

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри

_____ В.О. РОМАНКЕВИЧ
(підпис)

“ ____ ” _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та компоненти»
спеціальності **123 «Комп'ютерна інженерія»**

на тему: Система генерації і оновлення документації опису тестів мовою
Common Lisp

Виконав:

студент IV курсу, групи КВ-61

Касянчук Дмитро Павлович

(підпис)

Керівник, доц.каф. СПСКС, к.т.н. Марченко О.І.

(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М.

(підпис)

Рецензент

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.
Студент _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та компоненти»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Віталій РОМАНКЕВИЧ

(підпис) (ініціали, прізвище)

«__» _____ 20__ р.

ЗАВДАННЯ

на дипломний проєкт студента

Касянчука Дмитра Павловича

1. Тема проєкту «Система генерації і оновлення документації опису тестів мовою Common Lisp», керівник проєкту Марченко О.І., кандидат технічних наук, доцент кафедри системного програмування і спеціалізованих комп'ютерних систем, затверджені наказом по університету від « » травня 2020 р. №
2. Термін подання студентом проєкту
3. Вихідні дані до проєкту: див. Технічне завдання
4. Зміст пояснювальної записки
 - Аналіз існуючих генераторів документації.
 - Аналіз засобів реалізації.
 - Розробка системи.
 - Тестування системи.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

- Система генерації і оновлення документації опису тестів мовою Common Lisp. Структурна схема.
- Зіставлення формальних параметрів з фактичними. Схема алгоритму.
- Генерація і оновлення документації опису тестів. Схема алгоритму.
- Перетворення оголошення тесту у внутрішнє подання. Схема алгоритму.
- Презентації за темою роботи.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доцент каф. СПіСКС		

7. Дата видачі завдання 30 жовтня 2019 року.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематикою проєкту	15.11.2019	
2.	Розроблення та узгодження технічного завдання	30.11.2019	
3.	Аналіз існуючих рішень	05.02.2020	
4.	Підготовка матеріалів першого розділу дипломного проєкту	22.04.2020	
5.	Підготовка матеріалів другого розділу дипломного проєкту	25.04.2020	
6.	Підготовка матеріалів третього розділу дипломного проєкту	29.04.2020	
7.	Підготовка матеріалів четвертого розділу дипломного проєкту	07.05.2020	
8.	Підготовка графічної частини дипломного проєкту	15.05.2020	
9.	Оформлення документації дипломного проєкту	19.05.2020	
10.	Попередній огляд матеріалів диплому на кафедрі	20.05.2020	

Студент

(підпис)

Дмитро КАСЯНЧУК

Керівник проєкту

(підпис)

Олександр МАРЧЕНКО

* Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (70 с., 94 рис., 1 табл., 4 додатки).

Об'єкт розробки – створення системи генерації і оновлення документації опису тестів мовою Common Lisp, яка є зручною у використанні і дозволяє прискорити процес розробки програмного забезпечення, економлячи час, який розробник би витратив для написання документації вручну.

Розроблена система надає:

- можливість запуску тестів через інтерпретатор;
- можливість дізнатися імена тестів, опис яких аналогічний або ні, тому, який записаний у документації;
- можливість побачити різницю в разі, якщо поведінка програм змінилася, а з нею змінився і опис;
- зручний інтерфейс для створення тестів.

В процесі розробки була використана мова програмування Common Lisp з використанням бібліотеки lparallel для паралельної роботи, інструмент meld для порівняння файлів, org-mode для зберігання документації та середовище розробки Emacs з додатком Slime.

В ході виконання дипломного проєкту:

- розроблено архітектуру системи;
- проведений аналіз існуючих рішень;
- розроблений зручний інструмент для тестування роботи програмного забезпечення, з інтеграцією результатів у файл документації.

Впровадження цієї системи дозволить пришвидшити процес розробки, а також мінімізувати роботу програміста над документацією, інтегрувати тести у файл документації.

Ключові слова: ГЕНЕРАТОР ДОКУМЕНТАЦІЇ, ORG-MODE, LPARALLEL, COMMON LISP, EMACS, AST , ПОРІВНЯННЯ ФАЙЛІВ.

ABSTRACT

Qualifying work includes an explanatory note (70 p., 94 fig., 1 table, 4 applications).

The object of development is the creation of a system of generation and updating documentation of tests' describing of Common Lisp which is very comfortable in usage, and saves time, which would be left by a handy writing of documentation.

System allows you to:

- the ability to run tests through the interpreter;
- the ability to find out the names of tests which describing is same or not, those recorded at the documentation;
- the ability to see the difference if the behavior of the programs has changed and with it the describing has changed;
- user-friendly interface for creation tests.

The development process used the Common Lisp programming language using the lparallel library for parallel computing, the meld tool for files comparison, org-mode for storing documentation, and the Emacs development environment with the Slime application.

During the implementation of the diploma project:

- system architecture is developed;
- analysis of existent solution;
- developed a convenient tool for testing the software, with the integration of results into the documentation.

The introduction of this system will speed up the development process, as well as minimize the work of the developer on the documentation, integrate the tests into documentation file.

Keywords: DOCUMENTATION GENERATOR, ORG-MODE, LPARALLEL, COMMON LISP, EMACS, AST, FILE COMPARISON.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ. 045490.002 ТЗ	Система генерації і оновлення документації опису тестів мовою Common Lisp. Технічне завдання	4		
	A4	ІАЛЦ. 045490.003 ТП	Система генерації і оновлення документації опису тестів мовою Common Lisp. Відомість технічного проекту	2		
	A4	ІАЛЦ. 045490.004 ПЗ	Система генерації і оновлення документації опису тестів мовою Common Lisp. Пояснювальна записка	70		
	A4	ІАЛЦ. 045490.005 Д1	Система генерації і оновлення документації опису тестів мовою Common Lisp. Структурна схема	1		

					ІАЛЦ. 045490.001 ОА			
Змін.	Арк.	№ докум.	Підпис	Дата				
Розробив		Касянчук Д.П.			Система генерації і оновлення документації опису тестів мовою Common Lisp. Опис альбому	Літ.	Аркуш	Аркушів
Перевірив		Марченко О. І.					1	2
Н. контроль		Клятченко Я.М.			КПІ ім. Ігоря Сікорського, ФПМ, КВ-61			
Затвердив		Романкевич В.О.						

ЗМІСТ

1.	НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ _____	2
2.	ПІДСТАВА ДЛЯ РОЗРОБКИ _____	2
3.	ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ _____	2
4.	ДЖЕРЕЛА РОБОТИ _____	2
5.	ТЕХНІЧНІ ВИМОГИ _____	3
5.1.	Вимоги до системи, що розробляється _____	3
5.2.	Вимоги до апаратного забезпечення _____	3
5.3.	Вимоги до мінімального програмного забезпечення _____	3
	ЕТАПИ РОЗРОБКИ _____	4

					ІАЛЦ.045490.002 ТЗ						
Зм.	Арк.	№ докум.	Підп.	Дата	Система генерації і оновлення документації опису тестів мовою Common Lisp. Технічне завдання						
Розроб.	Касянчук Д.П.								Літ.	Аркуш	Аркушів
Перевір.	Марченко О.І.								1	4	
Н. контр.	Клятченко Я.М.								НТУУ "КПІ ім.Ігоря Сікорського" ФПМ КВ-61		
Затв.	Романкевич В.О.										

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування роботи – «Система генерації і оновлення документації опису тестів мовою Common Lisp».

Область застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування та спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є розробка системи генерації і оновлення документації опису тестів мовою Common Lisp, який є дуже зручний у використанні, а також дозволить зекономити час, який розробник витратив би для написання документації вручну.

4. ДЖЕРЕЛА РОБОТИ

Джерелами інформації для розроблення є технічна література, публікації у періодичних виданнях та Інтернет ресурси з питань розробки.

					ІАЛЦ.045490.002 ТЗ	Арк.
						2
Зм.	Арк.	№ докум.	Підп.	Дата		

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до системи, що розробляється

Генератор документації повинен забезпечувати такі функції:

- запуск тестів через інтерпретатор;
- зручний інтерфейс оголошення тестів;
- коректність зміни файлу документації;
- можливість порівняння різних версій файлів документації;
- можливість розширення системи;

5.2. Вимоги до апаратного забезпечення

- Процесор: Intel Core i3.
- Оперативна пам'ять: 2 Гб.
- Простір на диску: 1 Гб.

5.3. Вимоги до мінімального програмного забезпечення

- Операційна система Windows, Linux чи Mac OS X.
- Компілятор SBCL для мови Common Lisp.

					ІАЛЦ.045490.002 ТЗ	Арк.
						3
Зм.	Арк.	№ докум.	Підп.	Дата		

ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту
1	Вивчення літератури за тематикою проєкту	15.11.2019
2.	Розроблення та узгодження технічного завдання	30.11.2019
3.	Аналіз існуючих рішень	05.02.2020
4.	Підготовка матеріалів першого розділу дипломного проєкту	22.04.2020
5.	Підготовка матеріалів другого розділу дипломного проєкту	25.04.2020
6.	Підготовка матеріалів третього розділу дипломного проєкту	29.04.2020
7.	Підготовка матеріалів четвертого розділу дипломного проєкту	07.05.2020
8.	Підготовка графічної частини дипломного проєкту	15.05.2020
9.	Оформлення документації дипломного проєкту	19.05.2020
10.	Попередній огляд матеріалів диплому на кафедрі	20.05.2020

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4	ІАЛЦ.045490.000	Завдання на дипломний проєкт	2	
2	A4	ІАЛЦ.045490.001 ОА	Опис альбому	2	
3	A4	ІАЛЦ.045490.002 ТЗ	Технічне завдання	4	
4	A4	ІАЛЦ.045490.003 ТП	Відомість технічного проєкту	1	
5	A4	ІАЛЦ.045490.004 ПЗ	Пояснювальна записка	70	
6	A4	ІАЛЦ.045490.005 Д1	Система генерації і оновлення документації опису тестів мовою Common Lisp. Структурна схема.	1	
7	A4	ІАЛЦ.045490.006 Д2	Зіставлення формальних параметрів з фактичними. Схема алгоритму.	1	
8	A4	ІАЛЦ.045490.007 Д3	Генерація і оновлення документації опису тестів. Схема алгоритму.	1	
9	A4	ІАЛЦ.045490.008 Д4	Перетворення оголошення тесту у внутрішнє подання. Схема алгоритму.	1	

				ІАЛЦ.045490.003 ТП		
	ПІБ	Підп.	Дата			
Розробн.	Касянчук Д.П.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Марченко О.І.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-61	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Романкевич В.О.					

