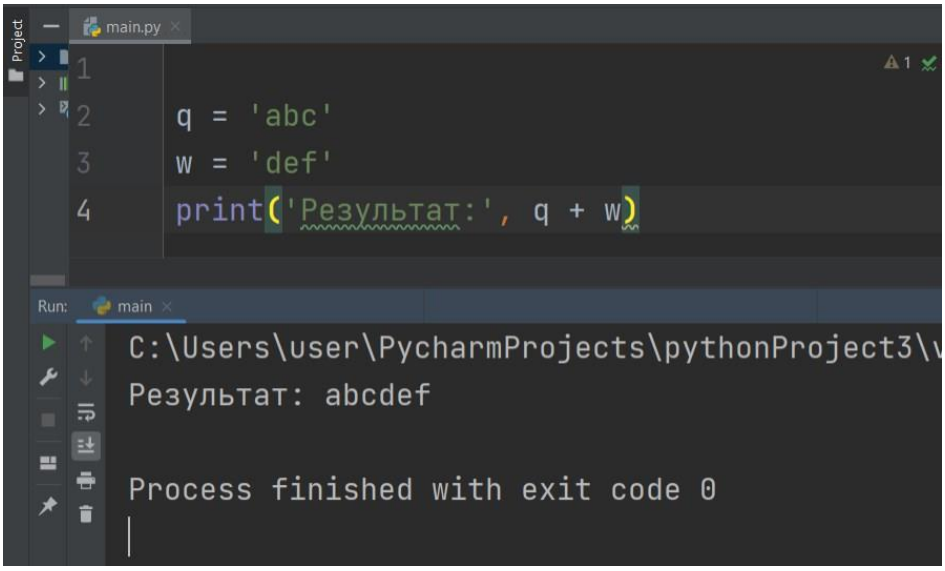


Рисунок 9 - Применение оператора строк “+” с двумя строками

Посмотрим на реализацию данного оператора с тремя строками (рис. 10):

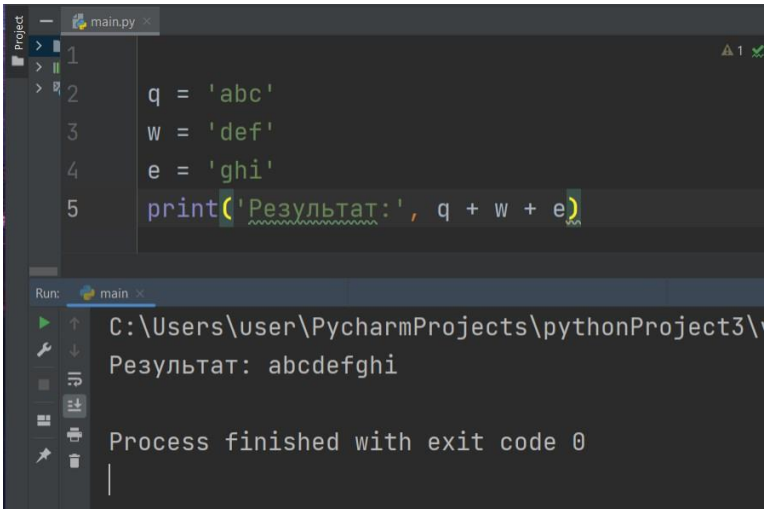


Рисунок 10 - Применение оператора строк “+” с тремя строками

Работая с данным оператором строк, можно не ограничиваться двумя или тремя строка ми (рис. 11):

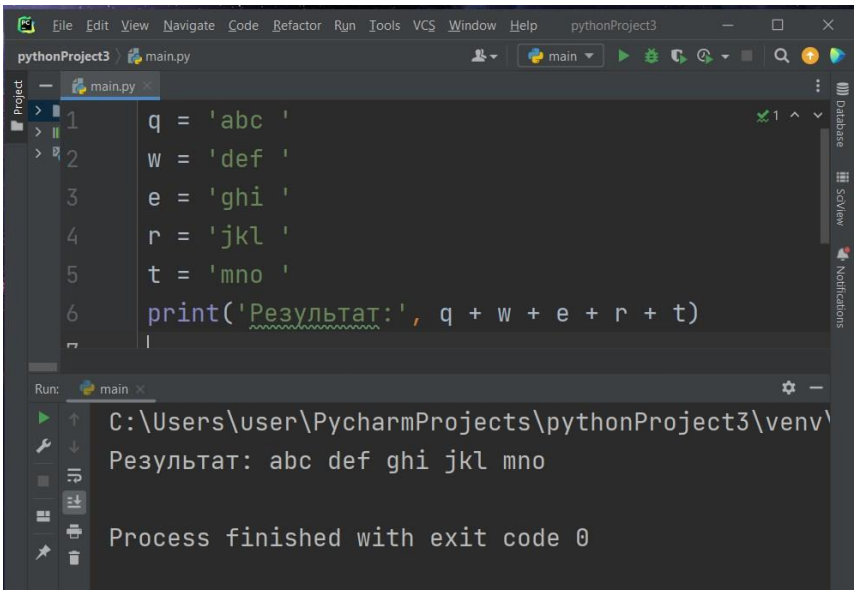


Рисунок 11 - Применение оператора строк “+” с большим количеством строк

Оператор “*”. Данный оператор создает несколько копий строк. Посмотрим на син таксис данного оператора (рис. 12):

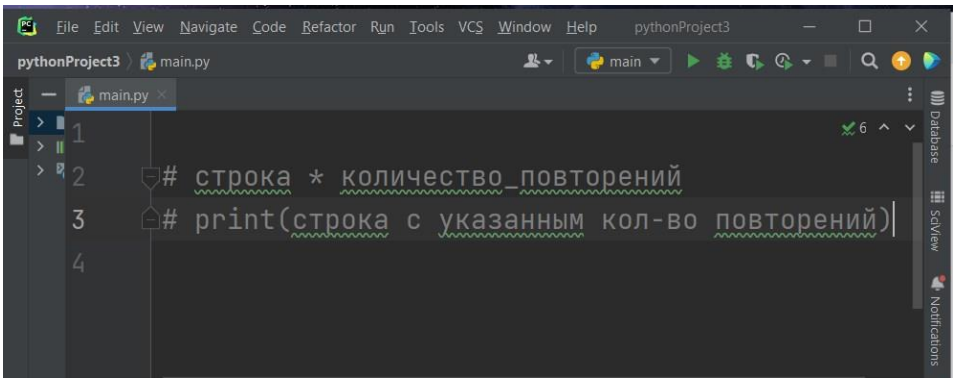


Рисунок 12 - Синтаксис использования оператора “*”

Применим данный оператор (рис. 13):

