БАКАЛАВРСЬКА РОБОТА

БР. ІП – 18.00.00.000 ПЗ

Група ІП-21-1

Кріцак Степан

2025

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра інженерії програмного забезпечення

Кріцак Степан Ігорович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Програмування Інтернету речей

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Кріцак С.І._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ __Тимків Дмитро Федорович, д.т.н., професор
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____202 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Кріцак Степан Ігорович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Програмування Інтернету речей”.

керівник проекту (роботи) д.т.н., професор Тимків Дмитро Федорович

затверджені наказом вищого навчального закладу від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Вступ до інтернету речей
2. Проєктування та мовні особливості iot-систем
3.Практична реалізація iot-проекту

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Зображення не масштабовано відповідно до фактичної дальності (рис.1.2, ст.19)
2. Блок-схема виконання програми (рис.2.1, ст.25)
3. Приклад оголошення змінних у мові зі статичною системою типів (рис.2.4, ст.41)
4 Raspberry Pi 3 Model B (рис.3.1, ст.52)
5. Теговане об'єднання у мові C (рис.3.6, ст.59)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки, бакалаврської роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 52 сторінки, 13 таблиць, 8 рисунків, список використаних джерел із 26 найменування,

Метою роботи є аналіз сучасних підходів, інструментів і платформ для розробки IoT-додатків, оцінка їх ефективності та надання рекомендацій для створення масштабованих і безпечних рішень.

Об'єкт дослідження: Інтернет промов (IoT) як сучасна технологічна платформа для створення розподілених обчислювальних систем.

Предмет дослідження: проектування мови програмування Daspel та архітектури інтерпретатора для пристроїв IoT із використанням Raspberry Pi та сенсорної плати Sense HAT.

Результати дослідження: Розроблено та реалізовано прототип інтерпретатора мови Daspel для IoT.

У першому розділі - розділі розглядаються основи Інтернету речей, включаючи визначення IoT, його ключові принципи та огляд пов’язаних проєктів, які демонструють сучасні тенденції в цій галузі

Другий розділ - присвячений проєктуванню та мовним особливостям IoT-систем, з фокусом на вимоги до мов програмування, проєктування специфічних компонентів, таких як Даспела, і використання стандартів, наприклад, символів Unicode.

Третій розділ - зосереджується на практичній реалізації IoT-проєкту, включаючи доказ концепції, використання платформи Raspberry Pi із Sense HAT і процес впровадження

Висновок: узагальнено ключові результати та окреслено перспективи розвитку програмування IoT-систем.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ, ПРОГРАМУВАННЯ, IOT-ДОДАТКИ, ARDUINO, RASPBERRY PI, MQTT, ХМАРНІ ПЛАТФОРМИ, ЕНЕРГОЕФЕКТИВНІСТЬ, БЕЗПЕКА, ПОРІВНЯЛЬНИЙ АНАЛІЗ.

ANNOTATION

The bachelor's thesis contains 52 pages, 8 figures, 13 tables a list of used sources with 26 names,

The purpose of the work is to analyze modern approaches, tools and platforms for developing IoT applications, assess their effectiveness and provide recommendations for creating scalable and secure solutions.

Object of research: Internet of Things (IoT) as a modern technological platform for creating distributed computing systems.

Subject of research: design of the Daspel programming language and interpreter architecture for IoT devices using Raspberry Pi and the Sense HAT sensor board.

Research results: A prototype of the Daspel language interpreter for IoT was developed and implemented.

The first chapter - the chapter discusses the basics of the Internet of Things, including the definition of IoT, its key principles and an overview of related projects that demonstrate current trends in the field.

The second chapter - is devoted to the design and language features of IoT systems, with a focus on the requirements for programming languages, the design of specific components such as Daspela, and the use of standards, such as Unicode characters.

The third chapter - focuses on the practical implementation of an IoT project, including a proof of concept, the use of the Raspberry Pi platform with the Sense HAT, and the implementation process.

Conclusion: the key results are summarized and the prospects for the development of IoT system programming are outlined.

KEYWORDS: INTERNET OF THINGS, PROGRAMMING, IOT APPLICATIONS, ARDUINO, RASPBERRY PI, MQTT, CLOUD PLATFORMS, ENERGY EFFICIENCY, SECURITY, COMPARATIVE ANALYSIS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT - Інтернет речей

ABS - антипробуксовувальні гальмівні системи

GPS - Глобальна система позиціонування

AST - абстрактне синтаксичне дерево

STD - Стандартна бібліотека

BLE - Bluetooth Low Energy

IEEE - Інститут електроніки та інженерів-електронщиків

WLAN - бездротової локальної мережі

SIG - Special Interest Group

AST - абстрактне синтаксичне дерево

ЗМІСТ

ВСТУП

Інтернет речей (IoT) є однією з найбільш трансформаційних технологій сучасності, що об’єднує фізичні пристрої, датчики та програмне забезпечення для збору, обробки та обміну даними в реальному часі. Від розумних будинків і промислових систем до медичних пристроїв і транспортної інфраструктури, IoT змінює спосіб взаємодії людини з навколишнім світом. Програмування IoT-систем відіграє ключову роль у реалізації цього потенціалу, забезпечуючи створення надійних, ефективних і безпечних рішень, здатних працювати в умовах обмежених ресурсів і складних мереж. Розробка таких систем вимагає не лише знань про апаратне забезпечення, але й спеціалізованих мов програмування, інструментів і підходів до проєктування, які враховують унікальні вимоги IoT.

Актуальність теми зумовлена стрімким зростанням кількості підключених пристроїв, яке, за прогнозами аналітичних компаній, таких як Gartner, досягне десятків мільярдів у найближчі роки. Розробники стикаються з викликами, пов’язаними з гетерогенністю апаратних платформ, обмеженнями енергоспоживання, а також необхідністю забезпечення безпеки та масштабованості. Програмування IoT-систем вимагає інтеграції таких технологій, як вбудовані системи, протоколи зв’язку (наприклад, MQTT, CoAP) і платформи, подібні до Raspberry Pi, що робить цю тему надзвичайно актуальною для досліджень і практичного застосування.

Метою цього дослідження є аналіз основ програмування Інтернету речей, включаючи принципи IoT, вимоги до мов програмування, проєктування систем і практичну реалізацію IoT-проєктів. Робота спрямована на систематизацію знань про сучасні підходи до розробки IoT-систем, а також на надання практичних рекомендацій для розробників, які працюють із вбудованими системами та мережами IoT. Дослідження охоплює як теоретичні аспекти, так і практичні приклади реалізації, що дозволяють створювати функціональні IoT-рішення.

Цей документ має на меті надати комплексне уявлення про програмування Інтернету речей, а також практичні рекомендації для розробників і дослідників, які прагнуть створювати інноваційні IoT-рішення в умовах швидкозмінного технологічного ландшафту.

Завданнями дослідження: Проаналізувати принципи роботи IoT та типові вимоги до мов програмування для цих систем, визначити критерії побудови легкої інтерпретованої мови з динамічною типізацією та подійно-орієнтованою моделлю. Забезпечити безпечне виконання скриптів із підтримкою заміни коду «на льоту». Впровадити механізми обробки помилок та відмовостійкості.

Об'єкт дослідження: Інтернет речей (IoT) як сучасна технологічна платформа для створення розподілених обчислювальних систем

Предмет дослідження: Проектування мови програмування Daspel та архітектури інтерпретатора для пристроїв IoT із використанням Raspberry Pi та сенсорної плати Sense

Методи дослідження: були використані аналіз літературних джерел, метод системного аналізу, проектування, програмна верифікація.

Наукова новизна: обґрунтовано ефективність підійно-орієнтованого підходу для сценаріїв IoT та запропоновано новий підхід до гарячої заміни скриптів на пристроях реальної години

Бакалаврська робота містить 65 сторінок,11 рисунків, 13 таблиць, три розділи, список використаних джерел із 27 найменуванням.

РОЗДІЛ 1. ВСТУП ДО ІНТЕРНЕТУ РЕЧЕЙ

1.1 Інтернет речей

Технології пройшли довгий шлях з часів появи обчислювальної техніки. Перші цифрові комп'ютери були великими, дорогими та споживали багато енергії. Написання програми для вирішення певного завдання кілька десятиліть тому могло зайняти більше часу, ніж сьогодні. З плином часу комп'ютери ставали меншими, швидшими, енергоефективнішими та, найголовніше, дешевшими. Потужність суперкомп'ютера того часу тепер можна знайти в сучасних смартфонах. Цей технологічний прогрес приніс багато переваг. Зв'язок та обмін інформацією по всьому світу і навіть у космосі можливі вже кілька десятиліть і є частиною нашого повсякденного життя. Усіляка інформація з різних носіїв зараз доступна на кінчиках наших пальців. В останні роки технології зробили крок у новому та незвіданому напрямку, і для цього явища з'явився новий термін: Інтернет речей (IoT).

Інтернет з можливістю збору та обміну даними. Ці пристрої або гаджети зазвичай оснащені мікроконтролерами, програмним забезпеченням, датчиками, виконавчими механізмами та мають підключення до Інтернету[2]. Такі гаджети можуть включати звичайні побутові предмети, такі як пральні машини, холодильники, звукові системи, кавоварки, будильники та багато іншого. Також у містах використовуються додатки IoT, такі як датчики, які контролюють дорожній рух, забруднення повітря та води, а також споживання електроенергії. У цій статті безпілотні автомобілі перевозитимуть людей до місць призначення, і ці автомобілі використовуватимуть датчики та бездротові технології для зв'язку один з одним у дорожньому русі. За оцінками, до кінця 2016 року буде підключено та використовуватиметься 6,4 мільярда (6,4 • 109) пристроїв, і що це число зросте до 20,8 мільярда у 2020 році [3].

«Датчик — це пристрій, який виявляє та реагує на певний тип вхідного сигналу з фізичного середовища. Конкретним вхідним сигналом може бути

світло, тепло, рух, вологість, тиск або будь-яке з багатьох інших явищ навколишнього середовища. Вихідним сигналом, як правило, є сигнал, який перетворюється на дисплей, зрозумілий для людини, у місці розташування датчика або передається електронним способом через мережу для зчитування чи подальшої обробки».[4] Пристрої Інтернету речей зазвичай оснащені одним або кількома датчиками, які вони використовують для збору інформації. Ця інформація потім обробляється внутрішньо на пристрої або надсилається назад на сервер (наприклад, Amazon Cloud Services або локальний комп'ютер).

Положення, розміщення та зміщення Цей датчик здатний визначати відстань до іншого об'єкта. Іншими словами, відстань. Інший тип такого датчика здатний визначати, яку відстань він пройшов від фіксованої точки, вимірюючи кутовий рух або обертання. Наприклад, робот може знати, яку відстань він пройшов, виходячи з того, скільки разів обертувалися його колеса.

Присутність і рух Датчики, що відчувають присутність, здатні виявляти, чи знаходиться об'єкт поблизу. Наприклад, лампа може бути оснащена датчиком присутності. Вона може засвітитися, якщо її датчик виявить, що людина увійшла в кімнату. Датчик може виявляти рух, тепло тіла або і те, й інше. Іншим способом виявлення певної присутності є використання пристрою відстеження. Цей пристрій можна, наприклад, розмістити на домашній тварині. Тоді датчик може виявляти, коли домашня тварина з пристроєм переміщується в зону його дії або виходить з неї.

Швидкість Ці датчики можуть вимірювати швидкість руху об'єкта за допомогою магнітів або світла. Прикладом використання є антипробуксовувальні гальмівні системи (ABS) в автомобілях, які вимірюють швидкість обертання коліс.

Температура Тип датчика, який зазвичай зустрічається в побуті. Відомим прикладом є цифровий термометр для вимірювання температури всередині та зовні будинку. Більш досконала система могла б вимірювати температуру в усіх кімнатах будинку та керувати пристроями регулювання температури. Система визначала б, коли потрібно вмикати нагрівальну панель, тепловий насос або

кондиціонер, якщо в приміщенні занадто спекотно або холодно, на основі показників температури. Здатність вимірювати кількість води в повітрі. Це також дуже поширений датчик, який можна знайти в пристроях, що вимірюють також температуру. Обладнання, встановлене з датчиками вологості, можна знайти в будинках у комплекті з термометром. Їх також можна знайти в теплицях, системах кондиціонування повітря, офісах та автомобілях[5].

Звук, акустика та вібрація Цей датчик виявлення звуку частіше називають мікрофоном. Мікрофон може бути розроблений по-різному. Він може бути розроблений для вловлювання лише певних типів шумів, таких як зачинення дверей або плескіт у долоні.

Світло Це датчики, які можуть виявляти видиме людиною світло. Подібно до датчиків наближення, ці датчики використовують видиме світло для виявлення змін і руху в середовищі. Наприклад, лампа може автоматично вмикатися, якщо виявляє, що в кімнаті занадто мало світла, і може вимикатися, якщо світла забагато.

Рух і обертання

Вони більш відомі як гіроскопи, і вони відчувають обертальний рух та зміни орієнтації. Камери можуть використовувати гіроскопи для виявлення тремтіння та нерівномірного руху, що допомагає їм залишатися на місці. Це призводить до менш тремтливих фотографій та відеозйомки.

GPS

Глобальна система позиціонування (GPS) – це система, яка дозволяє пристрою вимірювати своє положення та швидкість у будь-якій точці світу[6]. GPS-приймачі використовують супутники, що обертаються навколо Землі, для визначення свого положення. Для отримання точного положення потрібно щонайменше чотири супутники. Похибка зазвичай становить кілька метрів за умови хорошого сигналу. Супутники передають свої положення та час за допомогою радіохвиль, і ці сигнали вловлюються приймачем. Приймач знає, скільки часу минуло після відправлення сигналу, і він може визначити своє місцезнаходження на Землі. Деякі гаджети Інтернету речей мають встановлений

GPS-датчик, щоб вони могли реєструвати або сигналізувати про своє поточне місцезнаходження та швидкість. GPS-датчик можна використовувати для відстеження місцезнаходження людей, тварин та предметів. Домашня тварина, така як кішка, може носити нашийник із встановленим GPS-трекером, щоб власник міг знати, де знаходиться його улюбленець.

Енергоефективність

Низьке енергоспоживання є важливим ключовим елементом для Інтернету речей. Оскільки пристрої можна розміщувати у віддалених або важкодоступних місцях, зазвичай необхідно оснащувати їх внутрішнім джерелом живлення. Для використання Інтернету речей створюються нові технології з урахуванням низького енергоспоживання. Наприклад, нова версія Bluetooth під назвою Bluetooth Low Energy [7] та Wi-Fi HaLow [8]. Пристрої проводять більшу частину свого часу в стані низького енергоспоживання, де вони знаходяться в стані спокою та вмикаються лише тоді, коли датчики виявляють сигнал або пристрій має надсилати/отримувати інформацію. Це вимагає інтелектуального обладнання та сумісності з режимом сну з низьким енергоспоживанням [9]. 3D-пам'ять, 2.5D та 3D-процесори – це технології, розроблені для підвищення продуктивності з меншим енергоспоживанням, а також нижчими виробничими витратами [10][11]. Розвиток цих технологій частково зумовлений такими технологіями, як смартфони Інтернету речей.

