Конфигурационное управление

Сборник домашних заданий

Группа 12

РТУ МИРЭА – 2024

Оглавление

О домашних заданиях

Домашние задания (ДЗ) выполняются в публично доступном git-репозитории. Студент самостоятельно выбирает язык реализации. Ход разработки ДЗ должен быть отражен в истории коммитов с детальными сообщениями. Для автоматической сборки проекта используется файл-скрипт.

Документация по ДЗ оформляется в виде readme.md, который содержит:

1. Общее описание.
2. Описание всех функций и настроек.
3. Описание команд для сборки проекта.
4. Примеры использования в виде скриншотов, желательно в анимированном/видео формате, доступном для web-просмотра.
5. Результаты прогона тестов.

Версия readme.md с указанием URL-адреса репозитория сохраняется в СДО в формате PDF.

Защита ДЗ проводится с участием преподавателя практических занятий, очно и в специально отведенное время. Для успешной защиты, особенно в случае неубедительной истории коммитов, может понадобиться добавить новые, несущественные функции в проект.

Список публичных git-сервисов для репозиториев ДЗ:

- github.com
- gitea.com
- gitlab.com
- gitflic.ru
- hub.mos.ru
- gitverse.ru
- gitee.com

Вариант №1

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **json** и содержит все действия во время последнего сеанса работы с эмулятором.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `date`.
2. `who`.
3. `history`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого находятся списки файлов и папок. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным хеш-значением.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- Файл с заданным хеш-значением в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в стандартный вывод.

Однострочные комментарии:

```
C Это однострочный комментарий
```

Многострочные комментарии:

```
(*
Это многострочный
комментарий
*)
```

Словари:

```
{
  имя = значение,
  имя = значение,
  имя = значение,
  ...
}
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
def имя = значение
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
?[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `max()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—17	Биты 18—28
26	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=26, B=162, C=166):

```
0x5A, 0x14, 0x98, 0x02
```

Чтение из памяти

A	**B**	**C**
Биты 0—4	Биты 5—17	Биты 18—30
6	Адрес	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=6, B=713, C=633):

```
0x26, 0x59, 0xE4, 0x09
```

Запись в память

A	**B**	**C**	**D**
Биты 0—4	Биты 5—17	Биты 18—30	Биты 31—43
1	Смещение	Адрес	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле D) и смещения (поле B).

Тест (A=1, B=669, C=345, D=301):

```
0xA1, 0x53, 0x64, 0x85, 0x96, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—4	Биты 5—17	Биты 18—30
10	Адрес	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=10, B=183, C=724):

```
0xEA, 0x16, 0x50, 0x0B
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 7. Результат записать в исходный вектор.

Вариант №2

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tac`.
2. `du`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
:: Это однострочный комментарий
```

Многострочные комментарии:

```
{-
Это многострочный
комментарий
-}
```

Массивы:

```
list( значение, значение, значение, ... )
```

Словари:

```
table([
  имя = значение,
  имя = значение,
  имя = значение,
  ...
])
```

Имена:

```
[_a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
set имя = значение;
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—25	Биты 26—28
8	Константа	Адрес

Размер команды: 4 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=8, B=307, C=5):

```
0x38, 0x13, 0x00, 0x14
```

Чтение из памяти

A	**B**	**C**
Биты 0—3	Биты 4—19	Биты 20—22
14	Адрес	Адрес

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=14, B=193, C=1):

```
0x1E, 0x0C, 0x10
```

Запись в память

A	**B**	**C**
Биты 0—3	Биты 4—6	Биты 7—9
6	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=6, B=5, C=4):

```
0x56, 0x02
```

Унарная операция: `побитовое "не"`

A	B	C
Биты 0—3	Биты 4—19	Биты 20—22
9	Адрес	Адрес

Размер команды: 3 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=9, B=780, C=2):

```
0xC9, 0x30, 0x20
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "не"` над вектором длины 5. Результат записать в новый вектор.

Вариант №3

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `wc`.
2. `history`.
3. `uptime`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить только для коммитов позже заданной даты.

Конфигурационный файл имеет формат **yaml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Дата коммитов в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке xml** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Массивы:

```
'( значение значение значение ... )
```

Словари:

```
{
  имя : значение,
  имя : значение,
  имя : значение,
  ...
}
```

Имена:

```
[_a-zA-Z]+
```

Значения:

- Числа.
- Строки.

- Массивы.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
var имя значение;
```

Вычисление константы на этапе трансляции:

```
[имя]
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—4	Биты 5—33
2	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=2, B=1002):

```
0x42, 0x7D, 0x00, 0x00, 0x00
```

Чтение из памяти

A
Биты 0—4
6

Размер команды: 1 байт. Операнд: ячейка памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=6):

```
0x06
```

Запись в память

A
Биты 0—4
4

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=4):

```
0x04
```

Унарная операция: **унарный минус**

A

A
Биты 0—4
12

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=12):

```
0x0C
```

Тестовая программа

Выполнить поэлементно операцию `унарный минус` над вектором длины 4. Результат записать в новый вектор.

Вариант №4

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tac`.
2. `head`.
3. `cat`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

```
{#
Это многострочный
комментарий
#}
```

Массивы:

```
( значение, значение, значение, ... )
```

Словари:

```
$[
  имя : значение,
  имя : значение,
  имя : значение,
  ...
]
```

Имена:

```
[a-zA-Z]+
```

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
var имя = значение
```

Вычисление константного выражения на этапе трансляции (инфиксная форма), пример:

```
@{имя + 1}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `sort()`.
5. `max()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—7	Биты 8—32
13	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=13, B=2, C=179):

```
0x4D, 0xB3, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—20	Биты 21—23	Биты 24—26
25	Смещение	Адрес	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле B). Результат: регистр по адресу, которым является поле C.

Тест (A=25, B=1022, C=5, D=1):

```
0xD9, 0x7F, 0xA0, 0x01
```

Запись в память

A	**B**	**C**
Биты 0—4	Биты 5—7	Биты 8—36
30	Адрес	Адрес

Размер команды: 5 байт. Операнд: регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=30, B=7, C=350):

```
0xFE, 0x5E, 0x01, 0x00, 0x00
```

Бинарная операция: ">"

A	**B**	**C**
Биты 0—4	Биты 5—7	Биты 8—10
4	Адрес	Адрес

Размер команды: 2 байт. Первый операнд: регистр по адресу, которым является поле C. Второй операнд: регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=4, B=5, C=6):

