Лабораторная работа №6 Библиотека STL

Для получения оценки «3» (удовлетворительно) достаточно сделать свой вариант из ЗАДАНИЯ №1.

Для получения оценки «3-4» (удовлетворительно) достаточно сделать свой вариант из ЗАДАНИЯ №2.
В этих заданиях необходимо сделать подробную постановку задачи.

Для получения оценки «3-5» нужно сделать свой вариант из ЗАДАНИЯ 3.
Эти задания поинтереснее. Во многих задачах есть примеры кода.

Задания №2 и №3 содержат соответственно 11 и 15 вариантов.
Ваш вариант будет = остатку от деления своего номера по списку на количество задач (на 15 для задания №2 и на 11 для задания №3).

ЗАДАНИЕ 1.

Используя контейнер *vector* библиотеки STL решить следующие задачи:

1. Определить сколько раз менялся знак у элементов последовательности. Сформировать и вывести массив (вектор) номеров элементов, знак которых отличается от предыдущего элемента.

2. Если минимальный элемент массива лежит в правой его половин, отсортировать правую половину массива по возрастанию. Иначе переписать левую половину в обратном порядке.

3. Если максимальных элемент массива расположен во второй его половине, поменять местами левую и правую половины.

4. Если каждый элемент массива X больше соответствующего элемента массива C, то вычислить i=0n-1(xi2-ci2).

5. Если сумма индексов положительных элементов массива четна, поменять местами соседние нечетный и чётный элементы. Первый со вторым, третий с четвёртым и т.д.

6. Найти сумму положительных элементов массива, индексы которых находятся между индексами минимального и максимального элементов.

7. Удалить максимальный элемент массива. Все следующие за ним элементы должны быть сдвинуты влево, а последний элемент удалён. Например, в массиве [4,2,5,6,1,2,3]. Ответ: 4,2,5,1,2,3.

8. Найти максимальный отрицательный элемент в массиве с нечетным индексом. Также найти среднее арифметическое таких элементов, начиная от индекса минимального элемента массива.

9. Из исходного массива сформировать новый массив равноудалённых от краёв массива элементов. Например, для массива [1,2,3,4,5] ответ будет [6,6,3].

10. Сформировать новый массив из исходного массива, содержащий элементы, большие среднего арифметического всех элементов исходного массива.

11. В массиве отсортировать по убыванию все элементы, начиная с индекса максимального элемента.

12. Сформировать новый массив из исходного массива, содержащий элементы, индексы которых больше индекса максимального элемента исходного массива.

13. В массиве отсортировать по убыванию все элементы, начиная с индекса минимального элемента.

14. Сформировать новый массив из исходного массива, содержащий элементы, расположенные между максимальным и минимальным элементами.

15. Если сумма модулей отрицательных элементов массива больше суммы модулей положительных элементов массива, инвертировать знаки элементов массива. Иначе - отсортировать массив по убыванию.

16. Вставить максимальный элемент массива после каждого отрицательного элемента с четным индексом.

17. Преобразовать массив таким образом, чтобы отрицательные элементы чередовались с неотрицательными. При этом порядок отрицательных и положительных элементов должен сохраниться. Можно использовать дополнительные вектора.

18. Если в векторе сумма квадратов всех его элементов больше N, добавить её в конец массива, иначе изменить порядок элементов в массиве на противоположный.

19. Даны два массива размера N, обозначающих вектора в N-размерном пространстве из точки 01, 02, 03,..., 0n. Найти вектор суммы исходных векторов и его длину.

20. Сформировать из первых N чисел Фибоначчи, начиная с числа под номером K. Последовательность Фибоначчи считать начинающейся с 0, 1.

21. Инверсией называется ситуация, в которой нарушается упорядоченность соседних элементов массива. Например, в массиве при проверке упорядоченности по возрастанию [1,2,1,0] две инверсии: 2 и 1, 1 и 0.
    Отсортировать массив по такому порядку (по возрастанию или по убыванию), инверсий по которому меньше. Например, если в массиве 1 инверсия по возрастанию и 2 по убыванию, отсортировать по возрастанию.

