Конфигурационное управление

Сборник домашних заданий

Группа 11

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**.

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

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

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

1. `mkdir`.
2. `chown`.

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 средства для получения зависимостей использовать нельзя.

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

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Максимальная глубина анализа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
set имя = значение
```

Вычисление константы на этапе трансляции:

```
{имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—7	Биты 8—34
12	Константа

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

Тест (A=12, B=34):

```
0x0C, 0x22, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—23
182	Смещение

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

Тест (A=182, B=570):

```
0xB6, 0x3A, 0x02, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—23
113	Смещение

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

Тест (A=113, B=878):

```
0x71, 0x6E, 0x03, 0x00, 0x00
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	**B**
Биты 0—7	Биты 8—23
197	Смещение

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

Тест (A=197, B=725):

```
0xC5, 0xD5, 0x02, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг вправо` над двумя векторами длины 5. 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 **tar**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **xml** 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 **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. `chown`.
2. `find`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **языка JavaScript (npm)**. 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.
- Имя анализируемого пакета.
- 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 yaml** 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]+
```

Значения:

- Числа.
- Словари.

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. `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**
Биты 0—7	Биты 8—30
73	Константа

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

Тест (A=73, B=898):

```
0x49, 0x82, 0x03, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—31
221	Адрес

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

Тест (A=221, B=939):

```
0xDD, 0xAB, 0x03, 0x00
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—31
148	Адрес

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

Тест (A=148, B=739):

```
0x94, 0xE3, 0x02, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—7	Биты 8—31
131	Адрес

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

Тест (A=131, B=550):

```
0x83, 0x26, 0x02, 0x00
```

Тестовая программа

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

Вариант №3

Задание №1

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

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.
- 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. 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. `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 средства для получения зависимостей использовать нельзя.

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

Построить граф зависимостей для коммитов, в узлах которого находятся списки файлов и папок. 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 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
}}
```

Массивы:

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

Словари:

```
table([
  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 (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. `min()`.

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—28	Биты 29—49
87	Адрес	Константа

Размер команды: 10 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=87, B=593, C=327):

```
0xD7, 0x28, 0x01, 0xE0, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—28	Биты 29—50
104	Адрес	Адрес

Размер команды: 10 байт. Операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=104, B=394, C=282):

```
0x68, 0xC5, 0x00, 0x40, 0x23, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—6	Биты 7—28	Биты 29—39	Биты 40—61
102	Адрес	Смещение	Адрес

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

Тест (A=102, B=143, C=160, D=28):

```
0xE6, 0x47, 0x00, 0x00, 0x14, 0x1C, 0x00, 0x00, 0x00, 0x00
```

Бинарная операция: побитовое "или"

A	**B**	**C**	**D**
Биты 0—6	Биты 7—28	Биты 29—50	Биты 51—72
47	Адрес	Адрес	Адрес

Размер команды: 10 байт. Первый операнд: значение в памяти по адресу, которым является поле D. Второй операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=47, B=857, C=747, D=692):

```
0xAF, 0xAC, 0x01, 0x60, 0x5D, 0x00, 0xA0, 0x15, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "или"` над вектором длины 6 и числом 150. 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**.

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.

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.

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

1. `touch`.
2. `cal`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **ОС Ubuntu (apt)**. 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.
- Имя анализируемого пакета.
- 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 json** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
var имя = значение
```

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

```
^[имя + 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—30	Биты 31—34
2	Константа	Адрес

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

Тест (A=2, B=714, C=0):

```
0xA2, 0x2C, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—7	Биты 8—15	Биты 16—19
0	Адрес	Смещение	Адрес

Размер команды: 3 байт. Операнд: значение в 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=0, B=9, C=187, D=11):

```
0x90, 0xBB, 0x0B
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—28
5	Адрес	Адрес

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

Тест (A=5, B=4, C=342):

```
0x45, 0x56, 0x01, 0x00
```

Бинарная операция: `побитовый логический сдвиг вправо`

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—28
6	Адрес	Адрес

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

Тест (A=6, B=13, C=434):

```
0xD6, 0xB2, 0x01, 0x00
```

Тестовая программа

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

Вариант №5

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `whoami`.
2. `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 средства 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 дата, время и автор коммита. Граф необходимо строить для тега с заданным именем.

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
; Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

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. 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 **json**.

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—9	Биты 10—33

A	B	C
10	Адрес	Константа

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

Тест (A=10, B=6, C=632):

```
0x0A, 0xE3, 0x09, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—6	Биты 7—38	Биты 39—41
54	Адрес	Адрес

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

Тест (A=54, B=328, C=3):

```
0x36, 0xA4, 0x00, 0x00, 0x80, 0x01
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—9	Биты 10—12
39	Адрес	Адрес

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

Тест (A=39, B=1, C=5):

```
0xA7, 0x14
```

Унарная операция: `popcnt()`

A	B	C
Биты 0—6	Биты 7—9	Биты 10—41

A	**B**	**C**
18	Адрес	Адрес

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

Тест (A=18, B=6, C=310):

```
0x12, 0xDB, 0x04, 0x00, 0x00, 0x00
```

Тестовая программа

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

Вариант №6

Задание №1

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

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

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

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

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

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

1. `tac`.
2. `chown`.
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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
; Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Словари:

```
dict(
  имя = значение,
  имя = значение,
  имя = значение,
```

```
  ...
)
```

Имена:

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

Значения:

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

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
значение -> имя
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—7	Биты 8—27
137	Константа

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

Тест (A=137, B=749):

```
0x89, 0xED, 0x02, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—20
186	Смещение

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

Тест (A=186, B=693):

```
0xBA, 0xB5, 0x02, 0x00
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—31
91	Адрес

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

Тест (A=91, B=427):

```
0x5B, 0xAB, 0x01, 0x00
```

Бинарная операция: `побитовый логический сдвиг вправо`

A	**B**
Биты 0—7	Биты 8—31
178	Адрес

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

Тест (A=178, B=881):

```
0xB2, 0x71, 0x03, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовый логический сдвиг вправо` над двумя векторами длины 6. 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 **zip**. Эмулятор должен работать в режиме **GUI**.

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

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

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

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

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

1. `whoami`.
2. `clear`.
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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.

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
}}
```

Словари:

```
dict(
  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 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. Вычитание.
3. Умножение.
4. Деление.
5. `print()`.
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 **csv**.

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—13
10	Константа

Размер команды: 2 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=10, B=161):

```
0x2A, 0x14
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—35
14	Адрес

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

Тест (A=14, B=411):

```
0x6E, 0x33, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—35
21	Адрес

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

Тест (A=21, B=311):

```
0xF5, 0x26, 0x00, 0x00, 0x00
```

Бинарная операция: `побитовый циклический сдвиг влево`

A	**B**
Биты 0—4	Биты 5—35
16	Адрес

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

Тест (A=16, B=194):

```
0x50, 0x18, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг влево` над двумя векторами длины 6. Результат записать во второй 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 **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. `rm`.
2. `rev`.

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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени 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.

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 **языке json** 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:

```
(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. `concat()`.
6. `ord()`.

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—33
8	Константа

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

Тест (A=8, B=539):

```
0xB8, 0x21, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—16
4	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=4, B=163):

```
0x34, 0x0A, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—16
13	Адрес

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=13, B=671):

```
0xFD, 0x29, 0x00
```

Унарная операция: `abs()`

A
Биты 0—3
0

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

Тест (A=0):

```
0x00
```

Тестовая программа

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

Вариант №9

Задание №1

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

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- 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. `who`.
2. `history`.
3. `date`.

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

Задание №2

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

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

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами.

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

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
var имя значение;
```

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

```
|имя 1 +|
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `concat()`.
6. `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—2	Биты 3—6	Биты 7—37
5	Адрес	Константа

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

Тест (A=5, B=5, C=328):

```
0x2D, 0xA4, 0x00, 0x00, 0x00
```

Чтение значения из памяти

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

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

Тест (A=1, B=15, C=8):

```
0x79, 0x04, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C	D
Биты 0—2	Биты 3—6	Биты 7—18	Биты 19—22
2	Адрес	Смещение	Адрес

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

Тест (A=2, B=15, C=243, D=9):

```
0xFA, 0x79, 0x48, 0x00, 0x00
```

Бинарная операция: `">="`

A	B	C	D
Биты 0—2	Биты 3—6	Биты 7—10	Биты 11—14
7	Адрес	Адрес	Адрес

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

Тест (A=7, B=0, C=4, D=4):

```
0x07, 0x22, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `">="` над вектором длины 5 и числом 131. Результат записать в новый 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 **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. `wc`.
2. `rmdir`.
3. `cal`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние средства 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 номера коммитов в хронологическом порядке. Граф необходимо строить только для коммитов позже заданной даты.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- 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 файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** 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:

```
имя := значение;
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

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

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

Тест (A=62, B=38):

```
0x3E, 0x13, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—6
88

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=88):

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

Запись значения в память

A	**B**
Биты 0—6	Биты 7—38
57	Адрес

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

Тест (A=57, B=112):

```
0x39, 0x38, 0x00, 0x00, 0x00
```

Унарная операция: побитовое "не"

A	**B**
Биты 0—6	Биты 7—38
9	Адрес

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

Тест (A=9, B=689):

```
0x89, 0x58, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовое "не" над вектором длины 6. Результат записать в исходный вектор.

Вариант №11

Задание №1

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

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. `mv`.
2. `mkdir`.

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

Задание №2

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

Зависимости определяются по имени 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.
- Имя анализируемого пакета.
- 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.

Многострочные комментарии:

```
/#
Это многострочный
комментарий
#/
```

Массивы:

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

Имена:

```
[a-zA-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 **json**.

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

Загрузка константы

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

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

Тест (A=62, B=38):

```
0x3E, 0x13, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—6
88

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=88):

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

Запись значения в память

A	**B**
Биты 0—6	Биты 7—38
57	Адрес

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

Тест (A=57, B=112):

```
0x39, 0x38, 0x00, 0x00, 0x00
```

Унарная операция: побитовое "не"

A	**B**
Биты 0—6	Биты 7—38
9	Адрес

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

Тест (A=9, B=689):

```
0x89, 0x58, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовое "не" над вектором длины 6. 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 **tar**. 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. `cal`.
2. `chmod`.