```
0xA4, 0x06
```

Тестовая программа

Выполнить поэлементно операцию ">" над двумя векторами длины 8. Результат записать в первый вектор.

Вариант №5

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `chown`.
2. `uptime`.
3. `rev`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка Python (pip)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **ini** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.

- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке json** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
C Это однострочный комментарий
```

Многострочные комментарии:

```
<!--
Это многострочный
комментарий
-->
```

Массивы:

```
{ значение. значение. значение. ... }
```

Словари:

```
([
  имя : значение,
  имя : значение,
  имя : значение,
  ...
])
```

Имена:

```
[a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.

- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
def имя = значение
```

Вычисление константы на этапе трансляции:

```
?(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**

A	B
Биты 0—6	Биты 7—34
122	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=122, B=1013):

```
0xFA, 0xFA, 0x01, 0x00, 0x00
```

Чтение из памяти

A
Биты 0—6
29

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=29):

```
0x1D, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	B
Биты 0—6	Биты 7—19
19	Адрес

Размер команды: 5 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=19, B=176):

```
0x13, 0x58, 0x00, 0x00, 0x00
```

Унарная операция: `abs()`

A	B
Биты 0—6	Биты 7—22
57	Смещение

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Результат: новый элемент на стеке.

Тест (A=57, B=946):

```
0x39, 0xD9, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 7. Результат записать в исходный вектор.

Вариант №6

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указаны дата и время. Для каждого действия указан пользователь.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tree`.
2. `mkdir`.
3. `wc`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
:: Это однострочный комментарий
```

Многострочные комментарии:

```
{
Это многострочный
комментарий
}
```

Словари:

```
table(
  имя => значение,
  имя => значение,
  имя => значение,
  ...
)
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
имя = значение
```

Вычисление константного выражения на этапе трансляции (префиксная форма), пример:

```
[+ имя 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `pow()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—6	Биты 7—34
29	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=29, B=803):

```
0x9D, 0x91, 0x01, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—6	Биты 7—18
26	Смещение

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=26, B=236):

```
0x1A, 0x76, 0x00
```

Запись в память

A	**B**
Биты 0—6	Биты 7—30
91	Адрес

Размер команды: 4 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=91, B=640):

```
0x5B, 0x40, 0x01, 0x00
```

Бинарная операция: умножение

A	**B**
Биты 0—6	Биты 7—30
110	Адрес

Размер команды: 4 байт. Первый операнд: регистр-аккумулятор. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=110, B=670):

```
0x6E, 0x4F, 0x01, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 7 и числом 132. Результат записать в исходный вектор.

Вариант №7

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `uname`.
2. `uptime`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся сообщения. Граф необходимо строить для ветки с заданным именем.

Конфигурационный файл имеет формат **csv** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Имя ветки в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке toml** попадает в стандартный вывод.

Массивы:

```
[ значение, значение, значение, ... ]
```

Словари:

```
{
  имя => значение,
  имя => значение,
  имя => значение,
  ...
}
```

Имена:

```
[a-z][a-z0-9_]*
```

Значения:

- Числа.
- Строки.
- Массивы.

- Словари.

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
def имя := значение;
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	B	C
Биты 0—5	Биты 6—12	Биты 13—35
49	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=49, B=25, C=960):

```
0x71, 0x06, 0x78, 0x00, 0x00
```

Чтение из памяти

A	B	C	D
Биты 0—5	Биты 6—19	Биты 20—26	Биты 27—33
11	Смещение	Адрес	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле B). Результат: регистр по адресу, которым является поле C.

Тест (A=11, B=212, C=91, D=110):

```
0x0B, 0x35, 0xB0, 0x75, 0x03
```

Запись в память

A	B	C
Биты 0—5	Биты 6—12	Биты 13—25
63	Адрес	Адрес

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=63, B=111, C=803):

```
0xFF, 0x7B, 0x64, 0x00
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	B	C

A	**B**	**C**
Биты 0—5	Биты 6—12	Биты 13—19
31	Адрес	Адрес

Размер команды: 3 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=31, B=20, C=21):

```
0x1F, 0xA5, 0x02
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг вправо` над двумя векторами длины 6. Результат записать в новый вектор.

Вариант №8

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **xml** и содержит:

- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `pwd`.
2. `chown`.
3. `uniq`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

```
'( значение значение значение ... )
```

Словари:

```
$[
  имя : значение,
  имя : значение,
  имя : значение,
  ...
]
```

Имена:

```
[a-zA-Z]+
```

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
def имя = значение
```

Вычисление константы на этапе трансляции:

```
?{имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—2	Биты 3—32
0	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=0, B=559):

```
0x78, 0x11, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**

A	B
Биты 0—2	Биты 3—25
7	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=7, B=180):

```
0xA7, 0x05, 0x00, 0x00, 0x00
```

Запись в память

A	B
Биты 0—2	Биты 3—25
6	Адрес

Размер команды: 5 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=6, B=732):

```
0xE6, 0x16, 0x00, 0x00, 0x00
```

Бинарная операция: `побитовое "и"`

A
Биты 0—2
4

Размер команды: 5 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=4):

```
0x04, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "и"` над вектором длины 6 и числом 176. Результат записать в исходный вектор.

Вариант №9

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `head`.
2. `tac`.
3. `touch`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить только для коммитов позже заданной даты.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Дата коммитов в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке json** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Массивы:

```
#( значение значение значение ... )
```

Имена:

```
[_a-zA-Z]+
```

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
global имя = значение;
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
![имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `chr()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**

A	**B**
Биты 0—7	Биты 8—16
225	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=225, B=347):

```
0xE1, 0x5B, 0x01
```

Чтение из памяти

A	**B**
Биты 0—7	Биты 8—16
94	Смещение

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=94, B=168):

```
0x5E, 0xA8, 0x00
```

Запись в память

A	**B**
Биты 0—7	Биты 8—23
220	Адрес

Размер команды: 3 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=220, B=724):

```
0xDC, 0xD4, 0x02
```

Бинарная операция: "<"

A	**B**
Биты 0—7	Биты 8—23

A	**B**
114	Адрес

Размер команды: 3 байт. Первый операнд: регистр-аккумулятор. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=114, B=443):

```
0x72, 0xBB, 0x01
```

Тестовая программа

Выполнить поэлементно операцию "<" над вектором длины 5 и числом 125. Результат записать в исходный вектор.

Вариант №10

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `clear`.
2. `uname`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка Java (Maven)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Конфигурационный файл имеет формат **xml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке yaml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

