МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

Курсовая работа
по дисциплине «Программирование»

Тема: «Разработка электронной картотеки»

Студент гр. 3311 _____ Сапронов К.Д.

Преподаватель _____ Хахаев И.А.

Санкт-Петербург
2024

Введение

Электронная картотека пользователей — это система для хранения и управления данными пользователей социальной сети.

Цель работы:

Разработка программы, которая будет обеспечивать эффективное взаимодействие с электронной картотекой пользователей, хранящейся на диске. Программа должна выполнять следующие действия:

- занесение данных в электронную картотеку;
- внесение изменений (исключение, корректировка, добавление);
- поиск данных по различным признакам;
- сортировку по различным признакам;
- вывод результатов на экран и сохранение на диске.
- Выбор подлежащих выполнению команд должен быть реализован с помощью основного меню и вложенных меню.

Задача должна быть структурирована и отдельные части должны быть оформлены как функции.

Постановка задачи и описание решения

Предметная область - российские города

Изначально данные должны считываться из файлов, а в процессе работы - вводиться с клавиатуры. Картотека должна храниться в виде списков и массивов структур, связанных указателями. В программе должно быть реализовано меню для выбора команд.

Для реализации задачи была разработана программа на языке программирования C, в котором использована архитектура, базирующаяся на двусвязных списках для хранения данных. Программа структурирована таким образом, чтобы каждая часть задачи была оформлена как отдельная функция, обеспечивая модульность и удобство в поддержке кода.

Основные компоненты программы:

- Структуры данных:

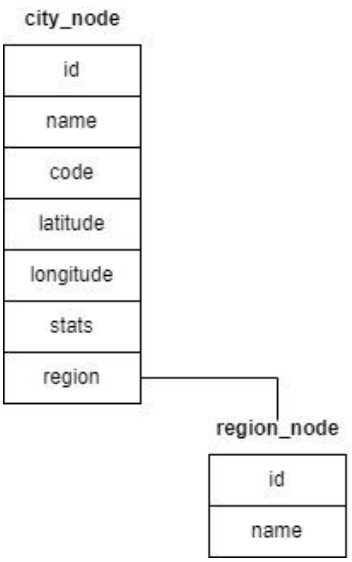
- city_node и city_head для хранения информации о городах.
- region_node и region_head для хранения информации о регионах.

- Функции для работы с данными:

1. Создание, добавление и удаление узлов: make_city_node, make_region_node, appendCityNode, appendRegionNode, deleteCityNode, deleteRegionNode;
2. Работа с файлами - чтение и запись: readRegionsFile, readCitiesFile, writeRegionsToFile, writeCitiesToFile;
3. Работа с данными - поиск, фильтрация и сортировка: filterCitiesByName, filterCitiesByRegionId, filterCitiesByCode, filterCitiesByLatitude, filterCitiesByLongitude, filterCitiesByArea, filterCitiesByPopulation, sortCities, compareCities, findRegionByName, findRegionById;
4. Ввод данных с клавиатуры: editCityNameGUI, editCityRegionGUI, editCityCodeGUI, editCityLatitudeGUI, editCityLongitudeGUI, editCityAreaGUI, editCityPopulationGUI;
5. Взаимодействие с пользователем - меню и работа с данными: menu, menuInput, options, addRegionGUI, addCityGUI, updateCityDataGUI, deleteRegionGUI, deleteCityGUI, clearRegionListGUI, clearCityListGUI;
6. Вспомогательные функции - обработка строк и интерфейс: trim, trimForDisplay, clearStdin, pressEnterToContinue, clearConsole.

Программа начинается с инициализации списков пользователей и профессий, которые считываются из соответствующих .csv файлов. После этого выводится меню для пользователя, через которое он может управлять картотекой. Основной цикл программы управляется функцией menu, которая вызывает функции по выбору пользователя. По завершению пользователю предлагается сохранить изменения в файлах.

Сущности и их назначения:



Описание переменных.

struct region_node

№	Имя переменной	Тип	Назначение
1	id	int	Номер элемента
2	name	char	Имя региона
3	next	region_node	Указатель на следующий элемент
4	prev	region_node	Указатель на предыдущий элемент

struct region_head

№	Имя переменной	Тип	Назначение
1	count	int	Количество элементов в списке
2	first	city_regions	Указатель на первый элемент
3	last	int	Указатель на последний элемент

struct city_node

№	Имя переменной	Тип	Назначение
1	id	int	Номер элемента
2	name	char	Название города
3	region	region_node	Регион
4	code	int	Код региона
5	latitude	float	Широта города
6	longitude	float	Долгота города
7	stats	int	Массив, содержащий значения площади и населения города
8	next	city	Указатель на следующий элемент
9	prev	city	Указатель на предыдущий элемент

struct city_head

№	Имя переменной	Тип	Назначение
1	count	int	Количество элементов в списке
2	first	city_node	Указатель на первый элемент
3	last	city_node	Указатель на последний элемент

int main()

№	Имя переменной	Тип	Назначение
1	cHead	region_head	Заголовок списка городов
2	rHead	city_head	Заголовок списка регионов

void menu()

№	Имя переменной	Тип	Назначение
1	cHead	region_head	Заголовок списка городов
2	rHead	city_head	Заголовок списка регионов
3	option	int	Ответ пользователя

4	saveOption	int	Выбор пользователя о сохранении изменений

int menuInput()

№	Имя переменной	Тип	Назначение
1	answer	int	Ответ пользователя

void options()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка регионов
2	cHead	city_head	Заголовок списка городов
3	option	int	Ответ пользователя

int startsWithIgnoreCase()

№	Имя переменной	Тип	Назначение
1	str	const char	Строка
2	prefix	const char	Строка для сравнения с началом
3	isPrefix	int	Флаг того, является ли одна строка началом другой

void clearStdin()

№	Имя переменной	Тип	Назначение
1	c	int	Считывание символов

void printAllRegions()

№	Имя переменной	Тип	Назначение
1	r_head	region_head	Заголовок списка регионов
2	current	region_node	Текущий элемент

void printAllCities()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	current	city_node	Текущий элемент

void printRegion()

№	Имя переменной	Тип	Назначение
1	region	region_node	Выводимый регион
2	trimmedName	char	Строка для вывода

void printCity()

№	Имя переменной	Тип	Назначение
1	city	city_node	Выводимый город
2	region	char	Выводимый регион
3	trimmedName	char	Строка для вывода города
4	trimmedRegion	char	Строка для вывода региона

void trimForDisplay()

№	Имя переменной	Тип	Назначение
1	output	char	Результат
2	input	const char	Исходная строка
3	mlen	int	Максимальная длина строки

void nullString()

№	Имя переменной	Тип	Назначение
1	str	char	Строка для обнуления
2	i	int	Счетчик в цикле

void trim()

№	Имя переменной	Тип	Назначение
1	str	char	Обрабатываемая строка
2	i	int	Счетчик в цикле
3	flag	int	Флаг в цикле

char split()

№	Имя переменной	Тип	Назначение
1	str	char	Исходная строка
2	length	int	Длина исходной строки
3	sep	char	Символ-разделитель
4	count	int	Счетчик новых строк
5	i	int	Счетчик в цикле
6	start	int	Номер символа начала слова
7	j	int	Счетчик в цикле
8	wordLen	int	Длина текущего слова
9	result	char	Массив новых строк
10	newStr	char	Хранение текущего слова
11	allocError	int	Ошибка выделения памяти

void readRegionsFile()

№	Имя переменной	Тип	Назначение
1	filename	char	Имя файла
2	rHead	region_head	Заголовок списка регионов
3	n	int	Количество строк
4	count	int	Счетчик записанных строк
5	i	int	Счетчик в цикле
6	temp	char	Текущая строка
7	region	region_node	Текущий элемент

void readCitiesFile()

№	Имя переменной	Тип	Назначение
1	filename	char	Имя файла
2	rHead	region_head	Заголовок списка регионов
3	cHead	city_head	Заголовок списка городов
4	n	int	Количество строк
5	count	int	Счетчик записанных строк
6	i	int	Счетчик в цикле
7	slen	int	Длина текущей строки
8	splitArray	char	Массив строк, полученных из считанной строки
9	temp	char	Текущая строка
10	city	city_node	Текущий элемент

void writeRegionsToFile()

№	Имя переменной	Тип	Назначение
1	filename	const char	Имя файла
2	rHead	region_head	Заголовок списка регионов
3	current	region_node	Текущий элемент

void writeCitiesToFile()

№	Имя переменной	Тип	Назначение
1	filename	const char	Имя файла
2	cHead	region_head	Заголовок списка регионов
3	current	city_node	Текущий элемент

void freeRegionsList()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка регионов
2	current	region_node	Текущий элемент
3	current1	region_node	Вспомогательный элемент

void freeCity()

№	Имя переменной	Тип	Назначение
1	city	city_node	Освобождаемый элемент

void freeCitiesList()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	current	city_node	Освобождаемый элемент
3	current1	city_node	Вспомогательный элемент

void appendRegionNode()

№	Имя переменной	Тип	Назначение
1	r_head	region_head	Заголовок списка городов

2	region	region_node	Добавляемый элемент

void appendCityNode()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	city	city_node	Добавляемый элемент

void deleteRegionNode()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	rHead	region_head	Заголовок списка регионов
3	region	region_node	Удаляемый элемент

void deleteCityNode()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	city	city_node	Удаляемый элемент

void deleteRegionGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	rHead	region_head	Заголовок списка регионов
3	id	int	Id удаляемого элемента
4	region	region_node	Удаляемый элемент

void deleteCityGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	id	int	Id удаляемого элемента
3	city	city_node	Удаляемый элемент

void addRegionGUI()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка городов
2	temp	char	Название региона
3	region	region_node	Новый элемент

void addCityGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	rHead	region_head	Заголовок списка регионов
3	city	city_node	Новый элемент

void fillCityNode()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка регионов

2	cHead	city_head	Заголовок списка городов
3	city	city_node	Заполняемый элемент
4	str	char	Массив строк с данными

void clearRegionListGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	rHead	region_head	Заголовок списка регионов
3	current	region_node	Текущий элемент
4	current1	region_node	Вспомогательный элемент
5	city	city_node	Элемент списка городов для очищения ссылок на регионы

void clearCityListGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	current	city_node	Текущий элемент
3	current1	city_node	Вспомогательный элемент

void clearCityRegionById()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	Id	int	Номер элемента
3	current	city_node	Текущий элемент

city_node findCityById()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	Id	int	Номер элемента
3	current	city_node	Текущий элемент

void filterCitiesByFieldGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	rHead	region_head	Заголовок списка регионов
3	option	int	Выбор пользователя

void filterCitiesByName()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	name	char	Имя для сравнения
3	current	city_node	Текущий элемент

void filterCitiesByRegionId()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов

2	rHead	region_head	Заголовок списка регионов
3	id	int	Номер региона
4	current	city_node	Текущий элемент

void filterCitiesByCode()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	minC	int	Минимальное значение кода
3	maxC	int	Максимальное значение кода
4	current	city_node	Текущий элемент

void filterCitiesByLatitude()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	minL	float	Минимальное значение широты
3	maxL	float	Максимальное значение широты
4	current	city_node	Текущий элемент

void filterCitiesByLongitude()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	minL	float	Минимальное значение долготы
3	maxL	float	Максимальное значение долготы
4	current	city_node	Текущий элемент

void filterCitiesByArea()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	minA	int	Минимальное значение площади
3	maxA	int	Максимальное значение площади
4	current	city_node	Текущий элемент

void filterCitiesByPopulation()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	minP	int	Минимальное значение населения
3	maxP	int	Максимальное значение населения
4	current	city_node	Текущий элемент

void sortCitiesGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	option	int	Выбор пользователя

void sortCities()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	option	int	Выбор пользователя
3	sorted	city_node	Элемент после отсортированной части списка
4	current	city_node	Текущий элемент
5	next	city_node	Элемент после текущего
6	temp	city_node	Вспомогательный элемент

int compareCities()

№	Имя переменной	Тип	Назначение
1	a	city_node	Элемент для сравнения
2	b	city_node	Элемент для сравнения
3	option	int	Выбор пользователя
4	result	int	Возвращаемое значение

region_node findRegionByName()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка регионов
2	name	char	Имя искомого элемента
3	current	int	Текущий элемент

region_node findRegionById()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка регионов
2	id	int	Id искомого элемента
3	current	int	Текущий элемент

void updateCityDataGUI()

№	Имя переменной	Тип	Назначение
1	cHead	city_head	Заголовок списка городов
2	rHead	region_head	Заголовок списка регионов
3	cityId	int	Номер элемента
4	option	int	Выбор пользователя
5	city	city_node	Редактируемый элемент

void editCityName()

№	Имя переменной	Тип	Назначение
1	city	city_node	Редактируемый элемент

2	temp	char	Новое название

void editCityRegion()

№	Имя переменной	Тип	Назначение
1	rHead	region_head	Заголовок списка регионов
2	city	city_node	Редактируемый элемент
3	success	int	Флаг для проверки успешности выполнения
4	regionId	int	Id нового региона
5	region	region_node	Указатель на регион

void editCityCode()

№	Имя переменной	Тип	Назначение
1	city	city_node	Редактируемый элемент
2	success	int	Флаг для проверки успешности выполнения
3	code	int	Новое значение

void editCityLatitude()

№	Имя переменной	Тип	Назначение
1	city	city_node	Редактируемый элемент
2	latitude	float	Новое значение
3	success	int	Флаг для проверки успешности выполнения

void editCityLongitude()

№	Имя переменной	Тип	Назначение
1	city	city_node	Редактируемый элемент
2	longitude	float	Новое значение
3	success	int	Флаг для проверки успешности выполнения

void editCityArea()

№	Имя переменной	Тип	Назначение
1	city	city_node	Редактируемый элемент
2	area	int	Новое значение
3	success	int	Флаг для проверки успешности выполнения

void editCityPopulation()

№	Имя переменной	Тип	Назначение
1	city	city_node	Редактируемый элемент
2	population	int	Новое значение
3	success	int	Флаг для проверки успешности выполнения

Контрольные примеры.

Пример 1.

```
Choose an option:
1.  Print all regions
2.  Print all cities
3.  Add new region
4.  Add new city
5.  Update city data
6.  Filter cities
7.  Sort cities
8.  Delete region
9.  Delete city
10. Clear regions list
11. Clear cities list
0.  Exit
> 
```

```
1.  Print all regions

| Id|    Region name|
+---+---------------+
| 1 | Central       |
| 2 | Volga         |
| 3 | South         |
| 4 | Siberia       |
| 5 | Northwest     |
| 6 | Ural          |
| 7 | Caucasus      |
| 8 | Far East      |
+---+---------------+


Press ENTER to continue 
```

```
2.  Print all cities

|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 1   | Moscow             | Central       | 77         | 55.76    | 37.62    | 2562     | 13010112      |
| 2   | Saint Petersburg   | Northwest     | 78         | 59.93    | 30.36    | 1439     | 5601911       |
| 3   | Novosibirsk        | Siberia       | 54         | 54.98    | 82.90    | 507      | 1633595       |
| 4   | Yekaterinburg      | Ural          | 66         | 56.84    | 60.65    | 468      | 1544376       |
| 5   | Kazan              | Volga         | 16         | 55.79    | 49.12    | 614      | 1308660       |
| 6   | Nizhny Novgorod    | Volga         | 52         | 56.33    | 44.01    | 411      | 1228199       |
| 7   | Chelyabinsk        | Ural          | 74         | 55.16    | 61.44    | 502      | 1189525       |
| 8   | Krasnoyarsk        | Siberia       | 24         | 56.02    | 92.89    | 354      | 1187771       |
| 9   | Samara             | Volga         | 63         | 53.20    | 50.16    | 541      | 1173299       |
| 10  | Ufa                | Volga         | 2          | 54.73    | 55.96    | 708      | 1144809       |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue 
```

```
3.  Add new region

Enter region name: test1

Region successfully added.

| Id|    Region name|
+---+---------------+
| 9 | test1         |
+---+---------------+

Press ENTER to continue 
```

```
4.  Add new city

Enter information for new city:

Enter city name: Petergof
| Id|    Region name|
+---+---------------+
| 1 | Central       |
| 2 | Volga         |
| 3 | South         |
| 4 | Siberia       |
| 5 | Northwest     |
| 6 | Ural          |
| 7 | Caucasus      |
| 8 | Far East      |
| 9 | test1         |
+---+---------------+

Enter region id: 10
Region not found
Enter city code: 1000
Invalid or impossible code
Enter city latitude: 59.9
Enter city longitude: 29.9
Enter city area: -10
Invalid or impossible area
Enter city population: 80123

City has been successfully added.
|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 11  | Petergof           | undefined     | 0          | 59.90    | 29.90    | 0        | 80123         |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue
```

```
5.  Update city data

|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 1   | Moscow             | Central       | 77         | 55.76    | 37.62    | 2562     | 13010112      |
| 2   | Saint Petersburg   | Northwest     | 78         | 59.93    | 30.36    | 1439     | 5601911       |
| 3   | Novosibirsk        | Siberia       | 54         | 54.98    | 82.90    | 507      | 1633595       |
| 4   | Yekaterinburg      | Ural          | 66         | 56.84    | 60.65    | 468      | 1544376       |
| 5   | Kazan              | Volga         | 16         | 55.79    | 49.12    | 614      | 1308660       |
| 6   | Nizhny Novgorod    | Volga         | 52         | 56.33    | 44.01    | 411      | 1228199       |
| 7   | Chelyabinsk        | Ural          | 74         | 55.16    | 61.44    | 502      | 1189525       |
| 8   | Krasnoyarsk        | Siberia       | 24         | 56.02    | 92.89    | 354      | 1187771       |
| 9   | Samara             | Volga         | 63         | 53.20    | 50.16    | 541      | 1173299       |
| 10  | Ufa                | Volga         | 2          | 54.73    | 55.96    | 708      | 1144809       |
| 11  | Petergof           | undefined     | 0          | 59.90    | 29.90    | 0        | 80123         |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
Enter city id: 11
```

```
Update city data

|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 11  | Petergof           | undefined     | 0          | 59.90    | 29.90    | 0        | 80123         |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Which field would you like to edit?

1. Name
2. Region
3. Code
4. Latitude
5. Longitude
6. Area
7. Population
8. All fields

Enter option: 3
```

```
3. Code

Enter city code: 78

Updated city:

|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 11  | Petergof           | undefined     | 78         | 59.90    | 29.90    | 0        | 80123         |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue
```

```
Choose an option:
1.  Print all regions
2.  Print all cities
3.  Add new region
4.  Add new city
5.  Update city data
6.  Filter cities
7.  Sort cities
8.  Delete region
9.  Delete city
10. Clear regions list
11. Clear cities list
0.  Exit
> 0

Do you want to save changes? (1 - yes, 0 - no): 1

Changes successfully saved


Press ENTER to continue
```

```
Process returned 0 (0x0)   execution time : 339.137 s
Press any key to continue.
```

