Лабораторная работа №3

Задание

Требуется реализовать программу, состоящую из двух частей. Первая часть программы должна реализовывать операции над простыми структурами данных. Вторая часть задания выполняет операции над матрицами. При выполнении задания необходимо использовать указатели и функции управления памятью приложения.

Вариант выбирать в соответствии с собственным порядковым номером в списке группы. В случае нехватки вариантов считать, что нумерация циклическая.

Детали реализации

При запуске первой строчкой выводится *Имя и Фамилия* автора программы и *номер варианта*. Следующей строкой программа должна спросить пользователя, какую часть задания необходимо продемонстрировать:

```
Please choose part of task:
```

Пользователь должен указать либо «*1*», либо «*2*». В противном случае, программа должна вывести сообщение «`Invalid part number, type '1' or '2'`» и завершиться с кодом ошибки 1.

При правильном выборе демонстрируемой части задания необходимо вывести заголовок:

```
Part #{Number} - {Description}
```

Где, `{Number}` – номер демонстрируемой части, а `{Description}` – краткое описание (*прим. Текст данной строки должен располагаться строго на одной строке*).

После завершения демонстрации, необходимо вывести строку «`Bye!`» и программа должна быть завершена с успешным кодом завершения (0).

При вводе одиночных числовых значений (например, размер массива или матрицы), необходимо предусмотреть обработку некорректного ввода.

При вводе числовых последовательностей (например, список значений для матрицы), необходимо предусмотреть простой способ ввода, при котором числа будут запрашиваться до тех пор, пока пользователь не введет необходимое количество чисел. В случае неправильного ввода, программа должна его проигнорировать и корректно прочитать следующее значение без вывода информации об ошибке. Так же необходимо реализовать возможность ввода списка чисел, разделенных не только переносом на следующую строку, но и с помощью пробела или табуляции.

Числовые последовательности, которые будут использоваться для заполнения структур данных, являются вещественными.

В качестве знака разделителя в вещественных числах необходимо использовать символ «`.`».

Детали реализации первой части задания

Программа ожидает команды от пользователя (список команд и их описания смотреть в каждом варианте) и выполняет их.

Команда должна читаться запрашиваться одной строкой. В случае если такой команды не существует, необходимо вывести сообщение об ошибке:

```
Incorrect command, please try again!
```

И с новой строки происходит запрос новой команды от пользователя.

Так же в некоторых командах необходимо предусмотреть корректность ввода (подробнее в каждом варианте), в случае неправильного ввода параметров или при возникновении иных некорректных условий необходимо вывести сообщение об ошибке в формате:

```
Invalid operation! {Description}
```

Где, *{Description}* – подробное описание возникшей проблемы.

Форматы вывода информации о структурах данных

1. **Стек**

```
Stack-{size} [{v1}, {v2}…]
```

Где, {size} – размер стека, {v1} – значение, размещенное в очереди. Порядок значений должен начинаться с вершины стека.

```
Stack-0 []
Stack-3 [9, 12, 0]
```

2. **Очередь**

```
Queue-{size} [{v1}, {v2}…]
```

Где, {size} – размер очереди, {v} – значение, размещенное в очереди. Порядок значений должен начинаться с головы очереди.

```
Queue-0 []
Queue-4 [1, 2, 3, 4]
```

3. **Список**

```
List-{size} [{v1}, {v2}…]
```

Где, {size} – размер списка, {v} – значение, размещенное в списке. Порядок значений при выводе должен начинаться с головы списка.

```
List-0 []
List-4 [1, 6, 2, -1]
```

4. **Словарь**

```
Map-{size} [({key1}, {value1}), ({key2}, {value2})]
```

Где, {size} – размер словаря, {key} – ключ записи словаря, {value} – значение записи словаря

Примеры:

```
Map-0 []
Map-2 [(hello, 5), (world, 5)]
```

5. **Бинарное дерево**

```
BinaryTree [({key}, {value}, ({left_node}), ({right_node}))]
```

Где, {key} – ключ узла в дереве, {value} – значение узла, {left_node} – представление левого поддерева для текущего узла, {right_node} – представление правого поддерева для текущего узла. Пустое поддерево обозначается круглыми скобками – ().

Примеры:

```
BinaryTree [(4, 10.9, (), (5, 11.4, (), ()))]
BinaryTree [()]
```

Детали реализации второй части задания

В каждом варианте задания у пользователя необходимо запросить ввод размера матрицы. Если требуется задать размер квадратной матрицы, то необходимо запросить только 1 значение:

```
Input matrix size:
```

Если необходимо задать прямоугольную матрицу, то необходимо запросить два значения, каждое по отдельности:

```
Input matrix width:
Input matrix height:
```

Для каждого вводимого значения необходимо реализовать проверку ввода. Если было введено некорректное значение для одного из запрашиваемого параметра, то необходимо завершить выполнение работы программы с кодом ошибки (1).

