Конфигурационное управление

Сборник домашних заданий

Группа 50

РТУ МИРЭА – 2024

Оглавление

О домашних заданиях

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

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

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

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

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

Список публичных git-сервисов для репозиториев ДЗ:

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

Вариант №1

Задание №1

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

Ключами командной строки задаются:

- 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. `whoami`.

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.

Ключами командной строки задаются:

- 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 файла, путь к которому задан ключом командной строки. Выходной текст на **языке xml** попадает в стандартный вывод.

Однострочные комментарии:

```
// Это однострочный комментарий
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
имя: значение
```

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

```
?(имя + 1)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**

A	B
Биты 0—4	Биты 5—15
12	Константа

Размер команды: 2 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=12, B=70):

```
0xCC, 0x08
```

Чтение значения из памяти

A	B
Биты 0—4	Биты 5—36
8	Адрес

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

Тест (A=8, B=63):

```
0xE8, 0x07, 0x00, 0x00, 0x00
```

Запись значения в память

A	B
Биты 0—4	Биты 5—36
24	Адрес

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

Тест (A=24, B=451):

```
0x78, 0x38, 0x00, 0x00, 0x00
```

Унарная операция: `bswap()`

A	B	C
Биты 0—4	Биты 5—36	Биты 37—49
16	Адрес	Смещение

Размер команды: 7 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле C).

Тест (A=16, B=195, C=606):

```
0x70, 0x18, 0x00, 0x00, 0xC0, 0x4B, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `bswap()` над вектором длины 4. Результат записать в исходный вектор.

Вариант №2

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `echo`.
2. `clear`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 toml** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

```
[a-z][a-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 **xml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—37
2	Адрес	Константа

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

Тест (A=2, B=4, C=995):

```
0x02, 0x1A, 0x1F, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—21	Биты 22—25	Биты 26—29
109	Смещение	Адрес	Адрес

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

Тест (A=109, B=812, C=10, D=4):

```
0x6D, 0x96, 0x81, 0x12
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—42
5	Адрес	Адрес

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

Тест (A=5, B=8, C=799):

```
0x05, 0xFC, 0x18, 0x00, 0x00, 0x00
```

Бинарная операция: `pow()`

A	**B**	**C**	**D**

A	**B**	**C**	**D**
Биты 0—6	Биты 7—10	Биты 11—14	Биты 15—46
49	Адрес	Адрес	Адрес

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

Тест (A=49, B=10, C=0, D=372):

```
0x31, 0x05, 0xBA, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `pow()` над двумя векторами длины 6. Результат записать во второй 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 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. `uniq`.
2. `whoami`.
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 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.

Ключами командной строки задаются:

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

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
имя: значение;
```

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

```
![имя + 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `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 **xml**.

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

Загрузка константы

A	B	C
Биты 0—3	Биты 4—22	Биты 23—25
0	Константа	Адрес

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

Тест (A=0, B=218, C=7):

```
0xA0, 0x0D, 0x80, 0x03
```

Чтение значения из памяти

A	B	C
Биты 0—3	Биты 4—6	Биты 7—21
15	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=15, B=6, C=196):

```
0x6F, 0x62, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—3	Биты 4—18	Биты 19—21
7	Адрес	Адрес

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

Тест (A=7, B=557, C=0):

```
0xD7, 0x22, 0x00, 0x00
```

Унарная операция: `bitreverse()`

A	B	C
Биты 0—3	Биты 4—6	Биты 7—9

A	B	C
12	Адрес	Адрес

Размер команды: 4 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=12, B=7, C=5):

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

Тестовая программа

Выполнить поэлементно операцию `bitreverse()` над вектором длины 8. Результат записать в исходный 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 **xml** и содержит:

- 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 указаны дата и время. Для каждого действия указан пользователь.

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

1. `pwd`.
2. `chmod`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 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 **языке yaml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
" Это однострочный комментарий
```

Многострочные комментарии:

```
{
Это многострочный
комментарий
}
```

Массивы:

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

Словари:

```
([
  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:

```
значение -> имя;
```

Вычисление константы на этапе трансляции:

```
$имя$
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—6	Биты 7—38
122	Константа

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

Тест (A=122, B=648):

```
0x7A, 0x44, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—15
113	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=113, B=7):

```
0xF1, 0x03, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—6	Биты 7—37
18	Адрес

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

Тест (A=18, B=873):

```
0x92, 0xB4, 0x01, 0x00, 0x00
```

Бинарная операция: `побитовое "или"`

A
Биты 0—6
100

Размер команды: 5 байт. Первый операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Второй операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=100):

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

Тестовая программа

Выполнить поэлементно операцию `побитовое "или"` над вектором длины 8 и числом 181. Результат записать в исходный 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 **zip**. Эмулятор должен работать в режиме **GUI**.

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

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

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

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

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

1. `date`.
2. `echo`.
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 **языка Java (Maven)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

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

- Путь к программе для визуализации графов.
- 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 toml** попадает в стандартный вывод.

Многострочные комментарии:

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

Словари:

```
{
  имя : значение;
  имя : значение;
  имя : значение;
  ...
}
```

Имена:

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

Значения:

- Числа.
- Словари.

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 **yaml**.

