Конфигурационное управление

Сборник домашних заданий

Группа 10

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

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
C Это однострочный комментарий
```

Словари:

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

Имена:

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

Значения:

- Числа.

- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
(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—5	Биты 6—13	Биты 14—44
1	Константа	Адрес

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

Тест (A=1, B=202, C=174):

```
0x81, 0xB2, 0x2B, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—5	Биты 6—36	Биты 37—67
61	Адрес	Адрес

Размер команды: 9 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=61, B=122, C=1001):

```
0xBD, 0x1E, 0x00, 0x00, 0x20, 0x7D, 0x00, 0x00, 0x00
```

Запись значения в память

A	B	C
Биты 0—5	Биты 6—36	Биты 37—67
56	Адрес	Адрес

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

Тест (A=56, B=322, C=609):

```
0xB8, 0x50, 0x00, 0x00, 0x20, 0x4C, 0x00, 0x00, 0x00
```

Бинарная операция: `pow()`

A	**B**	**C**	**D**	**E**
Биты 0—5	Биты 6—36	Биты 37—67	Биты 68—98	Биты 99—105
52	Адрес	Адрес	Адрес	Смещение

Размер команды: 14 байт. Первый операнд: значение в памяти по адресу, которым является поле C. Второй операнд: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле D) и смещения (поле E). Результат: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле B.

Тест (A=52, B=275, C=838, D=34, E=39):

```
0xF4, 0x44, 0x00, 0x00, 0xC0, 0x68, 0x00, 0x00, 0x20, 0x02, 0x00, 0x00, 0x38, 0x01
```

Тестовая программа

Выполнить поэлементно операцию `pow()` над двумя векторами длины 4. Результат записать во второй вектор.

Вариант №2

Задание №1

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

Ключами командной строки задаются:

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

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

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

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

1. `whoami`.
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 **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

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

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
# Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

```
{
  имя = значение;
  имя = значение;
  имя = значение;
```

```
  ...
}
```

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
значение -> имя
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—20
7	Константа

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

Тест (A=7, B=711):

```
0xE7, 0x58, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—14
23	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=23, B=146):

```
0x57, 0x12
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—14
24	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=24, B=140):

```
0x98, 0x11
```

Унарная операция: `popcnt()`

A	B
Биты 0—4	Биты 5—14
17	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: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=17, B=192):

```
0x11, 0x18
```

Тестовая программа

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

Вариант №3

Задание №1

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

Ключами командной строки задаются:

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

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

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

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

1. `du`.
2. `history`.

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

Задание №2

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

Зависимости определяются 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 дата, время и автор коммита. Граф необходимо строить только для коммитов ранее заданной даты.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Путь к анализируемому репозиторию.
- Путь к файлу с изображением графа зависимостей.
- Дата коммитов в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

Входной текст на **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке xml** попадает в 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-zA-Z][_a-zA-Z0-9]*
```

Значения:

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

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
set имя = значение
```

Вычисление константного выражения на этапе трансляции (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. `print()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—20
7	Константа

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

Тест (A=7, B=711):

```
0xE7, 0x58, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—14
23	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=23, B=146):

```
0x57, 0x12
```

Запись значения в память

A	B
Биты 0—4	Биты 5—14
24	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=24, B=140):

```
0x98, 0x11
```

Унарная операция: `popcnt()`

A
Биты 0—4
17

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=17):

```
0x11
```

Тестовая программа

Выполнить поэлементно операцию `popcnt()` над вектором длины 6. 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 **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. `cp`.
2. `history`.
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 **языка JavaScript (npm)**. 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.
- 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 json** попадает в стандартный вывод.

Однострочные комментарии:

```
NB. 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:

```
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—20	Биты 21—50
23	Константа	Адрес

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

Тест (A=23, B=140, C=841):

```
0x97, 0x11, 0x20, 0x69, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—34	Биты 35—64	Биты 65—74
18	Адрес	Адрес	Смещение

Размер команды: 10 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=18, B=444, C=192, D=219):

```
0x92, 0x37, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0xB6, 0x01
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—4	Биты 5—34	Биты 35—64	Биты 65—74
31	Адрес	Адрес	Смещение

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

Тест (A=31, B=849, C=479, D=763):

```
0x3F, 0x6A, 0x00, 0x00, 0xF8, 0x0E, 0x00, 0x00, 0xF6, 0x05
```

Бинарная операция: умножение

A	**B**	**C**	**D**
Биты 0—4	Биты 5—34	Биты 35—64	Биты 65—94
22	Адрес	Адрес	Адрес

Размер команды: 12 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=22, B=569, C=681, D=530):

```
0x36, 0x47, 0x00, 0x00, 0x48, 0x15, 0x00, 0x00, 0x24, 0x04, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 5 и числом 45. Результат записать в новый вектор.

Вариант №5

Задание №1

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

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

Задание №2

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

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

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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, ... >>
```

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
var имя значение
```

Вычисление константы на этапе трансляции:

```
#(имя)
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—20	Биты 21—27
23	Константа	Адрес

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

Тест (A=23, B=140, C=82):

```
0x97, 0x11, 0x40, 0x0A
```

Чтение значения из памяти

A	B	C	D
Биты 0—4	Биты 5—11	Биты 12—18	Биты 19—28
18	Адрес	Адрес	Смещение

Размер команды: 4 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=18, B=24, C=65, D=219):

```
0x12, 0x13, 0xDC, 0x06
```

Запись значения в память

A	B	C	D
Биты 0—4	Биты 5—11	Биты 12—18	Биты 19—28
31	Адрес	Адрес	Смещение

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

Тест (A=31, B=112, C=109, D=763):

```
0x1F, 0xDE, 0xDE, 0x17
```

Бинарная операция: умножение

A	B	C	D
Биты 0—4	Биты 5—34	Биты 35—41	Биты 42—48
22	Адрес	Адрес	Адрес

Размер команды: 7 байт. Первый операнд: регистр по адресу, которым является поле D. Второй операнд: значение в памяти по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=22, B=569, C=16, D=5):

```
0x36, 0x47, 0x00, 0x00, 0x80, 0x14, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 5 и числом 45. Результат записать в новый вектор.