Пояснювальна записка до дипломного проєкту

на тему: Система генерації і оновлення документації опису тестів мовою
Common Lisp

Київ – 2020 року

ЗМІСТ

ВСТУП.....	3
1.АНАЛІЗ ІСНУЮЧИХ ГЕНЕРАТОРІВ ДОКУМЕНТАЦІЇ	5
1.1 Основні визначення.....	5
1.2 Аналіз існуючих генераторів документації.....	5
1.3 Висновки	11
2.АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ	12
2.1 Полегшені мови розмітки	12
2.2 Інструменти порівняння файлів.....	14
2.3 Мова реалізації	17
2.4 Засоби розбору файлів	18
2.5 Засоби паралелізації	20
2.6 Висновки	22
3.РОЗРОБКА СИСТЕМИ.....	23
3.1 Основні визначення.....	23
3.2 Модуль common.....	24
3.3 Модуль org.....	36
3.4 Модуль utils.....	44
3.5 Висновки	51

					ІАЛЦ.045490.004 ПЗ			
Зм.	Арк.	№ докум.	Підп.	Дата	Система генерації і оновлення документації опису тестів мовою Common Lisp. Пояснювальна записка.	Літ.	Аркуш	Аркушів
Розроб.		Касянчук Д.П.				1		
Перевір.		Марченко О.І.						
Н. контр.		Клятенко Я.М.						
Затв.		Романкевич В.О.						
						НТУУ "КПІ ім.Ігоря Сікорського" ФПМ КВ-61		

4.ТЕСТУВАННЯ СИСТЕМИ.....	52
4.1 Тестування	52
4.2 Висновки	68
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	70

ДОДАТКИ

Додаток 1. Копії графічного матеріалу.

ІАЛЦ.045490.005 Д1. Система генерації і оновлення документації опису тестів мовою Common Lisp. Структурна схема.

ІАЛЦ.045490.006 Д2. Зіставлення формальних параметрів з фактичними. Схема алгоритму.

ІАЛЦ.045490.007 Д3. Генерація і оновлення документації опису тестів. Схема алгоритму.

ІАЛЦ.045490.008 Д4. Перетворення оголошення тесту у внутрішнє подання. Схема алгоритму.

Додаток 2. Фрагменти програмного коду.

Додаток 3. Публікація за темою роботи.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		2

ВСТУП

Технічна документація в процесі розробки програмного забезпечення – це узагальнене поняття до якого входять усі матеріали, що відносяться до розробки програмного продукту. Кожен продукт, незалежно від того, був створений він невеликою групою програмістів чи крупною корпорацією, потребує документації. Різні типи документації створюються на кожному етапі розробки[1].

Причини створення документації:

- опис роботи проекту;
- уніфікація інформації, яка пов'язана с програмним продуктом;
- обговорення істотних питань, які виникають під час процес розробки, між зацікавленими сторонами і розробниками.

Документацію можна розділити на 2 категорії:

- документація продукту;
- документація процесів.

Документація продукту описує програмне забезпечення, яке розробляється. Як правило, така документація містить в собі вимоги, технічні характеристики, бізнес логіку і інструкції для роботи з цим продуктом[1].

Існує 2 типи такої документації:

- системна документація, яка включає в себе документи, що описують саму систему та її частини. Вона включає в себе документи з вимогами, проєктні рішення, опис архітектури,

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		3

вихідний код продукту і поширені запитання;

- документація користувача, яка охоплює інструкції, що призначені для кінцевих користувачів продукту, а також системних адміністраторів.

Одним із найскладніших етапів розробки програмного забезпечення є його документування. Для того, щоб написати хорошу документацію для програмного продукту, необхідно використовувати правильні інструменти програмної документації. Без неї програмний продукт нагадує чорний ящик. Документація – це та річ, яка дозволяє перетворити чорний ящик у прозорий. Існують генератори, які дозволяють створити документацію в певному форматі, з коду, який задокументований спеціальними синтаксичними конструкціями. Такий підхід дозволяє зекономити час, який витрачається на її ручне написання і оформлення. Але крім того, також існує потреба у генеруванні документації, яка детально описує роботу програми при заданих значеннях вхідних параметрів і можливість швидкого її оновлення, для більш швидкої розробки.

Завданням дипломного проєкту є розробка програмного забезпечення, яке дозволяє згенерувати документацію опису поведінки програм при різних вхідних параметрах, які написанні мовою Common Lisp, а також можливість її швидкого оновлення.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		4

1.АНАЛІЗ ІСНУЮЧИХ ГЕНЕРАТОРІВ ДОКУМЕНТАЦІЇ

1.1 Основні визначення

Генератор документації — це програма, результатом виконання якої є отримання документації призначеної для програмістів або користувачів системи. Принцип роботи таких програм полягає в аналізі вихідного коду для знаходження синтаксичних конструкцій, які відповідають головним об'єктам програми (тип, метод, функція, процедура і т.д.). Спираючись на всю зібрану інформацію формується документація в одному з форматів — HTML, HTMLHelp, PDF, RTF та інших. Залежно від того, який генератор використовується, є відмінності і в синтаксисі конструкцій, які наявні в документованих коментарях[2].

Документований коментар — спеціальним чином оформлений коментар, що відповідає певному об'єкту програми. Як правило, вони можуть містити інформацію про автора коду, зміст вхідних і вихідних параметрів для функції і тому подібне. Загалом документація необхідна і дуже важлива для комерційної розробки, адже вона допомагає розібратися з тим, як працює система за менший період часу, а генератори, в свою чергу, значно зменшують кількість часу для створення документації[2].

1.2 Аналіз існуючих генераторів документації

Doxygen (рис. 1) — використовується з такими мовами як C++, C, C#, Fortran, PHP, IDL, Java, VHDL, Objective-C, Python. Даний генератор може сформувати документацію, яка буде містити в собі посилання, діаграми класів, викликів і т.п. в різних форматах: HTML, LaTeX, CHM, RTF, PDF, map-сторінки. Doxygen дуже простий і зрозумілий в налаштуванні та установці[2].

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		5

Його особливістю є можливість генерації документації не тільки на основі вихідного коду, а й на недокументованому вихідному коді. Також слід зазначити, що в документації, представлений в HTML форматі, реалізована можливість зручного пошуку, а також є посилання на зовнішню документацію. Якщо при роботі з Doxygen користувачу знадобилося задати додаткові параметри для майбутньої документації, то це можна зробити використовуючи конфігураційний файл, який являє собою звичайний текстовий формат. Проте слід зазначити і те, що для роботи з цим файлом існує декілька програм з графічним інтерфейсом, і це суттєво полегшує налаштування конфігураційного файлу. До мінусів можна віднести те, що цей генератор не має справжнього модульного налаштування[2].

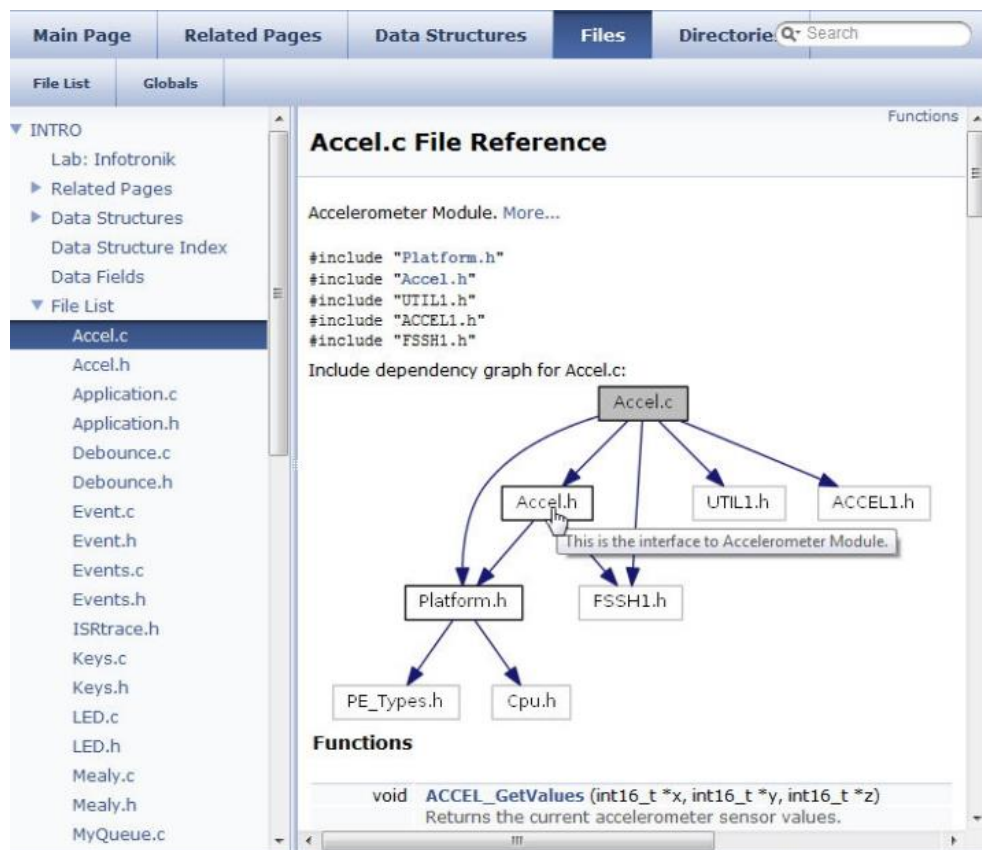


Рис. 1 - Результат роботи генератора Doxygen

JSDoc (рис.2) — це генератор документації в HTML форматі з вихідного коду на мові JavaScript. До переваг даного компілятора слід віднести те, що ядро генератора написано на мові JavaScript, а тому для генерації документації треба лише встановлена java-машина і більше ніяких додаткових файлів. Також плюсом є наявність великої кількості прикладів використання JSDoc, що значно полегшує його освоєння. Також реалізована можливість роботи з коментарями російською мовою. Але існує і певний недолік, а саме, неможливість генерації документації для класів, які знаходяться в анонімній функції[2].

```
/**
 * Creates a new Circle from a diameter.
 *
 * @param {number} d The desired diameter of the circle.
 * @return {Circle} The new Circle object.
 */
static fromDiameter(d) {
    return new Circle(d / 2)
}

/**
 * Calculates the circumference of the Circle.
 *
 * @deprecated since 1.1.0; use getCircumference instead
 * @return {number} The circumference of the circle.
 */
calculateCircumference() {
    return 2 * Math.PI * this.radius
}
```

Рис. 2 - Приклад синтаксису для JSDoc

Sphinx (рис.3) — генератор, який спочатку створювався лише для мови Python, але потім розробники додали підтримку C та C++ і планується додати підтримку ще більшій кількості популярних мов програмування. Принцип роботи генератора Sphinx в тому, що він перетворює файл в форматі reStructuredText в HTML, PDF, man та інші. reStructuredText — це полегшена

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		7

мова розмітки. Даний генератор легко встановити і він доступний для всіх операційних систем, які підтримують мову Python. До недоліків можна віднести те, що для використання Sphinx потрібно додатково розібратися з reStructuredText[2].