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

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—6	Биты 7—37	Биты 38—69
122	Адрес	Константа

Размер команды: 13 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=122, B=648, C=159):

```
0x7A, 0x44, 0x01, 0x00, 0xC0, 0x27, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—6	Биты 7—37	Биты 38—68	Биты 69—77
113	Адрес	Адрес	Смещение

Размер команды: 13 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=113, B=19, C=42, D=285):

```
0xF1, 0x09, 0x00, 0x00, 0x80, 0x0A, 0x00, 0x00, 0xA0, 0x23, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—37	Биты 38—68
18	Адрес	Адрес

Размер команды: 13 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=18, B=496, C=283):

```
0x12, 0xF8, 0x00, 0x00, 0xC0, 0x46, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Бинарная операция: `побитовое "или"`

A	B	C	D
Биты 0—6	Биты 7—37	Биты 38—68	Биты 69—99
100	Адрес	Адрес	Адрес

Размер команды: 13 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 значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=100, B=194, C=168, D=671):

```
0x64, 0x61, 0x00, 0x00, 0x00, 0x2A, 0x00, 0x00, 0xE0, 0x53, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовое "или" над вектором длины 5 и числом 202. Результат записать в исходный вектор.

Вариант №6

Задание №1

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

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. `touch`.
2. `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 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.

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

- 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 файла, путь к которому задан ключом командной строки. Выходной текст на **языке toml** попадает в стандартный вывод.

Однострочные комментарии:

```
// Это однострочный 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:

```
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 **xml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—37
2	Адрес	Константа

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

Тест (A=2, B=4, C=995):

```
0x02, 0x1A, 0x1F, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—6	Биты 7—21	Биты 22—25	Биты 26—29
109	Смещение	Адрес	Адрес

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

Тест (A=109, B=812, C=10, D=4):

```
0x6D, 0x96, 0x81, 0x12
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—10	Биты 11—42
5	Адрес	Адрес

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

Тест (A=5, B=8, C=799):

```
0x05, 0xFC, 0x18, 0x00, 0x00, 0x00
```

Бинарная операция: `pow()`

A	B	C	D
Биты 0—6	Биты 7—10	Биты 11—14	Биты 15—46
49	Адрес	Адрес	Адрес

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

Тест (A=49, B=10, C=0, D=372):

```
0x31, 0x05, 0xBA, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `pow()` над двумя векторами длины 6. Результат записать во второй 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 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. `find`.
2. `chown`.
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 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 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 yaml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в файл, путь к которому задан ключом командной строки.

Массивы:

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

Словари:

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

Имена:

```
[a-zA-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 (постфиксная форма), пример:

```
?(имя 1 +)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `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 **yaml**.

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

Загрузка константы

A	**B**
Биты 0—3	Биты 4—25
2	Константа

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

Тест (A=2, B=964):

```
0x42, 0x3C, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—3
14

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

Тест (A=14):

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

Запись значения в память

A	**B**
Биты 0—3	Биты 4—34
0	Адрес

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

Тест (A=0, B=995):

```
0x30, 0x3E, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**
Биты 0—3	Биты 4—34
11	Адрес

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

Тест (A=11, B=812):

```
0xCB, 0x32, 0x00, 0x00, 0x00
```

Тестовая программа

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

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

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

1. `uname`.
2. `rev`.
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 **PlantUML**. 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.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
set 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()`.
3. `mod()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—5	Биты 6—23

A	B	C
1	Адрес	Константа

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

Тест (A=1, B=1, C=637):

```
0x49, 0x9F, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—2	Биты 3—5	Биты 6—8	Биты 9—14
7	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=7, B=5, C=6, D=35):

```
0xAF, 0x47, 0x00
```

Запись значения в память

A	B	C
Биты 0—2	Биты 3—5	Биты 6—8
3	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=3, B=6, C=1):

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

Унарная операция: `унарный минус`

A	B	C
Биты 0—2	Биты 3—5	Биты 6—8

A	**B**	**C**
5	Адрес	Адрес

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

Тест (A=5, B=6, C=6):

```
0xB5, 0x01, 0x00
```

Тестовая программа

Выполнить поэлементно операцию унарный минус над вектором длины 7. 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**.

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

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

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

1. `clear`.
2. `du`.

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

Однострочные комментарии:

```
% Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
значение -> имя;
```

Вычисление константы на этапе трансляции:

```
.(имя).
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—30
99	Адрес	Константа

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

Тест (A=99, B=5, C=606):

```
0xE3, 0xF2, 0x12, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—6	Биты 7—10	Биты 11—30
96	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=96, B=11, C=622):

```
0xE0, 0x75, 0x13, 0x00
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—26	Биты 27—30
1	Адрес	Адрес

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

Тест (A=1, B=744, C=11):

```
0x01, 0x74, 0x01, 0x58
```

Унарная операция: `bswap()`

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

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

Тест (A=82, B=3, C=9):

```
0xD2, 0x49
```

Тестовая программа

Выполнить поэлементно операцию `bswap()` над вектором длины 4. Результат записать в исходный вектор.

Вариант №10

Задание №1

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

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

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

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

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

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

1. `rmdir`.
2. `du`.
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 для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого содержатся хеш-значения.

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:

```
#=
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 (постфиксная форма), пример:

```
?[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `len()`.
4. `pow()`.

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

Задание №4

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

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

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

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

Форматом для файла-лога и файла-результата является **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—16	Биты 17—23
10	Константа	Адрес

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

Тест (A=10, B=350, C=99):

```
0x8A, 0x57, 0xC6, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—12	Биты 13—19
50	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=50, B=31, C=60):

```
0xF2, 0x87, 0x07, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—5	Биты 6—35	Биты 36—42
41	Адрес	Адрес

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

Тест (A=41, B=520, C=88):

```
0x29, 0x82, 0x00, 0x00, 0x80, 0x05
```

Унарная операция: `popcnt()`

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

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

Тест (A=27, B=89, C=81):

```
0x5B, 0x36, 0x0A, 0x00, 0x00, 0x00
```

Тестовая программа

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

Вариант №11

Задание №1

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

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. `tree`.
2. `history`.
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 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.

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
set 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. `sort()`.

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—24
9	Константа

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

Тест (A=9, B=282):

```
0xA9, 0x11, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—11
5	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=5, B=225):

```
0x15, 0x0E, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—26
4	Адрес

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

Тест (A=4, B=457):

```
0x94, 0x1C, 0x00, 0x00
```

Бинарная операция: `побитовый циклический сдвиг влево`

A	**B**
Биты 0—3	Биты 4—26
11	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=11, B=951):

```
0x7B, 0x3B, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовый циклический сдвиг влево над двумя векторами длины 8. Результат записать в первый вектор.

Вариант №12

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `du`.
2. `touch`.

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

Задание №2

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

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
имя <- значение;
```

Вычисление константы на этапе трансляции:

```
^{имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 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—14	Биты 15—17
3	Константа	Адрес

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

Тест (A=3, B=797, C=5):

```
0xEB, 0x98, 0x02
```

Чтение значения из памяти

A	B	C
Биты 0—2	Биты 3—5	Биты 6—23
2	Адрес	Адрес

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

Тест (A=2, B=0, C=434):

```
0x82, 0x6C, 0x00
```

Запись значения в память

A	B	C	D
Биты 0—2	Биты 3—15	Биты 16—18	Биты 19—21
6	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=6, B=427, C=6, D=4):

```
0x5E, 0x0D, 0x26
```

Бинарная операция: `побитовый циклический сдвиг влево`

A	B	C	D
Биты 0—2	Биты 3—5	Биты 6—8	Биты 9—11
5	Адрес	Адрес	Адрес

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

Тест (A=5, B=6, C=4, D=5):

```
0x35, 0x0B
```

Тестовая программа

Выполнить поэлементно операцию `побитовый циклический сдвиг влево` над двумя векторами длины 7. Результат записать во второй 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 компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.

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

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

1. `uniq`.
2. `head`.
3. `wc`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 сообщения. Граф необходимо строить для тега с заданным именем.

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

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.
- 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 yaml** попадает в файл, путь к которому задан ключом командной строки.

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
имя <- значение;
```

Вычисление константы на этапе трансляции:

```
${имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—2	Биты 3—14
1	Константа

Размер команды: 2 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=1, B=752):

```
0x81, 0x17
```

Чтение значения из памяти

A
Биты 0—2
4

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

Тест (A=4):

```
0x04
```

Запись значения в память

A	**B**
Биты 0—2	Биты 3—15
3	Смещение

Размер команды: 2 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=3, B=438):

```
0xB3, 0x0D
```

Унарная операция: `sgn()`

A

A
Биты 0—2
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 `sgn()` 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**.

Конфигурационный файл имеет формат **json** 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. `uptime`.
2. `touch`.
3. `uniq`.

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

Задание №2

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

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
(def имя значение)
```

Вычисление константы на этапе трансляции:

```
#[имя]
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—17
26	Константа

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

Тест (A=26, B=282):

```
0x5A, 0x23, 0x00, 0x00
```

Чтение значения из памяти

A	**B**

A	**B**
Биты 0—4	Биты 5—27
18	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=18, B=898):

```
0x52, 0x70, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—27
23	Адрес

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

Тест (A=23, B=457):

```
0x37, 0x39, 0x00, 0x00
```

Бинарная операция: `"!="`

A	**B**
Биты 0—4	Биты 5—15
17	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=17, B=319):

```
0xF1, 0x27, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "!=" над вектором длины 8 и числом 154. Результат записать в исходный вектор.

Вариант №15

Задание №1

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

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

- Путь к архиву виртуальной файловой системы.