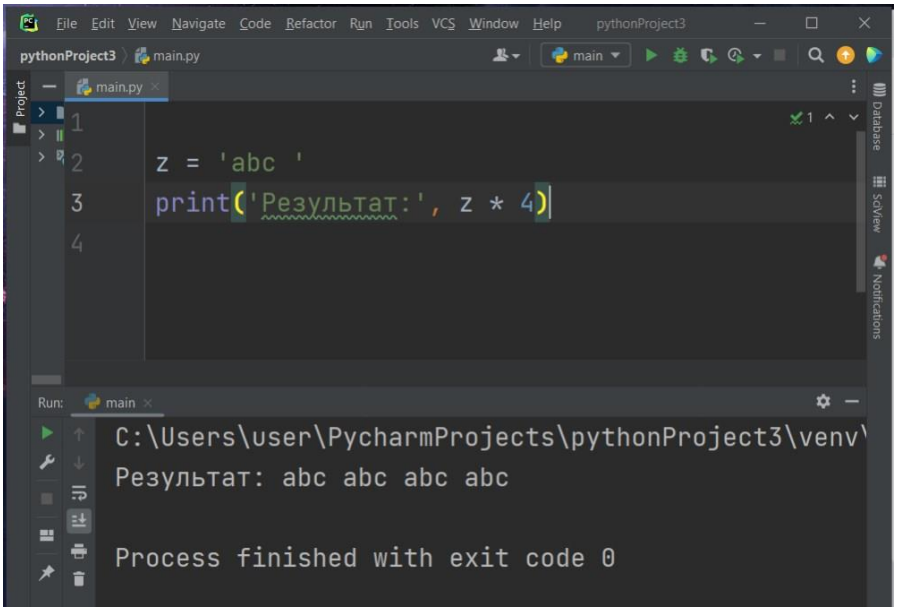


Рисунок 13 - Применение оператора строк “*” при работе с одной строкой

Применим данный оператор с двумя строками (рис. 14):

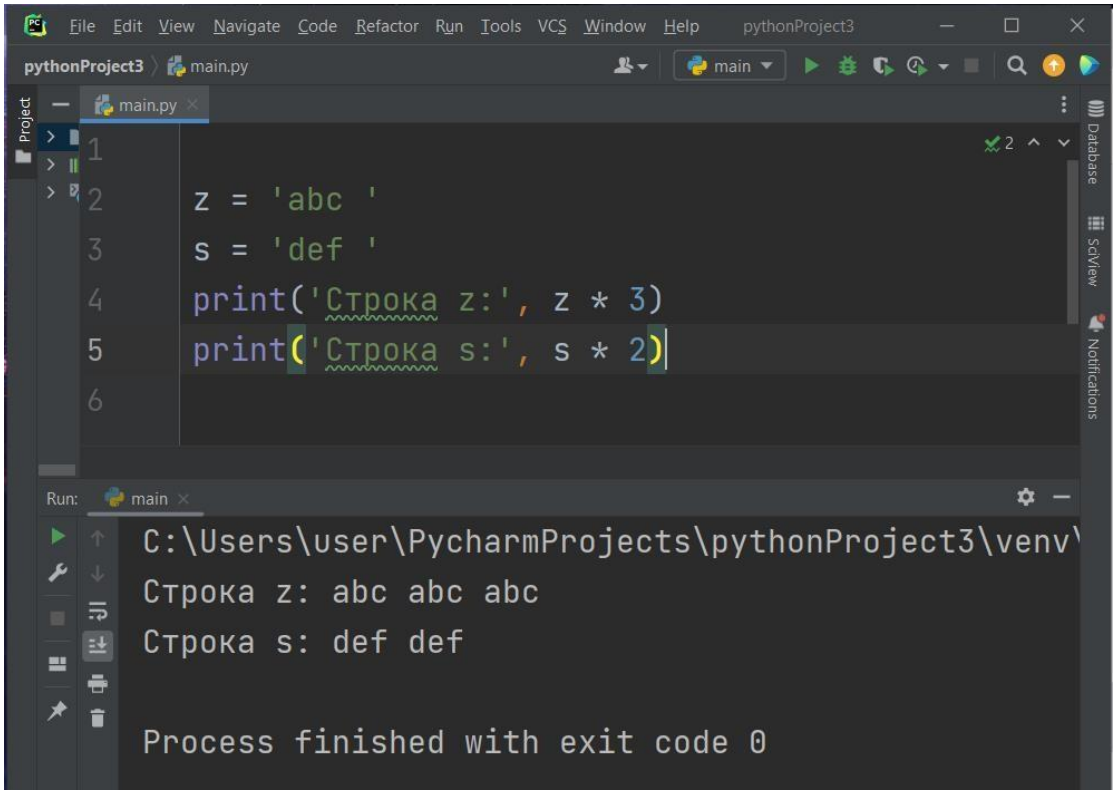


Рисунок 14 - Применение оператора строк “*” при работе с двумя строками

Функция **type()** возвращает тип объекта, который передается. Посмотрим на пример использования данной функции (рис. 15):

Рисунок 15 - Синтаксис функции *type()*

В данном случае в переменную **z** записывается целое число **3**, в переменную **y** - число с плавающей точкой **3.2**, а в переменную **x** - список чисел **[1, 2, 3]**. Соответственно, сначала выводится тип объекта **z** (**int**), затем тип объекта **y** (**float**) и тип объекта **x** (**type**).

Функция **issubclass()** проверяет, является ли класс подклассом одного или нескольких других классов, посмотрим на пример ее использования (рис. 16):

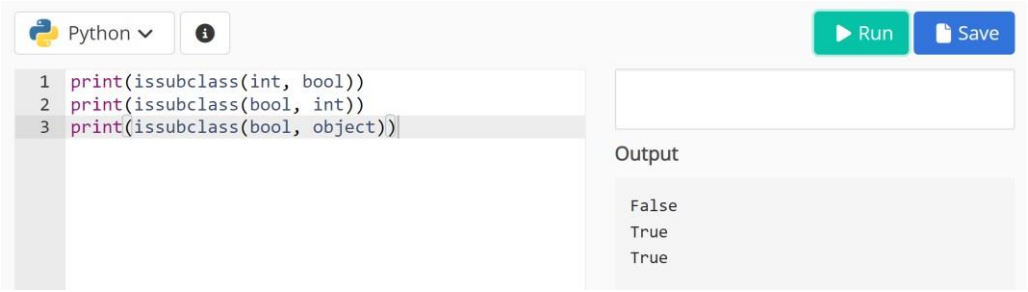


Рисунок 16 - Синтаксис функции *issubclass()*

Функция **isinstance()** проверяет, является ли объект экземпляром одного или несколько классов (рис. 17):