Пример 2.

```
Choose an option:
1.  Print all regions
2.  Print all cities
3.  Add new region
4.  Add new city
5.  Update city data
6.  Filter cities
7.  Sort cities
8.  Delete region
9.  Delete city
10. Clear regions list
11. Clear cities list
0.  Exit
> |
```

```
2.  Print all cities

|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 1   | Moscow             | Central       | 77         | 55.76    | 37.62    | 2562     | 13010112      |
| 2   | Saint Petersburg   | Northwest     | 78         | 59.93    | 30.36    | 1439     | 5601911       |
| 3   | Novosibirsk        | Siberia       | 54         | 54.98    | 82.90    | 507      | 1633595       |
| 4   | Yekaterinburg      | Ural          | 66         | 56.84    | 60.65    | 468      | 1544376       |
| 5   | Kazan              | Volga         | 16         | 55.79    | 49.12    | 614      | 1308660       |
| 6   | Nizhny Novgorod    | Volga         | 52         | 56.33    | 44.01    | 411      | 1228199       |
| 7   | Chelyabinsk        | Ural          | 74         | 55.16    | 61.44    | 502      | 1189525       |
| 8   | Krasnoyarsk        | Siberia       | 24         | 56.02    | 92.89    | 354      | 1187771       |
| 9   | Samara             | Volga         | 63         | 53.20    | 50.16    | 541      | 1173299       |
| 10  | Ufa                | Volga         | 2          | 54.73    | 55.96    | 708      | 1144809       |
| 11  | Petergof           | undefined     | 78         | 59.90    | 29.90    | 0        | 80123         |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue |
```

```
6.  Filter cities

1. Name
2. Region
3. Code
4. Latitude
5. Longitude
6. Area
7. Population

Enter option: 6
Enter min area: 500
Enter max area: 1000
|   ID|           City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 3   | Novosibirsk        | Siberia       | 54         | 54.98    | 82.90    | 507      | 1633595       |
| 5   | Kazan              | Volga         | 16         | 55.79    | 49.12    | 614      | 1308660       |
| 7   | Chelyabinsk        | Ural          | 74         | 55.16    | 61.44    | 502      | 1189525       |
| 9   | Samara             | Volga         | 63         | 53.20    | 50.16    | 541      | 1173299       |
| 10  | Ufa                | Volga         | 2          | 54.73    | 55.96    | 708      | 1144809       |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue |
```

```
7.  Sort cities

1. Sort by id
2. Sort by name
3. Sort by region
4. Sort by latitude
5. Sort by longitude
6. Sort by area
7. Sort by population
Enter option: 2
List successfully sorted

Press ENTER to continue |
```

```
2.  Print all cities

|   ID|          City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+
| 7   | Chelyabinsk        | Ural          | 74         | 55.16    | 61.44    | 502      | 1189525       |
| 5   | Kazan              | Volga         | 16         | 55.79    | 49.12    | 614      | 1308660       |
| 8   | Krasnoyarsk        | Siberia       | 24         | 56.02    | 92.89    | 354      | 1187771       |
| 1   | Moscow             | Central       | 77         | 55.76    | 37.62    | 2562     | 13010112      |
| 6   | Nizhny Novgorod    | Volga         | 52         | 56.33    | 44.01    | 411      | 1228199       |
| 3   | Novosibirsk        | Siberia       | 54         | 54.98    | 82.90    | 507      | 1633595       |
| 11  | Petergof           | undefined     | 78         | 59.90    | 29.90    | 0        | 80123         |
| 2   | Saint Petersburg   | Northwest     | 78         | 59.93    | 30.36    | 1439     | 5601911       |
| 9   | Samara             | Volga         | 63         | 53.20    | 50.16    | 541      | 1173299       |
| 10  | Ufa                | Volga         | 2          | 54.73    | 55.96    | 708      | 1144809       |
| 4   | Yekaterinburg      | Ural          | 66         | 56.84    | 60.65    | 468      | 1544376       |
+-----+--------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue
```

```
8.  Delete region

| Id|    Region name|
+---+---------------+
| 1 | Central       |
| 2 | Volga         |
| 3 | South         |
| 4 | Siberia       |
| 5 | Northwest     |
| 6 | Ural          |
| 7 | Caucasus      |
| 8 | Far East      |
| 9 | test1         |
+---+---------------+


Enter region id to delete (or 0 to return to menu): 9

Region with id 9:
| Id|    Region name|
+---+---------------+
| 9 | test1         |
+---+---------------+

Region with id 9 has been successfully removed.

Press ENTER to continue
```

```
1.  Print all regions

| Id|    Region name|
+---+---------------+
| 1 | Central       |
| 2 | Volga         |
| 3 | South         |
| 4 | Siberia       |
| 5 | Northwest     |
| 6 | Ural          |
| 7 | Caucasus      |
| 8 | Far East      |
+---+---------------+


Press ENTER to continue
```

```
10. Clear region list

List successfully cleared.

Press ENTER to continue
```

```
1.  Print all regions

| Id|    Region name|
+---+---------------+
+---+---------------+


Press ENTER to continue
```

```
2.  Print all cities

|   ID|          City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+-------------------+---------------+------------+----------+----------+----------+---------------+
| 7   | Chelyabinsk       | undefined     | 74         | 55.16    | 61.44    | 502      | 1189525       |
| 5   | Kazan             | undefined     | 16         | 55.79    | 49.12    | 614      | 1308660       |
| 8   | Krasnoyarsk       | undefined     | 24         | 56.02    | 92.89    | 354      | 1187771       |
| 1   | Moscow            | undefined     | 77         | 55.76    | 37.62    | 2562     | 13010112      |
| 6   | Nizhny Novgorod   | undefined     | 52         | 56.33    | 44.01    | 411      | 1228199       |
| 3   | Novosibirsk       | undefined     | 54         | 54.98    | 82.90    | 507      | 1633595       |
| 11  | Petergof          | undefined     | 78         | 59.90    | 29.90    | 0        | 80123         |
| 2   | Saint Petersburg  | undefined     | 78         | 59.93    | 30.36    | 1439     | 5601911       |
| 9   | Samara            | undefined     | 63         | 53.20    | 50.16    | 541      | 1173299       |
| 10  | Ufa               | undefined     | 2          | 54.73    | 55.96    | 708      | 1144809       |
| 4   | Yekaterinburg     | undefined     | 66         | 56.84    | 60.65    | 468      | 1544376       |
+-----+-------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue
```

```
11. Clear city list

List successfully cleared.

Press ENTER to continue
```

```
2.  Print all cities

|   ID|          City name|         Region| Region code|  Latitude| Longitude|      Area|     Population|
+-----+-------------------+---------------+------------+----------+----------+----------+---------------+
+-----+-------------------+---------------+------------+----------+----------+----------+---------------+

Press ENTER to continue
```

```
Choose an option:
1.  Print all regions
2.  Print all cities
3.  Add new region
4.  Add new city
5.  Update city data
6.  Filter cities
7.  Sort cities
8.  Delete region
9.  Delete city
10. Clear regions list
11. Clear cities list
0.  Exit
> 0

Do you want to save changes? (1 - yes, 0 - no): 0

Press ENTER to continue
```

```
Process returned 0 (0x0)   execution time : 199.627 s
Press any key to continue.
```

Заключение:

Заголовочный файл <stdio.h>

• printf - вывод информации на экран

• fgets - чтение строк из файла или ввода с клавиатуры

• scanf - чтение ввода с клавиатуры

• fprintf - запись строк в файл

• perror - вывод сообщений об ошибке

• fopen - открытие файлов

• fclose - закрытие файлов

• rewind - возвращение в начало файла

Заголовочный файл <stdlib.h>

• malloc - выделение динамической памяти

• free - освобождение выделенной динамической памяти

• atoi - конвертация строки в в целое число

• atof - конвертация строки в число с плавающей точкой

• system - выполнение системных команд (очищение консоли)

Заголовочный файл <string.h>

• strcpy - копирование строк

• strncpy - копирование указанного числа символов из одной строки в другую

• strcat - объединение двух строк

• strcmp - сравнение строк

• strlen - нахождение длины строки

Заголовочный файл <ctype.h>

• tolower - преобразование заглавных букв в строчные

Выводы:

В результате выполнения работы была изучена работа со структурами в языке С и получены практические навыки в создании электронных картотек.

Структура вызовов функций

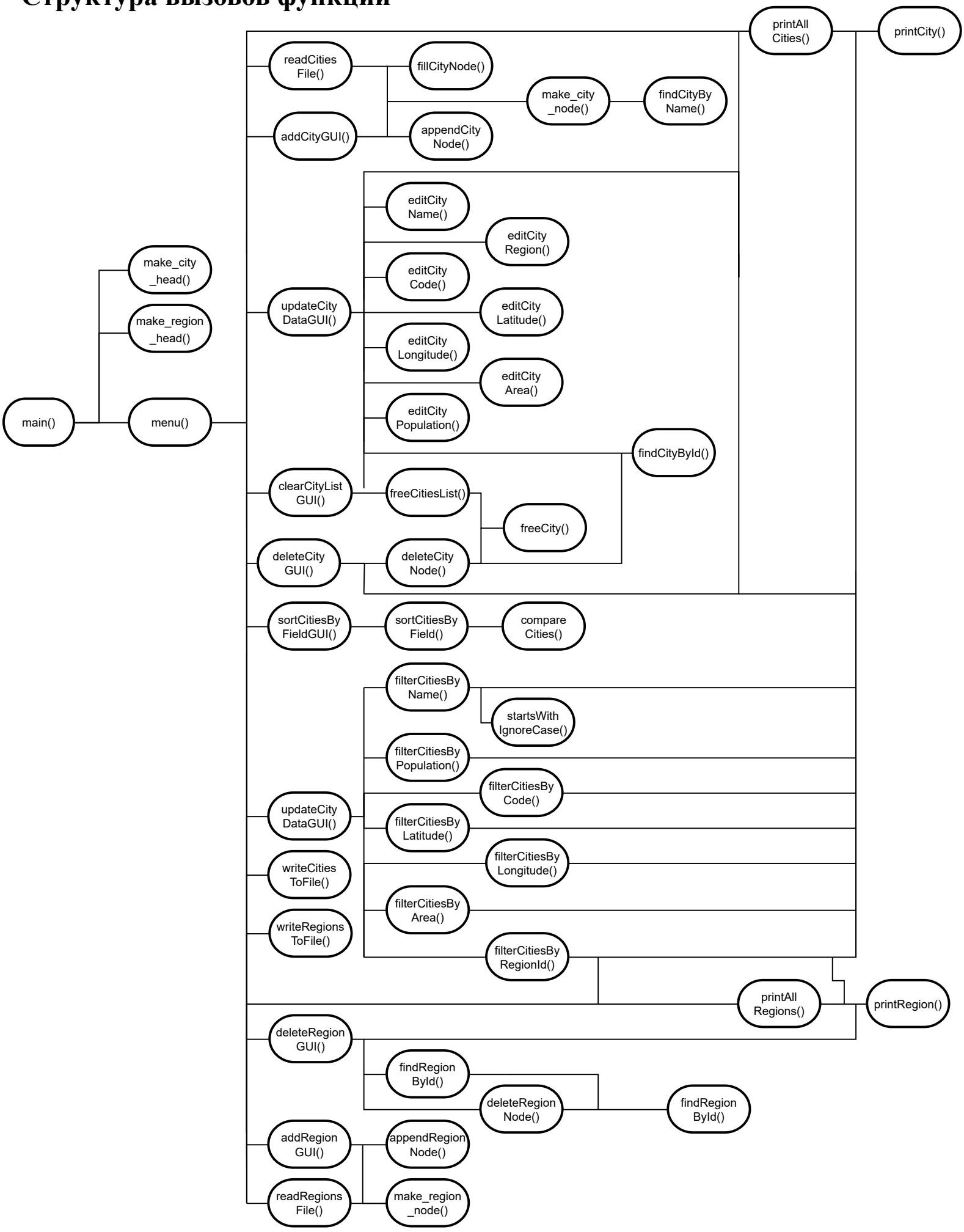
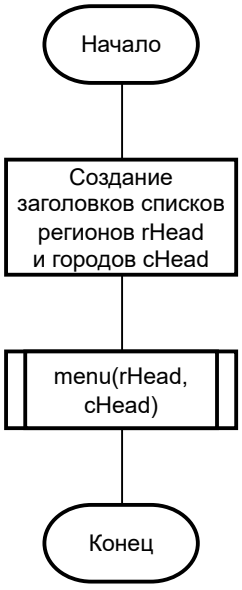
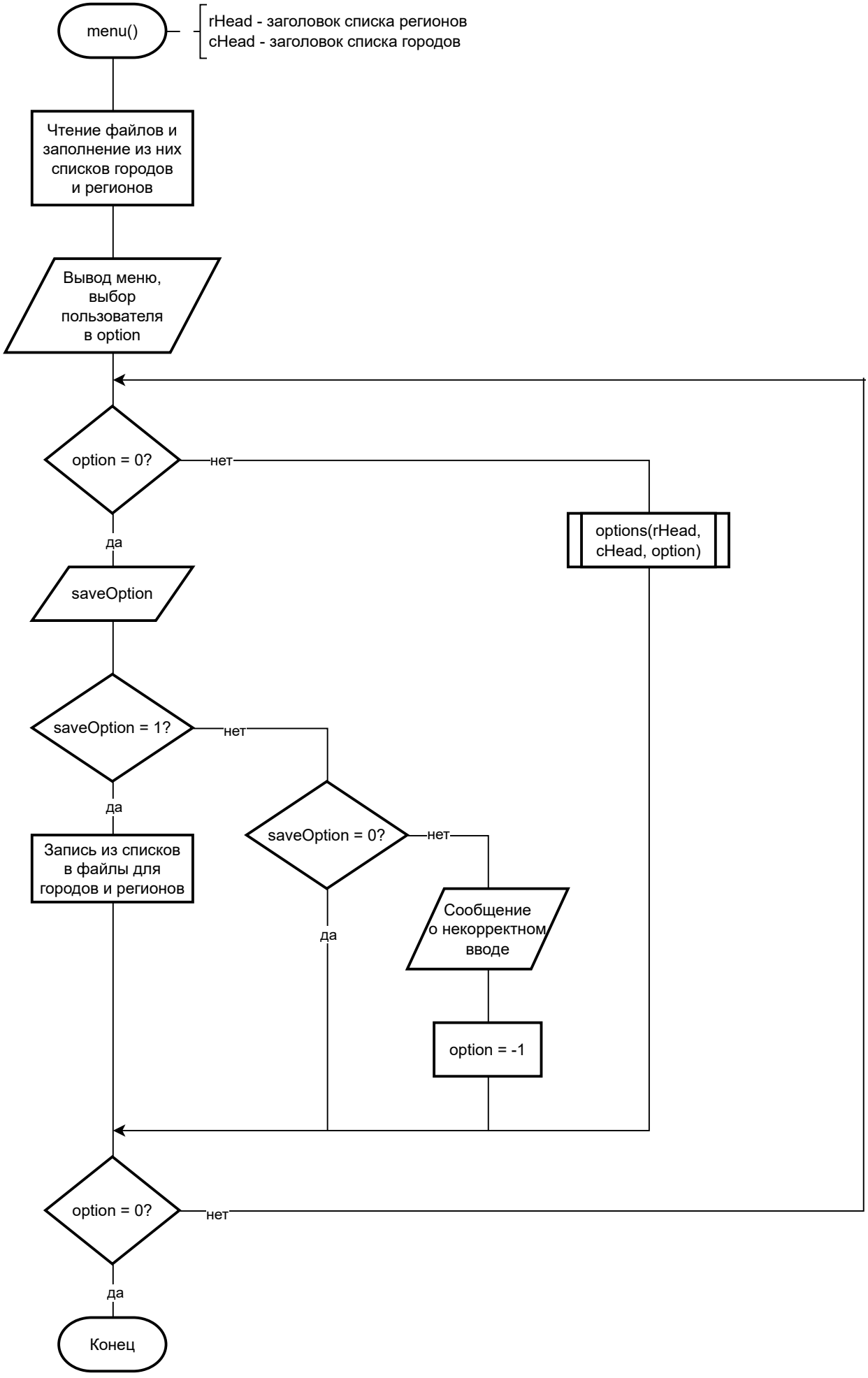
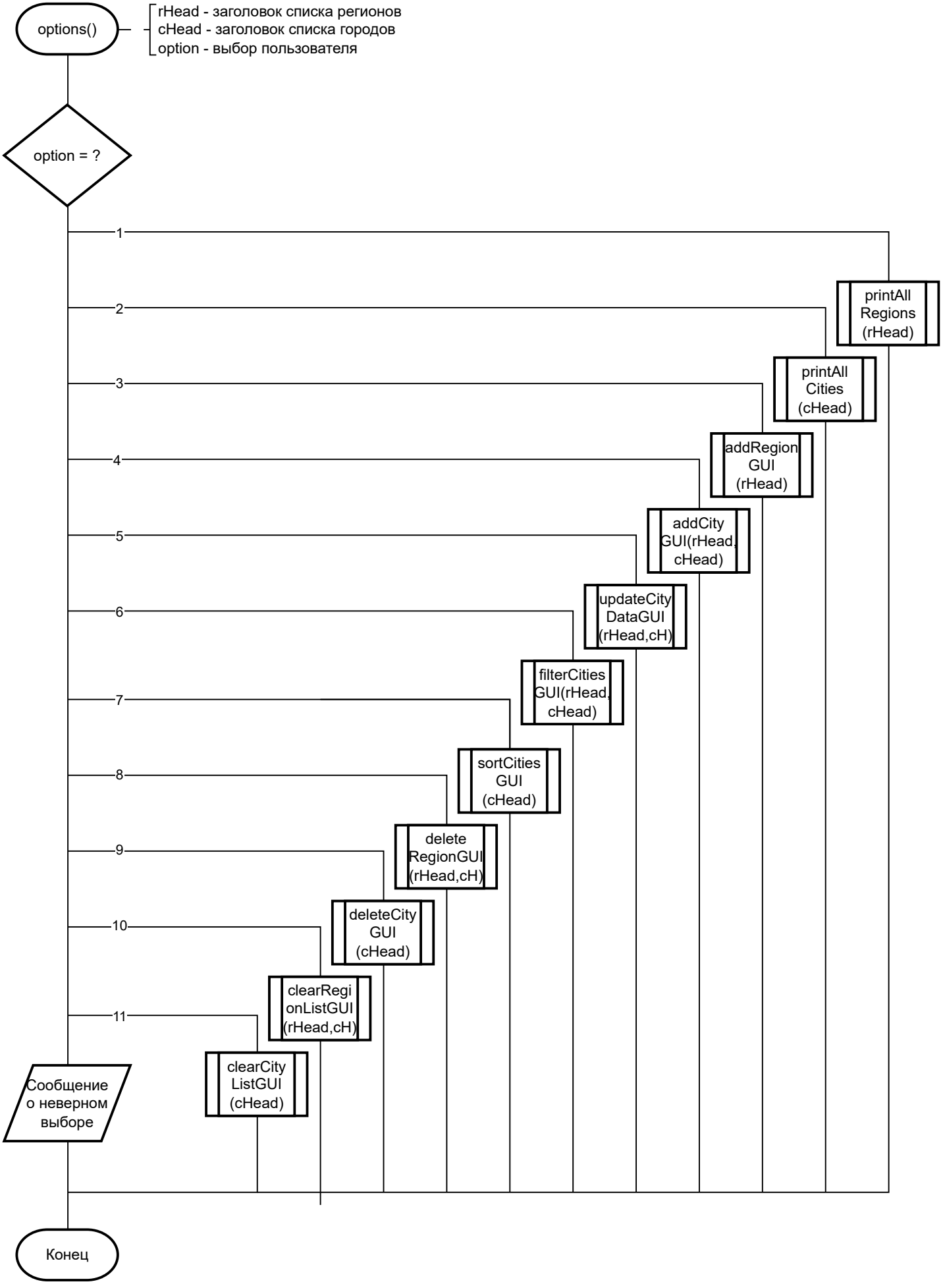
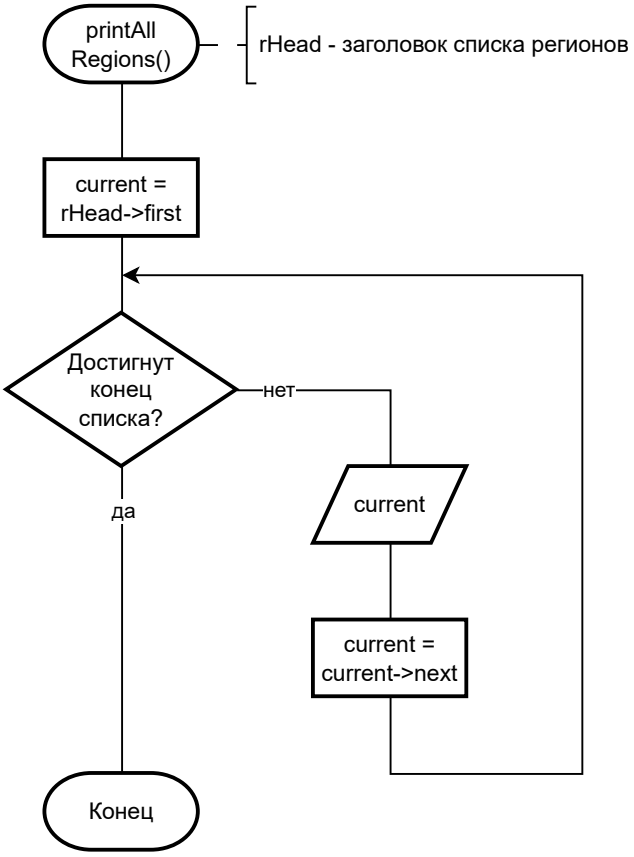


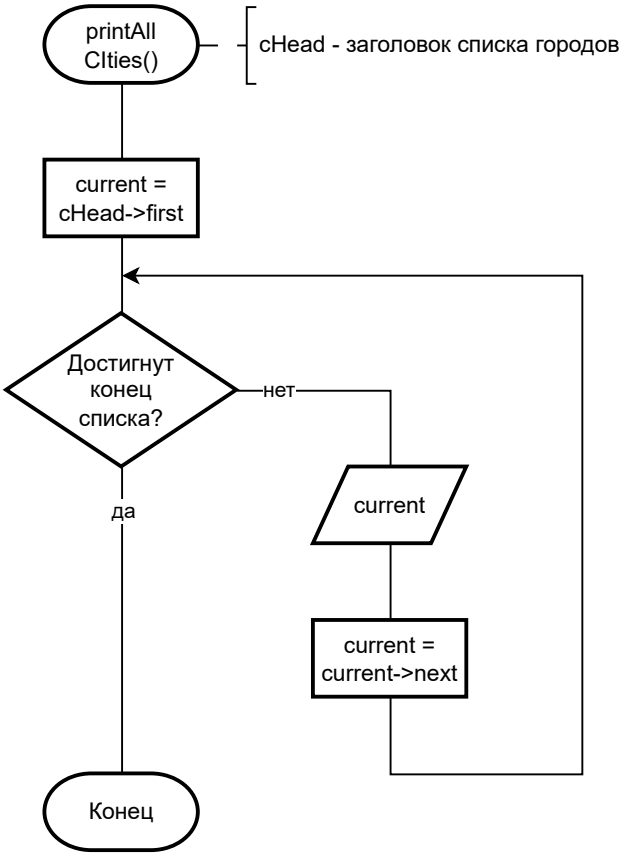
Схема алгоритма

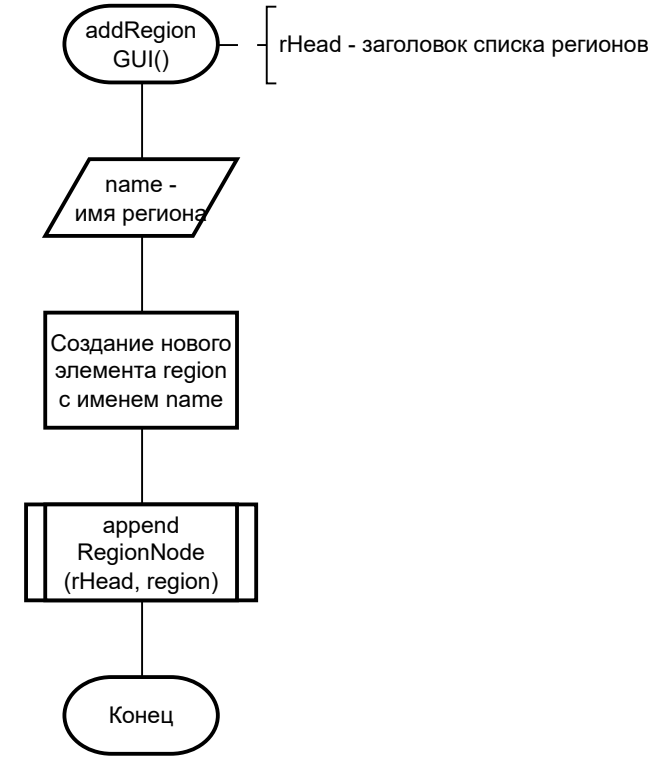


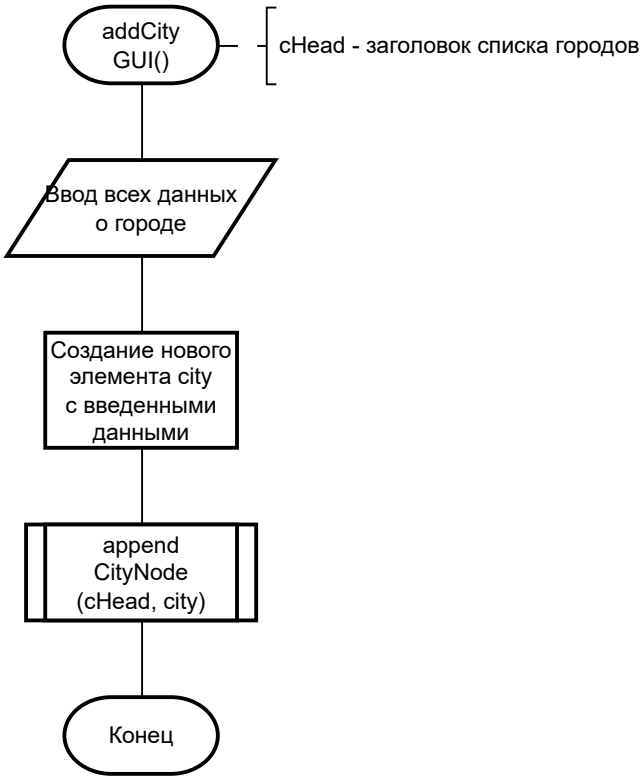

menu()
rHead - заголовок списка регионов
cHead - заголовок списка городов
Чтение файлов и заполнение из них списков городов и регионов
Вывод меню, выбор пользователя в option
option = 0?
нет
да
options(rHead, cHead, option)
saveOption
saveOption = 1?
нет
да
saveOption = 0?
нет
да
Запись из списков в файлы для городов и регионов
Сообщение о некорректном вводе
option = -1
option = 0?
нет
да
Конец