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

1. `du`.
2. `pwd`.
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 **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
NB. Это однострочный комментарий
```

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:

```
def имя := значение;
```

Вычисление константы на этапе трансляции:

```
!{имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—8	Биты 9—21

A	B	C
26	Адрес	Константа

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

Тест (A=26, B=5, C=804):

```
0xBA, 0x48, 0x06, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—4	Биты 5—8	Биты 9—31
18	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=18, B=14, C=457):

```
0xD2, 0x93, 0x03, 0x00
```

Запись значения в память

A	B	C
Биты 0—4	Биты 5—8	Биты 9—31
23	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=23, B=2, C=374):

```
0x57, 0xEC, 0x02, 0x00
```

Бинарная операция: `"!="`

A	B	C	D	E
Биты 0—4	Биты 5—8	Биты 9—12	Биты 13—16	Биты 17—27
17	Адрес	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 D) и смещения (поле E). Результат: регистр по адресу, которым является поле B.

Тест (A=17, B=5, C=15, D=1, E=289):

```
0xB1, 0x3E, 0x42, 0x02
```

Тестовая программа

Выполнить поэлементно операцию `"!="` над вектором длины 5 и числом 195. Результат записать в новый вектор.

Вариант №16

Задание №1

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

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

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

Лог-файл имеет формат **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. `cal`.
2. `who`.
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 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 json** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

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 **xml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—21	Биты 22—31
30	Константа	Адрес

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

Тест (A=30, B=348, C=310):

```
0x1E, 0x5C, 0x81, 0x4D
```

Чтение значения из памяти

A	B	C	D
Биты 0—7	Биты 8—17	Биты 18—27	Биты 28—33
88	Адрес	Адрес	Смещение

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

Тест (A=88, B=39, C=437, D=41):

```
0x58, 0x27, 0xD4, 0x96, 0x02
```

Запись значения в память

A	B	C	D
Биты 0—7	Биты 8—13	Биты 14—23	Биты 24—33
49	Смещение	Адрес	Адрес

Размер команды: 5 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 D) и смещения (поле B).

Тест (A=49, B=25, C=303, D=167):

```
0x31, 0xD9, 0x4B, 0xA7, 0x00
```

Бинарная операция: `"<="`

A	B	C	D
Биты 0—7	Биты 8—17	Биты 18—27	Биты 28—37
66	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 B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=66, B=82, C=334, D=494):

```
0x42, 0x52, 0x38, 0xE5, 0x1E
```

Тестовая программа

Выполнить поэлементно операцию `"<="` над двумя векторами длины 6. Результат записать во второй вектор.

Вариант №17

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **tar**. Эмулятор должен работать в режиме **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 **csv** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

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

1. `cat`.
2. `chown`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 списки файлов и папок. 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 **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:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
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 2 примера описания конфигураций из разных предметных областей.

Задание №4

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

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

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

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

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

Загрузка константы

A	B	C
Биты 0—2	Биты 3—8	Биты 9—33
4	Адрес	Константа

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

Тест (A=4, B=0, C=843):

```
0x04, 0x96, 0x06, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—2	Биты 3—22	Биты 23—28
6	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=6, B=365, C=3):

```
0x6E, 0x0B, 0x80, 0x01
```

Запись значения в память

A	B	C
Биты 0—2	Биты 3—22	Биты 23—28
1	Адрес	Адрес

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

Тест (A=1, B=622, C=0):

```
0x71, 0x13, 0x00, 0x00
```

Унарная операция: `sgn()`

A	B	C	D

A	**B**	**C**	**D**
Биты 0—2	Биты 3—12	Биты 13—18	Биты 19—38
7	Смещение	Адрес	Адрес

Размер команды: 5 байт. Операнд: значение в памяти 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=7, B=385, C=24, D=736):

```
0x0F, 0x0C, 0x03, 0x17, 0x00
```

Тестовая программа

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

Вариант №18

Задание №1

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

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

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

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

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

1. `uname`.
2. `mv`.

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 **Mermaid**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

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

- 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 файла, путь к которому задан ключом командной строки. Выходной текст на **языке toml** 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:

```
имя: значение;
```

Вычисление константы на этапе трансляции:

```
![имя]
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—3	Биты 4—17
11	Константа

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

Тест (A=11, B=983):

```
0x7B, 0x3D, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—22
10	Адрес

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

Тест (A=10, B=71):

```
0x7A, 0x04, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—22
14	Адрес

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

Тест (A=14, B=888):

```
0x8E, 0x37, 0x00
```

Унарная операция: `побитовое "не"`

A	**B**
Биты 0—3	Биты 4—8
3	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=3, B=23):

```
0x73, 0x01
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "не"` над вектором длины 7. Результат записать в исходный вектор.

Вариант №19

Задание №1

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

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

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

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

1. `rmdir`.
2. `uname`.
3. `cat`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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.
- 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
-->
```

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:

```
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. `max()`.
6. `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—3	Биты 4—18
15	Константа

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

Тест (A=15, B=68):

```
0x4F, 0x04, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—3
3

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

Тест (A=3):

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

Запись значения в память

A	**B**
Биты 0—3	Биты 4—26
4	Адрес

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

Тест (A=4, B=632):

```
0x84, 0x27, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—3	Биты 4—26
6	Адрес

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

Тест (A=6, B=381):

```
0xD6, 0x17, 0x00, 0x00
```

Тестовая программа

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

Вариант №20

Задание №1

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

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. 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. `chmod`.
2. `cp`.

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 кода.

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке. 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 xml** попадает в стандартный вывод.