22. Даны два массива символов: массив уникальных символов алфавита и массив символов сообщения. Сформировать новый массив, зашифрованный шифром Цезаря с использованием числа N, вводимого пользователем.

23. Циклическим сдвигом элементов массива называется такое преобразование, в котором индексы элементов массива меняются на некоторое число. Например, массив [1, 2, 3, 4] может быть циклически сдвинут влево на 1 для получения массива [2, 3, 4, 1]. Сформировать новый массив, полученный циклическим сдвигом исходного массива на N. Если N отрицательное, произвести циклический сдвиг влево, иначе вправо.

24. На основе входного вектора сформировать новый вектор, в котором удалены элементы, начиная с индекса N, заканчивая M. Если сумма удалённых элементов положительна, отсортировать результат по возрастанию.

25. Дан массив чисел. Известно, что многие подряд идущие числа в этом массиве равны. Реализовать сжатие исходного массива так, что в новом массиве вместо N подряд идущих чисел K появляется 2 подряд идущих числа N K. Например, в массиве [1, 1, 1, 3, 2, 2, 2, 2, 2, 11, 11, 11, 11, 11, 11] будет сжат до массива [3, 1, 1, 3, 5, 2, 6, 11]

ЗАДАНИЕ 2.

1. Используйте шаблон **vector** для массива данных об авто.
2. Используйте шаблон **list** для двусвязного списка данных об авто.
3. Используйте шаблон **deque** для учёта данных об очереди авто на заправке.
4. Используйте шаблон **set** для построения двух множеств целых чисел и вычисления их пересечения.
5. Используйте шаблон **multiset** для подсчета числа вхождений каждого числа во множество целых чисел с повторами.
6. Используйте шаблон **map** для исключения повторов среди множества целых чисел.
7. Используйте шаблон **multimap** для исключения повторов комбинаций среди множества пар целых чисел.
8. Используйте шаблон **stack** для стека вещественных чисел.
9. Используйте шаблон **queue** для очереди авто на мойке.
10. Используйте шаблон **priority_queue** для очереди заказов, чтобы обслуживать самые большие заказы в первую очередь.
11. Используйте шаблон **bitset** для хранения информации о простоте первых 10000 целых чисел.
12. Используйте шаблон **basic_string** для хранения фамилий имен и отчеств.
13. Используйте шаблон **valarray** для массива данных об авто.
14. Используйте шаблон **hash_map** для массива данных об авто.
15. Используйте шаблон **unordered_map** для массива данных об авто.

ЗАДАНИЕ 3.

Используя библиотеку STL, решить следующую задачу.
В отчете описать, что именно использовано из библиотеки STL при решении задачи.

Вариант 1. Задача «Map Proxy»

Необходимо реализовать Proxy (заместитель) для контейнера map, управляющий доступом.

Задача

Реализовать надстройку над стандартным контейнером map, которая позволяла бы управлять доступом к определённым ключам (с разграничением чтения, записи и удаления), а также подменять при необходимости возвращаемые результаты.

В этой системе присутствует три класса:

- std::map – базовый контейнер.
- Auditor – базовый класс «аудитора», реализующий проверку прав и замену значений.
- ProxiedMap – надстройка над контейнером, которая все операции с ним предваряет вызовом объекта Auditor.

Реализовать несколько подклассов для Auditor (всё разрешающий, разрешающий только чтение, подменяющий значения по заданным ключи константным значением и т.д.)

Тестирование

Разработать набор тестов, проверяющий, действительно ли блокируются попытки несанкционированного доступа (для разных Auditor).

Возможное усложнение

Реализовать несколько видов итераторов (например: итератор, обходящий все доступные ключи; итератор, обходящий все ключи, но сигнализирующий об ошибке доступа при попытке обращения и т.д.).

