

Лабораторная работа 2.2

Тема: Условные операторы и циклы в языке Python.

Цель работы: приобретение навыков программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Освоить операторы языка Python версии 3.x if, while, for, break и continue, позволяющих реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.

Порядок выполнения работы


1. Создаём аккаунт в GitHub. Затем создаём новый общедоступный репозиторий, в котором будет использована лицензия MIT и язык программирования Python.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

 RomanGorchakov ▾

Repository name *

/ Test

✓ Test is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-chainsaw](#) ?

Description (optional)

☒  Public

Anyone on the internet can see this repository. You choose who can commit.

☐  Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

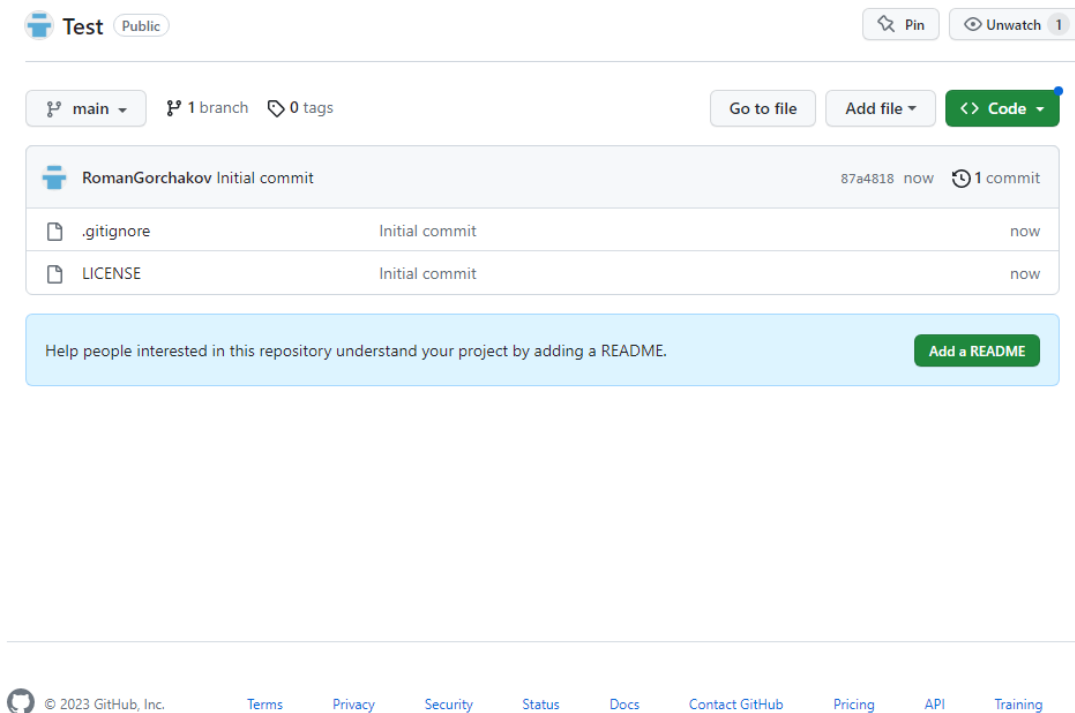
Choose a license

License: MIT License ▾


















A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository



2. Теперь необходимо дополнить файл `.gitignore` с необходимыми правилами для языка программирования Python. Для этого переходим по ссылке «<https://github.com/github/gitignore>» и скачиваем оттуда файл «Python.gitignore».

	Phalcon.gitignore	Remove trailing asterisks in Phalcon rules	10 years ago
	PlayFramework.gitignore	Added /project/project to PlayFramework.gitignore	7 years ago
	Plone.gitignore	Covered by global vim template	10 years ago
	Prestashop.gitignore	Update for Prestashop 1.7 (#3261)	3 years ago
	Processing.gitignore	Ignore transpiled .java and .class files (#3016)	4 years ago
	PureScript.gitignore	Update PureScript adding .spago (#3278)	3 years ago
	Python.gitignore	Update Python.gitignore	last year
	Qooxdoo.gitignore	Add gitignore for qooxdoo apps	13 years ago
	Qt.gitignore	Remove trailing whitespace	2 years ago
	R.gitignore	Merge pull request #3792 from jl5000/patch-1	2 years ago
	README.md	Merge pull request #3854 from AnilSeervi/patch-1	2 years ago
	ROS.gitignore	Added ignore for files created by <code>catkin_make_isolated</code>	6 years ago
	Racket.gitignore	Update Racket.gitignore	2 years ago
	Rails.gitignore	Ignore Rails .env according recommendations	2 years ago
	Raku.gitignore	Changes the name of Perl 6 to Raku (#3312)	3 years ago
	RhodesRhomobile.gitignore	Add Rhodes mobile application framework gitignore	13 years ago
	Ruby.gitignore	Ruby: ignore RuboCop remote inherited config files (#3197)	4 years ago

```
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

3. Теперь создаём файл «README.md», где вносим информацию о своей группе и ФИО. Сохраняем набранный текст через кнопку «Commit changes».



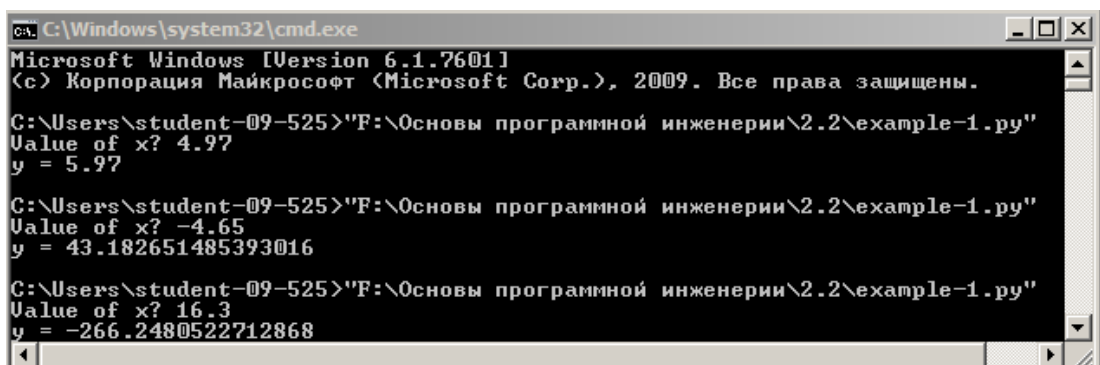
4. После этого нужно организовать репозиторий в соответствии с моделью ветвления Git-flow. Для этого В окне «Codespace» выбираем опцию «Create codespace on main», где введём команды: «git branch develop» и «git push -u origin develop» для создания ветки разработки; «git branch feature_branch» для создания ветки функций; «git branch release/1.0.0» для создания ветки релиза; «git checkout main» и «git branch hotfix» для создания веток hotfix.