Вариант №6

Задание №1

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

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

- Путь к архиву виртуальной файловой системы.

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

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

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу-результату в виде кода.

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.

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
var имя := значение;
```

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

```
?[имя + 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `ord()`.
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 **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—31	Биты 32—50
5	Адрес	Константа

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

Тест (A=5, B=229, C=979):

```
0x55, 0x0E, 0x00, 0x00, 0xD3, 0x03, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—31	Биты 32—40	Биты 41—68
2	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 D) и смещения (поле C). Результат: значение в памяти по адресу, которым является поле B.

Тест (A=2, B=92, C=4, D=106):

```
0xC2, 0x05, 0x00, 0x00, 0x04, 0xD4, 0x00, 0x00, 0x00
```

Запись значения в память

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

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

Тест (A=6, B=970, C=629):

```
0xA6, 0x3C, 0x00, 0x00, 0x75, 0x02, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C	D
Биты 0—3	Биты 4—12	Биты 13—40	Биты 41—68
10	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=10, B=226, C=178, D=487):

```
0x2A, 0x4E, 0x16, 0x00, 0x00, 0xCE, 0x03, 0x00, 0x00
```

Тестовая программа

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

Вариант №7

Задание №1

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

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

1. `chmod`.
2. `echo`.
3. `cal`.

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

Задание №2

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

Зависимости определяются 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.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя is значение
```

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

```
![имя + 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

Загрузка константы

A	B
Биты 0—6	Биты 7—34
87	Константа

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

Тест (A=87, B=535):

```
0xD7, 0x0B, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	B
Биты 0—6	Биты 7—20
81	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=81, B=531):

```
0xD1, 0x09, 0x01, 0x00, 0x00
```

Запись значения в память

A	B
Биты 0—6	Биты 7—19
25	Адрес

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

Тест (A=25, B=9):

```
0x99, 0x04, 0x00, 0x00, 0x00
```

Унарная операция: `sgn()`

A	B	C

A	**B**	**C**
Биты 0—6	Биты 7—19	Биты 20—33
9	Адрес	Смещение

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

Тест (A=9, B=894, C=245):

```
0x09, 0xBF, 0x51, 0x0F, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `sgn()` над вектором длины 5. 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 **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. `whoami`.
2. `pwd`.
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 для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

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

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
* Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя is значение
```

Вычисление 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. `chr()`.
6. `mod()`.

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

Задание №4

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

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

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

Форматом для файла-лога и файла-результата является **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—31	Биты 32—50
5	Адрес	Константа

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

Тест (A=5, B=229, C=979):

```
0x55, 0x0E, 0x00, 0x00, 0xD3, 0x03, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—31	Биты 32—40	Биты 41—68
2	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 D) и смещения (поле C). Результат: значение в памяти по адресу, которым является поле B.

Тест (A=2, B=92, C=4, D=106):

```
0xC2, 0x05, 0x00, 0x00, 0x04, 0xD4, 0x00, 0x00, 0x00
```

Запись значения в память

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

A	B	C
6	Адрес	Адрес

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

Тест (A=6, B=970, C=629):

```
0xA6, 0x3C, 0x00, 0x00, 0x75, 0x02, 0x00, 0x00
```

Унарная операция: `abs()`

A	B	C	D
Биты 0—3	Биты 4—12	Биты 13—40	Биты 41—68
10	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=10, B=226, C=178, D=487):

```
0x2A, 0x4E, 0x16, 0x00, 0x00, 0xCE, 0x03, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `abs()` над вектором длины 4. 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 **json** и содержит:

- Имя пользователя для показа в приглашении к вводу.
- Имя компьютера для показа в приглашении к вводу.
- Путь к архиву виртуальной файловой системы.
- Путь к лог-файлу.
- 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. `wc`.
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 для **git-репозитория**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Построить граф зависимостей для коммитов, в узлах которого находятся списки файлов и папок. Граф необходимо строить только для коммитов позже заданной даты.

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
global имя = значение
```

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

```
?[имя 1 +]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `concat()`.
3. `max()`.

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

Задание №4

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

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—22	Биты 23—25
6	Константа	Адрес

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

Тест (A=6, B=464, C=2):

```
0x06, 0x74, 0x00, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—8	Биты 9—27
56	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=56, B=5, C=1014):

```
0x78, 0xED, 0x07, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—5	Биты 6—8	Биты 9—11

A	B	C
51	Адрес	Адрес

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

Тест (A=51, B=5, C=0):

```
0x73, 0x01, 0x00, 0x00, 0x00, 0x00
```

Бинарная операция: "<"

A	B	C
Биты 0—5	Биты 6—8	Биты 9—27
17	Адрес	Адрес

Размер команды: 6 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=17, B=3, C=762):

```
0xD1, 0xF4, 0x05, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "<" над двумя векторами длины 8. Результат записать в новый вектор.

Вариант №10

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `head`.
2. `who`.
3. `touch`.

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

Задание №2

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

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

Построить граф зависимостей для коммитов, в узлах которого содержатся номера коммитов в хронологическом порядке. Граф необходимо строить только для 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 с изображением графа зависимостей.
- Файл с заданным именем в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.

- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
(def имя значение);
```

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

```
^{имя + 1}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Загрузка константы

A	**B**
Биты 0—7	Биты 8—21
118	Константа

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

Тест (A=118, B=894):

```
0x76, 0x7E, 0x03, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—23
231	Адрес

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

Тест (A=231, B=290):

```
0xE7, 0x22, 0x01, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—23
238	Адрес

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

Тест (A=238, B=229):

```
0xEE, 0xE5, 0x00, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—7	Биты 8—23	Биты 24—39
132	Адрес	Адрес

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

Тест (A=132, B=604, C=92):

```
0x84, 0x5C, 0x02, 0x5C, 0x00
```

Тестовая программа

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

Вариант №11

Задание №1

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

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

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

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

1. `touch`.
2. `date`.
3. `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 JavaScript (npm)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

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

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

- Путь к файлу с изображением графа зависимостей.

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.
- Строки.
- Массивы.

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. `ord()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—27
16	Константа

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

Тест (A=16, B=599):

```
0xF0, 0x4A, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—11
7	Смещение

Размер команды: 4 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=43):

```
0x67, 0x05, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—11
23	Смещение

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

Тест (A=23, B=124):

```
0x97, 0x0F, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—4	Биты 5—30
10	Адрес

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

Тест (A=10, B=691):

```
0x6A, 0x56, 0x00, 0x00
```

Тестовая программа

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

Вариант №12

Задание №1

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

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

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

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

1. `cp`.
2. `uptime`.

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

Задание №2

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

```
REM Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
(def имя значение);
```

Вычисление константы на этапе трансляции:

|имя|

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—27
16	Константа

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

Тест (A=16, B=599):

```
0xF0, 0x4A, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—11
7	Смещение

Размер команды: 4 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=43):

```
0x67, 0x05, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—11
23	Смещение

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

Тест (A=23, B=124):

```
0x97, 0x0F, 0x00, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—4	Биты 5—30
10	Адрес

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

Тест (A=10, B=691):

```
0x6A, 0x56, 0x00, 0x00
```

Тестовая программа

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

Вариант №13

Задание №1

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

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

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

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

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

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

1. `cal`.
2. `pwd`.

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

Задание №2

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

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

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

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 toml** 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-z]+
```

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
var имя значение;
```

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

```
^[+ имя 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—26	Биты 27—33
0	Константа	Адрес

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

Тест (A=0, B=482, C=22):

```
0x10, 0x0F, 0x00, 0xB0, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—34
7	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=7, B=45, C=310):

```
0x6F, 0xD9, 0x04, 0x00, 0x00
```

Запись значения в память

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

A	**B**	**C**	**D**
Биты 0—2	Биты 3—9	Биты 10—16	Биты 17—30
1	Адрес	Адрес	Смещение

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

Тест (A=1, B=95, C=33, D=543):

```
0xF9, 0x86, 0x3E, 0x04
```

Бинарная операция: `побитовое "или"`

A	**B**	**C**	**D**
Биты 0—2	Биты 3—9	Биты 10—16	Биты 17—30
3	Адрес	Адрес	Смещение

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

Тест (A=3, B=43, C=108, D=742):

```
0x5B, 0xB1, 0xCD, 0x05
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "или"` над двумя векторами длины 8. Результат записать в новый вектор.

Вариант №14

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **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. `rev`.
2. `du`.
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 **языка Java (Maven)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат на экран в виде кода.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу-результату в виде кода.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
// Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

```
array( 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 **csv**.

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

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—26	Биты 27—33
0	Константа	Адрес

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

Тест (A=0, B=482, C=22):

```
0x10, 0x0F, 0x00, 0xB0, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—9	Биты 10—34
7	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=7, B=45, C=310):

```
0x6F, 0xD9, 0x04, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—9	Биты 10—16	Биты 17—30
1	Адрес	Адрес	Смещение

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

Тест (A=1, B=95, C=33, D=543):

```
0xF9, 0x86, 0x3E, 0x04
```

Бинарная операция: побитовое "или"

A	**B**	**C**	**D**
Биты 0—2	Биты 3—9	Биты 10—16	Биты 17—30
3	Адрес	Адрес	Смещение

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

Тест (A=3, B=43, C=108, D=742):

```
0x5B, 0xB1, 0xCD, 0x05
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "или"` над двумя векторами длины 8. Результат записать в новый вектор.

Вариант №15

Задание №1

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

Ключами командной строки задаются:

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

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

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

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

1. `find`.
2. `tac`.
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 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 для тега с заданным именем.

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
global имя = значение;
```

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

```
@[+ имя 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

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

Загрузка константы

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

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

Тест (A=49, B=909):

```
0x71, 0xE3, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—5	Биты 6—33
41	Адрес

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

Тест (A=41, B=168):

```
0x29, 0x2A, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**

A	**B**
Биты 0—5	Биты 6—33
23	Адрес

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

Тест (A=23, B=394):

```
0x97, 0x62, 0x00, 0x00, 0x00
```

Унарная операция: `унарный минус`

A
Биты 0—5
42

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

Тест (A=42):

```
0x2A
```

Тестовая программа

Выполнить поэлементно операцию `унарный минус` над вектором длины 4. 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 **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. `rev`.
2. `clear`.
3. `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 по имени пакета **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде кода.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу-результату в виде кода.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

Входной текст на **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
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

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

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
def имя := значение
```

Вычисление константы на этапе трансляции:

```
!{имя}
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

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

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

Тест (A=53, B=604):

```
0x35, 0x2E, 0x01, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—34
110	Адрес

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

Тест (A=110, B=106):

```
0x6E, 0x35, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—6	Биты 7—34
71	Адрес

