S. Задание к лабораторным работам по АиСД

Касилов Василий

18 февраля 2024 г.

Версия 1.0

Задания к лабораторным работам 2-го семестра по дисциплине «Алгоритмы и структуры данных» выполняются и принимаются по порядку. В рамках лабораторных работ не разрешается использовать стандартные контейнеры и `std::stringstream`. Однако допускается использование `std::pair`, `std::tuple`, `std::function` и для обработки ввода `std::string`. Для реализации модульных тестов допустимо использовать любые классы стандартной библиотеки.

Все реализуемые сущности (за исключением функции `main` должны быть расположены в отдельном пространстве имен. Имя этого пространства должно совпадать с фамилией студента в нижнем регистре (соответственно, оно совпадает с частью имени каталога с работами до точки). Например, для Петрова Ивана каталог будет называться petrov.ivan, а имя пространства имен — `petrov`. Это пространство имен должно использоваться для всех работ. Для функций, являющихся деталью реализации допустимо использование анонимного пространства имён.

0 Вступительная работа

1. Реализуйте программу, которая выводит на стандартный вывод фамилию и имя студента, разделённые символом «.», на отдельной строке

2. Работа должна быть выполнена в виде 1-го исполняемого файла, не обрабатывающего параметры командной строки:

   ```
   $ ./lab
   petrov.ivan
   ```

1 Списки I

Реализуйте программу, которая считывает последовательности чисел со стандартного ввода и обрабатывает их в соответствии с заданием

1. Входные данные содержат именнованные последовательности чисел — по одной на каждой строке. В следующем примере введено четыре таких последовательности:

   ```
   first 1 1 1
   second 2 2 2 2
   third
   fourth 4 4
   ```

   Количество чисел в последовательностях может различаться. Все числа целые, положительные — проверка на корректность членов последовательности не требуется. Гарантируется, что название состоит только из букв латинского алфавита и ровно одного слова. Признаком конца ввода является `EOF` (на Linux: `Ctrl + D` | на Windows `Ctrl + Z` затем `Enter`)

2. Программа должна вывести следующие результаты:

   - Список названий введённых списков на отдельной строке
   - Затем со следующей строки последовательности чисел, составленные из элементов введённых по следующему правилу: сначала идёт последовательность из первых чисел всех введённых последовательностей (первое число первой последовательности, первое число второй последовательности и т.д.), затем последовательность вторых чисел и так далее, пока не закончатся числа во всех последовательностях. Числа каждого списка включаются в результат ровно один раз. Каждая последовательность выводится на отдельной строке
   - Затем на следующей строке список сумм значений составленных списков на отдельной строке, если такой расчёт возможен

   Например, для указанных входных данных в качестве результата ожидается следующий вывод:

   ```
   first second third fourth
   1 2 4
   1 2 4
   1 2
   2
   7 7 3 2
   ```

3. Если ввод не содержит последовательностей, программа должна вывести `0`

4. Если расчёт требуемой суммы невозможен, программа должна завершаться с соответствующим сообщением об ошибке и кодом возврата `1`. Во всех остальных случаях программа завершается с кодом возврата `0`

5. Для хранения чисел необходимо реализовать шаблон класса списка и соответствующие итераторы

```
1  template< typename T >
2  class List {
3    ...
4  };
```

   Разрабатываемый интерфейс должен быть безопасным относительно исключений

Совет-1 При реализации методов списка следует ориентироваться на контейнеры стандартной библиотеки `std::list` или `std::forward_list`

Совет-2 Сохранять введённые последовательности можно в виде списка пар (`std::pair`), где первый элемент пары — название списка, а второй — список соответствующих чисел

2 Стеки и очереди

1. Ниже приведены интерфейсы шаблонов классов «Стек» и «Очередь»

```
1  template< typename T >
2  class Queue {
3   public:
4    void push(T rhs);
5    T drop();
6    ...
7  };
8
9  template< typename T >
10 class Stack {
11  public:
12   void push(T rhs);
13   T drop();
14   ...
15 };
```

   Переработайте интерфейсы: исправьте и дополните их; сделайте безопасными. Реализуйте исправленные интерфейсы. Воспользуйтесь реализованными шаблонами классов для разбора арифметических выражений

2. Арифметические выражения содержатся в файле, имя которого задано параметром командной строки, по одному выражению на каждой строке.

