МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПрАТ «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра Інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Зав.кафедрою _____
д.е.н.,доцент Левицький С.І.

МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА

РОЗРОБКА СИСТЕМИ РОЗРАХУНКУ КЛІМАТИЧНИХ ПАРАМЕТРІВ ЗА БУДІВЕЛЬНИМИ НОРМАМИ

Виконав
ст. гр. КІ-211м _____ О.С. Соловьйова
(підпис)

Керівник
к.т.н. _____ О.А. Хараджян
(підпис)

Запоріжжя
2023

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ

ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра Інформаційних технологій

ЗАТВЕРДЖУЮ
Зав. кафедрою_____
д.е.н., доцент Левицький С.І.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ

Студенту гр. КІ – 211м, спеціальності «Комп'ютерна інженерія»

Соловьйова Ольга Сергіївна

1.Тема: *Розробка системи розрахунку кліматичних параметрів за будівельними нормами.*

затверджена наказом по інституту «___»_____ 2022 р. № ____

2. Термін здачі студентом закінченої роботи: «___»_____ 2023 р.

3. Перелік питань, що підлягають розробці:

1. Аналіз кліматичних параметрів, яки використовують при проектуванні будівель.
2. Аналіз стандартних кліматичних параметрів.
3. Розробка алгоритму апроксимації кліматичних параметрів.
4. Апроксимація даних таблиць ДСТУ.
5. Розробка структура таблиць бази даних.

6. Розробка інтерфейсу програми та основних структур даних.
7. Розробка програмних модулів користувальницького інтерфейсу.
8. Розробка модулів взаємодії з базою даних.

Дата видачі завдання: 03.09.2022 р.

Студент		О.С. Соловьйова
	(підпис)	(прізвище та ініціали)
Керівник роботи		О.А. Хараджян
	(підпис)	(прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить 108 стор., 11 рис., 10 таблиць, 2 додаток, 11 використаних джерел.

Об’єкт роботи: кліматичні параметрів.

Предмет роботи: апроксимація кліматичних параметрів за ДСТУ.

Мета роботи: розробка рограми для апроксимації кліматичних параметрів з використанням таблиць даних ДСТУ для довільного населеного пункту.

Задачі роботи: аналіз кліматичних параметрів, яки використовують при проектуванні будівель; аналіз стандартних кліматичних параметрів; розробка алгоритму апроксимації кліматичних параметрів; апроксимація даних таблиць ДСТУ; розробка структура таблиць бази даних; розробка інтерфейсу програми та основних структур даних; розробка програмних модулів користувальницького інтерфейсу; розробка модулів взаємодії з базою даних.

В Україні типові розрахункові кліматичні параметри стандартизовані у ДСТУ-Н Б В.1.1-27 2010 «Кліматологія». У стандарті наведено дані про кліматичні параметри в обмеженій кількості населених пунктів. Для районів проектування та будівництва, що не увійшли до стандарту, кліматичні параметри приймають рівними значенням кліматичних параметрів найближчого до них пункту, наведеного в таблиці, або апроксимують між найближчими точками.

АПРОКСИМАЦІЯ, КЛІМАТИЧНІ ПАРАМЕТРИ, СФЕРИЧНІ КООРДИНАТИ, QT, QXLSX

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Слово / словосполучення	Скорочення	Умови використання
Б		
База даних	БД	
Д		
Державний стандарт України	ДСТУ	
A		
Application Programming Interface	API	
S		
Structured query language	SQL	

ВСТУП

Будівлі постійно піддаються впливу кількох кліматичних та екологічних факторів. Від вітру, сонячного світла, температури, дощу та інших факторів будівлі по всьому світу унікальним чином взаємодіють з різними елементами навколишнього клімату. Через це проектування будівель і методи будівництва відрізняються від одного місця до іншого, щоб відповідати різним викликам.

Будівлі можуть бути вразливими до зміни клімату. У майбутньому може виникнути збільшення ризику обвалу, погіршення здоров'я та значної втрати вартості в результаті нових штормів, снігу або пошкоджень, пов'язаних із просіданням, потрапляння води, погіршення клімату в приміщенні та скорочення терміну служби будівлі. У короткостроковій перспективі сильніші шторми є найбільшою проблемою. Шторми становлять загрозу безпеці в тих частинах існуючих будівель, які не відповідають вимогам безпеки будівельних норм. У довгостроковій перспективі більш тривалі хвилі спеки можуть мати наслідки для здоров'я, особливо для людей похилого віку та слабких людей, наприклад, у будинках престарілих.

В Україні типові розрахункові кліматичні параметри стандартизовані у ДСТУ-Н Б В.1.1-27 2010 «Кліматологія». Цей стандарт установлює кліматичні параметри, що використовують при проектуванні будинків та споруд, систем опалення, вентиляції, кондиціонування, водозабезпечення, складанні енергетичного паспорта будинку. У стандарті наведено дані про кліматичні параметри в обмеженій кількості населених пунктів. Для районів проектування та будівництва, що не увійшли до стандарту, кліматичні параметри приймають рівними значенням кліматичних параметрів найближчого до них пункту, наведеного в таблиці, або апроксимують між найближчими точками.

Об'єкт роботи: кліматичні параметрів.

Предмет роботи: апроксимація кліматичних параметрів за ДСТУ.

Мета роботи: розробка програми для апроксимації кліматичних параметрів з використанням таблиць даних ДСТУ для довільного населеного пункту.

Задачі роботи:

- аналіз кліматичних параметрів, яки використовують при проектуванні будівель
- аналіз стандартних кліматичних параметрів
- розробка алгоритму апроксимації кліматичних параметрів
- апроксимація даних таблиць ДСТУ
- розробка структура таблиць бази даних
- розробка інтерфейсу програми та основних структур даних
- розробка програмних модулів користувальницького інтерфейсу
- розробка модулів взаємодії з базою даних

РОЗДІЛ 1
МОДЕЛЮВАННЯ КЛІМАТИЧНИХ ПАРАМЕТРІВ

1.1. Кліматичні параметри у будівництві

Будівлі постійно піддаються впливу кількох кліматичних та екологічних факторів. Від вітру, сонячного світла, температури, дощу та інших факторів будівлі по всьому світу унікальним чином взаємодіють з різними елементами навколишнього клімату. Через це проектування будівель і методи будівництва відрізняються від одного місця до іншого, щоб відповідати різним викликам. Отже, які найважливіші кліматичні фактори і що робиться, щоб підготуватися до їх дії або пом’якшити їх?

Вітер є важливим фактором у будівництві через його руйнівну здатність. Для того, щоб будівля могла витримати максимальну силу вітру, у процесі проектування необхідно враховувати як середню швидкість вітру, так і переважаючий напрямок. Будівельні елементи, такі як дахи та покриття, найбільш вразливі до вітру. Житлові будинки, зокрема, розроблені для опору силам вітру за допомогою конькових і вальмових конструкцій. Знизу розташовані міцні та довговічні сухі стегнові системи, які забезпечують стійкість до сил підйому.

Опади також є важливим критерієм для будівельних проектів. Накопичення опадів може завдати шкоди цілісності будівлі, тому більше будівель у вологих регіонах світу будують із двосхилим або вальмовим дахом на відміну від систем плоского даху. Захист внутрішніх конструкцій і матеріалів від опадів також важливий, знову ж таки покладаючись на міцний бар’єр даху та підстилаючу мембрану.

Температура має значний вплив на матеріали, які використовуються в будівельному проекті. Температура повітря визначає матеріали, які використовуються для надземної конструкції, тоді як температура ґрунту

відіграє велику роль у виборі фундаменту. Середня температура повітря також визначає тип та товщину ізоляції в будівлі, причому для холодніших кліматичних умов потрібна додаткова ізоляція для збереження тепла.

Сонячне світло є важливим фактором через роль сонячної енергії. У теплих регіонах світу, а саме в районах, розташованих ближче до екватора, застосовуються будівельні практики та проекти, щоб блокувати більше сонячного світла. Наприклад, менші вікна використовуються для обмеження кількості сонячного світла, що потрапляє в будівлю, на відміну від холодніших регіонів, де співвідношення вікон до стін змінюється, щоб отримати більше сонячного світла. Важливим завданням для архітекторів є проектування будівель з певним виглядом, щоб вловлювати світло в певний час доби й уникати його в інший.

Нарешті, вологість є важливим фактором при виборі будівельних матеріалів і проектуванні конструкцій. Високий рівень вологості збільшує ймовірність конденсації та водяної ерозії в будівлях, тому для обмеження цієї проблеми вибираються водостійкі та антикорозійні матеріали. Крім того, циркуляція в будівлях з високою внутрішньою або зовнішньою вологістю є більш важливим фактором для захисту структурної цілісності.

Будівлі, дороги тощо мають бути спроектовані відповідно до майбутніх кліматичних умов. Більш вологі зими та раптові сильні зливи роблять ще більш важливим відведення дощової та талої води від будинків, асфальтованих ділянок, доріг тощо. Більш м’який клімат зменшить довговічність будівельних матеріалів і вплине на внутрішній клімат будівель. Тепле літо викличе більшу потребу в охолодженні. Більш високий рівень ґрунтових вод, більш високий рівень води в струмках і водотоках, а також більший ризик штормових хвиль уздовж берегової лінії роблять доцільним захистити будівлі від просочування та затоплення.

Будівлі можуть бути вразливими до зміни клімату. У майбутньому може виникнути збільшення ризику обвалу, погіршення здоров’я та значної втрати вартості в результаті нових штормів, снігу або пошкоджень, пов’язаних із

просіданням, потрапляння води, погіршення клімату в приміщенні та скорочення терміну служби будівлі. У короткостроковій перспективі сильніші шторми є найбільшою проблемою. Шторми становлять загрозу безпеці в тих частинах існуючих будівель, які не відповідають вимогам безпеки будівельних норм. У довгостроковій перспективі більш тривалі хвилі спеки можуть мати наслідки для здоров'я, особливо для людей похилого віку та слабких людей, наприклад, у будинках престарілих.

Адаптація може стосуватися обмеження снігового навантаження та збитку від шторму, а також контролю клімату в приміщенні. Що стосується зміцнення існуючих будівель, однак автономна адаптація буде обмежена, якщо власники не знайомі зі слабкими місцями в несучих елементах своїх будівель. Адаптація відбуватиметься лише в новобудовах, якщо стандарти будуть посилені. Що стосується протидії наслідкам теплових хвиль, можна очікувати встановлення систем кондиціонування повітря в існуючих будівлях, а також попит на будівлі з більш ефективним контролем внутрішнього клімату.

Індивідуальні власники будівель відповідають за дотримання відповідних норм, і саме вони шукатимуть рішення для задовільного клімату в приміщенні. У короткостроковій перспективі не буде змін у законодавстві щодо безпеки будівництва за екстремальних погодних умов. Для протидії тепловим хвилям нові правила щодо енергетичної основи в будівельних нормах є кроком до просування сонцезахисних і тепловідвідних вікон, що полегшить регулювання клімату в приміщенні. Наразі не рекомендовано жодних заходів щодо розширення або реконструкції. У майбутньому може виникнути потреба інформувати власників існуючих будівель про характерні недоліки несучих елементів з відповідними інструкціями щодо їх усунення. Таким же чином може виникнути потреба в інструкціях щодо нових будівельних рішень для зниження екстремальних температур у приміщенні під час спеки, особливо для вразливих будівель. Нарешті, може виникнути потреба поінформувати техніків-будівельників про рекомендовані параметри проектування, орієнтовані на майбутнє, наприклад, щодо максимального снігового навантаження та

швидкості вітру, температури та тривалості майбутніх хвиль спеки та максимальної інтенсивності опадів, яку має витримати будівля. Дороги майбутнього Нові дороги необхідно проектувати відповідно до майбутніх вимог. Тому було розпочато розслідування, щоб з'ясувати, як оновити поточні вказівки щодо планування, будівництва та управління дорогами. Дощ – найбільший фактор ризику. Для існуючих доріг ризик збільшення дощу становить найбільшу проблему. Поверхневі води необхідно відвести від доріг, щоб забезпечити їх довговічність, уникнути аквапланування та зниження прохідності для учасників дорожнього руху. Таким чином, дорожні органи наразі розглядають, як дренажні системи доріг можна адаптувати до майбутнього клімату, як щодо нового будівництва, так і під час управління існуючою інфраструктурою.

Під час екстремальних погодних явищ, таких як сильні зливи, шторми тощо, звіти про дорожній рух відіграють центральну роль для учасників дорожнього руху. Вони можуть постійно отримувати свіжі звіти про дорожній рух і погоду, наприклад, через радіо, мобільний телефон або GPS, а також через системи керування дорожнім рухом. Дослідження доріг і клімату надає дорожнім органам нові знання про те, як найкраще здійснювати майбутнє будівництво та управління датськими дорогами, зважаючи на наслідки зміни клімату для суспільства. Потужні шторми та збільшення швидкості вітру можуть мати фінансові та транспортні наслідки для електричних залізниць, наприклад, оскільки повітряні дроти вразливі до вищих швидкостей вітру. Підвищення рівня ґрунтових вод може призвести до підвищення ризику ерозії залізничних рейок. Більш сильні зливи можуть створити проблеми для дренажної системи залізниці, а ризик ерозії може зрости там, де перетинаються водотоки.

1.2. Стандартні кліматичні параметри

Дані, закладені у нормативних документах з будівельної кліматології, є вихідними параметрами вирішення завдань, що з проектуванням об'єктів. Тому дуже важливою є достовірність кліматичних параметрів, що включаються до норм. Помилки у вхідних параметрах зводять нанівець всі зусилля, пов'язані з розв'язанням оптимізаційних завдань, у тому числі задач зниження енергоємності будівель. Ще однією проблемою при вирішенні оптимізаційних завдань щодо енергоефективності будівель є нестача кліматичних параметрів, закладених у нормах. Заповнення необхідної кліматичної інформації може бути здійснено геометричними методами на основі аналізу фізичних закономірностей зміни в часі та просторі кліматичних параметрів. В Україні запроваджено норми щодо будівельної кліматології – ДСТУ-Н Б В.1.1–27:2011 «Будівельна кліматологія». Ці норми включають широкий набір кліматичних параметрів, достатній для вирішення більшості завдань проектування енергоефективних будівель. Їх розробка була проведена на основі системного аналізу нормативних документів з будівельної кліматології, що діють у СНД, тенденцій зміни клімату останнім часом та цілей розробки відповідного документа. У 2001 р. обґрунтовувалася необхідність підготовки нових будівельних норм щодо будівельної кліматології. Це було пов'язано з тим, що наприкінці 70-х років минулого століття розпочався глобальний період потепління, який суттєво змінив клімат України. Причому зміни торкнулися як температури повітря, а й циркуляції атмосфери, її хмарності і прозорості, розподілу та інтенсивності опадів, тобто змінилися практично всі кліматичні показники. Тоді в Україні при вирішенні завдань, пов'язаних з урахуванням впливу кліматичних факторів у будівництві використовувалися кліматичні параметри, СНиП 2.01.01–82 «Будівельна кліматологія та геофізика» та додатки до СНиП 2.01.07–85 «Карти районування території СРСР характеристик». Ці параметри були отримані статистичною обробкою даних метеорологічних спостережень за 30-80 років у період з 1881 по 1975 рік.

ДСТУ–Н Б В.1.1–27:2011 [3] має 10 основних розділів: архітектурно-будівельне кліматичне районування території України, температура

зовнішнього повітря, вітер, сонячна радіація, теплова радіація атмосфери та Землі, вологість повітря, фактор каламутності атмосфери, хмарність, опади та сніговий покрив, природне освітлення. Таким чином, охоплено всі необхідні кліматичні параметри, які використовують у проектній практиці архітектори, містобудівники та інженери. В основу пропонованого архітектурно-будівельного кліматичного районування території України покладено фізико-географічне районування України, згідно з [105], яке було уточнено фахівцями УкрНДГМІ за кліматичними даними, розрахованими на основі інформації метеорологічних спостережень на 53 метеорологічних станцій за період 1961–2005 років. районування території СРСР, згідно [4].

Для розрахунку добового ходу середньомісячної температури повітря проектувальникам рекомендується використовувати формули, отримані інтерполяцією екстремальних значень синусоїдальної функцією, запропоновані в [2], а для розрахунку ефективної температури повітря біля зовнішньої поверхні огорожі – формулу, запропоновану в [9, 6], яка враховує прямої та розсіяної сонячної радіації, а також теплової радіації атмосфер. Розділ, присвячений вітру, включає такі показники: щомісячні значення напряму переважаючого вітру, його повторюваність та середню швидкість; повторюваність та середня швидкість вітру по восьми румбах і повторюваність штилів у січні та липні. Розрахунок прямої та розсіяної сонячної радіації проведено за результатами вимірювань актинометричних станцій України. За побудованими картами просторового розподілу прямої та розсіяної радіації територією для окремих місяців визначено радіацію в обласних центрах.

На актинометричних станціях фіксуються значення інтенсивності прямої сонячної радіації та енергетичної освітленості горизонтальної площини розсіяною радіацією при ясному небі та середніх умовах хмарності. Значення енергетичної освітленості на площинах інших орієнтацій були отримані за допомогою ППП «Atmospheric Radiation» та програми SOLAR [1]. За результатами розрахунків у ДСТУ–Н Б В.1.1–27: 2011 складено таблиці середньомісячних доз прямої та розсіяної сонячної радіації, що надходять на

горизонтальну та вертикальну поверхні (восьми орієнтацій) за ясного неба та середніх умов хмарності для обласних центрів України; доз сумарної сонячної радіації за опалювальний період на вказані поверхні; годинні значення енергетичного освітлення цих поверхонь у січні та липні при ясному небі та повній хмарності для різних широт України (44, 46, 48, 50° с. ш.). Теплова радіація підрахована за методикою [8]. Наводяться характеристики теплової радіації, аналогічні характеристикам сонячної радіації. Вологість повітря нормується середньодобовою відносною вологістю та її амплітудою коливання кожного місяця. Ці показники є найчастіше використовуваними при архітектурно-кліматичному аналізі місцевості. Фактор каламутності атмосфери, хмарність, опади та сніговий покрив, природне освітлення вперше введено до будівельних норм. Ці показники є дуже важливими при розрахунку природної енергії, що приходить до будівель. Вони були використані при визначенні енергетичної освітленості вертикальних та горизонтальних поверхонь сонячної та теплової радіації за існуючих природно-кліматичних умов.

