Конфигурационное управление

Сборник домашних заданий

Группа 20

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 **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.

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

1. `pwd`.
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 файла-пакета **ОС Alpine Linux (apk)**. 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 **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке yaml** попадает в стандартный вывод.

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
const 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. `concat()`.

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—30
8	Константа

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

Тест (A=8, B=600):

```
0x88, 0x25, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—3	Биты 4—10
10	Смещение

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

Тест (A=10, B=18):

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

Запись в память

A	**B**
Биты 0—3	Биты 4—14
9	Адрес

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

Тест (A=9, B=85):

```
0x59, 0x05, 0x00, 0x00
```

Унарная операция: `bswap()`

A
Биты 0—3
2

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

Тест (A=2):

```
0x02, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bswap()` над вектором длины 6. Результат записать в новый 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 **zip**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **ini** 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. `clear`.
3. `cat`.

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 **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.

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:

```
NB. Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
значение -> имя
```

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. `len()`.
3. `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—7	Биты 8—13	Биты 14—44
90	Адрес	Константа

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

Тест (A=90, B=18, C=207):

```
0x5A, 0xD2, 0x33, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—7	Биты 8—13	Биты 14—19	Биты 20—25
1	Адрес	Адрес	Смещение

Размер команды: 8 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле D). Результат: регистр по адресу, которым является поле B.

Тест (A=1, B=21, C=13, D=42):

```
0x01, 0x55, 0xA3, 0x02, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—19
62	Адрес	Адрес

Размер команды: 8 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=62, B=17, C=20):

```
0x3E, 0x11, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**	**D**
Биты 0—7	Биты 8—13	Биты 14—19	Биты 20—25
137	Адрес	Адрес	Смещение

Размер команды: 8 байт. Операнд: ячейка памяти по адресу, которым является 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=137, B=4, C=37, D=2):

```
0x89, 0x44, 0x29, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. 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 **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 **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. `chmod`.
2. `rev`.
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 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 **yaml** 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 стандартный вывод.

Словари:

```
[
  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. `max()`.
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 **json**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—38	Биты 39—69
90	Адрес	Константа

Размер команды: 14 байт. Операнд: поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=90, B=947, C=207):

```
0x5A, 0xB3, 0x03, 0x00, 0x80, 0x67, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—7	Биты 8—38	Биты 39—69	Биты 70—75
1	Адрес	Адрес	Смещение

Размер команды: 14 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле D). Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=1, B=822, C=918, D=42):

```
0x01, 0x36, 0x03, 0x00, 0x00, 0xCB, 0x01, 0x00, 0x80, 0x0A, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—7	Биты 8—38	Биты 39—69
62	Адрес	Адрес

Размер команды: 14 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле B.

Тест (A=62, B=135, C=464):

```
0x3E, 0x87, 0x00, 0x00, 0x00, 0xE8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**	**D**
Биты 0—7	Биты 8—38	Биты 39—69	Биты 70—75
137	Адрес	Адрес	Смещение

Размер команды: 14 байт. Операнд: ячейка памяти по адресу, которым является 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=137, B=920, C=334, D=2):

```
0x89, 0x98, 0x03, 0x00, 0x00, 0xA7, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. 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 **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 **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. `rm`.
2. `tree`.

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 номера коммитов в хронологическом порядке. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным хеш-значением.

Конфигурационный файл имеет формат **ini** 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 **языке xml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** 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:

```
let 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. Деление.
5. `pow()`.
6. `abs()`.

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**	**C**
Биты 0—6	Биты 7—38	Биты 39—50
94	Адрес	Константа

Размер команды: 7 байт. Операнд: поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=94, B=45, C=591):

```
0xDE, 0x16, 0x00, 0x00, 0x80, 0x27, 0x01
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—38	Биты 39—50	Биты 51—82
9	Адрес	Смещение	Адрес

Размер команды: 11 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле B) и смещения (поле C). Результат: ячейка памяти по адресу, которым является поле D.

Тест (A=9, B=255, C=225, D=956):

```
0x89, 0x7F, 0x00, 0x00, 0x80, 0x70, 0xE0, 0x1D, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**

A	**B**	**C**
Биты 0—6	Биты 7—38	Биты 39—70
88	Адрес	Адрес

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

Тест (A=88, B=477, C=917):

```
0xD8, 0xEE, 0x00, 0x00, 0x80, 0xCA, 0x01, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**	**D**
Биты 0—6	Биты 7—38	Биты 39—70	Биты 71—82
34	Адрес	Адрес	Смещение

Размер команды: 11 байт. Операнд: ячейка памяти по адресу, которым является 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=34, B=712, C=525, D=41):

```
0x22, 0x64, 0x01, 0x00, 0x80, 0x06, 0x01, 0x00, 0x80, 0x14, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 4. Результат записать в новый 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 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. `tree`.
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 **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. 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 находятся списки файлов и папок.