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
имя = значение
```

Вычисление константы на этапе трансляции:

```
$имя$
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—2	Биты 3—15
1	Константа

Размер команды: 2 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=1, B=414):

```
0xF1, 0x0C
```

Чтение значения из памяти

A
Биты 0—2
2

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

Тест (A=2):

```
0x02
```

Запись значения в память

A	B
Биты 0—2	Биты 3—19
3	Адрес

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

Тест (A=3, B=487):

```
0x3B, 0x0F, 0x00
```

Унарная операция: `побитовое "не"`

A
Биты 0—2
6

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

Тест (A=6):

```
0x06
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "не"` над вектором длины 4. Результат записать в новый вектор.

Вариант №21

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `tail`.
2. `pwd`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 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 **yaml** 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 json** попадает в стандартный вывод.

Однострочные комментарии:

```
REM Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.

- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя: значение;
```

Вычисление 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. `max()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

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

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

Тест (A=104, B=920):

```
0x68, 0xCC, 0x01, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—27
45	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=45, B=255):

```
0xAD, 0x7F, 0x00, 0x00
```

Запись значения в память

A
Биты 0—6
8

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

Тест (A=8):

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

Бинарная операция: сложение

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

Размер команды: 4 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=91, B=339):

```
0xDB, 0xA9, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию сложение над двумя векторами длины 4. Результат записать во второй 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 **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. `touch`.
2. `mv`.

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 графического изображения графа.

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Многострочные комментарии:

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

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:

```
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. `chr()`.
5. `pow()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

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

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

Тест (A=104, B=920):

```
0x68, 0xCC, 0x01, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—27
45	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=45, B=255):

```
0xAD, 0x7F, 0x00, 0x00
```

Запись значения в память

A
Биты 0—6
8

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

Тест (A=8):

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

Бинарная операция: сложение

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

Размер команды: 4 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=91, B=339):

```
0xDB, 0xA9, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию сложение над двумя векторами длины 4. Результат записать во второй 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 **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.

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

1. `tail`.
2. `uniq`.

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

Задание №2

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

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

- 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 стандартного ввода. Выходной текст на **языке yaml** попадает в стандартный вывод.

Однострочные комментарии:

```
REM Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя значение;
```

Вычисление константы на этапе трансляции:

```
${имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—19	Биты 20—47
36	Адрес	Константа

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

Тест (A=36, B=736, C=12):

```
0x24, 0x70, 0xC1, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—6	Биты 7—19	Биты 20—32	Биты 33—38
58	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=58, B=405, C=198, D=36):

```
0xBA, 0xCA, 0x60, 0x0C, 0x48, 0x00
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—19	Биты 20—32
25	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=25, B=919, C=959):

```
0x99, 0xCB, 0xF1, 0x3B, 0x00, 0x00
```

Бинарная операция: умножение

A	B	C	D
Биты 0—6	Биты 7—19	Биты 20—32	Биты 33—45
32	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=32, B=866, C=372, D=368):

```
0x20, 0xB1, 0x41, 0x17, 0xE0, 0x02
```

Тестовая программа

Выполнить поэлементно операцию умножение над двумя векторами длины 7. Результат записать в первый вектор.

Вариант №24

Задание №1

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

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

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

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

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

1. `uname`.
2. `pwd`.
3. `tree`.

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 **платформы .NET (nupkg)**. 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 стандартного ввода. Выходной текст на **языке xml** попадает в файл, путь к которому задан ключом командной строки.

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
def имя = значение
```

Вычисление константы на этапе трансляции:

[имя]

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—13	Биты 14—41
36	Адрес	Константа

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

Тест (A=36, B=32, C=12):

```
0x24, 0x10, 0x03, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—6	Биты 7—13	Биты 14—26
58	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=58, B=26, C=198):

```
0x3A, 0x8D, 0x31, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—6	Биты 7—13	Биты 14—26
25	Адрес	Адрес

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

Тест (A=25, B=48, C=919):

```
0x19, 0xD8, 0xE5, 0x00, 0x00, 0x00
```

Бинарная операция: умножение

A	B	C	D
Биты 0—6	Биты 7—13	Биты 14—20	Биты 21—27
32	Адрес	Адрес	Адрес

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

Тест (A=32, B=68, C=90, D=15):

```
0x20, 0xA2, 0xF6, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 6 и числом 128. Результат записать в новый вектор.

Вариант №25

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `head`.
2. `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 **платформы .NET (nupkg)**. 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.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
% Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

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 **xml**.

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

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—27	Биты 28—37
4	Адрес	Константа

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

Тест (A=4, B=736, C=12):

```
0x04, 0x2E, 0x00, 0xC0, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—19	Биты 20—43	Биты 44—67
3	Смещение	Адрес	Адрес

Размер команды: 9 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=3, B=405, C=198, D=82):

```
0x53, 0x19, 0x60, 0x0C, 0x00, 0x20, 0x05, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—27	Биты 28—51
1	Адрес	Адрес

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

Тест (A=1, B=919, C=959):

```
0x71, 0x39, 0x00, 0xF0, 0x3B, 0x00, 0x00
```

Бинарная операция: умножение

A	**B**	**C**	**D**	**E**
Биты 0—3	Биты 4—27	Биты 28—51	Биты 52—75	Биты 76—91
2	Адрес	Адрес	Адрес	Смещение

Размер команды: 12 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 B. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле C) и смещения (поле E).

Тест (A=2, B=866, C=372, D=368, E=685):

```
0x22, 0x36, 0x00, 0x40, 0x17, 0x00, 0x00, 0x17, 0x00, 0xD0, 0x2A, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение 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 **tar**. Эмулятор должен работать в режиме **GUI**.