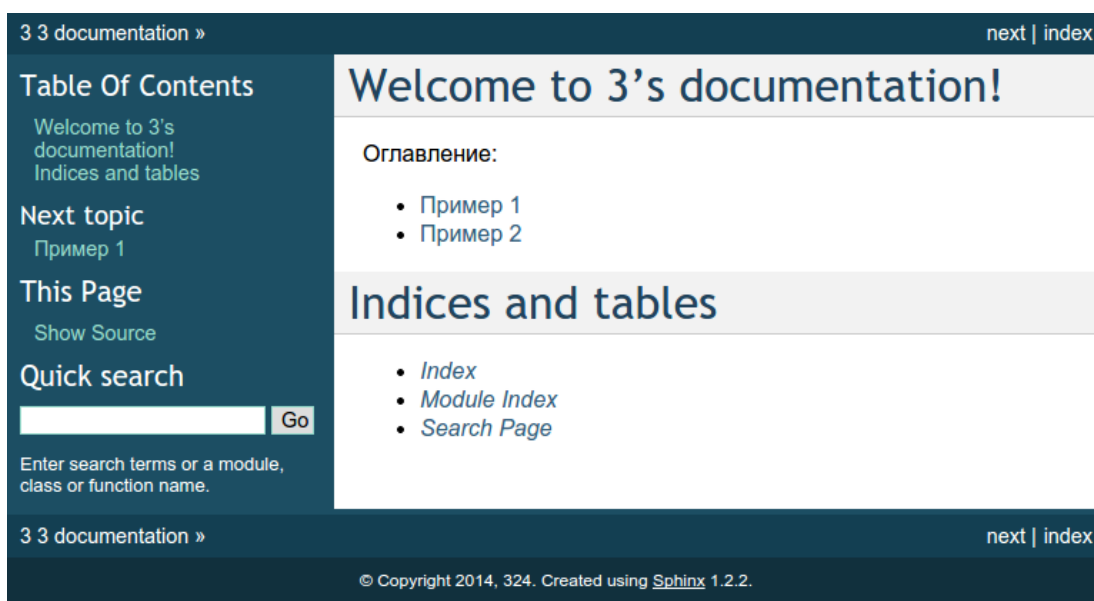


Рис. 3 - Результат роботи генератора Sphinx

JSDuck (рис.4) - генератор документації, цікавою особливістю якого є можливість використовувати його для будь-якого коду за умови, що цей код задокументовано відповідно до стандартів. Також JSDuck відзначається легкістю в установці та використанні. Ще однією особливістю можна назвати те, що цей генератор може аналізувати код і генерувати документацію навіть без стандартних блоків. Реалізована можливість додавання спеціальних сторінок, а саме: сторінка привітання, сторінка з категоріями, сторінка з інструкціями, сторінка з прикладами. По підсумкам опитування, яке проводилося серед користувачів JSDuck, було зроблено висновок, що це дуже простий в розумінні і використанні генератор документації, недоліків немає[2].

```

/** @example
 * Ext.create('Ext.form.Panel', {
 *   title: 'Contact Info',
 *   width: 300,
 *   bodyPadding: 10,
 *   renderTo: Ext.getBody(),
 *   items: [{
 *     xtype: 'textfield',
 *     name: 'name',
 *     fieldLabel: 'Name',
 *     allowBlank: false
 *   }, {
 *     xtype: 'textfield',
 *     name: 'email',
 *     fieldLabel: 'Email Address',
 *     vtype: 'email'
 *   }]
 * });
 */

```

Рис. 4 - Приклад синтаксису JSDuck

Slate (рис.5) — це генератор, який має інтуїтивно зрозумілий дизайн, він адаптивний і тому з ним можна працювати як на комп'ютерах так і на планшетах, ноутбуках, телефонах. Slate має посилання на різні пункти документації, що допомагає користувачу швидко знайти потрібний розділ. Також цей генератор використовує Markdown і це спрощує розуміння та редагування. Ще одним плюсом для даного генератора є реалізована можливість підсвічування для 60-ти різних мов програмування і це додає комфорту до роботи з ним[2].

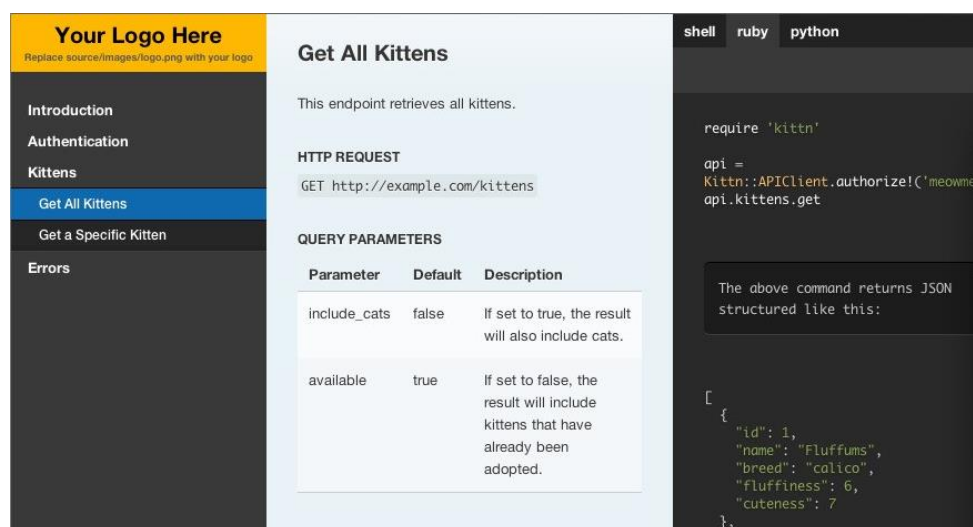


Рис. 5 - Інтерфейс генератора документації Slate

В таблиці 1 зображена оцінка за 5-ти бальною шкалою всіх засобів за трьома критеріями, які є важливими для генератора документації, а саме: кількість підтримуваних мов, кількість вихідних форматів та інтерфейс користувача. Інтерфейс користувача оцінювався за такими параметрами, як: зручність, відповідність сучасним тенденціям, а також нативність. Варто зазначити, що оцінка інтерфейсу – це суб’єктивна оцінка, а кількість підтримуваних мов і вихідних форматів – це значення, яке пропорційне статистичним даним взятим з різних джерел інформації. Сумарна оцінка вираховувалася, як середнє арифметичне цих оцінок[2].

Таблиця 1. Порівняння генераторів документації за різними критеріями

Інструмент \ Критерій	Кількість підтримуваних мов	Кількість вихідних форматів	Інтерфейс користувача	Сумарна оцінка
Doxygen	4	5	4	4.3
JSDoc	2	4	4	3.3
Sphinx	3	3	4	3.3
Slate	4	4	5	4.3
JSDuck	5	4	5	4.6

1.3 Висновки

З огляду на все вищесказане, можна назвати JSDuck беззаперечним лідером серед представлених генераторів документації. Цей генератор має всі якості, якими повинен бути наділений потужний інструмент для генерації документації. Реалізована підтримка для всіх мов програмування, що робить його універсальним генератором, який підійде для розробника на будь-якій мові програмування. Не поступається JSDuck і генератор Slate, він підтримує 60 мов програмування, а також з ним можна працювати знаходячись будь-де, адже він має адаптивний дизайн і чудово виглядає як на екрані комп'ютера, так і на телефоні. Проте, якщо Ви розробник на мові C, C++, Python, то для Вас будуть цікаві і такі генератори, як Sphinx та Doxygen. Вони зрозумілі, зручні та детальні. Хоча слід пам'ятати і те, що для використання Sphinx буде потрібно розібратися з reStructuredText. Якщо Вам цікаві генератори документації для мови JavaScript, то слід звернути увагу на JSDoc. З використанням цього генератору легко розібратися, адже існує велика кількість прикладів, а також він не потребує додаткових файлів для установки. Проте проблема генератору в тому, що він не генерує документацію для класів, які знаходяться в анонімній функції[2].

Після аналізу існуючих генераторів можна зробити висновок, що жоден із описаних інструментів не має можливості інтегрувати та/або оновлювати документацію опису тестів, які написані мовою Common Lisp, і саме тому, метою створення даного дипломного проєкту – є розробка системи, яка мала б ширші функціональні можливості, які включають в себе засоби для швидкого створення чи оновлення тестів.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		11

2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Полегшені мови розмітки

Полегшена мова розмітки - це мова розмітки документів з простим і ненав'язливим синтаксисом. Вона створена таким чином, щоб її можна було легко описати за допомогою будь-якого текстового редактора, а також прочитати в неопрацьованій формі[3]. Такі мови розмітки використовуються в програмному забезпеченні, в якому потрібно як неопрацьований документ, так і кінцевий результат. Розглянемо та проаналізуємо популярні полегшені мови розмітки.

Markdown (рис.6) – це полегшена мова розмітки, конструкція якої дозволяє перетворити її у багато популярних вихідних форматів. Варто зазначити, що оригінальний інструмент обробки такого формату, який має таку ж назву, дозволяє отримати лише HTML документ[4].

Markdown часто використовується для:

- написання readme файлів;
- обміну повідомлень на онлайн-форумах;
- створення стилізованого тексту у звичайному текстовому редакторі.

Оскільки початковий опис Markdown мав неточності і невирішені запитання, то реалізації, що з'являлися протягом багатьох років, мають невеликі відмінності, які проявляються в синтаксичних розширеннях. Створення багатьох реалізацій обумовлено потребою в додаткових функціях поверх базового синтаксису - такі як: таблиці, зноски, списки визначень (списки HTML) та Markdown всередині HTML блоків. Поведінка деяких з них

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		12

відрізняється від еталонної реалізації. У той же час увагу привернули ряд неоднозначностей у неофіційних специфікаціях. Це спонукало до створення інструменту для порівняння результатів Vabelmark, а також розробки Markdown парсера для стандартизації.

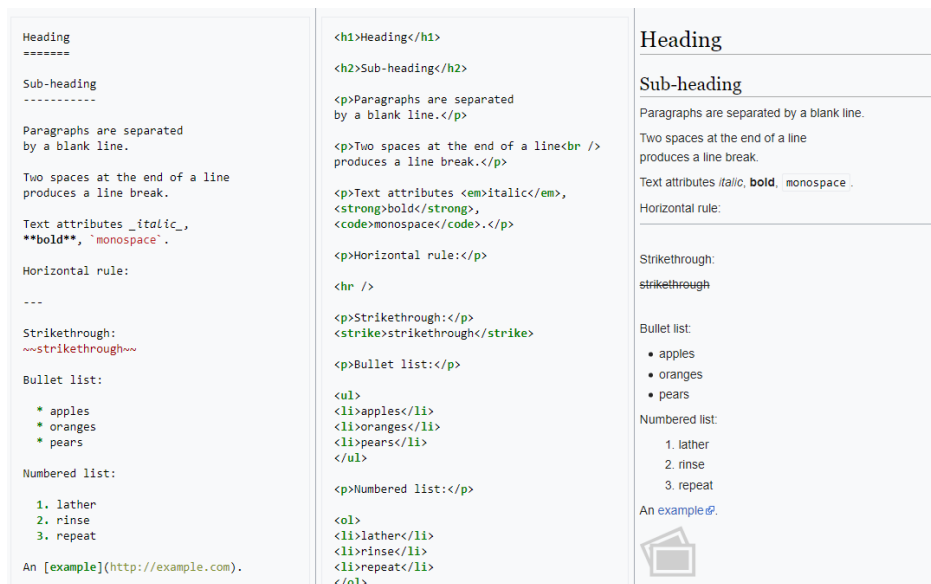
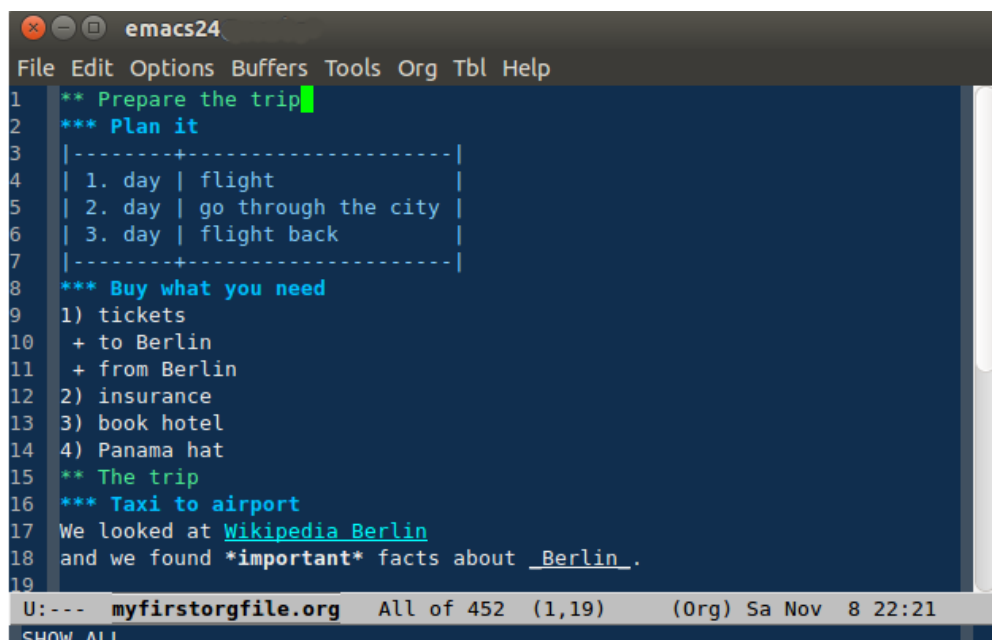