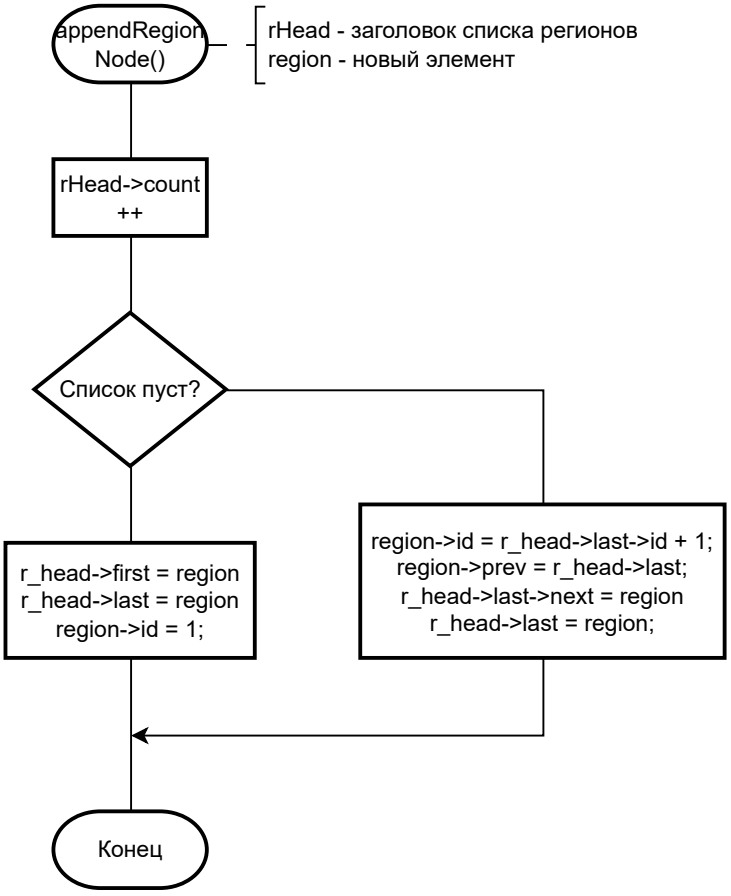

options()
rHead - заголовок списка регионов
cHead - заголовок списка городов
option - выбор пользователя
option = ?
1
2
3
4
5
6
7
8
9
10
11
printAll Regions (rHead)
printAll Cities (cHead)
addRegion GUI (rHead)
addCity GUI(rHead, cHead)
updateCity DataGUI (rHead,cH)
filterCities GUI(rHead, cHead)
sortCities GUI (cHead)
delete RegionGUI (rHead,cH)
deleteCity GUI (cHead)
clearRegi onListGUI (rHead,cH)
clearCity ListGUI (cHead)
Сообщение о неверном выборе
Конец

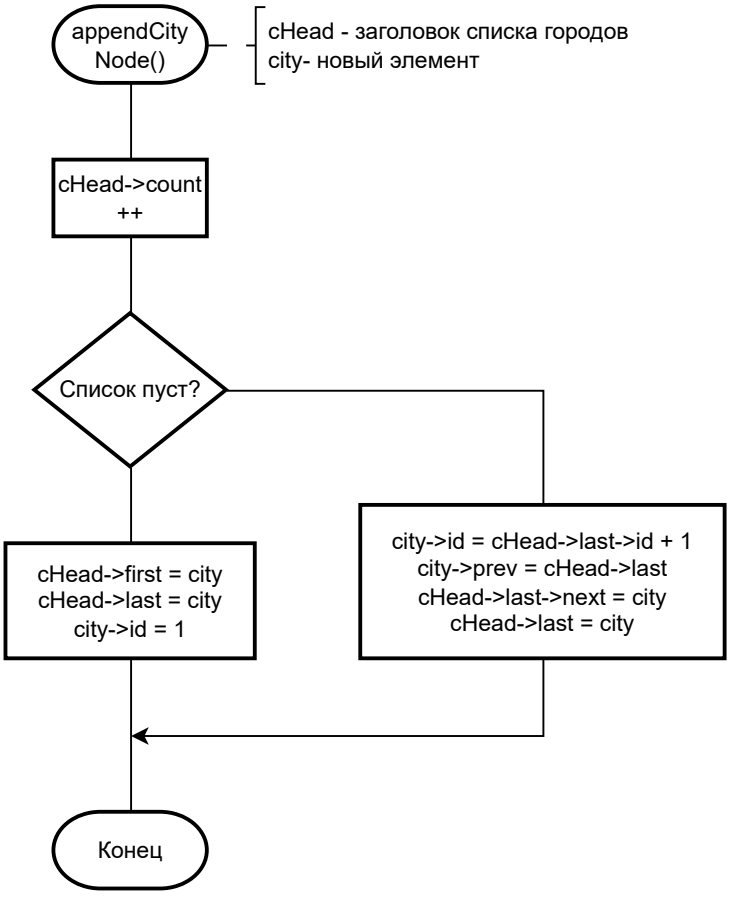

printAll Regions()
rHead - заголовок списка регионов
current = rHead->first
Достигнут конец списка?
нет
да
current
current = current->next
Конец

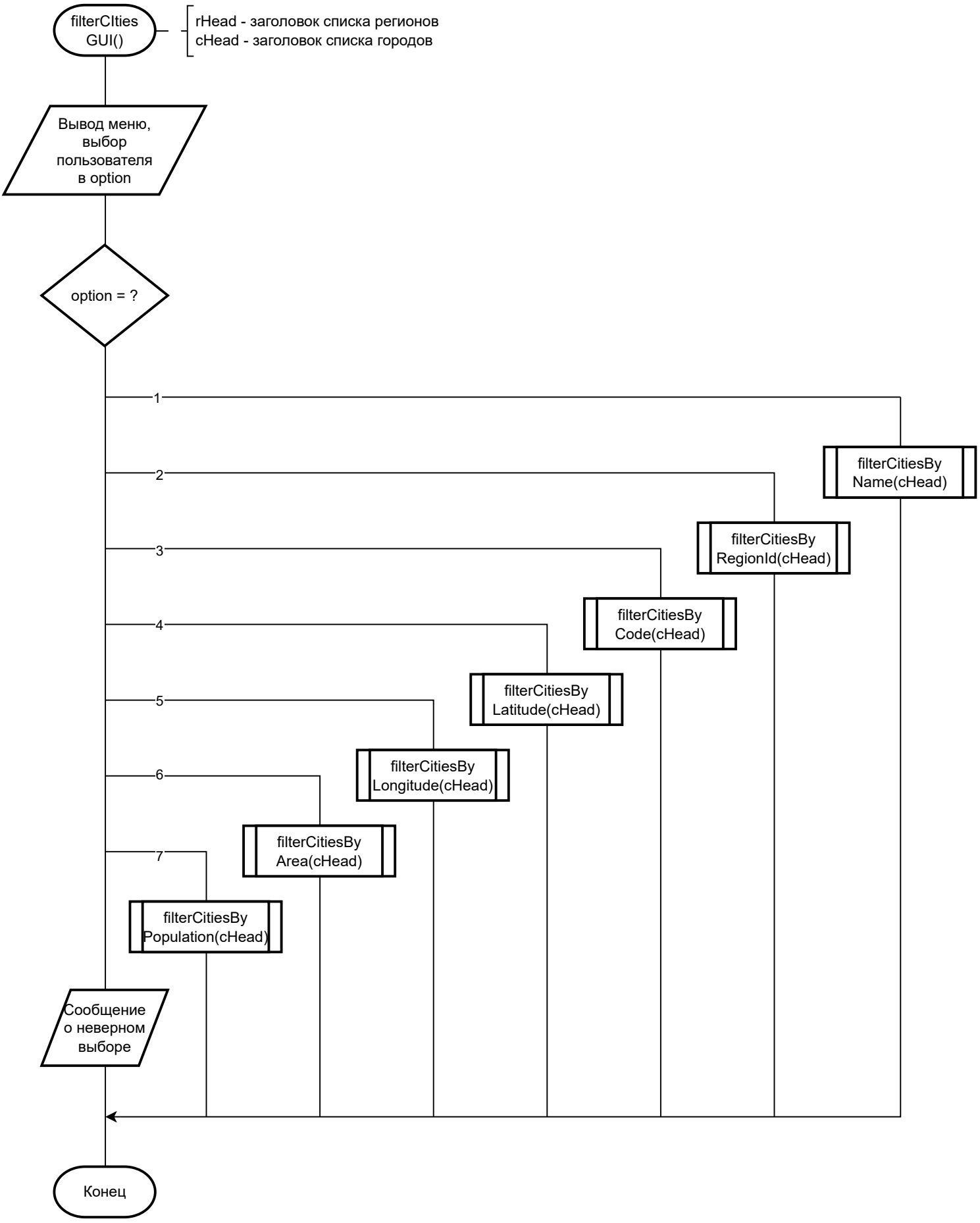

printAll CIties()
cHead - заголовок списка городов
current = cHead->first
Достигнут конец списка?
нет
да
current
current = current->next
Конец

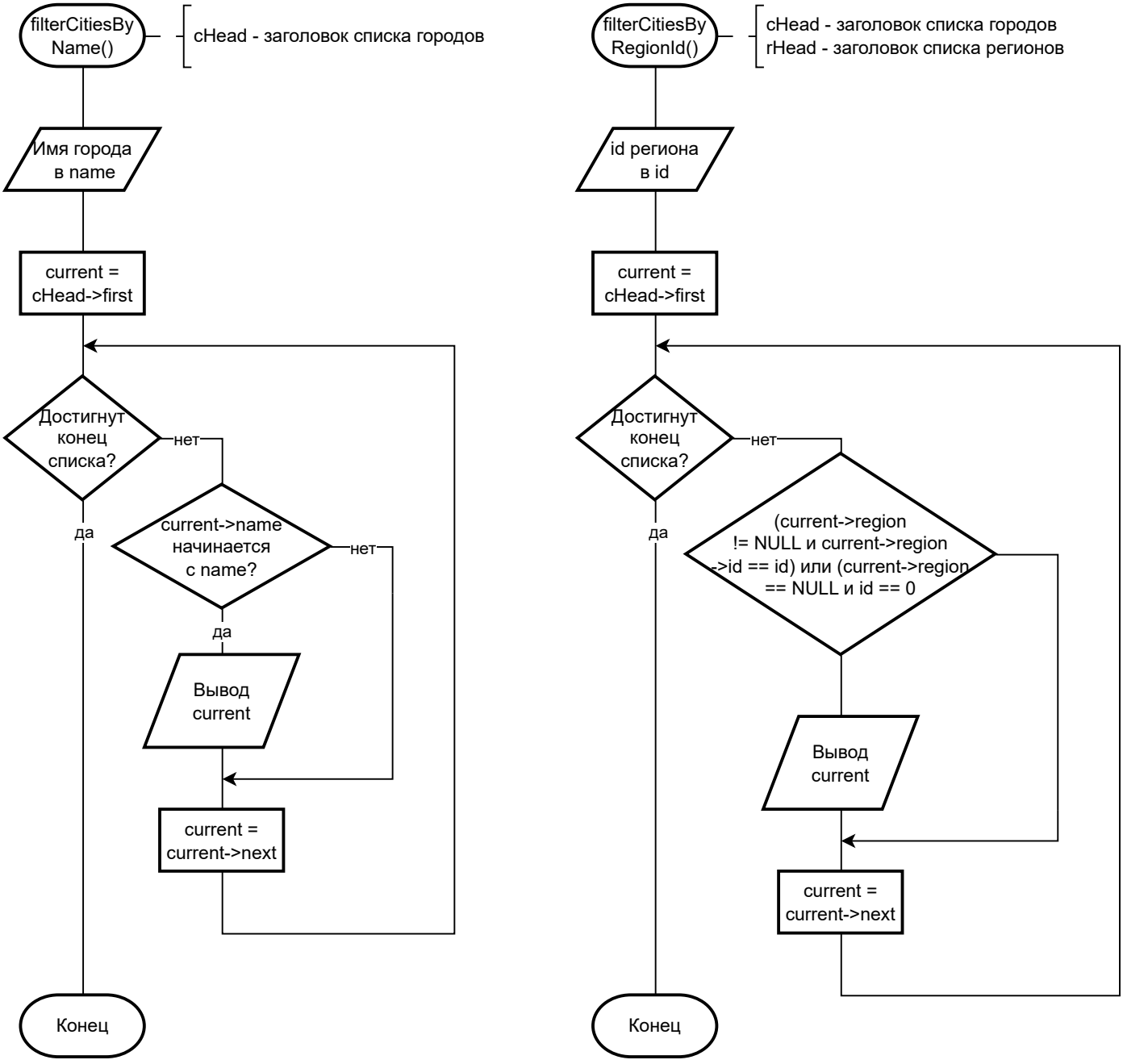

addRegion
GUI()
rHead - заголовок списка регионов
name -
имя региона
Создание нового
элемента region
с именем name
append
RegionNode
(rHead, region)
Конец

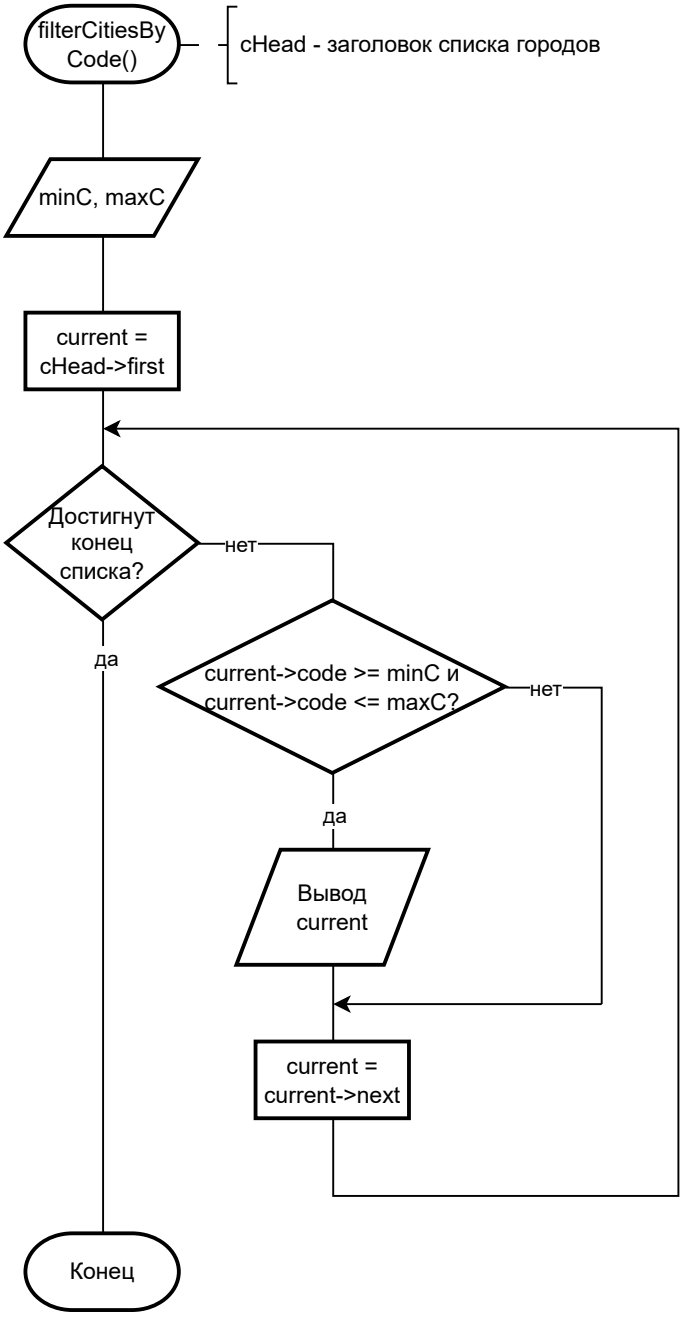

addCity
GUI()
cHead - заголовок списка городов
Ввод всех данных
о городе
Создание нового
элемента city
с введенными
данными
append
CityNode
(cHead, city)
Конец

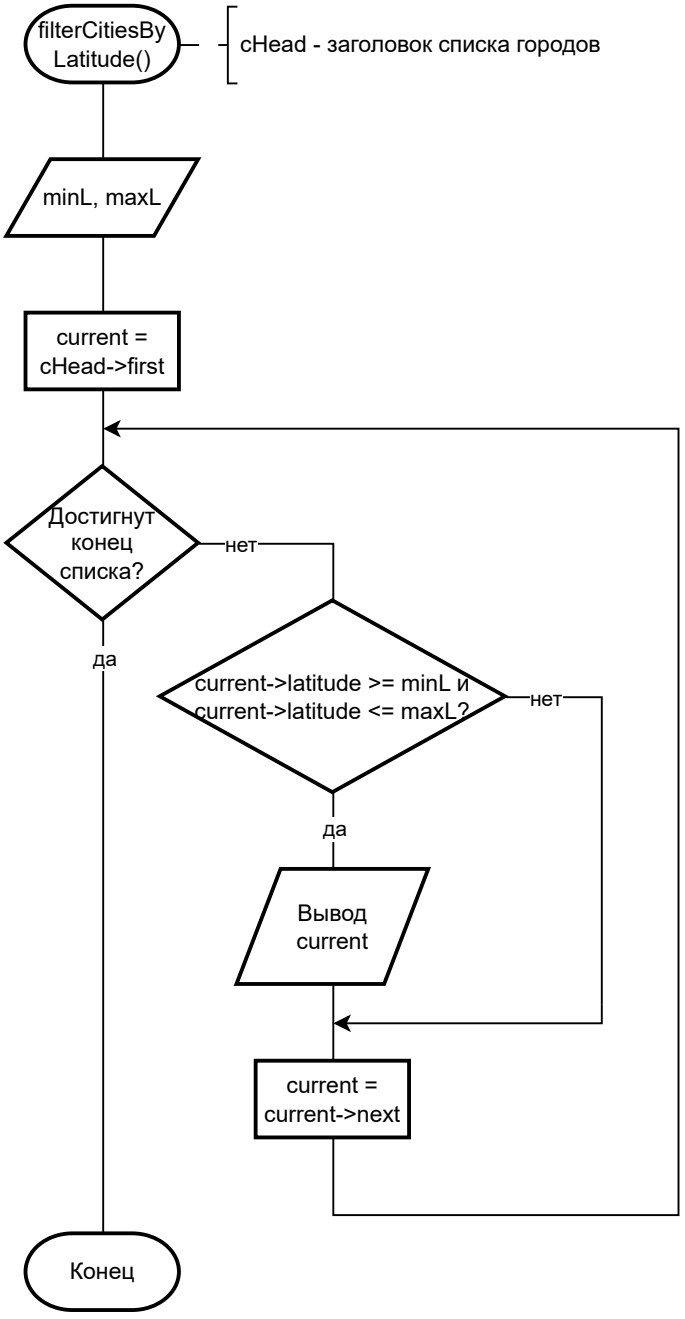

appendRegion Node()
rHead - заголовок списка регионов
region - новый элемент
rHead->count ++
Список пуст?
r_head->first = region
r_head->last = region
region->id = 1;
region->id = r_head->last->id + 1;
region->prev = r_head->last;
r_head->last->next = region
r_head->last = region;
Конец