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

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

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

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
(def имя значение);
```

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

```
|имя + 1|
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

командной строки. Дополнительный ключ командной строки задает путь к файлу-логу, в котором хранятся ассемблированные инструкции в духе списков “ключ=значение”, как в приведенных далее тестах.

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—13	Биты 14—41
36	Адрес	Константа

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

Тест (A=36, B=32, C=12):

```
0x24, 0x10, 0x03, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—13	Биты 14—20	Биты 21—26
58	Адрес	Адрес	Смещение

Размер команды: 6 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=58, B=26, C=51, D=36):

```
0x3A, 0xCD, 0x8C, 0x04, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—13	Биты 14—26
25	Адрес	Адрес

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

Тест (A=25, B=121, C=959):

```
0x99, 0xFC, 0xEF, 0x00, 0x00, 0x00
```

Бинарная операция: умножение

A	**B**	**C**	**D**
Биты 0—6	Биты 7—13	Биты 14—20	Биты 21—27
32	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 C. Результат: регистр по адресу, которым является поле B.

Тест (A=32, B=90, C=15, D=77):

```
0x20, 0xED, 0xA3, 0x09, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над двумя векторами длины 7. Результат записать в первый 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**.

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. `uname`.
2. `history`.
3. `mv`.

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

Задание №2

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

Построить граф зависимостей для коммитов, в узлах которого содержатся хеш-значения. 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 **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
```

Массивы:

```
{ 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:

```
def имя := значение;
```

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

```
.[имя + 1].
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. Деление.
5. `len()`.
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—3	Биты 4—13
8	Константа

Размер команды: 2 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=8, B=198):

```
0x68, 0x0C
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—19
7	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=7, B=187):

```
0xB7, 0x0B, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—27

A	B
2	Адрес

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

Тест (A=2, B=736):

```
0x02, 0x2E, 0x00, 0x00
```

Унарная операция: `sqrt()`

A
Биты 0—3
11

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

Тест (A=11):

```
0x0B
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 4. 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 **CLI**.

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 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. `find`.
2. `mkdir`.
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 **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **PlantUML**. 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 **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, ... )
```

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**	**C**

A	B	C
Биты 0—2	Биты 3—5	Биты 6—27
7	Адрес	Константа

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

Тест (A=7, B=1, C=368):

```
0x0F, 0x5C, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—2	Биты 3—5	Биты 6—8	Биты 9—15
1	Адрес	Адрес	Смещение

Размер команды: 5 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=1, B=3, C=2, D=12):

```
0x99, 0x18, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—2	Биты 3—5	Биты 6—37
5	Адрес	Адрес

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

Тест (A=5, B=1, C=82):

```
0x8D, 0x14, 0x00, 0x00, 0x00
```

Бинарная операция: `"<"`

A	B	C	D

A	**B**	**C**	**D**
Биты 0—2	Биты 3—5	Биты 6—8	Биты 9—15
4	Адрес	Адрес	Смещение

Размер команды: 5 байт. Первый операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле D). Второй 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=4, B=7, C=4, D=68):

```
0x3C, 0x89, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "<" над двумя векторами длины 6. Результат записать в новый вектор.

Вариант №29

Задание №1

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

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

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к 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. `chown`.
2. `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 **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.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
// Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
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**	**C**
Биты 0—2	Биты 3—29	Биты 30—35
4	Константа	Адрес

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

Тест (A=4, B=368, C=63):

```
0x84, 0x0B, 0x00, 0xC0, 0x0F
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—8	Биты 9—14
3	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=3, B=18, C=10):

```
0x93, 0x14, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—2	Биты 3—18	Биты 19—24
5	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=5, B=235, C=14):

```
0x5D, 0x07, 0x70, 0x00, 0x00
```

Бинарная операция: `"<="`

A	**B**	**C**
Биты 0—2	Биты 3—8	Биты 9—24
1	Адрес	Адрес

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

Тест (A=1, B=35, C=497):

```
0x19, 0xE3, 0x03, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"<="` над двумя векторами длины 5. Результат записать в первый вектор.

Вариант №30

Задание №1

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

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

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

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

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

1. `uptime`.
2. `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 Python (pip)**. 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.

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.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
(define имя значение)
```

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

```
!(имя 1 +)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.

3. `pow()`.
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 **json**.

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

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—8	Биты 9—30
27	Адрес	Константа

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

Тест (A=27, B=2, C=426):

```
0x5B, 0x54, 0x03, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—4	Биты 5—8	Биты 9—12	Биты 13—21
3	Адрес	Адрес	Смещение

Размер команды: 7 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=3, B=3, C=13, D=286):

```
0x63, 0xDA, 0x23, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—4	Биты 5—8	Биты 9—39
11	Адрес	Адрес

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

Тест (A=11, B=4, C=312):

```
0x8B, 0x70, 0x02, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	B	C	D
Биты 0—4	Биты 5—8	Биты 9—39	Биты 40—48
19	Адрес	Адрес	Смещение

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

Тест (A=19, B=11, C=937, D=390):