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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
% Это однострочный комментарий
```

Многострочные комментарии:

```
--[[
Это многострочный
комментарий
]]
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

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**
Биты 0—4	Биты 5—17
16	Константа

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

Тест (A=16, B=48):

```
0x10, 0x06, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—4
30

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=30):

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

Запись значения в память

A	**B**
Биты 0—4	Биты 5—15
19	Смещение

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

Тест (A=19, B=425):

```
0x33, 0x35, 0x00, 0x00
```

Бинарная операция: вычитание

A	**B**
Биты 0—4	Биты 5—15
26	Смещение

Размер команды: 4 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: значение в памяти по адресу, которым является сумма адреса

(элемент, снятый с вершины стека) и смещения (поле B). Результат: новый элемент на стеке.

Тест (A=26, B=784):

```
0x1A, 0x62, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `вычитание` над двумя векторами длины 5. Результат записать в новый вектор.

Вариант №13

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **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 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. `cp`.
2. `uptime`.
3. `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 средства 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 для ветки с заданным именем.

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
// Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
(def имя значение)
```

Вычисление константы на этапе трансляции:

```
$(имя)
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

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

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

Тест (A=62, B=38):

```
0x3E, 0x13, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—6
88

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=88):

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

Запись значения в память

A	**B**
Биты 0—6	Биты 7—38
57	Адрес

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

Тест (A=57, B=112):

```
0x39, 0x38, 0x00, 0x00, 0x00
```

Унарная операция: `побитовое "не"`

A	**B**
Биты 0—6	Биты 7—38
9	Адрес

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

Тест (A=9, B=689):

```
0x89, 0x58, 0x01, 0x00, 0x00
```

Тестовая программа

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

Вариант №14

Задание №1

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

Ключами командной строки задаются:

- 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. `mv`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 стандартный вывод.

Многострочные комментарии:

```
--[[
Это многострочный
комментарий
]]
```

Массивы:

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

Словари:

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

Имена:

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

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()`.

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—14	Биты 15—21
17	Константа	Адрес

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

Тест (A=17, B=62, C=3):

```
0xD1, 0x87, 0x01
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—11	Биты 12—25	Биты 26—32
3	Адрес	Смещение	Адрес

Размер команды: 5 байт. Операнд: значение в 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=3, B=103, C=101, D=76):

```
0xE3, 0x5C, 0x06, 0x30, 0x01
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—4	Биты 5—11	Биты 12—18	Биты 19—32
1	Адрес	Адрес	Смещение

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

Тест (A=1, B=61, C=52, D=812):

```
0xA1, 0x47, 0x63, 0x19, 0x00
```

Бинарная операция: `побитовый циклический сдвиг вправо`

A	**B**	**C**
Биты 0—4	Биты 5—33	Биты 34—40
20	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: регистр по адресу, которым является поле C. Второй операнд: значение в памяти по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=20, B=852, C=103):

```
0x94, 0x6A, 0x00, 0x00, 0x9C, 0x01
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг вправо` над двумя векторами длины 7. Результат записать во второй вектор.

Вариант №15

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **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. `uptime`.
2. `rmdir`.
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 средства 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 только для тех коммитов, где фигурирует файл с заданным хеш-значением.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- 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 yaml** попадает в стандартный вывод.

Многострочные комментарии:

```
/#
Это многострочный
комментарий
#/
```

Массивы:

```
[ 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 **json**.

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

Загрузка константы

A	**B**
Биты 0—3	Биты 4—23
8	Константа

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

Тест (A=8, B=960):

```
0x08, 0x3C, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—13
15	Смещение

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

Тест (A=15, B=563):

```
0x3F, 0x23
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—16
1	Адрес

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

Тест (A=1, B=913):

```
0x11, 0x39, 0x00
```

Бинарная операция: побитовое "или"

A	**B**
Биты 0—3	Биты 4—16
7	Адрес

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

Тест (A=7, B=700):

```
0xC7, 0x2B, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовое "или" над вектором длины 7 и числом 173. Результат записать в исходный вектор.

Вариант №16

Задание №1

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

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. `cat`.
2. `echo`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **языка Python (pip)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
C Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
имя is 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—3	Биты 4—9	Биты 10—41
6	Адрес	Константа

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

Тест (A=6, B=40, C=803):

```
0x86, 0x8E, 0x0C, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—9	Биты 10—25
10	Адрес	Адрес

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

Тест (A=10, B=28, C=934):

```
0xCA, 0x99, 0x0E, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—3	Биты 4—10	Биты 11—16	Биты 17—22
12	Смещение	Адрес	Адрес

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

Тест (A=12, B=46, C=31, D=60):

```
0xEC, 0xFA, 0x78, 0x00, 0x00, 0x00
```

Бинарная операция: `взятие остатка`

A	**B**	**C**	**D**	**E**
Биты 0—3	Биты 4—10	Биты 11—16	Биты 17—22	Биты 23—28
14	Смещение	Адрес	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле D. Второй операнд: регистр по адресу, которым является поле E. Результат: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле B).