Далее программа должна запросить у пользователя NxM значений для заполнения матрицы.

Пример ввода числовой последовательности для матрицы 5x5

```
12    13    10    -10   11
0     1     5     12    48
12    86    46    37    83
74    19    0     51    45
91    75    47    -5    84
```

Варианты заданий, часть 1

Вариант 1

Реализовать работу со структурой данных – стек на основе массива.

Необходимые операции:

1. **`push {number}`** – читает значение `{number}` из консоли и помещает его в стек. В следующей строке необходимо вывести информацию о стеке после выполнения данной операции.
2. **pop** – читает значение из стека и выводит его на экран в виде «`Readed value is {number}`», где `{number}` – прочитанное значение. Если стек пустой, то необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше). В случае если значение было получено, необходимо следующей строкой вывести информацию о стеке.
3. **done** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 2

Реализовать работу со структурой данных – стек на основе связанного списка.

Необходимые операции:

1. **`push {number}`** – читает значение `{number}` из консоли и помещает его в стек. В следующей строке необходимо вывести информацию о стеке после выполнения данной операции.
4. **pop** – читает значение из стека и выводит его на экран в виде «`Readed value is {number}`», где `{number}` – прочитанное значение. Если стек пустой, то необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше). В случае если значение было получено, необходимо следующей строкой вывести информацию о стеке.
2. **done** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 3

Реализовать работу со структурой данных – очередь на основе массива.

Необходимые операции:

1. **`enqueue {number}`** – читает значение {number} из консоли и помещает его в конец очереди. В следующей строке необходимо вывести информацию об очереди после выполнения данной операции.
2. **dequeque** – читает значение из очереди и выводит его на экран в виде «Readed value is {number}», где {number} – прочитанное значение из очереди. Так же при успешном выполнении данной команды следующей строкой необходимо вывести состояние очереди. В случае неуспешного выполнения команды требуется вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше).
3. **done** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 4

Реализовать работу со структурой данных – односвязный список.

Необходимые операции:

1. **`add {number}`** – читает значение {number} из консоли и помещает его в конец списка. В следующей строке необходимо вывести информацию о списке после выполнения данной операции.
2. **`add-after {number1} {number2}`** – читает два значения из консоли {number1} и {number2}, далее происходит поиск значения {number1} и в след за ним в список добавляется значение {number2}. Если значение {number1} не было найдено в списке, то необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше). В случае успешного выполнения данной операции необходимо отобразить информацию о списке.
3. **`delete {number}`** – читает значение из консоли {number} и пытается удалить его из списка. Если указанное значение не было найдено в списке, то необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше). В

случае успешного завершения операции, необходимо отобразить информацию о списке.

4. **done** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 5

Реализовать работу со структурой данных – словарь на основе массива.

Необходимые операции:

1. **`add {key} {value}`** – читает слово {key} (может содержать только латинские символы a-Z), и число {value} из консоли. Считанные значения размещаются в словарь. В случае, если ключ словаря дублируется, необходимо заменить его значение на новое. После успешного выполнения операции, необходимо вывести информацию о словаре.
2. **`get {key}`** – читает значение из словаря, которое соответствует считанному ключу {key} и выводит его на экран в формате «Readed value id {value}». В случае если указанный ключ не был найден в словаре, необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше).
3. **`delete {key}`** – читает слово-ключ {key} из консоли и предпринимает попытку удаления значения из словаря. Если ключ не был найден, необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше). В случае успешного выполнения операции необходимо вывести информацию о словаре.
4. **done** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 6

Реализовать работу со структурой данных – бинарное дерево.

Необходимые операции:

1. **`add {key} {value}`** – читает два числа из консоли {key} и {value} и размещает их в бинарном дереве. После выполнения данной операции необходимо вывести на новой строке информацию о результирующем дереве.
2. **`find {key}`** – читает число {key} и выполняет поиск данного ключа по дереву. В случае успешного поиска, необходимо вывести сообщение «Key {key} successfully found! Value is {value}». Если ключ не был найден, необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше).
3. **done** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 7

Реализовать работу со структурой данных – двусвязный список.