Рис.6 – Приклад перетворення Markdown розмітки

Org-mode (рис.7) – програмний засіб, який створений для введення нотатків, планування проєктів, створення списків завдання і працює всередині середовища розробки Gnu Emacs. В основі цього інструменту лежать звичайні текстові файли[5]. Варто зазначити, що org-mode містить багато можливостей, які значно розширюють його сферу застосування. Також org-mode, як і середовище проєктування Emacs, має багато комбінацій клавіш, що суттєво підвищує ефективність роботи. В основі org-mode лежить ієрархія заголовків, які в свою чергу можуть містити в собі вкладені заголовки. Текст, який міститься в заголовках можна приховати, показати частково (не розкривати вкладені розділи). Існують зручні комбінації клавіш, функціями яких є: створення, видалення і переміщення між заголовків. Також він має широкі

можливості для зв'язку з поточним або іншим файлом, веб-сторінками, електронними листами і дозволяє визначити власні посилання. Org-mod документ може з легкістю бути конвертований в такі популярні формати, як: HTML, LaTeX, OpenDocument або текстові данні.



```
emacs24
File Edit Options Buffers Tools Org Tbl Help
1  ** Prepare the trip
2  *** Plan it
3  |-----+-----|
4  | 1. day | flight |
5  | 2. day | go through the city |
6  | 3. day | flight back |
7  |-----+-----|
8  *** Buy what you need
9  1) tickets
10 + to Berlin
11 + from Berlin
12 2) insurance
13 3) book hotel
14 4) Panama hat
15 ** The trip
16 *** Taxi to airport
17 We looked at Wikipedia Berlin
18 and we found important facts about Berlin.
19
U:--- myfirstorgfile.org All of 452 (1,19) (Org) Sa Nov 8 22:21
SHOW ALL
```

Рис.7 – Приклад .org документу

2.2 Інструменти порівняння файлів

Під час написання програмних або звичайних файлів, у програмістів або звичайних користувачів з'явилась потреба у пошуку різниці між декількома файлами або різними версіями одного і того ж файлу. Коли порівнюються 2 файли в операційній системі Linux, різницю між ними називають diff. Існує багато інструментів для порівняння файлів і зараз ми розглянемо та проаналізуємо найпопулярніші з них.

diff – вбудована в операційну систему Linux команда, яка показує різницю між двома файлами. Вона порівнює файл рядок в одному файлі у відповідний йому рядок в іншому і повертає різницю між ними. В основі багатьох інших інструментів для пошуку різниці між файлами лежить diff.

На заміну інструментам, які запускаються в терміналі, прийшли нові з графічним інтерфейсом.

Meld (рис.8) – це легкий графічний інструмент, для порівняння і поєднання файлів. Він дозволяє користувачам порівнювати файли, директорії, а також програми з контролем версій[6]. Створений спеціально для розробників програмного забезпечення і має такі функції:

- двостороннє і тристороннє порівняння файлів, директорій;
- оновлення порівняння файлів, після того як користувач додав інформації;
- полегшує поєднання двох файлів, роблячи його автоматичним та можливість виконувати дії на змінених блоках;
- наочне порівняння за допомогою візуалізації;
- підтримує Git, Mercurial, Subversion, Bazaar і багато інших інструментів контролю версій.

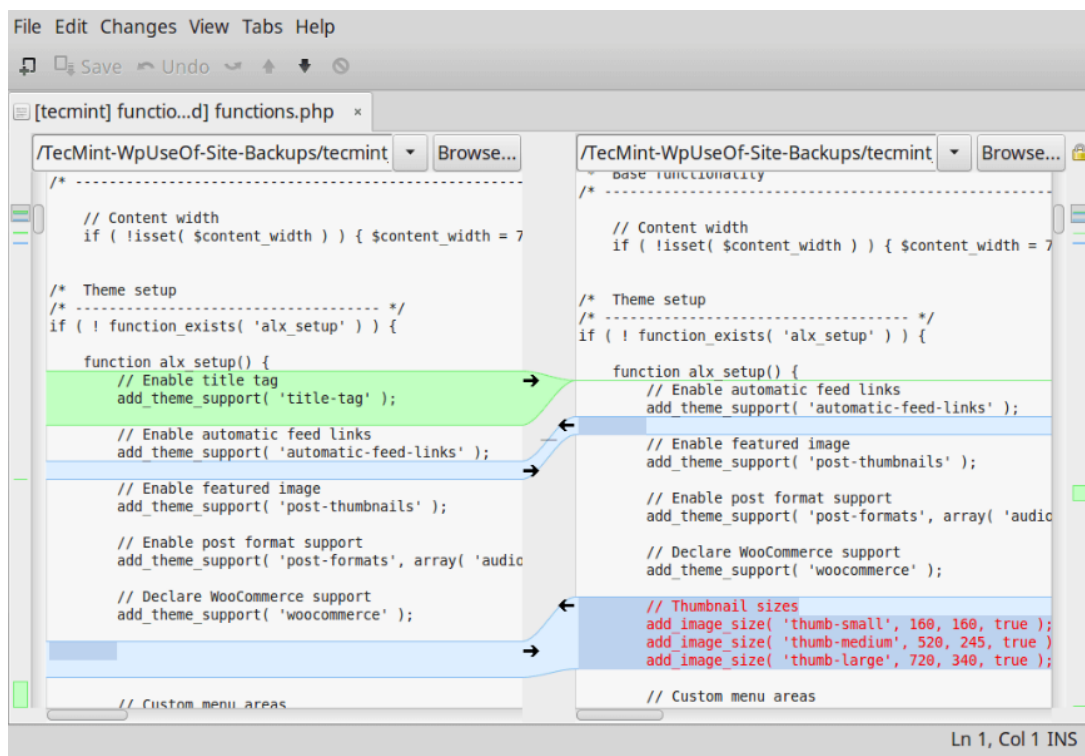


Рис.8 – Порівняння файлів за допомогою Meld

DiffUse (рис.9) – ще один популярний, безкоштовний і простий графічний інструмент для порівняння і поєднання файлів на Linux. Він дозволяє порівнювати файли, контроль версій, редагувати файли, об'єднувати файли, а також виводити різницю між ними[7]. Можна переглянути загальну різницю, вибирати рядки в тексті, використовуючи вказівник миші, поєднувати рядки в сусідніх файлах і змінювати ці файли.

Також він має такі переваги:

- підсвічування синтаксису;
- комбінації клавіш для швидкого переміщення по файлу;
- можливість відмінити свої зміни необмежену кількість разів;
- підтримка Юнікоду;
- підтримка Git, CVS, Darcs, Mercurial, RCS, Subversion, SVK and Monotone.

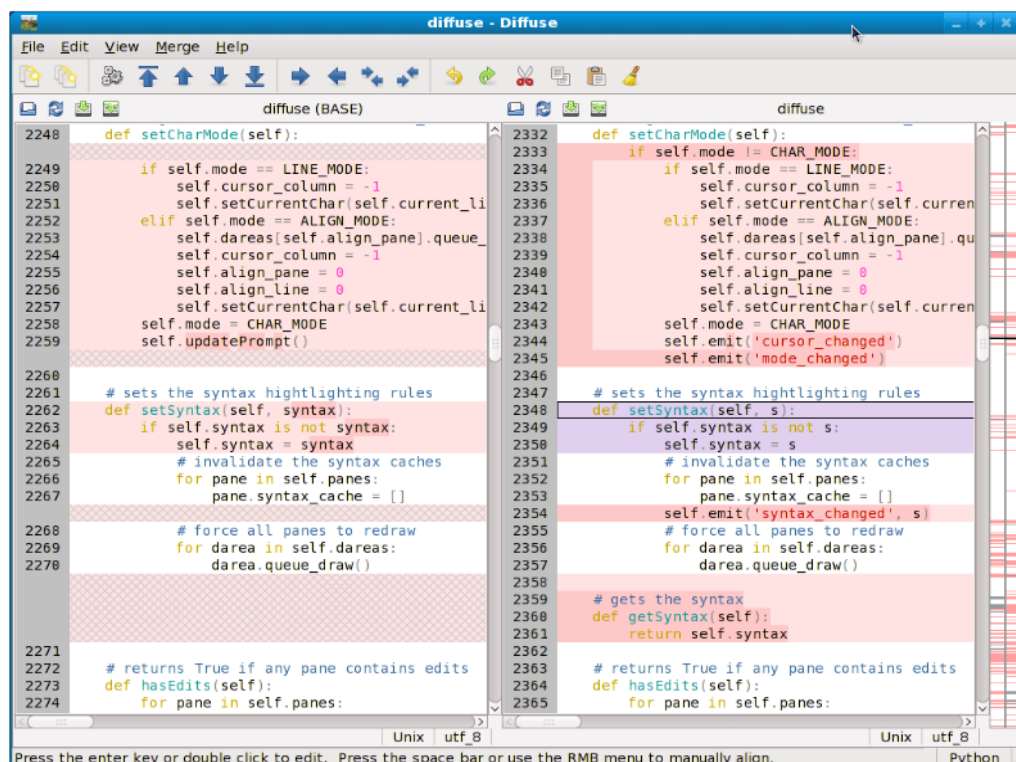


Рис.9 – Приклад порівняння файлів за допомогою DiffUse

Зм	Лист	№ докум.	Підп.	Дата

ІАЛЦ.045490.004 ПЗ

Арк.
16

2.3 Мова реалізації

Для реалізації генератора документації опису поведінки програм, які написані мовою Common Lisp при різних вхідних параметрах була використана мова – Common Lisp, а також бібліотеки та інструменти, які були створені для цієї мови.

Common Lisp – це мова програмування, яка підтримує декілька парадигм програмування, такі як:

- процедурна;
- функціональна;
- об'єктно-орієнтована.

Об'єктно-орієнтована парадигма реалізована, системою CLOS, яка включена в стандарт. Великою перевагою є так звана система лісп-макросів, які дозволяють додавати нові синтаксичні конструкції[8].

Common Lisp використовується в дипломному проєкті так, як має ряд переваг. Розглянемо деякі з них.