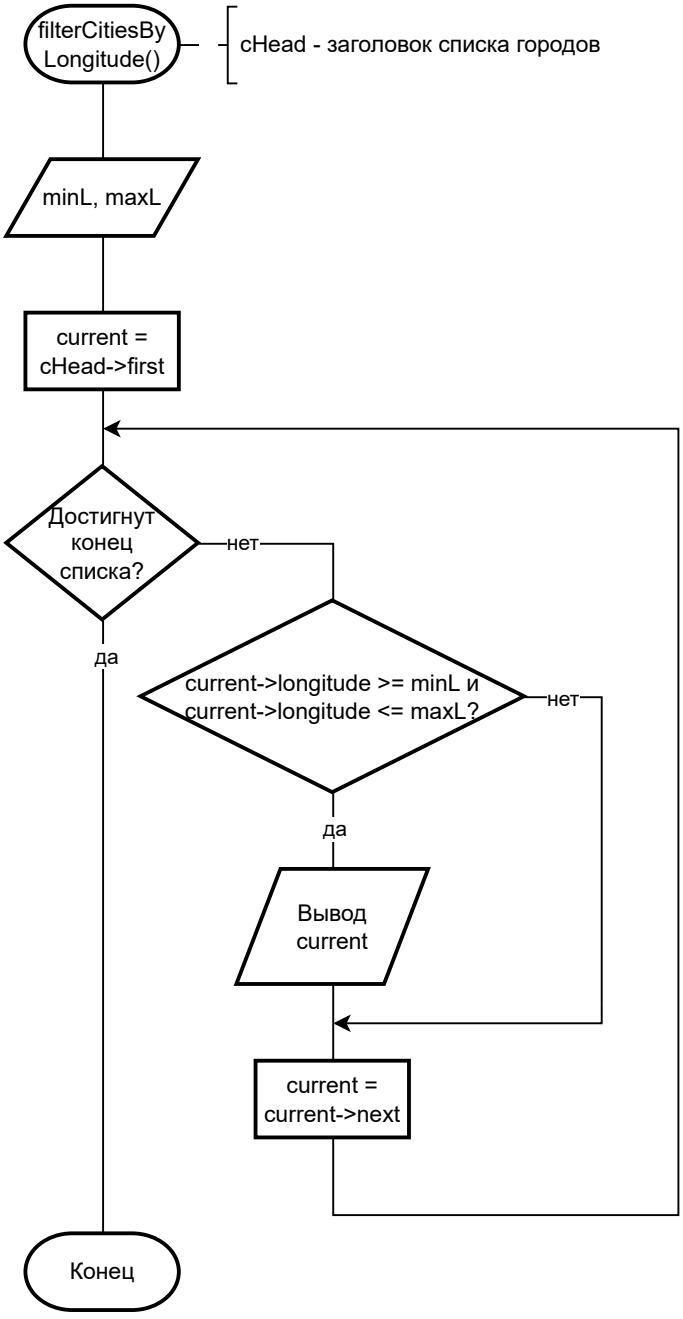

appendCity Node()
cHead - заголовок списка городов
city- новый элемент
cHead->count ++
Список пуст?
cHead->first = city
cHead->last = city
city->id = 1
city->id = cHead->last->id + 1
city->prev = cHead->last
cHead->last->next = city
cHead->last = city
Конец

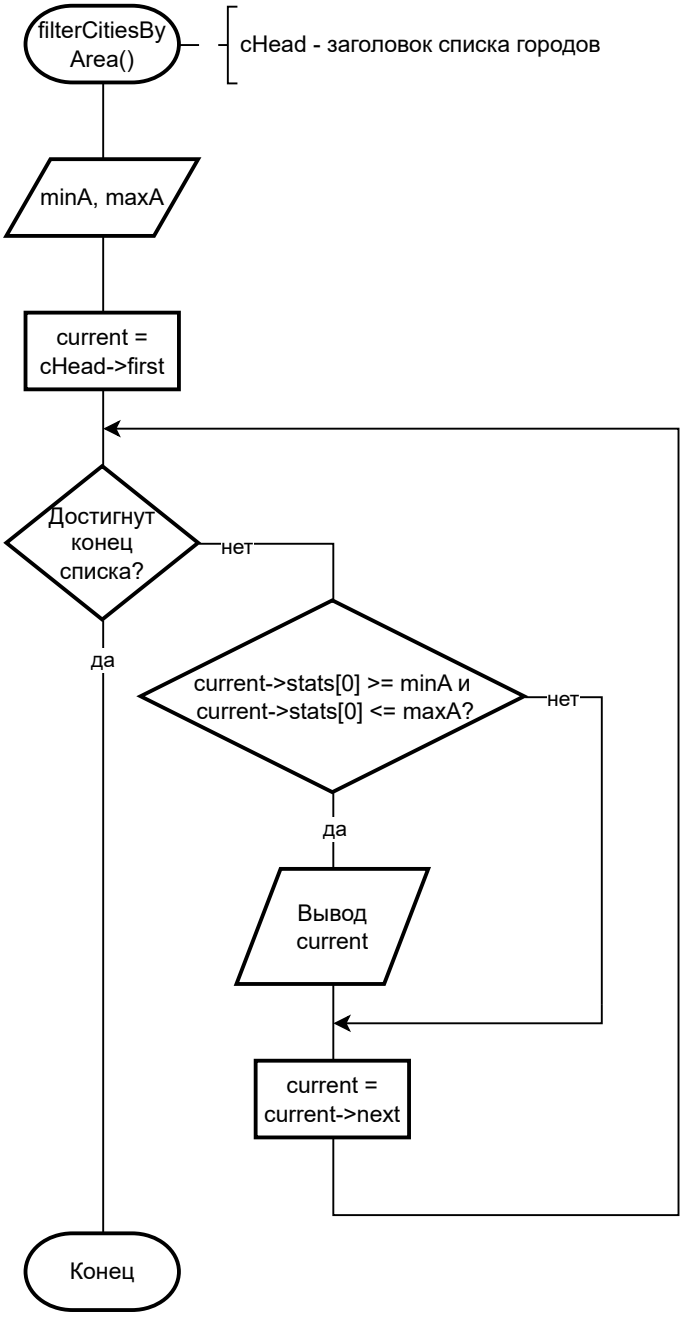

filterCIties GUI()
rHead - заголовок списка регионов
cHead - заголовок списка городов
Вывод меню, выбор пользователя в option
option = ?
1
2
3
4
5
6
7
filterCitiesBy Name(cHead)
filterCitiesBy RegionId(cHead)
filterCitiesBy Code(cHead)
filterCitiesBy Latitude(cHead)
filterCitiesBy Longitude(cHead)
filterCitiesBy Area(cHead)
filterCitiesBy Population(cHead)
Сообщение о неверном выборе
Конец

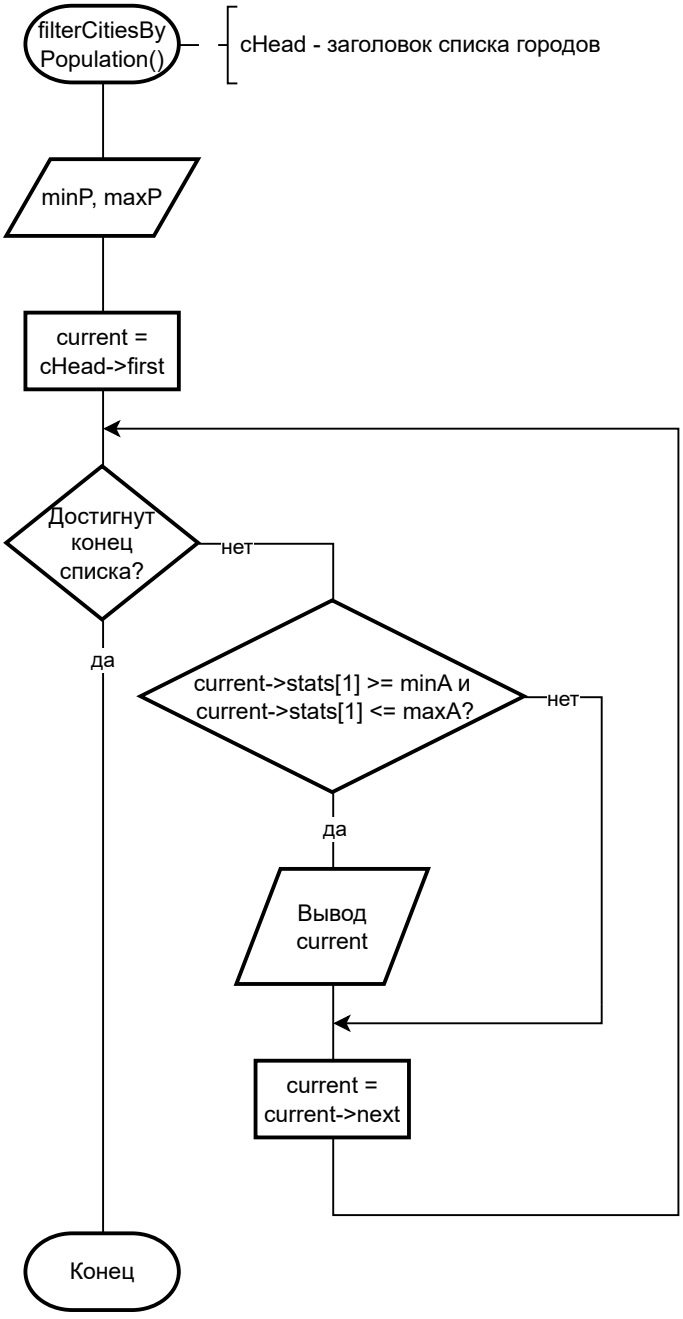

filterCitiesBy Name()
cHead - заголовок списка городов
Имя города в name
current = cHead->first
Достигнут конец списка?
нет
да
current->name начинается с name?
нет
да
Вывод current
current = current->next
Конец
filterCitiesBy RegionId()
cHead - заголовок списка городов
rHead - заголовок списка регионов
id региона в id
current = cHead->first
Достигнут конец списка?
нет
да
(current->region != NULL и current->region ->id == id) или (current->region == NULL и id == 0
Вывод current
current = current->next
Конец

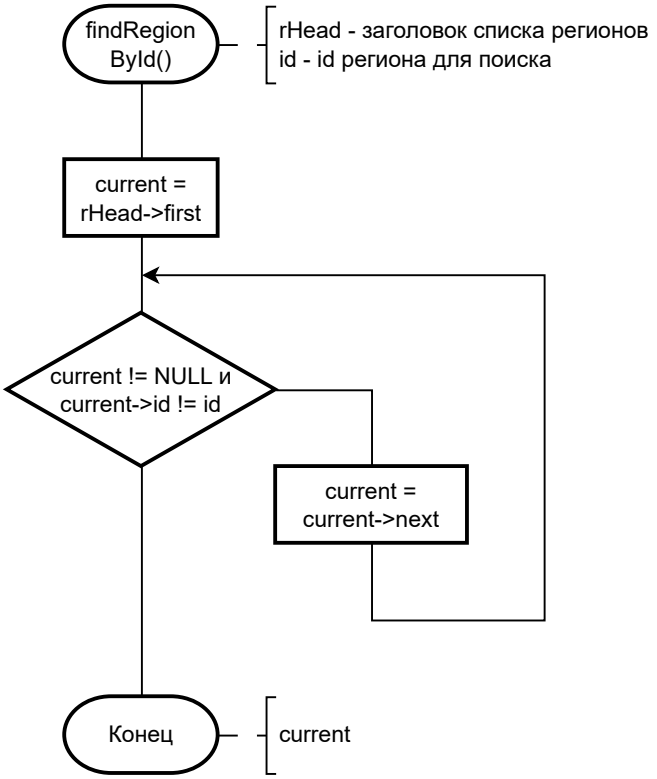

filterCitiesBy Code()
cHead - заголовок списка городов
minC, maxC
current = cHead->first
Достигнут конец списка?
нет
да
current->code >= minC и current->code <= maxC?
нет
да
Вывод current
current = current->next
Конец

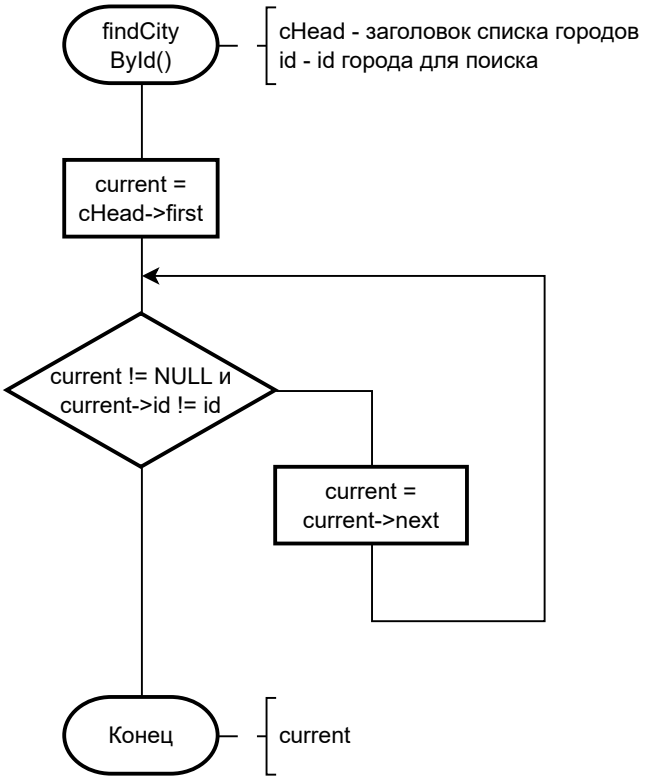

filterCitiesBy Latitude()
cHead - заголовок списка городов
minL, maxL
current = cHead->first
Достигнут конец списка?
нет
да
current->latitude >= minL и current->latitude <= maxL?
нет
да
Вывод current
current = current->next
Конец

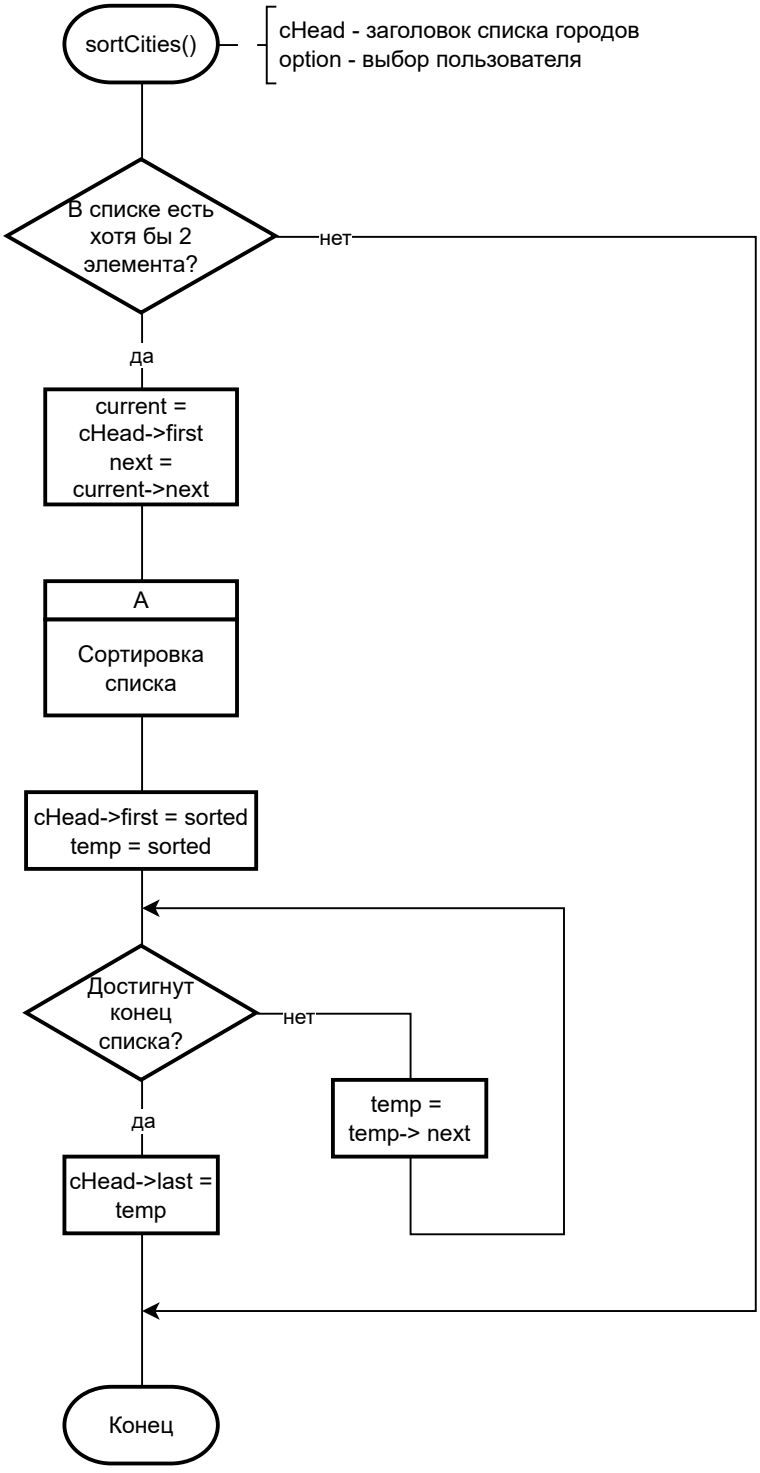

filterCitiesBy Longitude()
cHead - заголовок списка городов
minL, maxL
current = cHead->first
Достигнут конец списка?
нет
да
current->longitude >= minL и current->longitude <= maxL?
нет
да
Вывод current
current = current->next
Конец

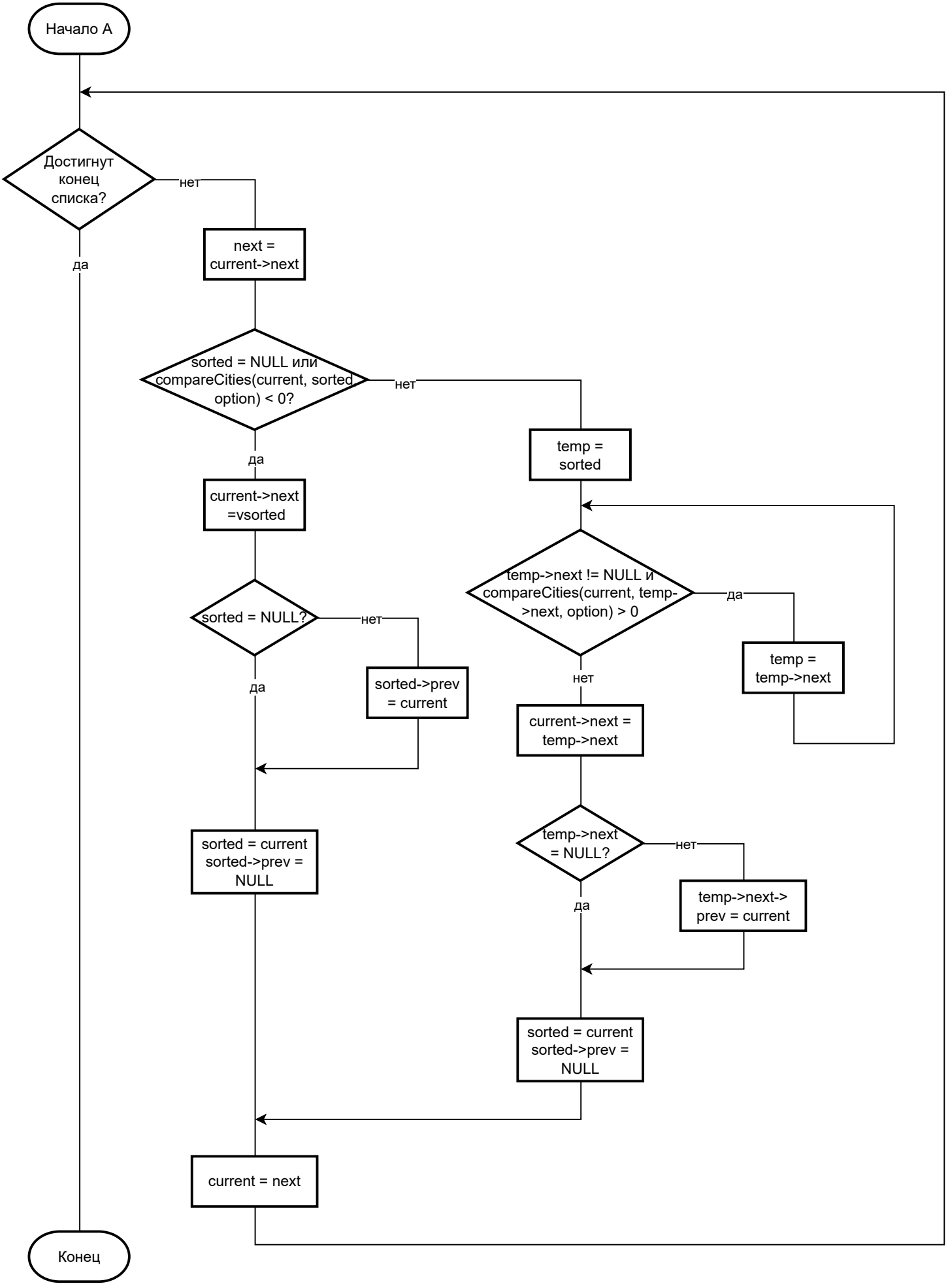

filterCitiesBy Area()
cHead - заголовок списка городов
minA, maxA
current = cHead->first
Достигнут конец списка?
нет
да
current->stats[0] >= minA и current->stats[0] <= maxA?
нет
да
Вывод current
current = current->next
Конец