1.2 Зв'язок

Інтернет речей зазвичай використовує технологію бездротового зв'язку для передачі інформації. Цю інформацію можна надсилати на інші пристрої Інтернету речей, термінали або сервери в Інтернеті. У деяких випадках надсилання даних через Інтернет може не бути обов'язковим. Наприклад, смартфон, який отримав повідомлення через додаток соціальної мережі, може надіслати сповіщення через Bluetooth на смарт-годинник власника. Годинник потім може відобразити повідомлення на своєму екрані. В інших ситуаціях

повідомлення надсилається через Інтернет, щоб сповістити віддалену сторону на різних відстанях, від кількох метрів до цілих континентів. Зі зростанням феномену Інтернету речей (IoT) виникла потреба в кращих та енергоефективніших технологіях. Бездротовий зв'язок – одна з технологій, які були модифіковані та розроблені для задоволення потреб IoT. Пристрої IoT мають дуже низьку ємність акумулятора, але зазвичай вони не надсилають багато інформації одночасно. Це означає, що розробники можуть створювати нові або модифікувати існуючі протоколи зв'язку, щоб споживати менше енергії за рахунок меншої пропускної здатності. У наступних розділах буде розглянуто два майбутніх та популярних бездротових протоколи, спрямованих на покращення часу роботи акумулятора та дальності дії сучасного IoT.

Bluetooth — це бездротова технологія, призначена для обміну даними на коротких відстанях. Ця технологія дозволяє безперервну потокову передачу даних між двома пристроями з високою швидкістю передачі (теоретична пропускна здатність 25 Мбіт/с). Bluetooth використовує радіосигнали та працює в діапазоні від 2,4 ГГц до 2485 ГГц, тобто він працює в частотному спектрі інших бездротових технологій, таких як WiFi[12]. Пристрої Bluetooth можуть з'єднуватися один з одним шляхом сполучення. Bluetooth не надсилає сигнали в загальному напрямку. Натомість він транслює сигнал, щоб інші пристрої поблизу могли його бачити та сполучатися. Однак вони повинні знаходитися на близькій відстані, зазвичай до 10 м. Bluetooth також має перевагу зворотної сумісності, тобто пристрої з новішими версіями Bluetooth все ще можуть взаємодіяти з пристроями зі старішими версіями Bluetooth.

Загальне вживання

Ця технологія дуже популярна та проста у використанні, тому вона використовується в багатьох поширених електронних пристроях, таких як смартфони. Наприклад, користувач може підключити телефон до бездротової гарнітури та передавати на неї музику. Bluetooth також можна використовувати для обміну даними з одного пристрою на інший, що спрощує передачу зображень з телефону на комп'ютер для резервного копіювання або синхронізацію даних на

двох пристроях. В останні роки спостерігається розвиток використання Bluetooth-маяків у роздрібних магазинах. Маяки можуть інформувати людину через її телефон про місцезнаходження та інформацію про магазин. Як Apple, так і Google, два основні розробники операційних систем для смартфонів, надають підтримку Bluetooth-маяків в iOS та Android[13][14]. Ця технологія працює таким чином, що телефон постійно прослуховує інформацію, що надсилається від таких маячків, споживаючи при цьому якомога менше енергії акумулятора. Коли він отримує сигнал від маячка, він може повідомити користувача, відобразивши повідомлення або відкривши призначену програму.

Низькоенергетичний Bluetooth (Bluetooth Smart)

Стандартний Bluetooth чудово підходить для безперервної передачі даних, але ця функція може бути не обов'язковою для пристроїв Інтернету речей, які не передають дані потоком, а передають їх короткими пакетами та перебувають у режимі низького енергоспоживання (сплячому режимі). Тому була створена нова версія Bluetooth: Bluetooth 4.0, також відома як Bluetooth Low Energy (BLE)[15]. Ця версія була розроблена з урахуванням потреб Інтернету речей, тобто вона пропонує низьке споживання енергії, перебуваючи в стані низького енергоспоживання, коли не зв'язується з іншими пристроями, та надсилаючи невеликі обсяги даних з високою швидкістю передачі (1 Мбіт/с). BLE також витрачає мало часу на сполучення з пристроями, що споживає менше енергії. У той час як класичний Bluetooth може витрачати близько 100 мс на підключення, BLE потрібно лише близько 6 мс. Що стосується батареї, ця технологія може забезпечити роботу невеликого пристрою протягом кількох років, використовуючи лише невелику батарейку розміром з монету як джерело живлення. Що стосується дальності дії, BLE має приблизно такий самий радіус дії, як і стандартний Bluetooth, приблизно 10 м у приміщенні та до 100 м на вулиці без перешкод.

Bluetooth версії 4.1 та 4.2

Bluetooth 4.0 раніше був відомий як Wibree та розроблявся компанією Nokia з 2001 по 2006 рік, перш ніж змінити назву [16]. Технологія була вперше

реалізована в iPhone 4S наприкінці 2011 року, а з того часу була додана в інші пристрої інших виробників. На момент написання статті було випущено дві нові версії специфікації Bluetooth 4.x: версія 4.1 та 4.2. Версія 4.1 була оновленням програмного забезпечення до існуючої специфікації, яке додало нові функції, що покращили зручність використання як для користувачів, так і для розробників. Одна з функцій вирішила загальновідому проблему, через яку сигнали Bluetooth 4.0 та 4G (LTE) заважали один одному, що призводило до погіршення продуктивності та збільшення споживання енергії. (версія 4.2)

Bluetooth 5

Наступну версію Bluetooth було анонсовано в червні 2016 року Bluetooth Special Interest Group (SIG), яка є хранителем та новатором технології Bluetooth[17]. Нова версія обіцяє збільшений радіус дії, швидкість передачі та можливості передачі повідомлень. Радіус дії збільшено в чотири рази: з теоретичного максимуму 100 м до 400 м на відкритому повітрі та з 10 м до 40 м у приміщенні. Швидкість передачі подвоєна з 1 Мбіт/с до 2 Мбіт/с. Bluetooth 4.2 дозволяє пристроям надсилати пакети розміром 31 байт, але Bluetooth 5 збільшує цей обсяг до 255 байт, що призводить до збільшення ємності на 800%. Ці значні покращення дозволяють пристроям залишатися на зв'язку навіть на великих відстанях і, можливо, зменшують кількість ретрансляторів, необхідних у мережі пристроїв Bluetooth. Передача даних вища, а це означає, що більше інформації можна передати за менший час. Як результат, споживається менше енергії через менший час, що використовується для надсилання даних. Починаючи з версії 4.2, пристрої можуть підключатися один до одного без сполучення, як це зазвичай відбувається під час сполучення телефону з бездротовою гарнітурою. Bluetooth-маяки використовують цей тип безпарного з’єднання для надсилання інформації та даних про місцезнаходження на смартфони. Bluetooth 5 дозволяє таким маякам надсилати набагато більше інформації в одному повідомленні, ніж попередні версії.

Wi-Fi

WiFi – це технологія, яка дозволяє пристроям підключатися до

бездротової локальної мережі (WLAN). Це дозволяє пристроям бездротово підключатися до інших пристроїв або до Інтернету. WiFi працює в частотному спектрі 2,4 ГГц та 5 ГГц. Інститут електроніки та інженерів-електронщиків (IEEE) – це асоціація, яка визначає стандарти та протоколи для комунікаційних технологій у таких галузях, як телекомунікації. IEEE використовує унікальні номери для кожного стандарту. 802 – це префікс, який використовується будь-яким протоколом або поправкою, що передбачає мережеве підключення. Наприклад, персональні мережі Bluetooth позначаються як 802.15, тоді як WLAN – як 802.11 [18]. Сьогодні більшість пристроїв та гаджетів використовують 802.11n або 802.11ac, також відомі як WiFi N та WiFi AC відповідно. WiFi N, випущений у 2007 році, має швидкість передачі даних близько 300-450 Мбіт/с залежно від кількості використовуваних антен та радіуса дії, тоді як WiFi AC з 2013 року може досягати швидкості 1 Гбіт/с. Однак, WiFi AC використовує лише діапазон частот 5 ГГц. Хоча висока частота екранує сигнал популярного діапазону 2,4 ГГц (що призводить до менших перешкод), вона значно втрачає радіус дії. WiFi N може працювати в обох діапазонах. Обидва ці стандарти мають зворотну сумісність зі старими стандартами A, B та G.

Хоча стандартний Wi-Fi забезпечує високу швидкість передачі даних, він використовує багато енергії для надсилання даних. Наприклад, Amazon IoT Button – це пристрій, який реєструє натискання кнопок і надсилає повідомлення через Інтернет до служби Amazon AWS. Хоча сам пристрій більшу частину часу перебуває в режимі сну, при натисканні він прокидається та підключається до Wi-Fi. Підключення до мережі Wi-Fi споживає багато енергії. Пристрій працює лише близько 1000 натискань, перш ніж його батарея розрядиться[20]. Це дуже неефективно.

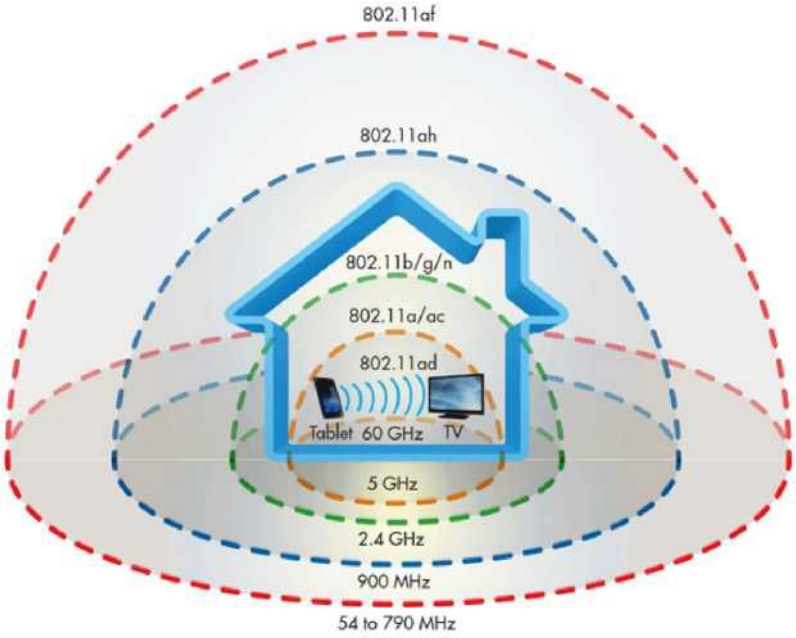

Рисунок 1.1 - Зображення не масштабовано відповідно до фактичної дальності. Джерело [19] 2.3.3 WiFi HaLow

Альянс WiFi зараз розробляє нове покоління WiFi[8]. Протокол WiFi наступного покоління називається WiFi HaLow, а його характеристики описані стандартом IEEE 802.11ah. Протокол WiFi HaLow, як і Bluetooth LE, розроблений з урахуванням потреб Інтернету речей. Він є конкурентом стандарту BLE, але ці два протоколи відрізняються тим, що WiFi HaLow дозволяє пристроям безпосередньо підключатися до Інтернету.

Специфікація

WiFi a/b/g/n/ac працює в діапазоні 2,4 ГГц, тоді як WiFi HaLow працює в діапазоні 900 МГц. Діапазон 900 МГц є нижчою частотою, тобто для передачі даних потрібно менше енергії. Сигнали WiFI HaLow можуть поширюватися далі, ніж звичайні сигнали WiFi. Очікується, що WiFi HaLow матиме радіус дії 1 км [21]. Менша пропускна здатність також означає, що сигнали краще проникають крізь стіни, а це означає, що будівлі можуть обійтися меншою кількістю ретрансляторів WiFI для своїх пристроїв Інтернету речей. Це посилює можливість розміщення пристроїв Інтернету речей у віддалених районах, оскільки вони можуть працювати на більших відстанях. WiFI Halow має

мінімальну пропускну здатність 100 Кбіт/с, що може здатися не такою великою кількістю, але цього має бути достатньо для коротких імпульсів передачі даних, які використовують пристрої Інтернету речей. Пристрої, що підтримують WiFI HaLow, повинні використовувати діапазони 2,4 ГГц, 5 ГГц та 900 МГц, тобто вони можуть взаємодіяти як з новими, так і зі старими технологіями, що робить WiFI HaLow більш доступним та простішим в інтеграції.

1.3 Пов'язані проекти

Скриптр Scriptr.io, або просто скрипт, — це хмарна платформа для Інтернету речей, мобільних та веб-додатків. Вона пропонує розробникам писати та розгортати серверні скрипти, які служать користувацькими API для їхніх рішень Інтернету речей [22]. Скрипти можуть бути написані на JavaScript або на Blocky. Blocky — це візуальна мова програмування, де користувач може перетягувати «блоки» коду для створення програм, і вона схожа на мову програмування Scratch. Scriptr пропонує веб-інтерфейс користувача та забезпечує безпечні з’єднання для пристроїв, що підключаються до сервісу за допомогою токенів автентифікації. На додаток до існуючої функціональності на JavaScript, Scriptr також пропонує різноманітні модулі, такі як керування пристроями та підключення до платформ соціальних мереж.

Libelium Waspmote Plug & Sense

Libelium — постачальник платформи бездротових сенсорних мереж для рішень Smart City та розробник Waspmote Plug & Sense (WPS) [23]. WPS — це інкапсульований бездротовий сенсорний пристрій, який дозволяє системним інтеграторам впроваджувати масштабовані та модульні бездротові сенсорні мережі [24]. Він може живитися від сонячної енергії через сонячну панель, має опції для багатьох типів технологій радіозв'язку, таких як WiFi, Zigbee, 3G, а його вісім модулів можна інтегрувати з більш ніж 60 датчиками [24]. Найголовніше, що його програму можна змінювати бездротовим способом, тобто його можна розміщувати в недоступних місцях, дозволяючи обслуговуючому персоналу

легко змінювати поточну програму. Програми, що запускаються пристроями WPS, написані на C/C++. Подібно до коду Arduino, код WPS на C/C++ також вимагає функцій налаштування та циклу. До WPS також можна отримати доступ через графічний інтерфейс програмування.

Джонні-П'ять

Johhny-Five — це платформа з відкритим кодом на JavaScript для робототехніки та Інтернету речей. Фреймворк підтримує широкий спектр мікроконтролерів та одноплатних комп'ютерів, таких як Arduino, Raspberry Pi та багато інших. Фреймворк використовує node.js для інтерпретації програм JavaScript. Johnny-Five підтримує широкий спектр датчиків та виконавчих механізмів, що використовуються пристроями Інтернету речей, таких як світлодіоди, сервоприводи, GPS, двигуни та датчики навколишнього середовища.

Тессел 2

Tessel 2 - це потужна платформа для розробки Інтернету речей та робототехніки [25]. Більш конкретно, Tessel 2 - це плата розробки з відкритим кодом та вбудованими можливостями Wi-Fi, яка дозволяє створювати скрипти в Node.js [25, 26]. Вона також здатна запускати програми Rust. Платформа пропонує офіційні та створені спільнотою модулі для різних датчиків та виконавчих механізмів [25].

Оцінювання

JavaScript — популярна мова програмування. Вона використовується в Інтернеті та настільних додатках [27], а також в Інтернеті речей [28]. JavaScript може працювати на пристроях Інтернету речей завдяки Node.js. Node.js — це кросплатформне середовище виконання JavaScript з відкритим кодом [29]. Воно використовує механізм виконання Google V8 для компіляції коду JavaScript у власний машинний код для підвищення швидкості та продуктивності [29, 30]. V8 може працювати на багатьох системах, включаючи процесори ARM. За допомогою Node.js та JavaScript стає легко писати додатки, які можуть підключатися до Інтернету та використовувати веб-сервіси безпосередньо на

пристрої Інтернету речей. JavaScipt також можна використовувати на стороні сервера. Фізичні пристрої можуть використовувати мови програмування, які працюють на нижчому рівні, такі як C. Такий підхід підвищує необхідні знання користувача, оскільки низькорівневу C важче правильно освоїти, ніж роботу з мовою з високорівневими абстракціями. Причина, чому Node.js та JavaScript не використовуються в цьому проєкті, полягає в тому, що JavaScript є великим та складним. Одна з цілей цього проєкту — створити невелику та просту мову програмування.