Тест (A=14, B=92, C=42, D=57, E=33):

```
0xCE, 0x55, 0xF3, 0x10, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `взятие остатка` над двумя векторами длины 6. 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 **zip**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **toml** 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. `rm`.
2. `rmdir`.
3. `history`.

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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
(define 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—3	Биты 4—9	Биты 10—41
6	Адрес	Константа

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

Тест (A=6, B=40, C=803):

```
0x86, 0x8E, 0x0C, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—3	Биты 4—9	Биты 10—25
10	Адрес	Адрес

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

Тест (A=10, B=28, C=934):

```
0xCA, 0x99, 0x0E, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C	D
Биты 0—3	Биты 4—10	Биты 11—16	Биты 17—22
12	Смещение	Адрес	Адрес

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

Тест (A=12, B=46, C=31, D=60):

```
0xEC, 0xFA, 0x78, 0x00, 0x00, 0x00
```

Бинарная операция: `взятие остатка`

A	B	C	D	E
Биты 0—3	Биты 4—10	Биты 11—16	Биты 17—22	Биты 23—28
14	Смещение	Адрес	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле D. Второй операнд: регистр по адресу, которым является поле E. Результат: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле B).

Тест (A=14, B=92, C=42, D=57, E=33):

```
0xCE, 0x55, 0xF3, 0x10, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `взятие остатка` над двумя векторами длины 6. Результат записать в новый вектор.

Вариант №18

Задание №1

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

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

Задание №2

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

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

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

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:

```
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. `mod()`.
4. `min()`.

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—27	Биты 28—59
82	Константа	Адрес

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

Тест (A=82, B=732, C=240):

```
0x52, 0xDC, 0x02, 0x00, 0x0F, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—39	Биты 40—71
200	Адрес	Адрес

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

Тест (A=200, B=552, C=534):

```
0xC8, 0x28, 0x02, 0x00, 0x00, 0x16, 0x02, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—7	Биты 8—39	Биты 40—71	Биты 72—86
57	Адрес	Адрес	Смещение

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

Тест (A=57, B=456, C=149, D=398):

```
0x39, 0xC8, 0x01, 0x00, 0x00, 0x95, 0x00, 0x00, 0x00, 0x8E, 0x01
```

Унарная операция: `bswap()`

A	**B**	**C**	**D**
Биты 0—7	Биты 8—39	Биты 40—71	Биты 72—86
184	Адрес	Адрес	Смещение

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

Тест (A=184, B=121, C=951, D=76):

```
0xB8, 0x79, 0x00, 0x00, 0x00, 0xB7, 0x03, 0x00, 0x00, 0x4C, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bswap()` над вектором длины 7. Результат записать в новый вектор.

Вариант №19

Задание №1

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

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

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

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

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

1. `tac`.
2. `pwd`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **ОС Alpine Linux (apk)**. 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 **ini** и содержит:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 стандартный вывод.

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
let имя = значение;
```

Вычисление константы на этапе трансляции:

```
!(имя)
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—19	Биты 20—35
233	Адрес	Константа

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

Тест (A=233, B=459, C=597):

```
0xE9, 0xCB, 0x51, 0x25, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—19	Биты 20—31
128	Адрес	Адрес

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

Тест (A=128, B=261, C=568):

```
0x80, 0x05, 0x81, 0x23, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—19	Биты 20—31
112	Адрес	Адрес

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

Тест (A=112, B=481, C=41):

```
0x70, 0xE1, 0x91, 0x02, 0x00, 0x00
```

Унарная операция: `sgn()`

A	**B**	**C**
Биты 0—7	Биты 8—19	Биты 20—31
206	Адрес	Адрес

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

Тест (A=206, B=844, C=555):

```
0xCE, 0x4C, 0xB3, 0x22, 0x00, 0x00
```

Тестовая программа

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

Ключами командной строки задаются:

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

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

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

1. `pwd`.
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 средства для получения зависимостей использовать нельзя.

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

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке. 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 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 **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
--}}
```

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 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. `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**
Биты 0—3	Биты 4—26
2	Константа

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

Тест (A=2, B=29):

```
0xD2, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—27
12	Адрес

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

Тест (A=12, B=234):

```
0xAC, 0x0E, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—27
11	Адрес

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

Тест (A=11, B=361):

```
0x9B, 0x16, 0x00, 0x00
```

Бинарная операция: `"=="`

A	**B**
Биты 0—3	Биты 4—27
7	Адрес

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

Тест (A=7, B=626):

```
0x27, 0x27, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"=="` над двумя векторами длины 5. 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**.

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. `find`.
2. `cp`.

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

Задание №2

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

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

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 yaml** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

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

Имена:

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

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 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—5	Биты 6—36
18	Константа

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

Тест (A=18, B=827):

```
0xD2, 0xCE, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B
Биты 0—5	Биты 6—24
29	Адрес

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

Тест (A=29, B=463):

```
0xDD, 0x73, 0x00, 0x00, 0x00
```

Запись значения в память

A	B
Биты 0—5	Биты 6—24
58	Адрес

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

Тест (A=58, B=459):

```
0xFA, 0x72, 0x00, 0x00, 0x00
```

Бинарная операция: "<"

A	B
Биты 0—5	Биты 6—24
33	Адрес

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

Тест (A=33, B=742):

```
0xA1, 0xB9, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "<" над вектором длины 6 и числом 199. Результат записать в новый вектор.

Вариант №22

Задание №1

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

Конфигурационный файл имеет формат **toml** 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. `touch`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени 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.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.

- 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 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-z]+
```

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
var 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. `max()`.
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 **csv**.

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

Загрузка константы

A	B
Биты 0—7	Биты 8—23
233	Константа

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

Тест (A=233, B=459):

```
0xE9, 0xCB, 0x01
```

Чтение значения из памяти

A	B
Биты 0—7	Биты 8—19
128	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=128, B=742):

```
0x80, 0xE6, 0x02
```

Запись значения в память

A
Биты 0—7
112

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=112):

```
0x70, 0x00, 0x00
```

Унарная операция: `sgn()`

A
Биты 0—7

A
206

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=206):

```
0xCE, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sgn()` над вектором длины 5. Результат записать в исходный 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 **CLI**.