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

Тест (A=71, B=713):

```
0xC7, 0x64, 0x01, 0x00, 0x00
```

Бинарная операция: `"=="`

A	**B**
Биты 0—6	Биты 7—34
56	Адрес

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

Тест (A=56, B=4):

```
0x38, 0x02, 0x00, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию "==" над вектором длины 6 и числом 43. Результат записать в исходный вектор.

Вариант №17

Задание №1

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

Ключами командной строки задаются:

- Имя 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. `uniq`.
2. `tree`.
3. `tail`.

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

Задание №2

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

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

Построить граф зависимостей для коммитов, в узлах которого содержатся дата, время и автор коммита. Граф необходимо строить для ветки с заданным именем.

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

Вычисление константы на этапе трансляции:

```
$имя$
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

Форматом для файла-лога и файла-результата является **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—12	Биты 13—16
15	Константа	Адрес

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

Тест (A=15, B=160, C=3):

```
0x0F, 0x6A, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—17
0	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=0, B=3, C=300):

```
0x30, 0x2C, 0x01
```

Запись значения в память

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

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

Тест (A=12, B=12, C=7):

```
0xCC, 0x07
```

Бинарная операция: `"<"`

A	**B**	**C**
Биты 0—3	Биты 4—13	Биты 14—17
10	Адрес	Адрес

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

Тест (A=10, B=527, C=10):

```
0xFA, 0xA0, 0x02
```

Тестовая программа

Выполнить поэлементно операцию "<" над двумя векторами длины 5. Результат записать в новый вектор.

Вариант №18

Задание №1

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

Ключами командной строки задаются:

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

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

1. `rmdir`.
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 по имени пакета **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **Graphviz**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
|| Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

Вычисление константы на этапе трансляции:

```
$имя$
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—4	Биты 5—21
30	Константа

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

Тест (A=30, B=684):

```
0x9E, 0x55, 0x00
```

Чтение значения из памяти

A	**B**

A	B
Биты 0—4	Биты 5—17
0	Адрес

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

Тест (A=0, B=327):

```
0xE0, 0x28, 0x00
```

Запись значения в память

A	B
Биты 0—4	Биты 5—17
8	Адрес

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

Тест (A=8, B=168):

```
0x08, 0x15, 0x00
```

Бинарная операция: ">"

A	B
Биты 0—4	Биты 5—17
20	Адрес

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

Тест (A=20, B=679):

```
0xF4, 0x54, 0x00
```

Тестовая программа

Выполнить поэлементно операцию ">" над вектором длины 6 и числом 67. Результат записать в новый вектор.

Вариант №19

Задание №1

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

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.
- Путь к стартовому скрипту.

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

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

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

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

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

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

Входной текст на **языке toml** 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:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
const имя = значение
```

Вычисление константы на этапе трансляции:

```
?(имя)
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	B
Биты 0—2	Биты 3—15
1	Константа

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

Тест (A=1, B=871):

```
0x39, 0x1B
```

Чтение значения из памяти

A
Биты 0—2
5

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

Тест (A=5):

```
0x05
```

Запись значения в память

A	B
Биты 0—2	Биты 3—19
0	Адрес

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

Тест (A=0, B=858):

```
0xD0, 0x1A, 0x00
```

Унарная операция: `унарный минус`

A
Биты 0—2

A
3

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

Тест (A=3):

```
0x03
```

Тестовая программа

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

Вариант №20

Задание №1

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

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

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу с изображением графа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
|| Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
имя <- значение
```

Вычисление константы на этапе трансляции:

```
.(имя).
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**

A	B	C
Биты 0—2	Биты 3—24	Биты 25—49
2	Адрес	Константа

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

Тест (A=2, B=80, C=886):

```
0x82, 0x02, 0x00, 0xEC, 0x06, 0x00, 0x00
```

Чтение значения из памяти

A	B	C
Биты 0—2	Биты 3—24	Биты 25—46
7	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=7, B=356, C=565):

```
0x27, 0x0B, 0x00, 0x6A, 0x04, 0x00
```

Запись значения в память

A	B	C	D
Биты 0—2	Биты 3—24	Биты 25—35	Биты 36—57
5	Адрес	Смещение	Адрес

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

Тест (A=5, B=487, C=309, D=838):

```
0x3D, 0x0F, 0x00, 0x6A, 0x62, 0x34, 0x00, 0x00
```

Бинарная операция: `">"`

A	**B**	**C**	**D**	**E**
Биты 0—2	Биты 3—24	Биты 25—46	Биты 47—68	Биты 69—79
1	Адрес	Адрес	Адрес	Смещение

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

Тест (A=1, B=76, C=876, D=756, E=332):

```
0x61, 0x02, 0x00, 0xD8, 0x06, 0x00, 0x7A, 0x01, 0x80, 0x29
```

Тестовая программа

Выполнить поэлементно операцию ">" над двумя векторами длины 5. Результат записать в новый вектор.

Вариант №21

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `wc`.
2. `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 **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 **xml** и содержит:

- 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 стандартный вывод.