```

@RomanGorchakov →/workspaces/Py2 (main) $ git checkout -b develop
Switched to a new branch 'develop'
• @RomanGorchakov →/workspaces/Py2 (develop) $ git checkout -b feature_branch
Switched to a new branch 'feature_branch'
• @RomanGorchakov →/workspaces/Py2 (feature_branch) $ git merge develop
Already up to date.
• @RomanGorchakov →/workspaces/Py2 (feature_branch) $ git checkout develop
Switched to branch 'develop'
• @RomanGorchakov →/workspaces/Py2 (develop) $ git branch release/1.0.0
• @RomanGorchakov →/workspaces/Py2 (develop) $ git merge release/1.0.0
Already up to date.
• @RomanGorchakov →/workspaces/Py2 (develop) $ git checkout release/1.0.0
Switched to branch 'release/1.0.0'
• @RomanGorchakov →/workspaces/Py2 (release/1.0.0) $ git merge develop
Already up to date.
• @RomanGorchakov →/workspaces/Py2 (release/1.0.0) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
• @RomanGorchakov →/workspaces/Py2 (main) $ git branch hotfix
• @RomanGorchakov →/workspaces/Py2 (main) $ git merge hotfix
Already up to date.
• @RomanGorchakov →/workspaces/Py2 (main) $ git checkout hotfix
Switched to branch 'hotfix'
• @RomanGorchakov →/workspaces/Py2 (hotfix) $ git merge main
Already up to date.
• @RomanGorchakov →/workspaces/Py2 (hotfix) $ git checkout develop
Switched to branch 'develop'
• @RomanGorchakov →/workspaces/Py2 (develop) $ git merge hotfix
Already up to date.
• @RomanGorchakov →/workspaces/Py2 (develop) $ git branch
* develop
  feature_branch
  hotfix
  main
  release/1.0.0
○ @RomanGorchakov →/workspaces/Py2 (develop) $

```

5. Создаём файл «example-1.py», в котором пользователю нужно ввести вещественное число, а программа вычислит значение функции, которая включает в себя это число.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-1.py"
Value of x? 4.97
y = 5.97

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-1.py"
Value of x? -4.65
y = 43.182651485393016

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-1.py"
Value of x? 16.3
y = -266.2480522712868

```

6. Создаём файл «example-2.py», в котором пользователю нужно ввести номер месяца, а программа выводит время года, в которое входит этот месяц

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-2.py"
Введите номер месяца: 6
Лето

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-2.py"
Введите номер месяца: 3
Весна