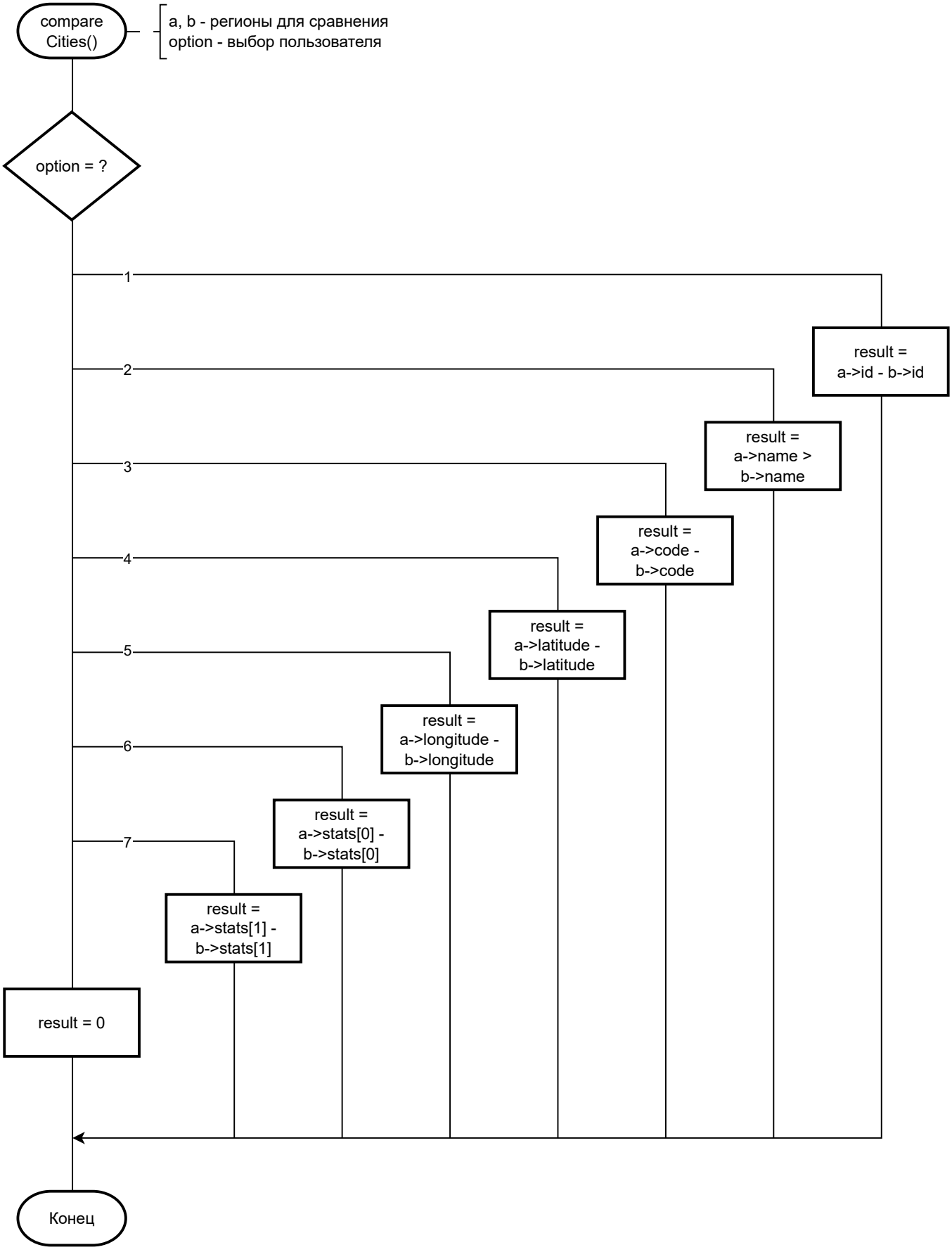

filterCitiesBy Population()
cHead - заголовок списка городов
minP, maxP
current = cHead->first
Достигнут конец списка?
нет
да
current->stats[1] >= minA и current->stats[1] <= maxA?
нет
да
Вывод current
current = current->next
Конец

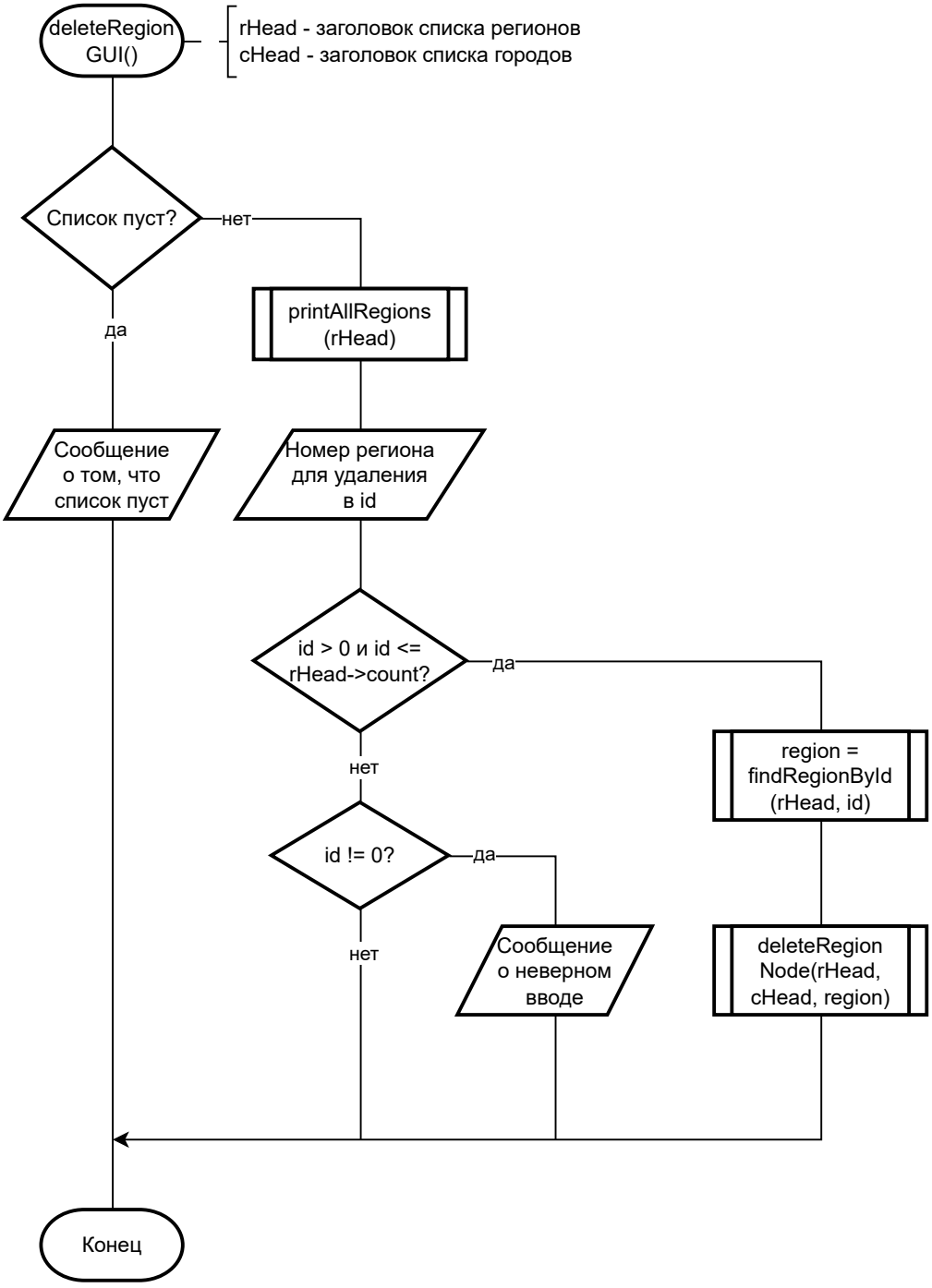

findRegion ById()
rHead - заголовок списка регионов
id - id региона для поиска
current = rHead->first
current != NULL и current->id != id
current = current->next
Конец
current

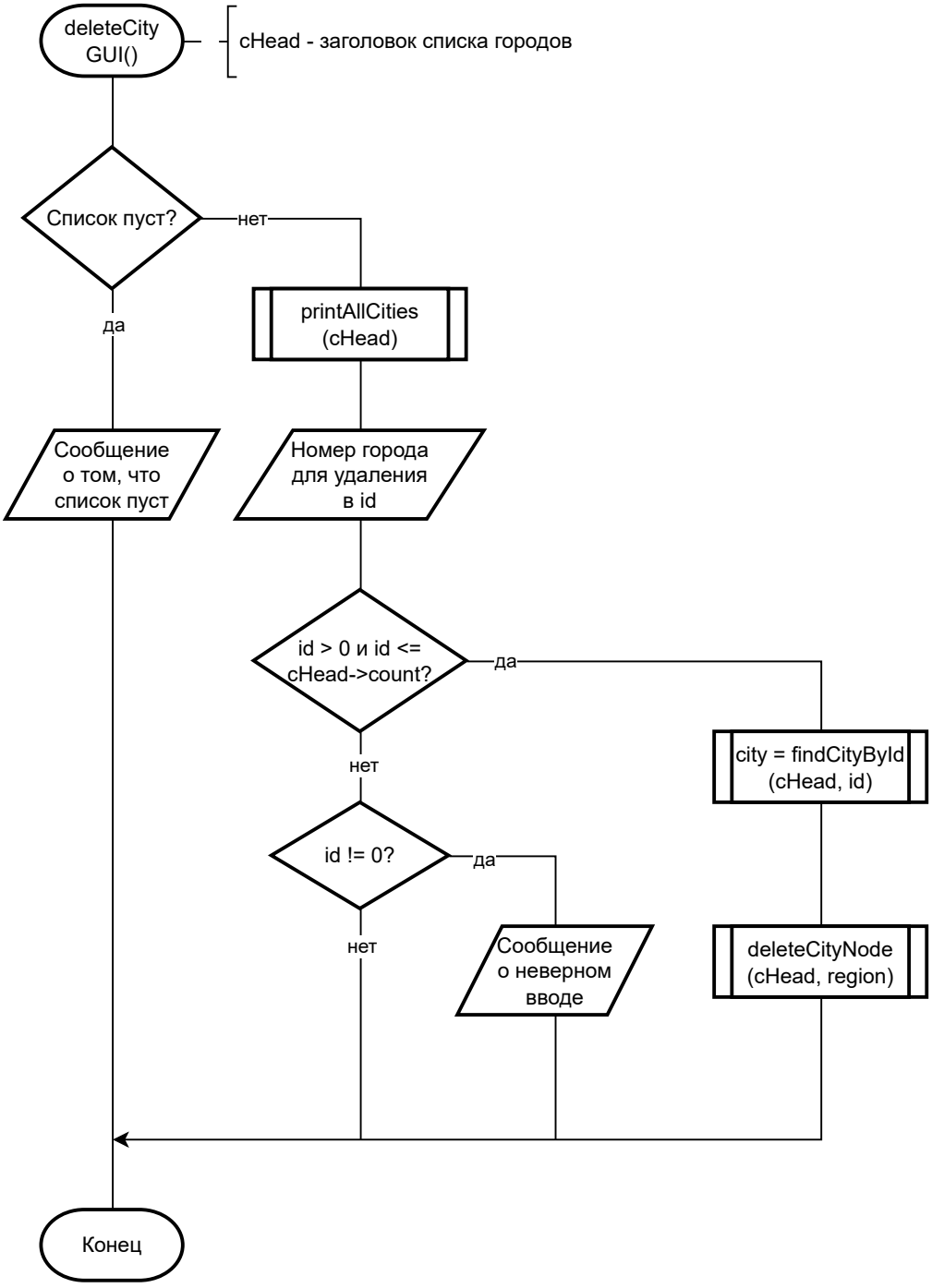

findCity ById()
cHead - заголовок списка городов
id - id города для поиска
current = cHead->first
current != NULL и current->id != id
current = current->next
Конец
current

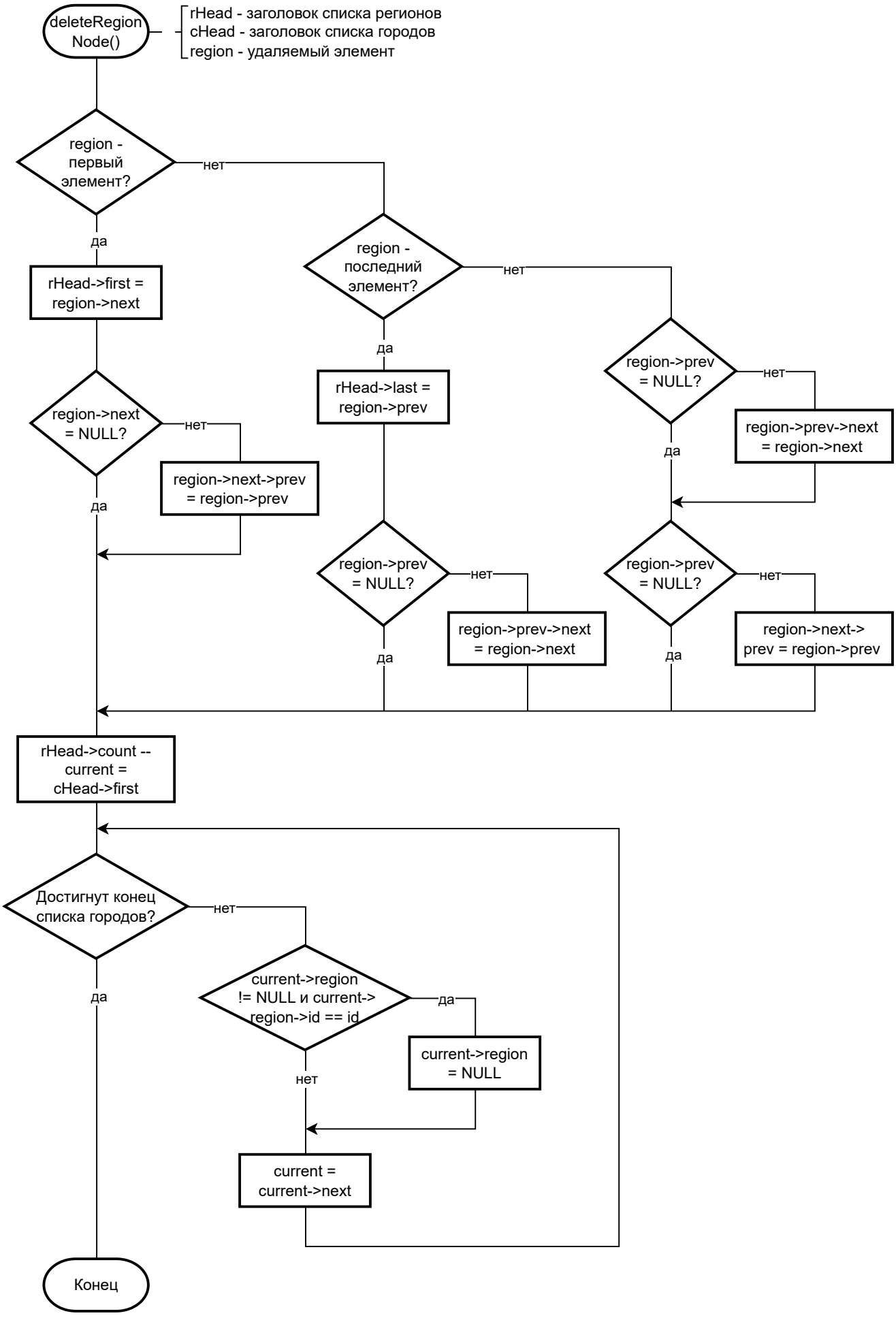

sortCities()
cHead - заголовок списка городов
option - выбор пользователя
В списке есть хотя бы 2 элемента?
нет
да
current = cHead->first
next = current->next
A
Сортировка списка
cHead->first = sorted
temp = sorted
Достигнут конец списка?
нет
да
temp = temp-> next
cHead->last = temp
Конец

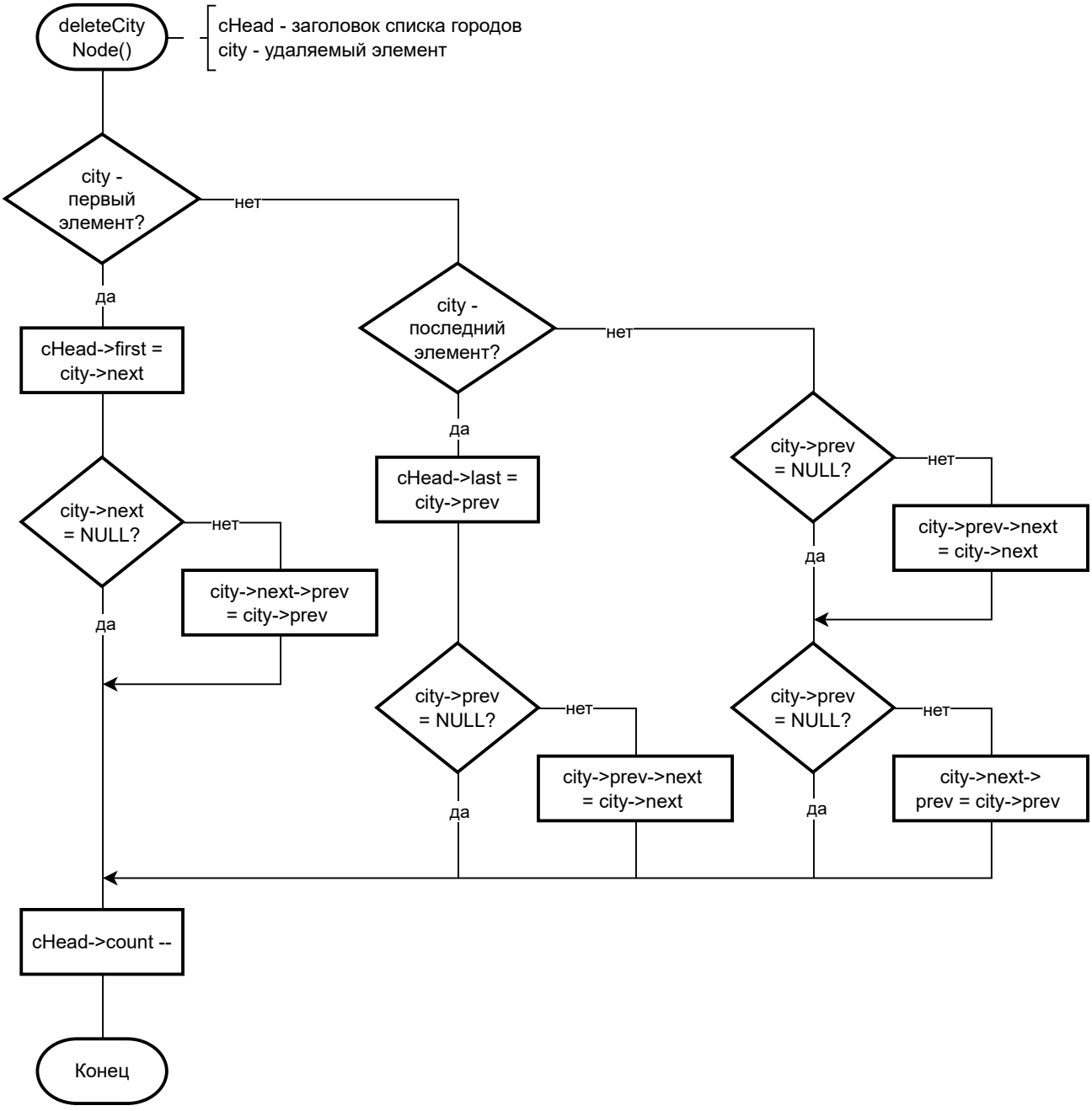

Начало A
Достигнут конец списка?
нет
да
next = current->next
sorted = NULL или compareCities(current, sorted option) < 0?
нет
да
current->next =vsorted
sorted = NULL?
нет
да
sorted->prev = current
sorted = current sorted->prev = NULL
current = next
temp = sorted
temp->next != NULL и compareCities(current, temp->next, option) > 0
да
temp = temp->next
нет
current->next = temp->next
temp->next = NULL?
нет
да
temp->next-> prev = current
sorted = current sorted->prev = NULL
Конец