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Массивы:

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

Словари:

```
struct {
  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 := 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—32	Биты 33—37

A	B	C
5	Константа	Адрес

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

Тест (A=5, B=506, C=28):

```
0xA5, 0x1F, 0x00, 0x00, 0x38
```

Чтение значения из памяти

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

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

Тест (A=1, B=31, C=5):

```
0xF1, 0x0B
```

Запись значения в память

A	B	C	D
Биты 0—3	Биты 4—8	Биты 9—13	Биты 14—28
9	Адрес	Адрес	Смещение

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

Тест (A=9, B=27, C=27, D=975):

```
0xB9, 0xF7, 0xF3, 0x00
```

Бинарная операция: сложение

A	B	C
Биты 0—3	Биты 4—22	Биты 23—27

A	**B**	**C**
0	Адрес	Адрес

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

Тест (A=0, B=856, C=22):

```
0x80, 0x35, 0x00, 0x0B
```

Тестовая программа

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

Вариант №22

Задание №1

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

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

- 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. `rev`.
2. `find`.
3. `date`.

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

Задание №2

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

Зависимости определяются по имени пакета **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Mermaid**.

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

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

- Путь к программе для визуализации графов.
- 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 **учебном конфигурационном языке** принимается из стандартного ввода. Выходной текст на **языке xml** попадает в стандартный вывод.

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
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 **xml**.

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

Загрузка константы

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

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

Тест (A=6, B=582):

```
0x36, 0x12, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—2	Биты 3—14
3	Адрес

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

Тест (A=3, B=528):

```
0x83, 0x10
```

Запись значения в память

A	**B**
Биты 0—2	Биты 3—14
5	Адрес

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

Тест (A=5, B=844):

```
0x65, 0x1A
```

Унарная операция: побитовое "не"

A
Биты 0—2
1

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

Тест (A=1):

```
0x01
```

Тестовая программа

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

Вариант №23

Задание №1

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

Ключами командной строки задаются:

- Путь к архиву виртуальной файловой системы.

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

1. `cp`.
2. `find`.
3. `tree`.

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

Задание №2

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

Зависимости определяются по имени пакета **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 json** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

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 **xml**.

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

Загрузка константы

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

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

Тест (A=7, B=582):

```
0x37, 0x12, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**

A	B
Биты 0—2	Биты 3—14
6	Адрес

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

Тест (A=6, B=528):

```
0x86, 0x10
```

Запись значения в память

A	B
Биты 0—2	Биты 3—14
5	Адрес

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

Тест (A=5, B=844):

```
0x65, 0x1A
```

Унарная операция: `побитовое "не"`

A	B	C
Биты 0—2	Биты 3—14	Биты 15—26
1	Адрес	Адрес

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

Тест (A=1, B=243, C=385):

```
0x99, 0x87, 0xC0, 0x00
```

Тестовая программа

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

Вариант №24

Задание №1

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

Ключами командной строки задаются:

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

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

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

1. `touch`.
2. `chmod`.

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

Задание №2

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

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

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

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

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
' Это однострочный 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. `max()`.
3. `pow()`.

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

Задание №4

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

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

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

Форматом для файла-лога и файла-результата является **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—28	Биты 29—31
98	Константа	Адрес

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

Тест (A=98, B=825, C=3):

```
0xE2, 0x9C, 0x01, 0x60
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—9	Биты 10—36
109	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=109, B=2, C=742):

```
0x6D, 0x99, 0x0B, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—6	Биты 7—9	Биты 10—25	Биты 26—28
17	Адрес	Смещение	Адрес

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

Тест (A=17, B=5, C=843, D=0):

```
0x91, 0x2E, 0x0D, 0x00
```

Унарная операция: `abs()`

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

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

Тест (A=57, B=2, C=3):

```
0x39, 0x0D
```

Тестовая программа

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

Вариант №25

Задание №1

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

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 **csv** и содержит все действия во время последнего сеанса работы с эмулятором.

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

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

1. `uname`.
2. `rmdir`.

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.

Конфигурационный файл имеет формат **yaml** 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 **языке json** 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:

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

Имена:

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

Значения:

- Числа.
- Словари.

Объявление константы на этапе трансляции:

```
имя <- значение
```

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

```
?(имя 1 +)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `sqrt()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	B	C
Биты 0—6	Биты 7—36	Биты 37—41
58	Константа	Адрес

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

Тест (A=58, B=99, C=20):

```
0xBA, 0x31, 0x00, 0x00, 0x80, 0x02
```

Чтение значения из памяти

A	B	C
Биты 0—6	Биты 7—11	Биты 12—21
3	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=3, B=23, C=258):

```
0x83, 0x2B, 0x10
```

Запись значения в память

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

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

Тест (A=30, B=5, C=18, D=790):

```
0x9E, 0x22, 0x2D, 0x06
```

Бинарная операция: `побитовое "или"`

A	**B**	**C**	**D**	**E**
Биты 0—6	Биты 7—11	Биты 12—26	Биты 27—31	Биты 32—36
124	Адрес	Смещение	Адрес	Адрес

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

Тест (A=124, B=1, C=56, D=25, E=27):

```
0xFC, 0x80, 0x03, 0xC8, 0x1B
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "или"` 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 **CLI**.

Ключами командной строки задаются:

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

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

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

1. `history`.
2. `mkdir`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 дата, время и автор коммита. Граф необходимо строить только для тех коммитов, где фигурирует файл с заданным хеш-значением.

Ключами командной строки задаются:

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

Однострочные комментарии:

```
// Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
@"Это строка"
```

Объявление константы на этапе трансляции:

```
let имя = значение;
```

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

```
{имя 1 +}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. Умножение.
4. `concat()`.
5. `ord()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—34	Биты 35—49
10	Адрес	Константа

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

Тест (A=10, B=183, C=593):

```
0x7A, 0x0B, 0x00, 0x00, 0x88, 0x12, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—34	Биты 35—65
6	Адрес	Адрес

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

Тест (A=6, B=568, C=38):

```
0x86, 0x23, 0x00, 0x00, 0x30, 0x01, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—3	Биты 4—34	Биты 35—65	Биты 66—70
12	Адрес	Адрес	Смещение

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

Тест (A=12, B=813, C=865, D=18):

```
0xDC, 0x32, 0x00, 0x00, 0x08, 0x1B, 0x00, 0x00, 0x48
```