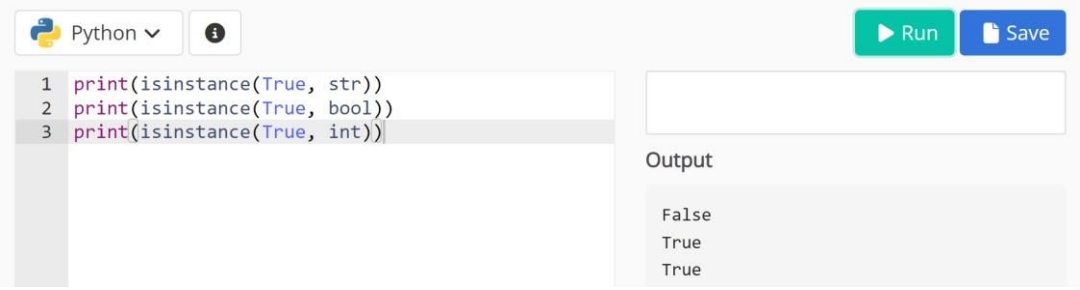


Рисунок 17 - Синтаксис функции *isinstance()*

Вложенные циклы (повторение)

Кроме того, одни циклы внутри себя могут содержать другие циклы. Рассмотрим п ример вывода простой таблицы умножения (рис. 18, рис. 19):

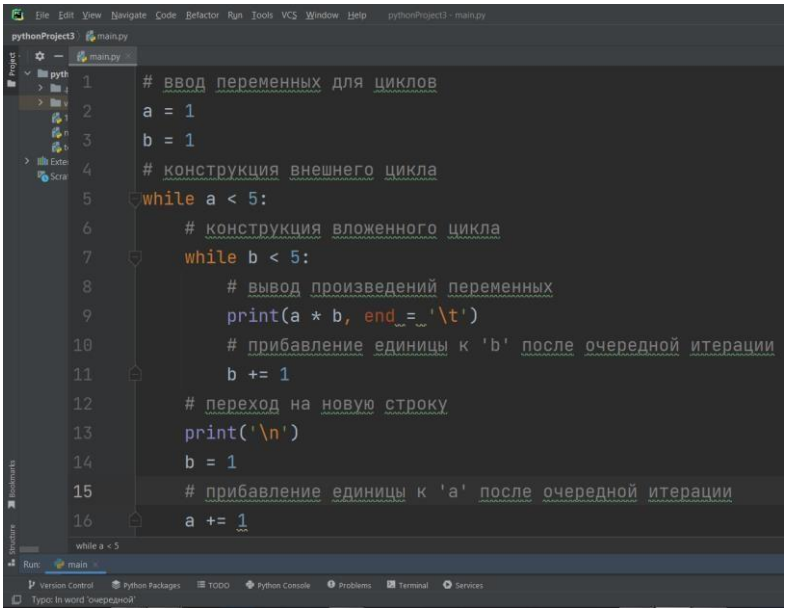


Рисунок 18 - Листинг кода при работе с вложенными циклами

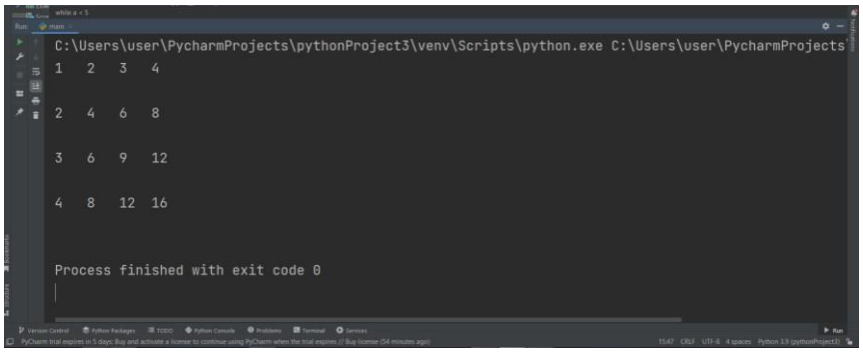


Рисунок 19 - Вывод консоли при работе с вложенными циклами

Произведем анализ данной программы: внешний цикл с конструкцией **‘while a < 5’** срабатывает столько раз, пока переменная **‘a’** не станет равна 10. Внутри данного цикла такжесрабатывает внутренний цикл **‘while b < 5’**. Внутренний цикл срабатывает до тех пор, пока значение переменной **‘b’** не станет равн о 5. Здесь стоит отметить, что все итерациивнутреннего цикла срабатывают в рамках одно й лишь итерации внешнего цикла. В каждой итерации внутреннего цикла на консоль (рис. 10) выводится произведение значений **‘a’** и**‘b’**. Когда внутренний цикл заканчивает свою р аботу, значение переменной **‘b’** сбрасывается в единицу, а значение переменной **‘a’** увели чивается на единицу. После этого происходитпереход к следующей итерации внешнего ци кла.

Приведем следующий пример вложенного цикла **for**, когда на консоли появляются всевозможные комбинации двух символов (рис. 20):

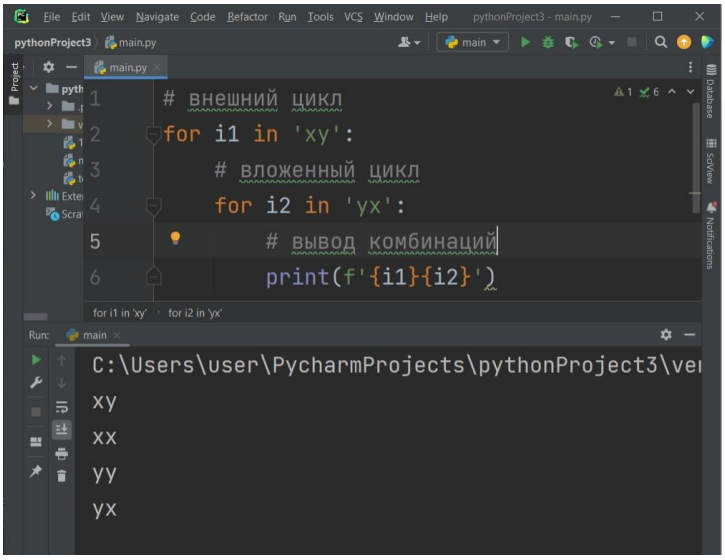


Рисунок 20 - Работа программы вложенного цикла for

Задание

Вариант 1

Создать программу, в которой вводятся списки **‘m’ = [1, 3, 5, 7, 9]** и **‘x’ = [11, 23, 35]**. Необходимо рассчитать количество элементов в данных списках и вывести данные значения на консоль.

Вариант 2

Создать программу, в которой происходит перебор элементов списка с именем **‘words’** при помощи цикла **while**.

Вариант 3

Создать программу, в которой вводится список **‘x’ = [1, 2, 3, 4, 5, 6]**. Необходимо найти минимальный элемент в данном списке и вывести его на консоль.

Вариант 4

Создать программу, в которой используется строка **‘z’ = ‘abc’**. Необходимо применить оператор **“*”** и вывести данную строку 4 раза подряд в строчку.

Вариант 5

Создать программу, в которой реализован оператор выхода из цикла **break.** Изначально вводится переменная **‘s’** со значением **0**. При каждой итерации цикла значение переменной **‘s’**увеличивается на единицу. Когда значение переменной **‘s’** становится равным **5**, оператор **break** завершает работу цикла (на консоли должны быть выведены значения с **1** по **4** включительно).