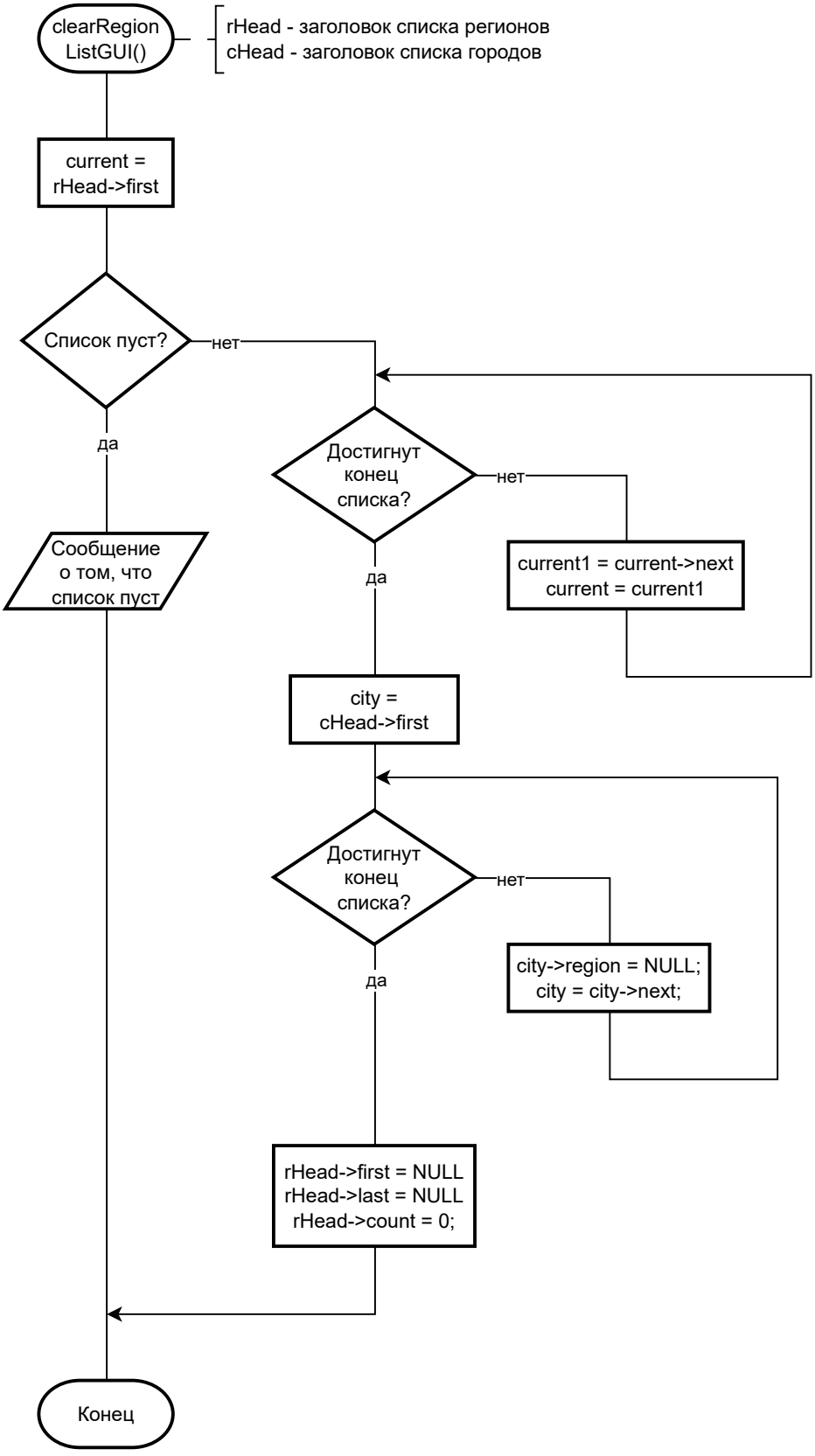

compare Cities()
a, b - регионы для сравнения
option - выбор пользователя
option = ?
1
result = a->id - b->id
2
result = a->name > b->name
3
result = a->code - b->code
4
result = a->latitude - b->latitude
5
result = a->longitude - b->longitude
6
result = a->stats[0] - b->stats[0]
7
result = a->stats[1] - b->stats[1]
result = 0
Конец

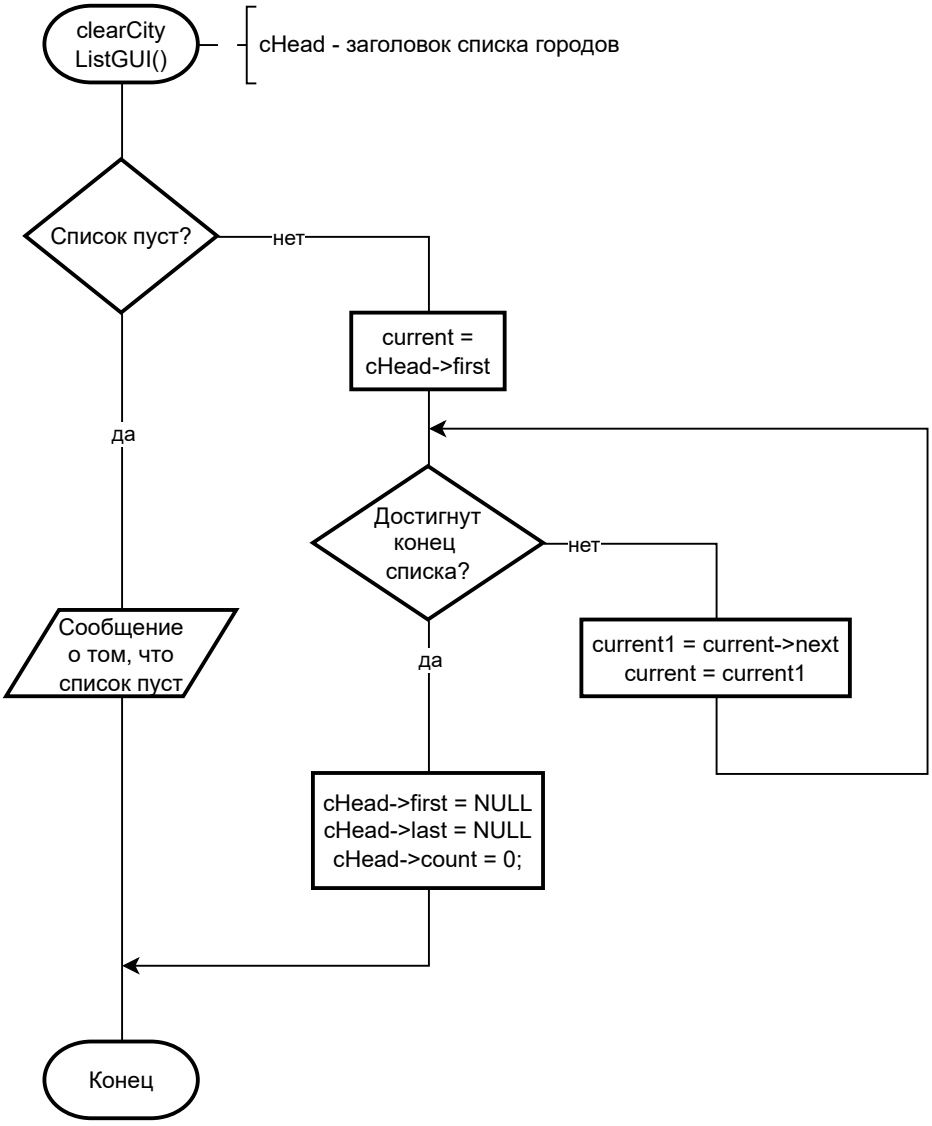

deleteRegion GUI()
rHead - заголовок списка регионов
cHead - заголовок списка городов
Список пуст?
нет
да
printAllRegions (rHead)
Сообщение о том, что список пуст
Номер региона для удаления в id
id > 0 и id <= rHead->count?
да
нет
region = findRegionById (rHead, id)
id != 0?
да
нет
Сообщение о неверном вводе
deleteRegion Node(rHead, cHead, region)
Конец

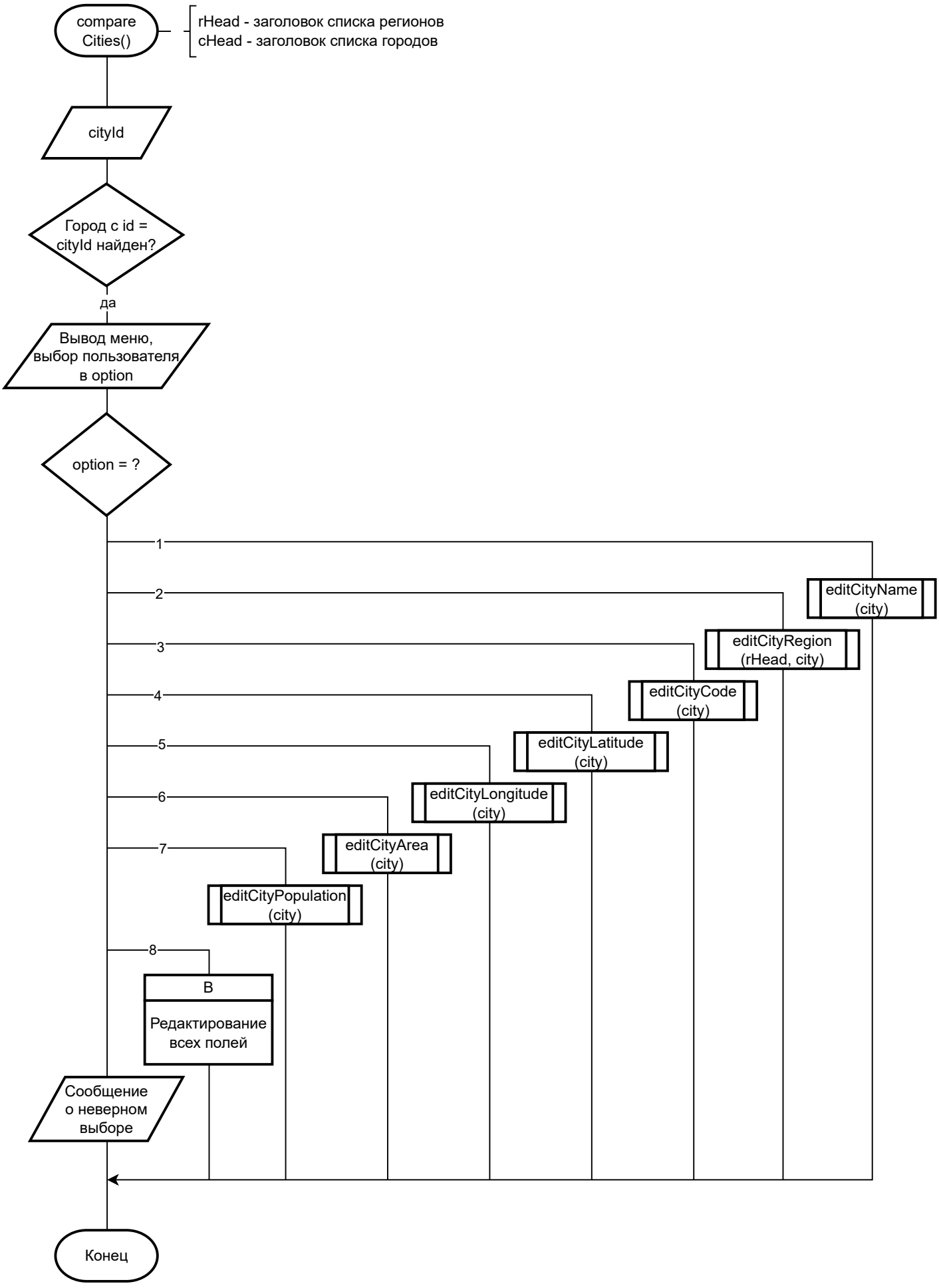

deleteCity GUI()
cHead - заголовок списка городов
Список пуст?
нет
да
printAllCities (cHead)
Сообщение о том, что список пуст
Номер города для удаления в id
id > 0 и id <= cHead->count?
да
нет
city = findCityById (cHead, id)
id != 0?
да
нет
Сообщение о неверном вводе
deleteCityNode (cHead, region)
Конец

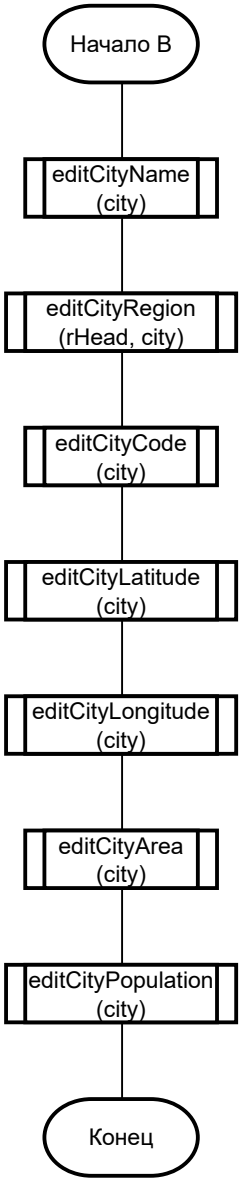

deleteRegion Node()
rHead - заголовок списка регионов
cHead - заголовок списка городов
region - удаляемый элемент
region - первый элемент?
нет
да
region - последний элемент?
нет
да
rHead->first = region->next
rHead->last = region->prev
region->prev = NULL?
нет
да
region->prev->next = region->next
region->next = NULL?
нет
да
region->next->prev = region->prev
region->prev = NULL?
нет
да
region->prev->next = region->next
region->prev = NULL?
нет
да
region->next-> prev = region->prev
rHead->count -- current = cHead->first
Достигнут конец списка городов?
нет
да
current->region != NULL и current-> region->id == id
да
нет
current->region = NULL
current = current->next
Конец


deleteCity Node()
cHead - заголовок списка городов
city - удаляемый элемент
city - первый элемент?
нет
да
cHead->first = city->next
city->next = NULL?
нет
city->next->prev = city->prev
да
city - последний элемент?
нет
да
cHead->last = city->prev
city->prev = NULL?
нет
city->prev->next = city->next
да
city->prev = NULL?
нет
да
city->prev->next = city->next
city->prev = NULL?
нет
да
city->next-> prev = city->prev
cHead->count --
Конец

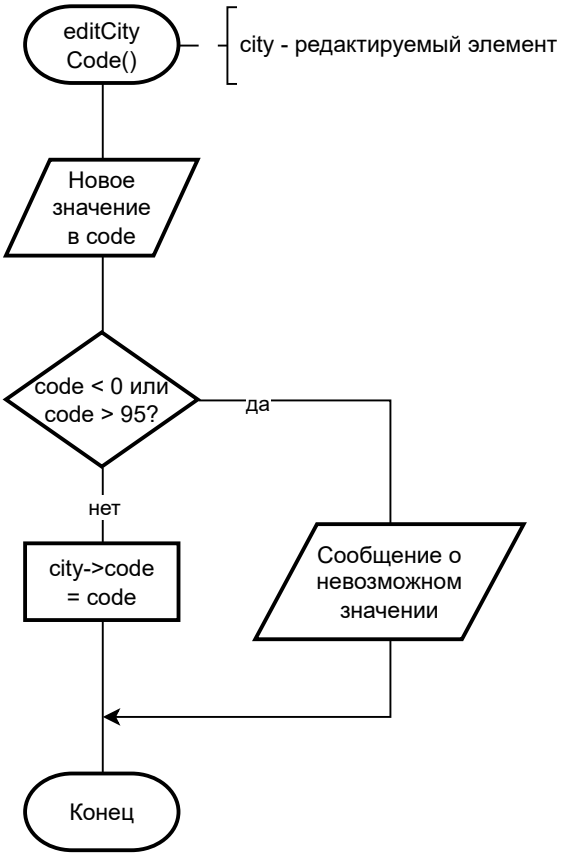

clearRegion ListGUI()
rHead - заголовок списка регионов
cHead - заголовок списка городов
current = rHead->first
Список пуст?
нет
да
Достигнут конец списка?
нет
да
current1 = current->next
current = current1
Сообщение о том, что список пуст
city = cHead->first
Достигнут конец списка?
нет
да
city->region = NULL;
city = city->next;
rHead->first = NULL
rHead->last = NULL
rHead->count = 0;
Конец

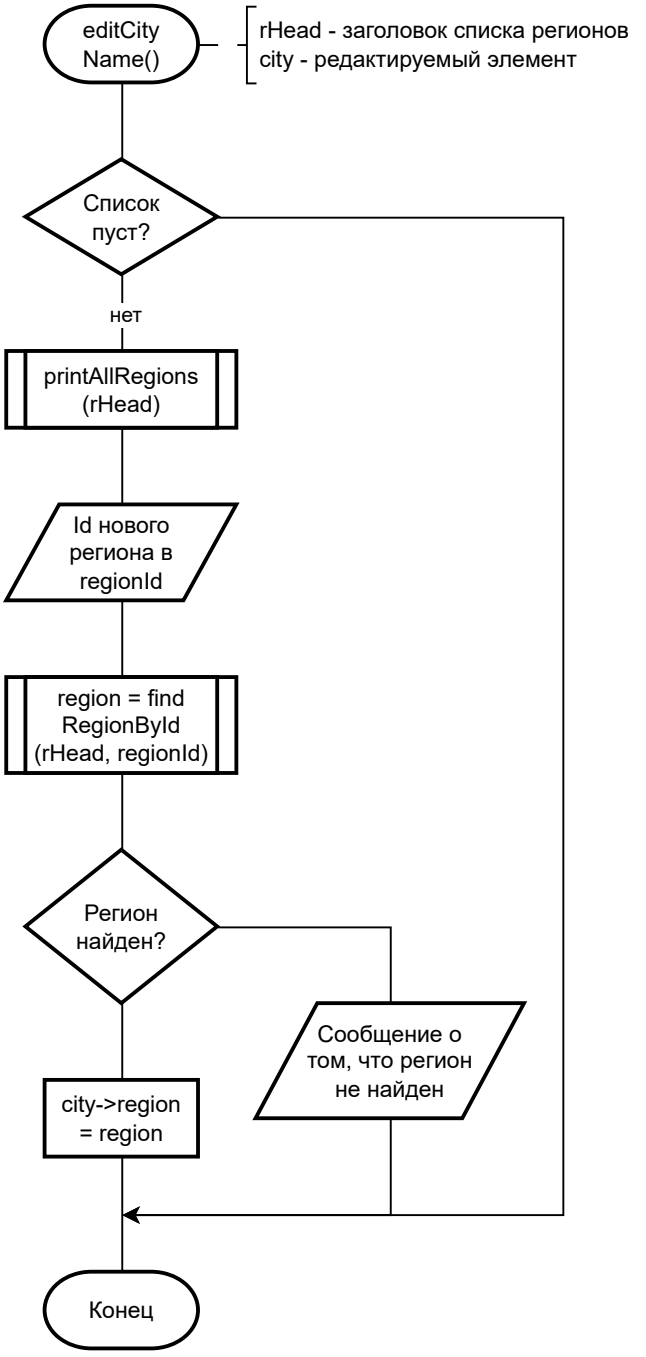

clearCity ListGUI()
cHead - заголовок списка городов
Список пуст?
нет
да
current = cHead->first
Достигнут конец списка?
нет
да
Сообщение о том, что список пуст
current1 = current->next
current = current1
cHead->first = NULL
cHead->last = NULL
cHead->count = 0;
Конец

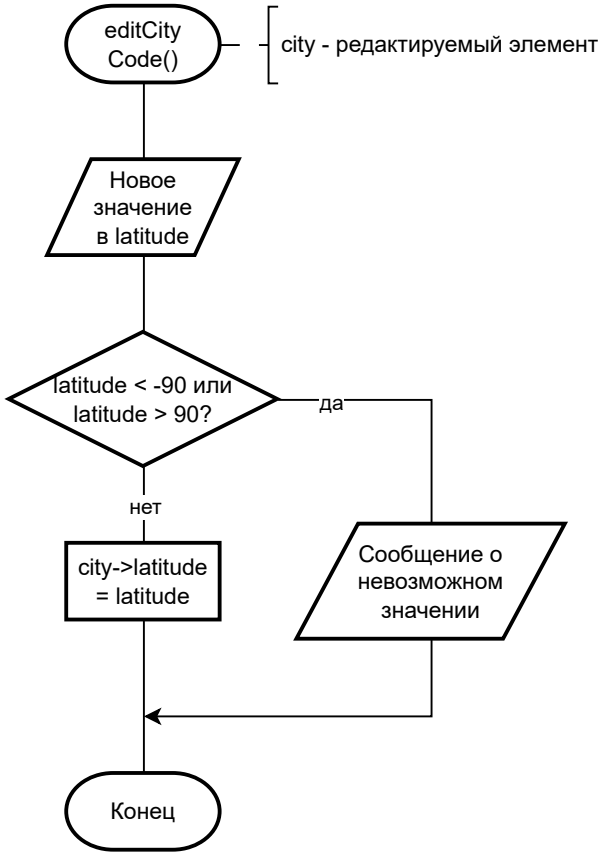

compare Cities()
rHead - заголовок списка регионов
cHead - заголовок списка городов
cityId
Город с id = cityId найден?
да
Вывод меню, выбор пользователя в option
option = ?
1
2
3
4
5
6
7
8
editCityName (city)
editCityRegion (rHead, city)
editCityCode (city)
editCityLatitude (city)
editCityLongitude (city)
editCityArea (city)
editCityPopulation (city)
B
Редактирование всех полей
Сообщение о неверном выборе
Конец

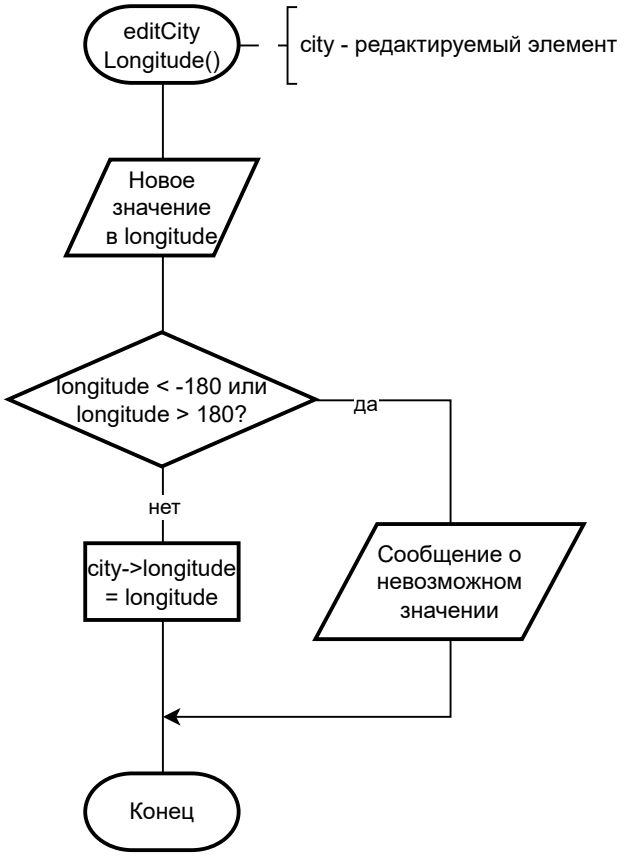

Начало B
editCityName (city)
editCityRegion (rHead, city)
editCityCode (city)
editCityLatitude (city)
editCityLongitude (city)
editCityArea (city)
editCityPopulation (city)
Конец