Ключами командной строки задаются:

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

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

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

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

1. `head`.
2. `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 средства для получения зависимостей использовать нельзя.

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

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке. Граф необходимо строить только для коммитов позже заданной даты.

Ключами командной строки задаются:

- 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 yaml** попадает в стандартный вывод.

Многострочные комментарии:

```
/*
Это многострочный
комментарий
*/
```

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. `min()`.
4. `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—7	Биты 8—23
233	Константа

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

Тест (A=233, B=459):

```
0xE9, 0xCB, 0x01
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—19
128	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=128, B=742):

```
0x80, 0xE6, 0x02
```

Запись значения в память

A
Биты 0—7
112

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=112):

```
0x70, 0x00, 0x00
```

Унарная операция: `sgn()`

A
Биты 0—7
206

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=206):

```
0xCE, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sgn()` над вектором длины 5. 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 **CLI**.

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

- Путь к архиву виртуальной файловой системы.

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

1. `mv`.
2. `tree`.
3. `uname`.

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

Задание №2

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

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

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

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
-- Это однострочный комментарий
```

Массивы:

```
{ 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:

```
def имя = значение
```

Вычисление константы на этапе трансляции:

```
!(имя)
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—6	Биты 7—38	Биты 39—42
69	Константа	Адрес

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

Тест (A=69, B=489, C=2):

```
0xC5, 0xF4, 0x00, 0x00, 0x00, 0x01
```

Чтение значения из памяти

A	B	C
Биты 0—6	Биты 7—10	Биты 11—23
49	Адрес	Адрес

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

Тест (A=49, B=15, C=960):

```
0xB1, 0x07, 0x1E, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—10	Биты 11—14
107	Адрес	Адрес

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

Тест (A=107, B=6, C=11):

```
0x6B, 0x5B, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—14
100	Адрес	Адрес

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

Тест (A=100, B=13, C=4):

```
0xE4, 0x26, 0x00, 0x00, 0x00, 0x00
```

Тестовая программа

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

Вариант №25

Задание №1

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

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

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

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

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

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

1. `rmdir`.
2. `cal`.
3. `wc`.

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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **языка Java (Maven)**. 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.
- Имя анализируемого пакета.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
! Это однострочный комментарий
```

Массивы:

```
(list 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:

```
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 **csv**.

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

Загрузка константы

A	**B**
Биты 0—5	Биты 6—36
18	Константа

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

Тест (A=18, B=827):

```
0xD2, 0xCE, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—5
29

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=29):

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

Запись значения в память

A	**B**
Биты 0—5	Биты 6—24
58	Адрес

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

Тест (A=58, B=828):

```
0x3A, 0xCF, 0x00, 0x00, 0x00
```

Бинарная операция: "<"

A	B
Биты 0—5	Биты 6—24
33	Адрес

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

Тест (A=33, B=597):

```
0x61, 0x95, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "<" над двумя векторами длины 5. Результат записать во второй 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 **zip**. Эмулятор должен работать в режиме **GUI**.

Ключами командной строки задаются:

- Имя пользователя для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к 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. `cal`.
2. `tac`.
3. `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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке toml** попадает в стандартный вывод.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя := значение
```

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

```
${имя 1 +}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `min()`.
5. `len()`.

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—2	Биты 3—8	Биты 9—36
1	Адрес	Константа

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

Тест (A=1, B=59, C=291):

```
0xD9, 0x47, 0x02, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—8	Биты 9—14
0	Адрес	Адрес

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

Тест (A=0, B=2, C=38):

```
0x10, 0x4C
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—8	Биты 9—14	Биты 15—26
5	Адрес	Адрес	Смещение

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

Тест (A=5, B=59, C=22, D=218):

```
0xDD, 0x2D, 0x6D, 0x00
```

Бинарная операция: `">"`

A	**B**	**C**	**D**
Биты 0—2	Биты 3—8	Биты 9—37	Биты 38—43
2	Адрес	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: регистр по адресу, которым является поле D. Второй операнд: регистр по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=2, B=55, C=609, D=43):

```
0xBA, 0xC3, 0x04, 0x00, 0xC0, 0x0A
```

Тестовая программа

Выполнить поэлементно операцию ">" над вектором длины 6 и числом 114. Результат записать в исходный 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.
- Путь к архиву виртуальной файловой системы.