Вариант 6

Создать программу, в которой используются переменные **‘a’** и **‘b’**. Реализовать в данной программе оператор **‘if’** со следующим условием: если значение переменной **‘a’** больше, чем значение переменной **‘b’**, то выводится значение переменной **‘b’**; если значения переменных равны, то выводится сообщение **‘Значения переменных равны’**; в противном случае, выводится значение переменной **‘b’**. Реализовать все три итерации (в каждой итерации выполняется свое условие). Результаты продемонстрировать на консоли.

Враинат 7

Создать программу, в которой происходит перебор элементов списка с именем **‘words’ = [‘one’, ‘two’, ‘three’, ‘four’]** при помощи цикла **while**.

Вариант 8

Создать программу с использованием цикла **while** и переменной **‘y’**, значение которой, изначально, равно 2. Данный цикл выводит значения переменной **‘y’** в третьей степени до тех пор, пока значение переменной **‘y’** не станет равным 10. При каждой итерации значение переменной **‘y’** увеличивается на 2.

Вариант 9

Создать программу, в которой реализован оператор выхода из цикла **break.** Изначально вводится переменная **‘s’** со значением **0**. При каждой итерации цикла значение переменной **‘s’** увеличивается на единицу. Когда значение переменной **‘s’** становится равным **5**, оператор **break** завершает работу цикла (на консоли должны быть выведены значения с **1** по **4** включительно).

Вариант 10

Создать программу, в которой вводится список **‘x’ = [104, 3452, 4543, 45, 5678, 36]**. Необходимо найти минимальный и максимальный элемент в данном списке и вывести его на консоль.

Контрольные вопросы

1. Какую роль выполняет цикл while в языке Python?
2. Что такое оператор прерывания цикла?
3. Как можно передавать аргументы циклу?
4. Какую роль выполняет функция max(x) в языке Python?
5. Как можно использовать оператор and в языке Python?

Практические занятия

Программная реализация принципов наследования

Основные принципы ООП: наследование в программировании

О принципе наследования в ООП простыми словами. Объясняем механизм наследования ООП и преимущества метода на примере Java-кода.

Принцип программирования наследование является одним из ключевых понятий в ООП. Он позволяет создавать иерархии классов, где один класс (подкласс) наследует свойства и методы другого класса (суперкласса). Это позволяет сокращать дублирование кода, упрощать структуру программы и создавать более логичные иерархии объектов.

Преимущества принципа наследования

Принцип наследования является одним из фундаментальных понятий объектно-ориентированного программирования и предоставляет несколько преимуществ, которые делают его важным и полезным инструментом при проектировании программных систем:

1. Повторное использование кода. Наследование позволяет создавать иерархии классов, где общая функциональность реализуется в родительском классе, и все подклассы автоматически наследуют этот код. Это способствует повторному использованию кода, что уменьшает дублирование и облегчает его поддержку.
2. Расширяемость. Принцип наследования позволяет создавать новые классы, расширяющие функциональность существующих классов. Подклассы могут добавлять новые свойства и методы, а также переопределять поведение унаследованных методов. Это делает код более гибким и позволяет легко вносить изменения.
3. Упрощение кода. Использование наследования позволяет разбивать большие и сложные классы на более мелкие и управляемые части. Каждый подкласс специализируется на определенном аспекте функциональности, что упрощает понимание и поддержку кода.
4. Полиморфизм. Наследование поддерживает концепцию полиморфизма, которая позволяет обращаться к объектам подклассов через ссылки на родительские классы. Это облегчает обработку групп объектов с различными типами, что упрощает написание общего и универсального кода.
5. Абстракция. Наследование позволяет выделить общие характеристики объектов и создать абстрактные классы, которые определяют интерфейс для группы связанных классов. Абстрактные классы предоставляют общую сущность без необходимости определения всех деталей реализации.
6. Структурирование кода. Наследование помогает упорядочить классы в логические иерархии, что улучшает структуру программы. Каждый класс наследует функциональность от одного или нескольких родительских классов, что улучшает организацию кода и делает его более понятным и легко поддерживаемым.

В целом, принцип наследования позволяет создавать более гибкие, модульные и расширяемые программы, что упрощает разработку и сопровождение сложных проектов. Он способствует повторному использованию кода и помогает соблюдать принципы DRY (Don’t Repeat Yourself) и SOLID, что в свою очередь способствует созданию качественного и эффективного кода.

Пример наследования в ООП

Рассмотрим пример кода на Java:

```
                // Родительский класс (суперкласс)
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void makeSound() {
        System.out.println("Some generic animal sound");
    }
}

// Подкласс, наследующий свойства и методы от Animal
class Dog extends Animal {
    String breed;

    Dog(String name, int age, String breed) {
        super(name, age); // Вызов конструктора суперкласса
        this.breed = breed;
    }

    @Override
    void makeSound() { // Переопределение метода makeSound()
в подклассе Dog
        System.out.println("Woof!");
    }

    void fetch() {
        System.out.println("Fetching the ball!");
    }
}

// Подкласс, наследующий свойства и методы от Animal
class Cat extends Animal {
    String furColor;

    Cat(String name, int age, String furColor) {
        super(name, age); // Вызов конструктора суперкласса
        this.furColor = furColor;
    }

    @Override
```

```
        void makeSound() { // Переопределение метода makeSound()
в подклассе Cat
            System.out.println("Meow!");
        }

        void purr() {
            System.out.println("Purring...");
        }
    }

    public class Main {
        public static void main(String[] args) {
            Dog dog = new Dog("Buddy", 3, "Golden Retriever");
            Cat cat = new Cat("Whiskers", 2, "Gray");

            System.out.println("Dog: " + dog.name + ", Age: " +
dog.age + ", Breed: " + dog.breed);
            dog.makeSound();
            dog.fetch();

            System.out.println("\nCat: " + cat.name + ", Age: " +
cat.age + ", Fur Color: " + cat.furColor);
            cat.makeSound();
            cat.purr();
        }
    }
```

В этом примере у нас есть родительский класс `Animal`, который содержит общие свойства и методы для всех животных. Затем есть два подкласса `Dog` и `Cat`, которые наследуют свойства и методы от класса `Animal`. Каждый подкласс также имеет свои собственные уникальные свойства и методы. Обратите внимание на использование ключевого слова `extends` при объявлении подклассов.

Принцип наследования позволяет нам использовать общие характеристики и функциональность из родительского класса и при этом иметь возможность расширять или переопределять их в подклассах для создания более специфичных типов объектов.

Программная реализация принципов полиморфизма

Основные принципы ООП: полиморфизм в программировании

Полиморфизм — один из основных принципов ООП: что это такое, в каких случаях используется, а также наглядный пример полиморфизма в ООП.

Одним из ключевых принципов ООП является полиморфизм — концепция, позволяющая создавать более гибкие, расширяемые и понимаемые программы. В этой статье мы рассмотрим суть полиморфизма, его типы и приведем примеры кода для лучшего понимания данного принципа ООП.

Для чего нужен полиморфизм в программировании?