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

- 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 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:

```
/+
Это многострочный
комментарий
+/
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

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. `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 **json**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—38	Биты 39—69
90	Адрес	Константа

Размер команды: 14 байт. Операнд: поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=90, B=947, C=207):

```
0x5A, 0xB3, 0x03, 0x00, 0x80, 0x67, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—7	Биты 8—38	Биты 39—69	Биты 70—75
1	Адрес	Адрес	Смещение

Размер команды: 14 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле D). Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=1, B=822, C=918, D=42):

```
0x01, 0x36, 0x03, 0x00, 0x00, 0xCB, 0x01, 0x00, 0x80, 0x0A, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—7	Биты 8—38	Биты 39—69
62	Адрес	Адрес

Размер команды: 14 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: ячейка памяти по адресу, которым является ячейка памяти по адресу, которым является поле B.

Тест (A=62, B=135, C=464):

```
0x3E, 0x87, 0x00, 0x00, 0x00, 0xE8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**	**D**
Биты 0—7	Биты 8—38	Биты 39—69	Биты 70—75
137	Адрес	Адрес	Смещение

Размер команды: 14 байт. Операнд: ячейка памяти по адресу, которым является 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=137, B=920, C=334, D=2):

```
0x89, 0x98, 0x03, 0x00, 0x00, 0xA7, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. Результат записать в новый 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 **CLI**.

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 **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. `cp`.
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 **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 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.
- 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:

```
\ Это однострочный комментарий
```

Словари:

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

Имена:

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

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 **xml**.

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

Загрузка константы

A	B	C
Биты 0—5	Биты 6—16	Биты 17—21
34	Константа	Адрес

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

Тест (A=34, B=32, C=7):

```
0x22, 0x08, 0x0E
```

Чтение из памяти

A	B	C
Биты 0—5	Биты 6—10	Биты 11—15
23	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=23, B=1, C=6):

```
0x57, 0x30
```

Запись в память

A	B	C
Биты 0—5	Биты 6—10	Биты 11—34
4	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=4, B=29, C=477):

```
0x44, 0xEF, 0x0E, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C

A	**B**	**C**
Биты 0—5	Биты 6—10	Биты 11—34
57	Адрес	Адрес

Размер команды: 5 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=57, B=26, C=399):

```
0xB9, 0x7E, 0x0C, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 8. Результат записать в новый 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 **GUI**.

Конфигурационный файл имеет формат **ini** 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. `mkdir`.
2. `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 **языка Java (Maven)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Конфигурационный файл имеет формат **json** 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 **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, ... )
```

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 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**

A	B	C
Биты 0—5	Биты 6—9	Биты 10—26
44	Адрес	Константа

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

Тест (A=44, B=7, C=928):

```
0xEC, 0x81, 0x0E, 0x00
```

Чтение из памяти

A	B	C	D
Биты 0—5	Биты 6—9	Биты 10—14	Биты 15—18
14	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=14, B=7, C=28, D=2):

```
0xCE, 0x71, 0x01, 0x00
```

Запись в память

A	B	C
Биты 0—5	Биты 6—9	Биты 10—13
2	Адрес	Адрес

Размер команды: 4 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=2, B=7, C=2):

```
0xC2, 0x09, 0x00, 0x00
```

Бинарная операция: сложение

A	B	C	D

A	**B**	**C**	**D**
Биты 0—5	Биты 6—9	Биты 10—13	Биты 14—28
29	Адрес	Адрес	Адрес

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

Тест (A=29, B=4, C=4, D=282):

```
0x1D, 0x91, 0x46, 0x00
```

Тестовая программа

Выполнить поэлементно операцию сложение над вектором длины 4 и числом 12. 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**.

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. `rmdir`.
2. `echo`.
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 **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 находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить только для коммитов ранее заданной даты.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- 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 yaml** 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 ... ]
```

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

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 **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—9	Биты 10—26

A	B	C
44	Адрес	Константа

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

Тест (A=44, B=7, C=928):

```
0xEC, 0x81, 0x0E, 0x00
```

Чтение из памяти

A	B	C	D
Биты 0—5	Биты 6—9	Биты 10—14	Биты 15—18
14	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=14, B=7, C=28, D=2):

```
0xCE, 0x71, 0x01, 0x00
```

Запись в память

A	B	C
Биты 0—5	Биты 6—9	Биты 10—13
2	Адрес	Адрес

Размер команды: 4 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=2, B=7, C=2):

```
0xC2, 0x09, 0x00, 0x00
```

Бинарная операция: сложение

A	B	C	D
Биты 0—5	Биты 6—9	Биты 10—13	Биты 14—28

A	**B**	**C**	**D**
29	Адрес	Адрес	Адрес

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

Тест (A=29, B=4, C=4, D=282):

```
0x1D, 0x91, 0x46, 0x00
```

Тестовая программа

Выполнить поэлементно операцию сложение над вектором длины 4 и числом 12. Результат записать в новый 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**.

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. `tail`.
2. `clear`.

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 JavaScript (npm)**. Для описания графа зависимостей используется представление **PlantUML**. 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 стандартного ввода. Выходной текст на **языке toml** попадает в стандартный вывод.

Однострочные комментарии:

```
* Это однострочный комментарий
```

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:

```
(def 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. `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 **json**.

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

Загрузка константы

A	**B**
Биты 0—7	Биты 8—38
90	Константа

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

Тест (A=90, B=947):

```
0x5A, 0xB3, 0x03, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—7	Биты 8—13
1	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=1, B=58):

```
0x01, 0x3A, 0x00, 0x00, 0x00, 0x00
```

Запись в память

A
Биты 0—7
62

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

Тест (A=62):

```
0x3E, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**
Биты 0—7	Биты 8—13
137	Смещение

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

Тест (A=137, B=42):

```
0x89, 0x2A, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 8. 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**.

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

- 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. `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 **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.

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

- 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 стандартный вывод.

Однострочные комментарии:

```
" Это однострочный комментарий
```

Словари:

```
{
  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 (постфиксная форма), пример:

```
[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `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**	**C**

A	B	C
Биты 0—4	Биты 5—18	Биты 19—47
19	Адрес	Константа

Размер команды: 7 байт. Операнд: поле C. Результат: ячейка памяти по адресу, которым является поле B.

Тест (A=19, B=820, C=213):

```
0x93, 0x66, 0xA8, 0x06, 0x00, 0x00, 0x00
```

Чтение из памяти

A	B	C
Биты 0—4	Биты 5—18	Биты 19—32
22	Адрес	Адрес

Размер команды: 7 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=22, B=576, C=52):

```
0x16, 0x48, 0xA0, 0x01, 0x00, 0x00, 0x00
```

Запись в память

A	B	C
Биты 0—4	Биты 5—18	Биты 19—32
24	Адрес	Адрес

Размер команды: 7 байт. Операнд: ячейка памяти по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=24, B=742, C=327):

```
0xD8, 0x5C, 0x38, 0x0A, 0x00, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C	D

A	**B**	**C**	**D**
Биты 0—4	Биты 5—18	Биты 19—24	Биты 25—38
23	Адрес	Смещение	Адрес

Размер команды: 7 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле B) и смещения (поле C). Результат: ячейка памяти по адресу, которым является поле D.