1.4 Висновок по розділу

Інтернет промов (IoT) відкриває нові можливості для автоматизації, збору даних та взаємодії між фізичними об'єктами, що об'єднані в єдину мережу. Завдяки широкому спектру датчиків, протоколів зв'язку та платформ розробки, IoT активно інтегрується у повсякденну життя, розширюючи функціональність побутових пристроїв, транспорту, міської інфраструктури та промисловості. Подальший розвиток цієї галузі залежить від створення енергоефективних, безпечних та масштабування.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА МОВНІ ОСОБЛИВОСТІ IoT-СИСТЕМ

2.1 Вимоги до мови програмування Інтернету речей

Цільова платформа. Мова програмування орієнтована на пристрої Інтернету речей. Ці пристрої не тільки дуже енергоефективні та зазвичай невеликі за розміром, але й обмежені в обчислювальній потужності, просторі та обсягу оперативної пам'яті. Пристрої Інтернету речей не виконують багато обробки даних і в більшості випадків лише збирають дані з навколишнього середовища та передають їх на зовнішній пристрій, такий як віддалений сервер. Пристрої Інтернету речей не обов'язково призначені для використання як універсальний комп'ютер або мікроконтролер, як Raspberry Pi чи Arduino, а радше виконують невеликі та конкретні завдання. Raspberry Pi більше підходить для загальних застосувань. Це не означає, що такі пристрої, як Raspberry Pi, не можна використовувати в Інтернеті речей, але вони можуть бути не такими ефективними у виконанні певного завдання, як спеціальний пристрій. У деяких випадках програми для вбудованих пристроїв пишуться на C, оскільки ця мова дозволяє програмісту писати швидкі програми з малим обсягом пам'яті та розміром файлу. C дає програмісту великий контроль над поведінкою програми, але це вимагає гарного розуміння апаратного забезпечення, на якому вона працює. З іншого боку, вже існують скриптові рішення для Інтернету речей, які використовують JavaScript для скриптової частини. Ці програми можуть поступитися частиною швидкості, яку забезпечує користувацький низькорівневий варіант, на користь простого налаштування та розгортання.

Цільова платформа Мова програмування орієнтована на пристрої Інтернету речей. Ці пристрої не тільки дуже енергоефективні та зазвичай невеликі за розміром, але й обмежені в обчислювальній потужності, просторі та обсягу оперативної пам'яті. Пристрої Інтернету речей не виконують багато обробки даних і в більшості випадків лише збирають дані з навколишнього середовища та передають їх на зовнішній пристрій, такий як віддалений сервер.

Пристрої Інтернету речей не обов'язково призначені для використання як універсальний комп'ютер або мікроконтролер, як Raspberry Pi чи Arduino, а радше виконують невеликі та конкретні завдання. Raspberry Pi більше підходить для загальних застосувань. Це не означає, що такі пристрої, як Raspberry Pi, не можна використовувати в Інтернеті речей, але вони можуть бути не такими ефективними у виконанні певного завдання, як спеціальний пристрій. У деяких випадках програми для вбудованих пристроїв пишуться на C, оскільки ця мова дозволяє програмісту писати швидкі програми з малим обсягом пам'яті та розміром файлу. C дає програмісту великий контроль над поведінкою програми, але це вимагає гарного розуміння апаратного забезпечення, на якому вона працює. З іншого боку, вже існують скриптові рішення для Інтернету речей, які використовують JavaScript для скриптової частини. Ці програми можуть поступитися частиною швидкості, яку забезпечує користувацький низькорівневий варіант, на користь простого налаштування та розгортання.

Основні вимоги. Мова має бути простою та зручною у використанні. Цільова аудиторія — програмісти-аматори, тому мова має бути обмежена за складністю. Це має спростити процес навчання та підвищити продуктивність користувачів.

Скрипти гарячої заміни. Ці програми – це скрипти. Пристрої Інтернету речей матимуть програмне забезпечення, яке може запускати ці скрипти. Частиною дизайну є можливість заміни скриптів новими. Це означає, що пристрій, на якому зараз виконується один скрипт, може отримати новий скрипт. Коли це відбувається, програмне забезпечення повинно перевірити, чи новий скрипт придатний для використання (тобто чи не містить він синтаксичних помилок). Якщо новий скрипт правильний, запущений скрипт зупиняється та видаляється разом з його даними (змінними тощо). Потім інтерпретатор починає виконувати новий скрипт. З іншого боку, новий скрипт видаляється, якщо інтерпретатор виявляє, що він містить помилку. Виконання старого скрипта відновлюється. Користувач повинен мати можливість завантажувати новий скрипт на потрібний пристрій незалежно від того, де він знаходиться. Це означає,

що скрипти можна завантажувати через бездротові з'єднання, такі як Wi-Fi та Bluetooth. Ці механізми роблять платформу гнучкішою та простішою у використанні. Блок-схема на рисунку 2.1 показує різні стани, в яких може перебувати інтерпретатор.

Компіляція. Компілятор здатний перевірити семантику програми перед її компіляцією у виконуваний формат. Компілятор також може створювати виконувані файли, адаптовані до певної архітектури для покращення швидкості та споживання пам'яті. Як результат, програми, що компілюються, як правило, швидкі. Залежно від мови, аналіз програми може бути використаний для виявлення помилок, які можуть виникнути під час виконання. Таким чином, компілятори можуть додавати додаткові гарантії безпеки. Наприклад, компілятор може виявляти невідповідності типів у статично типізованій мові.

Недоліком є те, що згенерований виконуваний файл (зазвичай) можна використовувати лише на одній цільовій архітектурі. Час компіляції також може бути повільним, на користь швидкого виконання.

Інтерпретація. Мова, що інтерпретується, виконується інтерпретатором. Це означає, що код не компілюється на мову, специфічну для апаратного забезпечення. Натомість він інтерпретується або безпосередньо інтерпретатором, або спочатку компілюється в байт-код. Наприклад, Java компілюється у байт-код, специфічний для Java, який може бути інтерпретований віртуальною машиною Java (JVM). Перевагою інтерпретованих програм є те, що їх можна швидко розгортати. Інтерпретовані мови також є портативними та можуть працювати на будь-якій машині, якщо підтримується архітектура цієї машини. Інтерпретовані програми зазвичай повільніші за скомпільовані. Компілятор може генерувати оптимізовані бінарні файли для апаратної платформи, тоді як інтерпретатор повинен постійно інтерпретувати програму. Тим не менш, можна оптимізувати інтерпретовані програми, генеруючи оптимізований байт-код. Інтерпретатор може виявляти сегменти коду, які часто використовуються, та витрачати деякий час на їх оптимізацію. Зрештою, інтерпретація може ставати швидшою з кожною оптимізацією.

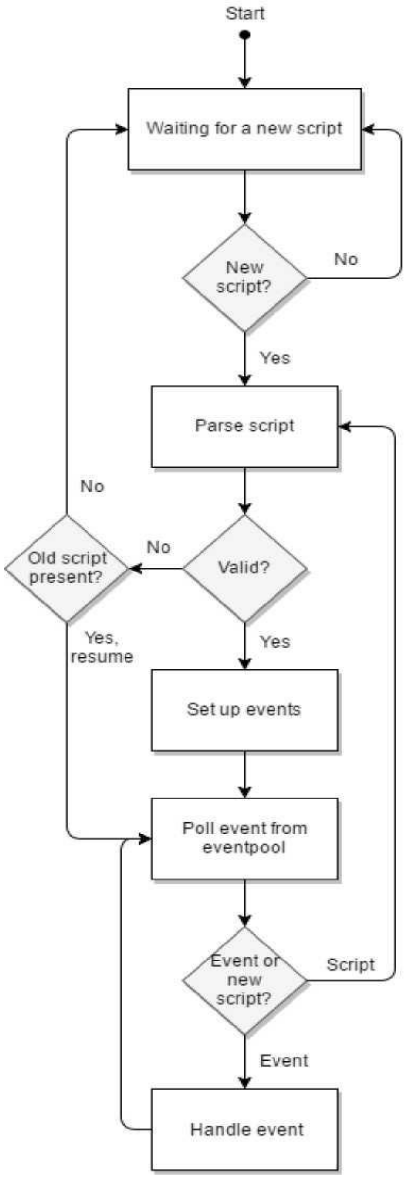

Рисунок 2.1- Блок-схема виконання програми

Оцінювання. Ця мова більше підходить як інтерпретована мова, оскільки вона використовуватиметься як мова сценаріїв. Проблема зі скомпільованими програмами полягає в тому, що їх важко перевірити на пристрої Інтернету речей. Завжди існує ймовірність того, що щось піде не так з програмою під час її надсилання на пристрій. Наприклад, не виключено, що біти переплутаються або деякі пакети з даними втрачаються. Компілятор повинен підтримувати широкий спектр різних архітектур. Тим не менш, цей аргумент може бути незрозумілим, оскільки інтерпретатор повинен мати можливість працювати на різних архітектурах. Частиною початкового дизайну було використання інтерпретатора для мови сценаріїв. Причиною була, і залишається, перевага перевірки коду на

кількох етапах розгортання. Це дозволяє як користувачеві, так і пристрою-отримувачу перевіряти сценарії. Це має зробити платформу більш надійною.

Статичне типізування

При статичній типізації користувач повинен вказати тип змінної під час її оголошення. Тип змінної не можна змінити після призначення. Це означає, що всі змінні мають свої типи, відомі під час компіляції. Компілятор може просто перевірити тип кожної змінної та повідомити користувача про невідповідність типів. Це також може запобігти помилкам, які можуть виникати під час виконання. Статична типізація може призвести до меншої кількості помилок, оскільки помилки виявляються на ранніх етапах розробки. Як приклад, розглянемо оголошення змінних у C. Цілочисельну змінну можна оголосити без присвоєння (int x;) або з одиницею (int y = 42;), але вона завжди повинна мати вказаний тип.

Динамічне типізування

Динамічна типізація дозволяє оголошувати змінні без необхідності явного оголошення їхнього типу. Змінна не пов'язана з жодним типом. Тому їй можна призначити значення іншого типу. Зазвичай інтерпретовані мови використовують динамічну типізацію, оскільки немає компілятора для перевірки типу. Будь-які помилки типу виявляються під час виконання. Нижче наведено простий приклад того, як динамічна типізація може призвести до малопомітних помилок. У більшій базі коду пошук помилок такого типу може зайняти багато часу.

```
1 def add(a, b):
2     return a + b
3
4 add(1, 2) # OK
5 add(3, "hello") # Run time error
6 add("abc", "def") # OK
```

Виведене типізування

Виведена типізація дозволяє компілятору визначити тип змінної на основі того, як вона використовується. Це дозволяє користувачам не записувати тип

змінної, але вони зберігають перевагу перевірки типу під час компіляції. Щоразу, коли компілятор не може визначити тип змінної, він завершує роботу з помилкою компілятора. Це змушує користувача вказати тип даних, щоб програма могла компілюватися. Виведена типізація належить до категорії статичної типізації, оскільки всі типи відомі під час компіляції. Як приклад: У Rust ми можемо написати наступний код

```
1 let x = 128;
2 let y: u16 = 20 + x;
```

Тут x спочатку не має типу, але компілятор знає, що це має бути ціле число. Y оголошено з типом u16, тому вираз у правій частині оператора присвоєння повинен повертати значення з типом u16. Тоді компілятор може зробити висновок, що x повинен мати тип u16. Виведена типізація складніша за звичайну статичну типізацію та важча в реалізації. Вона також не пропонує практичної користі, але може зробити код трохи менш багатослівним.

Оцінювання. Динамічна типізація означає, що користувачеві не потрібно турбуватися про оголошення типів. Динамічна типізація також має допомогти скоротити криву навчання. Розміри файлів скриптів мають бути якомога меншими. Ми можемо потенційно заощадити кілька байтів, не записуючи тип кожної змінної. Чим менше даних мають надсилати пристрої Інтернету речей, тим менше енергії вони споживають на бездротовий зв'язок. Це також означає, що ми потенційно можемо звільнити місце на диску та в оперативній пам'яті. Контраргументом на користь динамічних типів є те, що вони збільшують ймовірність помилок під час виконання через невідповідність типів. Одна з цілей мови полягає в тому, щоб вона ніколи не спричиняла збоїв, тому вибір динамічної типізації замість статичної типізації є нелогічним, оскільки остання може запобігти помилкам типів.

Більшість мов програмування, що використовуються для Інтернету речей, є імперативними мовами [28]. Це, ймовірно, найбільш використовувана парадигма, і її легко зрозуміти, оскільки легко відстежувати потік виконання програми під час читання імперативного коду. Однак існує інша парадигма програмування, яка більше підходить для Інтернету речей та для мови цього

проєкту: подієво-орієнтоване програмування. Пристрої Інтернету речей (IoT) – це пристрої, які очікують на реакцію своїх датчиків та виконавчих механізмів на зміни. Коли виникає подія, програмне забезпечення на пристрої має її обробити. Подійно-орієнтоване програмування дозволяє користувачеві писати програми, моделювання яких імітує поведінку пристрою. Це має зробити програмування IoT більш природним для користувача. Можна писати подійно-орієнтовані програми імперативними мовами, але вони можуть бути незграбними, багатослівними та не дуже інтуїтивно зрозумілими. Замість того, щоб робити обробку подій доповненням до мови, мова буде побудована навколо безпосередньої обробки подій.

Усі події будуть додані до пулу подій. Події будуть відсортовані за пріоритетом та часом запланованого виконання. Інтерпретатор опитуватиме одну подію за раз та оброблятиме її. Під час виконання обробника подій його таймаут не буде обмежено, і йому буде дозволено виконуватися до його завершення. Інтерпретатор буде однопотоковим. Це означає, що одночасно може оброблятися не більше однієї події. Це спрощує реалізацію та усуває можливість виникнення умов змагання для спільних даних між обробниками подій. Тип обробників подій варіюється від визначених користувачем до вхідних та вихідних даних від датчиків, виконавчих механізмів та пристроїв вводу-виводу.

Безпека є важливим фактором, коли йдеться про Інтернет речей. Пристрої досить вразливі, оскільки вони підключені до Інтернету. Таким чином, до них може отримати доступ зловмисник. Ці пристрої, як правило, мають мало або взагалі не мають захисту від зовнішніх атак. На жаль, механізми безпеки мови та перекладача ¬не були першочерговим завданням у цій роботі.

Помилки та винятки

Інтерпретатор повинен завжди працювати і ніколи не завершувати роботу, якщо пристрій не вимкнено. В ідеалі, інтерпретатор повинен мати можливість обробляти помилки, спричинені користувацькими скриптами, тобто він ніколи не повинен аварійно завершувати роботу, коли стикається з таким. З іншого боку, компілятори та інтерпретатори можуть виводити попередження,

якщо виявлять щось некоректне в коді, який вони аналізують. Програма, яка викликає помилку через виняток або логічну помилку, може безпечно завершитися завдяки вбудованій обробці помилок. Звичайно, деякі помилки знаходяться поза нашим контролем, тоді як інші виникають через недоліки реалізації або помилки. Наприклад, програмі може закінчитися пам'ять, вона намагається отримати доступ до даних, які виходять за межі дозволеного, вона стикається з сумнозвісним винятком нульового вказівника або просто викликає помилку сегментації. Якою б не була причина, багато з цих помилок не можуть бути оброблені самим інтерпретатором, і базова операційна система, найімовірніше, завершить роботу інтерпретатора. Пізніше ми розглянемо, як ми можемо спробувати підтримувати майже 100% безвідмовну роботу, використовуючи фоновий процес для моніторингу стану інтерпретатора.

Обробка програмного збою

Коли інтерпретатор виявляє помилку в користувацькому скрипті, він повинен обрати відповідний курс дій. Помилка може бути спричинена різними способами, залежно від реалізації та дизайну мови. Тепер ми розглянемо кілька різних способів обробки збою.

Альтернатива 1: Скидання

Інтерпретатор перезапускає інтерпретацію скрипта. Це означає, що він починає інтерпретувати скрипт з самого початку. Однак є й недоліки.