```
=begin
Это многострочный
комментарий
=cut
```

Словари:

```
{
  имя : значение,
  имя : значение,
  имя : значение,
  ...
}
```

Имена:

```
[_a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
(define имя значение);
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
$(имя 1 +)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `mod()`.
5. `max()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—2	Биты 3—32
0	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=0, B=559):

```
0x78, 0x11, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—2	Биты 3—25
7	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=7, B=180):

```
0xA7, 0x05, 0x00, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—2	Биты 3—25
6	Адрес

Размер команды: 5 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=6, B=732):

```
0xE6, 0x16, 0x00, 0x00, 0x00
```

Бинарная операция: `побитовое "и"`

A
Биты 0—2
4

Размер команды: 5 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=4):

```
0x04, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "и"` над вектором длины 6 и числом 176. Результат записать в исходный вектор.

Вариант №11

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **yaml** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указаны дата и время.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `rmdir`.
2. `echo`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Построить граф зависимостей для коммитов, в узлах которого содержатся дата, время и автор коммита.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке xml** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
C Это однострочный комментарий
```

Словари:

```
[
  имя => значение,
  имя => значение,
  имя => значение,
  ...
]
```

Имена:

```
[_a-zA-Z]+
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
имя := значение;
```

Вычисление константы на этапе трансляции:

```
^(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—7	Биты 8—10	Биты 11—28
245	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=245, B=0, C=629):

```
0xF5, 0xA8, 0x13, 0x00
```

Чтение из памяти

A	B	C	D
Биты 0—7	Биты 8—10	Биты 11—13	Биты 14—29
69	Адрес	Адрес	Смещение

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле D). Результат: регистр по адресу, которым является поле C.

Тест (A=69, B=7, C=7, D=307):

```
0x45, 0xFF, 0x4C, 0x00
```

Запись в память

A	B	C	D
Биты 0—7	Биты 8—23	Биты 24—26	Биты 27—29
102	Смещение	Адрес	Адрес

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле B).

Тест (A=102, B=972, C=7, D=7):

```
0x66, 0xCC, 0x03, 0x3F
```

Бинарная операция: `max()`

A	B	C	D

A	**B**	**C**	**D**
Биты 0—7	Биты 8—10	Биты 11—13	Биты 14—16
24	Адрес	Адрес	Адрес

Размер команды: 3 байт. Первый операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле D. Второй операнд: регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=24, B=0, C=1, D=7):

```
0x18, 0xC8, 0x01
```

Тестовая программа

Выполнить поэлементно операцию `max()` над вектором длины 5 и числом 158. Результат записать в исходный вектор.

Вариант №12

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **json** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `mv`.
2. `chmod`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить для тега с заданным именем.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- Имя тега в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке yaml** попадает в стандартный вывод.

Многострочные комментарии:

```
{-
Это многострочный
комментарий
-}
```

Массивы:

```
[ значение, значение, значение, ... ]
```

Словари:

```
@{
  имя = значение;
```

```
  имя = значение;
  имя = значение;
  ...
}
```

Имена:

```
[a-zA-Z]+
```

Значения:

- Числа.
- Строки.
- Массивы.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя := значение;
```

Вычисление константы на этапе трансляции:

```
.(имя).
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-

логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—6	Биты 7—18
7	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=7, B=15, C=200):

```
0x7F, 0x64, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—2	Биты 3—6	Биты 7—13	Биты 14—17
1	Адрес	Смещение	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле C). Результат: регистр по адресу, которым является поле D.

Тест (A=1, B=13, C=25, D=10):

```
0xE9, 0x8C, 0x02, 0x00
```

Запись в память

A	**B**	**C**

A	**B**	**C**
Биты 0—2	Биты 3—6	Биты 7—20
6	Адрес	Адрес

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=6, B=14, C=925):

```
0xF6, 0xCE, 0x01, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**	**D**
Биты 0—2	Биты 3—6	Биты 7—13	Биты 14—17
0	Адрес	Смещение	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле C). Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле D.

Тест (A=0, B=12, C=89, D=15):

```
0xE0, 0xEC, 0x03, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 5. Результат записать в новый вектор.

Вариант №13

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **yaml** и содержит:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `echo`.
2. `chmod`.
3. `uname`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке xml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Словари:

```
{
  имя = значение;
  имя = значение;
  имя = значение;
  ...
}
```

Имена:

```
[a-z][a-z0-9_]*
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
значение -> имя;
```

Вычисление константы на этапе трансляции:

```
$(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—19	Биты 20—22
29	Константа	Адрес

Размер команды: 3 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=29, B=554, C=0):

```
0x5D, 0x45, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—11	Биты 12—14	Биты 15—17
23	Смещение	Адрес	Адрес

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле B). Результат: регистр по адресу, которым является поле D.

Тест (A=23, B=121, C=1, D=3):

```
0x37, 0x9F, 0x01
```

Запись в память

A	**B**	**C**
Биты 0—4	Биты 5—7	Биты 8—10
21	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=21, B=2, C=5):

```
0x55, 0x05
```

Унарная операция: `sqrt()`

A	**B**	**C**
Биты 0—4	Биты 5—7	Биты 8—10
14	Адрес	Адрес

Размер команды: 2 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=14, B=3, C=3):

```
0x6E, 0x03
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 5. Результат записать в исходный вектор.

Вариант №14

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `who`.
2. `mkdir`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка Python (pip)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке yaml** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

```
|#
Это многострочный
комментарий
#|
```

Словари:

```
table(
  имя => значение,
  имя => значение,
  имя => значение,
  ...
)
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
def имя := значение;
```

Вычисление константного выражения на этапе трансляции (префиксная форма), пример:

```
![+ имя 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `chr()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—6	Биты 7—37
9	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=9, B=596):

```
0x09, 0x2A, 0x01, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—6	Биты 7—12
67	Смещение

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=67, B=47):

```
0xC3, 0x17, 0x00, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—6	Биты 7—31
7	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=7, B=821):

```
0x87, 0x9A, 0x01, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	**B**
Биты 0—6	Биты 7—12
48	Смещение

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=48, B=27):

```
0xB0, 0x0D, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 6. Результат записать в исходный вектор.