В Україні типові розрахункові кліматичні параметри стандартизовані у ДСТУ-Н Б В.1.1-27 2010 «Кліматологія». Цей стандарт установлює кліматичні параметри, що використовують при проектуванні будинків та споруд, систем опалення, вентиляції, кондиціонування, водозабезпечення, складанні енергетичного паспорта будинку.

У стандарті кліматичні параметри розраховано на основі метеорологічної інформації за період 1961 – 2005 рр.

У стандарті наведено дані про кліматичні параметри в обмеженій кількості населених пунктів. Для районів проектування та будівництва, що не увійшли до стандарту, кліматичні параметри приймають рівними значенням кліматичних параметрів найближчого до них пункту, наведеного в таблиці.

Територія України поділена на кліматичні райони на основі комплексного аналізу впливу середньомісячної температури повітря у січні та липні, середньої швидкості вітру у січні, середньої місячної відносної вологості повітря у липні та середньої річної кількості опадів.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМУ ВИЗНАЧЕННЯ КЛІМАТИЧНИХ ПАРАМЕТРІВ

2.1. Структура кліматичних даних ДСТУ

Для прозробки алгоритмів апроксимації кліматичних даних, що наведено в ДСТУ-Н Б В.1.1-27 2010 «Кліматологія», потрібно виконати аналіз структур їх таблиць. Можно виділити три основних типу таблиць:

- таблиці з даними для певного міста для кожного місяця та для року вцілому (тип А);
- таблиці з даними для певного міста для кожного місяця та для року вцілому для різних орієнтації поверхні (тип В);
- таблиці з даними для певної широти місця (тип С).

Перелік таблиць ДСТУ-Н Б В.1.1-27, що належать до типу А наведено в табл. 2.1.

Таблиця 2.1

Таблиці типу А

Номер таблиці	Назва
2	Температура зовнішнього повітря
4	Вітер
5	Характеристики вітру в січні
6	Характеристики вітру в липні
24	Відносна вологість повітря
25	Фактор мутності атмосфери (при оптичній масі атмосфери m=2)
26	Хмарність. Кількість хмарності
27	Хмарність. Кількість ясних та похмурих днів

29	Опади
30	Природна освітленість

Перелік таблиць ДСТУ-Н Б В.1.1-27, що належать до типу В наведено в табл. 2.2.

Таблиця 2.2

Таблиці типу В

Номер таблиці	Назва
7	Середньомісячні дози сонячної радіації, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за ясного неба
8	Середньомісячні дози сонячної радіації, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за середніх умов хмарності
9	Доза сумарної сонячної радіації за опалювальний період, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за середніх умов хмарності
19	Середньомісячні дози теплової радіації, що надходять на горизонтальну та вертикальні поверхні

Перелік таблиць ДСТУ-Н Б В.1.1-27, що належать до типу С наведено в табл. 2.3.

Таблиця 2.3

Таблиці типу С

Номер таблиці	Назва

10	Енергетична освітленість площин різної орієнтації сонячною радіацією у січні за умов ясного неба на 44º пн.ш.
11	Енергетична освітленість площин різної орієнтації сонячною радіацією у січні за умов ясного неба на 46º пн.ш.
12	Енергетична освітленість площин різної орієнтації сонячною радіацією у січні за умов ясного неба на 48º пн.ш.
13	Енергетична освітленість площин різної орієнтації сонячною радіацією у січні за умов ясного неба на 50º пн.ш.
14	Енергетична освітленість площин різної орієнтації сонячною радіацією у липні за умов ясного неба на 44º пн.ш.
15	Енергетична освітленість площин різної орієнтації сонячною радіацією у липні за умов ясного неба на 46º пн.ш.
16	Енергетична освітленість площин різної орієнтації сонячною радіацією у липні за умов ясного неба на 48º пн.ш.
17	Енергетична освітленість площин різної орієнтації сонячною радіацією у липні за умов ясного неба на 50º пн.ш.
18	Енергетична освітленість площин розсіяною сонячною радіацією за умов 10-бальної хмарності неба на різних широтах
20	Енергетична освітленість площин тепловою радіацією на 44° пн. ш.
21	Енергетична освітленість площин тепловою радіацією на 46° пн. ш.
22	Енергетична освітленість площин тепловою радіацією на 48° пн. ш.
23	Енергетична освітленість площин тепловою радіацією на 50° пн.ш.

З урахуванням особливостей таблиць розглянемо підходи до апроксимації даних таблиць та розробимо алгоритми апроксимації.

2.2. Апроксимація даних таблиць ДСТУ

Деякі кліматичні дані, що наведено у стандарті, доцільно попередньо апроксимувати для заданої точки проектування будівлі.

Апроксимацію необхідно виконувати з урахуванням кривизни Земної кулі, тому апроксимація виконується за географічними координатами найближчих міст, для яких є табличні дані, та точки проектування.

Розглянемо математичні основи такої апроксимації.

Обчислення кута утвореного трьома точками на сфері (рис. 2.1).

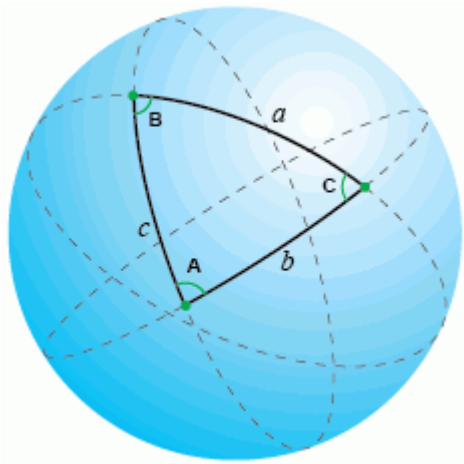

Рис. 2.1. Трикутник на сфері.

Відповідно до сферичної теореми косинусів: косинус однієї сторони сферичного трикутника дорівнює добутку косинусів двох інших його сторін плюс добуток синусів тих же сторін на косинус кута між ними. Одна з відмінностей сферичного трикутника від звичайного полягає в тому, що сума його кутів перевищує 180 градусів. У рівняннях використовуються кутові довжини у радіанах.

$$\cos a = \cos b * \cos c + \sin b * \sin c * \cos A \tag{2.1}$$

Звідси, знаючи координати трьох точок та обчисливши відстані між ними за формулами обчислення відстаней на сфері, можна отримати будь-який із трьох кутів, наприклад кут А:

$$\cos A = \arccos\left(\frac{\cos a - \cos b * \cos c}{\sin b * \sin c}\right) \quad (2.2)$$

Обчислення відстані та початкового азимуту між двома точками на сфері. Вимірювання відстані та початкового азимуту між точками без проекційних перетворень. Довжина дуги великого кола – найкоротша відстань між будь-якими двома точками, що знаходяться на поверхні сфери, виміряна вздовж лінії, що з'єднує ці дві точки (така лінія носить назву ортодромії) і проходить по поверхні сфери або іншої поверхні обертання. Сферична геометрія відрізняється від звичайної Евклідової та рівняння відстані також набувають іншої форми. У Евклідовій геометрії найкоротша відстань між двома точками – пряма лінія. На сфері прямих ліній не буває. Ці лінії на сфері є частиною великих кіл – кіл, центри яких збігаються з центром сфери. Початковий азимут - азимут, взявши який на початку руху з точки А, слідуючи по великому колу на найкоротшу відстань до точки В, кінцевою точкою буде точка В. При русі з точки А в точку В по лінії великого кола азимут з поточного положення на кінцеву точку В постійно змінюється. Початковий азимут, за яким азимут з поточної точки на кінцеву не змінюється, але маршрут слідування не є найкоротшою відстанню між двома точками.

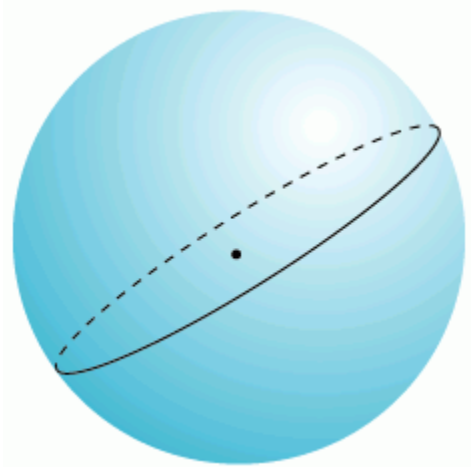

Рис. 2.2. Велике коло на сфері

Через будь-які дві точки на поверхні сфери, якщо вони не прямо протилежні один одному (тобто не є антиподами), можна провести унікальне велике коло. Дві точки розділяють велике коло на дві дуги. Довжина короткої дуги – найкоротша відстань між двома точками. Між двома точками-антиподами можна провести нескінченну кількість великих кіл, але відстань між ними буде однаково на будь-якому колі і дорівнює половині кола, або pi*R, де R - радіус сфери.

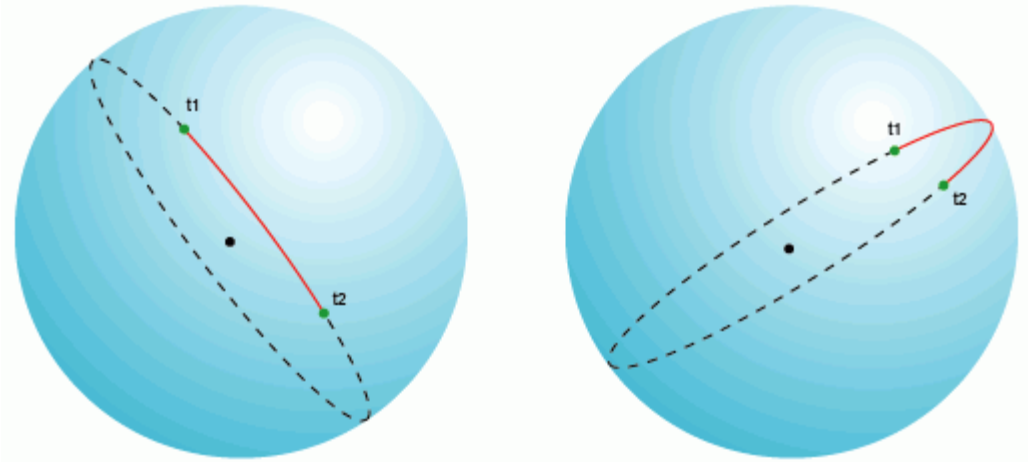

Рис. 2.3. Відстань великого кола

На площині (у прямокутній системі координат) великі кола та їх фрагменти, як було згадано вище, являють собою дуги у всіх проекціях, крім гномонічної, де великі кола - прямі лінії. На практиці це означає, що літаки та інший авіатранспорт завжди використовує маршрут мінімальної відстані між точками для економії палива, тобто політ здійснюється на відстані великого кола, на площині це виглядає як дуга.

Форма Землі може бути описана як сфера, тому рівняння для обчислення відстаней на великому колі є важливими для обчислення найкоротшої відстані між точками на поверхні Землі і часто використовуються в навігації. Обчислення відстані цим методом більш ефективно і в багатьох випадках більш точно, ніж обчислення його для спроектованих координат (у прямокутних

системах координат), оскільки, по-перше, для цього не треба переводити географічні координати прямокутну систему координат (здійснювати проекційні перетворення) і, по-друге, багато проекцій, якщо неправильно обрані, можуть призвести до значних спотворень довжин через особливості проекційних спотворень. Відомо, що точніше описує форму Землі не сфера, а еліпсоїд, однак у будемо розглядати обчислення відстаней саме на сфері, для обчислень використовується сфера радіусом 6372795 метрів, що може призвести до помилки обчислення відстаней близько 0.5%.

Існує три способи розрахунку сферичної відстані великого кола. Сферична теорема косінусів - у разі невеликих відстаней і невеликої розрядності обчислення (кількість знаків після коми), використання формули може призводити до значних помилок пов'язаних із округленням.

$$\Delta\sigma = \arccos\left\{\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda\right\} \tag{2.3}$$

де $\phi_1, \lambda_1;\ \phi_2, \lambda_2$ - широта та довгота двох точок у радіанах;

$\Delta\lambda$ - Різниця координат по довготі;

$\Delta\sigma$ - кутова різниця.

Для переведення кутової відстані в метричну, потрібно кутову різницю помножити на радіус Землі (6372795 м), одиниці кінцевої відстані дорівнюватимуть одиницям, в яких виражений радіус (в даному випадку - метри).

Формула гаверсінусів використовується, щоб уникнути проблем із невеликими відстанями.

$$\Delta\sigma = 2\arcsin\left\{\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right\}. \tag{2.4}$$

Попередня формула також схильна до проблеми точок-антиподів, щоб її вирішити використовується наступна її модифікація.

$$\Delta\sigma = \arctan\left\{\frac{\sqrt{[\cos\phi_2 \sin\Delta\lambda]^2 + [\cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos\Delta\lambda]^2}}{\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda}\right\} \quad (2.5)$$

Виміряти відстань та азимут між двома точками можна кількома різними способами. Розглянемо різницю між різними способами обчислення. Незважаючи на простоту, операція визначення довжини лінії, що з'єднує дві точки і азимуту з однієї точки на іншу може призводити до різних результатів в залежності від того, як проводиться це обчислення. Перед здійсненням цих обчислень слід чітко визначити, яку саме відстань/азимут потрібно обчислити і навіщо. Обчислення відстані може здійснюватися одним з наступних методів:

Геодезичні координати, відстань - довжина дуги великого кола, що проходить через дві точки. Довжина дуги великого кола може бути розрахована для двох випадків:

– сфери;
– еліпсоїда.

Прямокутні координати або геодезичні координати, відстань довжина прямої лінії, що з'єднує дві точки:

– довжина лінії румба (локсодромна відстань) - обчислюється в проекції Меркатора, в якій всі прямі лінії - лінії румба (перетинають паралелі під постійним кутом)
– вимірювання в інших проекціях - довжини ліній, що з'єднують дві точки, будуть різні для різних проекцій. Проекції, що має постійний масштаб довжин для будь-яких напрямків, не існує в природі.

Orthodrome – довжина дуги великого кола (найкоротша відстань) на сфері, Loxodrome – довжина лінії румба.

Локсодромна відстань ніколи не перевищує ортодромну, а значення спроектованих даних досить сильно відрізняються від вимірювань на сфері.

Якщо точки знаходяться на практично одній довготі, різниця між довжиною дуги великого кола та лінією румба мінімальна, оскільки меридіани є одночасно й великими колами та лініями румба.

Вимір азимуту з першої точки на другу також може здійснюватися по-різному:

- геодезичні координати, початковий (ортодромний) азимут - кут при старті з першої точки та руху по найкоротшій відстані, кінцевою точкою є друга точка. Початковий азимут не постійний протягом руху.
- геодезичні координати, постійний (локсодромний) азимут - постійний кут, за яким можна потрапити з першої точки в другу. Відстань між двома точками при цьому не є найкоротшою.
- прямокутні координати, в більшості випадків - азимут - напрямок від однієї точки на іншу (північ - 0). Частковий випадок: постійний (локсодромний) азимут у разі використання проекції Меркатора (результати розрахунку ідентичні варіанту 2.

В залежності від розрахунку, що використовується, результати можуть сильно відрізнятися від значення постійного (локсодромного) азимуту. До особливо сильно різним значенням наводить використання для розрахунків спроектованих даних. Якщо точки знаходяться на практично одній довготі, різниця між ортодромним і локсодромним азимутом мінімальна, оскільки меридіани є одночасно і великими колами та лініями румба.

2.3. Структура таблиць бази даних

Для зберігання даних таблиць ДСТУ використовуємо SQLite. SQLite - це бібліотека, яка реалізує самодостатній безсерверний механізм транзакційної бази даних SQL без конфігурації. Вихідний код для SQLite знаходиться у відкритому доступі. Ця база даних не потребує налаштування.

Бібліотека SQLite не автономний процес, як інші бази даних, її можно зв’язати статично чи динамічно з додатком. SQLite отримує прямий доступ до своїх файлів. Для роботи SQLite не потрібен окремий серверний процес або система.

Повна база даних SQLite зберігається в одному файлі.

Бібліотека SQLite дуже мала, менше 400 КБ у повністю налаштованому вигляді або менше 250 КБ без додаткових функцій. SQLite є самодостатньою, що означає відсутність зовнішніх залежностей.

Транзакції SQLite повністю сумісні з ACID, що забезпечує безпечний доступ з кількох процесів або потоків. SQLite підтримує більшість функцій мови запитів, наявних у стандарті SQL92 (SQL2).

Бібліотека написанв на мові ANSI-C, що забезпечує простий у використанні API. SQLite доступний для ОС UNIX (Linux, Mac OS-X, Android, iOS) і ОС Windows (Win32, WinCE, WinRT).

Є кілька непідтримуваних функцій SQL92 у SQLite, які перераховані в таблиці (табл. 2.4).

Таблиця 2.4

Обмеження SQLite

№	Функція	Опис
1	RIGHT OUTER JOIN	Реалізовано лише LEFT OUTER JOIN.
2	FULL OUTER JOIN	Реалізовано лише LEFT OUTER JOIN.
3	ALTER TABLE	Підтримуються варіанти RENAME TABLE і ADD COLUMN команди ALTER TABLE. DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT не підтримуються.
4	Підтримка тригера	Тригери FOR EACH ROW підтримуються, але не тригери FOR EACH STATEMENT.