• Усі тимчасові дані втрачаються.

• Програма може знову зіткнутися з помилкою, що призведе до нескінченного циклу скидання.

• Якщо цю функцію ввімкнено, користувач отримуватиме спам-повідомлення про помилки.

Здається, що це рішення не має особливих переваг. Однак користувачеві не потрібно виконувати жодного додаткового обслуговування, оскільки скрипти просто перезапускаються. Можливо, помилка була спричинена одноразовою помилкою, а це означає, що її виникнення було очікуваним.

Альтернатива 2: Видалити скрипт

Інтерпретатор повністю видаляє скрипт із носія інформації, тобто інтерпретатор більше не має скрипта для інтерпретації. Цей сценарій також може призвести до втрати проміжних даних. Інтерпретатор переходить у стан за замовчуванням, стан очікування. У цьому стані інтерпретатор тепер повинен чекати, поки користувач надасть йому скрипт, який він може запустити. Перевага такого підходу полягає в тому, що користувач знає, що інтерпретатор безпечно зупинить інтерпретацію скрипта, як і звичайна програма, яка зіткнулася з помилкою на настільному комп'ютері. Деякі винятки можуть бути спричинені серйозними помилками, і може бути безпечніше просто зупинити програму, замість того, щоб повторно запускати той самий скрипт.

Альтернатива 3: Ігноруйте це

Інтерпретатор може ігнорувати помилку та продовжувати інтерпретацію програми. У JavaScript існує значення null, яке представляє навмисну відсутність значення об'єкта. Користувач може використовувати null в арифметичних операціях з іншими типами даних, що може призвести до незрозумілих результатів.

```
1  > 1 + null
2  1
3  > 1 * null
4  0
5  > {} + null
6  0
7  > {} * null
8  Syntax Error
9  > null * {}
10 NaN
```

Тим не менш, він передбачений задумом і може призвести до збою програми в певних ситуаціях, що може змусити програміста перевіряти наявність нульових значень. Повертаючись до інтерпретатора, ми могли б реалізувати подібну механіку, якщо мова реалізує тип null для представлення помилки або відсутності даних. Тепер, якщо інтерпретатор виявить використання значення null, він може ігнорувати його та дозволити даним пошкодитися. Однак деякі винятки спричинені серйознішими проблемами, які не можна ігнорувати. Це рішення має бути реалізовано разом з одним або обома рішеннями, обговореними у попередніх розділах.

Альтернатива 4: Завершити

Інтерпретатор може повністю завершити роботу і ніколи не перезапуститися автоматично. Це означає, що користувачеві доводиться перезапускати програму вручну. Це можна зробити різними способами, залежно від апаратного та програмного забезпечення хоста. Наприклад, якщо пристрій — це Raspberry Pi з дистрибутивом Linux, користувач може перезапустити програму або віддалено (наприклад, через ssh), або локально. Для інших пристроїв з простішими операційними системами користувачеві, можливо, доведеться фізично підійти до пристрою та перезапустити його вручну. від'єднавши та знову підключивши джерело живлення або натиснувши вимикач живлення. Оскільки ми хочемо, щоб наші пристрої Інтернету речей завжди працювали, завершення запущеного процесу та скрипта, ймовірно, завдасть більше шкоди, ніж користі.

Альтернатива 5: Інформуйте користувача

Було б корисно повідомити користувача про те, що сталася помилка. У випадку, якщо інтерпретатор завершує роботу або завершує її, він не зможе надіслати користувачеві діагностичне повідомлення, якщо тільки немає програми-спостерігача (демона), яка виявляє збій і здатна повідомляти про помилки. З іншого боку, якщо лише скрипт завершує роботу або викликає помилку, сам інтерпретатор може повідомити користувача про те, коли і що пішло не так. Тому інтерпретатор повинен знати, як і кому він повинен надіслати повідомлення. Діагностичне повідомлення повинно містити таку інформацію, як коли сталася помилка, що її спричинило та позиція в скрипті, де вона сталася.

Сторожовий демон

Демон — це програма, яка працює у фоновому режимі та не перебуває під прямим контролем користувача. У Unix-подібних системах існує демон під назвою Watchdog, який перевіряє, чи правильно працює операційна система, і може викликати скидання ядра, якщо сталася помилка. У випадку, коли інтерпретатор завершує роботу з будь-якої причини, демон-процес може бути відповідальним за запуск нового екземпляра програми-інтерпретатора. Демон

повинен мати можливість визначити, чи запущено інтерпретатор, чи він просто спить. Сам демон має бути простим, невеликим за розміром та виконувати якомога менше операцій, водночас маючи можливість правильно оцінювати стан інтерпретатора, який він спостерігає. Це робиться для зменшення споживання енергії та економії якомога більшої кількості системних ресурсів, оскільки вони вже можуть бути обмежені через поточну природу пристроїв Інтернету речей. Додавання процесу демона може вимагати, щоб пристрій Інтернету речей міг запускати операційну систему, яка може запускати кілька процесів паралельно або одночасно. У випадку, коли інтерпретатор працює як інтегрована частина системи, тобто немає базової ОС з планувальником та менеджером ресурсів, реалізація бажаного демона може стати неможливою. Хоча, можливо, можна було б імітувати поведінку демона іншими засобами.

Мова забезпечить інтерфейс для датчиків, виконавчих механізмів та вводу-виводу. Це означає, що користувачеві ніколи не доведеться турбуватися про реалізацію низькорівневої функціональності. Це покладає велику відповідальність на того, хто відповідає за реалізацію інтерпретатора, оскільки він чи вона повинні реалізувати інтерфейс для датчиків та виконавчих механізмів. Однак інтерфейс стає ефективнішим, оскільки він реалізований мовою нижчого рівня та може бути безпосередньо доступний інтерпретатору. Одна з цілей цього проєкту — створити мову, яку можна використовувати для швидкого розгортання IoT-застосунків. Запропонувавши вбудований інтерфейс для датчиків та виконавчих механізмів, ми можемо досягти цієї мети.

Стандартна бібліотека (STD) міститиме функції, які дозволять користувачеві отримувати доступ до датчиків, виконавчих механізмів та вводу-виводу. Крім того, STD міститиме функції, які дозволять користувачеві маніпулювати та взаємодіяти з вбудованими типами даних.

2.2 Проектування Даспела

Daspel — це подієво-орієнтована мова сценаріїв, призначена для пристроїв Інтернету речей. Daspel забезпечує високорівневі абстракції для пристрою, включаючи датчики, виконавчі механізми, пристрої вводу-виводу та мережевий зв'язок. Найменування мови «В інформатиці є лише дві складні речі: анулювання кешу та іменування речей». - [21] За допомогою Ветле Волден-Фреберга я визначився з назвою для мови сценаріїв: Daspel. Назва походить від Event-Driven Actuator & Sensor Programming Language (мови програмування виконавчих механізмів та датчиків, керованих подіями), або скорочено EDASPL. Однак EDASPL нелегко вимовляти англомовним користувачам, тому літеру E було перенесено з початку між P та L.

Синтаксис натхненний синтаксисом C, JavaScript та Rust. Daspel написаний так, щоб нагадувати імперативні мови програмування з точки зору структури функцій та блоків.

У Daspel є 5 основних типів даних.

- ціл.
- справжній
- логічна змінна
- рядок
- список

Також існує тип nil, який використовується для значень помилок. З п'яти типів три можна вважати примітивними: int, real та bool. String та list є структурами даних і мають складнішу реалізацію. У наступних розділах посилатися на типи даних буде використано моноширинний шрифт. Це зроблено лише для візуального розділення типів даних та понять з однаковою назвою.

У Daspel int — це 32-бітне ціле число, доповнене до двох зі знаком. Таким чином, воно повинно охоплювати діапазон чисел, необхідний для простих обчислень, які виконуватиме пристрій Інтернету речей. Звичайно, не кожен пристрій Інтернету речей має 32-бітний процесор, оскільки вони також можуть бути 16-бітними або 8-бітними. Однак 16-бітний процесор повністю здатний обробляти 32-бітні та 64-бітні дані, але при цьому витрачає більше часу на

додаткові інструкції порівняно з 16-бітними даними. 32-бітний розмір — це поширений розмір для цілих чисел, який часто є розміром за замовчуванням. Але що ж, як користувач хотів би зробити, використовуючи менші цілі числа для економії місця? Динамічна система типів трохи спростила б визначення більш конкретного розміру цілого числа в розрядах, на відміну від статичної системи типів. В інших мовах програмування зазвичай немає способу вказати розмір цілого числа, не вдаючись до створення нової змінної або приведення типів. Однак Daspel міг би запозичити цілочисельну анотацію, що використовується в Rust. Синтаксис полягає в додаванні типу даних в кінці числа. Наприклад, 100i32 — це знакове 32-бітове ціле число зі значенням 100, тоді як 9u8 — це беззнакове 8-бітове ціле число зі значенням 9. Цю функцію можна було б додати, але заради простоти та через обмеження в часі вона не була додана в цій версії Daspel.

Термін "дійсні числа" використовується для представлення десяткових чисел. Існує два підтипи дійсних чисел: числа з фіксованою комою та числа з плаваючою комою. Daspel використовує нотацію з фіксованою комою. Тип даних, який представляє десяткові числа, називається дійсним. Візуально між цими двома типами немає жодної різниці. Однак числа з фіксованою комою зберігаються як цілі числа. Числа з фіксованою комою використовують цілочисельну арифметику на відміну від арифметики з плаваючою комою. Це дає їм перевагу, коли йдеться про швидкість обчислень. На пристроях, які не мають процесора з плаваючою комою [32] (FPU), арифметика з плаваючою комою дуже повільна. Для цього проекту передбачається, що пристрої Інтернету речей не мають FPU. Числа з плаваючою комою також мають вищу точність , ніж числа з фіксованою комою, оскільки кількість бітів, що використовуються для представлення дробів, є більшою для чисел з плаваючою комою. Однак припускається, що користувачеві Daspel не знадобиться повна точність числа з плаваючою комою. Набагато зручніше використовувати швидшу фіксовану кому.

Дійсне число в Даспелі — це 32-бітове число з фіксованою комою, доповнене до двох зі знаком. Воно використовує 16 бітів для представлення цілої

частини та 16 бітів для представлення дробової частини. Ми називаємо цей формат Q-форматом [33], який використовується для чисел з фіксованою комою. Іншими словами, дійсне число має формат Q16.16. Точніше, це Q15.16, оскільки один біт використовується для знака. 16-бітні дроби можуть представляти десяткові числа з точністю щонайменше до 0,9999. На рисунку 2.1 показано, як зберігаються цілочисельні та дробові біти. Як і у випадку з числами з плаваючою комою, точність може втрачатися під час множення та ділення. Це відбувається тому, що не завжди достатньо місця для представлення повного діапазону дробових розрядів.

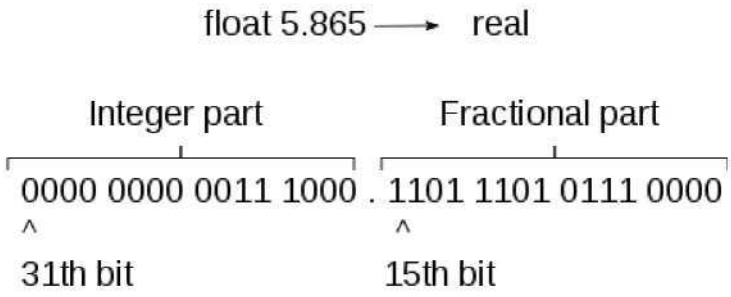

Рисунок 2.2 Зберігання цілочисельних та дробових бітів

```
1 let pi = 3.14
2 let r = 2;
3 let area = pi * r * r;
```

Рисунок 2.2 - зберігання цілочисельних та дробових біт

Булеве значення — це чітке представлення умови, яке використовується в умовних структурах. Вони роблять наміри програміста зрозумілішими та можуть зробити код більш читабельним. Інші мови програмування, такі як C та JS, дозволяють цілим числам представляти булеві значення. Будь-яке ненульове ціле число є істиною, а 0 — хибністю. Цей підхід можна розглядати як форму неявного перетворення типів, що дозволяє одному типу представляти кілька форм даних одночасно. У цьому підході немає нічого поганого, але в Daspel ми вирішили розділити дуальність цілих чисел у C і натомість розділити їх на два окремі типи. Якби Daspel не містив умовних структур, йому більше не потрібен був би булевий тип.

Рядки в Daspel — це контейнери динамічного розміру для тексту, закодованого в Unicode. Тип рядка використовує кодування Unicode замість звичайного ASCII. Це зроблено для того, щоб людям, які не володіють англійською мовою, було зручніше надсилати та отримувати текст мовою, якою вони розмовляють, зі свого пристрою Інтернету речей. На певному етапі процесу розробки емодзі та інші символи Unicode вважалися частиною синтаксису. Тому було логічно зробити рядковий текст також з підтримкою Unicode.

Динамічний розмір рядка визначається динамічно, тобто він може збільшуватися та зменшуватися. Розмір буде автоматично оброблятися інтерпретатором. Daspel міг би використовувати рядки фіксованого розміру, але тоді користувач повинен переконатися, що рядок достатньо великий, коли вставляє в нього нові дані. Daspel має бути максимально простим, а завдяки використанню рядків з динамічним розміром у користувача на одну проблему менше.

Daspel не має окремого типу даних для окремих символів. Це пояснюється тим, що символ можна виразити як рядок , і це зменшує складність мови. Звичайно, це означає, що Daspel трохи менш ефективний, оскільки йому доводиться розміщувати окремі рядки символів у купі, а не в стеку.

Рядок використовує той самий синтаксис, що й більшість інших мов програмування. У цьому прикладі символи Юнікоду не показані, оскільки L A T E X не підтримує символи Юнікоду.

```
1 let s = "this is a string";
2 let empty_string = "";
```

Список – це колекція, яка може зберігати будь-які типи даних Daspel. Так само, як і рядки, список розподіляє дані в купі. Це дозволяє їй збільшуватися та зменшуватися, коли елементи додаються до списку та видаляються зі списку. Розподіл обробляється автоматично інтерпретатором. Проблема зі списками фіксованого розміру полягає в тому, що користувачеві доводиться бути обережним під час додавання нових елементів. Користувач повинен

переконатися, що список завжди достатньо великий для розміщення нових елементів. Якщо це не так, користувач повинен виділити більший список. Натомість, list виконує цю роботу за вас автоматично. Це, звичайно, підкріплюється метою зробити Daspel якомога простішим. Через обмеження в часі синтаксис для додавання та видалення елементів до списку не було додано. Однак, найімовірніше, він буде схожим на ті, що використовуються в імплантації Python.

Список має такий самий синтаксис, як і списки в Python. Кожен елемент у списку розділяється комою.

```
1 let positive = [1, 2, 3, 4, 5];
2 let any = ["hello", 13, true];
3 let nested = [1, [2], [[3]]];
4 let empty_list = [];
```

nil — це тип помилки в Daspel. Він представляє відсутність значення. Він також використовується для представлення помилки. Цей тип був включений, щоб функції, які зчитують дані датчиків та виконавчих механізмів, могли повертати певний тип помилки. Наприклад, якщо датчик температури не підключений, а програма намагається зчитати температуру, функція просто поверне nil замість дійсного числа . Замість nil можна використовувати значення за замовчуванням. Тож у наведеному вище прикладі замість повернення nil функція може повернути 0.0. Це проблематично, оскільки 0.0 також є допустимим значенням. Тому наявність спеціального значення помилки робить помилки більш явними та легшими для виявлення.

Як і багато інших мов програмування, Daspel має змінні. Змінна - це іменований контейнер, який зберігає значення. У цьому розділі ми обговоримо, як оголошуються змінні. У розділі про область видимості ми розглянемо, де можна оголошувати змінні. Ключове слово для оголошення змінних у Daspel - let.