Тест (A=23, B=516, C=9, D=49):

```
0x97, 0x40, 0x48, 0x62, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 5. 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 **tar**. Эмулятор должен работать в режиме **CLI**.

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

- 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. `head`.
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 **Mermaid**. 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 хеш-значением.

Конфигурационный файл имеет формат **ini** 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 стандартного ввода. Выходной текст на **языке toml** попадает в стандартный вывод.

Многострочные комментарии:

```
%{
Это многострочный
комментарий
%}
```

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:

```
значение -> имя
```

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. `pow()`.
4. `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 **json**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—34	Биты 35—39
213	Константа	Адрес

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

Тест (A=213, B=809, C=19):

```
0xD5, 0x29, 0x03, 0x00, 0x98
```

Чтение из памяти

A	**B**	**C**
Биты 0—7	Биты 8—21	Биты 22—26
207	Адрес	Адрес

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

Тест (A=207, B=256, C=12):

```
0xCF, 0x00, 0x01, 0x03
```

Запись в память

A	**B**	**C**
Биты 0—7	Биты 8—12	Биты 13—17
135	Адрес	Адрес

Размер команды: 3 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=135, B=21, C=19):

```
0x87, 0x75, 0x02
```

Бинарная операция: сложение

A	**B**	**C**
Биты 0—7	Биты 8—12	Биты 13—26
245	Адрес	Адрес

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

Тест (A=245, B=24, C=827):

```
0xF5, 0x78, 0x67, 0x00
```

Тестовая программа

Выполнить поэлементно операцию сложение над вектором длины 4 и числом 137. 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 **CLI**.

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

- Путь к архиву виртуальной файловой системы.

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

1. `cal`.
2. `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 **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 **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.

Массивы:

```
({ значение, значение, значение, ... })
```

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

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. Умножение.
4. `ord()`.

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—4	Биты 5—23
1	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=1, B=284):

```
0x81, 0x23, 0x00
```

Чтение из памяти

A	**B**
Биты 0—4	Биты 5—18
10	Смещение

Размер команды: 3 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=198):

```
0xCA, 0x18, 0x00
```

Запись в память

A	**B**
Биты 0—4	Биты 5—30
22	Адрес

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

Тест (A=22, B=831):

```
0xF6, 0x67, 0x00, 0x00
```

Бинарная операция: умножение

A	**B**
Биты 0—4	Биты 5—18
30	Смещение

Размер команды: 3 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: элемент, снятый с вершины стека. Результат: ячейка памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B).

Тест (A=30, B=64):

```
0x1E, 0x08, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 8 и числом 170. Результат записать в исходный 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**.

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.

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. `rev`.
2. `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 **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

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

Конфигурационный файл имеет формат **xml** 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 стандартного ввода. Выходной текст на **языке json** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
C Это однострочный комментарий
```

Многострочные комментарии:

```
<#
Это многострочный
комментарий
#>
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
значение -> имя;
```

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. `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**	**C**
Биты 0—4	Биты 5—23	Биты 24—49
26	Константа	Адрес

Размер команды: 7 байт. Операнд: поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=26, B=214, C=831):

```
0xDA, 0x1A, 0x00, 0x3F, 0x03, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—18	Биты 19—44	Биты 45—70
4	Смещение	Адрес	Адрес

Размер команды: 9 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле B). Результат: ячейка памяти по адресу, которым является поле D.

Тест (A=4, B=64, C=84, D=752):

```
0x04, 0x08, 0xA0, 0x02, 0x00, 0x00, 0x5E, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—4	Биты 5—30	Биты 31—56
27	Адрес	Адрес

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

Тест (A=27, B=39, C=700):

```
0xFB, 0x04, 0x00, 0x00, 0x5E, 0x01, 0x00, 0x00
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	**B**	**C**	**D**	**E**
Биты 0—4	Биты 5—30	Биты 31—56	Биты 57—70	Биты 71—96
10	Адрес	Адрес	Смещение	Адрес

Размер команды: 13 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является поле E. Результат: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле D).

Тест (A=10, B=130, C=356, D=198, E=179):

```
0x4A, 0x10, 0x00, 0x00, 0xB2, 0x00, 0x00, 0x8C, 0x81, 0x59, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг вправо` над двумя векторами длины 7. Результат записать в новый 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 **CLI**.

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. `cat`.

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 графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся хеш-значения. 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 **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-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Массивы.

- 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. `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**	**C**
Биты 0—4	Биты 5—23	Биты 24—49
26	Константа	Адрес

Размер команды: 7 байт. Операнд: поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=26, B=214, C=831):

```
0xDA, 0x1A, 0x00, 0x3F, 0x03, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—18	Биты 19—44	Биты 45—70
4	Смещение	Адрес	Адрес

Размер команды: 9 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле B). Результат: ячейка памяти по адресу, которым является поле D.