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

1. `cat`.
2. `cp`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние средства 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 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 **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

Вычисление константы на этапе трансляции:

```
${имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 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—3	Биты 4—28	Биты 29—33
13	Константа	Адрес

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

Тест (A=13, B=455, C=19):

```
0x7D, 0x1C, 0x00, 0x60, 0x02
```

Чтение значения из памяти

A	B	C
Биты 0—3	Биты 4—8	Биты 9—38
14	Адрес	Адрес

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

Тест (A=14, B=7, C=928):

```
0x7E, 0x40, 0x07, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—3	Биты 4—33	Биты 34—38
6	Адрес	Адрес

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

Тест (A=6, B=590, C=10):

```
0xE6, 0x24, 0x00, 0x00, 0x28
```

Унарная операция: `popcnt()`

A	B	C
Биты 0—3	Биты 4—8	Биты 9—13
7	Адрес	Адрес

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

Тест (A=7, B=26, C=0):

```
0xA7, 0x01, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 6. Результат записать в исходный 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:

- 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. `chmod`.
2. `cal`.
3. `find`.

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

Задание №2

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

Зависимости определяются по имени пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.

- 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 стандартный вывод.

Однострочные комментарии:

```
# Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

```
(list 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:

```
var 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. `min()`.
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**	**C**
Биты 0—3	Биты 4—28	Биты 29—33
13	Константа	Адрес

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

Тест (A=13, B=455, C=19):

```
0x7D, 0x1C, 0x00, 0x60, 0x02
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—13
14	Адрес	Адрес

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

Тест (A=14, B=7, C=7):

```
0x7E, 0x0E, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—33	Биты 34—38
6	Адрес	Адрес

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

Тест (A=6, B=590, C=10):

```
0xE6, 0x24, 0x00, 0x00, 0x28
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—13
7	Адрес	Адрес

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

Тест (A=7, B=26, C=0):

```
0xA7, 0x01, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 6. Результат записать в исходный 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 **zip**. Эмулятор должен работать в режиме **CLI**.

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

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

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

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

1. `mkdir`.
2. `uname`.
3. `date`.

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

Задание №2

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

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

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 xml** попадает в стандартный вывод.

Однострочные комментарии:

```
% Это однострочный комментарий
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
(define имя значение)
```

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

```
|+ имя 1|
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `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 **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—37	Биты 38—40
39	Константа	Адрес

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

Тест (A=39, B=695, C=1):

```
0xE7, 0xAD, 0x00, 0x00, 0x40, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—5	Биты 6—20	Биты 21—23	Биты 24—26
22	Смещение	Адрес	Адрес

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

Тест (A=22, B=506, C=3, D=7):

```
0x96, 0x7E, 0x60, 0x07
```

Запись значения в память

A	B	C
Биты 0—5	Биты 6—19	Биты 20—22
2	Адрес	Адрес

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

Тест (A=2, B=603, C=7):

```
0xC2, 0x96, 0x70
```

Унарная операция: `bitreverse()`

A	**B**	**C**	**D**	**E**
Биты 0—5	Биты 6—8	Биты 9—23	Биты 24—38	Биты 39—41
3	Адрес	Смещение	Смещение	Адрес

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

Тест (A=3, B=1, C=533, D=284, E=7):

```
0x43, 0x2A, 0x04, 0x1C, 0x81, 0x03
```

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 6. 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 **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. `uniq`.
2. `echo`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние средства 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 находятся связи с файлами и папками, представленными уникальными узлами.

Конфигурационный файл имеет формат **yaml** 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 **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.

Однострочные комментарии:

```
% Это однострочный комментарий
```

Многострочные комментарии:

```
(comment
Это многострочный
комментарий
)
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

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. `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 **yaml**.

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

Загрузка константы

A	B	C
Биты 0—3	Биты 4—14	Биты 15—40
9	Адрес	Константа

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

Тест (A=9, B=109, C=365):

```
0xD9, 0x86, 0xB6, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—3	Биты 4—14	Биты 15—25	Биты 26—35
15	Адрес	Адрес	Смещение

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

Тест (A=15, B=609, C=162, D=256):

```
0x1F, 0x26, 0x51, 0x00, 0x04, 0x00
```

Запись значения в память

A	B	C
Биты 0—3	Биты 4—14	Биты 15—25
7	Адрес	Адрес

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

Тест (A=7, B=425, C=985):

```
0x97, 0x9A, 0xEC, 0x01, 0x00, 0x00
```

Бинарная операция: `"<"`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—14	Биты 15—25	Биты 26—36
4	Адрес	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле D. Второй операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=4, B=145, C=931, D=418):

```
0x14, 0x89, 0xD1, 0x89, 0x06, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "<" над вектором длины 7 и числом 98. Результат записать в новый вектор.

Вариант №31

Задание №1

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

Конфигурационный файл имеет формат **yaml** 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. `rmdir`.
2. `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 средства 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 находятся списки файлов и папок.

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

Входной текст на **языке yaml** 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:

```
dict(
  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:

```
def имя = значение;
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—5	Биты 6—37
43	Константа

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

Тест (A=43, B=457):

```
0x6B, 0x72, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—5	Биты 6—20
2	Смещение

Размер команды: 3 байт. Операнд: значение в 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=680):

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

Запись значения в память

A	**B**
Биты 0—5	Биты 6—19
53	Адрес

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=53, B=625):

```
0x75, 0x9C, 0x00
```

Бинарная операция: `max()`

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

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

Тест (A=61, B=506):

```
0xBD, 0x7E, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `max()` над двумя векторами длины 5. Результат записать во второй вектор.

Вариант №32

Задание №1

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

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

- 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. `wc`.
2. `clear`.
3. `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 средства 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 хеш-значения. Граф необходимо строить для тега с заданным именем.

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
-- Это однострочный комментарий
```

Многострочные комментарии:

```
/*
Это многострочный
комментарий
*/
```

Массивы:

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

Словари:

```
{
  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:

```
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. `pow()`.