5	VIEWs	VIEWs в SQLite доступні лише для читання.
6	GRANT and REVOKE	Єдині дозволи доступу, які можна застосувати, це звичайні дозволи доступу до файлів основної операційної системи.

Стандартні команди SQLite для взаємодії з реляційними базами даних подібні до SQL. Це CREATE, SELECT, INSERT, UPDATE, DELETE і DROP. Ці команди можна класифікувати на групи на основі їх операційної природи.

Для зберігання кліматичних даних з ДСТУ розроблена наступна система таблиць баз даних (табл. 2.5).

Таблиця 2.5

Структура БД

Назва таблиці	Опис
TableName	Назва таблиць
Name	Назва стовпців таблиць
City	Інформація про міста
Table1	Кліматологічні показники (характеристики) архітектурно-будівельних кліматичних районів та підрайонів
Table2	Температура зовнішнього повітря
Table3	Дати початку та закінчення опалювального періоду
Table4	Вітер
Table5	Характеристики вітру в січні
Table6	Характеристики вітру в липні
Table7	Середньомісячні дози сонячної радіації, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за ясного неба
Table8	Середньомісячні дози сонячної радіації, що надходить на горизонтальну та вертикальну поверхні різної орієнтації

	за середніх умов хмарності
Table9	Доза сумарної сонячної радіації за опалювальний період, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за середніх умов хмарності
Table10	Енергетична освітленість площин різної орієнтації сонячною радіацією у січні за умов ясного неба
Table14	Енергетична освітленість площин різної орієнтації сонячною радіацією у липні за умов ясного неба на
Table18	Енергетична освітленість площин розсіяною сонячною радіацією за умов 10-бальної хмарності неба
Table19	Середньомісячні дози теплової радіації
Table20	Енергетична освітленість площин тепловою радіацією
Table24	Відносна вологість повітря
Table25	Фактор мутності атмосфери
Table26	Хмарність. Кількість хмарності
Table27	Хмарність. Кількість ясних та похмурих днів
Table28	Хмарність. Висота нижньої межі хмар
Table29	Опади
Table30	Природна освітленість

Для забезпечення однотипності даних у таблицях бази даних необхідно назви стовпців зберігати окремо. Назви стовпців таблиць даних наведено в табл. 2.6-2.10). Назви стовпців таблиць даних об’єднані в одну таблицю бази даних.

Таблиця 2.6

Назва стовпців таблиць Table5 «Характеристики вітру в січні», Table6 «Характеристики вітру в липні»

Назва	Опис

Dir1, Dir2, Dir3, Dir4, Dir5, Dir6, Dir7, Dir8, Dir9, Dir10, Dir11, Dir12	Переважний напрям вітру
Poss1, Poss2, Poss3, Poss4, Poss5, Poss6, Poss7, Poss8, Poss9, Poss10, Poss11, Poss12	Повторюваність переважного напряму вітру
Speed1, Speed2, Speed3, Speed4, Speed5, Speed6, Speed7, Speed8, Speed9, Speed10, Speed11, Speed12,	Середня швидкість вітру, м/с
PossDir0, PossDir45, PossDir90, PossDir135, PossDir180, PossDir225, PossDir270, PossDir315	Повторюваність напряму вітру
PossSpeed0	Повторюваність штилю
SpeedDir0, SpeedDir45, SpeedDir90, SpeedDir135, SpeedDir180, SpeedDir225, SpeedDir270, SpeedDir315	Середня швидкість вітру

Таблиця 2.7

Назва стовпців таблиці Table2 «Температура зовнішнього повітря»

Назва	Опис
Tmm1, Tmm2, Tmm3, Tmm4, Tmm5, Tmm6, Tmm7, Tmm8, Tmm9, Tmm10, Tmm11, Tmm12	Середня місячна температура повітря
Tmpp1, Tmpp2, Tmpp3, Tmpp4, Tmpp5, Tmpp6, Tmpp7, Tmpp8, Tmpp9, Tmpp10, Tmpp11, Tmpp12	Середня місячна середня добова амплітуда температури
Tmyear	Середня за рік температура повітря
Tcday098	Температура повітря холодного періоду найхолоднішої доби забезпеченістю 0.98
Tcday092	Температура повітря холодного періоду найхолоднішої доби забезпеченістю 0.92
Tc5day098	Температура повітря холодного періоду найхолоднішої п’ятиденки забезпеченістю

	0.98
Tc5day092	Температура повітря холодного періоду найхолоднішої п’ятиденки забезпеченістю 0.92
Thday095	Температура повітря теплого періоду найжаркішої доби забезпеченістю 0,95
Th5day099	Температура повітря теплого періоду найжаркішої п’ятиденки забезпеченістю 0,99
Nday8	Тривалість періоду із середньою добовою температурою повітря <= 8 °С
Tmday8	Середня температура періоду із середньою добовою температурою повітря <= 8 °С
Nday10	Тривалість періоду із середньою добовою температурою повітря <=10 °С
Tmday10	Середня температура періоду із середньою добовою температурою повітря <=10 °С
Nday21	Тривалість періоду із середньою добовою температурою повітря >=21 °С
Tmday21	Середня температура періоду із середньою добовою температурою повітря >=21 °С

Таблиця 2.8

Назва стовпців таблиць Table24 «Відносна вологість повітря»

Назва	Опис
M1_HUMMEAN, M2_HUMMEAN, M3_HUMMEAN, M4_HUMMEAN, M5_HUMMEAN, M6_HUMMEAN, M7_HUMMEAN, M8_HUMMEAN, M9_HUMMEAN, M10_HUMMEAN, M11_HUMMEAN, M12_HUMMEAN	Середня місячна відносна вологість
Y_HUMMEAN	Середня за рік відносна вологість
M1_HUMPP, M2_HUMPP, M3_HUMPP, M4_HUMPP, M5_HUMPP, M6_HUMPP,	Середня місячна середня добова амплітуда відносної

M7_HUMPP, M8_HUMPP, M9_HUMPP, M10_HUMPP, M11_HUMPP, M12_HUMPP	вологості

Таблиця 2.9

Назва стовпців таблиць Table25 «Фактор мутності атмосфери»

Назва	Опис
M1_FMMEAN, M2_FMMEAN, M3_FMMEAN, M4_FMMEAN, M5_FMMEAN, M6_FMMEAN, M7_FMMEAN, M8_FMMEAN, M9_FMMEAN, M10_FMMEAN, M11_FMMEAN, M12_FMMEAN	Середній по місяцях фактор мутності
Y_FMMEAN	Середній фактор мутності за рік
M1_FMPP, M2_FMPP, M3_FMPP, M4_FMPP, M5_FMPP, M6_FMPP, M7_FMPP, M8_FMPP, M9_FMPP, M10_FMPP, M11_FMPP, M12_FMPP	Середній по місяцях середня добова амплітуда фактора мутності"

Таблиця 2.10

Назва стовпців таблиць Table26 «Хмарність. Кількість хмарності»

Назва	Опис
M1_CLALL, M2_CLALL, M3_CLALL, M4_CLALL, M5_CLALL, M6_CLALL, M7_CLALL, M8_CLALL, M9_CLALL, M10_CLALL, M11_CLALL, M12_CLALL	Загальна кількість хмарності
Y_CLALL	Середня за рік загальна кількість хмарності
M1_CLLOW, M2_CLLOW, M3_CLLOW, M4_CLLOW, M5_CLLOW, M6_CLLOW,	Кількість нижньої хмарності

M7_CLLOW, M8_CLLOW, M9_CLLOW, M10_CLLOW, M11_CLLOW, M12_CLLOW	
Y_CLLOW	Середня за рік кількість нижньої хмарності

На основі створених таблиць БД розроблені SQL запити до бази даних. Найбільш прості запити, наведено нижче.

Запит списку індексу та координат населеного пункту

```
SELECT Idx,City,Lat,Lon FROM City
```

Запит назви стовпців таблиці

```
SELECT Name FROM TableName WHERE TableNumber= %1
```

%1 – номер таблиці

Запит визначення координат населеного пункту

```
SELECT Table%1.Idx, City.Lat, City.Lon FROM Table%1, City WHERE Table%1.CityIndex=City.CityIndex
```

%1 – номер таблиці

Запити для визначення груп населених пунктів, широта яких більша та менша за вказану широту

```
SELECT Table%1.Idx, City.Lat, City.Lon FROM Table%1, City WHERE Table%1.CityIndex=City.CityIndex AND City.Lat<%2
SELECT Table%1.Idx, City.Lat, City.Lon FROM Table%1, City WHERE Table%1.CityIndex=City.CityIndex AND City.Lat>=%2
```

%1 – номер таблиці

%2 – вказане широта

Для отримання безпосередньо даних з таблиць віикористовуються динамічні запити.

2.4. Інтерфейс програми та основні структури даних

Розробка інтерфейсу програми виконується за допомогою QtCreator [10].

Qt Creator надає крос-платформне повне інтегроване середовище розробки (IDE) для розробників додатків для створення додатків для кількох платформ настільних комп'ютерів, вбудованих і мобільних пристроїв, таких як Android та iOS. Qt Creator доступний для операційних систем Linux, macOS і Windows.

Qt Creator надає інструменти для проектування та розробки додатків за допомогою інфраструктури додатків Qt. Qt призначений для одночасної розробки додатків та інтерфейсів користувача та їх розгортання в кількох настільних, вбудованих і мобільних операційних системах або веб-браузерах. Qt Creator надає інструменти для виконання задач протягом усього життєвого циклу розробки програми, від створення проекту до розгортання програми на цільових платформах.

Управління проектами. Щоб мати можливість створювати та запускати програми, Qt Creator потребує тієї ж інформації, що й компілятор. Ця інформація вказується в налаштуваннях проекту. Налаштувати новий проект у Qt Creator допомагає майстер, який при створенні проекту, створює необхідні файли та вказує параметри залежно від вибору, який ви робите.

Проектування інтерфейсів користувача. Щоб створити інтуїтивно зрозумілий, сучасний, плавний інтерфейс користувача, можна використовувати Qt Quick і Qt Design Studio. Якщо потрібен традиційний інтерфейс користувача, який має чітку структуру та забезпечує зовнішній вигляд платформи, можна скористатися інтегрованим редактором Qt Designer.

IDE Qt Creator дозволяє для мов C++ і QML використовувати такі функції, як семантичне підсвічування, перевірка синтаксису коду, доповнення коду та рефакторинг. Qt Creator підтримує деякі з цих служб також для інших мов програмування, таких як Python, для яких доступний мовний сервер, який надає інформацію про код до IDE.

Qt Creator інтегровано з кросплатформенними системами для автоматизації збирання: qmake, Qbs, CMake та Autotools. Крім того, можна імпортувати проекти як загальні проекти та повністю контролювати кроки та команди, які використовуються для створення проекту.

Qt Creator забезпечує підтримку запуску та розгортання програм Qt, створених для робочого середовища або пристрою. Комплекти, параметри збірки, запуску та розгортання дозволяють швидко перемикатися між різними налаштуваннями та цільовими платформами.

Qt Creator інтегрований у кілька зовнішніх налагоджувачів: GNU Symbolic Debugger (GDB), Microsoft Console Debugger (CDB) і внутрішній налагоджувач JavaScript. У режимі налагодження можна перевірити стан програми.

Обсяг пам’яті та потужності процесора, доступні на пристроях, обмежені, тому їх слід використовувати обережно. Qt Creator інтегрує інструменти аналізу коду Valgrind для виявлення витоків пам’яті та виконання функцій профілювання. Крім того, QML Profiler дає змогу профілювати програми Qt Quick.

Qt Creator надає інтегрований візуальний редактор для розробки додатків на основі віджетів у режимі «Design». Інтеграція включає в себе управління проектом і доповнення коду.

Можна виконувати розробку програми Qt Quick у режимі Edit або використовувати окремий візуальний редактор Qt Design Studio.

Розробка додатків Qt Quick. Можна використовувати майстри для створення проектів Qt Quick, що містять шаблонний код, який можна редагувати в режимі Edit.

Розробка додатків на основі віджетів. Віджети та форми, створені за допомогою Qt Designer, бездоганно інтегруються з запрограмованим кодом за допомогою механізму сигналів і слотів Qt, який дозволяє легко призначати поведінку графічним елементам. Усі властивості, встановлені в Qt Designer, можна динамічно змінювати в коді. Крім того, такі функції, як просування віджетів і користувацькі плагіни, дозволяють використовувати власні віджети з Qt Designer.

Для забезпечення функціонування алгоритму програми, необхідно надати користовачу можливість ввести координати місця розташування об’єкту та режим апроксимації кліматичних даних. Для вводу координат використовуються два поля QLineEdit.

У цих полях передбачається введення широти та довготи у наступних форматах:

– градуси_хвилини_секунди з дробною частиною;
– градуси_хвилини;
– градуси_з дробною частиною.

Макет вікна представлено на рис. 2.4.

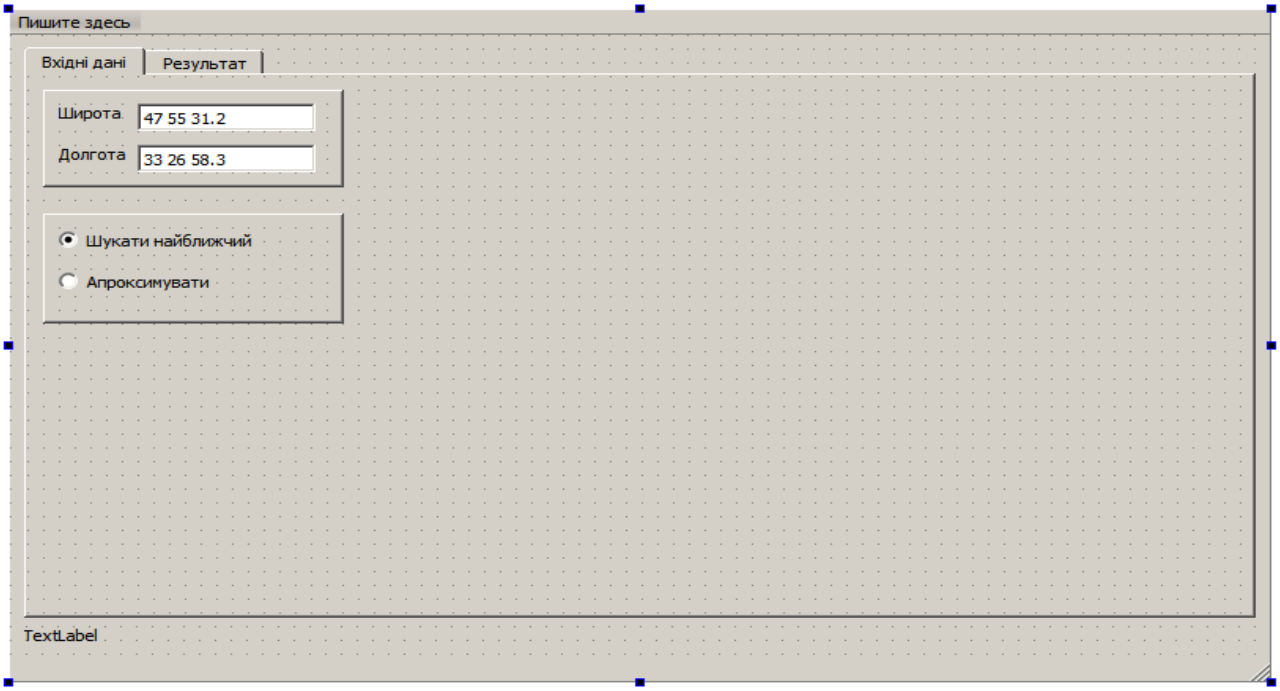

Рис. 2.4. Структура віджетів окна вводу даних.

Результат роботи програми відображається у вікні, де основним віджетом є QGridLayout у якому при виконанні програми додається віджет таблиці. (рис. 2.5).

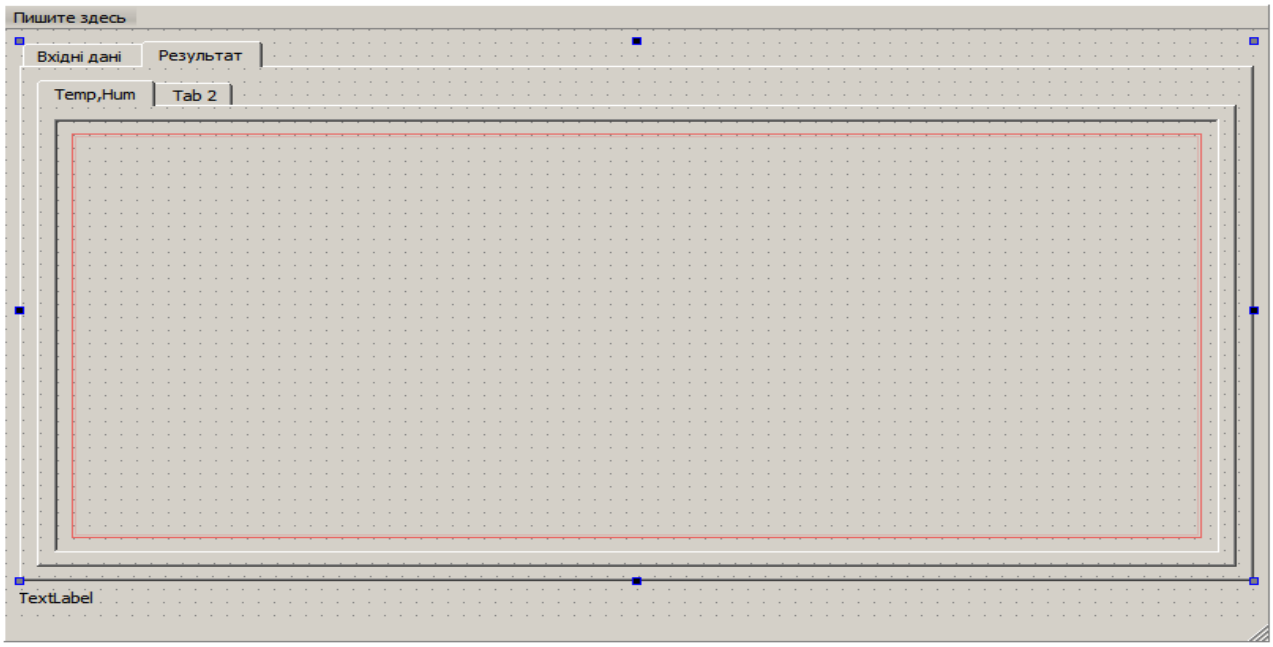

Рис. 2.5 Структура віджетів окна для відображення результатів

Структура віджетів, які використовуються, представлена на рис. 2.6.