Тест (A=4, B=64, C=84, D=752):

```
0x04, 0x08, 0xA0, 0x02, 0x00, 0x00, 0x5E, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—4	Биты 5—30	Биты 31—56
27	Адрес	Адрес

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

Тест (A=27, B=39, C=700):

```
0xFB, 0x04, 0x00, 0x00, 0x5E, 0x01, 0x00, 0x00
```

Бинарная операция: побитовый циклический сдвиг вправо

A	B	C	D	E
Биты 0—4	Биты 5—30	Биты 31—56	Биты 57—70	Биты 71—96
10	Адрес	Адрес	Смещение	Адрес

Размер команды: 13 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является поле E. Результат: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле D).

Тест (A=10, B=130, C=356, D=198, E=179):

```
0x4A, 0x10, 0x00, 0x00, 0xB2, 0x00, 0x00, 0x8C, 0x81, 0x59, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовый циклический сдвиг вправо над двумя векторами длины 7. Результат записать в новый 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 **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. `touch`.
2. `tail`.
3. `mv`.

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 графического изображения графа.

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.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- 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 стандартного ввода. Выходной текст на **языке json** 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:

```
let 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. `ord()`.

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—23	Биты 24—49
26	Константа	Адрес

Размер команды: 7 байт. Операнд: поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=26, B=214, C=831):

```
0xDA, 0x1A, 0x00, 0x3F, 0x03, 0x00, 0x00
```

Чтение из памяти

A	B	C	D
Биты 0—4	Биты 5—18	Биты 19—44	Биты 45—70
4	Смещение	Адрес	Адрес

Размер команды: 9 байт. Операнд: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле B). Результат: ячейка памяти по адресу, которым является поле D.

Тест (A=4, B=64, C=84, D=752):

```
0x04, 0x08, 0xA0, 0x02, 0x00, 0x00, 0x5E, 0x00, 0x00
```

Запись в память

A	B	C
Биты 0—4	Биты 5—30	Биты 31—56
27	Адрес	Адрес

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

Тест (A=27, B=39, C=700):

```
0xFB, 0x04, 0x00, 0x00, 0x5E, 0x01, 0x00, 0x00
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	B	C	D	E

A	**B**	**C**	**D**	**E**
Биты 0—4	Биты 5—30	Биты 31—56	Биты 57—70	Биты 71—96
10	Адрес	Адрес	Смещение	Адрес

Размер команды: 13 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: ячейка памяти по адресу, которым является поле E. Результат: ячейка памяти по адресу, которым является сумма адреса (ячейка памяти по адресу, которым является поле C) и смещения (поле D).

Тест (A=10, B=130, C=356, D=198, E=179):

```
0x4A, 0x10, 0x00, 0x00, 0xB2, 0x00, 0x00, 0x8C, 0x81, 0x59, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовый циклический сдвиг вправо над двумя векторами длины 7. Результат записать в новый 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 **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

- 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. `tac`.
2. `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 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.
- 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 стандартный вывод.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

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. `mod()`.
6. `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 **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—11	Биты 12—43
6	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=6, B=94, C=190):

```
0xC6, 0xEB, 0x0B, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**
Биты 0—4	Биты 5—11	Биты 12—36
8	Адрес	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=8, B=13, C=682):

```
0xA8, 0xA1, 0x2A, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—4	Биты 5—11	Биты 12—18
25	Адрес	Адрес

Размер команды: 6 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=25, B=24, C=22):

```
0x19, 0x63, 0x01, 0x00, 0x00, 0x00
```

Унарная операция: `унарный минус`

A	**B**	**C**
Биты 0—4	Биты 5—11	Биты 12—36
10	Адрес	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=10, B=43, C=841):

```
0x6A, 0x95, 0x34, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию унарный минус над вектором длины 8. Результат записать в новый 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 **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.

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

1. `clear`.
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 файла-пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Конфигурационный файл имеет формат **csv** 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 **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке toml** попадает в стандартный вывод.

Массивы:

```
({ значение, значение, значение, ... })
```

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

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 **yaml**.

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

Загрузка константы

A	**B**
Биты 0—5	Биты 6—24
30	Константа

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

Тест (A=30, B=436):

```
0x1E, 0x6D, 0x00, 0x00
```

Чтение из памяти

A	**B**

A	**B**
Биты 0—5	Биты 6—28
19	Адрес

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

Тест (A=19, B=93):

```
0x53, 0x17, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—5	Биты 6—28
17	Адрес

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

Тест (A=17, B=265):

```
0x51, 0x42, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A
Биты 0—5
23

Размер команды: 1 байт. Операнд: регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=23):

```
0x17
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 8. 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 **zip**. Эмулятор должен работать в режиме **GUI**.

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

- Путь к архиву виртуальной файловой системы.

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

1. `clear`.
2. `head`.
3. `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 **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 сообщения. Граф необходимо строить только для коммитов позже заданной даты.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.
- 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 xml** попадает в стандартный вывод.

Многострочные комментарии:

```
<#
Это многострочный
комментарий
#>
```

Словари:

```
dict(
  имя = значение,
  имя = значение,
  имя = значение,
  ...
)
```

Имена:

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

Значения:

- Числа.
- 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 **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—11	Биты 12—39
28	Адрес	Константа

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

Тест (A=28, B=4, C=882):

```
0x9C, 0x20, 0x37, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**
Биты 0—4	Биты 5—11	Биты 12—24
24	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=24, B=21, C=285):

```
0xB8, 0xD2, 0x11, 0x00
```

Запись в память

A	**B**	**C**
Биты 0—4	Биты 5—11	Биты 12—18
18	Адрес	Адрес

Размер команды: 3 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=18, B=4, C=57):

```
0x92, 0x90, 0x03
```

Бинарная операция: `побитовое исключающее "или"`

A	**B**	**C**
Биты 0—4	Биты 5—11	Биты 12—18
29	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=29, B=102, C=0):

```
0xDD, 0x0C, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовое исключающее "или" над двумя векторами длины 5. Результат записать в первый 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 **GUI**.

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.

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

1. `whoami`.
2. `chown`.

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 дата, время и автор коммита. 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 xml** 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:

```
* Это однострочный комментарий
```

Многострочные комментарии:

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

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:

```
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 **yaml**.