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

Задание №4

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

Для ассемблера необходимо разработать читаемое представление команд УВМ. Ассемблер принимает на вход файл с текстом исходной программы, путь к

которой задается из командной строки. Результатом работы ассемблера является бинарный файл в виде последовательности байт, путь к которому задается из командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

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

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

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

Загрузка константы

A	**B**
Биты 0—5	Биты 6—37
43	Константа

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

Тест (A=43, B=457):

```
0x6B, 0x72, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—5	Биты 6—20
2	Смещение

Размер команды: 3 байт. Операнд: значение в 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=680):

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

Запись значения в память

A	B
Биты 0—5	Биты 6—19
53	Адрес

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=53, B=625):

```
0x75, 0x9C, 0x00
```

Бинарная операция: `max()`

A	B
Биты 0—5	Биты 6—20
61	Смещение

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

Тест (A=61, B=506):

```
0xBD, 0x7E, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `max()` над двумя векторами длины 5. Результат записать во второй вектор.

Вариант №33

Задание №1

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

Конфигурационный файл имеет формат **csv** 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. `date`.
2. `pwd`.
3. `cal`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние средства 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.

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
" Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.

- Массивы.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

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

```
^{имя 1 +}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `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 **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—37	Биты 38—51
28	Константа	Адрес

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

Тест (A=28, B=680, C=695):

```
0x1C, 0xAA, 0x00, 0x00, 0xC0, 0xAD, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—19	Биты 20—33
35	Адрес	Адрес

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

Тест (A=35, B=898, C=506):

```
0xA3, 0xE0, 0xA0, 0x1F, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—5	Биты 6—19	Биты 20—33
59	Адрес	Адрес

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

Тест (A=59, B=72, C=166):

```
0x3B, 0x12, 0x60, 0x0A, 0x00
```

Унарная операция: `унарный минус`

A	**B**	**C**
Биты 0—5	Биты 6—19	Биты 20—33
39	Адрес	Адрес

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

Тест (A=39, B=985, C=935):

```
0x67, 0xF6, 0x70, 0x3A, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `унарный минус` над вектором длины 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 **zip**. Эмулятор должен работать в режиме **GUI**.

Конфигурационный файл имеет формат **yaml** 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. `head`.
2. `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 средства 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 списки файлов и папок. Граф необходимо строить для тега с заданным именем.

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

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. `mod()`.
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 **json**.

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=922, C=20):

```
0xD5, 0x1C, 0x00, 0xA0
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—23	Биты 24—28
4	Адрес	Адрес

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

Тест (A=4, B=615, C=7):

```
0x3C, 0x13, 0x00, 0x07
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—13	Биты 14—18	Биты 19—23
2	Смещение	Адрес	Адрес

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

Тест (A=2, B=380, C=4, D=2):

```
0xE2, 0x0B, 0x11
```

Бинарная операция: `min()`

A	**B**	**C**	**D**
Биты 0—2	Биты 3—23	Биты 24—28	Биты 29—33
3	Адрес	Адрес	Адрес

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

Тест (A=3, B=78, C=21, D=7):

```
0x73, 0x02, 0x00, 0xF5, 0x00
```

Тестовая программа

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

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

Задание №2

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

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

Построить граф зависимостей для коммитов, в узлах которого находятся связи с файлами и папками, представленными уникальными узлами. 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 **учебном конфигурационном языке** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **языке json** попадает в стандартный вывод.

Многострочные комментарии:

```
/#
Это многострочный
комментарий
#/
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
имя <- значение
```

Вычисление константы на этапе трансляции:

```
![имя]
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—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=570, C=4):

```
0xD6, 0x11, 0x00, 0x00, 0x02
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—2	Биты 3—5	Биты 6—8	Биты 9—18
5	Адрес	Адрес	Смещение

Размер команды: 3 байт. Операнд: значение в 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=5, B=4, C=2, D=68):

```
0xA5, 0x88, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—12	Биты 13—15	Биты 16—18
7	Смещение	Адрес	Адрес

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

Тест (A=7, B=562, C=0, D=6):

```
0x97, 0x11, 0x06
```

Бинарная операция: `побитовый циклический сдвиг влево`

A	**B**	**C**	**D**
Биты 0—2	Биты 3—5	Биты 6—8	Биты 9—18
2	Адрес	Адрес	Смещение

Размер команды: 3 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: значение в 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=2, B=6, C=6, D=566):

```
0xB2, 0x6D, 0x04
```

Тестовая программа

Выполнить поэлементно операцию побитовый циклический сдвиг влево над двумя векторами длины 5. Результат записать в первый 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 **GUI**.

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. `find`.
2. `uptime`.
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 средства для получения зависимостей использовать нельзя.

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

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
set имя = значение;
```

Вычисление константы на этапе трансляции:

```
!{имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—15	Биты 16—25
23	Константа	Адрес

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

Тест (A=23, B=320, C=206):

```
0x17, 0xA0, 0xCE, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—6	Биты 7—21	Биты 22—31	Биты 32—41
27	Смещение	Адрес	Адрес

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

Тест (A=27, B=914, C=320, D=438):

```
0x1B, 0xC9, 0x01, 0x50, 0xB6, 0x01
```

Запись значения в память

A	B	C	D
Биты 0—6	Биты 7—16	Биты 17—26	Биты 27—41
98	Адрес	Адрес	Смещение

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

Тест (A=98, B=721, C=875, D=429):

```
0xE2, 0x68, 0xD7, 0x6E, 0x0D, 0x00
```

Бинарная операция: вычитание

A	B	C	D
Биты 0—6	Биты 7—16	Биты 17—26	Биты 27—36
17	Адрес	Адрес	Адрес

Размер команды: 5 байт. Первый операнд: значение в памяти по адресу, которым является поле D. Второй операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=17, B=187, C=351, D=295):

```
0x91, 0x5D, 0xBE, 0x3A, 0x09
```

Тестовая программа

Выполнить поэлементно операцию `вычитание` над двумя векторами длины 6. Результат записать во второй вектор.