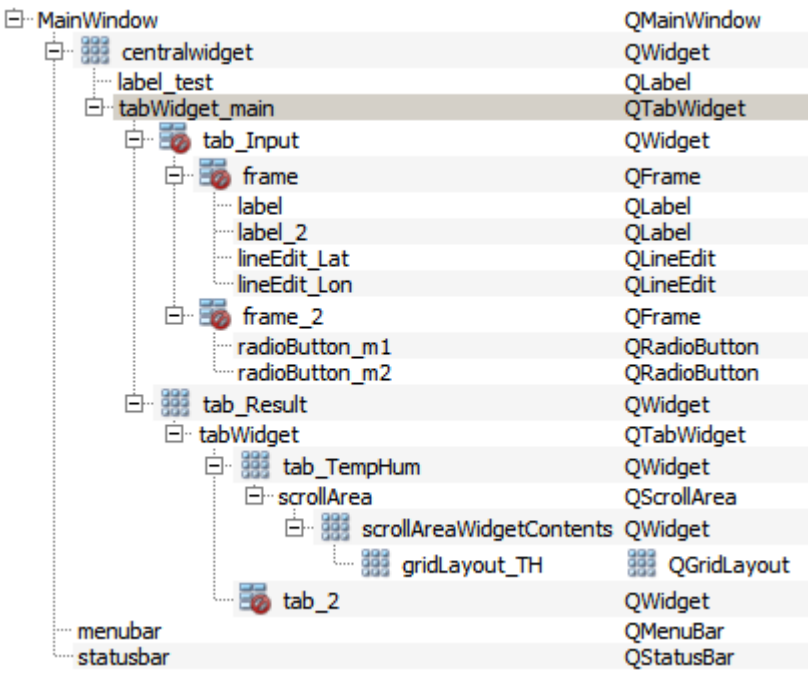

Рис. 2.6 Дерево віджетів та класів

Розглянемо структуру даних функціональної частини програми.

Для відображення, таблиці кліматичних даних зберігаються у векторах QVector<> структур наступного виду:

```
typedef struct
{
    QString table_num; // Номер таблиці
    QString table_name; // Назва таблиці
    QVector<TableItem_t> data; // Дані таблиці
    QTableWidget *table; // Віджет таблиці
    QLabel *label; // Віджет назви таблиці
}Table_Data_t;
```

Головне вікно програми пов’язане з класом MainWindow, нащадком від QMainWindow:

```
class MainWindow : public QMainWindow
{    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

Слоти для взаємодії з віджетами

```
private slots:
    void on_tabWidget_main_currentChanged(int index);
    void Lat_adjustTextColor(QString);
    void Lon_adjustTextColor(QString);
```

Захищені дані класу MainWindow

```
private:
```

```
        Ui::MainWindow *ui; // Об’єкт візуального інтерфейсу
        TableExcel *tabexl; // Об’єкт для створення файлу Excel
        long Mode; // Режим роботи апроксимації
        double Lat, Lon; // Широта та довгота розрахункової точки
        DB_data *DSTU_data; // Об’єкт бази даних
        QVector<Table_Data_t> TableData; // Масив таблиць даних
```

Захищені методи класу MainWindow

```
        double Convert_CoordAngel(QString str); // Перетворення строки географічних координат у число
        void Calculate_A(); // Обчислення типу A
        void Calculate_B();// Обчислення типу B
        void Calculate_C();// Обчислення типу C
        void CreateTable(Table_Data_t* d, QWidget *parent ); // Відображення однорядкової таблиці
        void CreateTableArr(Table_Data_t* d, QWidget *parent, QVector< QVector<QVariant> > *data_arr ); // Відображення багаторядкової таблиці
    };
```

Для збарігання кліматичних даних обрано систему управління базами даних SQLite, яка забезпечує її інтегрування до проекту і не потребує додаткового серверу. Реалізація інтерфейсу для організації запитів до бази даних виконується за допомогою класу DB_data.

Результат запитів до бази даних перетворюється за допомогою наступних структур:

```
    typedef struct{
        QString Name; // Назва міста
        uint Idx; // Індекс в таблиці
        double Lat; // Широта
        double Lon; // Довгота
```

```
} City_data_t;
typedef struct{
    long Idx; // Індекс в таблиці БД
    QString Item; // Назва стовпця в таблиці БД
    QString Name1; //Перша частина назви стовпця в таблиці БД
    QString Name2; //Друга частина назви стовпця в таблиці БД
    QString Name3; //Третя частина назви стовпця в таблиці БД
    QVariant Data; // Дані запита
} TableItem_t;
```

Структура класу взаємодії з БД

```
class DB_data{
public:
    DB_data(); // Конструктор
    QSqlDatabase *db; // Об’єкт бази даних
```

Публічні методи для взаємодії з базою даних

Отримання таблиць типу A

```
    QVector<TableItem_t> GetTable_A(QString num_table, double city_Lat, double city_Lon, QString *table_name);
```

Отримання таблиць типу B при апроксимації парамертів по двом найближчім точкам, які є в базі

```
    QVector<TableItem_t>  GetTable_B_appr(QString  num_table, double city_Lat, double city_Lon, QString *table_name);
```

Отримання таблиць типу C

```
        QVector<TableItem_t> GetTable_C(QString num_table, double
city_Lat, double city_Lon, QString *table_name, QVector<
QVector<QVariant> > *data_res);
```

Отримання таблиць типу В найближчій точці, що є в БД

```
        QVector<TableItem_t> GetTable_B_near(QString num_table,
double city_Lat, double city_Lon, QString *table_name);
```

Обчислення відстані між двома точками

```
        double Distance_2Point(double lat1, double lon1, double
lat2, double lon2, double *alpha=0);
```

Запит таблиці міст

```
        QVector<City_data_t> get_City_data();
```

Запит з БД

```
        QVector<QVariant> get_Record(QString req);
```

Запит назв стовпців таблиці

```
        QVector<TableItem_t> get_TableItem(QString num_table);
    };
```

Створені структури користувальницького інтерфейсу програми та структури даних дозволяють розробити фунціональну частину для апроксимації кліматичних даних.

РОЗДІЛ 3
ОПИС ПРОГРАМИ РОЗРАХУНКУ КЛІМАТИЧНИХ ПАРАМЕТРІВ

3.1. Структура програмних модулів користувальницького інтерфейсу

Головний модуль програми представлений головним вікном MainWindow. Створення об’єктів інтерфейсу виконується в конструкторі класу

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this); // створення графічного інтерфейсу
```

Для забезпечення коректного вводу координат місцевості передбачено створення валідатора на основі регулярного виразу для елементів вводу координат. Передбачено введення широти та довготи у наступних форматах:

градуси – хвилини - секунди з дробною частиною;

градуси – хвилини;

градуси – з дробною частиною.

```
        QRegExp
re("(\\d|\\d\\d|\\d\\d\\d)\\s+(\\d|\\d\\d)(\\s+((\\d|\\d\\d)(.\\d+
)?))?|((\\d|\\d\\d|\\d\\d\\d).\\d+)");
        QRegExpValidator *validator = new QRegExpValidator(re,
this);
        ui->lineEdit_Lat->setValidator(validator);
        ui->lineEdit_Lon->setValidator(validator);
```

У елементів введення координат добавлено слоти для зміни кольору тексту при некоректному рядку.

```
        connect(ui->lineEdit_Lat,    SIGNAL(textChanged(QString)),
this, SLOT(Lat_adjustTextColor(QString)));
        connect(ui->lineEdit_Lon,    SIGNAL(textChanged(QString)),
this, SLOT(Lon_adjustTextColor(QString)));
```

Створення об’єкта бази даних кліматичних параметрів

```
    DSTU_data = new DB_data;
```

Створення об’єкту для роботи з файлами Excel

```
    tabexl = new TableExcel();
}
```

В деструкторі виконується звільнення пам’яті, яка розподілялась в конструкторі.

```
MainWindow::~MainWindow()
{
    delete ui;
    delete tabexl;
    DSTU_data;
}
```

Методи класу для зміни кольору тексту

```
void MainWindow::Lat_adjustTextColor(QString s){
    if(!ui->lineEdit_Lat->hasAcceptableInput())
        ui->lineEdit_Lat->setStyleSheet("QLineEdit   {   color:
red;}");
    else
```

```
            ui->lineEdit_Lat->setStyleSheet("QLineEdit   {   color:
black;}");
    }
    void MainWindow::Lon_adjustTextColor(QString s){
        if(!ui->lineEdit_Lon->hasAcceptableInput())
            ui->lineEdit_Lon->setStyleSheet("QLineEdit   {   color:
red;}");
        else
            ui->lineEdit_Lon->setStyleSheet("QLineEdit   {   color:
black;}");
    }
```

Метод перетворення кута з рядка у число

```
    double MainWindow::Convert_CoordAngel(QString str)
    {
        double d=0;
        bool ok;
```

Розділення рядка на підрядки

```
        QStringList str_arr=str.split(QRegExp("\\s+"));
```

Якщо один елемент, то в рядку градуси з дробною частиною

```
        if(str_arr.size()==1) //
        {
            d = str_arr[0].toDouble(&ok);
        }
```

Якщо два елементи, то в рядку цілі градуси і хвилини з дробною частиною

```
    else if(str_arr.size()==2)
    {
        d = str_arr[0].toDouble(&ok);
        if(str_arr[1].toDouble(&ok)<60)
            d += str_arr[1].toDouble(&ok)/60.0;
    }
```

Якщо три елементи, то в рядку цілі градуси, цілі хвилини і секунди з дробною частиною

```
    else if(str_arr.size()==3)
    {
        d = str_arr[0].toDouble(&ok);
        if(str_arr[1].toDouble(&ok)<60)
            d += str_arr[1].toDouble(&ok)/60.0;
        if(str_arr[2].toDouble(&ok)<60)
            d += str_arr[2].toDouble(&ok)/3600.0;
    }
    return d;
}
```

Метод підготовки даних та обчислення усіх таблиць

```
void MainWindow::on_tabWidget_main_currentChanged(int index)
{
```

Зчитування та перетворювання у число координат точки та режиму обчислень

```
    QString str;
    Lat = Convert_CoordAngel( ui->lineEdit_Lat->text());
    Lon = Convert_CoordAngel( ui->lineEdit_Lon->text());
    Mode = ui->radioButton_m1->isChecked();
    str = QString("lat %1  lon %2").arg(Lat).arg(Lon);
```

```
    ui->label_test->setText(str);
```

Виконання обчислень для усіх таблиць

```
    if(index==1)
    {
        Calculate_A();
        Calculate_B();
        Calculate_C();
```

Збереження таблиць у файлі Excel

```
        tabexl->SaveTable();
    }
}
```

Метод формування таблиць типу А

```
void MainWindow::Calculate_A()
{
    long i;
```

Створюється список номерів таблиць

```
    QStringList lst={"2", "4", "5", "6", "24", "25", "26",
"27", "29", "30"};
    for(i=0; i<lst.size(); i++)
    {
```

Виконання запиту до бази даних для поточної таблиці

```
        Table_Data_t d;
        d.table_num = lst[i];
```

```
    //          d.data = DSTU_data->GetTable(d.table_num, city_idx,
&d.table_name);
            d.data = DSTU_data->GetTable_A(d.table_num, Lat, Lon,
&d.table_name);
```

Створення таблиці в інтерфейсі програми

```
            CreateTable(&d, ui->tab_TempHum);
            ui->gridLayout_TH->addWidget(d.label);
            ui->gridLayout_TH->addWidget(d.table);
            TableData.append(d);
        }
    }
```

Метод формування таблиць типу B

```
void MainWindow::Calculate_B()
{
    long i;
```

Створюється список номерів таблиць

```
    QStringList lst={"7", "8", "9", "19"};
    for(i=0; i<lst.size(); i++)
    {
```

Виконання запиту до бази даних для поточної таблиці з урахуванням режиму апроксимації

```
            Table_Data_t d;
            d.table_num = lst[i];
            if(Mode)
```

```
                d.data  =  DSTU_data->GetTable_B_near(d.table_num,
Lat, Lon, &d.table_name);
            else
                d.data  =  DSTU_data->GetTable_B_appr(d.table_num,
Lat, Lon, &d.table_name);
```

Створення таблиці в інтерфейсі програми

```
            CreateTable(&d, ui->tab_TempHum);
            ui->gridLayout_TH->addWidget(d.label);
            ui->gridLayout_TH->addWidget(d.table);
            TableData.append(d);
        }
    }
    void MainWindow::Calculate_C()
    {
        long i, j;
```

Створюється список номерів таблиць

```
        QVector< QVector<QVariant> > data_res;
        QStringList lst={"10", "14", "18", "20"};
        for(i=0; i<lst.size(); i++)
        {
```

Виконання запиту до бази даних для поточної таблиці

```
            Table_Data_t d;
            d.table_num = lst[i];
            d.data = DSTU_data->GetTable_C(d.table_num, Lat, Lon,
&d.table_name, &data_res);
```

Створення таблиці в інтерфейсі програми

```
            CreateTableArr(&d, ui->tab_TempHum, &data_res);
            ui->gridLayout_TH->addWidget(d.label);
            ui->gridLayout_TH->addWidget(d.table);
            TableData.append(d);
            for(j=0; j<data_res.size(); j++)
            {
                data_res[j].clear();
            }
            data_res.clear();
        }
    }
```

Метод створення відображення таблиці з одним рядком в інтерфейсі

```
    void   MainWindow::CreateTable(   Table_Data_t*   d,   QWidget
*parent )
    {
        long i;
```

Стровення назви таблиці

```
        QString str = "Табл." + d->table_num+". "+d->table_name;
        d->label = new QLabel(str, parent);
```

Встановення розмірів таблиці

```
        d->table = new QTableWidget(parent);
        d->table->setMaximumHeight(4*22);
        d->table->setMinimumHeight(4*22);
        d->table->setSizePolicy(QSizePolicy::MinimumExpanding,
QSizePolicy::MinimumExpanding);
        d->table->setRowCount(3);
        d->table->setColumnCount( d->data.size() );
```

Додавання до таблиці «шапки»

```
QHeaderView *hwh=d->table->horizontalHeader();
hwh->setVisible(true);
hwh->setSectionResizeMode(QHeaderView::Interactive);
QHeaderView *hwv=d->table->verticalHeader();
hwv->setVisible(false);
```

Запис назви таблиці в файл Excel

```
tabexl->write( tabexl->row_cnt, 1, QVariant(str) );
tabexl->row_cnt++;
```

Встановлення висоти рядків таблиці

```
for(i=0; i<d->table->rowCount(); i++)
{
    d->table->setRowHeight(i, 20);
}
}
```

Також до таблиці додається заголовок з назвами стовпців.

Метод створення відображення таблиці з декількома рядками в інтерфейсі

```
void MainWindow::CreateTableArr(Table_Data_t* d, QWidget
*parent, QVector< QVector<QVariant> > *data_arr )
{   long i;
```

Стровення назви таблиці

```
        QString str = "Табл." + d->table_num+". "+d->table_name;
        d->label = new QLabel(str, parent);
        d->table = new QTableWidget(parent);
```

Встановення розмірів таблиці

```
        d->table->setMaximumHeight((data_arr->size()+3)*22);
        d->table->setMinimumHeight((data_arr->size()+3)*22);
        d->table->setSizePolicy(QSizePolicy::MinimumExpanding,
QSizePolicy::MinimumExpanding);
        d->table->setRowCount((data_arr->size()+2));
        d->table->setColumnCount( (*data_arr)[0].size() );
        QHeaderView *hwh=d->table->horizontalHeader();
        hwh->setVisible(true);
        hwh->setSectionResizeMode(QHeaderView::Interactive);
        QHeaderView *hwv=d->table->verticalHeader();
        hwv->setVisible(false);
        tabexl->write( tabexl->row_cnt, 1, str );
```

Встановлення висоти рядків таблиці

```
        for(i=0; i<d->table->rowCount(); i++)
        {
            d->table->setRowHeight(i, 20);
        }
    }
```

Також до таблиці додається заголовок з назвами стовпців.

3.2. Модулі взаємодії з базою даних

Інтерфейс з базою даних SQLite виконується за допомогою класу DB_data, що реалізує алгоритми відкриття файлу БД, створення запросів до БД та інше. Також при створенні та отриманні результату запросу виконується апроксимація результатів при необхідності.

Для отримання координат міст, використовуєтося допоміжна структура

```
typedef struct
{
    long Idx; // індекс міста
    double Lat, Lon; // широта та довгота
} Idx_Lat_t;
```

Конструктор класу DB_data забезпечує створення об’єкту та відкриття файлу бази даних

```
DB_data::DB_data()
{
    db = new QSqlDatabase;
    *db = QSqlDatabase::addDatabase( "QSQLITE" );
    db->setDatabaseName("..\\TableDSTU.db");
    if( !db->open() )
    {
        QTextStream(stdout) << "Can't open DB " << db-
>lastError().text();
        return;
    }}
```

Запит таблиць типу A реалізовано наступним методом. Вхідними даними є:

`num_table` – номер таблиці;

`city_Lat` – широта точки, для якої виконується розрахунок;

`city_Lon` – довгота точки, для якої виконується розрахунок;

table_name – вказівник на назву таблиці.