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—34
7	Адрес	Константа

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

Тест (A=7, B=2, C=519):

```
0x97, 0x03, 0x01, 0x00, 0x00
```

Чтение из памяти

A	B	C
Биты 0—2	Биты 3—6	Биты 7—10
3	Адрес	Адрес

Размер команды: 5 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=3, B=7, C=7):

```
0xBB, 0x03, 0x00, 0x00, 0x00
```

Запись в память

A	B	C	D
Биты 0—2	Биты 3—6	Биты 7—11	Биты 12—15
1	Адрес	Смещение	Адрес

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

Тест (A=1, B=9, C=13, D=15):

```
0xC9, 0xF6, 0x00, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	B	C	D

A	**B**	**C**	**D**
Биты 0—2	Биты 3—6	Биты 7—11	Биты 12—15
4	Адрес	Смещение	Адрес

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

Тест (A=4, B=4, C=28, D=1):

```
0x24, 0x1E, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 6. Результат записать в новый 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 **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 **csv** 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. `wc`.
2. `chown`.

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 сообщения. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным хеш-значением.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.
- 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 toml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Массивы:

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

Имена:

```
[a-z]+
```

Значения:

- Числа.
- Массивы.

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 **yaml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—30	Биты 31—33
6	Константа	Адрес

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

Тест (A=6, B=297, C=2):

```
0x86, 0x94, 0x00, 0x00, 0x01
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—9	Биты 10—22	Биты 23—25
117	Адрес	Смещение	Адрес

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

Тест (A=117, B=0, C=489, D=5):

```
0x75, 0xA4, 0x87, 0x02
```

Запись в память

A	**B**	**C**	**D**
Биты 0—6	Биты 7—9	Биты 10—22	Биты 23—25
72	Адрес	Смещение	Адрес

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

Тест (A=72, B=6, C=961, D=5):

```
0x48, 0x07, 0x8F, 0x02
```

Бинарная операция: `"!="`

A	**B**	**C**	**D**
Биты 0—6	Биты 7—9	Биты 10—12	Биты 13—25
53	Адрес	Адрес	Смещение

Размер команды: 4 байт. Первый операнд: регистр по адресу, которым является поле C. Второй 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=53, B=7, C=1, D=307):

```
0xB5, 0x67, 0x26, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"!="` над двумя векторами длины 7. Результат записать во второй 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 **CLI**.

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

- 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. `date`.
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 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 файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

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 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—5	Биты 6—19	Биты 20—22
28	Константа	Адрес

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

Тест (A=28, B=889, C=2):

```
0x5C, 0xDE, 0x20
```

Чтение из памяти

A	B	C	D
Биты 0—5	Биты 6—16	Биты 17—19	Биты 20—22
62	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=62, B=101, C=4, D=3):

```
0x7E, 0x19, 0x38
```

Запись в память

A	B	C	D
Биты 0—5	Биты 6—8	Биты 9—11	Биты 12—22
3	Адрес	Адрес	Смещение

Размер команды: 3 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=3, B=1, C=0, D=688):

```
0x43, 0x00, 0x2B
```

Унарная операция: `bitreverse()`

A	**B**	**C**
Биты 0—5	Биты 6—8	Биты 9—30
58	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=58, B=5, C=224):

```
0x7A, 0xC1, 0x01, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 5. Результат записать в исходный 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 **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 **csv** 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. `cal`.
2. `cp`.
3. `rmdir`.

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.

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 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.

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

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 **csv**.

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

Загрузка константы

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

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

Тест (A=18, B=963):

```
0x92, 0xE1, 0x01, 0x00
```

Чтение из памяти

A
Биты 0—6
37

Размер команды: 1 байт. Операнд: ячейка памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=37):

```
0x25
```

Запись в память

A	**B**
Биты 0—6	Биты 7—29
99	Адрес

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

Тест (A=99, B=650):

```
0x63, 0x45, 0x01, 0x00
```

Унарная операция: `sqrt()`

A	**B**
Биты 0—6	Биты 7—19
52	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=52, B=1):

```
0xB4, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. Результат записать в новый 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**.

Конфигурационный файл имеет формат **toml** 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 **json** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указаны дата и время. Для каждого действия указан пользователь.

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

1. `echo`.
2. `tail`.

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 **ini** и содержит:

- Путь к программе для визуализации графов.
- Путь к анализируемому пакету.
- 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 **языке json** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

```
<#
Это многострочный
комментарий
#>
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

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 **yaml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—12	Биты 13—25
15	Адрес	Константа

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

Тест (A=15, B=47, C=1014):

```
0x8F, 0xD7, 0x7E, 0x00
```

Чтение из памяти

A	B	C
Биты 0—6	Биты 7—12	Биты 13—39
59	Адрес	Адрес

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

Тест (A=59, B=45, C=298):

```
0xBB, 0x56, 0x25, 0x00, 0x00
```

Запись в память

A	B	C
Биты 0—6	Биты 7—33	Биты 34—39
11	Адрес	Адрес

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

Тест (A=11, B=420, C=32):

```
0x0B, 0xD2, 0x00, 0x00, 0x80
```

Унарная операция: `унарный минус`

A	B	C
Биты 0—6	Биты 7—12	Биты 13—18
63	Адрес	Адрес

Размер команды: 3 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=63, B=32, C=25):