Вариант 2. Задача «Анализ GPS треков»

Провести анализ GPS трека

Входные данные

GPS-трек в формате GPX с информацией о высоте. Пример.

Задача

Необходимо для этого трека провести анализ и вычислить следующие показатели:

- Общая продолжительность по времени
- Расстояние
- Средняя скорость

- Время движения и время стоянок
- Средняя скорость движения (без учёта стоянок)
- Максимальная скорость
- Минимальная высота
- Максимальная высота
- Общий набор высоты
- Общий спуск
- Распределение скорости по времени (набор пар: диапазон скоростей, длительность). Например, пара 8-14, 15:37 будет означать, что движение со скоростью от 8 до 14 км/ч шло в течение 15 минут 37 сек. Предусмотреть задание параметров для вычисления гистограммы.

Вычисление расстояний по координатам

Выходные данные

Текстовый файл с результатом анализа.

Вариант 3. Задача «Робот-редактор»

Разработать программу, которая будет редактировать файл по заданному «рецепту» и сохранять результат в другой файл.

Входные данные

I. «Рецепт» — файл с командами редактирования, по одной на строку:

- delete from MM to NN — удалить строки с номерами MM..NN. Секции from MM и to NN могут опускаться, в этом случае подразумевается, что удалять нужно с начала или до конца файла.
- change from MM to NN with "TEXT". Заменить строки с MM до NN на текст TEXT. Текст может быть многострочным. Перевод строки записывается как \n, символ обратного слеша — как \\, кавычка — \". Секции from MM и to NN опциональны (см. описание команды delete).
- insert after MM "TEXT". Вставить TEXT после строки MM.
- replace from MM to NN "TEXT" with "TEXT2". Заменить в строках от MM до NN текст TEXT на TEXT2. Секции from MM и to NN опциональны (см. описание команды delete).
- undo. Отменить действие предыдущей команды.

II. Входной файл, к которому будет применяться рецепт.

Задача

Применить команды к входному файлу.

Выходные данные

Отредактированный входной файл.

Тестирование

Разработать набор тестов, проверяющий корректность исполнения различных команд. Уделить внимание командам с некорректными параметрами (например, где номер строки превышает количество строк в файле). Команды должны выполняться, но в границах реально поданного на вход файла. Например, delete from 1 to 1000 должна удалить все строки в файле из двух строк, не приводя к возникновению ошибки.

Вариант 4. Задача «Анаграммы»

Анаграмма — литературный приём, состоящий в перестановке букв или звуков определённого слова (или словосочетания), что в результате даёт другое слово или словосочетание.

Необходимо написать программу, которая расшифровывает заданную анаграмму, пользуясь словарём.

Входные данные

1. Текстовый файл, содержащий одно предложение. Каждое слово в нем является анаграммой. Само предложение также является анаграммой (то есть в нем переставлены слова). Пример:

```
ym hree lsvei fendri
```

1. Текстовый файл-словарь, содержащий в себе слова и связанные с ними веса, отражающие вероятность нахождения слов в тексте. Вероятность употребления слова с бóльшим весом больше, чем с меньшим. Пример:

```
my, 10

bye, 10

friend, 10

here, 10

elvis, 4

lives, 10
```

Файлы могут содержать пустые строки. Их необходимо игнорировать.

Постановка задачи

1) Расшифровать каждое слово. Основываясь на весах слов, выбрать наиболее вероятный вариант.

2) Распечатать все возможные предложения (все перестановки), используя только!! наиболее вероятные варианты слов.

Примечания

1. Если в словаре нет подходящего для анаграммы слова — выдать ошибку.
2. Если для одной анаграммы слова есть несколько вариантов в словаре — выбрать с большим весом. Если веса равны, выбрать любой.
3. Передачу имен файлов реализовать через аргументы командной строки.