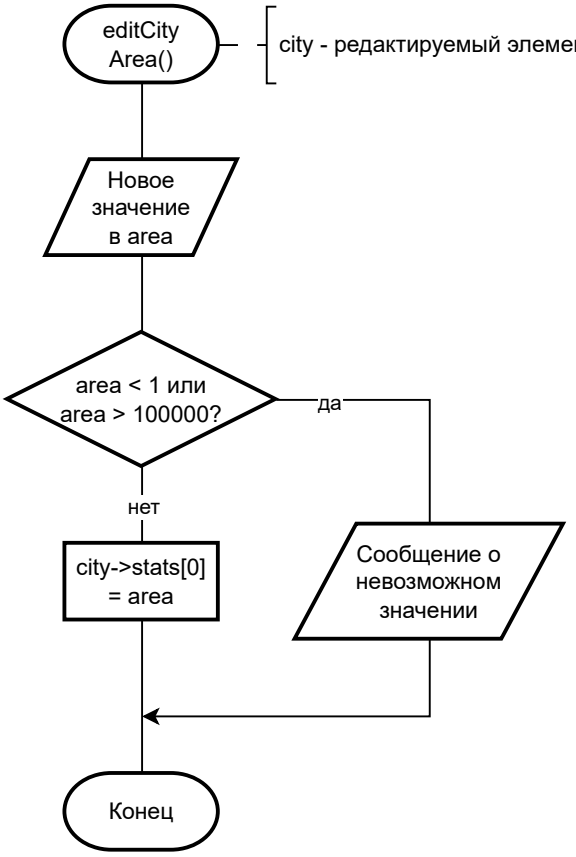

editCity Name()
city - редактируемый элемент
Новое имя в city
city->name = name
Конец

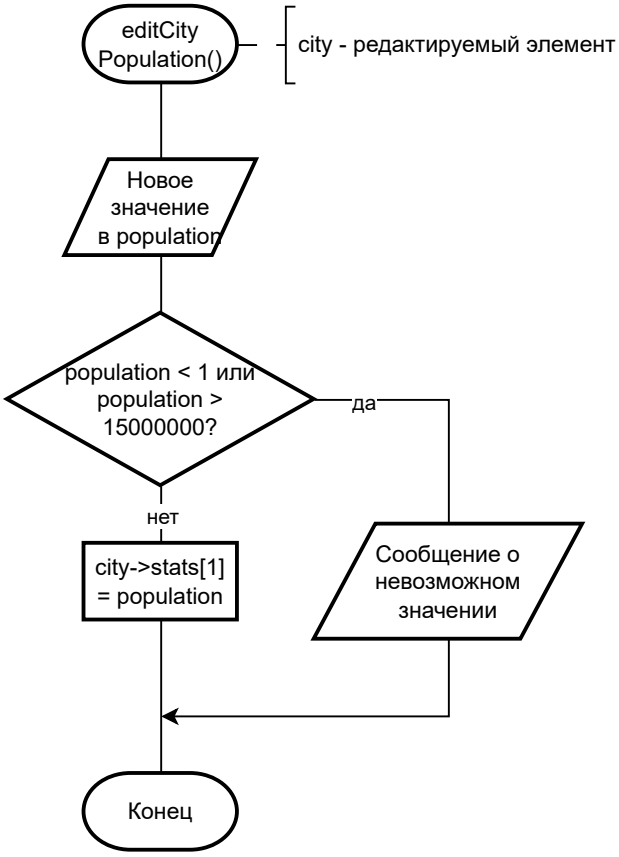

editCity Code()
city - редактируемый элемент
Новое значение в code
code < 0 или code > 95?
да
нет
city->code = code
Сообщение о невозможном значении
Конец


editCity Name()
rHead - заголовок списка регионов
city - редактируемый элемент
Список пуст?
нет
printAllRegions (rHead)
Id нового региона в regionId
region = find RegionById (rHead, regionId)
Регион найден?
Сообщение о том, что регион не найден
city->region = region
Конец


editCity Code()
city - редактируемый элемент
Новое значение в latitude
latitude < -90 или latitude > 90?
да
нет
city->latitude = latitude
Сообщение о невозможном значении
Конец


editCity Longitude()
city - редактируемый элемент
Новое значение в longitude
longitude < -180 или longitude > 180?
да
нет
city->longitude = longitude
Сообщение о невозможном значении
Конец


editCity Area()
city - редактируемый элемент
Новое значение в area
area < 1 или area > 100000?
да
нет
city->stats[0] = area
Сообщение о невозможном значении
Конец


editCity Population()
city - редактируемый элемент
Новое значение в population
population < 1 или population > 15000000?
да
нет
city->stats[1] = population
Сообщение о невозможном значении
Конец

Текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MSIZE 256

typedef struct city_regions{
    int id;
    char name[MSIZE];
    struct city_regions* next;
    struct city_regions* prev;
} region_node;

typedef struct head_city_regions{
    int count;
    struct city_regions* first;
    struct city_regions* last;
} region_head;

typedef struct city {
    int id;
    char *name;
    region_node *region;
    int code;
    float latitude;
    float longitude;
    int stats[2];
    struct city *next;
    struct city *prev;
} city_node;

typedef struct head_cities {
    int count;
    city_node *first;
    city_node *last;
} city_head;

region_head* make_region_head();
region_node* make_region_node(char name[MSIZE]);
city_head* make_city_head();
city_node* make_city_node();
```

```
void menu(region_head* cHead, city_head* rHead);
int menuInput();
void options(region_head* rHead, city_head* cHead, int option);
int startsWithIgnoreCase(const char *str, const char *prefix);
void pressEnterToContinue();
void clearStdin();
void clearConsole();

void printAllRegions(region_head* rHead);
void printAllCities(city_head* cHead);
void printRegion(region_node *region);
void printCity(city_node *city);
void trimForDisplay(char *output, const char *input, int mlen);
void nullString(char str[MSIZE]);
void trim(char str[MSIZE]);
char **split(char *str, int length, char sep);

void readRegionsFile(char* filename, region_head* rHead);
void readCitiesFile(char* filename, city_head* cHead, region_head* rHead);
void writeRegionsToFile(region_head* rHead, const char* filename);
void writeCitiesToFile(city_head* cHead, const char* filename);
void freeRegionsList(region_head* rHead);
void freeCity(city_node* city);
void freeCitiesList(city_head* cHead);

void appendRegionNode(region_head* r_head, region_node* region);
void appendCityNode(city_head* cHead, city_node* city);
void deleteRegionNode(region_head* rHead, city_head* cHead, region_node* region);
void deleteCityNode(city_head* cHead, city_node* city);
void deleteRegionGUI(region_head* rHead, city_head* cHead);
void deleteCityGUI(city_head* cHead);
void addRegionGUI(region_head* rHead);
void addCityGUI(region_head* rHead, city_head* cHead);
void fillCityNode(region_head* rHead, city_head* cHead, city_node* city, char** str);
void clearRegionListGUI(region_head* rHead, city_head* cHead);
void clearCityListGUI(city_head* cHead);
void clearCityRegionById(city_head* cHead, int id);
city_node* findCityById(city_head* cHead, int id);
void filterCitiesGUI(region_head* rHead, city_head* cHead);
void filterCitiesByName(city_head* cHead);
void filterCitiesByRegionId(city_head* cHead, region_head* rHead);
void filterCitiesByCode(city_head* cHead);
void filterCitiesByLatitude(city_head* cHead);
```

```
void filterCitiesByLongitude(city_head* cHead);
void filterCitiesByArea(city_head* cHead);
void filterCitiesByPopulation(city_head* cHead);
void sortCitiesGUI(city_head* cHead);
void sortCities(city_head* cHead, int option);
int compareCities(city_node* a, city_node* b, int option);
region_node* findRegionByName(region_head* rHead, char name[MSIZE]);
region_node* findRegionById(region_head* rHead, int id);
void updateCityDataGUI(region_head* rHead, city_head* cHead);
void editCityName(city_node* city);
void editCityRegion(region_head* rHead, city_node* city);
void editCityCode(city_node* city);
void editCityLatitude(city_node* city);
void editCityLongitude(city_node* city);
void editCityArea(city_node* city);
void editCityPopulation(city_node* city);

int main() {
    city_head* cHead = NULL;
    region_head* rHead = NULL;
    cHead = make_city_head();
    rHead = make_region_head();
    /*start if heads created successfully*/
    if (cHead != NULL && rHead != NULL) {
        menu(rHead, cHead);
    } else {
        puts("Memory allocation error\n");
    }
    return 0;
}

region_head* make_region_head() {
    region_head* rHead = NULL;
    rHead = (region_head*)malloc(sizeof(region_head));
    /*memory allocation check*/
    if (rHead != NULL) {
        rHead->count = 0;
        rHead->first = NULL;
        rHead->last = NULL;
    } else {
        perror("Memory allocation failed");
    }

    return rHead;
```

```
}

region_node* make_region_node(char name[MSIZE]) {
    region_node* region = NULL;
    region = (region_node*)malloc(sizeof(region_node));
    /*memory allocation check*/
    if (region != NULL) {
        region->id = 0;
        strcpy(region->name, name);
        region->next = NULL;
        region->prev = NULL;
    }

    return region;
}

city_head* make_city_head() {
    city_head* cHead = NULL;
    cHead = (city_head*)malloc(sizeof(city_head));
    /*memory allocation check*/
    if (cHead != NULL) {
        cHead->count = 0;
        cHead->first = NULL;
        cHead->last = NULL;
    } else {
        perror("Memory allocation failed");
    }

    return cHead;
}

city_node* make_city_node() {
    city_node* city = NULL;
    city = (city_node*)malloc(sizeof(city_node));
    /*memory allocation check*/
    if (city != NULL) {
        city->id = 0;
        city->name = NULL;
        city->region = NULL;
        city->code = 0;
        city->latitude = 0;
        city->longitude = 0;
        city->stats[0] = 0;
        city->stats[1] = 0;
```

```
        city->next = NULL;
        city->prev = NULL;
    }
    return city;
}

void menu(region_head* rHead, city_head* cHead) {
    int option, saveOption;
    /*fill lists from files*/
    readRegionsFile("regions.csv", rHead);
    readCitiesFile("cities.csv", cHead, rHead);
    /*print menu and get input until exit option is chosen*/
    do {
        clearConsole();
        option = menuInput();
        if (option != 0) {
            /*print menu for chosen option*/
            options(rHead, cHead, option);
        } else {
            /*user chooses to save changes or not*/
            saveOption = -1;
            printf("\nDo you want to save changes? (1 - yes, 0 - no): ");
            scanf("%d", &saveOption);
            clearStdin();
            if (saveOption == 1) {
                /*rewrite starting files*/
                writeCitiesToFile(cHead, "cities.csv");
                writeRegionsToFile(rHead, "regions.csv");
                puts("\nChanges successfully saved\n");
            } else if (saveOption != 0) {
                option = -1;
                puts("Wrong input - must be 0 or 1.\n");
            }
            pressEnterToContinue();
            clearConsole();
        }
    } while (option != 0);
    /*free allocated memory*/
    freeRegionsList(rHead);
    freeCitiesList(cHead);
}

int menuInput() {
    int answer;
```

```
    puts("Choose an option:");
    puts("1.  Print all regions");
    puts("2.  Print all cities");
    puts("3.  Add new region");
    puts("4.  Add new city");
    puts("5.  Update city data");
    puts("6.  Filter cities");
    puts("7.  Sort cities");
    puts("8.  Delete region");
    puts("9.  Delete city");
    puts("10. Clear regions list");
    puts("11. Clear cities list");
    puts("0.  Exit");
    printf("> ");
    /*get user answer and return it*/
    scanf("%d", &answer);
    clearStdin();
    return answer;
}

void options(region_head* rHead, city_head* cHead, int option) {
    clearConsole();
    /*call function based on user choice*/
    switch (option) {
        case 1:
            puts("1.  Print all regions\n");
            printAllRegions(rHead);
            break;
        case 2:
            puts("2.  Print all cities\n");
            printAllCities(cHead);
            break;
        case 3:
            puts("3.  Add new region\n");
            addRegionGUI(rHead);
            break;
        case 4:
            puts("4.  Add new city\n");
            addCityGUI(rHead, cHead);
            break;
        case 5:
            puts("5.  Update city data\n");
            updateCityDataGUI(rHead, cHead);
            break;
```

```
        case 6:
            puts("6.  Filter cities\n");
            filterCitiesGUI(rHead, cHead);
            break;
        case 7:
            puts("7.  Sort cities\n");
            sortCitiesGUI(cHead);
            break;
        case 8:
            puts("8.  Delete region\n");
            deleteRegionGUI(rHead, cHead);
            break;
        case 9:
            puts("9.  Delete city\n");
            deleteCityGUI(cHead);
            break;
        case 10:
            puts("10. Clear region list\n");
            clearRegionListGUI(rHead, cHead);
            break;
        case 11:
            puts("11. Clear city list\n");
            clearCityListGUI(cHead);
            break;
        default:
            clearConsole();
            puts("Invalid option.");
            break;
    }
    pressEnterToContinue();
}

void pressEnterToContinue() {
    /*convenient transition between menus*/
    printf("\nPress ENTER to continue ");
    clearStdin();
    clearConsole();
}

int startsWithIgnoreCase(const char *str, const char *prefix) {
    /*initialize flag for prefix check*/
    int isPrefix = 1;
    /*loop through characters of both the string and the prefix until either the end of one is
reached or mismatch found*/
```

```
    while (*str && *prefix && isPrefix) {
        if (tolower(*str) != tolower(*prefix)) {
            /* If characters don't match, set flag to 0*/
            isPrefix = 0;
        }
        /*move to next character*/
        str++;
        prefix++;
    }
    /*check if end of prefix reached*/
    if (*prefix != '\0') {
        isPrefix = 0;
    }
    /*return 1 if string starts with prefix, 0 if not*/
    return isPrefix;
}

void clearStdin() {
    int c;
    /*clear input buffer*/
    while ((c = getchar()) != '\n' && c != EOF) { }
}

void clearConsole() {
    /*clear screen depending on operation system*/
    #if defined(_WIN32) || defined(_WIN64)
        system("cls");
    #else
        system("clear");
    #endif
}

void printAllRegions(region_head* r_head) {
    region_node *current;
    /*print header*/
    printf("|%3s|%15s|\n", "Id", "Region name");
    puts("+---+---------------+");

    current = r_head->first;
    /*print regions until end of list reached*/
    while (current != NULL) {
        printRegion(current);
        current = current->next;
    }
```

```
    puts("+---+---------------+\n");
}

void printAllCities(city_head* cHead) {
    city_node *current;
    /*print header*/
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");

    current = cHead->first;
    /*print regions until end of list reached*/
    while (current != NULL) {
        printCity(current);
        current = current->next;
    }
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
}

void printRegion(region_node *region) {
    char trimmedName[15];
    /*shorten name to fit in the table*/
    trimForDisplay(trimmedName, region->name, 15);
    printf("| %-1d | %-13s |\n", region->id, trimmedName);
}

void printCity(city_node *city) {
    char region[MSIZE] = "undefined";
    char trimmedName[23], trimmedRegion[17];
    /*shorten names to fit in the table*/
    if (city->region != NULL) {
        trimForDisplay(region, city->region->name, sizeof(region));
    }
    trimForDisplay(trimmedName, city->name, 22);
    trimForDisplay(trimmedRegion, region, 16);

    printf("| %-3d | %-18s | %-13s | %-10d | %-8.2f | %-8.2f | %-8d | %-13d |\n",
           city->id, trimmedName, trimmedRegion, city->code, city->latitude, city->longitude, city-
>stats[0], city->stats[1]);
}

void trimForDisplay(char *output, const char *input, int mlen) {
    /* compare input string length to length allowed for display*/
    if (strlen(input) > mlen) {
```

```
        /*trim string and put ... in the end*/
        strncpy(output, input, mlen - 3);
        output[mlen - 3] = '\0';
        strcat(output, "...");
    } else {
        /*string remains the same*/
        strcpy(output, input);
    }
}

void nullString(char str[MSIZE]) {
    int i;
    /*nullify every string character*/
    for (i = 0; i < MSIZE; i++) {
        str[i] = '\0';
    }
}

void trim(char str[MSIZE]) {
    int i, flag = 0;
    str[MSIZE - 1] = '\0';
    /*check if end of string reached*/
    for (i = 0; str[i] != '\0' && !flag; i++) {
        /*if end of string found, set flag to leave loop*/
        if (str[i] == '\n' || str[i] == '\r') {
            str[i] = '\0';
            flag = 1;
        }
    }
}

char **split(char *str, int length, char sep) {
    int count = 0;
    int i = 0;
    int start = 0;
    int j = 0;
    int wordLen = 0;
    char **result = NULL;
    char *newStr = NULL;
    int allocError = 0;
    /*count number of words*/
    for (i = 0; i < length; i++) {
        if (str[i] == sep) count++;
    }
```

```
    count++;
    /*allocate memory and check for success*/
    result = malloc(count * sizeof(char *));
    if (result == NULL) {
        perror("Memory allocation failed");
    } else {
        /*loop through string*/
        for (i = 0; i < length; i++) {
            /*check for separator or end of string*/
            if (str[i] == ';' || str[i] == '\0') {
                /*calculate word length, allocate memory and check for success*/
                wordLen = i - start;
                newStr = malloc((wordLen + 1) * sizeof(char));
                if (newStr == NULL) {
                    perror("Memory allocation failed");
                    allocError = 1;
                    i = length;
                } else {
                    /*save string in word list*/
                    strncpy(newStr, str + start, wordLen);
                    newStr[wordLen] = '\0';
                    result[j++] = newStr;
                    /*update index for next word*/
                    start = i + 1;
                }
            }
        }
        /*free strings if memory not allocated*/
        if (allocError) {
            for (i = 0; i < j; i++) {
                free(result[i]);
            }
            free(result);
            result = NULL;
        }
    }

    return result;
}

void readRegionsFile(char* filename, region_head* rHead) {
    FILE* file;
    region_node* region;
    int n, count, i;
```

```
    char temp[MSIZE];
    region = NULL;
    n = count = 0;
    file = fopen(filename, "r");
    /*check if file opened successfully*/
    if (file != NULL) {
        /*count lines in file*/
        while ((fgets(temp, MSIZE, file)) != NULL) n++;
        rewind(file);
        /*read lines*/
        for (i = 0; i < n; i++) {
            /*clear temp string*/
            nullString(temp);
            /*read string in temp and trim it*/
            fgets(temp, MSIZE, file);
            trim(temp);
            /*create new node and check for success*/
            region = make_region_node(temp);
            if (region != NULL) {
                /*add new node to list and increase count*/
                appendRegionNode(rHead, region);
                count++;
            }
        }
        fclose(file);
    } else {
        perror("Failed to open file");
    }
    /*if read less lines than in file, free region list*/
    if (count != n) {
        perror("Failed to read from file");
        freeRegionsList(rHead);
    }
}

void readCitiesFile(char* filename, city_head* cHead, region_head* rHead) {
    FILE* file;
    city_node* city;
    int n, count, i, slen;
    char** splitArray;
    char temp[MSIZE];

    city = NULL;
    n = count = 0;
```

```
    file = fopen(filename, "r");
    /*check if file opened successfully*/
    if (file != NULL) {
        /*count lines in file*/
        while ((fgets(temp, MSIZE, file)) != NULL) n++;
        rewind(file);
        /*read lines*/
        for (i = 0; i < n; i++, count++) {
            /*clear temp string*/
            nullString(temp);
            /*read string in temp and trim it*/
            fgets(temp, MSIZE, file);
            slen = strlen(temp);
            trim(temp);
            /*split string and check for success*/
            splitArray = split(temp, slen, ';');
            if (splitArray != NULL) {
                /*create new node and check for success*/
                city = make_city_node();
                if (city != NULL) {
                    /*fill new node and add to list*/
                    fillCityNode(rHead, cHead, city, splitArray);
                    appendCityNode(cHead, city);
                }
            }
        }
        fclose(file);
    } else {
        perror("Failed to open file");
    }
    /*if read less lines than in file, free city list*/
    if (count != n) {
        perror("Failed to read from file");
        freeCitiesList(cHead);
    }
}

void writeRegionsToFile(region_head* rHead, const char* filename) {
    FILE* file = fopen(filename, "w");
    region_node* current = NULL;
    /*check if file opened successfully*/
    if (file != NULL) {
        current = rHead->first;
        /*rewrite file*/
```

```
        while (current != NULL) {
            fprintf(file, "%s\n", current->name);
            current = current->next;
        }

        fclose(file);
    } else {
        perror("Failed to open file");
    }
}

void writeCitiesToFile(city_head* cHead, const char* filename) {
    FILE* file = fopen(filename, "w");
    city_node* current = NULL;
    char* regionName;
    /*check if file opened successfully*/
    if (file != NULL) {
        current = cHead->first;
        /*rewrite file*/
        while (current != NULL) {
            regionName = "undefined";
            if (current->region != NULL) {
                regionName = current->region->name;
            }
            fprintf(file, "%s;%s;%d;%.2f;%.2f;%d;%d\n", current->name, regionName, current->code,
                    current->latitude, current->longitude, current->stats[0], current->stats[1]);
            current = current->next;
        }

        fclose(file);
    } else {
        printf("Failed to open file %s\n", filename);
    }
}

void freeRegionsList(region_head* rHead) {
    region_node *current, *current1;
    current = rHead->first;
    /*free list*/
    while (current != NULL) {
        current1 = current->next;
        free(current);
        current = current1;
    }
```