```
0x3F, 0x30, 0x03
```

Тестовая программа

Выполнить поэлементно операцию унарный минус над вектором длины 8. Результат записать в исходный 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 **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 **json** 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. `mkdir`.
2. `chmod`.
3. `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 **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 ранее заданной даты.

Конфигурационный файл имеет формат **ini** 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 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
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
def 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. `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 **csv**.

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

Загрузка константы

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

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

Тест (A=18, B=963):

```
0x92, 0xE1, 0x01, 0x00
```

Чтение из памяти

A
Биты 0—6
37

Размер команды: 1 байт. Операнд: ячейка памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=37):

```
0x25
```

Запись в память

A	**B**
Биты 0—6	Биты 7—29
99	Адрес

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

Тест (A=99, B=650):

```
0x63, 0x45, 0x01, 0x00
```

Унарная операция: `sqrt()`

A	**B**
Биты 0—6	Биты 7—19
52	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=52, B=1):

```
0xB4, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. 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 **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.
- 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. `tree`.
2. `cal`.

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 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.
- 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 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:

```
имя is 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. `len()`.
3. `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
Биты 0—6	Биты 7—29
18	Константа

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

Тест (A=18, B=963):

```
0x92, 0xE1, 0x01, 0x00
```

Чтение из памяти

A
Биты 0—6
37

Размер команды: 1 байт. Операнд: ячейка памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=37):

```
0x25
```

Запись в память

A	B
Биты 0—6	Биты 7—29
99	Адрес

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

Тест (A=99, B=650):

```
0x63, 0x45, 0x01, 0x00
```

Унарная операция: `sqrt()`

A	B
Биты 0—6	Биты 7—19

A	B
52	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=52, B=1):

```
0xB4, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. Результат записать в новый 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 **CLI**.

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

- 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. `clear`.
2. `uname`.
3. `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 **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **PlantUML**. 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.

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 файл, путь к которому задан ключом командной строки.

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
def 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. Деление.
5. `pow()`.
6. `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 **csv**.

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—29
18	Константа

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

Тест (A=18, B=963):

```
0x92, 0xE1, 0x01, 0x00
```

Чтение из памяти

A
Биты 0—6
37

Размер команды: 1 байт. Операнд: ячейка памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=37):

```
0x25
```

Запись в память

A	B
Биты 0—6	Биты 7—29
99	Адрес

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

Тест (A=99, B=650):

```
0x63, 0x45, 0x01, 0x00
```

Унарная операция: `sqrt()`

A	B
Биты 0—6	Биты 7—19
52	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=52, B=1):

```
0xB4, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 6. Результат записать в новый 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 **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.

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

1. `echo`.
2. `date`.

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 в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

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

Ключами командной строки задаются:

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

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:

```
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. `concat()`.

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—26
4	Константа

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

Тест (A=4, B=480):

```
0x04, 0x0F, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—2	Биты 3—16
2	Смещение

Размер команды: 3 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=2, B=856):

```
0xC2, 0x1A, 0x00
```

Запись в память

A
Биты 0—2
1

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

Тест (A=1):

```
0x01
```

Бинарная операция: **побитовый циклический сдвиг вправо**

A	**B**	**C**
Биты 0—2	Биты 3—21	Биты 22—40
0	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: ячейка памяти по адресу, которым является поле B. Результат: ячейка памяти по адресу, которым является поле C.

Тест (A=0, B=63, C=334):

```
0xF8, 0x01, 0x80, 0x53, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг вправо` над двумя векторами длины 8. Результат записать в первый 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 **GUI**.

Ключами командной строки задаются:

- 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 указаны дата и время.

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

1. `pwd`.
2. `touch`.
3. `uname`.

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 кода.

Построить граф зависимостей для коммитов, в узлах которого содержатся дата, время и автор коммита. Граф необходимо строить только для коммитов ранее заданной даты.

Ключами командной строки задаются:

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

Многострочные комментарии:

```
=begin
Это многострочный
комментарий
=end
```

Словари:

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

Имена:

```
[_a-zA-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. `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—3	Биты 4—27
12	Константа

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

Тест (A=12, B=515):

```
0x3C, 0x20, 0x00, 0x00
```

Чтение из памяти

A
Биты 0—3
10

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

Тест (A=10):

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

Запись в память

A	**B**
Биты 0—3	Биты 4—17
7	Адрес

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

Тест (A=7, B=844):

```
0xC7, 0x34, 0x00, 0x00
```

Бинарная операция: `взятие остатка`

A	B
Биты 0—3	Биты 4—17
6	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=6, B=512):

```
0x06, 0x20, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `взятие остатка` над вектором длины 7 и числом 148. 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 **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 **json** 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. `mv`.
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 **Mermaid**. 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.

Конфигурационный файл имеет формат **xml** 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 **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, ... ]
```

Имена:

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

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. Умножение.
4. Деление.
5. `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 **csv**.

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

Загрузка константы

A	B
Биты 0—2	Биты 3—22
3	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=3, B=450):

```
0x13, 0x0E, 0x00
```

Чтение из памяти

A	B
Биты 0—2	Биты 3—13
4	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=4, B=334):

```
0x74, 0x0A
```

Запись в память

A
Биты 0—2
6

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

Тест (A=6):

```
0x06
```

Бинарная операция: `min()`

A	**B**
Биты 0—2	Биты 3—23
0	Адрес

Размер команды: 3 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=0, B=672):

```
0x00, 0x15, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `min()` над вектором длины 4 и числом 20. Результат записать в исходный 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 **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.

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. `rmdir`.
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 **языка Python (pip)**. 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 **json** 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 стандартного ввода. Выходной текст на **языке yaml** попадает в файл, путь к которому задан ключом командной строки.