Пример вывода

(использует файл из примера выше)

```
friend here my lives
```

```
friend lives here my
friend lives my here
friend my here lives
friend my lives here
here friend lives my
here friend my lives
here lives friend my
here lives my friend
here my friend lives
here my lives friend
lives friend here my
lives friend my here
lives here friend my
lives here my friend
lives my friend here
lives my here friend
my friend here lives
my friend lives here
my here friend lives
my here lives friend
my lives friend here
my lives here friend
```

Входные данные для тестирования

Словарь:

```
time, 10
person, 10
year, 10
way, 10
day, 10
thing, 10
man, 10
```

```
world, 10
life, 10
hand, 10
part, 10
child, 10
eye, 10
woman, 10
place, 10
work, 10
week, 10
case, 10
point, 10
government, 10
company, 10
number, 10
group, 10
problem, 10
fact, 10
Verbs, 10
be, 10
have, 10
do, 10
say, 10
get, 10
make, 10
go, 10
know, 10
take, 10
see, 10
come, 10
think, 10
look, 10
```

```
want, 10
give, 10
use, 10
find, 10
tell, 10
ask, 10
work, 10
seem, 10
feel, 10
try, 10
leave, 10
call, 10
good, 10
new, 10
first, 10
last, 10
long, 10
great, 10
little, 10
own, 10
other, 10
old, 10
right, 10
big, 10
high, 10
different, 10
small, 10
large, 10
next, 10
early, 10
young, 10
important, 10
```

```
few, 10
public, 10
bad, 10
same, 10
able, 10
to, 10
of, 10
in, 10
for, 10
on, 10
with, 10
at, 10
by, 10
from, 10
up, 10
about, 10
into, 10
over, 10
after, 10
beneath, 10
under, 10
above, 10
Others, 10
the, 10
and, 10
a, 10
that, 10
I, 10
it, 10
not, 10
he, 10
as, 10
```

```
you, 10
this, 10
but, 10
his, 10
they, 10
her, 10
she, 10
or, 10
an, 10
will, 10
my, 10
one, 10
all, 10
would, 10
there, 10
their, 10
alert, 5
alter, 6
later, 10
arts, 5
rats, 6
star, 10
tars, 1
dare, 7
dear, 8
read, 10
book, 10
can, 10
drawer, 5
redraw, 10
reward, 7
warder, 3
```

```
warred, 2

please, 10

bye, 10

friend, 10

here, 10

elvis, 4

lives, 10
```

Файл анаграмм для расшифровки:

```
dera liwl hatt ralet kobo I

tras nca ese ouy shti

wedrar ti saeelp

ym hree lsvei fendri
```

Тестирование

Для всех разработанных модулей должны быть созданы наборы unit тестов. Функции ввода/вывода нужно тестировать с помощью std::stringstream.

Вариант 5. Задача «2D-интерполяция»

Необходимо разработать программу двумерной интерполяции.

Входные данные

Текстовый файл:

```
1.0 2.1 3.421

2.54 2.1 5.491

12.0 2.24 3.411
```

Постановка задачи

Реализовать программу двумерной интерполяции, которая читает входной файл, формирует std::vector<double>, который содержит в себе двумерный массив отсчётов на равномерной сетке, размера Nsrc * Nsrc. Размеры определяются из входного файла.

В результате работы должен сформироваться массив интерполированных значений, соответствующий более мелкой сетке Ndst * Ndst. (Ndst > Nsrc)

Массив необходимо записать в выходной файл.

Передачу имен файлов и размера результирующего массива, реализовать через аргументы командной строки.

Формулы интерполяции

Желательно реализовать обе формулы.

- Билинейная интерполяция
- Бикубическая интерполяция

Выбор формулы, осуществляется, так-же, через командную строку.

Выходные данные

Текстовый файл (не связан с вышеописанными входными данными)

```
1.0 2.1 3.421 1.3

2.54 2.1 5.491 1.4

12.0 2.24 3.411 1.5

12.0 2.24 3.411 1.5
```

Тестирование

Для всех разработанных модулей должны быть созданы наборы unit тестов. Функции ввода/вывода нужно тестировать с помощью std::stringstream.