Унарная операция: `abs()`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—34	Биты 35—39	Биты 40—70
3	Адрес	Смещение	Адрес

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

Тест (A=3, B=766, C=22, D=521):

```
0xE3, 0x2F, 0x00, 0x00, 0xB0, 0x09, 0x02, 0x00, 0x00
```

Тестовая программа

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

Вариант №27

Задание №1

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

Ключами командной строки задаются:

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

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

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

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

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

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

Задание №2

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

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

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- 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-z][a-z0-9_]*
```

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
global имя = значение;
```

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

```
$имя 1 +$
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**

A	B
Биты 0—6	Биты 7—36
66	Константа

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

Тест (A=66, B=307):

```
0xC2, 0x99, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	B
Биты 0—6	Биты 7—16
81	Адрес

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

Тест (A=81, B=375):

```
0xD1, 0xBB, 0x00
```

Запись значения в память

A	B
Биты 0—6	Биты 7—21
53	Смещение

Размер команды: 3 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=53, B=99):

```
0xB5, 0x31, 0x00
```

Унарная операция: `sgn()`

A	B	C
Биты 0—6	Биты 7—21	Биты 22—31

A	**B**	**C**
101	Смещение	Адрес

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

Тест (A=101, B=449, C=243):

```
0xE5, 0xE0, 0xC0, 0x3C
```

Тестовая программа

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

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

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

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

1. `find`.
2. `cat`.
3. `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 **платформы .NET (nupkg)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде графического изображения графа.

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

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

Однострочные комментарии:

```
" Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
имя: значение
```

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

```
|имя + 1|
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `print()`.

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—2	Биты 3—32
7	Константа

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

Тест (A=7, B=582):

```
0x37, 0x12, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—2
6

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

Тест (A=6):

```
0x06
```

Запись значения в память

A	**B**
Биты 0—2	Биты 3—14
5	Адрес

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

Тест (A=5, B=99):

```
0x1D, 0x03
```

Унарная операция: побитовое "не"

A	**B**	**C**
Биты 0—2	Биты 3—14	Биты 15—26
1	Адрес	Адрес

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

Тест (A=1, B=449, C=243):

```
0x09, 0x8E, 0x79, 0x00
```

Тестовая программа

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

Вариант №29

Задание №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата **zip**. Эмулятор должен работать в режиме **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. `rm`.
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 **Mermaid**. 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 xml** принимается из стандартного ввода. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
:: Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя is значение
```

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

```
?(+ имя 1)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `print()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—2	Биты 3—11
0	Константа

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

Тест (A=0, B=137):

```
0x48, 0x04
```

Чтение значения из памяти

A	**B**
Биты 0—2	Биты 3—20
6	Адрес

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

Тест (A=6, B=252):

```
0xE6, 0x07, 0x00
```

Запись значения в память

A	**B**
Биты 0—2	Биты 3—16
7	Смещение

Размер команды: 3 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=7, B=206):

```
0x77, 0x06, 0x00
```

Бинарная операция: `"=="`

A	**B**
Биты 0—2	Биты 3—16
1	Смещение

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

Тест (A=1, B=974):

```
0x71, 0x1E, 0x00
```

Тестовая программа

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

Вариант №30

Задание №1

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

Ключами командной строки задаются:

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

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

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

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

1. `pwd`.
2. `tree`.
3. `echo`.

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

Задание №2

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

Зависимости определяются для **git-репозитория**. Для описания графа зависимостей используется представление **Mermaid**. 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 **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 **языке json** принимается из файла, путь к которому задан ключом командной строки. Выходной текст на **учебном конфигурационном языке** попадает в стандартный вывод.

Однострочные комментарии:

```
REM Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.

- Строки.
- Массивы.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
var имя = значение
```

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

```
${+ имя 1}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. `ord()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—31	Биты 32—38
25	Константа	Адрес

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

Тест (A=25, B=877, C=78):

```
0x59, 0xDB, 0x00, 0x00, 0x4E
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—28	Биты 29—35
4	Адрес	Адрес

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

Тест (A=4, B=1004, C=32):

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

Запись значения в память

A	**B**	**C**	**D**
Биты 0—5	Биты 6—15	Биты 16—22	Биты 23—29
10	Смещение	Адрес	Адрес

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

Тест (A=10, B=662, C=109, D=93):

```
0x8A, 0xA5, 0xED, 0x2E
```

Бинарная операция: `"=="`

A	**B**	**C**	**D**
Биты 0—5	Биты 6—12	Биты 13—19	Биты 20—42
20	Адрес	Адрес	Адрес

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

Тест (A=20, B=44, C=116, D=798):

```
0x14, 0x8B, 0xEE, 0x31, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"=="` над двумя векторами 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 **zip**. Эмулятор должен работать в режиме **CLI**.

Ключами командной строки задаются:

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

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

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

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

1. `mv`.
2. `whoami`.
3. `pwd`.

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

Задание №2

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.

- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
(define имя значение)
```

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

```
$[+ имя 1]
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

1. Сложение.
2. Вычитание.
3. `min()`.
4. `abs()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—5	Биты 6—31
31	Константа

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

Тест (A=31, B=315):

```
0xDF, 0x4E, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—5	Биты 6—15
9	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=9, B=274):

```
0x89, 0x44
```

Запись значения в память

A	**B**
Биты 0—5	Биты 6—15
13	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=13, B=466):

```
0x8D, 0x74
```

Унарная операция: побитовое "не"

A	B
Биты 0—5	Биты 6—28
7	Адрес

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

Тест (A=7, B=259):

```
0xC7, 0x40, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `побитовое "не"` над вектором длины 4. 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 **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 **xml** и содержит все действия во время последнего сеанса работы с эмулятором. Для каждого действия указан пользователь.

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

1. `who`.
2. `du`.
3. `rmdir`.

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

Задание №2

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

Ключами командной строки задаются:

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

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
*> Это однострочный комментарий
```