Полиморфизм в контексте ООП означает, что разные объекты могут реагировать на один и тот же запрос, проявляя разное поведение в зависимости от своего типа. Это

позволяет сократить дублирование кода, улучшить читаемость и облегчить расширение программы.

Преимущества принципа полиморфизма

1. Гибкость и расширяемость. Полиморфизм позволяет добавлять новые типы объектов и операций без изменения существующего кода. Новые классы, реализующие общий интерфейс, могут быть легко интегрированы в существующую систему.
2. Упрощение кода. Полиморфизм способствует уменьшению дублирования кода. Общий интерфейс или абстрактный базовый класс позволяют описать общее поведение, и каждый конкретный класс реализует только свою специфичную логику.
3. Читаемость кода. Полиморфизм делает код более интуитивно понимаемым, так как работа с различными объектами происходит через общий интерфейс. Это упрощает восприятие кода другими разработчиками и способствует поддержке программы.
4. Расширение функциональности. Добавление новых функций или операций для существующих классов становится проще. Достаточно реализовать необходимые методы в новых классах, которые наследуют общий интерфейс.
5. Повторное использование кода. Полиморфизм позволяет использовать одни и те же методы для разных типов данных. Это устраняет необходимость создания аналогичных функций для разных классов.
6. Улучшение тестирования. Тестирование становится более удобным, так как можно создать общие тестовые сценарии для всех классов, реализующих один интерфейс. Это способствует повышению качества и надежности программы.
7. Абстракция и инкапсуляция. Полиморфизм позволяет абстрагироваться от конкретных реализаций и сосредоточиться на общем поведении объектов. Также он способствует инкапсуляции, разделяя интерфейс от деталей реализации.
8. Облегчение командной разработки. Когда разработчики работают над разными частями программы, полиморфизм позволяет им взаимодействовать через общие интерфейсы без необходимости глубокого понимания внутренней реализации друг друга.

Виды полиморфизма в объектно-ориентированном программировании

Разные виды полиморфизма в объектно-ориентированном программировании обеспечивают гибкость и расширяемость кода. Они позволяют обращаться с разными типами данных единообразно, что делает программы более понятными и удобными для разработки и обслуживания программного кода.

1. Полиморфизм подтипов (наследования)

Этот вид полиморфизма основан на наследовании и позволяет объектам дочерних классов использоваться как объекты родительского класса. Это делает код более гибким и облегчает добавление новых типов.

```
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

class Rectangle(Shape):
```

```
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

shapes = [Circle(5), Rectangle(4, 6)]

for shape in shapes:
    print("Area:", shape.area())
```

2. Параметрический полиморфизм (обобщённое программирование)

Параметрический полиморфизм позволяет создавать обобщенные функции и классы, которые могут работать с разными типами данных без знания их конкретной природы.

```
                    def print_items(items):
    for item in items:
        print(item)

numbers = [1, 2, 3]
names = ["Alice", "Bob", "Charlie"]

print_items(numbers)
print_items(names)
```

3. Полиморфизм в интерфейсах

Интерфейсный полиморфизм позволяет объектам разных классов реализовывать общий интерфейс и предоставлять схожее поведение без явного наследования.

```
                    from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        return "Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow!"

def animal_sounds(animal):
    return animal.make_sound()

dog = Dog()
cat = Cat()

print(animal_sounds(dog))  # Вывод: "Woof!"
print(animal_sounds(cat))  # Вывод: "Meow!"
```

Полиморфизм — это суть объектно-ориентированного программирования, позволяющая создавать гибкие и расширяемые программы. Благодаря различным видам

полиморфизма, разработчики могут писать более чистый, читаемый и эффективный код. Овладение этим принципом существенно обогатит навыки любого программиста и сделает его программы более элегантными и функциональными.

Разработка конструкторов и деструкторов

Конструкторы и деструкторы

Как упоминалось в предыдущем разделе, конструкторы и деструкторы представляют собой специальные методы класса, которые вызываются автоматически соответственно при создании и уничтожении объектов класса.

Конструкторы. По правилам C++ конструктор имеет то же имя, что и класс, может не иметь аргументов, никогда не возвращает значения и определяет операции, которые необходимо выполнить при создании объекта. Обычно это выделение памяти под динамические поля объекта и инициализация полей, но могут выполняться и другие операции.

1. Использование конструктора для инициализации полей объекта. Рассмотрим использование конструктора для инициализации полей объектов класса, описанного в примере 3.2, изменив доступность поля strl. Конструктор инициализирует поля объектов класса sstr и осуществляет действия, связанные с проверкой длины вводимой строки и коррекцией строки в случае несоответствия.

```
#tinclude <string.h>

#include <iostream.h>

#include <conio.h>

class sstr

{private: char strl[40];

public: int x,y;

void print(void) {

cout<< "содержимое общих полей: "<< endl;

cout<<"x= "<<x<<"y= "<<y<<endl;

cout<<"содержимое скрытых полей: "<< endl;

cout<<" strl = "<<strl<<endl;}

sstr(int vx,int vy,char *vs) //конструктор класса sstr

{ int len=strlen(vs); // проверка правильности задания длины

if(len>=40) {strncpy(strl,vs,40);strl[40]='\0';} else strcpy(strl,vs); x=vx;y=vy; }
```

```
};

void main() { clrscr();

//создание объекта класса sstr при наличии конструктора

sstr aa(200,150, "пример использования конструктора");

aa.print(); getch();

}
```

Как и любая другая компонентная функция, конструктор может быть переопределяемым, указывая соответствующие действия для различных списков параметров. Поэтому класс может иметь несколько конструкторов, позволяющих использовать разные способы инициализации объектов.

2. Использование переопределяемых конструкторов.

```
#include <string.h>

#include <iostream.h>

#include <conio.h>

class sstr

{private: char strl[40];

public: int x,y;

void print(void)

{cout<<" содержание полей : " <<endl;

cout<<" x= "<<x<<"y= "<<y<<endl;

cout<< " strl = "<<strl <<endl;}

sstr(void) // конструктор, определяющий поля по умолчанию

{strcpy(strl, "конструктор по умолчанию"); x=0;y=0; };

sstr(char *vs) // конструктор, получающий значение поля strl

{int len=strlen(vs);

if(len>=40) {strncpy(strl,vs,40);strl[40]='\0';}else strcpy(strl,vs); x=0; y=0; }

// значения x и y задаются по умолчанию

sstr(char *vs,int vx) // конструктор, получающий значения strl и x
```

```
{int len=strlen(vs);

if(len>=40) {strncpy(strl,vs,40);strl[40]='\0';} else strcpy(strl,vs); x=vx; y=0; }

// значение у задается по умолчанию

sstr(char *vs,int vx,int vy) // конструктор, получающий все значения

{int len=strlen(vs);

if(len>=40) {strncpy(strl,vs,40);strl[40]='\0';} else strcpy(strl,vs); x=vx;y=vy; }

};

void main()

{ clrscr();

sstr a0, a1(" пример конструктора 1 с x и y по умолчанию ");

sstr a2(" пример конструктора 2 cy по умолчанию ",100);

sstr a3(" пример конструктора 3 со строкой по умолчанию ",200,150};

a0.print(); a1.print(); a2.print(); a3.print(); getch();}
```

Кроме того, иногда в конструкторах целесообразно использовать параметры, значение которых задается по умолчанию. В этом случае вместо нескольких переопределяемых конструкторов обычно можно описать один. Вернемся к предыдущему примеру и заменим все переопределяемые конструкторы одним.