Вариант 6. Задача «Волновой алгоритм в лабиринте»

Найти наикратчайший путь для выхода из лабиринта, либо указать, что пути нет.

Входные данные

Текстовый файл с закодированным лабиринтом. Обозначения: 0 - пусто, 1 - стена, 2 - герой, 3 - выход.

Пример:

```
1 1 1 1 1 1 1 1 1 1

1 2 0 1 0 0 0 0 0 1

1 0 0 1 0 0 1 0 0 1

1 0 0 1 0 0 1 0 0 3

1 0 0 0 0 0 1 0 0 1

1 0 0 0 0 0 0 0 0 1

1 1 1 1 1 1 1 1 1 1
```

Задача

Найти наикратчайший путь от позиции героя до точки выхода из лабиринта используя волновой алгоритм.

Передачу имён файлов реализовать через аргументы командной строки.

Выходные данные

Файл аналогичный исходному, в котором символами * обозначен путь от позиции героя до точки выхода.

Например:

```
1 1 1 1 1 1 1 1 1 1

1 2 0 1 0 0 0 0 0 1

1 * * 1 0 0 1 0 0 1

1 0 * 1 0 0 1 * * 3

1 0 * * 0 0 1 * 0 1

1 0 0 * * * * * 0 1

1 1 1 1 1 1 1 1 1 1
```

Допускается также визуализация пути псевдографикой в консоли. Можно, также, реализовать анимацию "побега".

Тестирование

Необходимо реализовать набор тестов, проверяющих корректность и оптимальность найденного пути, а также проверяющих корректность работы в ситуациях когда выхода нет, герой "замурован", герой отсутствует, и т.п..

Функции ввода/вывода нужно тестировать с помощью std::stringstream.

Вариант 7. Задача «Календарь»

Написать программу, которая выводит календарь (аналог утилиты ncal в UNIX).

Входные данные

Входной текстовый файл формата:

```
<период> | <формат вывода>

...
```

Возможные периоды:

```
month <номер_месяца>

year <год>
```

```
range <номер_месяца_1> <номер_года_1> <номер_месяца_2> <номер_года_2>
```

Возможные форматы вывода (некоторые могут быть указаны совместно):

```
vert

horiz

year_for_every_month

year_once
```

Пример входного файла:

```
range 1 1990 2 1991 | vert year_once

year 1991 | vert year_for_every_month
```

Постановка задачи

Написать программу, которая выводит календарь. Предусмотреть следующие режимы:

1. month- вывод календаря для заданного месяца (год, месяц).
2. year - вывод календаря для заданного года (год).
3. range - вывод календаря для заданного диапазона месяцев

Программа также должна поддерживать управление форматированием:

1. Выбор между вертикальным (vert) и горизонтальным(horiz) расположением дней недели (см. примеры ниже).
2. Печатать число года для каждого месяца (year_for_every_month) или один раз для всех месяцев, принадлежащих году (year_once) (см. примеры).

Передачу имен файлов реализовать через аргументы командной строки.

Выходные данные

Записать результат работы программы в текстовый файл

Расположение дней недели

Горизонтальное

```
   Сентябрь 2014

Пн Вт Ср Чт Пт Сб Вс

 1  2  3  4  5  6  7

 8  9 10 11 12 13 14

15 16 17 18 19 20 21

22 23 24 25 26 27 28
```

```
29 30
```

Вертикальное

```
    Сентябрь 2014
Пн  1  8 15 22 29
Вт  2  9 16 23 30
Ср  3 10 17 24
Чт  4 11 18 25
Пт  5 12 19 26
Сб  6 13 20 27
Вс  7 14 21 28
```