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-2.py"
Введите номер месяца: -6
Ошибка!
```

7. Создаём файл «example-3.py», в котором пользователю нужно ввести натуральное и вещественное числа, а программа вычислит конечную сумму, включающую в себя эти числа.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-3.py"
Value of n? 8
Value of x? 0.5
S = -0.49037772521645756

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-3.py"
Value of n? 6
Value of x? 2.3
S = 1.7383382490486274

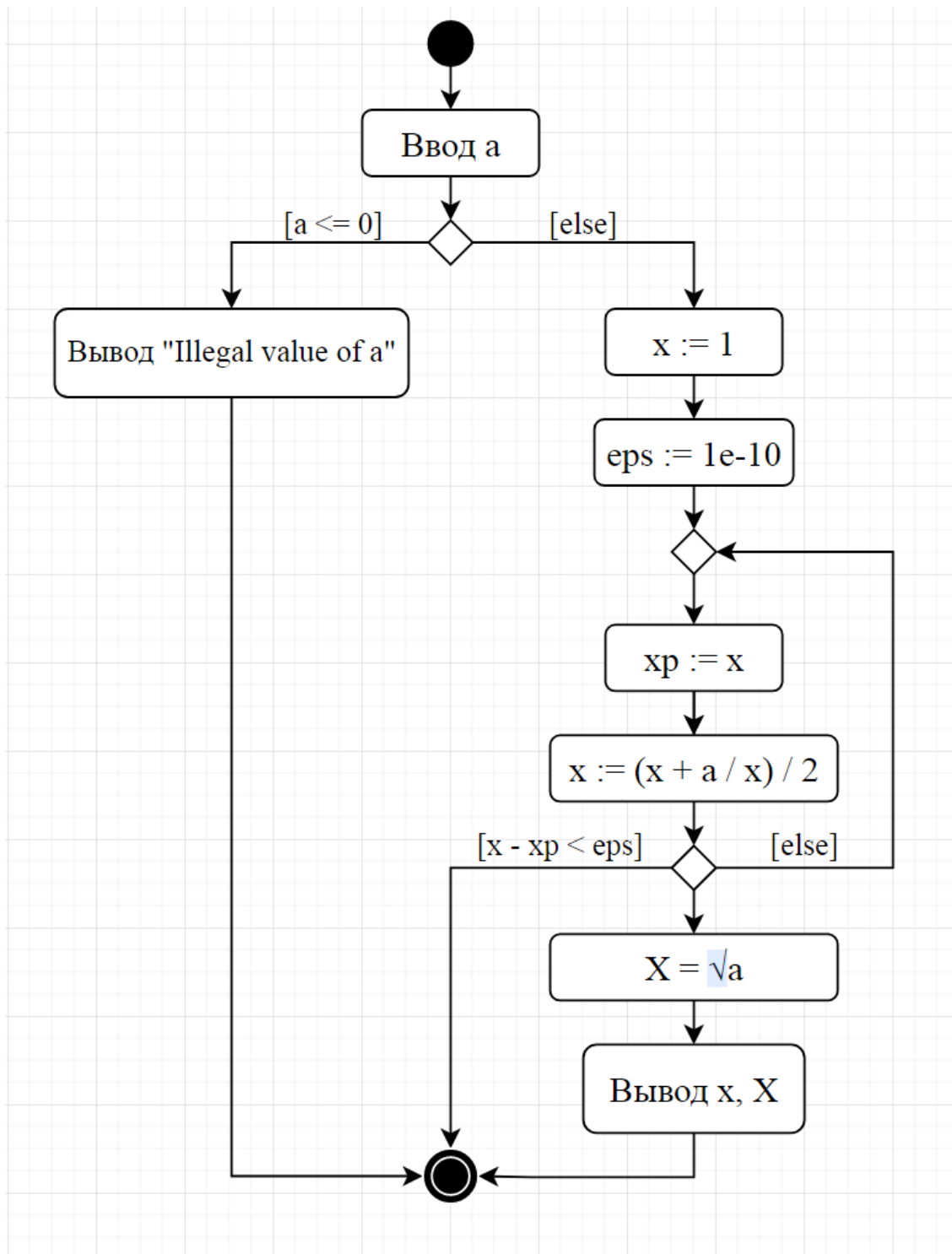
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-3.py"
Value of n? 2
Value of x? 9.9
S = 3.0389552415656667
```

8. Создаём файл «example-4.py», в котором пользователю нужно ввести вещественное число, а программа найдёт значение квадратного корня из этого числа с некоторой заданной точностью eps с помощью рекуррентного соотношения. Строим UML-диаграмму этой программы.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-4.py"
Value of a? 29.9
x = 5.468089245796927
X = 5.468089245796926

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-4.py"
Value of a? 13.2
x = 3.63318042491699
X = 3.63318042491699