Многострочные комментарии:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
[[Это строка]]
```

Объявление константы на этапе трансляции:

```
имя := значение
```

Вычисление константы на этапе трансляции:

```
$имя$
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—31	Биты 32—54
25	Константа	Адрес

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

Тест (A=25, B=877, C=839):

```
0x59, 0xDB, 0x00, 0x00, 0x47, 0x03, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—28	Биты 29—51
4	Адрес	Адрес

Размер команды: 7 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=4, B=1004, C=206):

```
0x04, 0xFB, 0x00, 0xC0, 0x19, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—5	Биты 6—15	Биты 16—38	Биты 39—61
10	Смещение	Адрес	Адрес

Размер команды: 8 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=10, B=662, C=724, D=439):

```
0x8A, 0xA5, 0xD4, 0x02, 0x80, 0xDB, 0x00, 0x00
```

Бинарная операция: `"=="`

A	**B**	**C**	**D**
Биты 0—5	Биты 6—28	Биты 29—51	Биты 52—74
20	Адрес	Адрес	Адрес

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

Тест (A=20, B=878, C=832, D=798):

```
0x94, 0xDB, 0x00, 0x00, 0x68, 0x00, 0xE0, 0x31, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"=="` над двумя векторами длины 8. Результат записать в новый вектор.

Вариант №33

Задание №1

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

Ключами командной строки задаются:

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

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

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

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

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.
- Путь к файлу-результату в виде кода.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
' Это однострочный комментарий
```

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Строки.
- Массивы.

- Словари.

Строки:

```
q(Это строка)
```

Объявление константы на этапе трансляции:

```
const имя = значение;
```

Вычисление константы на этапе трансляции:

```
|имя|
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	B
Биты 0—5	Биты 6—31
31	Константа

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

Тест (A=31, B=315):

```
0xDF, 0x4E, 0x00, 0x00
```

Чтение значения из памяти

A	B
Биты 0—5	Биты 6—15
9	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=9, B=274):

```
0x89, 0x44
```

Запись значения в память

A	B
Биты 0—5	Биты 6—15
13	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=13, B=466):

```
0x8D, 0x74
```

Унарная операция: `побитовое "не"`

A	B

A	**B**
Биты 0—5	Биты 6—28
7	Адрес

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

Тест (A=7, B=259):

```
0xC7, 0x40, 0x00, 0x00
```

Тестовая программа

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

Вариант №34

Задание №1

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

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

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

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

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

1. `uptime`.
2. `tac`.
3. `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 **ОС Alpine Linux (apk)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- 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 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:

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

Имена:

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

Значения:

- Числа.

- Строки.
- Массивы.
- Словари.

Строки:

```
'Это строка'
```

Объявление константы на этапе трансляции:

```
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. `sqrt()`.

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—11	Биты 12—19
117	Адрес	Константа

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

Тест (A=117, B=12, C=65):

```
0x75, 0x16, 0x04, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—11	Биты 12—25
81	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=81, B=28, C=880):

```
0x51, 0x0E, 0x37, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—11	Биты 12—25
31	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=31, B=27, C=833):

```
0x9F, 0x1D, 0x34, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—6	Биты 7—11	Биты 12—16
118	Адрес	Адрес

Размер команды: 4 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=118, B=9, C=18):

```
0xF6, 0x24, 0x01, 0x00
```

Тестовая программа

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

Вариант №35

Задание №1

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

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

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

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

1. `tree`.
2. `pwd`.
3. `rm`.

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

Задание №2

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

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

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу-результату в виде кода.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
C Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя = значение;
```

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

```
|имя + 1|
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Задание №4

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

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

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

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

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

Загрузка константы

A	**B**
Биты 0—6	Биты 7—31
114	Константа

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

Тест (A=114, B=355):

```
0xF2, 0xB1, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—20
106	Адрес

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

Тест (A=106, B=530):

```
0x6A, 0x09, 0x01, 0x00
```

Запись значения в память

A
Биты 0—6
10

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=10):

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

Бинарная операция: умножение

A	**B**
Биты 0—6	Биты 7—13
101	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=101, B=2):

```
0x65, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 7 и числом 163. Результат записать в исходный вектор.

Вариант №36

Задание №1

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

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. `uniq`.
2. `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 **языка JavaScript (npm)**. Для описания графа зависимостей используется представление **Mermaid**. Визуализатор должен выводить результат в виде сообщения об успешном выполнении и сохранять граф в файле формата png.

Ключами командной строки задаются:

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу с изображением графа зависимостей.
- Максимальная глубина анализа зависимостей.
- URL-адрес репозитория.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Массивы:

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

Имена:

```
[a-zA-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**
Биты 0—4	Биты 5—14
11	Константа

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

Тест (A=11, B=22):

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

Чтение значения из памяти

A	**B**

A	B
Биты 0—4	Биты 5—14
2	Смещение

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

Тест (A=2, B=455):

```
0xE2, 0x38, 0x00, 0x00, 0x00
```

Запись значения в память

A	B
Биты 0—4	Биты 5—34
26	Адрес

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

Тест (A=26, B=42):

```
0x5A, 0x05, 0x00, 0x00, 0x00
```

Унарная операция: `унарный минус`

A	B
Биты 0—4	Биты 5—34
4	Адрес

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

Тест (A=4, B=794):

```
0x44, 0x63, 0x00, 0x00, 0x00
```

Тестовая программа

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

Вариант №37

Задание №1

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

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. `chown`.
2. `echo`.
3. `cp`.