```
0x73, 0x53, 0x07, 0x00, 0x00, 0x86, 0x01
```

Тестовая программа

Выполнить поэлементно операцию `sqrt()` над вектором длины 8. 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 **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. `tail`.
2. `head`.
3. `uptime`.

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

Задание №2

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

Конфигурационный файл имеет формат **xml** 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 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
-->
```

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 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—17	Биты 18—43
3	Адрес	Константа

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

Тест (A=3, B=174, C=118):

```
0xE3, 0x0A, 0xD8, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—3	Биты 4—17	Биты 18—31
5	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=5, B=130, C=923):

```
0x25, 0x08, 0x6C, 0x0E, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—3	Биты 4—17	Биты 18—31
12	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=12, B=794, C=760):

```
0xAC, 0x31, 0xE0, 0x0B, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C

A	**B**	**C**
Биты 0—3	Биты 4—17	Биты 18—31
14	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=14, B=125, C=456):

```
0xDE, 0x07, 0x20, 0x07, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 6. Результат записать в новый вектор.

Вариант №32

Задание №1

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

Конфигурационный файл имеет формат **csv** 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. `date`.
2. `clear`.

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

Задание №2

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

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
var имя := значение
```

Вычисление константы на этапе трансляции:

```
[имя]
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—3	Биты 4—13	Биты 14—41
10	Константа	Адрес

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

Тест (A=10, B=117, C=487):

```
0x5A, 0xC7, 0x79, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—3	Биты 4—31	Биты 32—59
0	Адрес	Адрес

Размер команды: 8 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=0, B=741, C=42):

```
0x50, 0x2E, 0x00, 0x00, 0x2A, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C	D
Биты 0—3	Биты 4—19	Биты 20—47	Биты 48—75
3	Смещение	Адрес	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 C) и смещения (поле B).

Тест (A=3, B=841, C=427, D=224):

```
0x93, 0x34, 0xB0, 0x1A, 0x00, 0x00, 0xE0, 0x00, 0x00, 0x00
```

Бинарная операция: `"=="`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—31	Биты 32—59	Биты 60—87
9	Адрес	Адрес	Адрес

Размер команды: 11 байт. Первый операнд: значение в памяти по адресу, которым является поле B. 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 D.

Тест (A=9, B=61, C=25, D=403):

```
0xD9, 0x03, 0x00, 0x00, 0x19, 0x00, 0x00, 0x30, 0x19, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "==" над вектором длины 6 и числом 69. Результат записать в исходный вектор.

Вариант №33

Задание №1

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

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. `tree`.
2. `head`.

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 кода.

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

Ключами командной строки задаются:

- Путь к программе для визуализации графов.

- Путь к анализируемому репозиторию.
- Путь к файлу-результату в виде кода.
- Дата коммитов в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

Входной текст на **учебном конфигурационном языке** принимается из 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:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
def имя = значение
```

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

```
#[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

Загрузка константы

A	B
Биты 0—2	Биты 3—15
5	Константа

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

Тест (A=5, B=760):

```
0xC5, 0x17
```

Чтение значения из памяти

A	B
Биты 0—2	Биты 3—7
4	Смещение

Размер команды: 1 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=4, B=25):

```
0xCC
```

Запись значения в память

A	B
Биты 0—2	Биты 3—27
6	Адрес

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

Тест (A=6, B=799):

```
0xFE, 0x18, 0x00, 0x00
```

Унарная операция: `sgn()`

A	B
Биты 0—2	Биты 3—27

A	**B**
0	Адрес

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

Тест (A=0, B=563):

```
0x98, 0x11, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sgn()` над вектором длины 7. 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 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. `cal`.
2. `mv`.

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.

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

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя = значение;
```

Вычисление константы на этапе трансляции:

```
^[имя]
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—17	Биты 18—43
3	Адрес	Константа

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

Тест (A=3, B=174, C=118):

```
0xE3, 0x0A, 0xD8, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—3	Биты 4—17	Биты 18—31
5	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 поле C.

Тест (A=5, B=130, C=923):

```
0x25, 0x08, 0x6C, 0x0E, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—3	Биты 4—17	Биты 18—31
12	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=12, B=794, C=760):

```
0xAC, 0x31, 0xE0, 0x0B, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C
Биты 0—3	Биты 4—17	Биты 18—31
14	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=14, B=125, C=456):

```
0xDE, 0x07, 0x20, 0x07, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 6. Результат записать в новый вектор.

Вариант №35

Задание №1

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

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

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

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

1. `rm`.
2. `wc`.
3. `mv`.

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 **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

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

- Путь к программе для визуализации графов.

- 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 стандартный вывод.