3. Использование конструктора с аргументами по умолчанию.

```
#include <string.h>

#include <iostream.h>

#include <conio.h>

class sstr

{private: char strl[40]; public: int x,y;

void print(void)

{ cout<< " содержимое полей : ""<< endl;

cout<<" x= "<<x<<"y= "<<y<<endl;

cout<< " strl = "<<strl <<endl;}

sstr(char *vs,int vx,int vy);
```

```
/* прототип конструктора с параметрами, заданными по умолчанию */

};

sstr:: sstr(char vs="строка по умолчанию ",int vx=80,int vy=90)

/* тело конструктора с параметрами, заданными по умолчанию */

{int len=strlen(vs);

if(len>=40) {strncpy(strl,vs,40); strl[40]='\0';} else strcpy(strl,vs); x=vx;y=vy;

}

void main() { clrscr();

sstr a0, a1(" x и y по умолчанию");

sstr a2(" y по умолчанию ", 100);

sstr a3("определяет все поля",200,150);

a0.print(); // выводит: x=80 y=90 strl= строка по умолчанию

al.print(); // выводит: x=80 y=90 strl= x и y по умолчанию

a2.print(); // выводит: x=100 y=90 strl= y по умолчанию

a3.print(); // выводит: x=200 y= 150 strl = определяет все поля

getch(); }
```

В конструкторах может использоваться специальная конструкция -список инициализации. Список инициализации отделяется от заголовка конструктора двоеточием и состоит из записей вида

<имя>(<список выражений>),

где в качестве имени могут фигурировать:

• имя поля данного класса,

• имя объекта другого класса, включенного в данный класс,

• имя базового класса.

Список выражений определяет значения, используемые для инициализации соответствующих объектов.

Конструктор со списком инициализации применяют для задания начальных значений объектам, в которых используются фиксированные (константные) и ссылочные поля, а также поля, являющиеся объектами других, ранее определенных классов. В последнем случае, если в определении класса не предусмотрен метод инициализации полей класса,

список инициализации -это единственный способ вызвать конструктор объектного поля, так как явный вызов конструктора в C++ не возможен.

Иногда возникает необходимость создать объект, не инициализируя его поля. Такой объект называется неинициализированным и под него только резервируется память. Для создания подобного объекта следует предусмотреть неинициализирующий конструктор. У такого конструктора нет списка параметров и, обычно, отсутствует тело («пустой» конструктор). В этом случае при объявлении объекта с использованием такого конструктора значение полей объекта не определено.

Ниже приводится пример описания конструктора со списком инициализации и неинициализирующего конструктора для объектных и фиксированных полей.

4. Использование конструктора со списком инициализации и неинициализирующего конструктора.

```
#include <iostream.h>

#include <conio.h>

class integ

{ int num;

public:

void print(void) { cout<<" num= "<<num<<endl;}

integ(int v):num(v){} /* конструктор инициализирует поле num типа private с помощью списка инициализации */

integ(void){} // неинициализирующий конструктор

};

class obinteg

{ integ onum; // поле объектного типа

public: const int x,y,c; // поля фиксированного типа

void print(void)

{onum.print(); cout<<"x="<<x<<"y= "<<y<<"color ="<<c<<endl;}

obinteg(int va,int vx,int vy,int vc):onum(va),x(vx),y(vy),c(vc){}

/*конструктор инициализирует поле объектного типа и поля фиксированного типа списком инициализации*/

obinteg(void){} // неинициализирующий конструктор

void main()

{ integ aa(10); // инициализируемый объект
```

```
integ s1; // неинициализируемый объект

obinteg bb(20,40,100,13); // инициализируемый объект

obinteg s2; // неинициализируемый объект

cout<<"данные неинициализируемого объекта integ s1"<< endl;

s1.print(); // выводит случайные числа

cout<<"данные неинициализируемого объекта obinteg s2"<<endl;

s2.print(); // выводит случайные числа

cout << "данные объекта класса integ: "<<endl;

aa.print(); // выводит заданные значения

cout << "данные объекта с объектным полем integ: "<<endl;

bb.print(); // выводит заданные значения

getch();}
```

Не следует путать неинициализирующий конструктор с конструктором с параметрами, заданными по умолчанию, что возможно, если последний не имеет списка параметров (например, см. пример 3.7), так как в теле конструктора с параметрами по умолчанию задаются некоторые фиксированные значения полей.

Если в определении класса уже описан конструктор по умолчанию (с параметрами или без таковых), то его можно использовать вместо неинициализирующего. При этом необходимо помнить, что поля объекта, образованного с помощью такого конструктора, будут инициализированы фиксированными значениями. Определять же в одном и том же классе конструктор по умолчанию и неинициализирующий конструктор одновременно компилятор не позволяет, так как у них совпадает форма вызова.

При использовании в программах массивов объектов некоторого класса наличие неинициализирующего конструктора желательно, так как при создании массива объектов для каждого из его элементов автоматически вызывается конструктор класса, объектом которого является элемент.

Инициализацию полей массива объектов можно выполнять несколькими способами.

Первый способ - предусмотреть конструктор без параметров, который инициализирует поля объекта некоторыми значениями, формируемыми с помощью датчика случайных чисел. Автоматический вызов такого конструктора создаст массив с различными значениями. Однако такой способ в большинстве случаев не применим, так как значения полей объектов массива обычно должны задаваться определенным образом.

Второй способ - использовать конструктор класса, который инициализирует поля данными, вводимыми с клавиатуры. Данный способ требует ввода в момент определения массива объектов используемого класса, а это тоже не всегда удобно.

5. Использование предполагаемого копирующего конструктора.

```
#include <string.h>

#include <iostream.h>
```

```
#include <conio.h>

class child

{private: char *name; int age;

public: void print(void)

{cout<<" имя : "<<name; cout<<" возраст: "<<age<< endl;}

child(char *Name,int Age):name(Name),age(Age){}

};

void main()

{ clrscr();

child aa("Мария",6), bb("Евгений",5);

cout << " результаты работы программы: "<<endl;

aa.print(); // выводит: имя Мария возраст 6

bb.print(); // выводит: имя Евгений возраст 5

child dd=aa; // предполагается использование копирующего конструктора

dd.print(); // выводит: имя Мария возраст 6

getch(); }
```

По умолчанию предполагается конструктор:

```
child(const child & obj):name(obj.name),age(obj.age){}
```

В результате поля объекта aa без изменения копируются в поля dd.