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-4.py"
Value of a? 153
x = 12.36931687685298
X = 12.36931687685298
```

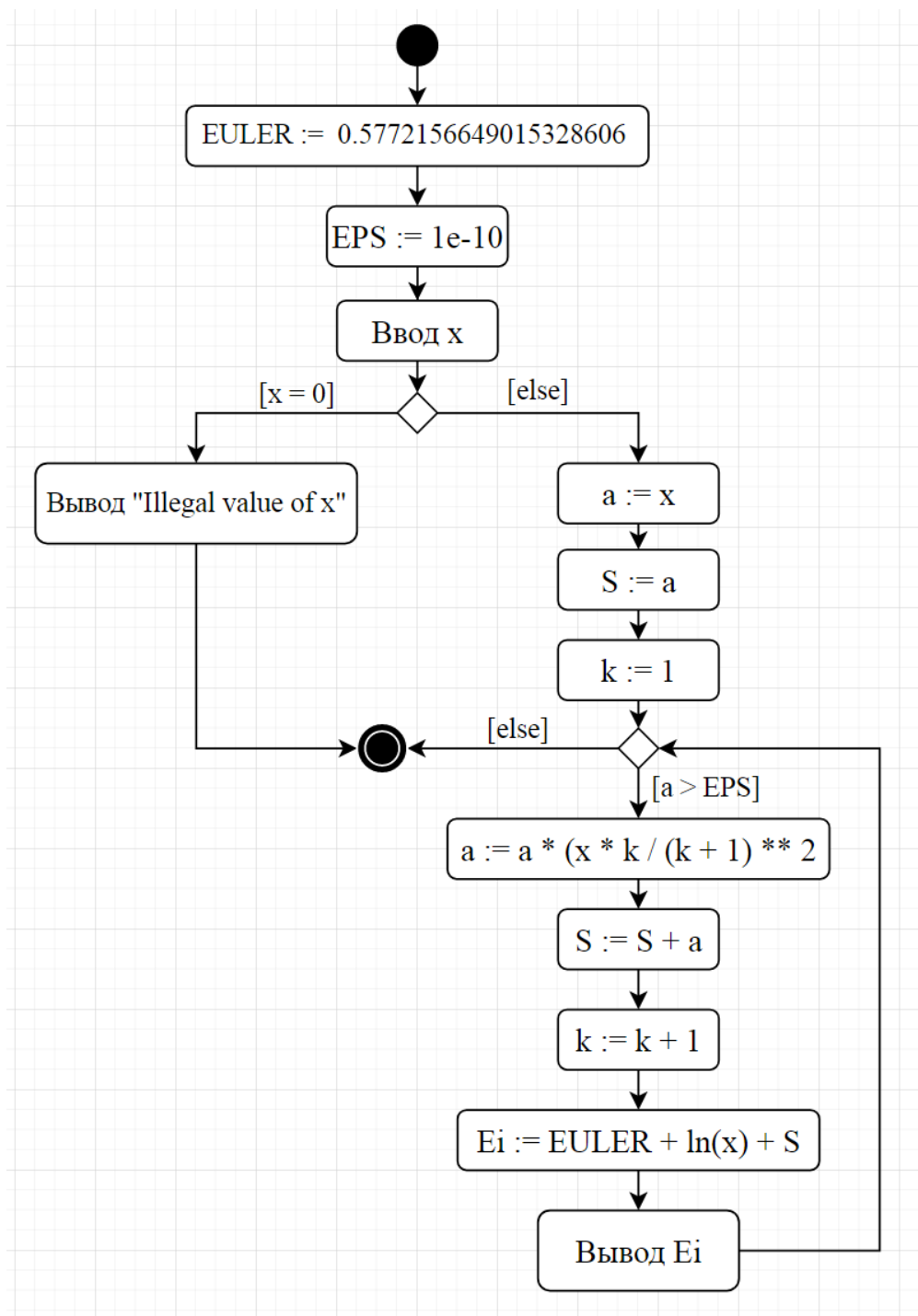


9. Создаём файл «example-5.py», в котором пользователю нужно ввести вещественное число, а программа высчитывает значение специальной интегральной показательной функции, включающей в себя это число. Строим UML-диаграмму этой программы.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-5.py"
Value of x? 76.4
Ei<76.4> = 2.008193749875176e+31

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-5.py"
Value of x? 3.52
Ei<3.52> = 14.115944297747639

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\example-5.py"
Value of x? 657.1
Ei<657.1> = 3.613558025226272e+282
```