   - Элементы выражения гарантировано разделены ровно одним пробелом (при этом выражения могут быть неверными с точки зрения математики). Пример входного файла:

     ```
     ( 1 + 2 ) * ( 3 - 4 )
     1 + 3
     ( 10 / ( 2 + 3 ) % 4 )
     4 * 7 - 3
     ```

   - Пустые строки в файле пропускаются
   - Если параметр командной строки filename не задан, выражения читаются со стандартного ввода. Признаком конца ввода является `EOF`(на Linux: `Ctrl + D` | на Windows `Ctrl + Z` затем `Enter`)
   - Если арифметические выражения верны, результатом работы программы являются результаты вычисления выражений выведенные в обратном порядке (относительно порядка выражений) в одну строку, разделённые ровно одним пробелом

- В арифмитических выражениях используются только бинарные операции: `+` — сложение, `-` — вычитание, `*` — умножение, `/` — деление, `%` — остаток от деления
- В случае ошибки во время вычислений (например, операнд не является числом или арифметическое выражение неверно) программа должна завершаться с ненулевым кодом возврата и сообщением об ошибке

3. Работа должна быть выполнена в виде 1-го исполняемого файла, принимающего параметры следующим образом:

   ```
   $ ./lab {filename}
   ```

   filename представляет собой опциональной параметр. Поведение программы меняется в зависимости от того передан он или нет

Совет-1 Для вычисления выражений необходимо преобразовать инфиксные арифметические выражения в постфиксные, а затем произвести вычисления

Совет-2 Стеков и очередей достаточно как для решения задачи преобразования инфиксных выражений в постфиксные, так и для вычислений

Совет-3 Проверять инфиксное выражение на корректность не нужно: алгоритм преобразования выражения и последующее вычисление являются алгоритмом проверки

Совет-4 Реализация стека и очереди может быть выполнена на основе реализованного списка или с использованием динамического массива

3 Списки II

1. Реализовать шаблон класса `BidirectionalList` и соответствующие итераторы

   ```
   1 template< typename T >
   2 class BidirectionalList {
   3   ...
   4 };
   ```

   Разрабатываемый интерфейс должен быть безопасным

2. Параметром командной строки задётся имя файла filename, который содержит внутри себя данные по некоторому количеству списков

   - Файл со списками имеет следующий вид:

     ```
     <list-1> <value-1-1> <value-1-2> ...
     <list-2> <value-2-2> <value-2-2> ...
     ...
     ```

   - `list` представляет собой имя списка; `value` целочисленное значение - элемент списка. Например:

     ```
     first 1 3 2 1
     second 6 0 -1 1
     ```

   - Пустые строки игнорируются

3. Реализуемая программа должна считывать списки, содержащиеся в файле, и обрабатывать команды, принимаемые со стандартного ввода от пользователя.

   - Каждая строка содержит ровно одну команду. Должны поддерживаться следующие команды:

     ```
     print <list>
     replace <destination-list> <value-1> <value-2>
     replace <destination-list> <value-1> <source-list>
     remove <destination-list> <value-1>
     remove <destination-list> <source-list-2>
     concat <new-list> <1st-list> <2nd-list> <3rd-list> ...
     equal <1st-list> <2nd-list> <3rd-list> ...
     ```

- Команда `print <list>` выводит данные списка с соответствующим именем. Например, для команды:

  ```
  print first
  ```

  Должен быть результат:

  ```
  first 1 3 2 1
  ```

  Если список пуст, то команда должна вывести сообщение <EMPTY>
- Команда `replace` заменяет заданные элементы в списке. В качестве последнего параметра может быть передано значение или другой список. Например, для команд:

  ```
  replace first 1 2
  replace first 3 second
  print first
  ```

  Должен быть результат:

  ```
  first 2 6 0 -1 1 2 2
  ```

- Команда `remove` удаляет из списка заданные элементы. Удаляемые элементы могут быть заданы значением или другим списком. Например, для команд:

  ```
  remove first 2
  remove first second
  print first
  ```

  Должен быть результат:

  ```
  first 3
  ```

- Команда `concat` созадёт новый список конкатенацией списков, перечисленных в качестве параметров. Например, для команд:

  ```
  concat third first second first
  concat yathird second first second first
  print third
  print yathird
  ```

  Должен быть результат:

  ```
  third 1 3 2 1 6 0 -1 1 1 3 2 1
  yathird 6 0 -1 1 1 3 2 1 6 0 -1 1 1 3 2 1
  ```

- Команда `equal` сравнивает списки, переданные в качестве параметров. Например, для команды:

  ```
  equal first first
  ```

  Должен быть результат:

  ```
  <TRUE>
  ```

  А для команды:

  ```
  equal first first second first
  ```

  Должен быть результат:

  ```
  <FALSE>
  ```

- Если команда по каким-то причинам некорректна, то команда должна вывести сообщение <INVALID COMMAND>
- Других команд реализовывать не требуется
- Признаком конца ввода команд является `EOF` (на Linux: `Ctrl + D` | на Windows `Ctrl + Z` затем `Enter`)

4. Работа должна быть выполнена в виде 1-го исполняемого файла, принимающего параметры следующим образом:

   ```
   $ ./lab filename
   ```

filename представляет собой обязательный параметр. Поведение программы меняется в зависимости от того передан он или нет. Если параметр filename не задан, программа должна завершаться с ненулевым кодом возврата и сообщением об ошибке

Совет-1 Вы можете воспользоваться контейнером `std::map` для хранения данных и диспетчеризации команд

Совет-2 Если двунаправленный список уже был реализован в работе «Списки I», используйте его

4 Бинарные деревья поиска I

1. Реализуйте шаблон класса `BinarySearchTree` и соответствующие итераторы

```
1 template< typename Key, typename Value, typename Compare >
2 class BinarySearchTree {
3  public:
4   ...
5   void push(Key k, Value v);
6   Value get(Key k);
7   Value drop(Key k);
8   ...
9 };
```

   Дерево позволяет добавлять элементы типа `Value` по ключю типа `Key`. Отношение порядка на множестве объектов типа `Key` устанавливается компаратором `Compare`. Переработайте интерфейс: исправьте и дополните его; сделайте безопасным

2. Параметром командной строки задётся имя файла filename, который содержит внутри себя данные по некоторому количеству словарей.

   Файл со словарями имеет следующий вид:

   ```
   <dataset-1> <key-1-1> <value-1-1> <key-1-2> <value-1-2> ...
   <dataset-2> <key-2-2> <value-2-2> <key-2-2> <value-2-2> ...
   ...
   ```

   - `dataset` представляет собой имя словаря; `key` целочисленный ключ; `value` значение в виде строки соответствующее ключу Например:

     ```
     first 1 name 2 surname
     second 4 mouse 1 name 2 keyboard
     ```

   - Пустые строки игнорируются. Данные в строке разделены ровно одним пробелом

3. Реализуемая программа должна считывать данные словарей из файла и выполнять команды, принимаемые от пользователя со стандартного ввода.

   - Каждая строка содержит ровно одну команду. Должны поддерживаться следующие команды:

     ```
     print <dataset>
     complement <newdataset> <dataset-1> <dataset-2>
     intersect <newdataset> <dataset-1> <dataset-2>
     union <newdataset> <dataset-1> <dataset-2>
     ```

   - Команда `print <dataset>` выводит данные словаря с соответствующем именем в порядке сортировки ключей. Например, для `second` должна быть напечатана строка

     ```
     second 1 name 2 keyboard 4 mouse
     ```

     Если словарь пуст, то команда должна вывести сообщение <EMPTY>

   - Команда complement `<newdataset> <dataset-1> <dataset-2>` строит словарь с новым именем как вычитание множеств двух других словарей. Например, для команд:

     ```
     complement third second first
     print third
     ```

     Должен быть результат:

```
third 4 mouse
```

- Команда `intersect <newdataset> <dataset-1> <dataset-2>` строит словарь с новыми именем как пересечение множеств двух других словарей. Например, для команд:

  ```
  intersect fourth first second
  print fourth
  ```

  Должен быть результат:

  ```
  fourth 1 name 2 surname
  ```

  Если ключи дублируются, в качестве значения выбираются данные из левого операнда, т. е. для команд:

  ```
  intersect yafourth second first
  print yafourth
  ```

  Должен быть результат:

  ```
  yafourth 1 name 2 keyboard
  ```

- Команда **`union`** `<newdataset> <dataset-1> <dataset-2>` строит словарь с новым именем как объединение множеств двух других словарей. Например, для команд:

  ```
  union fifth first second
  print fifth
  ```

  Должен быть результат:

  ```
  fifth 1 name 2 surname 4 mouse
  ```

  Если ключи дублируются, в качестве значения выбираются данные из левого операнда, т. е. для команд:

  ```
  union yafifth second first
  print yafifth
  ```

  Должен быть результат:

  ```
  yafifth 1 name 2 keyboard 4 mouse
  ```

- Если команда по каким-то причинам некорректна, то команда должна вывести сообщение <INVALID COMMAND>
- Других команд реализовывать не требуется
- Признаком конца ввода команд является `EOF` (на Linux: `Ctrl + D` | на Windows `Ctrl + Z` затем `Enter`)

4. Работа должна быть выполнена в виде 1-го исполняемого файла, принимающего параметры следующим образом:

   ```
   $ ./lab filename
   ```

   filename представляет собой обязательный параметр. Поведение программы меняется в зависимости от того передан он или нет. Если параметр filename не задан, программа должна завершаться с ненулевым кодом возврата и сообщением об ошибке

Совет-1 При разработке интерфейса дерева следует ориентироваться на шаблон из стандартной библиотеки `std::map`

Совет-2 Дерево инстанцируется парой (`std::pair`), в которой первый элемент является ключом, а второй — значением списком пар ключ-значение. При этом, для хранения словарей следует использовать словарь, где ключом являются имена словарей, а значениями — сами словари