Метод повертає назву таблиці та масив структур TableItem_t

```
QVector<TableItem_t>  DB_data::GetTable_A(QString  num_table,
double city_Lat, double city_Lon, QString *table_name)
{
```

Створення запиту для отримання назви таблиці

```
    QSqlQuery query( *db );
    QString str;
    str=QString("SELECT    Name    FROM    TableName    WHERE
TableNumber=%1").arg(num_table);
    QVector<QVariant> res1 = get_Record( str );
    if(table_name)
        (*table_name) = res1[0].toString();
```

Створення запиту для отримання назв стовпців таблиці

```
    QVector<TableItem_t> table;
    table = get_TableItem(num_table);
```

Створення запиту на отримання відстаней між поточним місцем та координатами міст рядків таблиці БД

```
    QVector<Idx_Lat_t> table1; // масив координат міст
    str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1,                        City                        WHERE
Table%1.CityIndex=City.CityIndex").arg(num_table);
    if( !query.exec(str) )
    { // помилка запиту
        QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
```

```
        }
        else
        {// запит виконано
            QSqlRecord rec = query.record();
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table1.append(d);
            }
        }
```

Обчислення індексів міст з мінімальною відстанью до поточної точки

```
        long i;
        long idx_min1=0;
        double  val_min  =  Distance_2Point(city_Lat,  city_Lon,
table1[0].Lat, table1[0].Lon);
        for(i=1; i<table1.size(); i++)
        {
            double  d    =  Distance_2Point(city_Lat,  city_Lon,
table1[i].Lat, table1[i].Lon);
            if(val_min > d)
            {               val_min  =  d;          idx_min1  =
i;//table1[i].Idx; }
        }
```

Запит основних даних для даної таблиці з БД

```
        str = QString("SELECT ");
        for(i=0; i<table.size(); i++)
```

```
            str += table[i].Item + ((i==table.size()-1) ? "":",");
        str += QString(" FROM Table%1 WHERE Idx=%2").arg( num_table ).arg( table1[idx_min1].Idx );
        QVector<QVariant> res = get_Record(str);
        for(int i=0; i<res.size(); i++)
        {
            table[i].Data = res[i];
        }
        return table;
    }
```

Запит таблиць типу B реалізовано наступним методом. Вхідними даними є:

`num_table` – номер таблиці;

`city_Lat` – широта точки, для якої виконується розрахунок;

`city_Lon` – довгота точки, для якої виконується розрахунок;

`table_name` – вказівник на назву таблиці.

Метод повертає назву таблиці та масив структур TableItem_t

```
    QVector<TableItem_t> DB_data::GetTable_B_appr(QString num_table, double city_Lat, double city_Lon, QString *table_name)
    {
        QVector<Idx_Lat_t> table1; // масив міст, широта яких менша за широту поточного міста
        QVector<Idx_Lat_t> table2; // масив міст, широта яких більша за широту поточного міста
        QSqlQuery query( *db );
        QString str;
```

Створення запиту для отримання назви таблиці

```
        str=QString("SELECT Name FROM TableName WHERE
TableNumber=%1").arg(num_table);
        QVector<QVariant> res_name = get_Record( str );
        if(table_name)
            (*table_name) = res_name[0].toString();
```

Створення запиту на отримання індексів міст, широта яких менша за широту поточного місця

```
        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon FROM
Table%1, City WHERE Table%1.CityIndex=City.CityIndex AND
City.Lat<%2").arg(num_table).arg(city_Lat);
        if( !query.exec(str) )
        {
            QTextStream(stdout) << "get_City_data DB " << db-
>lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table1.append(d);
            }
        }
```

Створення запиту на отримання індексів міст, широта яких більша за широту поточного місця

```
        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1,   City   WHERE   Table%1.CityIndex=City.CityIndex   AND
City.Lat>=%2").arg(num_table).arg(city_Lat);
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table2.append(d);
            }
        }
```

Обчислення індексів міст з мінімальною відстанью до поточної точки, широта яких менше широти поточної точки

```
        long i;
        long idx_min1=0, idx_min2=0;
        double  val_min  =  Distance_2Point(city_Lat,  city_Lon,
table1[0].Lat, table1[0].Lon);
        for(i=1; i<table1.size(); i++)
        {
            double  d    =  Distance_2Point(city_Lat,  city_Lon,
table1[i].Lat, table1[i].Lon);
            if(val_min > d)
            {
                val_min = d;
```

```
                idx_min1 = i
            }
        }
```

Обчислення індексів міст з мінімальною відстанью до поточної точки, широта яких більше широти поточної точки

```
        val_min     =     Distance_2Point(city_Lat,     city_Lon,
table2[0].Lat, table2[0].Lon);
        for(i=1; i<table2.size(); i++)
        {
            double  d    =  Distance_2Point(city_Lat,  city_Lon,
table2[i].Lat, table2[i].Lon);
            if(val_min > d)
            {
                val_min = d;
                idx_min2 = i;//table2[i].Idx;
            }
        }
```

Обчислення відстаней від поточної точки до найближчих точок з меншою та більшою широтою.

```
        double d1, d2, koef;
        d1       =       Distance_2Point(city_Lat,       city_Lon,
table1[idx_min1].Lat, table1[idx_min1].Lon);
        d2       =       Distance_2Point(city_Lat,       city_Lon,
table2[idx_min2].Lat, table2[idx_min2].Lon);
```

Обчислення коефіцієнту масштабування

```
        koef = d1/(d1+d2);
```

Створення запиту для отримання назв стовпців таблиці

```
    QVector<TableItem_t> table;
    table = get_TableItem(num_table);
```

Запит основних даних для даної таблиці з БД для двох міст

```
    QVector<QVariant> res;
    str = QString("SELECT ");
    for(i=0; i<table.size(); i++)
        str +=table[i].Item + ((i==table.size()-1) ? "":",");
    str   +=   QString("   FROM   Table%1   WHERE   Idx=").
arg(num_table);
    QVector<QVariant> res1 = get_Record( str + QString("%1").
arg(table1[idx_min1].Idx) );
    QVector<QVariant> res2 = get_Record( str + QString("%1").
arg(table2[idx_min2].Idx) );
```

Обчислення апроксимованих значеннь таблиці

```
    for(i=0; i<res1.size(); i++)
    {
        double delta;
        delta = res2[i].toDouble() - res1[i].toDouble();
        double d = res1[i].toDouble() + delta*koef ;
        table[i].Data = QVariant(d);
    }
    return table;
}
```

Запит таблиць типу С реалізовано наступним методом. Вхідними даними є:

`num_table` – номер таблиці;

city_Lat – широта точки, для якої виконується розрахунок;

city_Lon – довгота точки, для якої виконується розрахунок;

table_name – вказівник на назву таблиці;

data_res – масив рядків таблиці.

Метод повертає назву таблиці та масив структур TableItem_t.

```
QVector<TableItem_t> DB_data::GetTable_C(QString num_table, double city_Lat, double city_Lon, QString *table_name, QVector< QVector<QVariant> > *data_res)
{
    QString str;
    long i;
```

Створення запиту для отримання назви таблиці

```
    QSqlQuery query( *db );
    str=QString("SELECT Name FROM TableName WHERE TableNumber=%1").arg(num_table);
    QVector<QVariant> res_name = get_Record( str );
    if(table_name)
        (*table_name) = res_name[0].toString();
```

Визначення інтервалу широт, до яких належить поточна точка

```
    long Lat1, Lat2;
    if(city_Lat < 44.0) { Lat1 = 44; Lat2 = 46; }
    else if(city_Lat >= 44.0 && city_Lat < 46.0)
    { Lat1 = 44; Lat2 = 46; }
    else if(city_Lat >= 46.0 && city_Lat < 48.0)
    { Lat1 = 46; Lat2 = 48; }
    else if(city_Lat >= 48.0 && city_Lat < 50.0)
    { Lat1 = 48; Lat2 = 50; }
    else
```

```
        { Lat1 = 48; Lat2 = 50; }
```

Отримання назв стовпців таблиці

```
        QVector<TableItem_t> table1;
        str=QString("SELECT Idx,Item,Name1,Name2,Name3 FROM Name
WHERE TableNumber=%1 AND (INSTR( Item, 'Lat%2') > 0 OR
Item='Interval')").arg(num_table).arg(Lat1);
        if( !query.exec(str) )
        {QTextStream(stdout) << "get_City_data DB " << db-
>lastError().text();}
        else
        {
```

Перетворення результату запита для першої таблиці

```
            QSqlRecord rec = query.record();
            while(query.next()) {
                TableItem_t it;
                it.Idx = query.value(0).toInt();
                it.Item = query.value(1).toString();
                it.Name1 = query.value(2).toString();
                it.Name2 = query.value(3).toString();
                it.Name3 = query.value(4).toString();
                table1.append(it);
            }
        }
```

Створення запиту для другої таблиці

```
        QVector<TableItem_t> table2;
        str=QString("SELECT Idx,Item,Name1,Name2,Name3 FROM Name
WHERE TableNumber=%1 AND (INSTR( Item, 'Lat%2') > 0 OR
Item='Interval')").arg(num_table).arg(Lat2);
        if( !query.exec(str) )
```

```
        {QTextStream(stdout) << "get_City_data DB " << db->
lastError().text }
        else
        {
```

Перетворення результату запита для першої таблиці

```
            QSqlRecord rec = query.record();
            while(query.next()) {
                TableItem_t it;
                it.Idx = query.value(0).toInt();
                it.Item = query.value(1).toString();
                it.Name1 = query.value(2).toString();
                it.Name2 = query.value(3).toString();
                it.Name3 = query.value(4).toString();
                table2.append(it);
            }
        }
```

Визначення відстаней по дузі мерідіана, який проходить через поточну точку та коефіцієнту апроксимації

```
        double d1, d2, koef;
        d1 = Distance_2Point(city_Lat, city_Lon, Lat1, city_Lon);
        d2 = Distance_2Point(city_Lat, city_Lon, Lat2, city_Lon);
        koef = d1/(d1+d2);
```

Запит на отримання даних для першої точки

```
        str = QString("SELECT ");
        for(i=0; i<table1.size(); i++)
            str +=table1[i].Item+((i==table1.size()-1) ? "":",");
        str += QString(" FROM Table%1").arg(num_table);
        QVector<QVariant> res1 = get_Record( str );
```

Запит на отримання даних для другої точки

```
str = QString("SELECT ");
for(i=0; i<table2.size(); i++)
    str +=table2[i].Item+((i==table2.size()-1) ? "":",");
str += QString(" FROM Table%1").arg(num_table);
QVector<QVariant> res2 = get_Record( str );
```

Апроксимація даних таблиці

```
long j;
if( data_res )
{
    for(i=0; i<res1.size(); i+=table1.size())
    {
        QVector<QVariant> t_sub;
        int interval = res1[i].toInt();
```

Додавання спецфічних рядків

```
        if(interval == 24)
            t_sub.append( QString("Усього") );
        else if(interval == 25)
            t_sub.append( QString("Середня")  );
        else
            t_sub.append( interval );
```

Обчислення значення для заданої точки

```
        double delta;
        for(j=1; j<table1.size(); j++)
        {
```

```
                    delta  =  res2[i+j].toDouble()  -  res1[i+j].
toDouble();
                   double d = res1[i+j].toDouble() + delta*koef ;
                    t_sub.append( QVariant(d) );
                }
                data_res->append( t_sub );
            }
        }
        return table1;
    }
```

Запит таблиць типу B з пошуком найближчої точки реалізовано наступним методом. Вхідними даними є:

num_table – номер таблиці;

city_Lat – широта точки, для якої виконується розрахунок;

city_Lon – довгота точки, для якої виконується розрахунок;

table_name – вказівник на назву таблиці.

Метод повертає назву таблиці та масив структур TableItem_t

```
    QVector<TableItem_t>          DB_data::GetTable_B_near(QString
num_table, double city_Lat, double city_Lon, QString *table_name)
    {
        QVector<Idx_Lat_t> table1;
        QVector<Idx_Lat_t> table2;
        QSqlQuery query( *db );
        QString str;
```

Створення запиту для отримання назви таблиці

```
        str=QString("SELECT    Name    FROM    TableName    WHERE
TableNumber=%1").arg(num_table);
        QVector<QVariant> res_name = get_Record( str );
        if(table_name)
```

```
            (*table_name) = res_name[0].toString();
```

Створення запиту на отримання індексів міст

```
        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1, City WHERE Table%1.CityIndex =City"). arg( num_table ).
arg(city_Lat);
        if( !query.exec(str) )
        {
            QTextStream(stdout) << "get_City_data DB " << db->
lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table1.append(d);
            }
        }
```

Пошук індекса міста, до якого відстань буде мінімальною

```
        long i;
        long idx_min1=0;
        //idx_min1 = table1[0].Idx;
        double  val_min  =  Distance_2Point(city_Lat,  city_Lon,
table1[0].Lat, table1[0].Lon);
        for(i=1; i<table1.size(); i++)
        {
```

```
            double d = Distance_2Point(city_Lat, city_Lon,
table1[i].Lat, table1[i].Lon);
            if(val_min > d)
            {
                val_min = d;
                idx_min1 = i;
            }
        }
```

Створення запиту для отримання назв стовпців таблиці

```
        QVector<TableItem_t> table;
        table = get_TableItem(num_table);
```

Запит основних даних для даної таблиці з БД знайденого міста

```
        QVector<QVariant> res;
        str = QString("SELECT ");
        for(i=0; i<table.size(); i++)
            str += table[i].Item + ((i==table.size()-1) ?
"":",");
        str += QString(" FROM Table%1 WHERE
Idx=").arg(num_table);//.arg(table1[idx_min1].Idx);
        QVector<QVariant> res1 = get_Record( str +
QString("%1").arg(table1[idx_min1].Idx) );
```

Перетворення даних

```
        for(i=0; i<res1.size(); i++)
        {
            table[i].Data = QVariant( res1[i].toDouble() );
        }
        return table;
    }
```

Отримання назв стовпців виконується за допомогою методу get_TableItem. Вхідними даними якого є:

num_table – номер таблиці;

Метод повертає масив елементів TableItem_t.

```
QVector<TableItem_t> DB_data::get_TableItem( QString num_table )
{
    QString str;
    QVector<TableItem_t> table;
```

Створення запиту

```
    str=QString("SELECT Idx,Item,Name1,Name2,Name3 FROM Name WHERE TableNumber=%1").arg(num_table);
```

Перетворення результату запиту

```
    QSqlQuery query( *db );
    if( !query.exec(str) )
    {
        QTextStream(stdout) << "get_City_data DB " << db->lastError().text();
    }
    else
    {
        QSqlRecord rec = query.record();
        while(query.next())
        {
            TableItem_t it;
            it.Idx = query.value(0).toInt();
            it.Item = query.value(1).toString();
```

```
                it.Name1 = query.value(2).toString();
                it.Name2 = query.value(3).toString();
                it.Name3 = query.value(4).toString();
                table.append(it);
            }
        }
        return table;
    }
```

Формування запиту до БД у загальному виді виконується методом get_Record. Вхідними даними якого є:

req – сформований рядок SQL запита;

Метод повертає масив елементів QVariant.

```
    QVector<QVariant> DB_data::get_Record(QString req)
    {
        QVector<QVariant> res;
```

Створення запиту до БД

```
        QSqlQuery query( *db );
        if( !query.exec(req) )
        {
            QTextStream(stdout) << "get_City_data DB " << db-
>lastError().text();
        }
        else
        {
```

Переміщення результату до вихідного масиву

```
            QSqlRecord rec = query.record();
```

```
            while(query.next())
            {
                for(int i=0; i<rec.count(); i++)
                {
                    res.append( query.value(i) );
                }
            }
        }
        return res;
    }
```

Метод обчислення відстані між двома точками з урахуванням формул з розділу 2. Вхідні дані:

lat1 – широта першої точки;

lon1 – довгота першої точки;

lat2 – широта другої точки;

lon2 – довгота другої точки;

alpha – вказівник на кут між двома точками.

```
    double DB_data::Distance_2Point(double lat1, double lon1,
double lat2, double lon2, double *alpha)
    {
```

Визначення констант перетворення

```
        const double Earth_a = 6378137.0; // Основна піввісь
        const double Earth_b = 6356752.314245; // Неосновна
піввісь
        const double Earth_e2 = 0.006739496742337; // квадрат
ексцентричності елепсоїду
        const double Earth_f = 0.003352810664747;
        const double GRAD2RAD = M_PI/180.0;
```

Розрахунок допоміжних змінних

```
        double fdLambda = (lon1 - lon2) * GRAD2RAD;
        double fdPhi = (lat1 - lat2) * GRAD2RAD;
        double fPhimean = ((lat1 + lat2) / 2.0) * GRAD2RAD;
        double fTemp = 1 - Earth_e2 * (pow(sin(fPhimean), 2));
        double fRho = (Earth_a * (1 - Earth_e2)) / pow(fTemp,
1.5);
        double fNu = Earth_a / (sqrt(1 - Earth_e2 *
(sin(fPhimean) * sin(fPhimean))));
```

Визначення кута між точками

```
        double fz = sqrt(pow(sin(fdPhi / 2.0), 2)+cos(lat2 *
GRAD2RAD) * cos(lat1 * GRAD2RAD)*pow(sin(fdLambda / 2.0), 2));
        if(fz>1) fz=1;
        else if(fz<-1) fz=-1;
        fz = 2 * asin(fz);
        if(alpha!=nullptr) {*alpha = fz; }
```

Визначення радіусу в заданих точках

```
        double fAlpha, sinfz=sin(fz);
        if( sinfz!=0.0 ) fAlpha = cos(lat2 * GRAD2RAD) *
sin(fdLambda) / sin(fz);
        else fAlpha = 0;
        if(fAlpha>1) fAlpha=1;
        else if(fAlpha<-1) fAlpha=-1;
        fAlpha = asin(fAlpha);
        double fR = (fRho * fNu) / ((fRho * pow(sin(fAlpha), 2))
+ (fNu *pow(cos(fAlpha), 2)));
```

Розрахунок відстані між точками

```
    return (fz * fR);
}
```

3.3. Керівництво користувача

Розроблена програма базується на бібліотеці Qt. Для роботи програми необхідна наявність наступних бібліотек:

```
platforms/qminimal.dll
platforms/qwindows.dll
libgcc_s_seh-1.dll
libstdc++-6.dll
libwinpthread-1.dll
Qt5Core.dll
Qt5Gui.dll
Qt5SerialBus.dll
Qt5SerialPort.dll
Qt5Widgets.dll
```

Ці бібліотеки розташовуються в одному каталозі з програмою.