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 (префиксная форма), пример:

```
?{+ имя 1}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `concat()`.

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—22
3	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=3, B=450):

```
0x13, 0x0E, 0x00
```

Чтение из памяти

A	**B**
Биты 0—2	Биты 3—13
4	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=4, B=334):

```
0x74, 0x0A
```

Запись в память

A
Биты 0—2
6

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

Тест (A=6):

```
0x06
```

Бинарная операция: `min()`

A	**B**
Биты 0—2	Биты 3—23
0	Адрес

Размер команды: 3 байт. Первый операнд: ячейка памяти по адресу, которым является поле B. Второй операнд: элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=0, B=672):

```
0x00, 0x15, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `min()` над вектором длины 4 и числом 20. Результат записать в исходный 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**.

Конфигурационный файл имеет формат **xml** 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. `chmod`.
2. `cal`.
3. `tac`.

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 графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке. 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 **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке toml** попадает в стандартный вывод.

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:

```
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. `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 **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—23	Биты 24—27
26	Константа	Адрес

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

Тест (A=26, B=469, C=11):

```
0xBA, 0x3A, 0x00, 0x0B
```

Чтение из памяти

A	B	C
Биты 0—4	Биты 5—8	Биты 9—12
13	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=13, B=3, C=8):

```
0x6D, 0x10
```

Запись в память

A	B	C
Биты 0—4	Биты 5—19	Биты 20—23
31	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=31, B=326, C=3):

```
0xDF, 0x28, 0x30
```

Унарная операция: `sqrt()`

A	B	C

A	**B**	**C**
Биты 0—4	Биты 5—8	Биты 9—12
24	Адрес	Адрес

Размер команды: 2 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=24, B=1, C=2):

```
0x38, 0x04
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 7. Результат записать в новый 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 **zip**. Эмулятор должен работать в режиме **CLI**.

Конфигурационный файл имеет формат **ini** 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. `mkdir`.
2. `clear`.

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 в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Конфигурационный файл имеет формат **yaml** 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 **языке xml** 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-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Словари.

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 **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—31
5	Константа	Адрес

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

Тест (A=5, B=313, C=14):

```
0xCD, 0x09, 0x00, 0x70
```

Чтение из памяти

A	**B**	**C**	**D**
Биты 0—2	Биты 3—7	Биты 8—21	Биты 22—26
3	Адрес	Смещение	Адрес

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

Тест (A=3, B=10, C=995, D=14):

```
0x53, 0xE3, 0x83, 0x03
```

Запись в память

A	**B**	**C**
Биты 0—2	Биты 3—7	Биты 8—12
6	Адрес	Адрес

Размер команды: 2 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=6, B=5, C=1):

```
0x2E, 0x01
```

Унарная операция: `bswap()`

A	**B**	**C**	**D**
Биты 0—2	Биты 3—7	Биты 8—26	Биты 27—40
1	Адрес	Адрес	Смещение

Размер команды: 6 байт. Операнд: ячейка памяти 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=1, B=3, C=352, D=593):

```
0x19, 0x60, 0x01, 0x88, 0x12, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bswap()` 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 **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.

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. `cat`.
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 файла-пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

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

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

Значения:

- Числа.
- 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—4	Биты 5—29
15	Константа

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

Тест (A=15, B=908):

```
0x8F, 0x71, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—4	Биты 5—26
23	Адрес

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

Тест (A=23, B=595):

```
0x77, 0x4A, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—4	Биты 5—26
7	Адрес

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

Тест (A=7, B=531):

```
0x67, 0x42, 0x00, 0x00
```

Бинарная операция: `pow()`

A	**B**
Биты 0—4	Биты 5—26
13	Адрес

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

Тест (A=13, B=485):

```
0xAD, 0x3C, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `pow()` над вектором длины 6 и числом 4. 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 **yaml** 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. `uptime`.
2. `touch`.

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 **PlantUML**.

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.
- 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 json** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

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

Массивы:

```
'( 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:

```
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. `max()`.
6. `abs()`.

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—4	Биты 5—29
23	Константа

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

Тест (A=23, B=575):

```
0xF7, 0x47, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—4	Биты 5—26
7	Адрес

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

Тест (A=7, B=87):

```
0xE7, 0x0A, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—4	Биты 5—26
0	Адрес

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

Тест (A=0, B=785):

```
0x20, 0x62, 0x00, 0x00
```

Бинарная операция: вычитание

A	**B**
Биты 0—4	Биты 5—26
29	Адрес

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

Тест (A=29, B=626):

```
0x5D, 0x4E, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию вычитание над вектором длины 5 и числом 132. Результат записать в новый 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 **CLI**.

Ключами командной строки задаются:

- 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. `find`.
2. `cal`.
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 **xml** 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 стандартного ввода. Выходной текст на **языке json** попадает в стандартный вывод.

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Массивы:

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

Словари:

```
begin
  имя := значение;
  имя := значение;
  имя := значение;
  ...
end
```

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.
- 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. Умножение.
4. Деление.
5. `ord()`.
6. `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 **json**.

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—29
23	Константа

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

Тест (A=23, B=575):

```
0xF7, 0x47, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—4	Биты 5—26
7	Адрес

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

Тест (A=7, B=87):

```
0xE7, 0x0A, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—4	Биты 5—26
0	Адрес

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

Тест (A=0, B=785):

```
0x20, 0x62, 0x00, 0x00
```

Бинарная операция: вычитание

A	**B**
Биты 0—4	Биты 5—26
29	Адрес

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

Тест (A=29, B=626):

```
0x5D, 0x4E, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию вычитание над вектором длины 5 и числом 132. Результат записать в новый 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. `mv`.
2. `date`.
3. `find`.