Число года

Один раз

```
                                2014

    Сентябрь          Октябрь           Ноябрь            Декабрь
Пн  1  8 15 22 29        6 13 20 27        3 10 17 24     1  8 15 22 29
Вт  2  9 16 23 30        7 14 21 28        4 11 18 25     2  9 16 23 30
Ср  3 10 17 24        1  8 15 22 29        5 12 19 26     3 10 17 24 31
Чт  4 11 18 25        2  9 16 23 30        6 13 20 27     4 11 18 25
Пт  5 12 19 26        3 10 17 24 31        7 14 21 28     5 12 19 26
Сб  6 13 20 27        4 11 18 25        1  8 15 22 29     6 13 20 27
Вс  7 14 21 28        5 12 19 26        2  9 16 23 30     7 14 21 28


                                2015

    Январь
Пн     5 12 19 26
Вт     6 13 20 27
Ср     7 14 21 28
Чт  1  8 15 22 29
```

```
Пт  2  9 16 23 30
Сб  3 10 17 24 31
Вс  4 11 18 25
```

Много раз

```
    Август 2014       Сентябрь 2014     Октябрь 2014
Пн     4 11 18 25     1  8 15 22 29        6 13 20 27
Вт     5 12 19 26     2  9 16 23 30        7 14 21 28
Ср     6 13 20 27     3 10 17 24        1  8 15 22 29
Чт     7 14 21 28     4 11 18 25        2  9 16 23 30
Пт  1  8 15 22 29     5 12 19 26        3 10 17 24 31
Сб  2  9 16 23 30     6 13 20 27        4 11 18 25
Вс  3 10 17 24 31     7 14 21 28        5 12 19 26
```

Тестирование

Для всех разработанных модулей должны быть созданы наборы unit тестов. Функции ввода/вывода нужно тестировать с помощью std::stringstream.

Вариант 8. Задача «Построение минимального остовного дерева»

Построить минимальное остовное дерево из точек на плоскости.

Входные данные

Текстовый файл с набором пар координат на плоскости.

```
<pont_id1>, <x1>, <y1>
<pont_id2>, <x2>, <y2>
...
```

Пример:

```
1, 7, 5
2, 3, -11
3, 2, -2
4, -1, -8
```

```
5, 14, -14
```

Файл может содержать пустые и невалидные строки. Программа должна корректно это обрабатывать.

Задача

Необходимо связать множестве точек на плоскости таким образом, чтобы сумма длин рёбер была минимальна. Для решения задачи необходимо использовать Алгоритм Крускала. Длинна ребра между любой парой точек вычисляется стандартно, как геометрическое расстояние между точками.

Выходные данные

Текстовый файл, описывающий ребра построенного дерева:

```
<pont_id1> - <pont_id2>
...
```

Пример:

```
1 - 3
2 - 4
2 - 5
3 - 4
```

Тестирование

Для всех разработанных модулей должны быть созданы наборы unit тестов. Функций ввода/вывода нужно тестировать с помощью std::stringstream.

Вариант 9. Задача «Простая бухгалтерия»

Входные данные

На вход программы поступает файл в формате:

```
Дата  Счёт1  Счёт2  Сумма  Примечание
```

Поля разделены пробелами или символами табуляции. Дата имеет формат YYYY-MM-DD или DD.MM.YYYY (2014-09-22, 22.09.2014). Счёт1 и Счёт2 – строки без пробелов. Сумма — число. Примечание – любой текст, он игнорируется.

Каждая строка отражает перевод суммы с одного счёта на другой (транзакция).

Пример файла:

```
01.01.2014 Старт Кошелёк 990     Изначально в кошельке было 990 руб
01.01.2014 Старт Карта 7000      А на карте 7000
20.09.2014 Кошелёк Проезд 70     Заплатил за проезд
20.09.2014 Кошелёк Базар 500     Купил фруктов на базаре
2014-09-21 Карта Книги 770.31    Купил книжку по карточке
21.02.2014 Карта Кошелёк 1000    Снял тысячу в банкомате
```

Файл может содержать пустые и невалидные строки. Программа должна корректно это обрабатывать.

Задача

Программа должна поддерживать два режима:

1. Вывести для всех счетов состояние на текущий момент (после обработки всех транзакций). Например, в кошельке должно остаться 990-70-500+1000 = 1420.
2. Для произвольного счёта и заданного периода времени (две даты) вычислить *входящий остаток* (сумму на счёте до начала периода), *приход* (все поступления на счёт в периоде), *расход* (все списания со счёта в периоде) и *исходящий остаток* (входящий + приход - расход).