Вариант №15

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **csv** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `wc`.
2. `rm`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Построить граф зависимостей для коммитов, в узлах которого содержатся дата, время и автор коммита. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным именем.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.
- Файл с заданным именем в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке yaml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
% Это однострочный комментарий
```

Массивы:

```
#( значение, значение, значение, ... )
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
def имя = значение;
```

Вычисление константы на этапе трансляции:

```
@{имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—6	Биты 7—37
9	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=9, B=596):

```
0x09, 0x2A, 0x01, 0x00, 0x00
```

Чтение из памяти

A	**B**

A	B
Биты 0—6	Биты 7—12
67	Смещение

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=67, B=47):

```
0xC3, 0x17, 0x00, 0x00, 0x00
```

Запись в память

A	B
Биты 0—6	Биты 7—31
7	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=7, B=821):

```
0x87, 0x9A, 0x01, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	B
Биты 0—6	Биты 7—12
48	Смещение

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=48, B=27):

```
0xB0, 0x0D, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 6. Результат записать в исходный вектор.

Вариант №16

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **xml** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `uname`.
2. `tail`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
% Это однострочный комментарий
```

Многострочные комментарии:

```
#=
Это многострочный
комментарий
=#
```

Массивы:

```
{ значение, значение, значение, ... }
```

Имена:

```
[a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
значение -> имя
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
^[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `chr()`.
4. `mod()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—6	Биты 7—28
122	Константа

Размер команды: 4 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=122, B=485):

```
0xFA, 0xF2, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—6	Биты 7—25
32	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=32, B=204):

```
0x20, 0x66, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—6	Биты 7—25
100	Адрес

Размер команды: 4 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=100, B=444):

```
0x64, 0xDE, 0x00, 0x00
```

Бинарная операция: `max()`

A	**B**
Биты 0—6	Биты 7—25
118	Адрес

Размер команды: 4 байт. Первый операнд: регистр-аккумулятор. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=118, B=409):

```
0xF6, 0xCC, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `max()` над двумя векторами длины 6. Результат записать в первый вектор.

Вариант №17

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `chmod`.
2. `rm`.
3. `find`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке yaml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
' Это однострочный комментарий
```

Многострочные комментарии:

```
{-
Это многострочный
комментарий
-}
```

Массивы:

```
[ значение, значение, значение, ... ]
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
def имя = значение;
```

Вычисление константы на этапе трансляции:

```
$имя$
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—2	Биты 3—10	Биты 11—15
4	Константа	Адрес

Размер команды: 3 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=4, B=237, C=30):

```
0x6C, 0xF7, 0x00
```

Чтение из памяти

A	B	C
Биты 0—2	Биты 3—7	Биты 8—17
5	Адрес	Адрес

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=5, B=27, C=110):

```
0xDD, 0x6E, 0x00
```

Запись в память

A	B	C
Биты 0—2	Биты 3—12	Биты 13—17
6	Адрес	Адрес

Размер команды: 3 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=6, B=220, C=15):

```
0xE6, 0xE6, 0x01
```

Унарная операция: `bitreverse()`

A	B	C
Биты 0—2	Биты 3—7	Биты 8—12

A	**B**	**C**
3	Адрес	Адрес

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=3, B=24, C=18):

```
0xC3, 0x12, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 4. Результат записать в новый вектор.

Вариант №18

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **json** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `uptime`.
2. `uname`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным хеш-значением.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Файл с заданным хеш-значением в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке yaml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
:: Это однострочный комментарий
```

Массивы:

```
{ значение, значение, значение, ... }
```

Имена:

```
[A-Z]+
```

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
имя = значение;
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—3	Биты 4—26
7	Константа

Размер команды: 4 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=7, B=219):

```
0xB7, 0x0D, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—3	Биты 4—9
11	Смещение

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Результат: новый элемент на стеке.

Тест (A=11, B=25):

```
0x9B, 0x01, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—3	Биты 4—27
1	Адрес

Размер команды: 4 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=1, B=249):

```
0x91, 0x0F, 0x00, 0x00
```

Унарная операция: `унарный минус`

A
Биты 0—3
10

Размер команды: 4 байт. Операнд: элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=10):

```
0x0A, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию унарный минус над вектором длины 7. Результат записать в новый вектор.

Вариант №19

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **ini** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **json** и содержит все действия во время последнего сеанса работы с эмулятором.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `find`.
2. `uptime`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся сообщения. Граф необходимо строить только для коммитов ранее заданной даты.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Дата коммитов в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в стандартный вывод.

Однострочные комментарии:

```
' Это однострочный комментарий
```

Массивы:

```
[ значение; значение; значение; ... ]
```

Имена:

```
[_a-zA-Z]+
```

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
(def имя значение)
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
!(имя 1 +)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `concat()`.
4. `sqrt()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—5	Биты 6—14
22	Константа

Размер команды: 6 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=22, B=62):

```
0x96, 0x0F, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—5	Биты 6—20
10	Смещение

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Результат: новый элемент на стеке.

Тест (A=10, B=893):

```
0x4A, 0xDF, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A
Биты 0—5
56

Размер команды: 6 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=56):

```
0x38, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—5	Биты 6—20	Биты 21—38
35	Смещение	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=35, B=104, C=646):

```
0x23, 0x1A, 0xC0, 0x50, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 5. Результат записать в исходный вектор.

Вариант №20

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `mkdir`.
2. `uname`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого содержатся сообщения.

Конфигурационный файл имеет формат **yaml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

```
{{!--
Это многострочный
комментарий
--}}
```

Массивы:

```
<< значение, значение, значение, ... >>
```

Имена:

```
[a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя := значение
```

Вычисление константы на этапе трансляции:

```
![имя]
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—3	Биты 4—29
2	Константа

Размер команды: 4 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=2, B=646):

```
0x62, 0x28, 0x00, 0x00
```

Чтение из памяти

A	B
Биты 0—3	Биты 4—10
6	Смещение

Размер команды: 2 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Результат: новый элемент на стеке.

Тест (A=6, B=11):

```
0xB6, 0x00
```

Запись в память

A
Биты 0—3
10

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=10):

```
0x0A
```

Бинарная операция: деление

A	B
Биты 0—3	Биты 4—14
4	Адрес

Размер команды: 2 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=4, B=881):

```
0x14, 0x37
```

Тестовая программа

Выполнить поэлементно операцию деление над вектором длины 5 и числом 10. Результат записать в новый вектор.

Вариант №21

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `head`.
2. `wc`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **json** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.

- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке toml** попадает в стандартный вывод.

Многострочные комментарии:

```
#=
Это многострочный
комментарий
=#
```

Словари:

```
$[
  имя : значение,
  имя : значение,
  имя : значение,
  ...
]
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
имя := значение
```

Вычисление константы на этапе трансляции:

```
@(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—5	Биты 6—21
13	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=13, B=937):

```
0x4D, 0xEA, 0x00
```

Чтение из памяти

A	**B**
Биты 0—5	Биты 6—10
41	Смещение

Размер команды: 2 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=41, B=15):

```
0xE9, 0x03
```

Запись в память

A	**B**
Биты 0—5	Биты 6—32
50	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=50, B=994):

```
0xB2, 0xF8, 0x00, 0x00, 0x00
```

Бинарная операция: `">="`

A	**B**
Биты 0—5	Биты 6—32
24	Адрес

Размер команды: 5 байт. Первый операнд: регистр-аккумулятор. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=24, B=792):

```
0x18, 0xC6, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `">="` над двумя векторами длины 4. Результат записать в новый вектор.

Вариант №22

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **yaml** и содержит:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tac`.
2. `rm`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого содержатся сообщения. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным именем.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- Файл с заданным именем в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке json** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
! Это однострочный комментарий
```

Массивы:

```
[ значение значение значение ... ]
```

Словари:

```
{
  имя -> значение.
  имя -> значение.
  имя -> значение.
  ...
}
```

Имена:

```
[_a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
const имя = значение;
```

Вычисление константы на этапе трансляции:

```
?(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—36
9	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=9, B=2, C=890):

```
0x29, 0x7A, 0x03, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—8	Биты 9—12	Биты 13—16
15	Смещение	Адрес	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле B). Результат: регистр по адресу, которым является поле C.

Тест (A=15, B=28, C=10, D=12):

```
0xCF, 0x95, 0x01, 0x00, 0x00
```

Запись в память

A	**B**	**C**	**D**
Биты 0—3	Биты 4—7	Биты 8—11	Биты 12—16
5	Адрес	Адрес	Смещение

Размер команды: 5 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле D).

Тест (A=5, B=12, C=12, D=19):

```
0xC5, 0x3C, 0x01, 0x00, 0x00
```

Бинарная операция: `"!="`

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—11
12	Адрес	Адрес

Размер команды: 5 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=12, B=5, C=6):

```
0x5C, 0x06, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"!="` над двумя векторами длины 7. Результат записать в первый вектор.

Вариант №23

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `clear`.
2. `find`.
3. `uniq`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу с изображением графа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке toml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

```
#( значение, значение, значение, ... )
```

Словари:

```
{
  имя : значение,
  имя : значение,
  имя : значение,
  ...
}
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.

- Строки.
- Массивы.
- Словари.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
имя := значение;
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
|имя 1 +|
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `sort()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—24	Биты 25—30
11	Константа	Адрес

Размер команды: 4 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=11, B=149, C=39):

```
0x5B, 0x09, 0x00, 0x4E
```

Чтение из памяти

A	**B**	**C**
Биты 0—3	Биты 4—9	Биты 10—15
0	Адрес	Адрес

Размер команды: 2 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=0, B=21, C=10):

```
0x50, 0x29
```

Запись в память

A	**B**	**C**	**D**
Биты 0—3	Биты 4—9	Биты 10—15	Биты 16—27
15	Адрес	Адрес	Смещение

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле D).

Тест (A=15, B=28, C=10, D=777):

```
0xCF, 0x29, 0x09, 0x03
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—9	Биты 10—15	Биты 16—21
13	Адрес	Адрес	Адрес

Размер команды: 3 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: регистр по адресу, которым является поле D. Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C.

Тест (A=13, B=18, C=3, D=36):

```
0x2D, 0x0D, 0x24
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг вправо` над двумя векторами длины 5. Результат записать в новый вектор.

Вариант №24

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **ini** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `uniq`.
2. `du`.
3. `cal`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Построить граф зависимостей для коммитов, в узлах которого содержатся дата, время и автор коммита. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным именем.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.
- Файл с заданным именем в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке json** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Массивы:

```
#( значение значение значение ... )
```

Имена:

```
[a-zA-Z][a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
set имя = значение
```

Вычисление константного выражения на этапе трансляции (инфиксная форма), пример:

`.[имя + 1].`

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `sqrt()`.
4. `mod()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—3	Биты 4—24	Биты 25—56
11	Константа	Адрес

Размер команды: 8 байт. Операнд: поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=11, B=149, C=511):

```
0x5B, 0x09, 0x00, 0xFE, 0x03, 0x00, 0x00, 0x00
```

Чтение из памяти

A	B	C
Биты 0—3	Биты 4—35	Биты 36—67
0	Адрес	Адрес

Размер команды: 9 байт. Операнд: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=0, B=35, C=456):

```
0x30, 0x02, 0x00, 0x00, 0x80, 0x1C, 0x00, 0x00, 0x00
```

Запись в память

A	B	C	D
Биты 0—3	Биты 4—35	Биты 36—67	Биты 68—79
15	Адрес	Адрес	Смещение

Размер команды: 10 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле B) и смещения (поле D).

Тест (A=15, B=777, C=882, D=873):

```
0x9F, 0x30, 0x00, 0x00, 0x20, 0x37, 0x00, 0x00, 0x90, 0x36
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	B	C	D

A	**B**	**C**	**D**
Биты 0—3	Биты 4—35	Биты 36—67	Биты 68—99
13	Адрес	Адрес	Адрес

Размер команды: 13 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является поле D. Результат: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле C.

Тест (A=13, B=444, C=568, D=561):

```
0xCD, 0x1B, 0x00, 0x00, 0x80, 0x23, 0x00, 0x00, 0x10, 0x23, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовый циклический сдвиг вправо над двумя векторами длины 8. Результат записать в новый вектор.

Вариант №25

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **json** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tree`.
2. `clear`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке xml** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Словари:

```
@{
  имя = значение;
  имя = значение;
  имя = значение;
  ...
}
```

Имена:

```
[a-z][a-z0-9_]*
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
значение -> имя;
```

Вычисление константы на этапе трансляции:

[имя]

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—36
9	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=9, B=2, C=890):

```
0x29, 0x7A, 0x03, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—8	Биты 9—12	Биты 13—16
15	Смещение	Адрес	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле B). Результат: регистр по адресу, которым является поле C.

Тест (A=15, B=28, C=10, D=12):

```
0xCF, 0x95, 0x01, 0x00, 0x00
```

Запись в память

A	**B**	**C**	**D**
Биты 0—3	Биты 4—7	Биты 8—11	Биты 12—16
5	Адрес	Адрес	Смещение

Размер команды: 5 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле D).

Тест (A=5, B=12, C=12, D=19):

```
0xC5, 0x3C, 0x01, 0x00, 0x00
```

Бинарная операция: `"!="`

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—11
12	Адрес	Адрес

Размер команды: 5 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=12, B=5, C=6):

```
0x5C, 0x06, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"!="` над двумя векторами длины 7. Результат записать в первый вектор.