Совет-3 Названия команд и их конкретные реализации представимы в виде словаря

Совет-4 Исправьте работу «Списки II», если она была реализована ранее, , заменив используемый контейнер `std::map` на дерево

5 Бинарные деревья поиска II

1. Дополните интерфейс шаблона класса бинарного дерева поиска:

```
1  class BinarySearchTree {
2   public:
3    ...
4    template< typename F >
5    F traverse_lnr(F f) const {
6      ...
7      return f;
8    }
9    template< typename F >
10   F traverse_rnl(F f) const {
11     ...
12     return f;
13   }
14   template< typename F >
15   F traverse_breadth(F f) const {
16     ...
17     return f;
18   }
19 };
```

   Реализуйте инфиксные обходы дерева (слева направо и справо налево), а также обход дерева в ширину. Во время обхода над элементами дерева (парой <ключ-значеие>) должна выполняться операция обхода, передаваемая в виде функционального объекта.

   - Пример функционального объекта, подходящего для обхода дерева:

```
1  struct Key_summ {
2    void operator()(const std::pair< const int, std::string > & key_value){
3      result += key_value.first;
4    }
5    int result_ = 0;
6  };
```

     Использование такого функционального объекта позволяет посчитать сумму ключей в дереве типа `BinarySearchTree< int, std::string, ... >`.

   - Реализация обходов должна быть выполнена итеративно с использованием ранее реализованных шаблонов классов «Стек» и «Очередь»

2. Работа должна быть выполнена в виде 1-го исполняемого файла, принимающего параметры следующим образом:

```
$ ./lab [ascending|descending|breadth] filename
```

   - Все параметры обязательные. `ascending`, `descending` или `breadth` указывает напрвление обхода дерева, которое будет построено в процессе работы программы. Данные дерева должны содержаться в файле с именем filename.
   - Данные для дерева в файле filename имеют следующий вид:

     ```
     key1 value1 key2 value2 key3 value3 ...
     ```

     Например:

     ```
     10 keyboard 12 monitor -5 mouse
     ```

   - Программа должна выполнить обход дерева по указанному направлению и сформировать на выходе строку, состоящую из суммы ключей и значений, разделённых пробелами. Порядок значений определяется направлением обхода. Например, для представленного примера, если обойти дерево по возрастанию, то результат должен быть следующий:

     ```
     17 mouse keyboard monitor
     ```

Если расчёт суммы ключей произвести невозможно (в следствие переполнения), то программа должна завершаться с ненулевым кодом возврата и сообщением об ошибке

- Для пустого дерева выводится <EMPTY>.

6 Сортировки

1. Реализуйте указанные преподавателем алгоритмы сортировки (всего не менее 3-х)
   - По крайней мере один из реализованных алгоритмов должен быть применим для однонаправленного списка, если он был реализован (в противном случае используется `std::forward_list`)
   - По крайней мере два из реализованных алгоритмов должны быть применимы для двунаправленного списка, если он был реализован (в противном случае используется `std::list`)
   - Все реализованные алгоритмы должны быть применимы для стандартного контейнера `std::deque`
2. Реализуемая программа должна сортировать числа и выводить результаты сортировки на стандартный вывод:
   - Заполните нужное количество экземпляров реализованных контейнеров: однонаправленного списка и двунаправленного списка, а также стандартный контейнер `std::deque` сгенерированными числами (одинаковым набором) и отсортируйте данные различными алгоритмами.
   - Исходные данные и результаты сортировок выведите на стандартный вывод на отдельных строках. Например, результаты работы программы могут быть такими:

     ```
     $ ./lab ascending ints 4
     1 3 2 4
     1 2 3 4
     1 2 3 4
     1 2 3 4
     1 2 3 4
     1 2 3 4
     1 2 3 4
     ```

3. Работа должна быть выполнена в виде 1-го исполняемого файла, принимающего параметры следующим образом:

   ```
   $ ./lab [ascending|descending] [ints|floats] [size]
   ```

   Все параметры обязательные

   - Числа сортируются по возрастанию или по убыванию в зависимости от параметра командной строки (`ascending` — по возрастанию, `descending` — по убыванию).
   - Числа должны быть сгенерированы с использованием генератора псевдослучайных чисел в количестве, определяемым параметром командной строки `size`
   - Тип сортируемых чисел также определяется параметром командной строки (`ints` — для знаковых целых и `floats` — для вещественных).

Совет-1 Обратите внимание на шаблон `std::sort()`

Совет-2 Обратите внимание на другие стандартные алгоритмы, например, `std::copy_n()` или `std::copy()`. При реализации алгоритма, подумайте как было бы удобней для реализации: принимать диапазон итераторов или итератор на начало и размер

Совет-3 Выбирайте наиболее простой алгоритм сортировки для реализации сортировки однонаправленного списка