Символи - це унікальні об'єкти, які повністю визначаються своїми іменами. Порівняння символів відбувається за фіксований час. Символи, як і функції являються сутностями першого класу.

Множинні значення – можливість повертати декілька значення з функції, без поєднання їх в одну структуру даних.

Функція `format` – дозволяє виразити різні правила генерації тексту у зручному форматі.

Функції вищого порядку – це функції аргументами і значеннями, які повертаються, можуть бути функціями.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		17

Анонімні функції – можуть використовуватися замість імені функції, яка передається в якості аргументу. Це дуже зручний механізм, коли ми не хочемо створювати зайві функції, які засмічують програму.

Лямбда-списки – визначають обов’язкові, необов’язкові, іменовані та так званні rest параметри для функцій, а також значення за замовчуванням.

Узагальнені функції (рис.10.1 – 10.2) – CLOS не прив’язує методи до конкретного класу і дозволяє використовувати узагальнені функції. Вони задають сигнатуру виклику, якій може підходити декілька методів. При виклику викликається метод, який краще відповідає аргументам.

```
(defgeneric key-input (key-name))
```

Рис.10.1 – Оголошення узагальненою функції

```
(defmethod key-input (key-name)
  ;; Default case
  (format nil "No keybinding for ~a" key-name))

(defmethod key-input ((key-name (eql :escape)))
  (format nil "Escape key pressed"))

(defmethod key-input ((key-name (eql :space)))
  (format nil "Space key pressed"))
```

Рис.10.2 – Оголошення методів

2.4 Засоби розбору файлів

Для оновлення існуючих тестів в документації необхідно представити файл у зручному форматі у вигляді певних структур даних. Для цієї задачі в

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		18

дипломному проєкті структура файлу представляється у вигляді абстрактного синтаксичного дерева (AST), яке можна отримати після розбору файлу

Для розбору файлів і перетворення його у абстрактне синтаксичне дерево була розглянуто декілька популярних бібліотек. Розглянемо їх більш детально.

CL-Yacc (рис.11) – це генератор синтаксичних аналізаторів для мови Common Lisp, який схожий на інші популярні генератори для інших мов програмування, такі як:

- AT & T Yacc;
- Berkeley Yacc;
- GNU Bison;
- Zebu;
- lalr.cl або lalr.scm.

З самого початку він був створений для розбору граматик для розширеного набору мов програмування C (більш ніж 400 екземплярів)[9].

Переваги:

- безкоштовне програмне забезпечення;
- єдиний файл, який ви можете розповсюджувати разом зі своїм продуктом;
- підтримка неоднозначних граматик (пріоритет операторів і асоціативність);
- хороша(не відмінна) продуктивність.

CL-Yacc має багато переваг і дуже потужний інструмент для генерації парсерів, але через те що, як було сказано в попередньому розділі, в дипломному проєкті для генерації документації використовується org-mode, для задачі розбору файлу і перетворення його в AST, було обрано бібліотеку CL-ORG-MODE.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		19

CL-ORG-MODE – це інструмент для розбору файлів org-mode, який використовує синтаксичний аналізатор, методом рекурсивного спуску, на основі CLOS для створення дерева вузлів режиму org[10].

2.5 Засоби паралелізації

Паралельні обчислення – це форма обчислень, при яких кілька дій виконуються одночасно. Вони базуються на тому, що великі задачі можна розділити на кілька менших, кожен з яких можна розв’язати незалежно від інших.

В дипломному проєкті робота генератора була розпаралелена для підвищення продуктивності. Мова Common Lisp має декілька інструментів для створення і синхронізації паралельних потоків. Проаналізуємо дві найпопулярніших бібліотеки.

Bordeaux-threads – забезпечує, незалежний від платформи, спосіб обробляти основні поняття для потоків в декількох реалізаціях Common Lisp. Цікавий факт полягає в тому, що вона не створює власну реалізацію для потоків, а цілком використовує базову[11]. З іншого боку в ній додано декілька корисних можливостей для абстракцій над низькорівневими потоками. Варто зазначити, що багато функцій з цієї бібліотеки є досить схожими на ті, що використовуються в sb-thread.

lparallel – забезпечує широку підтримку асинхронного програмування, і не є суто бібліотекою для паралельних потоків. Бібліотека побудована поверх Bordeaux-threads бібліотеки[12].

Вона має такі переваги:

- проста модель створення задачі з отриманням черги;
- обробка асинхронних умов через потоки;
- паралельні версії таких функцій як: map, reduce, sort, remove і

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		20

багато інших стандартних функцій;

- підтримка `promise`, `future` і `delay`;
- дерева обчислень для розпаралелювання взаємопов'язаних задач;
- обмежені та необмежені FIFO черги;
- канали;
- задачі з високим і низьким пріоритетом;
- завершення потоків з певною категорією;
- інтегровані тайм-аути.

Алгоритм використання `lparallel` (для базового випадку) такий:

- створити екземпляр, так званого ядра (компонент, який планує та виконує задачі) за допомогою `make-kernel`;
- розробити код з точки зору `futures`, `promises` та інших високорівневих концепцій;
- виконувати операції, використовуючи так звані `cognates` (див. нижче);
- закрити створене ядро за допомогою функції `end-kernel`.

`Cognates` (рис.11) – головна причина існування бібліотеки `lparallel` і вони справді створюють паралелізм у ній. Однак, варто зауважити, що майже всі з цих конструкцій реалізовані на основі `futures` і `promises`. Якщо говорити коротко, то `cognates` – це просто функції, які є паралельними еквівалентами стандартних функцій `Common Lisp`. Однак, існують кілька функцій, які не мають аналогів в стандарті[12].

`Cognates` реалізовано для 2-ох задач:

- конструкції для низькорівневої паралелізації: `defpun`, `plet`, `plet-if`, тощо.
- явні функції та макроси для виконання паралельних операцій – `ptar`, `preduce`, `psort`, тощо.

```

(defun test-por ()
  (let ((a 100)
        (b 200)
        (c nil)
        (d 300))
    (por a b c d)))

LPARALLEL-USER> (dotimes (i 10)
                  (print (test-por)))

300
300
100
100
100
300
100
100
100
100

```

Рис.11 – Приклад cognates

2.6 Висновки

В даному розділі дипломного проєкту було розглянуто засоби для реалізації задач, які повинна виконувати система. Серед полегшених мов розмітки був вибраний org-mode через популярність в колах розробників Common Lisp, а також чудову сумісність з середовищем розробки Emacs. Також були обґрунтовані переваги мови Common Lisp, яка була обрана для реалізації дипломного проєкту, а також бібліотек для паралелізації роботи програми – lparallel і Bordeaux-threads, і засобів для розбору org-mode файлів – cl-org-mode. Важливою задачею в проєкті є порівняння різних версій файлів документації, для цієї задачі було порівняно 2 майже рівних інструменти DiffUse і Meld, але обраний Meld через популярність, а також широкі функціональні можливості, який він має.

3. РОЗРОБКА СИСТЕМИ

3.1 Основні визначення

Для реалізації дипломного проєкту була використана `package-inferred-system`, де кожен модуль представляється так званим «пакетом». Всі пакети компонуються в `.asd` файл, який потім компілюється і збирається, коли з'являється необхідність у використанні системи. Також в `.asd` файлі можуть вказуватися і інші системи, які програміст використовує в своїй системі. Спочатку вказується ім'я (як правило, ім'я пакету — це символ, який починається з двокрапки і складається з шляху відносно `.asd` файлу системи). Кожна наступна форма на початку містить певне ключове слово і параметри.

В дипломному проєкті використовувались такі ключові слова

- `use` — для позначення залежностей від інших пакетів(в цьому випадку пакет, який використовує інший пакет, отримує доступ до всіх символів, які експортуються з нього);
- `import-from` — для позначення символів, які пакет хоче використовувати. Варто зазначити, що в разі використання цього ключового слова, також можна отримати доступ до тих символів, які не експортуються з пакету;
- `export` — для позначення символів, що пакет екпортує, і які можуть використовувати інші пакети.

Для того, щоб символи створювалися у даному пакеті, на початку варто вказати конструкцію (`in-package <package-name>`). Всі форми, які будуть створені після цієї конструкції, відносяться до контексту `<package-name>`.

Як правило, система ділиться на модулі, кожен з яких містить пакети,

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		23

які реалізують функції даного модуля. Кожен модуль має містити, так званий, all файл, який компонує всі пакети з нього, і експортує тільки ті елементи, що користувачі можуть використовувати.

Розроблена система містить 3 модулі, кожен з яких призначений для реалізації певною задачі. Розглянемо кожен модуль розробленої системи. docgen.asd (рис.12) – файл, мета якого:

- скомпонувати всі пакети в розробленій системі;
- встановити залежності від зовнішніх систем, які використовуються.

```
(defsystem :docgen
  :class :package-inferred-system
  :defsystem-depends-on (:asdf-package-system)
  :pathname #p"./"
  :depends-on (:alexandria
              :cl-fad
              :lparallel
              :cl-org-mode
              :docgen/utils/all
              :docgen/common/all
              :docgen/org/all))
```

Рис.12 – Інтерфейс основного файлу системи

3.2 Модуль common

common – містить головні інтерфейсні функції, які призначені для зовнішнього користування. Складається з 3 пакетів. Розглянемо кожен пакет, а також функціональні можливості, який він реалізує.

docgen/common/core (рис.13) – містить реалізацію для паралельного оновлення файлу документації, а також деякі інші корисні функції.

```

(uiop:define-package :docgen/common/core
  (:use :cl :docgen/utils/all :docgen/org/all)
  (:import-from :cl-fad
    #:with-open-temporary-file)
  (:export #:make-test-info
    #:update-docfile
    #:skip
    #:run-meld
    #:fail
    #:pass
    #:disable
    #:to-doc
    #:collect-test-infos
    #:report
    ;; builder
    #:builder
    #:passed-tests
    #:failed-tests
    #:disabled-tests
    #:dots
    #:mutex))

```

Рис.13 – Інтерфейс пакету docgen/common/core

В цьому пакеті оголошений спеціальний клас builder (рис.14) для обміну інформації між паралельними процесами, а також функції роботи з цим класом.

```

(defclass builder ()
  ((stream-output
    :initarg :stream
    :initform nil
    :accessor stream-output)
   (failed-tests
    :initform nil
    :accessor failed-tests)
   (passed-tests
    :initform nil
    :accessor passed-tests)
   (disabled-tests
    :initform nil
    :accessor disabled-tests)
   (dots
    :initform 0
    :accessor dots)
   (mutex
    :initform (make-lock)
    :reader mutex)))

```

Рис.14 – Клас builder

Розглянемо кожне поле окремо:

- `stream-output` – вказівник потоку, в який буде здійснено вивід про тести які пройшли, не пройшли або були проігноровані. Значення за замовчуванням цього поля `nil`, що означає, що інформація не буде нікуди виводитися;
- `failed-tests` – список, що складається з тестів, результати яких відрізняються від тих, що в файлі документації. Також в цьому списку будуть тести, які в ньому не існують;
- `passed-tests` – список, що складається з тестів, результати яких повністю ідентичні тим, що лежать у файлі;
- `disabled-tests` - список тестів, які не аналізувались і були проігноровані;
- `dots` – поле, яке містить числове значення, необхідне для форматowanego виводу інформації про тести;
- `mutex` – містить об'єкт м'ютексу для запобігання одночасного доступу паралельними процесами полів з цього класу.