Головне вікно програми наведено на рис. 3.1.

В програмі передбачено два режими роботи:

- розрахунок кліматичних параметрів для найближчої точки з таблиці;
- розрахунок кліматичних параметрів з апроксимацією.

На вкладці «Вхідні дані» вводиться широта та довгота точки розрахунку иа обирається режим роботи.

На вкладці «Результат» виводиться таблиця з розрахованими параметрами.

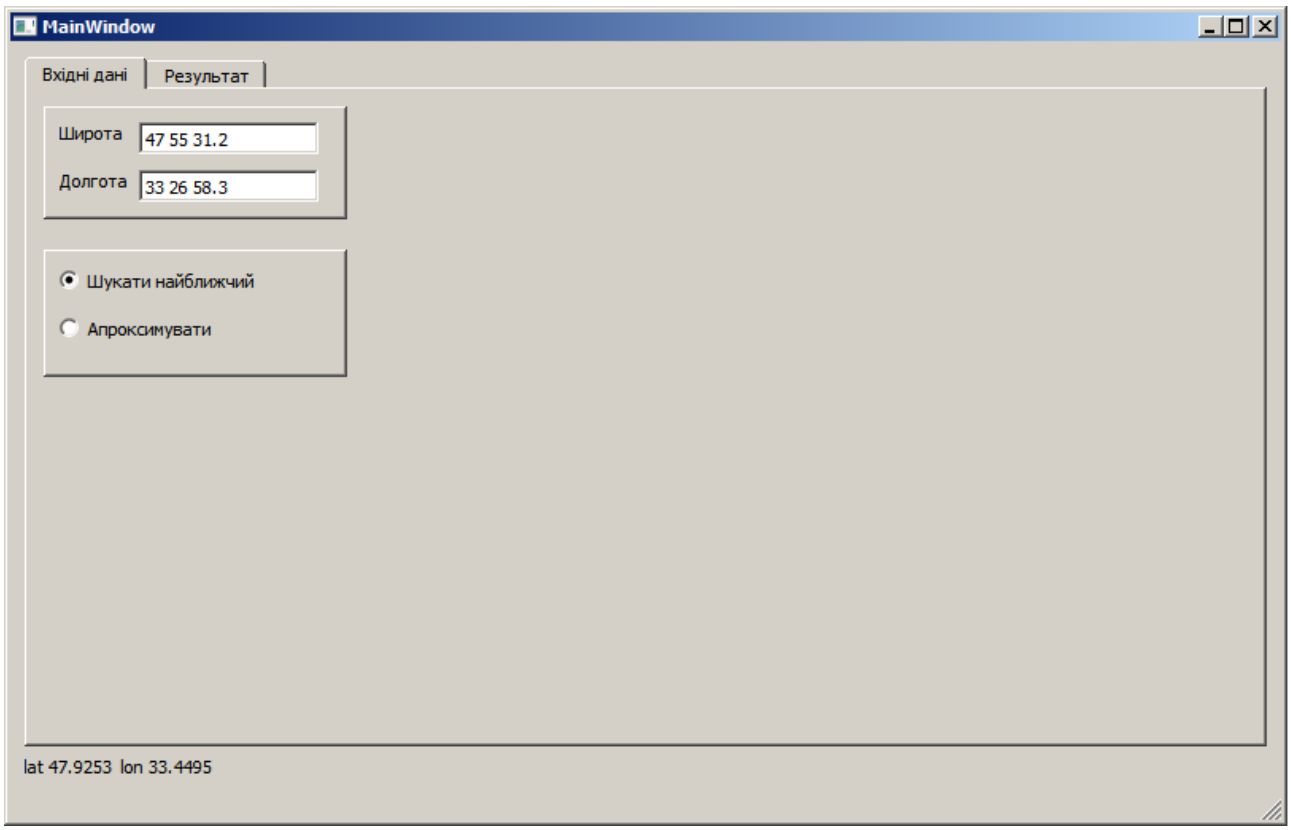

Рис. 3.1 – Головне вікно програми

Так як таблиці розділені на три типа A, B, C, то і відображення їх також відрізняється. Так таблиці A, B відображаються у вигляді однордкових таблиць (рис. 3.2), а таблиці типу C – у вигляді багаторядкових таблиць (рис. 3.3).

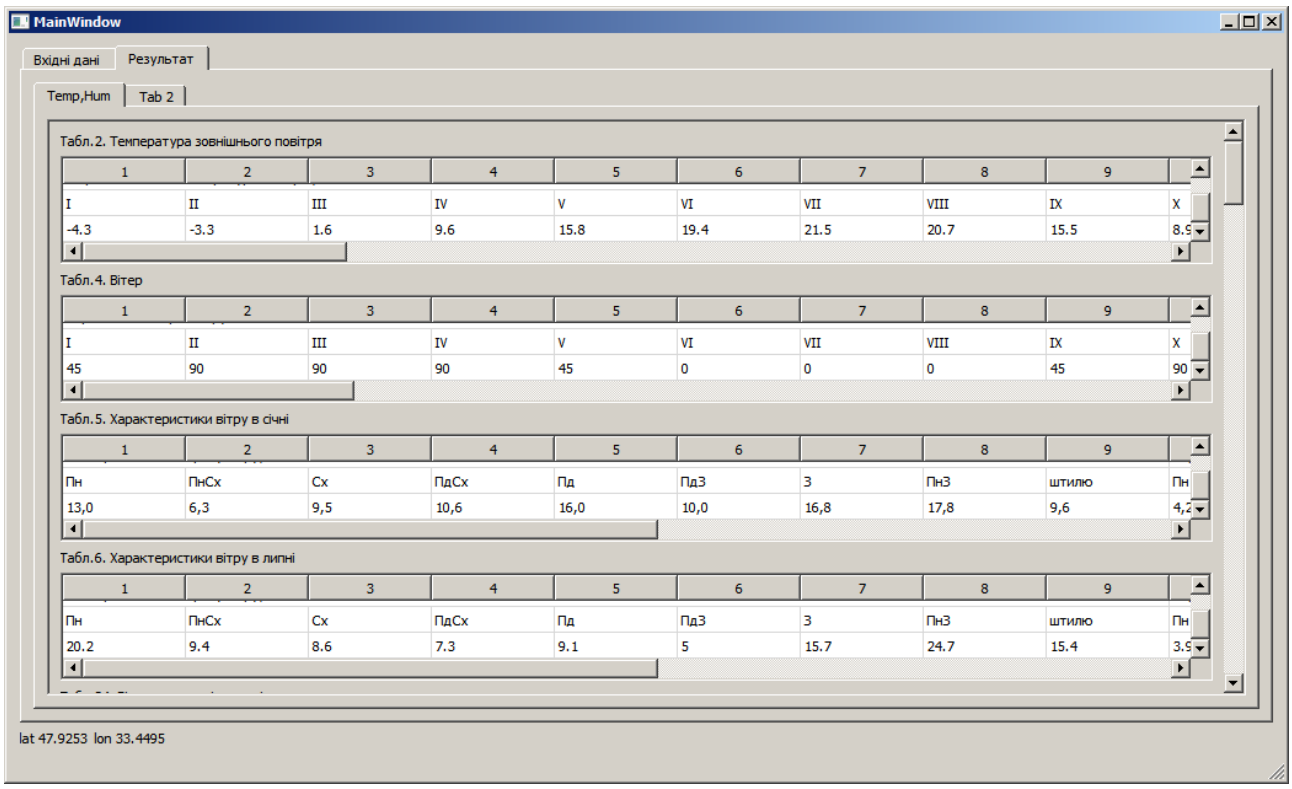

Рис. 3.2 – Вікно з таблицею результатів у вигляді однорядкових даних

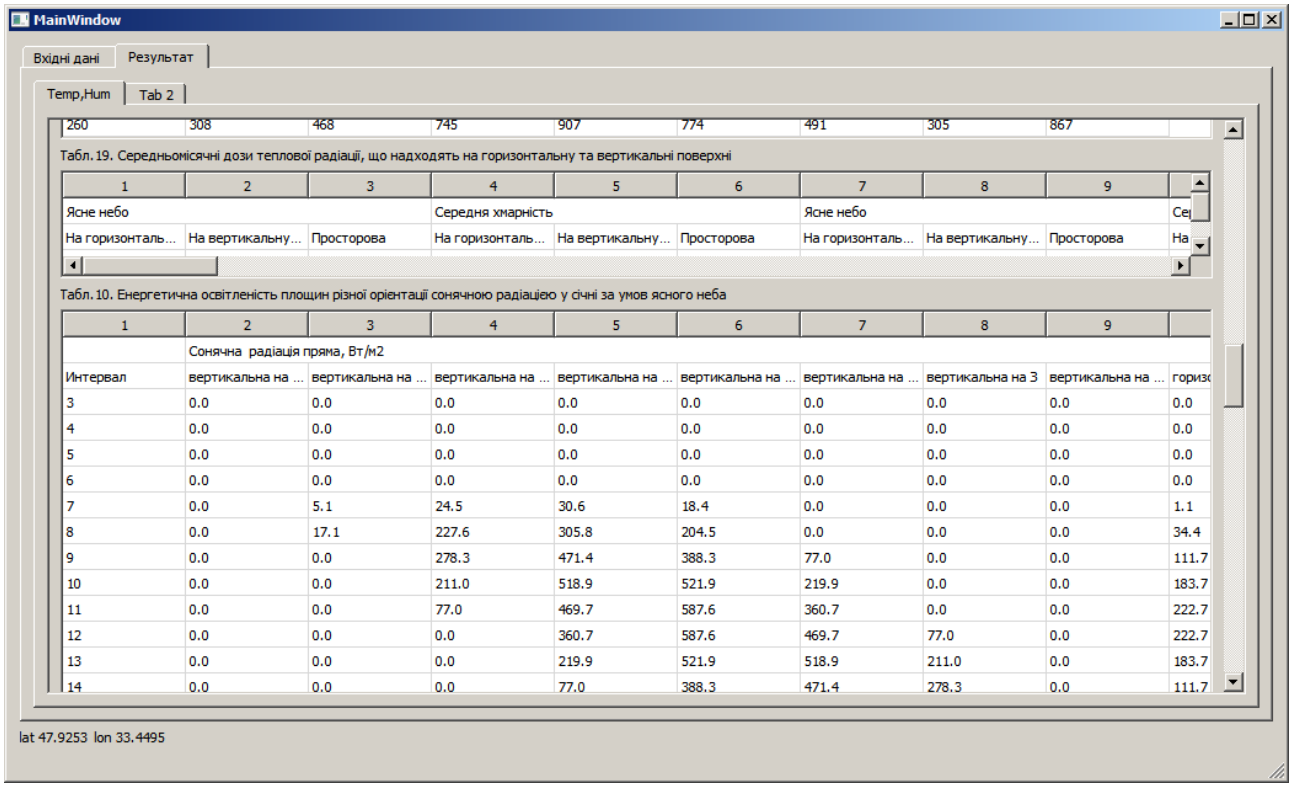

Рис. 3.3 – Вікно з таблицею результатів у вигляді багаторядкових даних

Результат роботи програми також зберігається у файлі формату Excel. На рис. 3.4 наведено приклад таблиць однорядкових даних, а на рис. 3.5. – приклад багаторядкових таблиць.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1		Табл.2. Температура зовнішнього повітря																		
2	Середня місячна температура повітря ,?C												Середня місячна середня добова амплітуда температури ,?C							
3	-4.3	-3.3	1.6	9.6	15.8	19.4	21.5	20.7	15.5	8.9	2.7	-2	1000	1000	1000	1000	1000	1000	1000	1000
4																				
5		Табл.4. Вітер																		
6	Переважний напрям вітру												Переважний напрям вітру, його повторюваність, %							
7	45	90	90	90	45	0	0	0	45	90	90	90/270	17	24	20	18	19	19	25	24
8																				
9		Табл.5. Характеристики вітру в січні																		
10	Повторюваність напряму вітру, %									Середня швидкість вітру, м/с										
11	13,0	6,3	9,5	10,6	16,0	10,0	16,8	17,8	9,6	4,2	3,6	4,0	4,3	4,5	4,2	4,3	4,2			
12																				
13		Табл.6. Характеристики вітру в липні																		
14	Повторюваність напряму вітру, %									Середня швидкість вітру, м/с										
15	20.2	9.4	8.6	7.3	9.1	5	15.7	24.7	15.4	3.9	3.8	3.7	3.4	3.9	4	3.8	3.6			
16																				
17		Табл.24. Відносна вологість повітря																		
18	Середня місячна відносна вологість, %												Середня	Середня місячна середня добова амплітуда відносної вологості, %						
19	85	84	79	66	62	65	63	61	66	75	85	87	73	1000	1000	1000	1000	1000	1000	1000
20																				
21		Табл.25. Фактор мутності атмосфери (при оптичній масі атмосфери m=2)																		
22	Середній по місяцях фактор мутності												Середній	Середній по місяцях середня добова амплітуда фактора мутності						
23	2,50	2,80	3,20	3,70	3,90	4,25	4,30	4,25	3,50	2,90	2,70	2,70	3,39	0,40	0,20	0,15	0,20	0,50	0,60	0,60
24																				
25		Табл.26. Хмарність. Кількість хмарності																		
26	Кількість хмарності загальна, бали													Кількість хмарності нижня, бали						
27	7,4	7,3	6,9	6,4	5,6	5,4	4,7	4,3	4,6	5,6	7,7	8,0	6,2	5,6	5,2	4,4	3,2	2,5	2,6	2,4

Рис. 3.4 – Приклад Excel-файлу з результатами у вигляді однорядкових даних

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
46		Табл.8. Середньомісячні дози сонячної радіації, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за середніх умов хмарності																		
47	Сонячна радіація пряма, МДж/м2																			
48	I									II									III	
49	0	0	14	57	90	66	21	1	28	0	3	28	77	110	88	35	3	55	0	20
50																				
51		Табл.9. Доза сумарної сонячної радіації за опалювальний період, що надходить на горизонтальну та вертикальну поверхні різної орієнтації за середніх умов хмарності																		
52	Сумарна сонячна радіація за опалювальний період, МДж/м2																			
53	260	308	468	745	907	774	491	305	867											
54																				
55		Табл.19. Середньомісячні дози теплової радіації, що надходять на горизонтальну та вертикальні поверхні																		
56	Ясне небо			Середня хмарність			Ясне небо			Середня хмарність			Ясне небо			Середня хмарність			Ясне небо	
57	574	706	673	685	751	734	528	648	618	621	686	670	640	778	743	729	814	793	716	860
58																				
59		Сонячна радіація пряма, Вт/м2									Сонячна радіація розсіяна, Вт/м2									
60	Интервал	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	горизонта	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	вертикалі	горизонтальна	
61	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
62	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
63	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
64	6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.1	2.0	1.0	1.0	1.0	1.0	1.0	
65	7	0.0	5.1	24.5	30.6	18.4	0.0	0.0	0.0	1.1	10.1	13.1	20.1	22.1	16.1	11.0	10.0	10.1	15.1	
66	8	0.0	17.1	227.6	305.8	204.5	0.0	0.0	0.0	34.4	29.1	36.2	59.3	70.3	56.3	34.1	29.1	29.1	40.2	
67	9	0.0	0.0	278.3	471.4	388.3	77.0	0.0	0.0	111.7	42.2	47.2	79.4	104.5	93.4	57.3	42.2	42.2	61.3	
68	10	0.0	0.0	211.0	518.9	521.9	219.9	0.0	0.0	183.7	50.2	53.2	82.3	121.4	121.4	83.3	54.2	50.2	77.3	
69	11	0.0	0.0	77.0	469.7	587.6	360.7	0.0	0.0	222.7	54.2	55.2	74.3	119.4	135.4	105.3	63.2	54.2	84.3	
70	12	0.0	0.0	0.0	360.7	587.6	469.7	77.0	0.0	222.7	54.2	54.2	63.2	105.3	135.4	119.4	74.3	55.2	84.3	
71	13	0.0	0.0	0.0	219.9	521.9	518.9	211.0	0.0	183.7	50.2	50.2	54.2	83.3	121.4	121.4	82.3	53.2	77.3	
72	14	0.0	0.0	0.0	77.0	388.3	471.4	278.3	0.0	111.7	42.2	42.2	42.2	57.3	93.4	104.5	79.4	47.2	61.3	

Рис. 3.5 – Приклад Excel-файлу з результатами у вигляді багаторядкових даних

Таким чином програма дозволяє розрахувати нормативні кліматичні параметри для будь якої точки України.

ВИСНОВКИ

Будівлі постійно піддаються впливу кількох кліматичних та екологічних факторів. Від вітру, сонячного світла, температури, дощу та інших факторів будівлі по всьому світу унікальним чином взаємодіють з різними елементами навколишнього клімату. Через це проектування будівель і методи будівництва відрізняються від одного місця до іншого, щоб відповідати різним викликам.