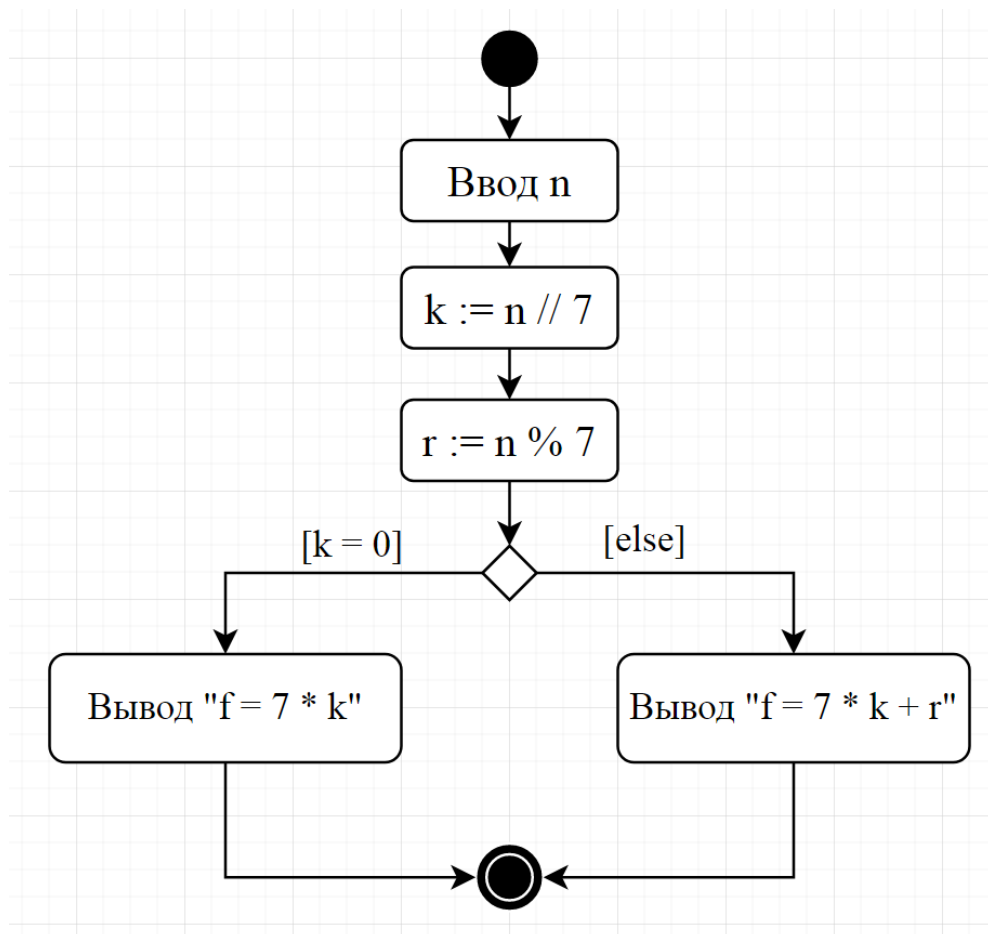


10. Создаём файл «individual-1.py», в котором пользователю нужно ввести натуральное число, а программа находит результат целочисленного деления этого числа на 7 и остаток от этого деления. Строим UML-диаграмму этой программы.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-1.py"
Введите число: 33
n == 7 * 4 + 5

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-1.py"
Введите число: -73
n == 7 * -11 + 4

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-1.py"
Введите число: 49
n == 7 * 7
```

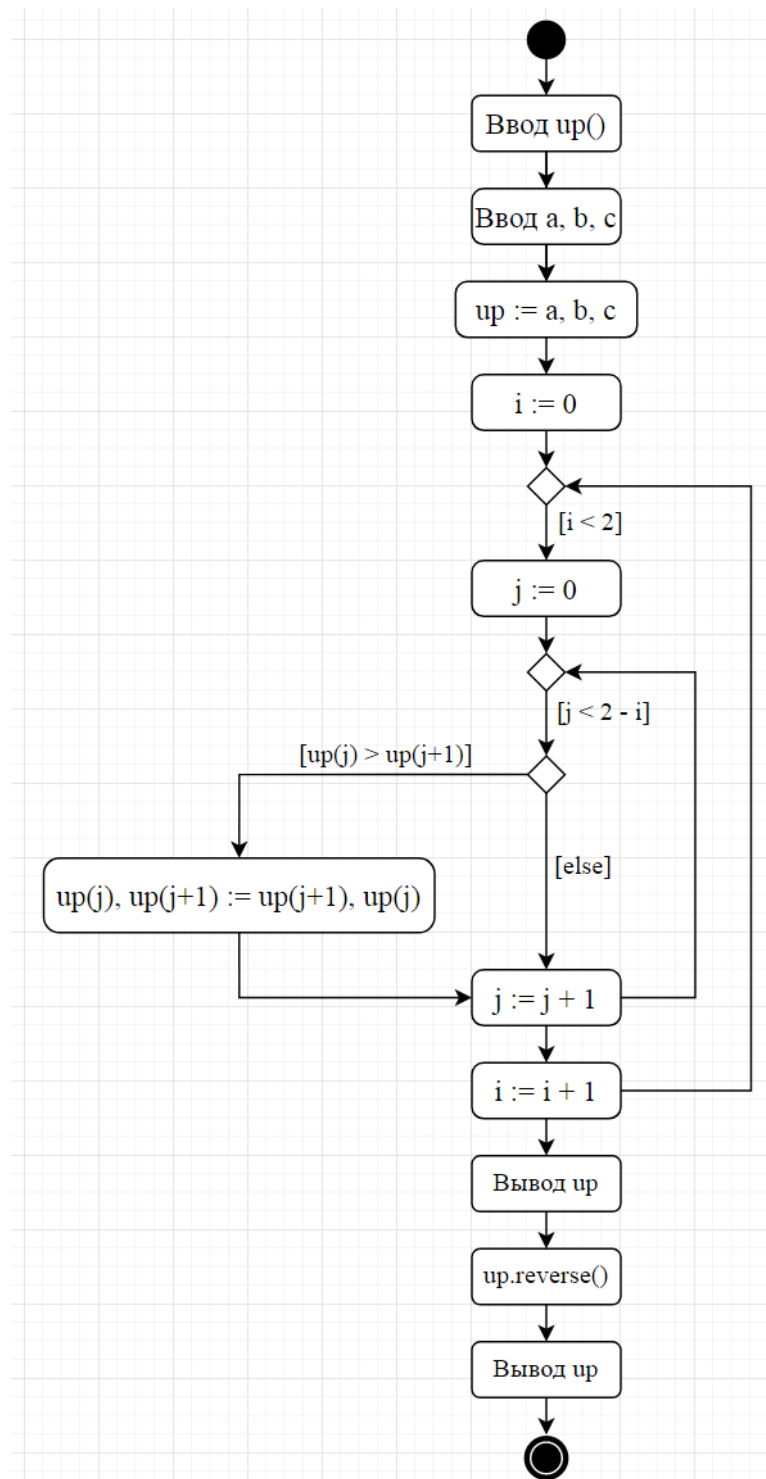


11. Создаём файл «individual-2.py», в котором пользователю нужно ввести три вещественных числа, а программа выводит их сначала в порядке возрастания, затем – в порядке убывания. Строим UML-диаграмму этой программы.

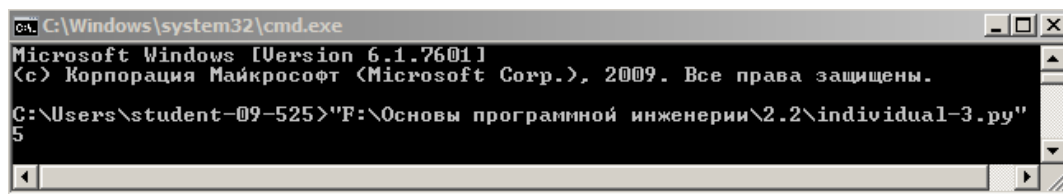
```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-2.py"
Введите три числа: 3.94 7.77 -3.81
[-3.81, 3.94, 7.77]
[7.77, 3.94, -3.81]