Тестирование

Для всех разработанных модулей должны быть созданы наборы unit тестов. Функций ввода/вывода нужно тестировать с помощью `std::stringstream`.

Вариант 10. Задача «Горячие клавиши»

Разработать класс, позволяющий манипулировать горячими клавишами. Горячая клавиша - клавиша или сочетание клавиш на клавиатуре, которому сопоставлено некоторое действие, команда.

Входные данные

Текстовый файл (или несколько), содержащий набор горячих клавиш и назначенных им команд. Пример:

```
Ctrl+Alt+Del = reset
Ctrl+C = copy
Ctrl+Ins = copy
Ctrl+V = paste
Shift+Ins = paste
Ctrl+X = cut
```

```
Shift+Del = cut

F1 = help
```

Сочетания клавиш даны как строки, в рамках задачи можно считать, что названия сочетаний оформлены всегда правильно и однотипно, например, сперва Ctrl (если есть), затем Alt (если есть), затем Shift (если есть), затем название клавиши с заглавной буквы. Названия команд также заданы как строки.

Постановка задачи

Разработать класс, позволяющий манипулировать горячими клавишами. Примерный интерфейс:

```
#include <string>
#include <vector>


class HotkeyMap {
public:
    const std::string & operator[](const std::string & key) const;


    bool add_key(const std::string & key, const std::string & cmd);


    size_t add_cmd(const C & cmd, std::vector < std::string > keys);


    bool remove_key(const std::string & key);
    bool remove_cmd(const std::string & cmd);


    std::vector<std::string> get_cmds(void) const;
    std::vector<std::string> get_keys(const std::string & cmd) const;


    HotkeyMap copy(void) const;
    std::vector<std::string> merge(const HotkeyMap & map);
    void clean(void);
};
```

- operator [] - находит команду связанную с горячей клавишей. По логике, эта функция наиболее часто вызываемая и потому должна быть быстрой.

- add_key - добавляет связку “горячая клавиша - команда”. Если данная клавиша уже привязана к другой команде (конфликт при добавлении в контейнер), то ничего не делает, возвращает false.
- add_cmd - добавляет набор горячих клавиш для указанной команды. Если возник конфликт, процесс добавления прерывается, и возвращаемое значение указывает на номер конфликтной горячей клавиши в наборе.
- remove_key - удаляет горячую клавишу. Если указанная горячая клавиша отсутствует в контейнере - возвращает false.
- remove_cmd - удаляет команду и все связанные с ней горячие клавиши. Если заданная команда отсутствует в контейнере - возвращает false.
- get_cmds - возвращает все команды.
- get_keys - возвращает все горячие клавиши для указанной команды.
- copy - создаёт полную копию контейнера
- merge - объединяет содержимое контейнеров и возвращает список конфликтующих горячих клавиш.
- clean - очищает контейнер.

Выходные данные

Варианты:

1) Даны два файла с наборами горячих клавиш и назначенных им команд. Загрузить файлы, распечатать, объединить, распечатать список конфликтов, объединенный результат записать в третий файл.

Пример вывода программы:

```
file1:
copy    Ctrl+C, Ctrl+Ins,
paste   Ctrl+V, Shift+Ins,
reset   Ctrl+Alt+Del,

file2:
move    Ctrl+M, F6,
copy    Ctrl+C, F5,
search  Ctrl+V, F7,

conflicts: 1
Ctrl+V

file3 = file1.merge(file2):
copy    Ctrl+C, Ctrl+Ins, F5,
paste   Ctrl+V, Shift+Ins,
```

```
reset   Ctrl+Alt+Del,
move    Ctrl+M, F6,
search  F7,
```

2) Дан файл с набором горячих клавиш и назначенных им команд, а так же файл со списком горячих клавиш и команд которые нужно удалить. Загрузить оба файла, удалить горячие клавиши и команды, сохранить результат в третий файл.