Будівлі, дороги тощо мають бути спроектовані відповідно до майбутніх кліматичних умов. Будівлі можуть бути вразливими до зміни клімату.

Дані, закладені у нормативних документах з будівельної кліматології, є вихідними параметрами вирішення завдань проектування об'єктів. Однією з проблем при вирішенні оптимізаційних завдань щодо енергоефективності будівель є нестача кліматичних параметрів, закладених у нормах. Заповнення необхідної кліматичної інформації може бути здійснено геометричними методами на основі аналізу фізичних закономірностей зміни в часі та просторі кліматичних параметрів. В Україні запроваджено норми щодо будівельної кліматології ДСТУ-Н Б В.1.1–27:2011 «Будівельна кліматологія». Ці норми включають широкий набір кліматичних параметрів та мають 10 основних розділів: архітектурно-будівельне кліматичне районування території України, температура зовнішнього повітря, вітер, сонячна радіація, теплова радіація атмосфери та Землі, вологість повітря, фактор каламутності атмосфери, хмарність, опади та сніговий покрив, природне освітлення.

На основі аналізу наведених даних в ДСТУ, розроблено алгоритми апроксимації кліматичних даних. В ДСТУ виділено три основних типу таблиць:

- таблиці з даними для певного міста для кожного місяця та для року вцілому (тип А);
- таблиці з даними для певного міста для кожного місяця та для року вцілому для різних орієнтації поверхні (тип В);
- таблиці з даними для певної широти місця (тип С).

Розробка програми апроксимації кліматичних даних виконано з використанням крос-платформне повне інтегроване середовище розробки (IDE) Qt Creator. Це середовище дозволяє розролювати додатків для кількох платформ настільних комп’ютерів, вбудованих і мобільних пристроїв, таких як Android та iOS.

Для зберігання даних таблиць ДСТУ використвано SQLite. Бібліотека SQLite статично скомпонована з додатком. SQLite отримує прямий доступ до своїх файлів. Для роботи SQLite не потрібен окремий серверний процес або система. Повна база даних SQLite зберігається в одному файлі.

В програмі передбачено два режими роботи: розрахунок кліматичних параметрів для найближчої точки з таблиці; розрахунок кліматичних параметрів з апроксимацією.

Таким чином програма дозволяє розрахувати нормативні кліматичні параметри для будь якої точки України.

РЕКОМЕНДАЦІЇ

Розроблена програма скомпільована для роботи у середовище Windows, але може бути достатньо легко перенесена на платформу Android. Такий перехід можливий, завдяки використанню бібліотек та оточення Qt, яке дозволяє перекомпілювати додаток для обраної платформи.

Розроблена програма має потенційні можливості для визначення кліматичних зон, та умов будівництва.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Геометрична комп’ютеризована модель «Atmospheric Radiacion» для енергоефективного будівництва / О.В. Сергейчук // Енергозбереження в будівництві та архітектурі. – К. : КНУБА, 2011. – Вип. 1. – С. 22–28.
2. Геометричний аналіз кліматичних показників / О.В. Сергейчук, В.П. Шитюк // Праці Тавр. держ. агротехн. акад. – Мелітополь: ТДАТА, 2009. – Вип. 4: прикл. геометрія та інж. графіка. – Т. 43. – С. 81–87.
3. ДСТУ–Н Б В.1.1–27:2011
4. Лицкевич. В.К. Жилище и климат ч. – М.: Стройиздат, 1984. – 288 с.
5. Мартинов В.Л. Моделирование расчетных показателей при проектировании энергоактивных жилых зданий // Прикладная геометрия и инженерная графика. Труды ТДАТА. – Вип. 4, Т. 45. – Мелитополь, 2009. – С. 109–113.
6. Мартинов В.Л. Щодо питання розрахунку надходження сонячної радіації на площину будинку за рік «за збільшеними показниками» // Прикладна геометрія та інженерна графіка. – Харків: ХДУХТ, 2010. – Вип. 26. – С.163–167.
7. Містобудування. Планування і забудова міських і сільських поселень: ДБН 360–92**. [Чинні від 1992–01–01] / Мінбудархітектури України. – К.: Укрархбудінформ, 1993. – 107 с. – (Державні будівельні норми України).
8. Сергейчук О.В. Геометричне моделювання променевого теплообміну в атмосфері при хмарному небі // Геометричне та комп’ютерне моделювання: зб. наук. праць. — Харків: ХДУХТ, 2006. – Вип. 15. – С. 100–106.
9. Сергейчук О.В. Оптимізація форми енергоефективної будівлі, зовнішня оболонка якої *n*-параметрична поверхня // Прикл. геометрія та інж. графика. – К. : КНУБА, 2010. – Вип.85. – С. 150– 155.
10. Qt Group. – [Цит. 20.10.2022]. – Доступний з https://www.qt.io.

11. QXlsx. Excel file(*.xlsx) reader/writer library using Qt 5 or 6. – [Цит. 10.10.2022]. – Доступний з https://qtexcel.github.io/QXlsx/.

ДОДАТКИ

ДОДАТОК А

Вихідний код програми mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QRegExp>
#include <QRegExpValidator>
#include <QStringList>
#include <math.h>
#include <QtAlgorithms>
#include <QHeaderView>
#include <QStringList>

#include <QTextStream>
#include <QVariant>


MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);



    QRegExp
re("(\\d|\\d\\d|\\d\\d\\d)\\s+(\\d|\\d\\d)(\\s+((\\d|\\d\\d)(.\\d+
)?))?|((\\d|\\d\\d|\\d\\d\\d).\\d+)");
    QRegExpValidator *validator = new QRegExpValidator(re,
this);
    ui->lineEdit_Lat->setValidator(validator);
    ui->lineEdit_Lon->setValidator(validator);

    //connect(      this,      SIGNAL(on_open_server(long)),
websocket_worker, SLOT(slot_open(long)) );
```

```
        connect(ui->lineEdit_Lat,    SIGNAL(textChanged(QString)),
this, SLOT(Lat_adjustTextColor(QString)));
        connect(ui->lineEdit_Lon,    SIGNAL(textChanged(QString)),
this, SLOT(Lon_adjustTextColor(QString)));

        //======================
        DSTU_data = new DB_data;

        tabexl = new TableExcel();

    }

    MainWindow::~MainWindow()
    {
        delete ui;
        delete tabexl;
    }

    void MainWindow::Lat_adjustTextColor(QString s){
        if(!ui->lineEdit_Lat->hasAcceptableInput())
            ui->lineEdit_Lat->setStyleSheet("QLineEdit   {   color:
red;}");
        else
            ui->lineEdit_Lat->setStyleSheet("QLineEdit   {   color:
black;}");
    }
    void MainWindow::Lon_adjustTextColor(QString s){
        if(!ui->lineEdit_Lon->hasAcceptableInput())
            ui->lineEdit_Lon->setStyleSheet("QLineEdit   {   color:
red;}");
        else
            ui->lineEdit_Lon->setStyleSheet("QLineEdit   {   color:
black;}");
    }
```

```
double MainWindow::Convert_CoordAngel(QString str)
{
    double d=0;
    bool ok;
    QStringList str_arr=str.split(QRegExp("\\s+"));
    if(str_arr.size()==1)
    {
        d = str_arr[0].toDouble(&ok);
    }
    else if(str_arr.size()==2)
    {
        d = str_arr[0].toDouble(&ok);
        if(str_arr[1].toDouble(&ok)<60)
            d += str_arr[1].toDouble(&ok)/60.0;
    }
    else if(str_arr.size()==3)
    {
        d = str_arr[0].toDouble(&ok);
        if(str_arr[1].toDouble(&ok)<60)
            d += str_arr[1].toDouble(&ok)/60.0;
        if(str_arr[2].toDouble(&ok)<60)
            d += str_arr[2].toDouble(&ok)/3600.0;
    }

    return d;
}

void MainWindow::on_tabWidget_main_currentChanged(int index)
{ // Изменены входные данные
    QString str;
    Lat = Convert_CoordAngel( ui->lineEdit_Lat->text());
    Lon = Convert_CoordAngel( ui->lineEdit_Lon->text());
    Mode = ui->radioButton_m1->isChecked();

    str = QString("lat %1  lon %2").arg(Lat).arg(Lon);
    ui->label_test->setText(str);
```

```
        if(index==1)
        {
            Calculate_A();
            Calculate_B();
            Calculate_C();

            tabexl->SaveTable();
        }
    }

    void MainWindow::Calculate_A()
    {
        long i;
     //   long city_idx = find_Index_A();
    //2 4 5 6 24 25 26 27 29 30

        QStringList lst={"2", "4", "5", "6", "24", "25", "26",
"27", "29", "30"};
        for(i=0; i<lst.size(); i++)
        {
            Table_Data_t d;
            d.table_num = lst[i];
    //         d.data = DSTU_data->GetTable(d.table_num, city_idx,
&d.table_name);
            d.data = DSTU_data->GetTable_A(d.table_num, Lat, Lon,
&d.table_name);
            CreateTable(&d, ui->tab_TempHum);
            ui->gridLayout_TH->addWidget(d.label);
            ui->gridLayout_TH->addWidget(d.table);

            TableData.append(d);
        }
    }
    void MainWindow::Calculate_B()
    {
```

```
        // 7,8,9,19
        long i;
        QStringList lst={"7", "8", "9", "19"};
        for(i=0; i<lst.size(); i++)
        {
            Table_Data_t d;
            d.table_num = lst[i];
            if(Mode)
                d.data  =  DSTU_data->GetTable_B_near(d.table_num,
Lat, Lon, &d.table_name);
            else
                d.data  =  DSTU_data->GetTable_B_appr(d.table_num,
Lat, Lon, &d.table_name);
            CreateTable(&d, ui->tab_TempHum);
            ui->gridLayout_TH->addWidget(d.label);
            ui->gridLayout_TH->addWidget(d.table);

            TableData.append(d);
        }
    }
    void MainWindow::Calculate_C()
    {
        //10, 14, 18, 20
        QVector< QVector<QVariant> > data_res;
        //DSTU_data->GetTable_C("10",    Lat,    Lon,    nullptr,
&data_res);
        long i, j;
        QStringList lst={"10", "14", "18", "20"};
        for(i=0; i<lst.size(); i++)
        {
            Table_Data_t d;
            d.table_num = lst[i];
            d.data = DSTU_data->GetTable_C(d.table_num, Lat, Lon,
&d.table_name, &data_res);
            CreateTableArr(&d, ui->tab_TempHum, &data_res);
            ui->gridLayout_TH->addWidget(d.label);
```

```
            ui->gridLayout_TH->addWidget(d.table);

            TableData.append(d);

            for(j=0; j<data_res.size(); j++)
            {
                data_res[j].clear();
            }
            data_res.clear();
        }

    }
    void   MainWindow::CreateTable(   Table_Data_t*   d,   QWidget
*parent )//QTableWidget *table, QVector<TableItem_t> *vec_table)
    {
        long i;

        QString str = "Табл." + d->table_num+". "+d->table_name;
        d->label = new QLabel(str, parent);
        d->table = new QTableWidget(parent);
        d->table->setMaximumHeight(4*22);
        d->table->setMinimumHeight(4*22);
        d->table->setSizePolicy(QSizePolicy::MinimumExpanding,
QSizePolicy::MinimumExpanding);

        d->table->setRowCount(3);
        d->table->setColumnCount( d->data.size() );
        QHeaderView *hwh=d->table->horizontalHeader();
        hwh->setVisible(true);
        hwh->setSectionResizeMode(QHeaderView::Interactive);
        QHeaderView *hwv=d->table->verticalHeader();
        hwv->setVisible(false);


        tabexl->write( tabexl->row_cnt, 1, QVariant(str) );
        tabexl->row_cnt++;
```

```
        for(i=0; i<d->table->rowCount(); i++)
        {
            d->table->setRowHeight(i, 20);
        }
        long cnt=0, pos=0;
        long flag_name3=0;
        for(i=0; i<d->data.size(); i++)
            if( (d->data)[0].Name3!="" ) {flag_name3 =1; break;}

        if( flag_name3 )
        {
            for(i=0; i<d->data.size(); i++)
            {
                if( (d->data)[i].Name3=="--" && cnt==0 )
                {
                    d->table->setItem(pos,i,                    new
QTableWidgetItem( (d->data)[i].Name3 ) );

                    tabexl->write(    tabexl->row_cnt,    i,    (d-
>data)[i].Name3 );

                    cnt++;
                }
                if( (d->data)[i].Name3!="--" && cnt!=0 )
                {
                    d->table->setSpan(pos,i-cnt,1,cnt);
                    tabexl->merge(tabexl->row_cnt,  i-cnt,  tabexl-
>row_cnt, i-1);
                    cnt = 0;
                }
                if(cnt>0) cnt++;
                if( (d->data)[i].Name3!="--" && cnt==0 )
                {
                    d->table->setItem(pos,i,                    new
QTableWidgetItem( (d->data)[i].Name3 ) );
```

```
                    tabexl->write(    tabexl->row_cnt,    i,    (d-
>data)[i].Name3 );
                }
            }
            if(cnt>0)
            {
                d->table->setSpan(pos,i-cnt,1,cnt);
                tabexl->merge(tabexl->row_cnt,    i-cnt,    tabexl-
>row_cnt, i-1);
            }
            pos++;
            tabexl->row_cnt++;
        }

        for(i=0; i<d->data.size(); i++)
        {
            if( (d->data)[i].Name1=="--" && cnt==0 )
            {
                d->table->setItem(pos,i,   new   QTableWidgetItem(
(d->data)[i].Name1 ) );
                tabexl->write(     tabexl->row_cnt,     i,     (d-
>data)[i].Name1 );
                cnt++;
            }
            if( (d->data)[i].Name1!="--" && cnt!=0 )
            {
                d->table->setSpan(pos,i-cnt,1,cnt);
                tabexl->merge(tabexl->row_cnt,    i-cnt,    tabexl-
>row_cnt, i-1);
                cnt = 0;
            }
            if(cnt>0) cnt++;
            if( (d->data)[i].Name1!="--" && cnt==0 )
            {
                d->table->setItem(pos,i,   new   QTableWidgetItem(
(d->data)[i].Name1 ) );
```

```
                tabexl->write(     tabexl->row_cnt,     i,     (d-
>data)[i].Name1 );
            }
        }
        if(cnt>0)
        {
            d->table->setSpan(pos,i-cnt,1,cnt);
            tabexl->merge(tabexl->row_cnt,     i-cnt,     tabexl-
>row_cnt, i-1);
        }
        tabexl->row_cnt++;

        for(i=0; i<d->data.size(); i++)
        {
            d->table->setItem(1,i,   new   QTableWidgetItem(   (d-
>data)[i].Name2 ) );
            tabexl->write( tabexl->row_cnt, i, (d->data)[i].Name2
);
        }
        for(i=0; i<d->data.size(); i++)
        {
            //if((*vec_table)[i].Data.toInt()!=1000)
            d->table->setItem(2,i,      new      QTableWidgetItem(
QString("%1").arg((d->data)[i].Data.toString()) ) );
            tabexl->write(           tabexl->row_cnt,           i,
QString("%1").arg((d->data)[i].Data.toString()) );
        }
        tabexl->row_cnt++;
        tabexl->row_cnt++;
    }

    void   MainWindow::CreateTableArr(Table_Data_t*   d,   QWidget
*parent, QVector< QVector<QVariant> > *data_arr )
    {
        long i;
```

```
        QString str = "Табл." + d->table_num+". "+d->table_name;
        d->label = new QLabel(str, parent);
        d->table = new QTableWidget(parent);
        d->table->setMaximumHeight((data_arr->size()+3)*22);
        d->table->setMinimumHeight((data_arr->size()+3)*22);
        d->table->setSizePolicy(QSizePolicy::MinimumExpanding,
QSizePolicy::MinimumExpanding);

        d->table->setRowCount((data_arr->size()+2));
        d->table->setColumnCount( (*data_arr)[0].size() );
        QHeaderView *hwh=d->table->horizontalHeader();
        hwh->setVisible(true);
        hwh->setSectionResizeMode(QHeaderView::Interactive);
        QHeaderView *hwv=d->table->verticalHeader();
        hwv->setVisible(false);

        tabexl->write( tabexl->row_cnt, 1, str );

        for(i=0; i<d->table->rowCount(); i++)
        {
            d->table->setRowHeight(i, 20);
        }
        long cnt=0;
        for(i=0; i<d->data.size(); i++)
        {
            if( (d->data)[i].Name1=="--" && cnt==0 )
            {
                d->table->setItem(0,i, new QTableWidgetItem( (d-
>data)[i].Name1 ) );
                tabexl->write(    tabexl->row_cnt,    i,    (d-
>data)[i].Name1  );
                cnt++;
            }
            if( (d->data)[i].Name1!="--" && cnt!=0 )
            {
                d->table->setSpan(0,i-cnt,1,cnt);
```

```
                tabexl->merge(tabexl->row_cnt,   i-cnt,   tabexl-
>row_cnt, i-1);
                cnt = 0;
            }
            if(cnt>0) cnt++;
            if( (d->data)[i].Name1!="--" && cnt==0 )
            {
                d->table->setItem(0,i, new QTableWidgetItem( (d-
>data)[i].Name1 ) );
                tabexl->write(    tabexl->row_cnt,    i,    (d-
>data)[i].Name1  );
            }

            //if((*vec_table)[i].Data.toInt()!=1000)
            //      d->table->setItem(2,i, new QTableWidgetItem(
QString("%1").arg((d->data)[i].Data.toString()) ) );
        }
        if(cnt>0)
        {
            d->table->setSpan(0,i-cnt,1,cnt);
            tabexl->merge(tabexl->row_cnt,    i-cnt,    tabexl-
>row_cnt, i-1);
        }
        tabexl->row_cnt++;

        for(i=0; i<d->data.size(); i++)
        {
            d->table->setItem(1,i,   new   QTableWidgetItem(   (d-
>data)[i].Name2 ) );
            tabexl->write( tabexl->row_cnt, i, (d->data)[i].Name2
);
        }
        tabexl->row_cnt++;
        long j;
        for(i=0; i<data_arr->size(); i++)
        {
```