C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-2.py"
Введите три числа: -2.87 5.82 1.44
[-2.87, 1.44, 5.82]
[5.82, 1.44, -2.87]

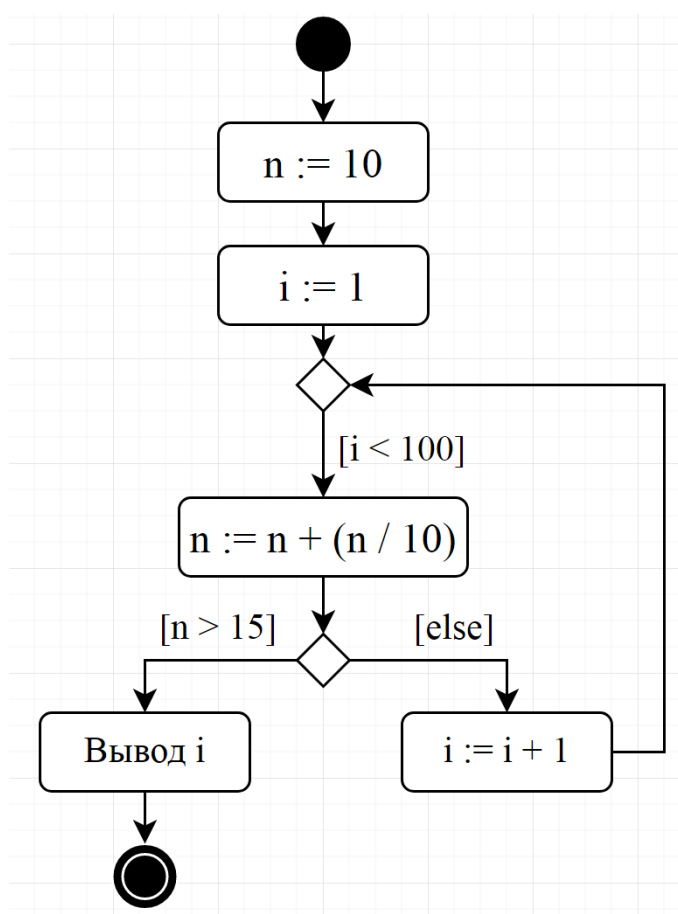
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-2.py"
Введите три числа: 82.47 -38.34 -59.24
[-59.24, -38.34, 82.47]
[82.47, -38.34, -59.24]
```



12. Создаём файл «individual-3.py», в котором программа высчитывает, через сколько дней спортсмен будет пробегать в день больше 15 километров, если в первый день он пробежал 10 километров, а норма увеличивается на 10% каждый день. Строим UML-диаграмму этой программы.

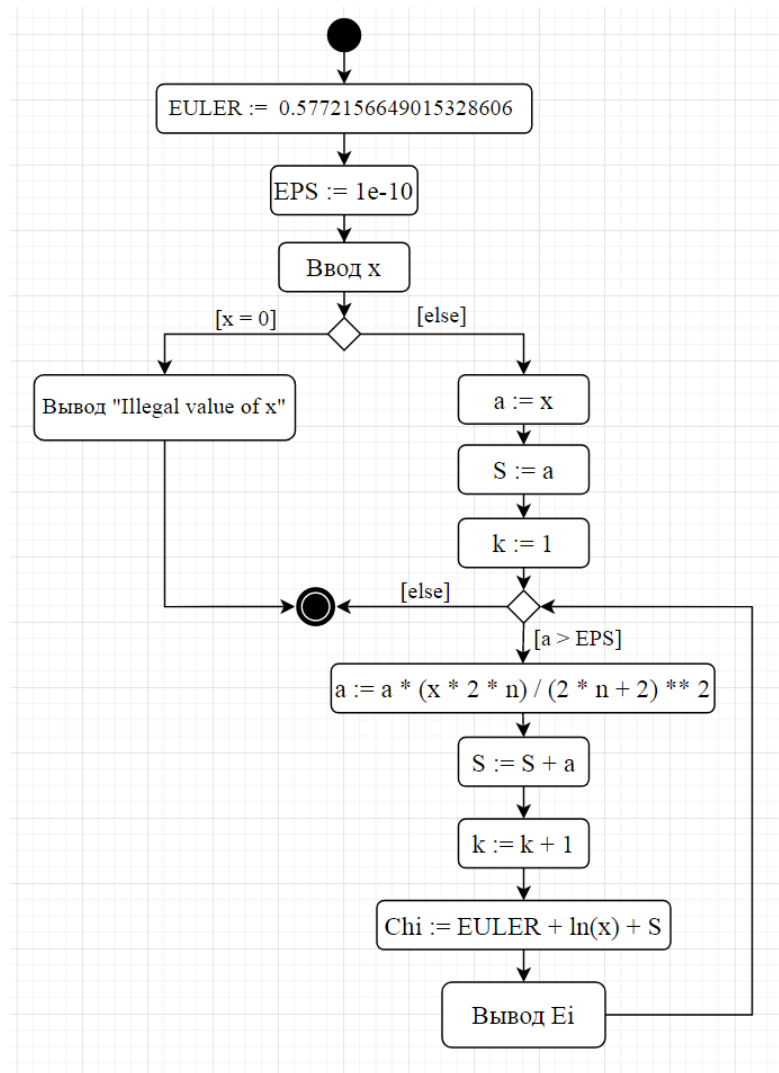