Необходимые операции:

1. **`add-first {number}`** - читает значение {number} из консоли и помещает его в начало списка. В следующей строке необходимо вывести информацию о содержании списка после выполнения данной операции.
2. **`add-last {number}`** - читает значение {number} из консоли и помещает его в конец списка. В следующей строке необходимо вывести информацию о содержании списка после выполнения данной операции.
3. **`min {index}`** - читает значение {index} из консоли. Выводит минимальное значение списка из элементов, следующих за заданным значением {index}, считая его от начала и до конца списка. Если заданное значение {index} отрицательное, считать, что значение индекса задается с конца списка.
4. **`between {number1} {number2}`** - читает значения {number1} и {number2} из консоли. Выводит список элементов, находящихся в диапазоне заданных значений. Условия попадания в диапазон - строго не равны. В случае если не было найдено ни одного такого значения, выдается сообщение об ошибке (формат вывода ошибки смотрите выше).
5. **`done`** - завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 8

Реализовать работу со структурой данных – очередь на основе массива.

Необходимые операции:

1. **`enqueue {number}`** – читает значение {number} из консоли и помещает его в конец очереди. В следующей строке необходимо вывести информацию об очереди после выполнения данной операции.
2. **`dequeque`** – читает значение из очереди и выводит его на экран в виде «Readed value is {number}», где {number} – прочитанное значение из очереди. Так же при успешном выполнении данной команды следующей строкой необходимо вывести состояние очереди. В случае неуспешного выполнения команды требуется вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше).
3. **`done`** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 9

Реализовать работу со структурой данных – односвязный список.

Необходимые операции:

1. **`add {number}`** - читает значение {number} из консоли и помещает его в конец списка. В следующей строке необходимо вывести информацию о содержании списка после выполнения данной операции.
2. **`delete {number}`** - читает значение {number} из консоли. Если заданное значение существует в списке, удаляет его, соединяя предыдущий и последующий элементы между собой (не допускает образования разрывов). В следующей строке

необходимо вывести информацию о содержании списка после выполнения данной операции. В противном случае необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше).

3. **`sort`** - сортирует список по возрастанию. Можно использовать любой известный Вам метод сортировки.
4. **`where {number}`** - читает значение {number} из консоли. Выводит числа, содержащиеся в списке, которые большие по модулю заданного числа {number}.
5. **`done`** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Вариант 10

Реализовать работу со структурой данных – список на основе динамического массива.

Необходимые операции:

1. **`add {number}`** - читает значение {number} из консоли и помещает его в конец массива. В следующей строке необходимо вывести информацию о содержании массива после выполнения данной операции.
2. **`insert {index} {number}`** - читает значения {index} и {number} из консоли. Вставляет значение {number} по индексу {index}, если он существует, в противном случае выводится сообщение об ошибке (формат сообщения об ошибке смотрите выше). Если вставка удалась, последующие элементы сдвигаются вправо. Индекс равный размеру списка является корректным.
3. **`delete {index}`** - читает значение {index} из консоли. Если существует значение по заданному индексу, удаляет его, сдвигая значения влево. В следующей строке необходимо вывести информацию о содержании списка после выполнения данной операции. В противном случае необходимо вывести сообщение об ошибке (формат сообщения об ошибке смотрите выше).
4. **`sort`** - сортирует массив по убыванию. Можно использовать любой известный Вам метод сортировки.
5. **`done`** – завершает демонстрацию задания.

Формат вывода информации смотрите выше.

Варианты заданий, часть 2

Вариант 1

Дана целочисленная прямоугольная матрица. Определить:

- количество строк, не содержащих ни одного нулевого элемента;
- максимальное из чисел, встречающихся в заданной матрице более одного раза.

Формат вывода:

```
Lines without zero: {count}
Maximum: {number}
```

Вариант 2

Дана целочисленная прямоугольная матрица. Определить количество столбцов, не содержащих ни одного нулевого элемента.

Характеристикой строки целочисленной матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.

Формат вывода:

```
Columns without zero: {count}
Sorted matrix:
0     0     0
1     1     1
2     2     2
```

Вариант 3

Дана целочисленная прямоугольная матрица. Определить:

- количество столбцов, содержащих хотя бы один нулевой элемент
- номер строки, в которой находится самая длинная серия одинаковых элементов.

Формат вывода:

```
Columns with zero: {count}
Line with same elements: {index}
```

Вариант 4

Дана целочисленная квадратная матрица. Определить:

- произведение элементов в тех строках, которые не содержат отрицательных элементов
- максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Формат вывода:

```
Element product: {value}
Maximum value: {index}
```

Вариант 5

Дана целочисленная квадратная матрица. Определить:

- сумму элементов в тех столбцах, которые не содержат отрицательных элементов
- минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.