Як і багато інших мов програмування, Daspel має змінні. Змінна - це іменований контейнер, який зберігає значення. У цьому розділі ми обговоримо,

як оголошуються змінні. У розділі про область видимості ми розглянемо, де можна оголошувати змінні. Ключове слово для оголошення змінних у Daspel - let.

```
let x = 32;
x = x + 20 / 3;
```

У Python ми можемо оголосити змінну без використання ключового слова.

```
x = 1
if condition
    x = 2
```

```
x = 32
x = x + 20 / 3
```

Однак, існує проблема з реалізацією на Python. Оскільки немає різниці між оголошенням змінної та присвоєнням, ми отримуємо семантичну неоднозначність. Розглянемо наступний код, написаний на Python. Можливо, користувач мав намір оголосити нову змінну x в операторі if, але замість цього він змінив вищевказану змінну x.

```
let x = 1;
if (condition) {
    x = 3; // Mutates the x above
    let x = 2; // Shadows x
}
```

Досвідченим програмістам на Python це може здатися тривіальною проблемою, але серед початківців це може призвести до плутанини та помилок під час виконання. Завдяки явному оголошенню змінних ми можемо уникнути таких помилок. Щоб уникнути такої неоднозначності, Daspel вимагає явного оголошення змінних. Це дає нам чітке розмежування між оголошенням та присвоєнням.

У розділі визначення області видимості ми обговоримо, де можна оголошувати змінні. Наразі просто знайте, що змінні можна оголошувати лише на початку кожної області видимості. Цей розділ називається частиною оголошення змінних. Оскільки оголошення змінних можна виконувати лише

один раз у кожній області видимості, легко перевірити наявність неправильно розміщених оголошень змінних. Частина оголошення змінної складається з трьох варіантів. Якщо змінних для оголошення немає, частина оголошення або порожня, або повністю пропускається. Другий варіант складається з одного оголошення. Останній варіант складається з кількох оголошень. Питання полягає в наступному: як ми можемо представити частину оголошення змінної та її три стани простим та елегантним способом? Одним із рішень є вимога окремого оголошення кожної змінної, як показано нижче.

```
# single
let x = true;

# empty
let a = 1,
    b = "heat",
    c = [4, 3, 2, 1];
```

```
# empty
let {}

# single
let { x = true }

# multiple
let {
    a = 1,
    b = "heat",
    c = [4, 3, 2, 1],
}
```

Рисунок 2.3 Вимога окремого оголошення кожної змінної

Наступним рішенням є групування всіх оголошень в одному блоці. Блок може бути порожнім або пропущеним, якщо немає локальних змінних. Третя альтернатива - це ланцюжок оголошення. Щось подібне можна зробити в мовах програмування, подібних до C. Остання пропозиція є найпростішим альтернативним варіантом, який також вимагає найменшої кількості символів. Таким чином, це найкращий синтаксис для частини оголошення змінних.

Цей розділ здебільшого стосується змінних, визначених в області видимості функції. Коли викликається користувацька функція, вона може оголошувати деякі змінні. Після завершення виклику функції виділені дані

потрібно очистити. У деяких мовах програмування це робиться збирачем сміття. У Daspel немає ручного керування пам'яттю, але є автоматичне керування пам'яттю. Типи даних рядків та списків розподіляють дані в купі. Щоб уникнути переповнення купи до точки, коли в ній закінчується місце, Daspel потрібно очистити виділені дані. Тому, коли змінні, що вказують на рядкові та спискові дані, виходять за межі області видимості, інтерпретатор автоматично звільняє дані в купі.

У цьому розділі ми розглянемо правила визначення області видимості в Daspel. Існує загалом три області видимості: область видимості бібліотеки, глобальна область видимості та область видимості функції. Область видимості бібліотеки є найкоротшою, за нею йде глобальна область видимості. Область видимості функції може існувати лише всередині глобальної області видимості. Існує лише одна бібліотека та глобальна область видимості, але користувач може визначити багато функцій і, отже, багато областей видимості функцій. Однак область видимості функції не може бути вкладеною. Іншими словами, ви не можете визначати функції всередині функцій.

Область видимості змінних визначає, де можна оголошувати змінні та звідки до них можна отримати доступ. Під час проектування було розглянуто два варіанти оголошення змінних. Перший дизайн запозичений з імперативних мов і дозволяє оголошувати змінні будь-де в області видимості. На рисунку 2.3 ми бачимо, як оголошуються змінні в статично типізованій мові та як довго вони живуть залежно від області видимості, в якій вони визначені. Цей дизайн дещо ускладнює деякі речі, оскільки інтерпретатор повинен точно знати, де в області видимості оголошується змінна. Це робиться для того, щоб вирази та оператори не використовували змінні, які технічно ще не визначені. Другий варіант просто вимагає оголошення всіх змінних на початку кожної області видимості. Хоча це не є обов'язковим, ця практика не є незвичайною в програмах на C, де деякі змінні повторно використовуються для різних цілей, таких як лічильники в циклах for. Примусове розміщення оголошень змінних на початку кожної області видимості дещо спрощує проектування парсера. Це також дає зрозуміти, скільки

змінних використовується в кожній області видимості. З цих причин було вирішено, що змінні мають бути оголошені в частині оголошення змінних.

Область видимості бібліотеки є найвищою та містить функції, що знаходяться у стандартній бібліотеці. До цієї області видимості можна отримати доступ з двох інших областей видимості. Зверніть увагу, що ці функції та змінні, визначені в цій області видимості, не можуть бути змінені користувачем через скрипт Daspel, оскільки це визначено в реалізації інтерпретатора .

```
{
    let x = 1, y = 2;
    # Some code...
    let z = 3; # <- Illegal! Must be declared at the top
}
```

```
var global; // Lives in the global scope.
            // Goes "out of scope" when the
            // program terminates.
{
    var x;
    ... // Some code...
    var y;
    {
        ...
        var x; // Shadows variable x.
               // x in the scope above is untouched.
        var z;
    } // x and z go out of scope here.
} // x and y go out of scope here.
```

Рисунок 2.4: Приклад оголошення змінних у мові зі статичною системою типів

Це область видимості, яку бачить користувач, коли пише програму на Daspel. У цій області видимості можна оголошувати змінні та функції. Змінні мають бути оголошені на початку області видимості в частині оголошення змінних. Будь-яка функція, визначена в цій області видимості, може бути доступна з інших функцій, незалежно від порядку, в якому вони були визначені. Іншими словами, Daspel уникає проблеми, яка виникає в C, коли функція не може отримати доступ до інших функцій, визначених нижче, якщо користувач не оголошує прототипи функцій на початку файлу або у файлі заголовка.

Область видимості функції — це область видимості, яка існує у функціях. Ця область видимості доступна лише для функції-власника, тобто жодні зовнішні функції не можуть ЯГРСС ifc rrvn+mi+c

```
# The global scope
# Cannot see what's inside
# foo or bar
let glob = 10;

fn foo() {
    # Can see glob and bar,
    # but cannot see bar's zap
}

fn bar() {
    # Can see glob and bar
    # Can see zap
    let zap = [];
}
```

Лістинг 2.1 - Область видимості функції

Daspel містить стандартні арифметичні та логічні оператори. Типи даних у Daspel не можна використовувати разом із цими операторами. Винятком є те, що int та real можна використовувати разом в арифметичних операціях. При змішуванні цих двох типів результуюче значення завжди буде дійсним числом. Це подібно до того, як це роблять інші мови програмування, включаючи Python. Причина, чому типи не можуть змішуватися та використовуватися разом операторами, полягає в тому, що не завжди зрозуміло, якою має бути відповідь. Наприклад,

- плюс +
- мінус –
- множення *
- відділ /
- дорівнює ==
- не дорівнює ! = • ні!
- більше ніж >
- більше або дорівнює > =

- менше ніж <
- менше або дорівнює < =

Яким має бути результат true + 3 - "hello" ? Це неоднозначно і потенційно може створювати дивні значення, які можуть лише заплутати користувача та призвести до помилок. Існує спеціальний оператор, який використовується для перевірки значень nil. Він називається оператором знака питання ? і схожий на метод nil? в Ruby. Додаючи ? до значення або змінної, Daspel виконає перевірку значення на nil. Якщо воно дорівнює nil, повертається значення true, інакше повертається значення false.

```
let x = nil;
if x? {
    # ...
}
```

Лістинг 2.2 - Яким має бути результат true + 3 - "hello"

Daspel має типи операторів: while-loop, for-loop, if-else та функції. Тільки функції можна визначити в глобальній області видимості. Оператори while-loop, for-loop та if-else можуть бути визначені в області видимості функцій, і вони можуть бути вкладеними. 1 2 3 Функції не можуть бути вкладені всередину функцій або інших операторів.

Цикл for схожий на цикл у мові програмування Rust. Структура визначає значення акумулятора та діапазон для ітерації. Діапазон позначається двома цілими числами , розділеними двома крапками (..). Цикл for використовує фігурні дужки для фіксації тіла циклу. Нотація діапазону виражає виключний діапазон. Точніше: <початок>..<виключна-зупинка>. Число ліворуч – це перше значення, присвоєне акумулятору. Цикл for збільшуватиме значення акумулятора на одиницю для кожного проходу. Він зупиняється, коли значення акумулятора стає рівним значенню праворуч. Наприклад, 0..5 дає 1,2,3,4. Використовуючи три крапки (...), діапазон стає включним, тобто він виконує ще один крок перед зупинкою. Наприклад, 0...5 дає 1,2,3,4,5. На даний момент немає синтаксису для визначення кроку взаємодії.

```
while t > 3 {
    t /= 2;
}
```

```
if x > 10 {
    # ...
} else if x == 2 {
    # ...
} else {
    # ...
}
```

Лістинг 2.3 - Цикл for схожий на цикл у мові програмування Rust.

Цикл while такий самий, як і в інших мовах програмування. Він складається з умовного виразу та тіла, оточеного фігурними дужками. Умовний вираз повинен повертати логічне значення.

Функції в Daspel оголошуються за допомогою ключового слова fn, після якого вказується ім'я функції та параметри. Оператор функції не оголошує жодного типу повернення, оскільки Daspel має динамічну типізацію. Таким чином, оператор функції нагадує той, що знаходиться в Python.

Функції використовуються для створення обробників подій. Додаючи анотацію перед функцією, Daspel зрозуміє, що це обробник подій. Анотації використовуються для визначення того, яку подію має обробляти функція або як часто її слід викликати. Анотації починаються з символу @, за яким йде ім'я та необов'язкова умова. В анотаціях користувач може вказати поріг для тригера події. Наприклад, щоб створити обробник, який викликається лише тоді, коли температура перевищує 20 градусів Цельсія, ми пишемо наступне:

```
@Temperature > 30 C;
fn handler() {
    # handle event
}
```

Лістинг 2.4 – Функції в Daspel

Варіативна функція — це функція, яка приймає змінну кількість аргументів [34]. Можна стверджувати, що варіативні функції не є необхідними в

динамічно типізованій мові, оскільки списки можуть містити будь-які типи, і такі списки можуть імітувати варіативну функціональність.

```
fn variadic(*args) {
    # args is a list
}

variadic(1, 2, 3);
variadic(2);


fn normal(list) {
    # takes a single argument
}

normal([1, 2, 3]);
normal([2]);
```

Лістинг 2.5 - Варіативна функція

Кортеж - це скінченний список елементів [35]. Кортежі можна використовувати для повернення кількох значень з функцій. Однак, списки можна використовувати для тієї ж мети, тому кортежі не додадуть нічого цінного до Daspel.

Дозвіл користувачеві додавати тип до параметрів функцій полегшив би розуміння того, якими мають бути аргументи. Це також зменшило б кількість помилок перевірки типів під час виконання, оскільки вони будуть виявлені під час перевірки типів. Однак, скрипти Daspel мають бути дуже маленькими, тому в будь-якому випадку легко визначити типи параметрів. Користувач також може просто використовувати коментарі для документування типу параметрів.

У мові програмування Rust примітивні типи даних реалізують певні методи. Це також стосується складних структур даних, таких як рядки та вектори. Це також можна застосувати до Daspel. Стандартні бібліотечні функції для взаємодії та маніпулювання типами даних Daspel можна було б реалізувати як методи. Це зробило б мову більш об'єктно-орієнтованою, на відміну від поточного поєднання імперативного, подієво-орієнтованого та функціонального.

```
# std function
let x = [2, 3, 1, 4];
sort(x);
# x is now [1, 2, 3, 4];

# built in method
x.sort();
```

Лістинг 2.6 – Стандартні бібліотечні функції для взаємодії та маніпулювання типами даних Daspel

2.3 Символи Unicode

Мій керівник запропонував використовувати символи Unicode для ключових слів та назв функцій. Ідея полягала в тому, щоб використовувати символ Unicode як альтернативний спосіб позначення ключових слів, функцій, вводу-виводу, датчиків та виконавчих механізмів. Довгі ключові слова використовують більше байтів, але символ Unicode використовує 1-4 байти, залежно від символу. Скрипти, які використовують символи Unicode, у більшості випадків можуть бути меншими, а отже, зменшувати обсяг даних, які потрібно передавати на пристрій Інтернету речей. Це також означає, що вони займають менше місця в оперативній пам'яті та на диску пристрою Інтернету речей. Нижче ви знайдете таблиці символів Unicode. Зверніть увагу, що для кожного поняття використовуватиметься лише один символ. Кожна таблиця показує лише альтернативи. Unicode як ключові слова ніколи не потрапив до специфікації мови, принаймні на цьому етапі. Усі символи були знайдені за адресою https: //unicode-table.com/en/ та http://getemoji.com/ .

Таблиця 2.1:

«Надіслати» означає надіслати повідомлення. Лист або бульбашка з текстом – це гарне візуальне представлення для надсилання чогось.

Send	
U+1F4E7	U+2709
U+1F583	U+1F582
U+1F4EE	U+1F4AC

Таблиця 2.2:

Перш ніж зупинитися на динамічній типізації, типи потрібно було прописати.

Nil		
U+2205	U+2300	U+00D8
U+2298	U+2297	U+2BBE
U+29B8	U+1F6AB	U+274C
U+2717	U+2613	U+1F5F4
U+1F5F6	U+1F5F5	U+1F5F7
U+00D7	U+1F5D9	U+2A2F
U+1F480	U+2620	U+1F571
U+26A0	U+26D4	U+1F635
U+1F44E	U+1F4A3	U+1F6A7
U+1F4A9	U+2757	

У цій таблиці наведено символи для значення nil. nil представляє помилку або порожнє значення, тому перевага надавалася Unicode, що зображує негативні символи.

Таблиця 2.3:

Ключове слово або функція power повертатиме значення, яке відображає, скільки заряду залишилося в батареї

Power	
U+1F50B	U+1F5F2
U+26A1	U+1F50C

Таблиця 2.4:

Годинник представляє час, а функція часу переведе пристрій у режим сну на заданий період

Time/Timer	
U+23F0	U+23F1
U+23F2	U+1F570
U+231B	U+23F3

Таблиця 2.5:

Мікрофон, вухо та навушники чітко вказують, який датчик використовувати

Sound (microphone)		
U+1F3A4	U+1F508	U+1F509
U+1F50A	U+1F3B5	U+266A
U+266B	U+1F3B6	U+1F39C
U+1F3A7	U+1F442	

Таблиця 2.6:

Відтворення звуку.

Sound (speaker)		
U+1F3A4	U+1F508	U+1F509
U+1F50A	U+1F3B5	U+266A
U+266B	U+1F3B6	U+1F39C
U+1F399	U+1F444	

Таблиця 2.7:

Планета Земля та супутники найкраще підходили для GPS.