Вариант №26

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `touch`.
2. `date`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка Java (Maven)**. Для описания графа зависимостей используется представление **Graphviz**.

Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Конфигурационный файл имеет формат **xml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу с изображением графа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
" Это однострочный комментарий
```

Словари:

```
{
  имя = значение,
  имя = значение,
  имя = значение,
  ...
}
```

Имена:

```
[A-Z]+
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
set имя = значение
```

Вычисление константного выражения на этапе трансляции (инфиксная форма), пример:

```
?(имя + 1)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `pow()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	B	C
Биты 0—7	Биты 8—11	Биты 12—30
217	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=217, B=5, C=35):

```
0xD9, 0x35, 0x02, 0x00
```

Чтение из памяти

A	B	C
Биты 0—7	Биты 8—11	Биты 12—22
178	Адрес	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=178, B=7, C=337):

```
0xB2, 0x17, 0x15, 0x00
```

Запись в память

A	B	C	D
Биты 0—7	Биты 8—23	Биты 24—27	Биты 28—31
110	Смещение	Адрес	Адрес

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле D. Результат: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле B).

Тест (A=110, B=882, C=4, D=0):

```
0x6E, 0x72, 0x03, 0x04
```

Бинарная операция: `взятие остатка`

A	**B**	**C**
Биты 0—7	Биты 8—11	Биты 12—15
197	Адрес	Адрес

Размер команды: 4 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=197, B=6, C=13):

```
0xC5, 0xD6, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `взятие остатка` над двумя векторами длины 5. Результат записать во второй вектор.

Вариант №27

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `cat`.
2. `chmod`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке xml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
REM Это однострочный комментарий
```

Массивы:

```
[ значение, значение, значение, ... ]
```

Словари:

```
struct {
  имя = значение,
  имя = значение,
  имя = значение,
  ...
}
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.
- Строки.
- Массивы.
- Словари.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
(def имя значение);
```

Вычисление константного выражения на этапе трансляции (префиксная форма), пример:

```
^(+ имя 1)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `sqrt()`.
3. `print()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—5	Биты 6—34
10	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=10, B=882):

```
0x8A, 0xDC, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—5	Биты 6—22
2	Адрес

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=2, B=392):

```
0x02, 0x62, 0x00
```

Запись в память

A	**B**
Биты 0—5	Биты 6—22
27	Адрес

Размер команды: 3 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=27, B=121):

```
0x5B, 0x1E, 0x00
```

Унарная операция: `sqrt()`

A
Биты 0—5
44

Размер команды: 1 байт. Операнд: ячейка памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=44):

```
0x2C
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 5. Результат записать в новый вектор.

Вариант №28

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `pwd`.
2. `history`.
3. `uptime`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить для ветки с заданным именем.

Конфигурационный файл имеет формат **xml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- Имя ветки в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке json** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
" Это однострочный комментарий
```

Многострочные комментарии:

```
%{
Это многострочный
комментарий
%}
```

Массивы:

```
<< значение, значение, значение, ... >>
```

Словари:

```
dict(
  имя = значение,
  имя = значение,
  имя = значение,
  ...
)
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Массивы.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

Вычисление константы на этапе трансляции:

```
?{имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из

командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **json**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—3	Биты 4—15
11	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=11, B=171):

```
0xBB, 0x0A, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—3	Биты 4—19
9	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=9, B=130):

```
0x29, 0x08, 0x00, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—3	Биты 4—19

A	B
8	Адрес

Размер команды: 5 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=8, B=427):

```
0xB8, 0x1A, 0x00, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C
Биты 0—3	Биты 4—19	Биты 20—35
6	Адрес	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=6, B=445, C=293):

```
0xD6, 0x1B, 0x50, 0x12, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 6. Результат записать в новый вектор.

Вариант №29

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указаны дата и время.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `rm`.
2. `uptime`.
3. `who`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке toml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
-- Это однострочный комментарий
```

Многострочные комментарии:

```
=begin
Это многострочный
комментарий
=end
```

Словари:

```
{
  имя : значение,
  имя : значение,
  имя : значение,
  ...
}
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
имя is значение
```

Вычисление константного выражения на этапе трансляции (префиксная форма), пример:

```
$+ имя 1$
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `pow()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является

бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—5	Биты 6—18
37	Константа

Размер команды: 4 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=37, B=393):

```
0x65, 0x62, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—5	Биты 6—24
10	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=10, B=426):

```
0x8A, 0x6A, 0x00, 0x00
```

Запись в память

A	**B**

A	B
Биты 0—5	Биты 6—24
38	Адрес

Размер команды: 4 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=38, B=809):

```
0x66, 0xCA, 0x00, 0x00
```

Бинарная операция: `побитовый арифметический сдвиг вправо`

A	B
Биты 0—5	Биты 6—24
21	Адрес

Размер команды: 4 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=21, B=94):

```
0x95, 0x17, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовый арифметический сдвиг вправо` над двумя векторами длины 8. Результат записать во второй вектор.

Вариант №30

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **xml** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указаны дата и время.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `cal`.
2. `uniq`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Mermaid**.

Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу с изображением графа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
|| Это однострочный комментарий
```

Многострочные комментарии:

```
{-
Это многострочный
комментарий
-}
```

Массивы:

```
list( значение, значение, значение, ... )
```

Имена:

```
[a-zA-Z]+
```

Значения:

- Числа.
- Строки.

- Массивы.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
имя <- значение
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
?[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `mod()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—7	Биты 8—31
136	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=136, B=455):

```
0x88, 0xC7, 0x01, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—7	Биты 8—33
51	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=51, B=931):

```
0x33, 0xA3, 0x03, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—7	Биты 8—33
103	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=103, B=339):

```
0x67, 0x53, 0x01, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—7	Биты 8—33
148	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=148, B=127):

```
0x94, 0x7F, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 6. Результат записать в новый вектор.

Вариант №31

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `echo`.
2. `whoami`.
3. `uniq`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным именем.

Конфигурационный файл имеет формат **csv** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- Файл с заданным именем в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке toml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
REM Это однострочный комментарий
```

Массивы:

```
list( значение, значение, значение, ... )
```

Имена:

```
[a-zA-Z][a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя = значение
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
.[имя 1 +].
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `len()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	B	C
Биты 0—2	Биты 3—9	Биты 10—20
1	Адрес	Константа

Размер команды: 8 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=1, B=33, C=999):

```
0x09, 0x9D, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	B	C
Биты 0—2	Биты 3—9	Биты 10—16
6	Адрес	Адрес

Размер команды: 8 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=6, B=42, C=95):

```
0x56, 0x7D, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	B	C
Биты 0—2	Биты 3—9	Биты 10—16
7	Адрес	Адрес

Размер команды: 8 байт. Операнд: регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C.

Тест (A=7, B=57, C=69):

```
0xCF, 0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—33
5	Адрес	Адрес

Размер команды: 8 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=5, B=21, C=261):

```
0xAD, 0x14, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 5. Результат записать в исходный вектор.

Вариант №32

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **csv** и содержит:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `mv`.
2. `chmod`.
3. `pwd`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **PlantUML**.

Визуализатор должен выводить результат на экран в виде графического изображения графа.

Конфигурационный файл имеет формат **toml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Словари:

```
table(
  имя => значение,
  имя => значение,
  имя => значение,
  ...
)
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
const имя = значение;
```

Вычисление константы на этапе трансляции:

```
!{имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—2	Биты 3—13
1	Константа

Размер команды: 7 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=1, B=236):

```
0x61, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A
Биты 0—2
6

Размер команды: 7 байт. Операнд: ячейка памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=6):

```
0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A
Биты 0—2
7

Размер команды: 7 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=7):

```
0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—2	Биты 3—26
5	Адрес

Размер команды: 7 байт. Операнд: ячейка памяти по адресу, которым является элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=5, B=468):

```
0xA5, 0x0E, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 6. Результат записать в исходный вектор.

Вариант №33

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `chown`.
2. `cp`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить для ветки с заданным именем.

Конфигурационный файл имеет формат **xml** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Имя ветки в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке toml** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
' Это однострочный комментарий
```

Массивы:

```
{ значение. значение. значение. ... }
```

Словари:

```
dict(
  имя = значение,
  имя = значение,
  имя = значение,
  ...
)
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
(def имя значение);
```

Вычисление константы на этапе трансляции:

```
@[имя]
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—7	Биты 8—31
136	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=136, B=455):

```
0x88, 0xC7, 0x01, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—7	Биты 8—33
51	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=51, B=931):

```
0x33, 0xA3, 0x03, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—7	Биты 8—33
103	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=103, B=339):

```
0x67, 0x53, 0x01, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—7	Биты 8—33
148	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=148, B=127):

```
0x94, 0x7F, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 6. Результат записать в новый вектор.

Вариант №34

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **toml** и содержит:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **xml** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `whoami`.
2. `uname`.
3. `tac`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **json** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке xml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
! Это однострочный комментарий
```

Массивы:

```
list( значение, значение, значение, ... )
```

Словари:

```
{
  имя = значение
  имя = значение
  имя = значение
  ...
}
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Массивы.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
имя <- значение
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—20
1	Адрес	Константа

Размер команды: 8 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=1, B=33, C=999):

```
0x09, 0x9D, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—16
6	Адрес	Адрес

Размер команды: 8 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=6, B=42, C=95):

```
0x56, 0x7D, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—16
7	Адрес	Адрес

Размер команды: 8 байт. Операнд: регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является регистр по адресу, которым является поле C.

Тест (A=7, B=57, C=69):

```
0xCF, 0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—33
5	Адрес	Адрес

Размер команды: 8 байт. Операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=5, B=21, C=261):

```
0xAD, 0x14, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 5. Результат записать в исходный вектор.

Вариант №35

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- Путь к стартовому скрипту.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tac`.
2. `history`.
3. `pwd`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся хеш-значения. Граф необходимо строить только для коммитов позже заданной даты.

Конфигурационный файл имеет формат **json** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Дата коммитов в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке json** попадает в стандартный вывод.

Однострочные комментарии:

```
" Это однострочный комментарий
```

Словари:

```
dict(
  имя = значение,
  имя = значение,
  имя = значение,
  ...
)
```

Имена:

```
[a-z]+
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

Вычисление константного выражения на этапе трансляции (постфиксная форма), пример:

```
.{имя 1 +}.
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `max()`.
6. `chr()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является

бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—26	Биты 27—37
1	Адрес	Константа

Размер команды: 10 байт. Операнд: поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=1, B=236, C=999):

```
0x61, 0x07, 0x00, 0x38, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**
Биты 0—2	Биты 3—26	Биты 27—50
6	Адрес	Адрес

Размер команды: 10 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=6, B=166, C=468):

```
0x36, 0x05, 0x00, 0xA0, 0x0E, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	B	C
Биты 0—2	Биты 3—26	Биты 27—50
7	Адрес	Адрес

Размер команды: 10 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле C.

Тест (A=7, B=663, C=470):

```
0xBF, 0x14, 0x00, 0xB0, 0x0E, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	B	C
Биты 0—2	Биты 3—26	Биты 27—50
5	Адрес	Адрес

Размер команды: 10 байт. Операнд: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=5, B=439, C=261):

```
0xBD, 0x0D, 0x00, 0x28, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 5. Результат записать в исходный вектор.

Вариант №36

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `wc`.
2. `mkdir`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка Python (pip)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **ini** и содержит:

- Путь к программе для визуализации графов.

- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
REM Это однострочный комментарий
```

Словари:

```
table(
  имя => значение,
  имя => значение,
  имя => значение,
  ...
)
```

Имена:

```
[_A-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
имя is значение
```

Вычисление константного выражения на этапе трансляции (инфиксная форма), пример:

```
.[имя + 1].
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `sqrt()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **csv**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	B	C
Биты 0—5	Биты 6—17	Биты 18—23
52	Константа	Адрес

Размер команды: 3 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=52, B=716, C=41):

```
0x34, 0xB3, 0xA4
```

Чтение из памяти

A	B	C
Биты 0—5	Биты 6—16	Биты 17—22
12	Адрес	Адрес

Размер команды: 3 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=12, B=813, C=13):

```
0x4C, 0xCB, 0x1A
```

Запись в память

A	B	C
Биты 0—5	Биты 6—16	Биты 17—22
0	Адрес	Адрес

Размер команды: 3 байт. Операнд: регистр по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=0, B=158, C=33):

```
0x80, 0x27, 0x42
```

Бинарная операция: `"<="`

A	B	C

A	**B**	**C**
Биты 0—5	Биты 6—11	Биты 12—17
21	Адрес	Адрес

Размер команды: 3 байт. Первый операнд: регистр по адресу, которым является поле C. Второй операнд: ячейка памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=21, B=37, C=58):

```
0x55, 0xA9, 0x03
```

Тестовая программа

Выполнить поэлементно операцию "<=" над вектором длины 8 и числом 136. Результат записать в новый вектор.

Вариант №37

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `tail`.
2. `du`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **языка Python (pip)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Максимальная глубина анализа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **языке toml** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Словари:

```
{
  имя = значение
  имя = значение
  имя = значение
  ...
}
```

Имена:

```
[A-Z]+
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
const имя = значение
```

Вычисление константного выражения на этапе трансляции (инфиксная форма), пример:

```
.(имя + 1).
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `ord()`.
4. `min()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	B
Биты 0—7	Биты 8—37
101	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=101, B=853):

```
0x65, 0x55, 0x03, 0x00, 0x00
```

Чтение из памяти

A	B
Биты 0—7	Биты 8—31
236	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=236, B=830):

```
0xEC, 0x3E, 0x03, 0x00
```

Запись в память

A
Биты 0—7
215

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=215):

```
0xD7
```

Унарная операция: `sgn()`

A
Биты 0—7

A
227

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=227):

```
0xE3
```

Тестовая программа

Выполнить поэлементно операцию `sgn()` над вектором длины 8. Результат записать в исходный вектор.

Вариант №38

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **json** и содержит:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `mv`.
2. `whoami`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 3 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

Конфигурационный файл имеет формат **csv** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

```
{ значение, значение, значение, ... }
```

Имена:

```
[_a-z]+
```

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя = значение;
```

Вычисление константного выражения на этапе трансляции (инфиксная форма), пример:

```
@(имя + 1)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `sqrt()`.
4. `mod()`.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 3 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—6	Биты 7—34
74	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=74, B=574):

```
0x4A, 0x1F, 0x01, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—6	Биты 7—37
0	Адрес

Размер команды: 5 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=0, B=86):

```
0x00, 0x2B, 0x00, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—6	Биты 7—37
100	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=100, B=570):

```
0x64, 0x1D, 0x01, 0x00, 0x00
```

Бинарная операция: сложение

A	**B**
Биты 0—6	Биты 7—37
114	Адрес

Размер команды: 5 байт. Первый операнд: регистр-аккумулятор. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=114, B=931):

```
0xF2, 0xD1, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию сложение над двумя векторами длины 4. Результат записать в новый вектор.

Вариант №39

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `cp`.
2. `mkdir`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.

- Путь к файлу с изображением графа зависимостей.
- Максимальная глубина анализа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке yaml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
C Это однострочный комментарий
```

Словари:

```
begin
  имя := значение;
  имя := значение;
  имя := значение;
  ...
end
```

Имена:

```
[a-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя = значение;
```

Вычисление константы на этапе трансляции:

```
$(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **xml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**
Биты 0—7	Биты 8—37
101	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=101, B=853):

```
0x65, 0x55, 0x03, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—7	Биты 8—31
236	Адрес

Размер команды: 4 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=236, B=830):

```
0xEC, 0x3E, 0x03, 0x00
```

Запись в память

A
Биты 0—7
215

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=215):

```
0xD7
```

Унарная операция: `sgn()`

A
Биты 0—7
227

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=227):

```
0xE3
```

Тестовая программа

Выполнить поэлементно операцию `sgn()` над вектором длины 8. Результат записать в исходный вектор.

Вариант №40

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **xml** и содержит:

- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к стартовому скрипту.

Стартовый скрипт служит для начального выполнения заданного списка команд из файла.

Необходимо поддержать в эмуляторе команды `ls`, `cd` и `exit`, а также следующие команды:

1. `history`.
2. `mkdir`.

Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние программы или библиотеки для получения зависимостей использовать нельзя.

Зависимости определяются для файла-пакета **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

Разработать инструмент командной строки для учебного конфигурационного языка, синтаксис которого приведен далее. Этот инструмент преобразует текст из входного формата в выходной. Синтаксические ошибки выявляются с выдачей сообщений.

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке toml** попадает в стандартный вывод.

Многострочные комментарии:

```
{{!
Это многострочный
комментарий
}}
```

Словари:

```
begin
  имя := значение;
  имя := значение;
  имя := значение;
  ...
end
```

Имена:

```
[a-z][a-z0-9_]*
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
значение -> имя;
```

Вычисление константы на этапе трансляции:

```
^(имя)
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 2 примера описания конфигураций из разных предметных областей.

Задание №4

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ). Система команд УВМ представлена далее.

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

Интерпретатор принимает на вход бинарный файл, выполняет команды УВМ и сохраняет в файле-результате значения из диапазона памяти УВМ. Диапазон также указывается из командной строки.

Форматом для файла-лога и файла-результата является **yaml**.

Необходимо реализовать приведенные тесты для всех команд, а также написать и отладить тестовую программу.

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—6	Биты 7—12	Биты 13—26
113	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=113, B=63, C=982):

```
0xF1, 0xDF, 0x7A, 0x00, 0x00, 0x00
```

Чтение из памяти

A	B	C	D
Биты 0—6	Биты 7—12	Биты 13—25	Биты 26—31
16	Адрес	Смещение	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле C). Результат: регистр по адресу, которым является поле D.

Тест (A=16, B=63, C=419, D=4):

```
0x90, 0x7F, 0x34, 0x10, 0x00, 0x00
```

Запись в память

A	B	C
Биты 0—6	Биты 7—12	Биты 13—44
48	Адрес	Адрес

Размер команды: 6 байт. Операнд: регистр по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=48, B=47, C=110):

```
0xB0, 0xD7, 0x0D, 0x00, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	B	C

A	**B**	**C**
Биты 0—6	Биты 7—12	Биты 13—18
46	Адрес	Адрес

Размер команды: 6 байт. Операнд: регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=46, B=20, C=3):

```
0x2E, 0x6A, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 8. Результат записать в исходный вектор.