Для зміни таких полів, як: `failed-tests`, `passed-tests`, `disabled-tests` - були створені методи (рис.15), які приймають об'єкт класу `builder` і ім'я тесту.

```
(define-builder-function fail (test)
  (print-char builder #\x)
  (push test (failed-tests builder)))

(define-builder-function pass (test)
  (print-char builder #\.)
  (push test (passed-tests builder)))

(define-builder-function disable (test)
  (push test (disabled-tests builder)))
```

Рис.15 – Методи для зміни полів об'єкту класу `builder`

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		26

Для того, щоб тест потрапив у список failed-tests, passed-tests, disabled-tests, необхідно використовувати методи fail, pass, disable відповідно. Дані методи створюються за допомогою макроса-обгортки для полегшення читання коду сторонніми розробниками, і збільшення його компактності. Якщо тест додається у failed-tests, то у потік виводу виводиться символ «x», а passed-tests – символ «.». У випадку коли тест ігнорується, то нічого не виводиться

Також у цьому пакеті була створена функція(рис.16), яка приймає в якості аргументів шлях до файлу документації і список спеціальних об'єктів test-info (рис.17), які були створені для зручного внутрішнього представлення тестів.

```
define-builder-function update-docfile (pathname tests)
```

Рис.16 – Інтерфейс функції update-docfile

```
(defstruct (test-info (:conc-name test-))  
  id  
  name  
  header-name  
  description  
  options  
  result)
```

Рис.17 – Структура test-info

Структура test-info складається з таких полів:

- id – унікальне id, яке присвоюється тесту на етапі компіляції. Необхідне для впорядкованого додавання тестів у документації, у випадках, коли вони там відсутні;
- name – лісовий символ. Необхідний для додавання у списки тестів,

які пройшли, не пройшли або були проігноровані;

- header-name – рядок, який містить перетворене ім'я лісового символу у ім'я org-mode заголовку. Необхідне для пошуку даного тесту в документації;
- description – рядок, який містить опис тесту;
- options – рядок виду: «опція1: значення1...»;
- result – рядок, який містить текстове представлення результату.

На початку роботи update-docfile:

- документ перетворюється у дерево розбору у вигляді об'єкту типу term (див. нижче в docgen/org/core);
- створюється пустий список нових піддерев;
- створюється пуста хеш-табличка заміни старих піддерев на нові.

Потім кожен об'єкт типу test-info паралельно перевіряється у наступному порядку:

- якщо у поле result встановлене значення :disabled, то захоплюється м'ютекс і викликається функція disable;
- якщо у дереві розбору знайдено піддерево, яке представляє собою заголовок з іменем header-name, то об'єкт типу test-info перетворюється у об'єкт типу term, який потім порівнюється з знайденим піддеревом. Захоплюється м'ютекс. У випадку рівності викликається функція pass, у іншому випадку викликається функція fail і в хеш-табличку додається запис;
- у інших випадках об'єкт типу test-info перетворюється у об'єкт типу term, захоплюється м'ютекс, викликається функція fail і об'єкт додається до списку нових піддерев.

У випадку, якщо хоча-б один тест відрізняється, користувачу пропонується два варіанти:

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		28

- подивитися різницю в meld;
- нічого не робити.

В першому випадку створюється нове піддерево, яке містить зміни з попереднього етапу. Потім створюється тимчасовий файл, в який буде записане перетворене у текст нове дерево, і показана різниця між тимчасовим файлом і вхідним файлом документації. Користувач на свій розсуд може додати зміни, які він вважає прийнятними. В другому випадку нічого не відбувається.

Також в даному пакеті реалізована функція collect-test-infos (рис.18), яка паралельно перетворює лісові символи, що представляють собою функцію, у об'єкти test-info, або у випадку помилки додають тест у failed-tests.

```
define-builder-function collect-test-infos (tests)
```

Рис.18 – Інтерфейс функції collect-test-infos

Функція report (рис.19) – виводить в потік виводу інформацію про тести з об'єкту builder.

```
define-builder-function report (print-passed print-failed print-disabled)
```

Рис.19 – Інтерфейс функції report

Приймає такі параметри:

- print-passed – сигналізує про те, чи потрібно виводити імена тестів, які пройшли;

- `print-failed` – сигналізує про те, чи потрібно виводити імена тестів, які не пройшли;
- `print-disabled` – сигналізує про те, чи потрібно виводити імена тестів, які були проігноровані.

Ця функція завжди виводить кількість елементів в відповідних списках.

В цьому пакеті також реалізована узагальнена функція `to-doc` (рис.20), яку користувач має реалізувати згідно своїм потребам. Вона повинна повертати рядкове представлення вхідного параметру.

```
defgeneric to-doc (data)
```

Рис.20 – Інтерфейс узагальненої функції `to-doc`

`docgen/common/engine` (рис.21) – даний пакет містить реалізацію основних інтерфейсних функцій, які може використовувати людина, яка користується розробленою системою.

```
(uiop:define-package :docgen/common/engine
  (:use :cl
        :docgen/common/core
        :docgen/utils/all
        :docgen/org/all)
  (:export #:define-doctest
           #:disable-doctest
           #:run-doctests))
```

Рис.21 – Інтерфейс пакету `docgen/common/engine`

На рис.22 зображено оголошення глобальної змінної, яка представляє

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		30

собою хеш-табличку, з якою працюють в даному дипломному проєкті, ключами якої, є шляхи до файлів документації, а значеннями – імена тестів, які створенні для цього файлу.

```
(defvar *tests* (make-hash-table :test #'equal))
```

Рис.22 – Оголошення хеш-таблички тестів

На рис.23 зображено оголошення глобальної змінною, яка представляє собою унікальний ідентифікатор, який присвоюється кожному тесту під час його оголошення. Даний прийом дозволяє інтегрувати тести у файл документації у тому порядку, у якому вони були оголошені.

```
(defvar *id* 0)
```

Рис.23 – Оголошення унікального ідентифікатора

Функція register-test (рис.24) додає запис у глобальну хеш-табличку *tests* (див. вище). Викликається при оголошенні тесту

```
defun register-test (func-name pathname)
```

Рис.24 – Інтерфейс функції register-test

На рис.25 зображено інтерфейс основного макросу для створення тесту,

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		31

який пов'язується з документацією.

```
defmacro define-doctest (name pathname &body body)
```

Рис.25 – Інтерфейс макросу `define-doctest`

Даний макрос приймає має такі вхідні параметри:

- `name` – унікальне ім'я тесту;
- `pathname` – шлях до файлу документації, в який цей тест буде інтегрований;
- `body` – необмежена кількість форм, які будуть виконуватися під час виклику даного тесту. Передбачається, що остання форма повертає результат, який вважається результатом даного тесту, і буде порівнюватися вже з існуючим.

Під час оголошення тесту даний макрос розкривається у функцію, яка перетворює вхідні параметри цього тесту у внутрішнє подання тесту (об'єкт `test-info` (див. вище)):

Наступні дії залежать від вхідного параметру

- повертає внутрішнє подання, якщо ця функцію була викликана з аргументом `builder`;
- викликається функція `update-doc-file` у випадку , якщо `builder` не був переданий в якості вхідного параметру.

Розглянемо алгоритм перетворення у внутрішнє подання тесту:

- створити пустий об'єкт `test-info`;
- записати унікальний номер тесту, який був вирахований на етапі

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		32

компіляції у поле id;

- перетворити лісьове ім'я тесту у рядок за допомогою функції `convert-lisp-name-to-org-header` (див. оголошення в `docgen/org/utils`). Записати отримане значення у поле `header-name`;
- перевірити на існування вхідного файлу документації, і в разі його відсутності - створити новий;
- у випадку , якщо перша форма має тип `string`, то записати цей рядок у поле `description`. У іншому випадку – записати пустий рядок;
- виконати всі форми, крім останньої;
- якщо остання форма – це не функцію, то записати у поле `options` пустий рядок. У іншому випадку за допомогою спеціальної функції (`function-lambda-list` з бібліотеки `sb-introspect`) отримати список формальних параметрів, потім вирахувати список фактичних параметрів, і отримати список виду ((ім'я параметру1 . значення1) ...) за допомогою функції `match-formal-args-with-actual-args` (див. оголошення в `docgen/utils/common`). Обійти кожен елемент цього списку, викликати над другим елементом функцію `to-doc` (див. вище) і перетворити цей елемент у рядок виду «ім'я1: значення1 ...». Записати кінцевий результат у поле `options`;
- обчислити значення останньої форми. Над отриманим значенням викликати метод `to-doc`. Отриманий результат записати у поле `result`;
- повернути внутрішнє подання у випадку, якщо ця функція була викликана з вхідним параметром типу `builder`. У іншому випадку викликати функцію `update-docfile`.

Макрос `disable-doctest` (рис.26) дозволяє ігнорувати заданий тест, і не враховувати його під час оновлення файлу документації. Він приймає на вхід оголошення тесту за допомогою макросу `define-doctest`.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		33

```
defmacro disable-doctest (doctest)
```

Рис.26 – Оголошення макросу `disable-doctest`

Функція `run-doctests` (рис.27) , виконує тести, які були скомпільовані і знаходяться у таблиці `*tests*`.

```
defun run-doctests (&key
                  pathnames
                  (stream *standard-output*)
                  print-passed
                  (print-failed t)
                  print-disabled
                  run-meld-for)
```

Рис.27 – Оголошення функції `run-doctests`

Дана функція приймає такі вхідні параметри:

- `pathnames` – список, кожен елемент якого – це шлях до файлу документації, для якого повинні бути викликані тести. У разі, якщо цей параметр не був переданий, викликаються всі тести, які знаходяться в таблиці `*tests*`;
- `stream` – потік в який буде виведена вся інформація, що стосується тестів. За замовчування – це стандартний потік виводу;
- `print-passed` – булева змінна, яка вказує чи потрібно виводити імена тестів, які пройшли. За замовчуванням імена не виводяться;
- `print-failed` – булева змінна, яка вказує чи потрібно виводити імена тестів, які не пройшли. За замовчуванням імена виводяться;
- `print-disabled` – булева змінна, яка вказує чи потрібно виводити імена тестів, які були проігноровані. За замовчування імена не

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		34

виводяться;

- `run-meld-for` – список, кожен елемент якого – це шлях до файлу документації, для якого, у випадку, якщо хоча-б один тест не пройшов, показується різниця в `meld-i`;

Розглянемо більш детально алгоритм:

- записати у локальну змінну `pathnames` значення вхідного параметру `pathnames`, а у випадку його відсутності, всі ключі таблицьки `*tests*`;
- записати у локальну змінну `tests` імена всіх тестів, які були створені для файлів, шляхи до яких, містяться у змінній `pathnames`;
- створити об'єкт класу `builder`, і записати у поле `stream-output` – значення вхідного параметру `stream`;
- вирахувати кількість паралельних потоків, які повинні бути створені за допомогою наступної формули:

$$N = p + t$$

де p – довжина списку `pathnames`, t – довжина списку `tests`;

- перетворити список `tests`, який містить імена тестів, у список об'єктів `test-infos`;
- паралельно обійти список об'єктів `test-infos`, і викликати функцію `update-docfile`. У випадку, якщо, хоча-б один тест не пройшов, і шлях до файлу належить списку `run-meld-for` - показати різницю в `meld`;
- викликати функцію `report`, яка виводить загальну інформацію про тести, які не пройшли, пройшли, а також були проігноровані.