GPS	
U+1F310	U+1F30E
U+1F30D	U+1F30F
U+1F5FA	U+1F6F0
U+1F6F0	U+1F4E1

Humidity	
U+1F4A7	U+1F4A6

Було важко знайти хороший символ для позначення вологості. Краплі води були найточнішими, оскільки вологість відображає кількість водяної пари в повітрі.

Тиск також було важко зобразити. У цьому випадку тиск відображає тиск повітря, тому хмари та стрілки, спрямовані вниз, були найкращим, що я зміг знайти

Pressure		
U+1F32C	U+1F300	U+2601
U+1F327	U+26C5	U+1F326
U+1F32A	U+1F32B	U+1F301
U+1F38F	U+2193	U+21E9
U+2B07	U+1F83B	U+1F847
U+1F89B		

Таблиця 2.8:

Датчик світла вимірює світло

Light (sensor)
U+2600
U+263C
U+1F308

Він не випромінює світло, але сприймає його, тому знайти точне представлення було складно

Таблиця 2.9:

Монітор досить простий у використанні

Monitor
U+1F4FA
U+1F5B5
U+1F4BB
U+1F5B3

Таблиця 2.10:

Кнопки «Увімкнено» та «Вимкнено» представляють дію ввімкнення або вимкнення пристрою. Вони також символізують дії «прийняти» та «відхилити»

On/Off		
U+1F5F8	U+2713	U+2714
U+2716	U+2717	U+2718
U+1F197	U+270B	U+1F44D
U+1F44E	U+1F592	U+1F593
U+1F44C		

Таблиця 2.11:

Мисляче обличчя символізує вибір. Проблема полягала в тому, що не було знайдено жодних хороших альтернатив для else if та else .

IF-ELSE
U+1F914

Таблиця 2.12

Замість запису for або while користувач використовуватиме символ Unicode

Loop	
↻ U+21BA	↻ U+21BB
⟲ U+27F2	⟳ U+27F3
⥀ U+2940	⥁ U+2941
⭯ U+267B	

Таблиця 2.13:

Замість запису for або while користувач використовуватиме символ Unicode

Function	
λ U+03BB	𝛌 U+1D6CC
𝝺 U+1D77A	𝑓 U+1D453
𝒇 U+1D487	

2.4 Висновок по розділу

У цьому розділі окреслено ключові вимоги до мови програмування для Інтернету промов, зокрема простоту, енергоефективність, динамічну типізацію та подієво-орієнтовану модель. Враховуючи обмежені ресурси IoT-пристроїв, інтерпретована мова зі вбудованим інтерфейсом до сенсорів та виконавчих механізмів дозволяє створити гнутку та надежну среду для швидкого розгортання застосунків.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ IoT-ПРОЕКТУ

3.1 Доказ концепції

Метою реалізації є створення підтвердження концепції для інтерпретатора Daspel. У наступних розділах ми розглянемо апаратне забезпечення, на якому працюватиме інтерпретатор. Ми також розглянемо надані датчики та зміну пікселів на дисплеї. Нарешті, ми розглянемо реалізацію коду.

У цій роботі як платформа для розробки та запуску інтерпретатора використовується Raspberry Pi 3 Model B (RPi). Датчики додаткової плати Sense HAT використовуються для тестування показників середовища, що зчитуються скриптами.

Інтерпретатор складається з кількох частин.

- Сканування, парсинг та аналіз скриптів Daspel.
- Генерація абстрактного синтаксичного дерева.
- Налаштування черги подій.
- Опитує подію з черги та інтерпретує її.

По суті, коли пристрій отримує новий скрипт, він спочатку має його проаналізувати. Інтерпретатор сканує новий скрипт і перевіряє його на наявність синтаксичних помилок. Після цього генерується Це дерево є структурою даних, яка представляє скрипт. Інтерпретатор використовує AST для міркувань про програму. Звідси інтерпретатор повинен перевірити, чи скрипт (AST) не містить жодних неприпустимих дій, таких як виклик невизначеної функції або використання невизначеної змінної. Потім ініціалізується пул подій. Пул містить події, які може запускати інтерпретатор. Події від датчиків, скрипта та вводу-виводу поміщаються в пул подій.

3.2 Raspberry Pi та Sense HAT

Raspberry Pi 3 — це Raspberry Pi третього покоління, що входить до серії невеликих одноплатних комп'ютерів розміром приблизно з кредитну картку [26, 27]. RPi — це потужний пристрій, повністю здатний запускати операційну систему, таку як дистрибутив Linux. Пристрій, наданий для цієї роботі, постачався з SD-картою з встановленою операційною системою Raspbian. Raspbian базується на операційній системі Debian та оптимізований для апаратного забезпечення Raspberry Pi [28]. Він постачається з набором програмних інструментів, які можна використовувати для навчання, програмування та загального використання [26]. Повний список апаратних специфікацій Raspberry Pi можна знайти на рисунку 3.1.

Рисунок 3.1 - Raspberry Pi 3 Model B.

- 1,2 ГГц 64-бітний чотириядерний процесор ARMv8
- Бездротова локальна мережа 802.11n
- Bluetooth 4.1
- Низькоенергетичний Bluetooth (BLE)
- 1 ГБ оперативної пам'яті
- 4 USB-порти
- 40 контактів GPIO

- Повноцінний порт HDMI
- Порт Ethernet
- Комбінований 3,5-мм аудіороз'єм та композитний відеовхід
- Інтерфейс камери (CSI)
- Інтерфейс дисплея (DSI)
- Слот для карт Micro SD
- 3D-графічне ядро VideoCore IV

Для цього проєкту нам довелося завантажити та встановити додаткове програмне забезпечення, щоб мати змогу реалізувати інтерпретатор Daspel. Двома найважливішими програмними компонентами були бібліотека Sense HAT для Python, яка постачається з бібліотекою RTIMULib C++, та компілятор Rust. Компілятор Rust було встановлено за допомогою rustup, який є інсталятором Rust у ланцюжку інструментів. Rustup також надає Cargo, менеджер пакетів для Rust, який керує зовнішніми залежностями для проєкту коду. Іншими інструментами, необхідними для розробки на RPi, були SSH, GIT та Vim, але потрібно було встановити лише Vim, оскільки перші два вже були попередньо встановлені Raspbian.

Sense Hat — це додаткова плата для Raspberry Pi, розроблена спеціально для місії Astro Pi [20]. Плата оснащена світлодіодним дисплеєм, джойстиком та шістьма датчиками, доступ до яких можна отримати через бібліотеку Python. Повний список функцій можна знайти на рисунку 3.2.

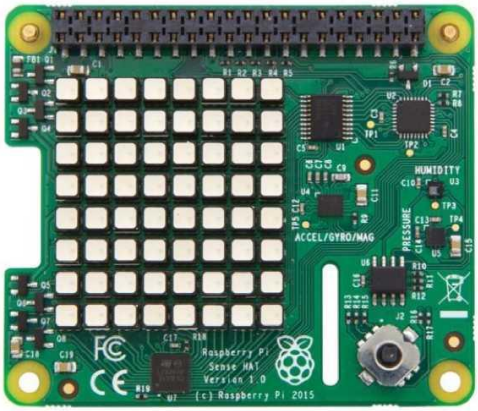

Рисунок 3.2 - Sense HAT.

Як датчики, так і матриця світлодіодів RGB 8x8 використовують протокол I²C для зв'язку. На RPi протокол I²C має бути ввімкнено, перш ніж можна буде використовувати датчики Sense HAT. Ядро Linux надає функції C для зв'язку I²C , що дозволяє відносно легко написати власну функцію читання та запису в Sense HAT.

Хоча на веб-сайті Sense HAT зазначено шість різних датчиків, плата має лише три сенсорні мікросхеми. Датчики тиску – це датчики LPS25H, датчики відносної вологості – HTS221, а датчики 9 ступенів свободи (DoF) – LSM9DS1 [22]. Як датчики LPS25H, так і HTS221 здатні зчитувати температуру за тиском та відносною вологістю відповідно. LSM9DS1 – це IMU, що розшифровується як Inertial Measurement Unit (Інерційний вимірювальний блок) [23]. IMU насправді складається з трьох датчиків: 3D-акселерометра, 3D-гіроскопа та 3D-магнітометра. Користувач може використовувати дані з IMU для визначення руху, який виявляє Sense HAT [43]. Зображення на рисунку 3.4 показує три осі, які може виявити Sense HAT.

- LPS25H (тиск)
- HTS221 (Вологість)
- LSM9DS1 (Акселерометр, гіроскоп та магнітометр)

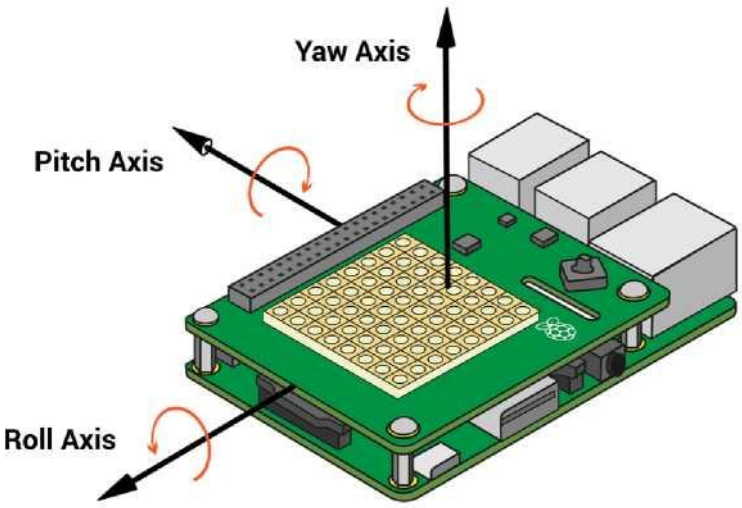

Рисунок 3.3 - Орієнтація

Світлодіодна матриця 8x8 RGB керується LED2472G, підключеним до Atmel ATTINY88, який взаємодіє через I²C з RPi [42]. Світлодіодна матриця має частоту оновлення 60 кадрів/с та роздільну здатність кольору 15 біт [44]. Для використання світлодіодної матриці не потрібно безпосередньо використовувати протокол I²C . Натомість, буфер кадру дисплея можна зчитувати з файлу або відображати в пам'ять за допомогою mmap. Буфер кадру - це частина оперативної пам'яті, що містить растрове зображення, яке використовується для оновлення відео з буфера пам'яті, що містить повний кадр даних [45]. У бібліотеці Sense HAT Python буфер кадру спочатку зчитується з файлу. Потім у файл записуються нові дані, які, у свою чергу, оновлюють пікселі на світлодіодній матриці.

High Byte								Low Byte							
R7	R6	R5	R4	R3	G7	G6	G5	G4	G3	G2	B7	B6	B5	B4	B3

Рисунок 3.5 - Поле RGB 565 біт після кодування значення RGB 888 бібліотекою Python.

Пікселі на світлодіодному дисплеї використовують кольоровий формат RGB 565. Кожен піксель потребує 16 біт даних, де 5 бітів використовуються для значень червоного та синього кольорів, а 6 бітів – для значень зеленого. На рисунку 7.4 ми бачимо, як кожен піксель форматується бібліотекою Python Sense HAT. Більш конкретно, бібліотека приймає кольори, відформатовані як RGB 888, який є стандартним кольоровим форматом, що використовується в Інтернеті [6], і стискає їх до формату RGB 565. Оскільки RGB 565 на 1 байт менший за RGB 888 (24-бітний), деякі дані втрачаються під час перетворення. Червоні та сині значення втрачають свої 3 молодші біти, тоді як зелені значення втрачають лише 2.

API Python Sense HAT забезпечує простий у використанні інтерфейс для плати Sense HAT. Однак доступ до даних датчиків здійснюється не безпосередньо через API, а через бібліотеку C++ під назвою RTIMULib [7], яка підтримується Raspberry Foundation. Бібліотека написана на об'єктно-

орієнтованому C++ та використовує специфічні для Linux бібліотеки C для зв'язку через I²C. RTIMULib здатна виявляти кілька видів датчиків навколишнього середовища та IMU, включаючи ті, що знаходяться на платі Sense HAT. RTIMULib також створює файл налаштувань, який дозволяє користувачеві змінювати налаштування для кожного датчика.

- Світлодіодна матриця RGB 8x8
- Джойстик з п'ятьма кнопками
- Гіроскоп
- Акселерометр
- Магнітометр
- Температура
- Атмосферний тиск
- Вологість

Через конструкцію плати Sense HAT неможливо зчитати точну температуру з Sense HAT, якщо вона встановлена безпосередньо над RPi. Показники температури показуватимуть значення, які значно вищі за очікувані. Це спричинено теплом, що генерується процесором RPi, а гаряче повітря навколо RPi впливає на показання датчиків. Щоб протидіяти цьому явищу, Sense HAT довелося б відокремити від RPi або відкалібрувати показання. Однак точні показання не були першочерговим завданням для реалізації , тому це питання було проігноровано.

3.3 Впровадження

Критерії для інтерпретатора Даспеля такі:

- Розмір бінарного файлу має бути якомога меншим.
- Він повинен займати невеликий обсяг пам'яті.
- Він повинен споживати якомога менше енергії.
- Він повинен мати можливість працювати на будь-якому пристрої Інтернету речей.

- Він повинен мати можливість перемикати запущені скрипти.
- Він має бути міцним, тобто ніколи не повинен руйнуватися.

Першим кроком написання інтерпретатора Daspel є вибір мови програмування. Виходячи з критеріїв, наведених у розділі вище, ми з моїм керівником вирішили не використовувати мови, які використовують збирання сміття для управління пам'яттю. Це виключає такі мови, як Python та Java, або інтерпретатори, такі як Node.js. Одна з причин полягає в тому, що ці мови потребують більшого обсягу пам'яті, ніж мови з ручним управлінням пам'яттю. Друга причина полягає в тому, що збирання сміття займає час і в деяких реалізаціях може призвести до паузи програми під час виконання. Пристрої Інтернету речей використовують системи, що працюють у режимі реального часу, тому пауза для звільнення пам'яті - це не той сценарій, з яким ми хочемо мати справу. Тим не менш, Java є найпопулярнішою мовою програмування для розробки Інтернету речей, а JavaScript посідає третє місце [28]. Можливо, що пристрої Інтернету речей, які працюють на Java, працюють нарівні з Raspberry Pi, тому пам'ять не є такою вже й великою проблемою. В інших випадках на сервері та шлюзі використовуються Java та JavaScript, тобто пристрій Інтернету речей не використовує ці мови безпосередньо. З іншого боку, можна було б зв'язатися з пристроєм Sense HAT через протокол I2C з Java, JS та Python через зовнішні залежності. Це робить можливим... Враховуючи часові обмеження на дисертацію, бажано, щоб ¬мова реалізації дозволяла легко представляти рядки та списки динамічних розмірів. Мова або її екосистема (сторонні бібліотеки) повинні забезпечувати підтримку тексту ¬Unicode. Чим менше часу витрачається на їх вирішення, тим більше часу можна витратити на реалізацію.

Python коротко розглядався, хоча ця мова інтерпретується та використовує автоматичне керування пам'яттю. Причиною вибору Python є те, що Sense HAT API написано на Python. Це означало б, що потрібно було б реалізувати лише сканер, парсер та інтерпретатор.

C є другою за популярністю мовою програмування у світі [49]. C також є другою за популярністю мовою програмування, що використовується в

Інтернеті речей [28]. Мова C невелика та проста. Вона займає мало пам'яті, має дуже хороші характеристики та може генерувати невеликі бінарні файли. Мова дозволяє розробнику тісно взаємодіяти з апаратним забезпеченням. Як бачимо, C є дуже перспективним вибором. Вона ідеально відповідає вимогам. Вона також портативна та, найімовірніше, підтримуватиме більше платформ, ніж її конкуренти. Однак її не обрали з наступних причин. Спочатку планувалося використовувати C як мову реалізації інтерпретатора Daspel. Однак я не є досвідченим програмістом на C і не дуже люблю C. Зізнаюся, це одна з причин, чому я не хотів використовувати C. Я також не хотів витрачати час на реалізацію динамічно змінених списків та рядків на C.