Пример вывода программы:

```
file1:
copy    Ctrl+C, Ctrl+Ins, F5,
paste   Ctrl+V, Shift+Ins,
reset   Ctrl+Alt+Del,
move    Ctrl+M, F6,
search  F7,

remove items:
Ctrl+Alt+Del
Ctrl+Ins
move

file2:
copy    Ctrl+C, F5,
paste   Ctrl+V, Shift+Ins,
search  F7,
```

Вариант 11. Задача «Адреса веб-сайта»

Требуется реализовать класс для распознавания адресов страниц на веб-сайте.

Входные данные

Два текстовых файла: с набором правил распознавания и с набором запросов. Если файл с набором запросов не указан, то запросы считываются из стандартного входа (std::cin).

Файл правил

Адрес страницы состоит из фрагментов, разделённых символом /. Правило распознавания имеет такую же структуру, как и адрес, и описывает, какие фрагменты адреса должны принимать фиксированное значение, а какие — могут принимать произвольное. Фрагментам с произвольным содержимым присваивается имя. В записи правила такой фрагмент начинается с символа :, после которого идёт имя.

Формат файла правил:

```
/<фикс_фрагмент>
/:<имя_фрагмента>
/<фикс_фрагмент>/:<имя_фрагмента>
/:<имя_фрагмента>/<фикс_фрагмент>
/<фикс_фрагмент>/:<имя_фрагмента>/<фикс_фрагмент>
...
```

Пример набора правил для сайта с блогами:

```
/posts/new
/posts/:id
/posts/:id/delete
/blogs/:name/posts/:id
```

Здесь id и name — имена фрагментов с произвольным содержимым.

Файл запросов состоит из строк с адресами страниц.

Например:

```
/posts/1
/posts/new
/posts/2
/posts/super-mega-long-title
```

```
/blogs/petya/posts/999-abc
```

Задача

Разработать класс Router, который позволяет сопоставить адреса страниц тому или иному правилу из заранее заданного набора.

Класс Router может быть полезен при обработке сервером входящих HTTP-запросов, когда ответное действие сервера должно соответствовать запрашиваемому адресу.

Интерфейс класса должен содержать методы:

1. **Добавление правила**. Метод должен принимать в качестве аргумента строку с правилом и возвращать уникальный целочисленный идентификатор, соответствующий правилу.
2. **Поиск адреса**. Метод принимает в качестве аргумента строку с адресом и пытается найти правило, которому она соответствует, возвращая номер подходящего правила. Если ничего не найдено, должно возвращаться выделенное значение (например, –1). В случае многозначности (адрес подходит под несколько правил), должно сработать правило, которое было добавлено раньше прочих. Дополнительно должна формироваться и возвращаться структура данных, содержащая в себе значения именованных фрагментов (можно использовать std::map<string, string>).

Тестирование

Необходимо реализовать набор тестов, проверяющих работу класса Router.

Выходные данные

Текстовый файл со списком найденных правил и значениями именованных фрагментов (если они есть в правиле). Если выходной файл не указан, то результаты пишутся в стандартный поток вывода (std::cout).

Например, для приведённых выше правил и запросов, результат будет выглядеть следующим образом:

```
2 id='1'

1

2 id='2'

2 id='super-mega-long-title'

4 name='petya' id='999-abc'
```

Возможное усложнение

Протокол HTTP поддерживает различные методы доступа к страницам. В процессе запроса передаётся не только адрес, но и метод: GET, POST, DELETE, PUT, PATCH.

Добавить в класс Router поддержку HTTP-методов. Каждое правило должно соответствовать одному или нескольким HTTP-методам (подумайте, как их задавать и хранить!). У метода поиска по правилам добавляется ещё один аргумент: HTTP-метод запроса. Теперь он должен искать первое правило, которое будет соответствовать HTTP-методу запроса и адресу.