```
    free(rHead);
}

void freeCity(city_node* city) {
    /*free node fields*/
    if (city->name != NULL) {
        free(city->name);
        city->name = NULL;
    }
    if (city->region != NULL) {
        city->region = NULL;
    }
    free(city);
}

void freeCitiesList(city_head* cHead) {
    city_node *current = NULL, *current1 = NULL;
    current = cHead->first;
    /*free list*/
    while (current != NULL) {
        current1 = current->next;
        freeCity(current);
        current = current1;
    }
    free(cHead);
}

void appendRegionNode(region_head* r_head, region_node* region) {
    r_head->count++;
    /*check if list is empty*/
    if (r_head->first == NULL) {
        /*new node is first*/
        r_head->first = region;
        r_head->last = region;
        region->id = 1;

    } else  {
        /*new node is last*/
        region->id = r_head->last->id + 1;
        region->prev = r_head->last;
        r_head->last->next = region;
        r_head->last = region;
    }
}
```

```
void appendCityNode(city_head* cHead, city_node* city) {
    cHead->count++;
    /*check if list is empty*/
    if (cHead->first == NULL) {
        /*new node is first*/
        cHead->first = city;
        cHead->last = city;
        city->id = 1;

    } else  {
        /*new node is lasr*/
        city->id = cHead->last->id + 1;
        city->prev = cHead->last;
        cHead->last->next = city;
        cHead->last = city;
    }
}

void deleteRegionNode(region_head* rHead, city_head* cHead, region_node* region) {
    /*check if node is first or last*/
    if (rHead->first == region) {
        /*second node is now first*/
        rHead->first = region->next;
        if (region->next != NULL) {
            region->next->prev = region->prev;
        }
    } else if (rHead->last == region) {
        /*prelast node is now last*/
        rHead->last = region->prev;
        if (region->prev != NULL) {
            region->prev->next = region->next;
        }
    } else {
        /*changing pointers of neighboring nodes*/
        if (region->prev != NULL) {
            region->prev->next = region->next;
        }
        if (region->next != NULL) {
            region->next->prev = region->prev;
        }
    }
    /*delete region in city list*/
    clearCityRegionById(cHead, region->id);
```

```
    free(region);
    rHead->count--;
}

void deleteCityNode(city_head* cHead, city_node* city) {
    /*check if node is first or last*/
    if (cHead->first == city) {
        /*second node is now first*/
        cHead->first = city->next;
        if (city->next != NULL) {
            city->next->prev = city->prev;
        }
    } else if (cHead->last == city) {
        /*prelast node is now last*/
        cHead->last = city->prev;
        if (city->prev != NULL) {
            city->prev->next = city->next;
        }
    } else {
        /*changing pointers of neighboring nodes*/
        if (city->prev != NULL) {
            city->prev->next = city->next;
        }
        if (city->next != NULL) {
            city->next->prev = city->prev;
        }
    }
    freeCity(city);
    cHead->count--;
}

void deleteRegionGUI(region_head* rHead, city_head* cHead) {
    int id;
    region_node* region = NULL;
    /*check if list is empty*/
    if (rHead->first != NULL) {
        printAllRegions(rHead);
        printf("\nEnter region id to delete (or 0 to return to menu): ");
        scanf("%d", &id);
        clearStdin();
        /*check if id is in the list*/
        if (id > 0 && id <= rHead->count) {
            region = findRegionById(rHead, id);
            printf("\nRegion with id %d:\n", id);
```

```
            printf("|%3s|%15s|\n", "Id", "Region name");
            puts("+---+---------------+");
            printRegion(region);
            puts("+---+---------------+");
            deleteRegionNode(rHead, cHead, region);
            printf("\nRegion with id %d has been successfully removed.\n", id);
        } else if (id != 0) {
            puts("\nWrong ID input.");
        }
    } else {
        puts("List is empty.");
    }
}

void deleteCityGUI(city_head* cHead) {
    int id;
    city_node* city = NULL;
    /*check if list is empty*/
    if (cHead->first != NULL) {
        printAllCities(cHead);
        printf("\nEnter city id to delete city (or 0 to return to menu): ");
        scanf("%d", &id);
        clearStdin();
        /*check if id is in the list*/
        if (id > 0 && id <= cHead->count) {
            city = findCityById(cHead, id);
            printf("\nCity with id %d:\n", id);
            printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
            puts("+-----+--------------------+---------------+------------+----------+----------+----------
+---------------+");
            printCity(city);
            puts("+-----+--------------------+---------------+------------+----------+----------+----------
+---------------+");
            deleteCityNode(cHead, city);
            printf("\nCity with id %d has been successfully removed.\n", id);
        } else if (id != 0) {
            puts("\nWrong ID input.");
        }
    } else {
        puts("List is empty.");
    }
}
```

```
void addRegionGUI(region_head* rHead) {
    char temp[MSIZE];
    region_node* region = NULL;

    printf("Enter region name: ");
    /*check if name was entered*/
    if (fgets(temp, MSIZE, stdin) != NULL) {
        trim(temp);
        /*create new node and check for success*/
        region = make_region_node(temp);
        if (region != NULL) {
            /*add node to end of list and print it*/
            appendRegionNode(rHead, region);
            puts("\nRegion successfully added.\n");
            printf("|%3s|%15s|\n", "Id", "Region name");
            puts("+---+---------------+");
            printRegion(region);
            puts("+---+---------------+");
        } else {
            puts("\nFailed: memory error");
        }
    } else {
        puts("\nFailed: memory error");
    }
}

void addCityGUI(region_head* rHead, city_head* cHead) {
    city_node* city = NULL;
    /*create new node and check for success*/
    city = make_city_node();
    if (city != NULL) {
        puts("Enter information for new city:\n");
        /*entering all fields*/
        editCityName(city);
        editCityRegion(rHead, city);
        editCityCode(city);
        editCityLatitude(city);
        editCityLongitude(city);
        editCityArea(city);
        editCityPopulation(city);
        /*add node to end of list*/
        appendCityNode(cHead, city);
        puts("\nCity has been successfully added.");
```

```
        printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
        puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------
+");
        printCity(city);
        puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------
+");
    } else {
        perror("\nFailed: memory error\n");
    }
}

void fillCityNode(region_head* rHead, city_head* cHead, city_node* city, char** str) {
    if (city != NULL) {
        /*filling node with data in strings list*/
        city->name = str[0];
        city->region = findRegionByName(rHead, str[1]); /*fill with existing region name or
undefined*/
        free(str[1]);
        city->code = atoi(str[2]) ; /*convert to int type from string*/
        free(str[2]);
        city->latitude = atof(str[3]); /*convert to float type from string*/
        free(str[3]);
        city->longitude = atof(str[4]);
        free(str[4]);
        city->stats[0] = atoi(str[5]);
        free(str[5]);
        city->stats[1] = atoi(str[6]);
        free(str[6]);
        free(str);

        city->next = NULL;
        city->prev = NULL;
    } else {
        perror("Memory allocation failed");
    }
}

void clearRegionListGUI(region_head* rHead, city_head* cHead) {
    region_node *current, *current1;
    city_node* city;
    /*check if list is empty*/
    current = rHead->first;
    if (current == NULL) {
```

```
        puts("List is empty.");
    } else {
        /*free list*/
        while (current != NULL) {
            current1 = current->next;
            free(current);
            current = current1;
        }
        /*delete all region pointers in city nodes*/
        city = cHead->first;
        while (city != NULL) {
            city->region = NULL;
            city = city->next;
        }
        /*clear head*/
        rHead->first = NULL;
        rHead->last = NULL;
        rHead->count = 0;
        puts("List successfully cleared.");
    }
}

void clearCityListGUI(city_head* cHead) {
    city_node* current, *current1;
    /*check if list is empty*/
    if (cHead->first != NULL) {
        /*free list*/
        current = cHead->first;
        while (current != NULL) {
            current1 = current->next;
            freeCity(current);
            current = current1;
        }
        /*clear head*/
        cHead->last = NULL;
        cHead->first = NULL;
        cHead->count = 0;
        puts("List successfully cleared.");
    } else {
        puts("List is empty.");
    }
}

void clearCityRegionById(city_head* cHead, int id) {
```

```
    city_node* current = NULL;

    current = cHead->first;
    while (current != NULL) {
        /*clear region pointer in city node if region id matches*/
        if (current->region != NULL && current->region->id == id) {
            current->region = NULL;
        }
        current = current->next;
    }
}

city_node* findCityById(city_head* cHead, int id) {
    city_node* current = NULL;
    current = cHead->first;
    /*loop until end of list reached or id matches*/
    while (current != NULL && current->id != id) {
        current = current->next;
    }
    return current;
}

void filterCitiesGUI(region_head* rHead, city_head* cHead) {
    int option;

    puts("1. Name");
    puts("2. Region");
    puts("3. Code");
    puts("4. Latitude");
    puts("5. Longitude");
    puts("6. Area");
    puts("7. Population");
    printf("\nEnter option: ");
    scanf("%d", &option);
    clearStdin();
    /*call function depending on user choice*/
    switch (option) {
        case 1:
            filterCitiesByName(cHead);
            break;
        case 2:
            filterCitiesByRegionId(cHead, rHead);
            break;
        case 3:
```

```
            filterCitiesByCode(cHead);
            break;
        case 4:
            filterCitiesByLatitude(cHead);
            break;
        case 5:
            filterCitiesByLongitude(cHead);
            break;
        case 6:
            filterCitiesByArea(cHead);
            break;
        case 7:
            filterCitiesByPopulation(cHead);
            break;
        default:
            puts("Wrong option.");
            break;
    }
}


void filterCitiesByName(city_head* cHead) {
    city_node *current;
    char name[MSIZE];
    /*name input and trim*/
    printf("Enter name: ");
    fgets(name, MSIZE, stdin);
    trim(name);
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
    current = cHead->first;
    /*print cities with names starting with entered string*/
    while (current != NULL) {
        if (startsWithIgnoreCase(current->name, name) == 1) {
            printCity(current);
        }
        current = current->next;
    }
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
}

void filterCitiesByRegionId(city_head* cHead, region_head* rHead) {
    city_node *current;
```

```
    int id;
    /*print region list and get region input*/
    printAllRegions(rHead);
    printf("Enter region id (0 to show cities with undefined region): ");
    scanf("%d", &id);
    clearStdin();
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
    current = cHead->first;
    /*print all cities with matching region id or all cities with undefined region if id is 0*/
    while (current != NULL) {
        if ((current->region != NULL && current->region->id == id) || (current->region == NULL &&
id == 0)) {
            printCity(current);
        }
        current = current->next;
    }
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
}

void filterCitiesByCode(city_head* cHead) {
    city_node *current;
    int minC, maxC;
    /*input of min and max values*/
    printf("Enter min code: ");
    scanf("%d", &minC);
    clearStdin();
    printf("Enter max code: ");
    scanf("%d", &maxC);
    clearStdin();
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
    current = cHead->first;
    /*print all cities with code in interval*/
    while (current != NULL) {
        if (current->code >= minC && current->code <= maxC) {
            printCity(current);
        }
        current = current->next;
    }
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------+");
}
```

```
void filterCitiesByLatitude(city_head* cHead) {
    city_node *current;
    float minL, maxL;
    /*input of min and max values*/
    printf("Enter min latitude: ");
    scanf("%f", &minL);
    clearStdin();
    printf("Enter max latitude: ");
    scanf("%f", &maxL);
    clearStdin();
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+-------------------+--------------+-----------+----------+----------+----------+--------------+");
    current = cHead->first;
    /*print all cities with latitude in interval*/
    while (current != NULL) {
        if (current->latitude >= minL && current->latitude <= maxL) {
            printCity(current);
        }
        current = current->next;
    }
    puts("+-----+-------------------+--------------+-----------+----------+----------+----------+--------------+");
}

void filterCitiesByLongitude(city_head* cHead) {
    city_node *current;
    float minL, maxL;
    /*input of min and max values*/
    printf("Enter min longitude: ");
    scanf("%f", &minL);
    clearStdin();
    printf("Enter max longitude: ");
    scanf("%f", &maxL);
    clearStdin();
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+-------------------+--------------+-----------+----------+----------+----------+--------------+");
    current = cHead->first;
    /*print all cities with longitude in interval*/
    while (current != NULL) {
        if (current->longitude >= minL && current->longitude <= maxL) {
            printCity(current);
        }
```

```
        current = current->next;
    }
    puts("+-----+------------------+--------------+-----------+----------+----------+----------+--------------+");
}

void filterCitiesByArea(city_head* cHead) {
    city_node *current;
    int minA, maxA;
    /*input of min and max values*/
    printf("Enter min area: ");
    scanf("%d", &minA);
    clearStdin();
    printf("Enter max area: ");
    scanf("%d", &maxA);
    clearStdin();
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+------------------+--------------+-----------+----------+----------+----------+--------------+");
    current = cHead->first;
    /*print all cities with area in interval*/
    while (current != NULL) {
        if (current->stats[0] >= minA && current->stats[0] <= maxA) {
            printCity(current);
        }
        current = current->next;
    }
    puts("+-----+------------------+--------------+-----------+----------+----------+----------+--------------+");
}

void filterCitiesByPopulation(city_head* cHead) {
    city_node *current;
    int minP, maxP;
    /*input of min and max values*/
    printf("Enter min population: ");
    scanf("%d", &minP);
    clearStdin();
    printf("Enter max population: ");
    scanf("%d", &maxP);
    clearStdin();
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+------------------+--------------+-----------+----------+----------+----------+--------------+");
    current = cHead->first;
    /*print all cities with population in interval*/
```

```
    while (current != NULL) {
        if (current->stats[1] >= minP && current->stats[1] <= maxP) {
            printCity(current);
        }
        current = current->next;
    }
    puts("+-----+------------------+--------------+-----------+----------+----------+----------+--------------+");
}

void sortCitiesGUI(city_head* cHead) {
    int option;
    /*check if list is empty*/
    if (cHead->first != NULL) {
        /*print menu and get user input*/
        puts("1. Sort by id");
        puts("2. Sort by name");
        puts("3. Sort by region");
        puts("4. Sort by latitude");
        puts("5. Sort by longitude");
        puts("6. Sort by area");
        puts("7. Sort by population");
        printf("Enter option: ");
        scanf("%d", &option);
        clearStdin();
        /*sort if option was in the menu*/
        if (option > 0 && option <= 7) {
            sortCities(cHead, option);
            puts("List successfully sorted");
        } else {
            puts("Wrong option");
        }
    } else {
        puts("List is empty");
    }
}

void sortCities(city_head* cHead, int option) {
    city_node* sorted = NULL;
    city_node* current = cHead->first;
    city_node* next = NULL;
    city_node* temp = NULL;
    /*check if list includes more than two elements*/
    if (cHead->first != NULL && cHead->first->next != NULL) {
        while (current != NULL) {
```

```
            next = current->next;
            if (sorted == NULL || compareCities(current, sorted, option) < 0) {
                /*Insert element in the beginning*/
                current->next = sorted;
                if (sorted != NULL) sorted->prev = current;
                sorted = current;
                sorted->prev = NULL;
            } else {
                /* Find the insertion point within the sorted list*/
                temp = sorted;
                while (temp->next != NULL && compareCities(current, temp->next, option) > 0) {
                    temp = temp->next;
                }
                /*insert element*/
                current->next = temp->next;
                if (temp->next != NULL) temp->next->prev = current;
                temp->next = current;
                current->prev = temp;
            }
            current = next;
        }
        /*update list head*/
        cHead->first = sorted;
        temp = sorted;
        while (temp != NULL && temp->next != NULL) {
            temp = temp->next;
        }
        cHead->last = temp;
    }
}

int compareCities(city_node* a, city_node* b, int option) {
    int result;
    /*result calculated based on chosen field*/
    switch (option) {
        case 1:
            result =  a->id - b->id;
            break;
        case 2:
            result = strcmp(a->name, b->name);
            break;
        case 3:
            result = a->code - b->code;
            break;
```

```
        case 4:
            result = (a->latitude > b->latitude) ? 1 : (a->latitude < b->latitude) ? -1 : 0;
            break;
        case 5:
            result = (a->longitude > b->longitude) ? 1 : (a->longitude < b->longitude) ? -1 : 0;
            break;
        case 6:
            result = a->stats[0] - b->stats[0];
            break;
        case 7:
            result = a->stats[1] - b->stats[1];
            break;
        default:
            result = 0;
            break;
    }
    /*returns 0 if a equal to b, positive if a is greater than b, negative if lower*/
    return result;
}

region_node* findRegionByName(region_head* rHead, char name[MSIZE]) {
    region_node* current = NULL;
    current = rHead->first;
    /*search and return region with same name*/
    while (current != NULL && strcmp(current->name, name) != 0) {
        current = current->next;
    }
    return current;
}

region_node* findRegionById(region_head* rHead, int id) {
    region_node* current = NULL;
    current = rHead->first;
    /*search and return region with same id*/
    while (current != NULL && current->id != id) {
        current = current->next;
    }
    return current;
}

void updateCityDataGUI(region_head* rHead, city_head* cHead) {
    city_node* city;
    int cityId, option;
    /*user choose city to update*/
```

```
    printAllCities(cHead);
    printf("Enter city id: ");
    scanf("%d", &cityId);
    clearStdin();
    city = findCityById(cHead, cityId);
    /*check if city with such id found*/
    if (city != NULL) {
        clearConsole();
        /*print city to update*/
        puts("Update city data\n");
        printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
        puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------
+");
        printCity(city);
        puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------
+");
        /*print menu and get input*/
        puts("\nWhich field would you like to edit?\n");
        puts("1. Name");
        puts("2. Region");
        puts("3. Code");
        puts("4. Latitude");
        puts("5. Longitude");
        puts("6. Area");
        puts("7. Population");
        puts("8. All fields");
        printf("\nEnter option: ");
        scanf("%d", &option);
        clearStdin();
        clearConsole();
        /*updates field based on users choice*/
        switch (option) {
            case 1:
                puts("1. Name\n");
                editCityName(city);
                break;
            case 2:
                puts("2. Region\n");
                editCityRegion(rHead, city);
                break;
            case 3:
                puts("3. Code\n");
                editCityCode(city);
```

```
            break;
        case 4:
            puts("4. Latitude\n");
            editCityLatitude(city);
            break;
        case 5:
            puts("5. Longitude\n");
            editCityLongitude(city);
            break;
        case 6:
            puts("6. Area\n");
            editCityArea(city);
            break;
        case 7:
            puts("7. Population\n");
            editCityPopulation(city);
            break;
        case 8:
            puts("8. All fields\n");
            editCityName(city);
            editCityRegion(rHead, city);
            editCityCode(city);
            editCityLatitude(city);
            editCityLongitude(city);
            editCityArea(city);
            editCityPopulation(city);
            break;
        default:
            puts("\nWrong option.");
            break;
    }
    /*print updated city*/
    puts("\nUpdated city:\n");
    printf("|%5s|%20s|%15s|%12s|%10s|%10s|%10s|%15s|\n", "ID", "City name", "Region",
"Region code", "Latitude", "Longitude", "Area", "Population");
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------
+");
    printCity(city);
    puts("+-----+--------------------+---------------+------------+----------+----------+----------+---------------
+");
  } else {
    puts("\nCity not found\n");
  }
}
```

```
void editCityName(city_node* city) {
    char temp[MSIZE];

    printf("Enter city name: ");
    /*get input of new name and check for success*/
    if (fgets(temp, MSIZE, stdin) != NULL) {
        trim(temp);
        /*delete old name*/
        if (city->name != NULL) {
            free(city->name);
            city->name = NULL;
        }
        /*memory allocation and name update*/
        city->name = (char*)malloc(strlen(temp) + 1);
        if (city->name != NULL) {
            strcpy(city->name, temp);
        } else {
            puts("Memory error\n");
        }
    } else {
        puts("Memory error\n");
    }
}

void editCityRegion(region_head* rHead, city_node* city) {
    region_node* region;
    int success;
    int regionId;
    /*check if list is empty*/
    if (rHead->first == NULL) {
        puts("Region list is empty.");
    } else {
        /*get region id from region list*/
        printAllRegions(rHead);
        printf("Enter region id: ");
        success = scanf("%d", &regionId);
        clearStdin();
        if (success != 1) {
            regionId = 0;
        }
        region = findRegionById(rHead, regionId);
```

```
        /*check if region found and update the field*/
        if (region != NULL) {
            city->region = region;
        } else {
            puts("Region not found");
        }
    }
}

void editCityCode(city_node* city) {
    int code;
    int success;
    /*get new value*/
    printf("Enter city code: ");
    success = scanf("%d", &code);
    clearStdin();
    /*validate input and update the field*/
    if (code < 0 || code > 95 || success != 1) {
        puts("Invalid or impossible code");
    } else {
        city->code = code;
    }
}

void editCityLatitude(city_node* city) {
    float latitude;
    int success;
    /*get new value*/
    printf("Enter city latitude: ");
    success = scanf("%f", &latitude);
    clearStdin();
    if (latitude < -90 || latitude > 90 || success != 1) {
        puts("Invalid or impossible latitude");
    } else {
        city->latitude = latitude;
    }
}

void editCityLongitude(city_node* city) {
    float longitude;
    int success;
    /*get new value*/
    printf("Enter city longitude: ");
    success = scanf("%f", &longitude);
```

```
    clearStdin();
    /*validate input and update the field*/
    if (longitude < -180 || longitude > 180 || success != 1) {
        puts("Invalid or impossible longitude");
    } else {
        city->longitude = longitude;
    }
}

void editCityArea(city_node* city) {
    int area;
    int success;
    /*get new value*/
    printf("Enter city area: ");
    success = scanf("%d", &area);
    clearStdin();
    /*validate input and update the field*/
    if (area < 1 || area > 100000 || success != 1) {
        puts("Invalid or impossible area");
    } else {
        city->stats[0] = area;
    }
}

void editCityPopulation(city_node* city) {
    int population;
    int success;
    /*get new value*/
    printf("Enter city population: ");
    success = scanf("%d", &population);
    clearStdin();
    /*validate input and update the field*/
    if (population < 1 || population > 15000000 || success != 1) {
        puts("Invalid or impossible population");
    } else {
        city->stats[1] = population;
    }
```