```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-3.py"
5
```



13. Создаём файл «individual-hard.py», в котором пользователю нужно ввести вещественное число, а программа высчитывает значение интегрального гиперболического косинуса из этого числа. Строим UML-диаграмму этой программы.

```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-hard.py"
Value of x? 2.12
Chi(2.12) = 4.17428220678033
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-hard.py"
Value of x? -4.45
Chi(-4.45) = -0.7557346044673059
C:\Users\student-09-525>"F:\Основы программной инженерии\2.2\individual-hard.py"
Value of x? 57.98
Chi(57.98) = 278500173269.77124
```



14. Выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.

```
@RomanGorchakov →/workspaces/Py2 (develop) $ git add .
@RomanGorchakov →/workspaces/Py2 (develop) $ git commit "Python files"
error: pathspec 'Python files' did not match any file(s) known to git
@RomanGorchakov →/workspaces/Py2 (develop) $ git commit -m "Python files"
[develop 06d6e37] Python files
15 files changed, 791 insertions(+)
create mode 100644 example-1.py
create mode 100644 example-2.py
create mode 100644 example-3.py
create mode 100644 example-4.py
create mode 100644 example-5.py
create mode 100644 example4.drawio
create mode 100644 example5.drawio
create mode 100644 individual-1.py
create mode 100644 individual-2.py
create mode 100644 individual-3.py
create mode 100644 individual-hard.drawio
create mode 100644 individual-hard.py
create mode 100644 individual1.drawio
create mode 100644 individual2.drawio
create mode 100644 individual3.drawio
@RomanGorchakov →/workspaces/Py2 (develop) $ git merge main
Already up to date.
@RomanGorchakov →/workspaces/Py2 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
@RomanGorchakov →/workspaces/Py2 (main) $ git merge develop
Updating e61cd95..06d6e37
Fast-forward
   | 14 ++++++++
   | 17 ++++++++
   | 13 ++++++++
   | 18 ++++++++
   | 22 ++++++++
   | 123 ++++++++
   | 126 ++++++++
   | 8 ++++++
   | 17 ++++++++
   | 7 ++++++
   | 126 ++++++++
   | 22 ++++++++
   | 64 ++++++++
   | 139 ++++++++
   | 75 ++++++++
15 files changed, 791 insertions(+)
create mode 100644 example-1.py
create mode 100644 example-2.py
create mode 100644 example-3.py
create mode 100644 example-4.py
create mode 100644 example-5.py
create mode 100644 example4.drawio
create mode 100644 example5.drawio
create mode 100644 individual-1.py
create mode 100644 individual-2.py
create mode 100644 individual-3.py
create mode 100644 individual-hard.drawio
create mode 100644 individual-hard.py
create mode 100644 individual1.drawio
create mode 100644 individual2.drawio
create mode 100644 individual3.drawio
@RomanGorchakov →/workspaces/Py2 (main) $
```

main1 branch0 tags

Go to fileAdd fileCode

RomanGorchakov Python files06d6e378 minutes ago4 commits

.gitignore	Create .gitignore	2 weeks ago
LICENSE	Create LICENSE	2 weeks ago
README.md	Create README.md	2 weeks ago
example-1.py	Python files	8 minutes ago
example-2.py	Python files	8 minutes ago
example-3.py	Python files	8 minutes ago
example-4.py	Python files	8 minutes ago
example-5.py	Python files	8 minutes ago
example4.drawio	Python files	8 minutes ago
example5.drawio	Python files	8 minutes ago
individual-1.py	Python files	8 minutes ago
individual-2.py	Python files	8 minutes ago
individual-3.py	Python files	8 minutes ago
individual-hard.drawio	Python files	8 minutes ago
individual-hard.py	Python files	8 minutes ago
individual1.drawio	Python files	8 minutes ago
individual2.drawio	Python files	8 minutes ago
individual3.drawio	Python files	8 minutes ago

README.md

Информация обо мне

Имя: Роман.

Фамилия: Горчаков.

Отчество: Владимирович.

Название группы: ПИЖ-6-о-22-1.

Контрольные вопросы

1. Для чего нужны диаграммы деятельности UML?

Диаграммы деятельности - это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов поведения системы. Диаграмма деятельности - это, по существу, блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой, однако, по сравнению с последней, у ней есть явные преимущества: поддержка многопоточности и объектно-ориентированного проектирования.

Диаграмма деятельности показывает поток переходов от одной деятельности к другой. Деятельность – это продолжающийся во времени неатомарный шаг вычислений в автомате. Деятельности в конечном счете приводят к выполнению некоего действия, составленного из выполняемых атомарных вычислений, каждое из которых либо изменяет состояние системы, либо возвращает какое-то значение. Действие может заключаться в вызове другой операции, отправке сигнала, создании или уничтожении объекта либо в простом вычислении - скажем, значения выражения.

2. Что такое состояние действия и состояние деятельности?

В потоке управления, моделируемом диаграммой деятельности, происходят различные события. Вы можете вычислить выражение, в результате чего изменяется значение некоторого атрибута или возвращается некоторое значение. Также, например, можно выполнить операцию над объектом, послать ему сигнал или даже создать его или уничтожить. Все эти выполняемые атомарные вычисления называются состояниями действия, поскольку каждое из них есть состояние системы, представляющее собой выполнение некоторого действия.

В противоположность этому состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время. Можно считать, что состояние действия – это

частный вид состояния деятельности, а конкретнее – такое состояние, которое не может быть подвергнуто дальнейшей декомпозиции. А состояние деятельности можно представлять себе, как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

3. Какие нотации существуют для обозначения переходов и ветвлений в диаграммах деятельности?

Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока используются переходы, показывающие путь из одного состояния действия или деятельности в другое. В UML переход представляется простой линией со стрелкой.

Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования любого потока управления. Как и в блок-схеме, вы можете включить в модель ветвление, которое описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Точка ветвления представляется ромбом.

4. Какой алгоритм является алгоритмом разветвляющейся структуры?

Алгоритм разветвляющейся структуры – это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия.

5. Чем отличается разветвляющийся алгоритм от линейного?

В линейных алгоритмах команды выполняются подряд, а в разветвляющихся их последовательность зависит от выполнения некоторого условия.

6. Что такое условный оператор? Какие существуют его формы?

Условный оператор – это оператор, предназначенный для выбора к исполнению одного из возможных действий в зависимости от некоторого условия. Возможны следующие варианты его использования: конструкция if, конструкция if – else и конструкция if – elif – else.

7. Какие операторы сравнения используются в Python?

В языке программирования Python для сравнения используют следующие операторы: < (меньше); <= (меньше или равно); = и == (равно); != (не равно); > (больше); >= (больше или равно).

8. Что называется простым условием? Приведите примеры.

Простым условием называется конструкция, включающая в себя только одно условие.

Пример:

```
lst = [1, 2, 3]

if lst[2] == 3:
    lst.append(4)
else:
    lst.append(3)
print(lst)
```

9. Что такое составное условие? Приведите примеры.

Составным условием называется конструкция, включающая в себя больше одного условия.

Пример:

```
lst = [1, 2, 4]

if (lst[2] == 3) and (lst[3] == 4):
    lst.append(4)
elif lst[2] == 4:
    lst.pop()
    lst.append(5)
else:
    lst.append(3)
print(lst)
```

10. Какие логические операторы допускаются при составлении сложных условий?

Для записи сложных условий можно применять логические операции:

- and – логическое «И» для двух условий. Возвращает True, если оба условия истинны, иначе возвращает False;
- or – логическое «ИЛИ» для двух условий. Возвращает False, если оба условия ложны, иначе возвращает True;
- not – логическое «НЕ» для одного условия.

11. Может ли оператор ветвления содержать внутри себя другие ветвления?

Да, может.

12. Какой алгоритм является алгоритмом циклической структуры?

Алгоритм циклической структуры – это алгоритм, в котором происходит многократное повторение одного и того же участка программы.

13. Типы циклов в языке Python.

В Python есть два вида циклов: while и for. В while условие задаётся явным образом. В for перебирается каждый элемент коллекции.

14. Назовите назначение и способы применения функции range.

Функция range возвращает неизменяемую последовательность чисел в виде объекта range. Также она хранит только информацию о значениях start, stop и step и вычисляет значения по мере необходимости. Это значит, что независимо от размера диапазона, который описывает функция range, она всегда будет занимать фиксированный объем памяти.

15. Как с помощью функции range организовать перебор значений от 15 до 0 с шагом 2?

```
print(list(range(15, 0, -2)))  
# [15, 13, 11, 9, 7, 5, 3, 1]
```

16. Могут ли быть циклы вложенными?

Да, могут.

17. Как образуется бесконечный цикл и как выйти из него?

Бесконечный цикл while – это цикл, в котором условие никогда не становится ложным. Это значит, что тело выполняется снова и снова, а цикл никогда не заканчивается. Выйти из него можно, нажав комбинацию клавиш Ctrl + C

18. Для чего нужен оператор break?

Оператор break предназначен для досрочного прерывания работы цикла while.

19. Где употребляется оператор continue и для чего он используется?

Оператор continue запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

20. Для чего нужны стандартные потоки stdout и stderr?

В операционной системе по умолчанию присутствуют стандартных потока вывода на консоль: буферизованный поток `stdout` для вывода данных и информационных сообщений, а также небуферизованный поток `stderr` для вывода сообщений об ошибках.

21. Как в Python организовать вывод в стандартный поток `stderr`?

Для того, чтобы использовать поток `stderr` необходимо передать его в параметре `file` функции `print`.

22. Каково назначение функции `exit`?

В Python завершить программу и передать операционной системе заданный код возврата можно посредством функции `exit`. В данном примере выполняется вызов `exit(1)`, что приводит к немедленному завершению программы и операционной системе передается 1 в качестве кода возврата, что говорит о том, что в процессе выполнения программы произошли ошибки.