```
            d->table->setItem(2+i,0,     new     QTableWidgetItem(
(((*data_arr)[i])[0]).toString() ) );
            tabexl->write(           tabexl->row_cnt,           0,
(((*data_arr)[i])[0]).toString() );
            for(j=1; j<(*data_arr)[i].size(); j++)
            {
                QVariant val = (((*data_arr)[i])[j]);
                double val_d = val.toDouble();
                d->table->setItem(2+i,j,   new   QTableWidgetItem(
QString::asprintf("%.1f", val_d) ) );
                tabexl->write(         tabexl->row_cnt,         j,
QString::asprintf("%.1f", val_d) );
            }
            tabexl->row_cnt++;
        }
        tabexl->row_cnt++;
    }
```

ДОДАТОК Б

Вихідний код програми db_data.cpp

```
#include "db_data.h"

#include <math.h>
#include <QtSql/QSqlDatabase>
#include <QtSql/QSqlQueryModel>
#include <QtSql/QSqlError>
#include <QtSql/QSqlQuery>
#include <QtSql/QSqlRecord>

#include<QTextStream>

typedef struct
{
    long Idx;
    double Lat, Lon;
} Idx_Lat_t;


DB_data::DB_data()
{
    db = new QSqlDatabase;
    *db = QSqlDatabase::addDatabase( "QSQLITE" );

    db->setDatabaseName("..\\TableDSTU.db");

    if( !db->open() )
    {
        QTextStream(stdout) << "Can't open DB " << db-
>lastError().text();
        return;
    }
```

```
}

QVector<City_data_t> DB_data::get_City_data()
{
    QVector<City_data_t> res;

    QSqlQuery query( *db );
    QString str;

    str=QString("SELECT Idx,City,Lat,Lon FROM City");

    if( !query.exec(str) )
    {
      //  str = QString("Can't CREATE table \"%1\": %2")
       //                 .arg(    Variables->getName(var_idx)
).arg(query.lastError().text());
         QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
    }
    else
    {
        QSqlRecord rec = query.record();
        while(query.next())
        {
            City_data_t d;
            d.Idx = query.value(0).toUInt();
            d.Name = query.value(1).toString();
            d.Lat = query.value(2).toDouble();
            d.Lon = query.value(3).toDouble();
            res.append(d);
        }
    }
    return res;
}
```

```
    //====
    QVector<TableItem_t>  DB_data::GetTable_A(QString  num_table,
double city_Lat, double city_Lon, QString *table_name)
    {
        QSqlQuery query( *db );
        QString str;

        //=================
        str=QString("SELECT    Name    FROM    TableName    WHERE
TableNumber=%1").arg(num_table);
        QVector<QVariant> res1 = get_Record( str );
    //    QTextStream(stdout) << "i1 " << res1[0].toString();
        if(table_name)
            (*table_name) = res1[0].toString();

        //=================

        QVector<TableItem_t> table;
        table = get_TableItem(num_table);
        //=================

        QVector<Idx_Lat_t> table1;

        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1,                          City                          WHERE
Table%1.CityIndex=City.CityIndex").arg(num_table);
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
```

```
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table1.append(d);
            }
        }
        long i;
        long idx_min1=0;
        double val_min = Distance_2Point(city_Lat, city_Lon,
table1[0].Lat, table1[0].Lon);
        for(i=1; i<table1.size(); i++)
        {
            double d = Distance_2Point(city_Lat, city_Lon,
table1[i].Lat, table1[i].Lon);
            if(val_min > d)
            {
                val_min = d;
                idx_min1 = i;//table1[i].Idx;
            }
        }

    //    long city_idx = table1[idx_min1].Idx;
        //---
        QTextStream(stdout) << "Table " << num_table << " Idx="
<< table1[idx_min1].Idx <<endl;

        str = QString("SELECT ");
        for(i=0; i<table.size(); i++)
            str += table[i].Item + ((i==table.size()-1) ?
"":",");
        str += QString(" FROM Table%1 WHERE Idx=%2").arg(
num_table ).arg( table1[idx_min1].Idx );

        QVector<QVariant> res = get_Record(str);
```

```
        for(int i=0; i<res.size(); i++)
        {
            table[i].Data = res[i];
        }

        return table;
    }


    QVector<TableItem_t>          DB_data::GetTable_B_appr(QString
num_table, double city_Lat, double city_Lon, QString *table_name)
    {
        QVector<Idx_Lat_t> table1;
        QVector<Idx_Lat_t> table2;
        QSqlQuery query( *db );
        QString str;

        //=================
        str=QString("SELECT    Name    FROM    TableName    WHERE
TableNumber=%1").arg(num_table);
        QVector<QVariant> res_name = get_Record( str );
    //    QTextStream(stdout) << "i1 " << res_name[0].toString();
        if(table_name)
            (*table_name) = res_name[0].toString();
        //================
        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1,    City    WHERE    Table%1.CityIndex=City.CityIndex    AND
City.Lat<%2").arg(num_table).arg(city_Lat);
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
```

```
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table1.append(d);
            }
        }
        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1,   City   WHERE   Table%1.CityIndex=City.CityIndex   AND
City.Lat>=%2").arg(num_table).arg(city_Lat);
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table2.append(d);
            }
        }

        long i;
        long idx_min1=0, idx_min2=0;
        //idx_min1 = table1[0].Idx;
        double  val_min  =  Distance_2Point(city_Lat,  city_Lon,
table1[0].Lat, table1[0].Lon);
        for(i=1; i<table1.size(); i++)
```

```
        {
            double d = Distance_2Point(city_Lat, city_Lon,
table1[i].Lat, table1[i].Lon);
            if(val_min > d)
            {
                val_min = d;
                idx_min1 = i;//table1[i].Idx;
            }
        }
        //idx_min2 = table2[0].Idx;
        val_min = Distance_2Point(city_Lat, city_Lon,
table2[0].Lat, table2[0].Lon);
        for(i=1; i<table2.size(); i++)
        {
            double d = Distance_2Point(city_Lat, city_Lon,
table2[i].Lat, table2[i].Lon);
            if(val_min > d)
            {
                val_min = d;
                idx_min2 = i;//table2[i].Idx;
            }
        }
    //      QTextStream(stdout) << "i1 " << idx_min1<< "i2 " <<
idx_min2;

        double d1, d2, koef;
        d1      =       Distance_2Point(city_Lat,       city_Lon,
table1[idx_min1].Lat, table1[idx_min1].Lon);
        d2      =       Distance_2Point(city_Lat,       city_Lon,
table2[idx_min2].Lat, table2[idx_min2].Lon);
        koef = d1/(d1+d2);

        //==================================
        QVector<TableItem_t> table;
        table = get_TableItem(num_table);
        //=================
```

```
        QVector<QVariant> res;

        str = QString("SELECT ");
        for(i=0; i<table.size(); i++)
            str += table[i].Item + ((i==table.size()-1) ? "":",");
        str += QString(" FROM Table%1 WHERE Idx=").arg(num_table);//.arg(table1[idx_min1].Idx);

        QVector<QVariant> res1 = get_Record( str + QString("%1").arg(table1[idx_min1].Idx) );
        QVector<QVariant> res2 = get_Record( str + QString("%1").arg(table2[idx_min2].Idx) );

        QTextStream(stdout) << "Table " << num_table << " Idx1=" << table1[idx_min1].Idx << " Idx2= " << table2[idx_min2].Idx<<endl;

        //QVector<QVariant> result;
        for(i=0; i<res1.size(); i++)
        {
            double delta;
            delta = res2[i].toDouble() - res1[i].toDouble();
            double d = res1[i].toDouble() + delta*koef ;
            table[i].Data = QVariant(d);
        }
        return table;
    }

    QVector<TableItem_t> DB_data::GetTable_C(QString num_table, double city_Lat, double city_Lon, QString *table_name, QVector< QVector<QVariant> > *data_res)
    {
        QSqlQuery query( *db );
        QString str;
```

```
        long i;

        //=================
        str=QString("SELECT    Name    FROM    TableName    WHERE
TableNumber=%1").arg(num_table);
        QVector<QVariant> res_name = get_Record( str );
        if(table_name)
            (*table_name) = res_name[0].toString();
        //================

        //44, 46, 48, 50
        long Lat1, Lat2;
        if(city_Lat < 44.0) { Lat1 = 44; Lat2 = 46; }
        else if(city_Lat >= 44.0 && city_Lat < 46.0)
        { Lat1 = 44; Lat2 = 46; }
        else if(city_Lat >= 46.0 && city_Lat < 48.0)
        { Lat1 = 46; Lat2 = 48; }
        else if(city_Lat >= 48.0 && city_Lat < 50.0)
        { Lat1 = 48; Lat2 = 50; }
        else
        { Lat1 = 48; Lat2 = 50; }
        //================

        QVector<TableItem_t> table1;
        str=QString("SELECT  Idx,Item,Name1,Name2,Name3  FROM  Name
WHERE   TableNumber=%1   AND   (INSTR(   Item,   'Lat%2')   >   0   OR
Item='Interval')").arg(num_table).arg(Lat1);
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
            QTextStream(stdout) << "Req: " << str<<endl;
        }
        else
        {
            QSqlRecord rec = query.record();
```

```
            while(query.next())
            {
                TableItem_t it;
                it.Idx = query.value(0).toInt();
                it.Item = query.value(1).toString();
                it.Name1 = query.value(2).toString();
                it.Name2 = query.value(3).toString();
                it.Name3 = query.value(4).toString();
                table1.append(it);
            }
        }
        QVector<TableItem_t> table2;
        str=QString("SELECT Idx,Item,Name1,Name2,Name3 FROM Name
WHERE TableNumber=%1 AND (INSTR( Item, 'Lat%2') > 0 OR
Item='Interval')").arg(num_table).arg(Lat2);
        if( !query.exec(str) )
        {
            QTextStream(stdout) << "get_City_data DB " << db-
>lastError().text();
            QTextStream(stdout) << "Req: " << str<<endl;
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                TableItem_t it;
                it.Idx = query.value(0).toInt();
                it.Item = query.value(1).toString();
                it.Name1 = query.value(2).toString();
                it.Name2 = query.value(3).toString();
                it.Name3 = query.value(4).toString();
                table2.append(it);
            }
        }
```

```
        //=================
        double d1, d2, koef;
        d1 = Distance_2Point(city_Lat, city_Lon, Lat1, city_Lon);
        d2 = Distance_2Point(city_Lat, city_Lon, Lat2, city_Lon);
        koef = d1/(d1+d2);

        //===================================
        //=================

    //    QVector< QVector<QVariant> > table;
        QTextStream(stdout) << "Table " << num_table << " Lat1="
<< Lat1 << " Lat2= " << Lat2<<endl;

        str = QString("SELECT ");
        for(i=0; i<table1.size(); i++)
            str  +=  table1[i].Item  +  ((i==table1.size()-1)  ?
"":",");
        str += QString(" FROM Table%1").arg(num_table);

        QVector<QVariant> res1 = get_Record( str );

        str = QString("SELECT ");
        for(i=0; i<table2.size(); i++)
            str  +=  table2[i].Item  +  ((i==table2.size()-1)  ?
"":",");
        str += QString(" FROM Table%1").arg(num_table);

        QVector<QVariant> res2 = get_Record( str );

        //QVector<QVariant> result;
        long j;
        if( data_res )
        {
            for(i=0; i<res1.size(); i+=table1.size())
            {
                QVector<QVariant> t_sub;
```

```
                int interval = res1[i].toInt();
                if(interval == 24)
                    t_sub.append( QString("Усього") );
                else if(interval == 25)
                    t_sub.append( QString("Середня")  );
                else
                    t_sub.append( interval );

                double delta;

                for(j=1; j<table1.size(); j++)
                {
                    delta      =      res2[i+j].toDouble()      -
res1[i+j].toDouble();
                    double  d  =  res1[i+j].toDouble()  +  delta*koef
;
                    t_sub.append( QVariant(d) );
                }
                data_res->append( t_sub );
            }
        }

        return table1;
    }

    QVector<TableItem_t>          DB_data::GetTable_B_near(QString
num_table, double city_Lat, double city_Lon, QString *table_name)
    {
        QVector<Idx_Lat_t> table1;
        QVector<Idx_Lat_t> table2;
        QSqlQuery query( *db );
        QString str;

        //=================
        str=QString("SELECT    Name    FROM    TableName    WHERE
TableNumber=%1").arg(num_table);
```

```
        QVector<QVariant> res_name = get_Record( str );
        if(table_name)
            (*table_name) = res_name[0].toString();
        //================
        str=QString("SELECT Table%1.Idx, City.Lat, City.Lon  FROM
Table%1,   City   WHERE   Table%1.CityIndex=City.CityIndex   AND
City.Lat<%2").arg(num_table).arg(city_Lat);
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                Idx_Lat_t d;
                d.Idx = query.value(0).toInt();
                d.Lat = query.value(1).toDouble();
                d.Lon = query.value(2).toDouble();
                table1.append(d);
            }
        }

        long i;
        long idx_min1=0;
        //idx_min1 = table1[0].Idx;
        double  val_min  =  Distance_2Point(city_Lat,  city_Lon,
table1[0].Lat, table1[0].Lon);
        for(i=1; i<table1.size(); i++)
        {
            double  d    =  Distance_2Point(city_Lat,  city_Lon,
table1[i].Lat, table1[i].Lon);
            if(val_min > d)
            {
```

```
                val_min = d;
                idx_min1 = i;//table1[i].Idx;
            }
        }

        //==================================
        QVector<TableItem_t> table;
        table = get_TableItem(num_table);
        //=================

        QVector<QVariant> res;

        str = QString("SELECT ");
        for(i=0; i<table.size(); i++)
            str   +=   table[i].Item   +   ((i==table.size()-1)   ?
"":",");
        str      +=      QString("      FROM      Table%1      WHERE
Idx=").arg(num_table);//.arg(table1[idx_min1].Idx);

        QVector<QVariant>    res1    =    get_Record(    str    +
QString("%1").arg(table1[idx_min1].Idx) );

        QTextStream(stdout) << "Table " << num_table << " Idx1="
<< table1[idx_min1].Idx << endl;

        for(i=0; i<res1.size(); i++)
        {
            table[i].Data = QVariant( res1[i].toDouble() );
        }
        return table;
    }


    QVector<TableItem_t>            DB_data::get_TableItem(QString
num_table)
    {
```

```
        QString str;
        QVector<TableItem_t> table;

        str=QString("SELECT  Idx,Item,Name1,Name2,Name3  FROM  Name
WHERE TableNumber=%1").arg(num_table);

        QSqlQuery query( *db );
        if( !query.exec(str) )
        {
            QTextStream(stdout)  <<  "get_City_data  DB  "  <<  db-
>lastError().text();
            QTextStream(stdout) << "Req: " << str<<endl;
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                TableItem_t it;
                it.Idx = query.value(0).toInt();
                it.Item = query.value(1).toString();
                it.Name1 = query.value(2).toString();
                it.Name2 = query.value(3).toString();
                it.Name3 = query.value(4).toString();

                table.append(it);
            }
        }
        return table;
    }
    QVector<QVariant> DB_data::get_Record(QString req)
    {
        QVector<QVariant> res;
        QSqlQuery query( *db );
        if( !query.exec(req) )
        {
```

```
            QTextStream(stdout) << "get_City_data DB " << db->lastError().text();
            QTextStream(stdout) << "Req: " << req<<endl;
        }
        else
        {
            QSqlRecord rec = query.record();
            while(query.next())
            {
                for(int i=0; i<rec.count(); i++)
                {
                    res.append( query.value(i) );
                }
            }
        }
        return res;
    }

    double DB_data::Distance_2Point(double lat1, double lon1, double lat2, double lon2, double *alpha)
    {
        const double Earth_a = 6378137.0; // Основные полуоси
        const double Earth_b = 6356752.314245; // Неосновные полуоси
        const double Earth_e2 = 0.006739496742337; // Квадрат эксцентричности эллипсоида
        const double Earth_f = 0.003352810664747; // Выравнивание эллипсоида
        const double GRAD2RAD = M_PI/180.0;

        double fdLambda = (lon1 - lon2) * GRAD2RAD;
        double fdPhi = (lat1 - lat2) * GRAD2RAD;
        double fPhimean = ((lat1 + lat2) / 2.0) * GRAD2RAD;
        double fTemp = 1 - Earth_e2 * (pow(sin(fPhimean), 2));
        double fRho = (Earth_a * (1 - Earth_e2)) / pow(fTemp, 1.5);
```

```
        double fNu = Earth_a / (sqrt(1 - Earth_e2 *
(sin(fPhimean) * sin(fPhimean))));
        double fz = sqrt(pow(sin(fdPhi / 2.0), 2)+cos(lat2 *
GRAD2RAD) * cos(lat1 * GRAD2RAD)*pow(sin(fdLambda / 2.0), 2));

        if(fz>1) fz=1;
        else if(fz<-1) fz=-1;
        fz = 2 * asin(fz);

        if(alpha!=nullptr)
        {
            *alpha = fz;
        }

        double fAlpha, sinfz=sin(fz);
        if( sinfz!=0.0 ) fAlpha = cos(lat2 * GRAD2RAD) *
sin(fdLambda) / sin(fz);
        else fAlpha = 0;

        if(fAlpha>1) fAlpha=1;
        else if(fAlpha<-1) fAlpha=-1;
        fAlpha = asin(fAlpha);

        double fR = (fRho * fNu) / ((fRho * pow(sin(fAlpha), 2))
+ (fNu *pow(cos(fAlpha), 2)));

        return (fz * fR);
    }
```