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

Задание №2

Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние 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 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 с изображением графа зависимостей.
- Файл с заданным хеш-значением в репозитории.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

Входной текст на **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-zA-Z][_a-zA-Z0-9]*
```

Значения:

- Числа.
- Строки.
- Словари.

Строки:

```
"Это строка"
```

Объявление константы на этапе трансляции:

```
имя <- значение;
```

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

```
?(+ имя 1)
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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—31
114	Константа

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

Тест (A=114, B=355):

```
0xF2, 0xB1, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—20
106	Адрес

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

Тест (A=106, B=530):

```
0x6A, 0x09, 0x01, 0x00
```

Запись значения в память

A
Биты 0—6
10

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=10):

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

Бинарная операция: умножение

A	**B**
Биты 0—6	Биты 7—13
101	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=101, B=2):

```
0x65, 0x01, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию умножение над вектором длины 7 и числом 163. Результат записать в исходный вектор.

Вариант №38

Задание №1

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

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

Многострочные комментарии:

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

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
var имя значение
```

Вычисление константы на этапе трансляции:

```
.{имя}.
```

Результатом вычисления константного выражения является значение.

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

Задание №4

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

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

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

Форматом для файла-лога и файла-результата является **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—23
83	Адрес	Константа

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

Тест (A=83, B=6, C=786):

```
0x53, 0x93, 0x18, 0x00, 0x00
```

Чтение значения из памяти

A	B	C	D
Биты 0—6	Биты 7—10	Биты 11—14	Биты 15—29
31	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=31, B=4, C=8, D=152):

```
0x1F, 0x42, 0x4C, 0x00, 0x00
```

Запись значения в память

A	B	C	D
Биты 0—6	Биты 7—21	Биты 22—25	Биты 26—29
58	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=58, B=721, C=7, D=9):

```
0xBA, 0x68, 0xC1, 0x25, 0x00
```

Бинарная операция: `"<="`

A	B	C	D	E
Биты 0—6	Биты 7—21	Биты 22—25	Биты 26—29	Биты 30—33
71	Смещение	Адрес	Адрес	Адрес

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

Тест (A=71, B=304, C=6, D=13, E=15):

```
0x47, 0x98, 0x80, 0xF5, 0x03
```

Тестовая программа

Выполнить поэлементно операцию `"<="` над вектором длины 4 и числом 10. 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 **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. `chmod`.
2. `cal`.

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

Задание №2

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

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

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

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

Однострочные комментарии:

```
" Это однострочный комментарий
```

Многострочные комментарии:

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

Массивы:

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

Словари:

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

Имена:

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

Значения:

- Числа.
- Массивы.
- Словари.

Объявление константы на этапе трансляции:

```
имя: значение
```

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

```
!{+ имя 1}
```

Результатом вычисления константного выражения является значение.

Для константных вычислений определены операции и функции:

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

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

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—10	Биты 11—38
24	Адрес	Константа

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

Тест (A=24, B=2, C=115):

```
0x18, 0x9A, 0x03, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—10	Биты 11—13
10	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=10, B=3, C=1):

```
0x0A, 0x0B, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—7	Биты 8—17	Биты 18—20	Биты 21—23
253	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=253, B=62, C=0, D=5):

```
0xFD, 0x3E, 0xA0, 0x00, 0x00
```

Бинарная операция: `"=="`

A	**B**	**C**	**D**
Биты 0—7	Биты 8—17	Биты 18—20	Биты 21—23
244	Смещение	Адрес	Адрес

Размер команды: 5 байт. Первый операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле B). Второй 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=244, B=129, C=7, D=1):

```
0xF4, 0x81, 0x3C, 0x00, 0x00
```

Тестовая программа

Выполнить поэлементно операцию `"=="` над двумя векторами длины 8. Результат записать во второй вектор.

Вариант №40

Задание №1

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

Ключами командной строки задаются:

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

Лог-файл имеет формат **csv** и содержит все действия во время последнего сеанса работы с эмулятором. 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. `head`.
2. `rm`.
3. `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 по имени пакета **ОС Ubuntu (apt)**. Для описания графа зависимостей используется представление **PlantUML**. Визуализатор должен выводить результат на экран в виде кода.

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

- Путь к программе для визуализации графов.
- Имя анализируемого пакета.
- Путь к файлу-результату в виде кода.
- Максимальная глубина анализа зависимостей.

Все функции визуализатора зависимостей должны быть покрыты тестами.

Задание №3

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

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

Однострочные комментарии:

```
% Это однострочный комментарий
```

Массивы:

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

Имена:

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

Значения:

- Числа.
- Массивы.

Объявление константы на этапе трансляции:

```
(define имя значение)
```

Вычисление константы на этапе трансляции:

```
#{имя}
```

Результатом вычисления константного выражения является значение.

Все конструкции учебного конфигурационного языка (с учетом их возможной вложенности) должны быть покрыты тестами. Необходимо показать 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—19
83	Константа

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

Тест (A=83, B=439):

```
0xD3, 0xDB, 0x00
```

Чтение значения из памяти

A	B
Биты 0—6	Биты 7—21
31	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=31, B=58):

```
0x1F, 0x1D, 0x00
```

Запись значения в память

A	B
Биты 0—6	Биты 7—21
58	Смещение

Размер команды: 3 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=58, B=595):

```
0xBA, 0x29, 0x01
```

Бинарная операция: `"<="`

A	B
Биты 0—6	Биты 7—21
71	Смещение

Размер команды: 3 байт. Первый операнд: элемент, снятый с вершины стека. Второй 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=71, B=813):

```
0xC7, 0x96, 0x01
```

Тестовая программа

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