`docgen/common/all` (рис.28) – компонентний пакет системи.


```
(uiop:define-package :docgen/common/all
  (:nicknames :docgen)
  (:use :docgen/common/core)
  (:use-reexport :docgen/common/engine))
```

Рис.28 – Оголошення пакету docgen/common/all

Варто зазначити, що на відміну від пакету docgen/common/core, пакет docgen/common/engine, буде не тільки скомпільований, також всі символи, а саме: disable-doctest, define-doctest, run-doctests – будуть експортовані з цього пакету для зовнішнього користування. Для більш зручної взаємодії з даною системою, даний пакет має так званий «nickname», що дозволяє використувати коротке ім'я – docgen, замість довгого імені пакету – docgen/common/all.

3.3 Модуль org

org - містить функції, для перетворення org-mode файлів у внутрішнє представлення, а також перетворення внутрішнього представлення у текст (відповідно і у файл). Складається з 5 пакетів.

docgen/org/core (рис.29) – пакет, який містить реалізацію структур даних, для представлення дерева розбору org-mode елементів.

```
(uiop:define-package :docgen/org/core
  (:use :cl :docgen/utils/all)
  (:export #:make-term
           #:make-term-from-list
           #:term-p
           #:term-head
           #:term-components
           #:compare-terms
           ;; slots
           #:head
           #:components))
```

Рис.29 – Оголошення пакету docgen/org/core

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		36

Term (рис.30) – це внутрішнє представлення дерева. Дана структура містить 2 поля:

- head – ім'я вузла дерева;
- components – список піддерев.

```
(defstruct term
  head
  components)
```

Рис.30 – Структура term

Функція make-term-from-list (рис.31) перетворює вхідний список у об'єкт типу term.

```
defun make-term-from-list (list)
```

Рис.31 – Інтерфейс функції make-term-from-list

Розглянемо більше детально алгоритм перетворення:

- створити пустий об'єкт типу term;
- у поле head записати значення першого елемента вхідного параметру;
- обійти хвіст списку. У разі якщо, поточний елемент – це також список, то рекурсивно викликати функцію make-term-from-list, і в якості параметру передати їй даний елемент. Результат додати в кінець поля components. В іншому випадку, також додати значення елемента в кінець.

Функція `compare-terms` (рис.32) порівнює два об'єкти типу `term` і повертає `t` у випадку рівності. В іншому випадку – `nil`.

```
defgeneric compare-terms (term1 term2)
```

Рис. 32 – Інтерфейс функції `compare-terms`

Розглянемо більш детально алгоритм порівняння:

- якщо один елемент – це об'єкт типу `term`, а інший – ні, то повернути `nil`;
- якщо обидва елементи не являються об'єктами типу `term`, то порівняти їх значення. У випадку рівності повернути `t`, у іншому випадку - `nil`;
- якщо обидва елементи, мають тип `term`, то спочатку порівняти значення полів `head`;
- якщо значення полів `head` рівні, то порівняти довжини списків, які лежать у полі `components`;
- у випадку рівності довжин, порівняти кожен елемент з першого списку `components` з відповідним йому елементом з другого списку `components` за допомогою функції `compare-terms`. У разі, якщо кожен елемент аналогічний відповідному йому елементу, повернути `t`, у іншому випадку – `nil`;

`docgen/org/dump` (рис.33) – містить функцію перетворення об'єкту типу `term` у текстове представлення.

```
(uiop:define-package :docgen/org/dump
  (:use :cl :docgen/org/core)
  (:import-from :alexandria
    #:plist-alist)
  (:export #:dump-org))
```

Рис.33 – Інтерфейс пакету docgen/org/dump

Функція dump-org (рис.34) приймає такі 2 аргументи:

- stream – потік виводу, у який буде виведено перетворений об'єкт типу term;
- term – об'єкт типу term, який повинен бути перетворений у текстове представлення.

```
defun dump-org (stream term)
```

Рис.34 – Інтерфейс функції dump-org

Розглянемо алгоритм роботи даної функції:

- якщо вхідний аргумент має тип string, то повернути його значення;
- якщо вхідний аргумент не має тип term, то повернути пустий рядок;
- якщо вхідний аргумент має тип term, то знайти правило перетворення цього типу дерева у текст за значенням поля head. На рис.35 зображено приклад правила і його обробку.

```
((:org :entry)
 (format nil "~A~A~{~A~}"
  (%dump (first components))
  (%dump (second components))
  (mapcar #'%dump (cddr components))))
```

Рис.35 – Приклад правила перетворення

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		39

docgen/org/parser (рис.36) – містить функції перетворення текстового представлення org-mode елементів у об'єкт типу term.

```
(uiop:define-package :docgen/org/parser
  (:use :cl
        :docgen/org/core
        :docgen/utils/all)
  (:import-from :alexandria
                #:read-file-into-string)
  (:import-from :parser-combinators
                #:*parser-cache*)
  (:import-from :cl-org-mode-raw
                #:org-raw-parse)
  (:export #:load-org
           #:parse-org-header))
```

Рис.36 – Інтерфейс пакету docgen/org/parser

Функція load-org (рис.37) перетворює вхідний аргумент у об'єкт типу term.

```
defun load-org (org)
```

Рис.37 – Інтерфейс функції load-org

Розглянемо більш детально алгоритм перетворення:

- якщо вхідний параметр - це шлях до файлу, то зчитати файл у текстовий буфер;
- якщо вхідний параметр – це текст, то записати його у текстовий буфер;
- перетворити текстовий буфер у список за допомогою функції org-

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		40

raw-parse, яка імпортується з бібліотеки cl-org-mode;

- перетворити список у об'єкт типу term за допомогою функції make-term-from-list.

Функція parse-org-header (рис.38) перетворює текстове представлення org-mode заголовку у об'єкт типу term.

```
defun parse-org-header (string)
```

Рис.38 – Інтерфейс функції parse-org-header

Розглянемо алгоритм перетворення:

- викликати функцію load-org і отримати об'єкт типу term;
- так як після виклику ми отримаємо представлення всього org-mode файлу, але нам потрібно лише піддерево, що представляє цей заголовок, то взяти з поля components третій елемент, що згідно граматиці, яка описана в бібліотеці cl-org-mode, відповідає заголовку.

docgen/org/utils (рис.39) – містить функції для обробки внутрішнього представлення org-mode файлів.

```
(uiop:define-package :docgen/org/utils
  (:use :cl :docgen/utils/all :docgen/org/core)
  (:export #:find-entry-by-name
           #:get-depth-of-entry
           #:convert-lisp-name-to-org-header
           #:replace-nodes-in-tree))
```

Рис.39 – Інтерфейс пакету docgen/org/utils

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		41

Функція `find-entry-by-name` (рис.40) повертає піддерево, що представляє `org-mode` заголовок, який має ім'я – `name`.

```
defun find-entry-by-name (name term)
```

Рис.40 – Інтерфейс функції `find-entry-by-name`

Розглянемо алгоритм пошуку:

- розділити вхідне ім'я на слова за допомогою функції `split-string`(див. реалізацію в `docgen/utills/common`);
- якщо вхідний аргумент має тип `term`:
 1. перевірити, чи представляє цей терм `org-mode` заголовок;
 2. якщо так, то розділити ім'я на слова за допомогою функції `split-string`, і порівняти з вхідним;
 3. у випадку рівності повернути об'єкт;
 4. у разі невиконання хоча-б однією умови, викликати функцію рекурсивно над кожним елементом з поля `components`, поки не буде знайдений потрібний об'єкт.
- у інших випадках повернути `nil`;

Функція `name-of-header`(рис.41) повертає ім'я вхідного параметру, що представляє `org-mode` заголовок.

```
defun name-of-header (headline)
```

Рис.41 – Інтерфейс функції `name-of-header`

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		42

Функція `get-depth-of-entry` (рис.42) повертає кількість зірочок, які відповідають вхідному параметру, що представляє `org-mode` заголовок.

```
defun get-depth-of-entry (term)
```

Рис.42 – Інтерфейс функції `get-depth-of-entry`

Функція `replace-nodes-in-tree` (рис.43) замінює об'єкти типу `term`, що мають заміну, яка містить в `repl-map`.

```
defun replace-nodes-in-tree (term repl-map)
```

Рис.43 – Інтерфейс функції `replace-nodes-in-tree`

Розглянемо алгоритм заміни:

- якщо вхідний параметр має тип `term` то:
 1. створити пустий об'єкт типу `term`;
 2. записати у поле `head` нового об'єкту, значення поля `head` вхідного об'єкту;
 3. обійти поле `components`, і у разі , замінити елемент новим(у випадку, якщо заміна існує) або рекурсивно викликати функції `replace-nodes-in-tree`. Результат додати в кінець поля `components` нового об'єкту.
- якщо вхідний параметр має інший тип, то повернути значення цього параметру.

Функція `convert-lisp-name-to-org-header` (рис.44) перетворює вхідний параметр, що має тип лісового символу, у відповідне йому ім'я `org-mode` заголовку.

```
defun convert-lisp-name-to-org-header (symbol)
```

Рис.44 – Інтерфейс функції `convert-list-name-to-org-header`

Розглянемо алгоритм перетворення:

- отримати текстове представлення лісового символу;
- перетворити всі дефіси на пробіли;
- першу літеру записати у верхньому реєстрі, всі останні у нижньому.

`docgen/org/all` (рис.45) - компонує всі вище описані пакети, і експортує всі символи, які експортуються з них.

```
(uiop:define-package :docgen/org/all  
  (:use-reexport #:docgen/org/core  
                 #:docgen/org/utils  
                 #:docgen/org/parser  
                 #:docgen/org/dump))
```

Рис.45 – Інтерфейс пакету `docgen/org/all`

3.4 Модуль `utils`

`utils` – містить утиліти, які так чи інакше використовуються в пакетах системи. Складається з 3 пакетів.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		44

docgen/utils/common (рис.46) – містить функції загального користування, а також експортує символи з різних пакетів, які потім використовуються в пакетах описаних вище.

```
(uiop:define-package :docgen/utils/common
  (:use :cl :docgen/utils/macros)
  (:import-from :alexandria
    #:hash-table-keys
    #:parse-ordinary-lambda-list)
  (:import-from :sb-ext
    #:run-program)
  (:import-from :sb-introspect
    #:function-lambda-list)
  (:export #:hashset-from-list
    #:hashset-member
    #:hashset-insert
    #:hashset-to-list
    #:hash-table-keys
    #:split-string
    #:compare-strings-without-whitespaces
    #:string-empty-p
    #:ensure-file-exist
    #:lists-equal
    #:show-difference-in-meld
    #:match-formal-args-with-actual-args
    #:function-lambda-list))
```

Рис.46 – Інтерфейс пакету docgen/utils/common

Функція split-string (рис.47) розділяє вхідний рядок на слова, які розділені певними символами. Повертає список слів.

```
defun split-string (string &optional (separators '(\Space #\Tab #\Newline)))
```

Рис.47 – Інтерфейс функції split-string

Вона приймає 2 аргументи:

- рядок, який буде розділений;

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		45

- список символів, по яким будуть розділені слова. За замовчування містить символи табуляції, нового рядку та пробілу.