Крім того, мені також довелося б мати справу з ручним керуванням пам'яттю. Під цим я маю на увазі, що мені довелося б відстежувати розподіл malloc та переконатися, що я використовую free у потрібних місцях. Я здебільшого використовую Java під час навчання в університеті, тому маю дуже мало досвіду з ручним керуванням пам'яттю. Щоб було зрозуміло, я знаю про інструменти, які можуть перевіряти витоки пам'яті, використовувати після free та undefined поведінку в програмах на C. Ці інструменти, безумовно, були б дуже корисними. Списки можна легко реалізувати за допомогою зв'язаних списків. Рядки, з іншого боку, повинні підтримувати символи Unicode. Як наслідок, синтаксичний аналізатор також повинен мати можливість розбирати символи Unicode. Daspel має динамічну типізацію, тому інтерпретатор повинен мати можливість представляти динамічні типи. Найпростіший спосіб зробити це – використовувати теговані об'єднання, як показано на рисунку 8.1. За допомогою тегованих об'єднань стає можливим представляти кілька типів даних за допомогою однієї структури. Це також дає змогу змінити динамічну змінну на інший тип, змінивши значення об'єднання та помінявши тег перерахування. Проблема полягає в тому, що мені довелося б бути обережним і завжди перевіряти тег перед зміною значення об'єднання, оскільки значення може бути вказівником.

```c
#include "stdio.h"
#include "stdbool.h"

typedef enum TypeKind {
    Integer, Boolean, String
} TypeKind;

typedef struct {
    enum TypeKind kind;
    union Value {
        int num;
        bool boolean;
        char *str;
    } value;
} Type;

Type type_new_int(int v) {
    Type t;
    t.kind = Int;
    t.value.num = v;
    return t;
}
```

Рисунок 3.4 - Теговане об'єднання у мові C

Зрештою, RTIMULib не може використовуватися C безпосередньо, оскільки вона реалізована в C++. C не може викликати функції C++. Це пов'язано зі зміною імен функцій та методів, яку виконує компілятор C++. Використання конструкції extern "C" запобігатиме зміні імен функцій компілятором C++. Одним із рішень є переписування необхідних частин RTIMULib на C. Швидшим рішенням є написання обгортки на C. Обгортка діє як інтерфейс, який дозволяє C здійснювати непрямі виклики функцій C++. Обгортка — це просто заголовковий файл C, функція якого реалізована на C++. Як код C++, так і обгортка компілюються разом компілятором C++, таким як gcc або clang. Програма AC може потім просто використовувати заголовковий файл і викликати заголовкові функції, як звичайні функції C.

C++ Як і C, C++ є популярною мовою, що використовується для IoT-застосунків, і вона посідає 6-е місце в списку популярності [28]. З C++ я міг би написати інтерпретатор майже так само, як і з C. Головною перевагою використання C++ є те, що бібліотеку RTIMULib можна використовувати

безпосередньо. C++ також має типи std::String та std::Vector, визначені в стандартній бібліотеці. Обидва ці типи можуть виділяти дані в купі та автоматично змінювати розмір. Ці типи також реалізують метод деструктора, що означає, що коли змінна, яка володіє std::String або std::Vector, виходить за межі області видимості, виділені дані автоматично звільняються. Це спрощує реалізацію рядкових та спискових типів Daspels. Проблема полягає в тому, що ці типи неможливо використовувати всередині об'єднання, оскільки об'єднання підтримують лише примітивні типи даних. Щоб вирішити цю проблему, мені довелося б використовувати бібліотеку Boost та використовувати тип Any. Тип Any здатний містити складніші типи даних та забезпечує безпечний спосіб перевірки типу значення. Зрештою, я не обрав C++, бо не хотів покладатися на бібліотеку Boost для реалізації, і не був впевнений, що вона взагалі компілюється на Raspberry Pi. Крім того, у мене немає жодного досвіду програмування на C++ і я дещо упереджений проти C++. Тим не менш, я вважаю C++ хорошим кандидатом завдяки своїй портативності, швидкості та обсягу пам'яті.

Rust - це мова системного програмування, розроблена Mozilla. Її перший стабільний реліз відбувся у 2015 році [20]. Rust прагне конкурувати з C++ за продуктивністю, безпекою та споживанням пам'яті [21]. Безпека в цьому контексті означає, що Rust гарантує надійність типів, безпеку пам'яті та запобігає перегонам даних. Компілятор Rust запобігає більшості, якщо не всім, небезпечним операціям під час компіляції, використовуючи статичний аналіз та відстежуючи власників даних. У Rust є String та Vec, які є структурами даних, подібними до std::String та std::Vector. Різниця полягає в тому, що рядки в Rust кодуються в UTF-8. Rust також має вбудовані теговані об'єднання, які називаються Enum. На відміну від об'єднань C/C++, перелічення в Rust можуть зберігати будь-який тип, будь то структура, вказівник або звичайний тип даних. Ці перелічення працюють подібно до типу даних у Standard ML. Rust може викликати функції в коді C, але не може викликати функції C++.

Щоб реалізація Rust працювала з RTIMULib, вона повинна або використовувати обгортку C, або перереалізувати RTIMULib в Rust. Компілятор

Rust покладається на серверну частину компілятора LLVM для перетворення свого проміжного представлення (IR) на асемблер [22]. Компілятор C/C++ , Clang, також використовує LLVM. Таким чином, можливо, що код Rust та C++ можна компілювати в подібні бінарні файли. Проблема з компілятором Rust полягає в тому, що використовується статичне зв'язування замість динамічного, що може призвести до більших бінарних файлів. Це може бути проблемою для певних платформ Інтернету речей з дуже обмеженою ємністю дискового простору. Окрім цього, Rust зазвичай працює подібно до коду C++ з точки зору швидкості виконання. Деякі бенчмарки Порівняння Rust, C++ та C можна знайти на сайті The Computer Language Benchmarks Game . !!!!! ДІЛЯНКА КОРОБКИ ТУТ!!!!! Зрештою, я обрав Rust з причин, згаданих вище, а також тому, що я більше знайомий з програмуванням на Rust. Rust самостійно обробляє більшу частину, якщо не все, управління пам'яттю, і він забезпечує безпечні методи кодування завдяки статичному аналізу під час компіляції.

Існує щонайменше два способи взаємодії з Raspberry Pi. Перший – підключити монітор, клавіатуру та комп’ютерну мишу безпосередньо до RPi. Це дозволяє використовувати RPi як звичайний комп’ютер, оскільки операційна система Raspbian має графічний інтерфейс користувача. Робоче середовище можна вимкнути, якщо ви бажаєте безпосередньо використовувати інтерфейс командного рядка. Другий підхід полягає у підключенні до Raspberry Pi через SSH через віддалений комп'ютер. Це легко налаштувати вдома, де можна встановити статичну IP-адресу у вашій особистій мережі. З іншого боку, Raspberry Pi, здається, не може підключитися до Eduroam WiFi на IFI. Рішенням є використання кабелю Ethernet, підключення RPi до комп'ютера та запуск SSH через локальне з'єднання. Для Windows існує програма під назвою MobaXterm, яка має графічний інтерфейс користувача для SSH-з'єднань. Вона також може відкривати віддалені файли в локальному редакторі. Це дозволяє використовувати середовища програмування у Windows для запису у файли програм, розташовані на Raspberry Pi, якщо ви не хочете використовувати Vim або Emacs через звичайний SSH. Коли я вперше почав працювати над Raspberry

Pi, я використовував перший підхід. Через деякий час стало занадто важко розмістити на одному столі дві клавіатури, дві комп'ютерні миші та два монітори, тому я перейшов на SSH. Я почав використовувати MobaXterm після того, як почав працювати над реалізацією на Rust.

Єдиним важливим програмним забезпеченням, яке потрібно завантажити, є компілятор Rust, оскільки він не входить до дистрибутива Raspbian. Для тестування та підтвердження того, що код Rust правильно взаємодіє з пристроєм Sense HAT, використовується модуль Python Sense HAT. Цей модуль також потрібно завантажити, але на відміну від Rust, він доступний через менеджер пакетів Raspbian. Бібліотека RTIMULib встановлюється разом з модулем Python. Модуль Python встановлює кілька прикладів програм, які показують, як використовувати модуль Python та бібліотеку C++ RTIMULib. Ці програми були корисними для розуміння роботи з Sense HAT.

Єдиним важливим програмним забезпеченням, яке потрібно завантажити, є компілятор Rust, оскільки він не входить до дистрибутива Raspbian. Для тестування та підтвердження того, що код Rust правильно взаємодіє з пристроєм Sense HAT, використовується модуль Python Sense HAT. Цей модуль також потрібно завантажити, але на відміну від Rust, він доступний через менеджер пакетів Raspbian. Бібліотека RTIMULib встановлюється разом з модулем Python. Модуль Python встановлює кілька прикладів програм, які показують, як використовувати модуль Python та бібліотеку C++ RTIMULib. Ці програми були корисними для розуміння роботи з Sense HAT.

Як зазначалося раніше, код C не може безпосередньо викликати функції C++. Щоб мати змогу використовувати бібліотеку C++ з C, спочатку потрібно створити обгортку. За допомогою обгортки стає можливим конвертувати типи C++ у типи C. Обгортка надає інтерфейс, який код C може бачити та використовувати для взаємодії з кодом C++. По суті, C просто викликає оголошені функції-обгортки, оголошені у заголовковому файлі. Фактична реалізація цих функцій знаходиться у файлах C++, які відповідають домовленостям C++. Оскільки в C не існує класів, обгортка повинна

використовувати спеціальні вказівники, які називаються непрозорими вказівниками, які C може використовувати замість них. Для спрощення створюється один клас Wrapper. Цей клас містить усі чотири об'єкти, необхідні для використання Sense HAT, з методами для зчитування та запису даних на датчики та IMU. Оскільки C не має концепції класів, вона також не має концепції методів. Неможливо використовувати непрозорий вказівник класу-обгортки як об'єкт C++ безпосередньо в C. Натомість обгортка оголошує функції C, які приймають непрозорий вказівник Wrapper як параметр. Потім функції перетворюють непрозорий вказівник на відповідний клас C++ та викликають відповідний метод, перш ніж остаточно повернути значення. Існує проблема з викликом коду C++ з C: винятки. C не має поняття винятків, не кажучи вже про винятки C++. Таким чином, якщо C++ викличе один виняток, код C не зможе його обробити. У найкращому випадку програма просто завершить роботу. Щоб запобігти цьому, функції-обгортки повинні намагатися перехоплювати всі винятки, як під час перетворення непрозорого вказівника, так і під час виклику методів об'єкта. Реалізація обгортки C ніколи не просувалася настільки далеко, щоб забезпечити хороший механізм передачі повідомлення до коду C про помилки, які могли виникнути. Методи класу Wrapper імітували реалізацію модуля Sense HAT на Python, і обгортка справді працювала. Озираючись назад, можна було б перенести реалізацію C++ майже безпосередньо на код C, замість написання обгортки. RTIMULib використовує функції C для читання та запису через протокол I2C, і все калібрування та обчислення можна легко реалізувати на C. Однак це може зайняти деякий час, оскільки для зчитування IMU потрібно багато калібрування та обчислень. Датчики тиску та вологості порівняно досить прості у взаємодії.

Модуль Sense HAT реалізує функціональність для взаємодії зі світлодіодною матрицею. Він може записувати по одному пікселю за раз, заповнювати весь екран або очищувати його, одночасно обертаючи поточне зображення. Він також здатний відображати зображення та текст розміром 8x8 пікселів. Ми розглянемо два способи доступу до буфера кадру світлодіодної

матриці. Перший підхід використовується модулем Python і передбачає запис у буфер кадру у вигляді файлу, тоді як підхід на C полягає у відображенні буфера кадру в пам'ять. Модуль Python спочатку знаходить і зберігає шлях до буфера кадру під час свого запуску. Він ніколи не записує нові піксельні дані безпосередньо у буфер кадру. Натомість він записує всі нові дані у двовимірну матрицю NumPy, яка діє як поточний кадр. Модуль записує цей кадр у буфер кадру так само, як він записував би дані у звичайний файл. Пристрої Sense HAT автоматично оновлюють матрицю світлодіодів новим кадром.

Варто зазначити, що файл буфера кадру відкривається повторно щоразу, коли програма оновлює кадр. Часті виклики функцій малювання можуть призвести до уповільнення часу виконання, оскільки програмі доводиться чекати на операції вводу-виводу для зчитування файлу. Завдяки відображенню пам'яті ми можемо записувати дані безпосередньо у фреймбуфер. Приклад використання фреймбуфера на C можна знайти у наданій грі Snake, яка знаходиться в папці з прикладами Sense HAT. Гра використовує світлодіодну матрицю для відображення поточного стану гри. Підхід схожий на модуль Python, оскільки спочатку потрібно знайти правильний фреймбуфер. Як фрейм використовується структура з 16-бітним масивом 8x8.

Потім фреймбуфер відображається в пам'ять на структуру, яка розташована на стеку. Після цього програма може отримати доступ до будь-якого індексу фрейму та безпосередньо читати або записувати його дані. Будь-яка зміна, внесена до масиву, призводить до зміни фреймбуфера, що, у свою чергу, змінює зображення, що відображається в даний момент. Для реалізації на C я вирішив використовувати постійне відображення пам'яті. Це означає, що буфер кадру відображається в пам'яті протягом усього часу виконання програми. Це тому, що я мав на меті максимально швидкий доступ до буфера кадру. Однак це може бути не найкращим рішенням, оскільки цільові пристрої мають дуже мало доступної пам'яті. Це також неякісне рішення, якщо дисплей ніколи не використовується скриптом. Кращим рішенням було б, щоб інтерпретатор

обчислював, як часто скрипт виводить зображення на екран. Тоді він зможе визначити, чи слід відкривати буфер кадру постійно чи ні.

Реалізацію Rust для взаємодії зі світлодіодною матрицею було створено паралельно з реалізацією на C. Виявилося, що вже існує модуль Rust для взаємодії з буфером кадрів. Цей модуль використовує обгортку Rust для функцій C, таких як mmap, та надає інтерфейс для запису в область відображення пам'яті. Переписування функцій C з моєї реалізації Sense HAT на C було досить тривіальним. Обидві реалізації зрештою мали однакову функціональність. У певний момент реалізація переключила фокус з C на Rust. На той час лише код на C міг взаємодіяти з датчиками завдяки обгортці RTIMULib. Реалізація Rust мала використовувати обгортку на C, оскільки Rust не має інтерфейсу зовнішніх функцій (FFI) для C++. Це означає, що Rust повинен використовувати код на C для взаємодії з C++, тоді як реалізація на C взаємодіє з C++ безпосередньо. Наприкінці лютого було випущено модуль Rust для Sense HAT [53]. У поточному випуску він забезпечує лише функціональність для зчитування тиску та вологості. Цей модуль був розгалужений, і була додана функціональність буфера кадрів. Була розпочата робота над додаванням можливості зчитування даних IMU, але на цей момент часу було обмаль, і модуль Rust Sense HAT довелося залишити незавершеним. Варто зазначити, що модуль Rust Sense Hat стороннього виробника не використовує обгортку для бібліотеки RTIMULib, а натомість копіює реалізацію C++ RTIMULib безпосередньо. Іншими словами, він використовує протокол I2C для зв'язку з датчиками та світлодіодною матрицею на платі Sense HAT.