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
q(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**	**C**
Биты 0—2	Биты 3—15	Биты 16—18
3	Константа	Адрес

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

Тест (A=3, B=456, C=5):

```
0x43, 0x0E, 0x05
```

Чтение значения из памяти

A	B	C
Биты 0—2	Биты 3—27	Биты 28—30
7	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=7, B=563, C=7):

```
0x9F, 0x11, 0x00, 0x70
```

Запись значения в память

A	B	C
Биты 0—2	Биты 3—5	Биты 6—30
4	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=4, B=7, C=985):

```
0x7C, 0xF6, 0x00, 0x00
```

Бинарная операция: умножение

A	B	C
Биты 0—2	Биты 3—27	Биты 28—30
2	Адрес	Адрес

Размер команды: 4 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=2, B=424, C=1):

```
0x42, 0x0D, 0x00, 0x10
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 5 и числом 88. Результат записать в новый вектор.

Вариант №36

Задание №1

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

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 **PlantUML**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- 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 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.

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
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 **yaml**.

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

Загрузка константы

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

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

Тест (A=66, B=863):

```
0xC2, 0xAF, 0x01, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—24
126	Адрес

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

Тест (A=126, B=342):

```
0x7E, 0xAB, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—6	Биты 7—24
57	Адрес

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

Тест (A=57, B=21):

```
0xB9, 0x0A, 0x00, 0x00
```

Бинарная операция: `"!="`

A	**B**
Биты 0—6	Биты 7—24
102	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=102, B=638):

```
0x66, 0x3F, 0x01, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"!="` над вектором длины 8 и числом 161. Результат записать в исходный 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 **zip**. Эмулятор должен работать в режиме **GUI**.

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

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

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. `date`.
2. `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 для **git-репозитория**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен

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

Построить граф зависимостей для коммитов, в узлах которого содержатся сообщения. Граф необходимо строить 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 xml** попадает в стандартный вывод.

Словари:

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

Имена:

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

Значения:

- Числа.
- 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 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—8	Биты 9—37
14	Адрес	Константа

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

Тест (A=14, B=3, C=424):

```
0x3E, 0x50, 0x03, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—39
8	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=8, B=13, C=791):

```
0xD8, 0x2E, 0x06, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—3	Биты 4—19	Биты 20—24	Биты 25—29
2	Смещение	Адрес	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 C) и смещения (поле B).

Тест (A=2, B=727, C=25, D=9):

```
0x72, 0x2D, 0x90, 0x13, 0x00
```

Бинарная операция: `"=="`

A	**B**	**C**
Биты 0—3	Биты 4—8	Биты 9—39
7	Адрес	Адрес

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

Тест (A=7, B=26, C=439):

```
0xA7, 0x6F, 0x03, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"=="` над двумя векторами длины 6. Результат записать в первый 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**.

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. `mkdir`.
2. `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 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 для ветки с заданным именем.

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

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

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
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. `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
Биты 0—7	Биты 8—28
149	Константа

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

Тест (A=149, B=34):

```
0x95, 0x22, 0x00, 0x00
```

Чтение значения из памяти

A	B
Биты 0—7	Биты 8—14
31	Смещение

Размер команды: 2 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=31, B=16):

```
0x1F, 0x10
```

Запись значения в память

A	B
Биты 0—7	Биты 8—25
5	Адрес

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

Тест (A=5, B=466):

```
0x05, 0xD2, 0x01, 0x00
```

Бинарная операция: `">="`

A
Биты 0—7

A
68

Размер команды: 1 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=68):

```
0x44
```

Тестовая программа

Выполнить поэлементно операцию ">=" над двумя векторами 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**.

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. `touch`.
2. `chmod`.

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

Задание №2

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

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.

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
let имя = значение
```

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

```
#{+ имя 1}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

Форматом для файла-лога и файла-результата является **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—26	Биты 27—40
4	Адрес	Константа

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

Тест (A=4, B=354, C=584):

```
0x24, 0x16, 0x00, 0x40, 0x12, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—3	Биты 4—26	Биты 27—41	Биты 42—64
6	Адрес	Смещение	Адрес

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

Тест (A=6, B=26, C=351, D=141):

```
0xA6, 0x01, 0x00, 0xF8, 0x0A, 0x34, 0x02, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—3	Биты 4—26	Биты 27—49
8	Адрес	Адрес

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

Тест (A=8, B=544, C=269):

```
0x08, 0x22, 0x00, 0x68, 0x08, 0x00, 0x00
```

Бинарная операция: `взятие остатка`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—26	Биты 27—49	Биты 50—72
12	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 поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=12, B=429, C=182, D=741):

```
0xDC, 0x1A, 0x00, 0xB0, 0x05, 0x00, 0x94, 0x0B, 0x00, 0x00
```

Тестовая программа

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

Вариант №40

Задание №1

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

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

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

1. `chmod`.
2. `rev`.
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 **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **Graphviz**.

Визуализатор должен выводить результат на экран в виде графического изображения графа.

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

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
имя := значение
```

Вычисление константы на этапе трансляции:

```
^[имя]
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	B
Биты 0—5	Биты 6—32
11	Константа

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

Тест (A=11, B=503):

```
0xCB, 0x7D, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B
Биты 0—5	Биты 6—19
45	Смещение

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

Тест (A=45, B=282):

```
0xAD, 0x46, 0x00
```

Запись значения в память

A	B
Биты 0—5	Биты 6—31
29	Адрес

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

Тест (A=29, B=635):

```
0xDD, 0x9E, 0x00, 0x00
```

Бинарная операция: `побитовое исключающее "или"`

A	B
Биты 0—5	Биты 6—31

A	B
54	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=54, B=75):

```
0xF6, 0x12, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию побитовое исключающее "или" над вектором длины 4 и числом 73. Результат записать в исходный вектор.