Если же необходимо при копировании полей изменять их содержимое, следует явно определить в описании класса копирующий конструктор, предусмотрев действия по изменению значений всех или некоторых полей.

6. Явное определение копирующего конструктора.

```
#include <string.h>

#include <iostream.h>

#include <conio.h>

class intpole

{ int cif;
```

```
public: void print(void)

{cout<<"cif = "<<cif<<" "<< endl;}

intpole(int va) { cif=va;}

intpole(const lntpole& ob1) // копирующий конструктор

{ cif=ob1.cif*3;}

};

class obpole

{ int num; intpole intob; // объектное поле

public:

void print (void)

{intob.print(); cout<<" num= "<<num<<endl;}

obpole(int vn,int va):num(vn),intob(va){}

};

void main()

{ clrscr();

obpole aa(10,40);

cout << " результаты работы программы; "<<endl;

aa.print(); // выводит cif = 40 num = 10

obpole bb=aa; // активизируется копирующий конструктор

bb.print(); // выводится cif = 120 num = 10

getch(); }
```

В приведенном примере для класса obpole предполагается копирующий конструктор по умолчанию:

```
obpole(const obpole &obj):num(obj.num), intob(obj.intob){}
```

При инициализации объекта intob вызывается копирующий конструктор класса intpole, что приводит к инициализации соответствующего поля cif утроенным значением.

Кроме того, бывают случаи, когда поля объекта при копировании не изменяются, однако использование предполагаемого конструктора, автоматически создаваемого компилятором, может привести к непредсказуемым результатам.

Такая ситуация возникает, например, при вызове функции, получающей объект в качестве параметра, переданного «по значению». В этом случае, как известно, в памяти создается копия объекта, с которой должна работать функция. Для этого вызывается копирующий конструктор независимо от его наличия в описании класса. Автоматически генерируемый конструктор может не учесть особенностей объекта и его полей. Очень часто такие случаи происходят при работе с динамическими объектами и объектами, содержащими динамические поля (см. пример 3.30).

Деструкторы. При уничтожении объекта, так же, как при его конструировании, автоматически вызывается специальный метод - деструктор. Имя деструктора по аналогии с именем конструктора, совпадает с именем класса, но перед ним стоит символ «~» (префикс «тильда»). Деструктор определяет операции, которые необходимо выполнить при уничтожении объекта. Обычно он используется для освобождения памяти, выделенной под поля объекта данного класса конструктором. Деструктор не возвращает значения, не имеет параметров и не наследуется производными классами. Класс может иметь только один деструктор или не иметь ни одного. Так как деструктор - это функция, то он может быть виртуальным (см. пример 3.33). Однако отсутствие параметров не позволяет перегружать деструктор.

Деструктор вызывается неявно, автоматически, как только объект класса удаляется из памяти. Момент вызова определяется моделью памяти, выбранной для объекта (локальный, глобальный, внешний и т. д.). Если программа завершается с использованием функции exit, то вызываются деструкторы только глобальных объектов. При завершении программы, использующей объекты некоторого класса, функцией abort деструктор не вызывается. Однако, в отличие от конструктора, деструктор можно вызвать и явно.

Если в классе не объявлены конструктор и деструктор, то некоторые компиляторы автоматически производят их построение. При этом конструктор неявно используется для выделения памяти и размещения объекта после его описания, а деструктор - для корректного освобождения памяти после того, как имя объекта становится недействительным. В тех компиляторах, которые не производят автоматического построения конструкторов и деструкторов при их отсутствии, выдается диагностическое сообщение, требующее явного объявления этих компонентов класса.

Примечание. Если объект содержит указатель на некоторую динамически выделенную область памяти, то по умолчанию эта память не освобождается.

7. Определение в классе деструктора.

```
#include <iostream.h>

#include <string.h>

class din_string

{ char *str; // поле класса - указатель на строку

unsigned size;

public:

din_string(char *s, unsigned SIZE=100); // прототип конструктора

~din_string(); // прототип деструктора

void display() {cout <<str <<endl;}

};
```

```
din_string::din_string(char *s, unsigned SIZE) // конструктор

{str = new char [size = SIZE]; strcpy(str,s);}

din_string::~din_string() // описание деструктора вне класса

{delete [] str ;}

void main()

{ din_string A ("Пример строки 1");

char ss [] = "Строка для динамического объекта";

din_string *p = new din_string(ss,sizeof(ss));

p->display():

delete p; // уничтожить объект - неявный вызов деструктора

A.display();} // деструктор A будет вызван после окончания программы
```

Практические занятия
Разработка многомодульных приложений

Разработка многомодульных Windows приложений Управление формами

Для создания новой программы следует выбрать в меню **File – New – Application**. Откроется новый проект с пустой формой (**Form1**). В дальнейшем ее можно переименовать, задав новое значение свойства**Name**(программное имя формы).

Сохранение проекта следует осуществлять **File – Save Project As…**. В диалоговом окне можно задать новое имя для модуля формы (вместо Unit1) и для файла проекта (вместо Project1) или оставить имена, предлагаемые системой. Каждое программное приложение следует сохранять в отдельной специально созданной папке.

Разработка главной формы приложения

После перечисленных подготовительных действий можно перейти к проектированию главного окна приложения. Открытая форма (**Form1**) является его макетом. Заголовок окна должен содержать название программы (свойство**Caption**), главное окно приложения, как правило, может изменять размеры (свойствоBorderStyle = bsSizeAble) и содержать меню, отражающее функциональность программы.

Главное окно приложения обычно имеет меню. На рисунке приведен пример формы Form1 приложения, демонстрирующего выполнение вычислительных задач (Caption =“Работа с массивами”). Меню приложения содержит следующие пункты:

Файл (Открыть… – Сохранить… – Разделитель - Выход);

Вычисления (Настройки… - Разделитель - Вычислить)

Сервис (Фон… - Шрифт…)

Помощь (О программе…).

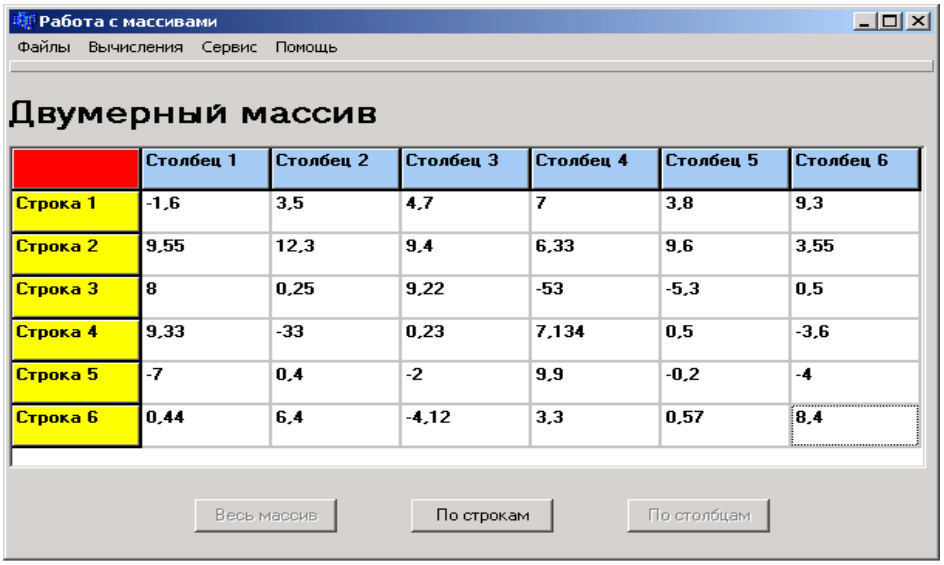


Рис. 1.1 Главное окно приложения (Form1)

На форме расположены компоненты Label1 (Caption = “Двумерные массивы”), StringGrid1, кнопки Button1 (Caption=”Весь массив”), Button2 (Caption=”По строкам”, Enabled = false) и Button3 (Caption=”По столбцам”, Enabled = false).