Перший етап інтерпретації починається з перетворення тексту програми в абстрактне синтаксичне дерево (AST). Це завдання сканера. Перший крок - перетворити підрядки на токени, пропускаючи пробіли, символи нового рядка та коментарі. Завданням сканера є виконання сканування та генерація токенів. Парсер використовує токени для генерації дерева розбору (AST). Він відповідає за перевірку того, що програма дотримується правильного синтаксису. Будь-які помилки, що виникають на цьому етапі, повертають помилку. Наприклад, сканер

може знайти недопустимий символ, тоді як сканер може виявити недопустиму послідовність токенів.

Жодної роботи над сканером не велося, доки синтаксис Daspel не почав формуватися. На цей момент мова реалізації змінилася на Rust. Деякі приклади сканерів, написаних на Rust, були знайдені в Інтернеті та використані для натхнення. Основна механіка сканера полягає в тому, що він поводиться як ітератор. Вхідна програма зчитується з файлу та зберігається у рядку. Об'єкт сканера приймає рядкове посилання на вхідний рядок та створює ітератор рядка, який можна переглянути . Символ у Rust - це 32-бітний символ, закодований у UTF-8. Більш конкретно, символ - це скалярне значення Unicode [54]. Сканер має метод під назвою next_token , який повертає токен. Токен - це тип переліку, який може представляти ключові слова, типи даних та ідентифікатори. Коли викликається next_token , сканер спочатку переглядає наступний символ у файлі. Потім він виконує зіставлення зі зразком для символу, щоб визначити, яку дію виконати.

Наприклад, якщо він знаходить ", то він знає, що це початок рядкового літерала. Потім сканер викликає метод, відповідальний за сканування рядкових літералів, який, у свою чергу, повертає токен рядкового літерала. Сканування завершується, коли внутрішній ітератор вичерпується, що відбувається, коли досягнуто кінця файлу (кінця вхідного рядка). Сканер дуже модульний, що дозволяє легко додавати або видаляти токени для пошуку. Сканер зчитує весь файл у рядок, замість того, щоб зчитувати його рядок за рядком. Проблема полягає в тому, що йому потрібно виділити достатньо місця для розміщення всього вхідного поля, замість того, щоб просто виділяти по одному рядку за раз. Однак, якщо весь скрипт не містить символів нового рядка, весь файл все одно буде зчитуватися в оперативну пам'ять. Ця потенційна проблема не є проблемою для Raspberry Pi, враховуючи його характеристики, але для дуже маленького пристрою вона може стати проблемою, якщо скрипти можуть стати дуже великими за розміром. Друга проблема полягає в тому, що поточна реалізація сканера не дозволяє йому перемотувати назад.

Це означає, що після того, як якийсь символ або текст було прочитано, його не можна прочитати знову. Це пов'язано з тим, як працюють ітератори. Щоб обійти цю проблему, потрібно змінити спосіб ітерації сканера по рядку. Замість створення ітератора символів , сканер може відстежувати поточну позицію та позицію останнього токена. Він переміщує індекс поточної позиції на основі предиката, а символи між двома індексами є підрядком. Згенеровані токени базуються на семантичному значенні підрядків. Саме такий підхід використовують лексери мов Go та Rust. Такий підхід дозволяє сканеру легше повертатися назад та переглядати вперед, ніж наша реалізація, але він також повинен враховувати символи Unicode під час сканування.

Парсер так і не був завершений, оскільки не залишилося достатньо часу для його реалізації. Я почав працювати над двома різними реалізаціями: рукописним парсером та генератором парсерів. Перший парсер мав просто приймати рядкове посилання (вхідного файлу) та внутрішньо створювати об'єкт сканера. Потім парсер використовував би сканер для генерації токенів. Потім він використовував би токени для визначення поточної синтаксичної конструкції. Наприклад, токен Token::Function означатиме, що парсер переглядає початок оператора функції. Потім парсер викличе відповідний метод обробника синтаксису, який перевірить вхідну послідовність токенів. Якщо послідовність синтаксично правильна, метод поверне об'єкт AST. Цей парсер є рекурсивним пристойним парсером. Це означає, що синтаксичні методи використовують рекурсію для обходу вхідних даних. Таким чином, реалізація буде нагадувати нотацію EBNF синтаксису Daspel. Другий варіант парсера створюється за допомогою nom crat e [25], який є комбінатором парсерів.

Nom дозволяє користувачеві створювати невеликі парсери та об'єднувати їх для створення більшого парсера, що робиться за допомогою макросів Rust. Коли макроси розширюються під час компіляції, вони генерують великий кінцевий автомат, який може розібрати фрагмент u8 (char* у C). Використання nom робить сканер зайвим, оскільки nom діє як сканер, так і парсер. Макроси парсера зрештою нагадують нотацію EBNF, яку вони представляють. Недоліком

використання nom є те, що він розширюється до дуже великого кінцевого автомата. Точніше, він генерує багато рядків операторів зіставлення зі зразком. В результаті бінарний файл може стати більшим, ніж бінарний файл рукописного сканера та парсера. З іншого боку, стає відносно легко додавати нові парсери для кожної мовної конструкції. Цей парсер у своєму поточному стані здатний лише розбирати вирази.

Просту реалізацію для розбору дійсних чисел було додано, коли я працював над сканером. Тип даних real представлений у реалізації як структура. Структура називається Real і має один член типу i32. i32 - це знакове двокомпонентне ціле число в Rust. Зверніть увагу, що в Rust немає вбудованого типу real , лише числа з плаваючою комою. Структура має метод, який приймає рядок як аргумент і повертає Real. Рядок має бути текстовим представленням десяткового числа.

Арифметичні операції з дійсними числами досить тривіальні. Віднімання та додавання виконуються так само, як і зі звичайними цілими числами. Додаткових операцій не потрібно. З іншого боку, множення та ділення трохи складніші. Числа з фіксованою комою можна представити у форматі чисел Q [23]. Наше дійсне число — це число з фіксованою комою у форматі Q15.16. Коли ми множимо два дійсних числа, формат добутку стає Q30.32 [26]. Це означає, що добуток вимагає 62 біти, що, очевидно, набагато більше, ніж може представити наш i32. Щоб впоратися з цим, нам потрібно привести обидва множники до i64, помножити їх, зсунути добуток на 16 бітів праворуч і, нарешті, привести його назад до i32. При діленні, якщо ми просто поділимо два дійсних числа, ми втратимо дробову частину. Процес майже ідентичний множенню. Ми приводимо обидва числа до i64, зсуваємо ділене на 16 бітів ліворуч, ділимо і, нарешті, приводимо частку до i32. На жаль, як множення, так і ділення можуть призвести до втрати дробових розрядів, тобто втрати точності. Як ви могли помітити, ми використовуємо лише цілочисельні операції. Це означає, що операції виконуються швидко, набагато швидше, ніж арифметика з плаваючою комою, особливо якщо пристрій не має графічного процесора (FPU). Тип Real можна

використовувати в арифметичних операціях разом із числами типу i32 та f32. Це спрощує роботу з цілими та дійсними числами під час інтерпретації, оскільки Daspel дозволяє використовувати ці два типи даних разом в арифметичних операціях. Змінну a типу Real можна використовувати так: a + 3- (2 * a) або 3.14 + a + a - 1.2. Це працює, оскільки Rust перетворить арифметичну операцію на виклик методу. Наприклад, a + 3 стає a.add(3).

3.4 Висновок по розділу

Реалізація інтерпретатора Daspel на базі Raspberry Pi та Sense HAT продемонструвала ефективність використання Rust як мови програмування для створення компактного, енергоефективного та безпечного середовища виконання скриптів на пристроях Інтернету мов. Вибраний підхід дозволив успішно взаємодіяти з сенсорами, обробляти події та працювати з LED-дисплеєм, водночас забезпечуючи належний рівень продуктивності та стабільності. Досвід створення обгортки для виклику C++-бібліотеки з інших мов підкресливши важливість гнучкого вибору інструментів для специфічних цілей IoT-проє

ВИСНОВКИ

Дослідження, проведене в рамках цієї роботи, підкреслює, що програмування Інтернету речей (IoT) є ключовим елементом у створенні інноваційних рішень, які трансформують сучасний світ через інтеграцію фізичних пристроїв і цифрових систем. Аналіз основ IoT, включаючи його визначення, ключові принципи та пов'язані проєкти, показав, що успішна розробка IoT-систем вимагає глибокого розуміння як апаратного, так і програмного забезпечення. Розглянуті проєкти демонструють різноманітність застосувань IoT, від автоматизації дому до промислових рішень, що підкреслює універсальність цієї технології.

Розгляд проєктування та мовних особливостей IoT-систем виявив, що вибір мови програмування та інструментів є критично важливим для забезпечення ефективності, безпеки та масштабованості. Вимоги до мов програмування для IoT, такі як підтримка вбудованих систем, низьке енергоспоживання та висока продуктивність, визначають необхідність використання спеціалізованих підходів, таких як проєктування Даспела. Використання стандартів, наприклад, символів Unicode, забезпечує сумісність і інтероперабельність між різними пристроями та платформами.

Практична реалізація IoT-проєкту, описана в третьому розділі, продемонструвала, як теоретичні знання можуть бути застосовані на практиці. Доказ концепції, використання платформи Raspberry Pi із Sense HAT і процес впровадження показали, що розробка IoT-систем вимагає не лише програмних навичок, але й розуміння апаратних обмежень і мережевих протоколів. Цей розділ підкреслює важливість інтеграції апаратного та програмного забезпечення для створення функціональних і надійних IoT-рішень.

Узагальнюючи, програмування Інтернету речей відкриває нові можливості для автоматизації, оптимізації та інновацій у різних галузях. Результати дослідження можуть бути використані розробниками, інженерами та дослідниками для створення ефективних IoT-систем, які відповідають сучасним

вимогам безпеки, продуктивності та масштабованості. Перспективи подальших досліджень включають інтеграцію штучного інтелекту для аналізу даних із IoT-пристроїв, розробку енергоефективних алгоритмів і вдосконалення протоколів безпеки для захисту від кіберзагроз. Ці напрямки сприятимуть розвитку наступного покоління IoT-систем, здатних відповідати викликам майбутнього.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bahga A., Madisetti V. *Internet of Things: A Hands-On Approach.* — VPT, 2014. — 446 с. — Режим доступу: https://www.vpt.edu/books/internet-of-things

2. Burd B. *Raspberry Pi IoT Projects: Prototyping Experiments for Makers.* - Apress, 2021. - 300 с. - Режим доступу: https://www.apress.com/gp/book/9781484269107

3. Cirani S., Ferrari G., Veltri L. *Internet of Things: Architectures, Protocols and Standards.* — Wiley, 2018. — 408 с. — Режим доступу: https://www.wiley.com/en-us/Internet+of+Things-p-9781119359678

4. Donat W. *Learn Raspberry Pi Programming with Python.* — 2nd ed. — Apress, 2018. — 389 с. — Режим доступу: https://www.apress.com/gp/book/9781484237687

5. Doukas C. *Building Internet of Things with the Arduino.* — CreateSpace, 2012. — 336 с. — Режим доступу: https://www.amazon.com/Building-Internet-Things-Arduino/dp/B007ZNGVPW

6. Greengard S. *The Internet of Things.* — MIT Press, 2021. — 232 с. — Режим доступу: https://mitpress.mit.edu/books/internet-things

7. Hanes D., Salgueiro G., Grossetete P. *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things.* — Cisco Press, 2017. — 576 с. — Режим доступу: https://www.ciscopress.com/store/iot-fundamentals-networking-technologies-protocols-9781587144561

8. Herrero R. *Programming the Internet of Things: An Introduction to Building Integrated IoT Solutions.* — O'Reilly Media, 2021. — 350 с. — Режим доступу: https://www.oreilly.com/library/view/programming-the-internet/9781492081401/

9. Karvinen T., Karvinen K. *Getting Started with Sensors: Measure the World with Electronics, Arduino, and Raspberry Pi.* — Maker Media, 2014. — 202 с.

— Режим доступу: https://www.makershed.com/products/getting-started-with-sensors

10. Kaur A., Gupta P. *IoT Security: Challenges and Solutions.* — CRC Press, 2022. — 312 с. — Режим доступу: https://www.taylorfrancis.com/books/9780367506889

11. Kurniawan A. *IoT Projects with Arduino and ESP8266.* — PE Press, 2020. — 250 с. — Режим доступу: https://www.amazon.com/IoT-Projects-Arduino-ESP8266-Kurniawan/dp/B08B3N8Z5P

12. Lea P. *Internet of Things for Architects.* - Packt Publishing, 2018. — 524 с. — Режим доступу: https://www.packtpub.com/product/internet-of-things-for-architects/9781788470599

13. McEwen A., Cassimally H. *Designing the Internet of Things.* — Wiley, 2013. - 338 с. - Режим доступу: https://www.wiley.com/en-us/Designing+the+Internet+of+Things-p-9781118430620

14. Monk S. *Programming Arduino: Getting Started with Sketches.* — 3rd ed. — McGraw-Hill, 2020. - 192 с. - Режим доступу: https://www.mhprofessional.com/9781260488241-usa-programming-arduino-getting-started-with-sketches-third-edition

15. Norris D. *The Internet of Things: Do-It-Yourself at Home Projects for Arduino, Raspberry Pi, and BeagleBone Black.* — McGraw-Hill, 2015. — 352 с. — Режим доступу: https://www.mhprofessional.com/9780071835206-usa-the-internet-of-things-do-it-yourself-at-home-projects-for-arduino-raspberry-pi-and-beaglebone-black

16. Pfister C. *Getting Started with the Internet of Things.* — O'Reilly Media, 2011. — 194 с. — Режим доступу: https://www.oreilly.com/library/view/getting-started-with/9781449303570/

17. Rayes A., Salam S. *Internet of Things From Hype to Reality: The Road to Digitization.* — 2nd ed. — Springer, 2019. — 373 с. — Режим доступу: https://www.springer.com/gp/book/9783030116125

18. Schwartz M. *Internet of Things with ESP8266.* — Packt Publishing, 2016. — 188 с. — Режим доступу: https://www.packtpub.com/product/internet-of-things-with-esp8266/9781786466679

19. Shovic J. *Raspberry Pi IoT Projects.* — Apress, 2016. — 240 с. — Режим доступу: https://www.apress.com/gp/book/9781484213780

20. Slama D., Puhlmann F., Morrish J. *Enterprise IoT: Strategies and Best Practices for Connected Products and Services.* — O'Reilly Media, 2015. — 492 с. — Режим доступу: https://www.oreilly.com/library/view/enterprise-iot/9781491924839/

21. Uckelmann D., Harrison M., Michahelles F. *Architecting the Internet of Things.* — Springer, 2011. — 356 с. — Режим доступу: https://www.springer.com/gp/book/9783642191565

22. Vermesan O., Friess P. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems.* — River Publishers, 2013. — 364 с. — Режим доступу: https://www.riverpublishers.com/book_details.php?book_id=201

23. Watt A. *Programming the Raspberry Pi: Getting Started with Python.* — 3rd ed. — McGraw-Hill, 2021. — 208 с. — Режим доступу: https://www.mhprofessional.com/9781264257355-usa-programming-the-raspberry-pi-getting-started-with-python-third-edition

24. Youssef M. *IoT Programming with Python.* — Independently Published, 2022. — 280 с. — Режим доступу: https://www.amazon.com/IoT-Programming-Python-Mohamed-Youssef/dp/B09TZY5K9P

25. Zeadally S., Tsikerdekis M. *Securing Internet of Things (IoT) with Machine Learning.* — IEEE Internet of Things Journal, 2020. — Vol. 7, No. 10. — С. 9378–9388. — Режим доступу: https://doi.org/10.1109/JIOT.2020.300

26. Arduino Team. *Arduino Official Documentation: Programming IoT Devices.* — 2024. — Режим доступу: https://www.arduino.cc/en/Guide

27. Raspberry Pi Foundation. *Raspberry Pi Documentation: IoT Development.* — 2024. — Режим доступу: https://www.raspberrypi.org/documentation/iot/