Вариант №37

Задание №1

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

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

- Путь к архиву виртуальной файловой системы.

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

1. `whoami`.
2. `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 средства 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.

Конфигурационный файл имеет формат **yaml** 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 yaml** 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
|#
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
const имя = значение
```

Вычисление константы на этапе трансляции:

```
!{имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—15	Биты 16—20
23	Константа	Адрес

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

Тест (A=23, B=320, C=6):

```
0x17, 0xA0, 0x06
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—21	Биты 22—26	Биты 27—31
27	Смещение	Адрес	Адрес

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

Тест (A=27, B=914, C=21, D=28):

```
0x1B, 0xC9, 0x41, 0xE5
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—6	Биты 7—11	Биты 12—16	Биты 17—31
98	Адрес	Адрес	Смещение

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

Тест (A=98, B=22, C=27, D=429):

```
0x62, 0xBB, 0x5B, 0x03
```

Бинарная операция: вычитание

A	**B**	**C**	**D**
Биты 0—6	Биты 7—11	Биты 12—16	Биты 17—26
17	Адрес	Адрес	Адрес

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

Тест (A=17, B=5, C=10, D=295):

```
0x91, 0xA2, 0x4E, 0x02
```

Тестовая программа

Выполнить поэлементно операцию `вычитание` над двумя векторами длины 6. Результат записать во второй 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 **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 **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. `du`.
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 средства для получения зависимостей использовать нельзя.

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

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
|| Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
[[Это строка]]
```

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. `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 **csv**.

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

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—14	Биты 15—19
1	Константа	Адрес

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

Тест (A=1, B=303, C=5):

```
0xF1, 0x92, 0x02
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—30
12	Адрес	Адрес

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

Тест (A=12, B=21, C=314):

```
0x5C, 0x75, 0x02, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—13
2	Адрес	Адрес

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

Тест (A=2, B=19, C=2):

```
0x32, 0x05
```

Унарная операция: `побитовое "не"`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—8	Биты 9—13	Биты 14—23
5	Адрес	Адрес	Смещение

Размер команды: 3 байт. Операнд: значение в 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=5, B=9, C=8, D=17):

```
0x95, 0x50, 0x04
```

Тестовая программа

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

Вариант №39

Задание №1

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

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

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

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

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

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

1. `touch`.
2. `cp`.
3. `chmod`.

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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени пакета **языка Java (Maven)**. 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 **yaml** и содержит:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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 стандартного ввода. Выходной текст на **языке yaml** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
var 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. `len()`.

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

Задание №4

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

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

командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—14	Биты 15—19
1	Константа	Адрес

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

Тест (A=1, B=303, C=5):

```
0xF1, 0x92, 0x02
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—30
12	Адрес	Адрес

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

Тест (A=12, B=21, C=314):

```
0x5C, 0x75, 0x02, 0x00
```

Запись значения в память

A	**B**	**C**

A	B	C
Биты 0—3	Биты 4—8	Биты 9—13
2	Адрес	Адрес

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

Тест (A=2, B=19, C=2):

```
0x32, 0x05
```

Унарная операция: `побитовое "не"`

A	B	C	D
Биты 0—3	Биты 4—8	Биты 9—13	Биты 14—23
5	Адрес	Адрес	Смещение

Размер команды: 3 байт. Операнд: значение в 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=5, B=9, C=8, D=17):

```
0x95, 0x50, 0x04
```

Тестовая программа

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

Вариант №40

Задание №1

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

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

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

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

1. `date`.
2. `tac`.
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 средства для получения зависимостей использовать нельзя.

Зависимости определяются по имени 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.
- Имя анализируемого пакета.
- 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 toml** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
; Это однострочный комментарий
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
имя is значение
```

Вычисление константы на этапе трансляции:

```
^(имя)
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—14	Биты 15—23
40	Адрес	Константа

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

Тест (A=40, B=76, C=44):

```
0x28, 0x4C, 0x16
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—22	Биты 23—29
216	Адрес	Адрес

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

Тест (A=216, B=110, C=34):

```
0xD8, 0x6E, 0x00, 0x11
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—14	Биты 15—21
32	Адрес	Адрес

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

Тест (A=32, B=51, C=106):

```
0x20, 0x33, 0x35
```

Бинарная операция: побитовое "и"

A	**B**	**C**	**D**
Биты 0—7	Биты 8—14	Биты 15—21	Биты 22—31
175	Адрес	Адрес	Смещение

Размер команды: 4 байт. Первый операнд: регистр по адресу, которым является поле C. Второй операнд: значение в 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=175, B=11, C=25, D=366):

```
0xAF, 0x8B, 0x8C, 0x5B
```

Тестовая программа

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