Формат вывода:

```
Sum of elements: {value}
Minimum value: {value}
```

Вариант 6

Дана целочисленная прямоугольная матрица. Определить:

- сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент
- номера строк и столбцов всех седловых точек матрицы.

Формат вывода:

```
Sum: {value}
Anchor points: [({line1}, {column1}), ({line2}, {line2})]
      Или если не было найдено таких точек
Anchor points: []
```

Примечание

> Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в t-й строке и максимальным — в j-м столбце.

Вариант 7

Для заданной матрицы размером N x N найти такие k, при которых k-я строка матрицы совпадает с k-м столбцом.

Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Формат вывода:

```
Equal index: {k}
Sum: {value}
```

Вариант 8

Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

Характеристикой столбца целочисленной матрицы назовём сумму модулей его отрицательных нечетных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик.

Формат вывода:

```
Columns without zero: {count}
Sorted matrix:
0     0     0
1     1     1
2     2     2
```

Вариант 9

Операция сглаживания матрицы даёт новую матрицу того же размера, каждый элемент которой вычисляется как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы.

Построить результат сглаживания заданной вещественной матрицы размером N x N. В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.

Формат вывода:

```
Sorted matrix:
0     0     0
1     1     1
2     2     2
Sum: {value}
```

Вариант 10

Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей.

- Подсчитать количество локальных минимумов заданной матрицы размером N x N.
- Найти сумму модулей элементов, расположенных выше главной диагонали.

Формат вывода:

```
Count of local min: {count}
Sum: {value}
```

Вариант 11

Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду.

Найти количество строк, среднее арифметическое элементов каждой из которых меньше заданной величины.

Формат вывода:

```
Triangle matrix:
1     2     3
0     2     1
0     0     3
Count: {count}
```

Вариант 12

Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями. Найти номер первой из строк, содержащих хотя бы один положительный элемент.

Формат вывода:

```
Compacted matrix:
5     2     6
1     1     1
7     2     4
Index of positive line: {index}
```

Вариант 13

У пользователя запрашивается команда вида:

```
roll-right {number}
roll-down {number}
```

Осуществить циклический сдвиг элементов прямоугольной матрицы на N элементов вправо или вниз (в зависимости от указанного пользователем режима), N может быть больше количества элементов в строке или столбце.

Формат вывода:

```
Rolled matrix:
0     0     0
1     1     1
2     2     2
```

Вариант 14

Осуществить циклический сдвиг элементов квадратной матрицы размером M x N вправо на k элементов таким образом: элементы первой строки сдвигаются в последний столбец сверху вниз, из него — в последнюю строку справа налево, из нее — в первый столбец снизу-вверх, из него — в первую строку; для остальных элементов — аналогично.

Формат вывода:

```
Rolled matrix:
0     0     0
1     1     1
2     2     2
```

Вариант 15

Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.

Характеристикой строки целочисленной матрицы назовем сумму ее отрицательных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.

Формат вывода:

```
Column with zero: {index}
Sorted matrix:
0     0     0
1     1     1
2     2     2
```

Вариант 16

Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.

Упорядочить строки целочисленной прямоугольной матрицы по возрастанию количества одинаковых элементов в каждой строке.

Формат вывода:

```
Column without negative: {index}
Sorted matrix:
0     0     0
1     1     1
2     2     2
```

Вариант 17

Найти номер первой из строк, не содержащих ни одного положительного элемента.

Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу, следующий по величине — в позиции (2, 2), следующий по величине — в позиции (3, 3) и т. д., заполнив таким образом всю главную диагональ.

Формат вывода:

```
Column without positive: {index}
Sorted matrix:
8     0     0
1     6     1
2     2     4
```

Вариант 18

Дана целочисленная прямоугольная матрица. Определить:

- количество строк, содержащих хотя бы один нулевой элемент
- номер столбца, в котором находится самая длинная серия одинаковых элементов.

Формат вывода:

```
Count of lines with zero: {count}
Column with same elements: {index}
```

Вариант 19

Дана целочисленная квадратная матрица. Определить:

- сумму элементов в тех строках, которые не содержат отрицательных элементов
- минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Формат вывода:

```
Count of elements: {count}
Minimum: {value}
```

Вариант 20

Дана целочисленная прямоугольная матрица. Определить:

- количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент
- номера строк и столбцов всех седловых точек матрицы.

Формат вывода:

```
Count of negative: {value}
Anchor points: [({line1}, {column1}), ({line2}, {line2})]
      Или если не было найдено таких точек
Anchor points: []
```

Примечание

Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в t-й строке и максимальным — в j-м столбце.