Компонент StringGrid позволяет хранить двумерный массив строковых значений, Его свойства: ColCount – количество столбцов (в примере равно 7), RowCount – количество строк (7), FixedCols и FixedRows – количество фиксированных столбцов и строк (по умолчанию 1), DefaultColWidth (82)- ширина столбца и DefaultRowHeight - высота строки в пикселях, принимаемые по умолчанию (применяются при начальной инициализации таблицы).

Векторное свойство StringGrid1->Cells [Acol] [ARow] (ACol – номер столбца, ARow – номер строки) позволяет осуществить доступ к ячейкам массива. Для разрешения редактирования значений таблицы во время работы свойство Options.goEdit должно быть равно true.

Вспомогательные формы

Кроме главной формы приложение может содержать вспомогательные формы. Для добавления новой формы используется меню **File – New – Form**. Если окно не будет изменять размер, то свойство**BorderStyle**=bsSingle или bsDialog (специально для диалоговых окон). Для сохранения модуля новой формы можно воспользоваться командой**File – Save As…**и также при желании изменить его название (свойство Name). На рисунке представлено диалоговое окно (Name=Form5, например, а заголовокCaption= “Настройки вычислений”), позволяющее указать производимые вычисления.

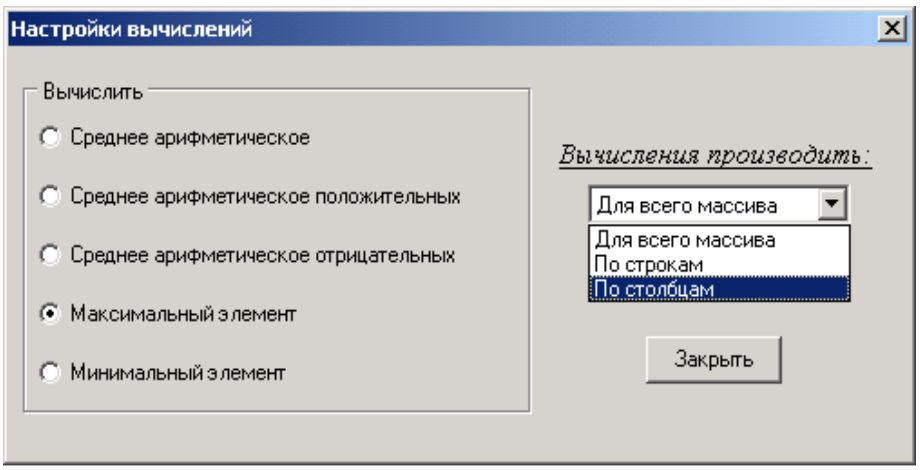


Рис. 1.2 Диалоговое окно

В инструментальной среде Builder C++ можно воспользоваться меню **Options | Project…**и с помощью диалогового окна**Project Options**(вкладка Forms) определить, каким образом приложение управляет своими окнами. В списке**Main Form**(Главная форма) можно выбрать главную форму, которая будет отображаться сразу после старта приложения. По умолчанию – это первая форма, которая была определена при проектировании.

Список **Auto-create forms**содержит формы, которые строятся при старте приложения. По умолчанию в него помещаются все формы, что ускоряет их отображение. Однако при большом количестве форм программа долго загружается и занимает в оперативной памяти много места. Разработчик может поместить часть форм в список Available forms (даже все, за исключением главной формы). Но при этом ему придется программно создавать каждую из форм списка перед ее вызовом.

Это можно сделать следующими способами (динамическое создание формы во время выполнения приложения):

- С использованием оператора new

Form2= new TForm2(Form1);

- С использованием метода объекта Application

Application->CreateForm(__classid(TForm2), &Form2);

Следующий оператор удаляет созданную ранее форму:

delete(Form2);

В нашем примере динамически создаются формы, в которых отображаются результаты вычислений. Примеры таких вспомогательных форм приведены на рисунке.

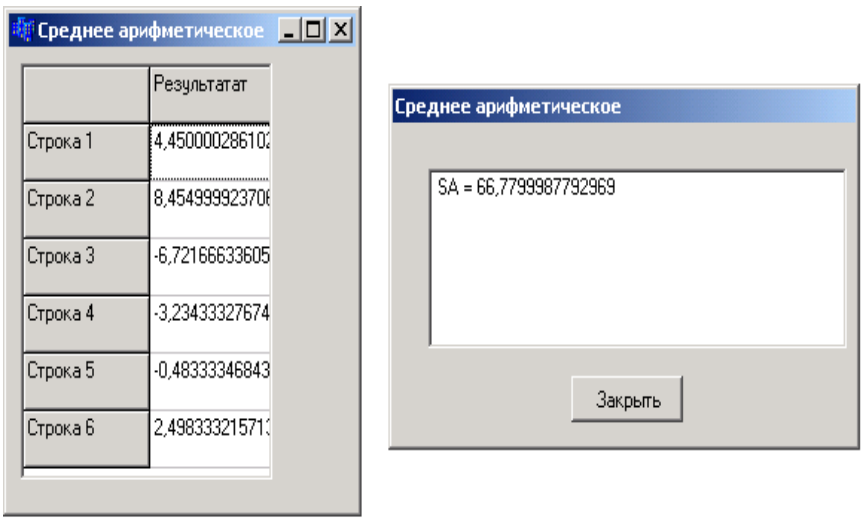


Рис.1.3 Результаты вычислений (вспомогательные формы Form3 и Form6)

Показать форму на экране можно в обычном режиме: Form2-> Show() или в модальном:Form2->ShowModal(),методHide()делает форму невидимой. Однако чтобы в программе можно было из одной формы обратиться к компонентам другой формы, необходимо воспользоваться менюFile - Include Unit Hdr… . Эта команда предложит программисту список разработанных, но еще «невидимых» из данной формы модулей, а после выбора вставит строку#include, например, #include“Unit2.h”.

Завершение приложения осуществляется методом Application->Terminate()из любой формы приложения, для главной формы можно воспользоваться ее методомClose().