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 находятся связи с файлами и папками, представленными уникальными узлами. Граф необходимо строить только для коммитов ранее заданной даты.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- 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 toml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
|| Это однострочный комментарий
```

Многострочные комментарии:

```
/+
Это многострочный
комментарий
+/
```

Массивы:

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

Словари:

```
begin
  имя := значение;
  имя := значение;
  имя := значение;
```

```
  ...
end
```

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 **yaml**.

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—29
15	Константа

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

Тест (A=15, B=908):

```
0x8F, 0x71, 0x00, 0x00
```

Чтение из памяти

A	**B**
Биты 0—4	Биты 5—26
23	Адрес

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

Тест (A=23, B=595):

```
0x77, 0x4A, 0x00, 0x00
```

Запись в память

A	**B**
Биты 0—4	Биты 5—26
7	Адрес

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

Тест (A=7, B=531):

```
0x67, 0x42, 0x00, 0x00
```

Бинарная операция: `pow()`

A	B
Биты 0—4	Биты 5—26
13	Адрес

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

Тест (A=13, B=485):

```
0xAD, 0x3C, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `pow()` над вектором длины 6 и числом 4. 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 **GUI**.

Конфигурационный файл имеет формат **ini** 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. `whoami`.
2. `tail`.
3. `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 файла-пакета **языка Python (pip)**. Для описания графа зависимостей используется представление **Mermaid**. 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 **языке yaml** 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:

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

Имена:

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

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 **yaml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—29
139	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=139, B=25, C=939):

```
0x8B, 0xD9, 0xEA, 0x00, 0x00, 0x00
```

Чтение из памяти

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—45
35	Адрес	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=35, B=4, C=743):

```
0x23, 0xC4, 0xB9, 0x00, 0x00, 0x00
```

Запись в память

A	**B**	**C**	**D**
Биты 0—7	Биты 8—13	Биты 14—25	Биты 26—31
203	Адрес	Смещение	Адрес

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

Тест (A=203, B=46, C=425, D=29):

```
0xCB, 0x6E, 0x6A, 0x74, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	**B**	**C**	**D**
Биты 0—7	Биты 8—13	Биты 14—19	Биты 20—31
225	Адрес	Адрес	Смещение

Размер команды: 6 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=225, B=16, C=29, D=6):

```
0xE1, 0x50, 0x67, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` 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 **tar**. Эмулятор должен работать в режиме **GUI**.

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

- 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 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. `who`.
2. `du`.
3. `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 **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.

Конфигурационный файл имеет формат **ini** 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 стандартного ввода. Выходной текст на **языке yaml** попадает в файл, путь к которому задан ключом командной строки.

Словари:

```
table(
  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:

```
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. `print()`.
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 **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—20
3	Адрес	Константа

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

Тест (A=3, B=4, C=63):

```
0xA3, 0x1F, 0x00
```

Чтение из памяти

A	**B**	**C**
Биты 0—2	Биты 3—6	Биты 7—10
4	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=4, B=1, C=15):

```
0x8C, 0x07
```

Запись в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—11	Биты 12—15	Биты 16—19
6	Смещение	Адрес	Адрес

Размер команды: 3 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=6, B=44, C=2, D=5):

```
0x66, 0x21, 0x05
```

Бинарная операция: `min()`

A	**B**	**C**
Биты 0—2	Биты 3—6	Биты 7—10
0	Адрес	Адрес

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

Тест (A=0, B=13, C=5):

```
0xE8, 0x02
```

Тестовая программа

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

Конфигурационный файл имеет формат **ini** 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. `cp`.
2. `who`.

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.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.

- Путь к анализируемому репозиторию.

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.

Однострочные комментарии:

```
\ Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

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 **yaml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—29
139	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=139, B=25, C=939):

```
0x8B, 0xD9, 0xEA, 0x00, 0x00, 0x00
```

Чтение из памяти

A	B	C
Биты 0—7	Биты 8—13	Биты 14—45
35	Адрес	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=35, B=4, C=743):

```
0x23, 0xC4, 0xB9, 0x00, 0x00, 0x00
```

Запись в память

A	B	C	D
Биты 0—7	Биты 8—13	Биты 14—25	Биты 26—31
203	Адрес	Смещение	Адрес

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

Тест (A=203, B=46, C=425, D=29):

```
0xCB, 0x6E, 0x6A, 0x74, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	B	C	D
Биты 0—7	Биты 8—13	Биты 14—19	Биты 20—31
225	Адрес	Адрес	Смещение

Размер команды: 6 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=225, B=16, C=29, D=6):

```
0xE1, 0x50, 0x67, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` 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 **tar**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **ini** 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.
- Путь к стартовому скрипту.

Лог-файл имеет формат **csv** 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. `rev`.
2. `chown`.

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

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. `len()`.
3. `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 **yaml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—29

A	B	C
139	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=139, B=25, C=939):

```
0x8B, 0xD9, 0xEA, 0x00, 0x00, 0x00
```

Чтение из памяти

A	B	C
Биты 0—7	Биты 8—13	Биты 14—45
35	Адрес	Адрес

Размер команды: 6 байт. Операнд: ячейка памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=35, B=4, C=743):

```
0x23, 0xC4, 0xB9, 0x00, 0x00, 0x00
```

Запись в память

A	B	C	D
Биты 0—7	Биты 8—13	Биты 14—25	Биты 26—31
203	Адрес	Смещение	Адрес

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

Тест (A=203, B=46, C=425, D=29):

```
0xCB, 0x6E, 0x6A, 0x74, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	B	C	D
Биты 0—7	Биты 8—13	Биты 14—19	Биты 20—31

A	B	C	D
225	Адрес	Адрес	Смещение

Размер команды: 6 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=225, B=16, C=29, D=6):

```
0xE1, 0x50, 0x67, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 8. Результат записать в исходный вектор.