Розглянемо алгоритм розділення:

- якщо перший символ не роздільник, то записати його у акумулятор. Викликати рекурсивно функцію `split-string` над вхідним рядком без першого символу;
- якщо поточний символ роздільник, то додати значення акумулятора в кінець результату. Очистити акумулятор;
- викликати рекурсивно функцію `split-string` над вхідним рядком без першого символу;
- якщо рядок закінчився, то видалити пусті рядки з результату і перетворити кожне слово у верхній реєстр;

Функція `string-empty-p` (рис.48) повертає `t` у випадку, якщо вхідний параметр – це пустий рядок, у іншому випадку – `nil`.

```
defun string-empty-p (string)
```

Рис.48 – Інтерфейс функції `string-empty-p`

Функція `ensure-file-exist` (рис.49) перевіряє чи існує файл за шляхом, який переданий в якості вхідного параметру, і у випадку його відсутності створює новий файл.

```
defun ensure-file-exist (pathname)
```

Рис.49 – Інтерфейс функції `ensure-file-exist`

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		46

Функція `lists-equal`(рис.50) перевіряє вхідні списки на рівність.

```
defun lists-equal (list1 list2 &key (test #'equal) (key #'identity))
```

Рис.50 – Інтерфейс функції `lists-equal`

Приймає 4 аргументи:

- `list1` – перший список;
- `list2` – другий список;
- `test` – функція для порівняння відповідних елементів списків;
- `key` – функція, яка викликається над кожним елементом кожного списку.

Розглянемо алгоритм порівняння:

- якщо довжини списків відрізняються повернути `nil`;
- якщо довжини списків рівні, то перевірити на рівність, відповідні елементи списків за допомогою функції `test`, попередньо викликав над ними функцію `key`. У випадку рівності всіх елементів повернути `t`, у протилежному випадку – `nil`.

Функція `show-difference-in-meld` (рис.51) викликає `meld` і показує різницю між файлами. Приймає 2 аргументи – шляхи до файлів.

```
defun show-difference-in-meld (path1 path2)
```

Рис.51 – Інтерфейс функції `shod-difference-in-meld`

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		47

Функція `match-formal-args-with-actual-args` (рис.52) зіставляє формальні параметри з фактичними.

```
defun match-formal-args-with-actual-args (lambda-list actual)
```

Рис.52 – Інтерфейс функції `match-formal-args-with-actual-args`

Розглянемо більш детально алгоритм зіставлення формальних параметрів з фактичними:

- у випадку, якщо список актуальних параметрів не пустий, то:
 1. якщо список обов'язкових параметрів не пустий, то додати до результату список, перший елемент якого – це вилучене ім'я зі списку обов'язкових параметрів, а другий – вилучене значення зі списку фактичних параметрів, виконати п.1. У іншому випадку виконати п.2.
 2. якщо список опціональних параметрів не пустий, то додати до результату список, перший елемент якого – це вилучене ім'я зі списку опціональних параметрів, а другий – вилучене значення зі списку фактичних параметрів, виконати п.1. У іншому випадку виконати п.3.
 3. якщо існує `rest` параметр, то додати до результату список де перший елемент – це ім'я `rest` параметру, а хвіст цього списку- це список фактичних параметрів, які лишились. Очистити значення `rest` параметру. У випадку, коли `allow-other-keys` присутній то виконати п.4, у іншому випадку – очистити список фактичних параметрів.
 4. вилучити ймовірне ім'я ключового параметру зі списку

фактичних параметрів. Якщо існує ключовий параметр з таким ім'ям, то додати до результату список, перший елемент якого – це вилучене ім'я зі списку ключових параметрів, а другий –вилучене значення зі списку фактичних параметрів. Виконати п.1.

- у випадку, якщо список актуальних параметрів пустий, то:
 1. обійти список опціональних параметрів, які лишились, і додати до результату список , де перший елемент – це ім'я параметру, а другий – це значення за замовчуванням;
 2. виконати те саме, що описано в п.1, але над списком ключових параметрів.

docgen/utils/macros (рис.53) – пакет, який містить реалізацію зручних макросів.

```
(uiop:define-package :docgen/utils/macros
  (:use :cl)
  (:import-from :bordeaux-threads
    #:make-lock
    #:with-lock-held)
  (:import-from :lparallel
    #:*kernel*
    #:make-kernel
    #:end-kernel
    #:pmapc
    #:pmapcar
    #:premove-if
    #:plet
    #:pevery
    #:task-handler-bind)
  (:export #:aand
    #:acond
    #:awhen
    #:aif
    #:mv-let*
    #:it
    #:with-parallel
    #:pmapc
    #:pmapcar
    #:premove-if
    #:plet
    #:pevery
    #:task-handler-bind
    #:make-lock
    #:with-lock-held))
```

Рис.53 – Інтерфейс пакету docgen/utils/macros

Макрос `with-parallel` – це обгортка, яка створює об’єкт типу `kernel` з бібліотеки `lparallel` (див. попередній розділ), і дозволяє викликати в його контексті паралельні альтернативи стандартним функціям `Common Lisp`.

Приймає кількість паралельних процесів, так званих `workers`, які будуть виконувати операції.

Макрос `aand` робить те саме, що і відповідний макрос зі стандарту `Common Lisp`, за виключенням того, що результат виконання попередньої форми запам’ятовується у змінну `it`.

Макроси `awhen`, `aif`, `ascond` виконують ті самі дії, що і стандартні макроси `Common Lisp`, за винятком того, що значення умови запам’ятовується у змінну `it`. Варто зазначити, що змінна `it` може використовуватись тільки у разі, успішного виконання умови.

Макрос `mv-let*` поєднує в собі два стандартних макроси `let` і `multiple-value-bind`.

Також цей пакет експортує паралельні альтернативи стандартним функціям `Common Lisp` з бібліотеки `lparallel`, а також функцію для створення об’єкту м’ютексу і макрос для роботи з ним з бібліотеки `bordeaux-threads`.

`docgen/utils/all` (рис.54) – компонує два вище описаних пакети, і експортує символи, які вони експортують.

```
(uiop:define-package :docgen/utils/all
  (:use-reexport :docgen/utils/common
                 :docgen/utils/macros))
```

Рис.54 – Інтерфейс пакету `docgen/utils/all`

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		50

3.5 Висновки

В даному розділі дипломного проєкту було розглянуто архітектуру розробленої системи. Вона має 3 головних модуля, кожен з яких виконує покладену на нього задачу:

- `common` – містить реалізацію основних інтерфейсних функцій для створення тестів, а також отримання актуальної інформації про тести, які співпадають або не співпадають з тими, що знаходяться в документації і ті, які ігноруються;
- `org` – містить засоби для розбору `org-mode` елементів (файл, заголовки), зручні функції для роботи з ними, а також перетворення їх внутрішнього представлення у текстовий формат;
- `utils` – містить засоби для обробки різних структур даних, макроси для введення нових синтаксичних конструкцій, які полегшують читання коду, а також експортує різні корисні функції з зовнішніх бібліотек для позбавлення зайвих залежностей від них в основних модулях системи.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		51

4.ТЕСТУВАННЯ СИСТЕМИ

4.1 Тестування

Для тестування системи створимо умови, при яких звичайний програміст може використовувати розроблений проєкт.

Створимо окремий пакет `docgen/test` (рис.55), який буде імпортувати основні функції з розробленого продукту, а саме: `define-doctest`, `disable-doctest`.

```
(uiop:define-package :docgen/test
  (:use :cl :docgen))
```

Рис.55 – Інтерфейс пакету для тестів

Створимо декілька тестів за допомогою макросу `define-doctest` та розглянемо поведінку розробленої системи при різних вхідних даних.

На рис.56 показано оголошення тесту в якому викликається певна тестова функція (рис.57). Дана функція приймає на вхід 3 аргументи: `x` та `y` — обов'язкові і `z` — опціональний, що має значення за замовчуванням - 0, і обчислює їх суму. Даний тест пов'язується з неіснуючим файлом документації, який має шлях - `~/tmp/test1.org`.

```
(define-doctest example-1 "~/tmp/test1.org"
  (test-fn-1 1 2))
```

Рис.56 – Приклад оголошення тесту

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		52

```
(defun test-fn-1 (x y &optional (z 0))  
  (+ x y z))
```

Рис.57 – Інтерфейс тестової функції

Запускаємо тест (рис.58) так як цей тест новий для нового файлу, то бачимо вікно, яке зображено на рис.59.

```
DOCGEN/TEST> (example-1)
```

Рис.58 – Приклад запуску тесту

```
Tests from ~/tmp/test1.org are failed.  
[Condition of type SIMPLE-ERROR]  
  
Restarts:  
0: [RUN-MELD] Show difference in meld and apply needed changes.  
1: [SKIP] Do nothing.  
2: [RETRY] Retry SLIME REPL evaluation request.  
3: [*ABORT] Return to SLIME's top level.  
4: [ABORT] abort thread (#<THREAD "repl-thread" RUNNING {1003E007B3}>)
```

Рис.59 – Вікно вибору

Як бачимо на вибір користувачу пропонуються 2 дії:

- показати різницю в meld і здійснити потрібні зміни;
- нічого не робити.

Виберемо перший пункт. Після цієї дії буде показана різниця в meld (рис.60). Зліва зображено вміст основного файлу, а справа — тимчасовий файл, в який було записано перетворене дерево.

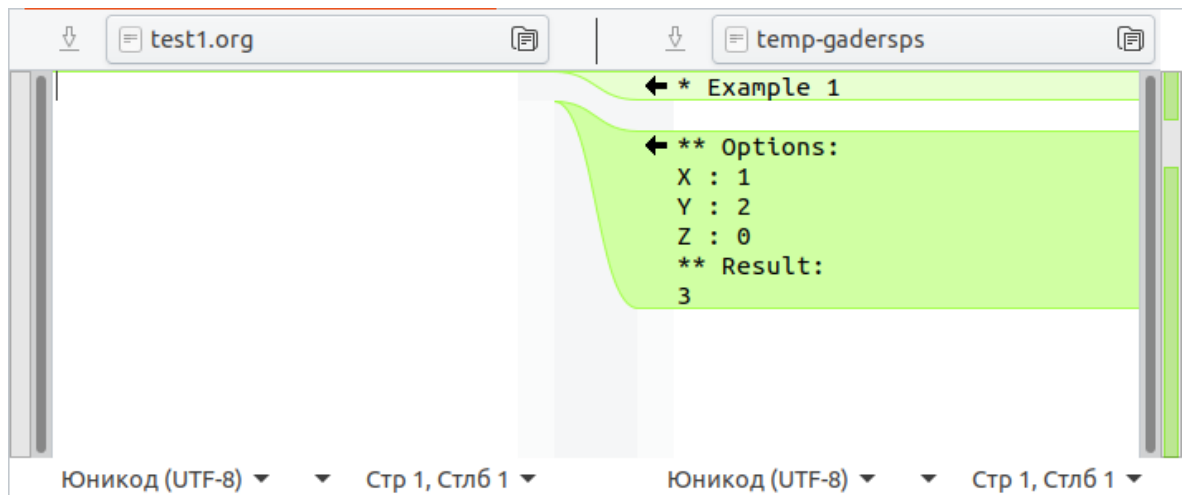


Рис.60 – Результат запуску тесту

Натиснемо на стрілку, тим самим змінюючи файл документації, збережемо внесені зміни.

Додамо аналогічний тест, але який передає в якості z — конкретне значення - 3. Запустимо даний тест і отримаємо такий результат в meld (рис.61).

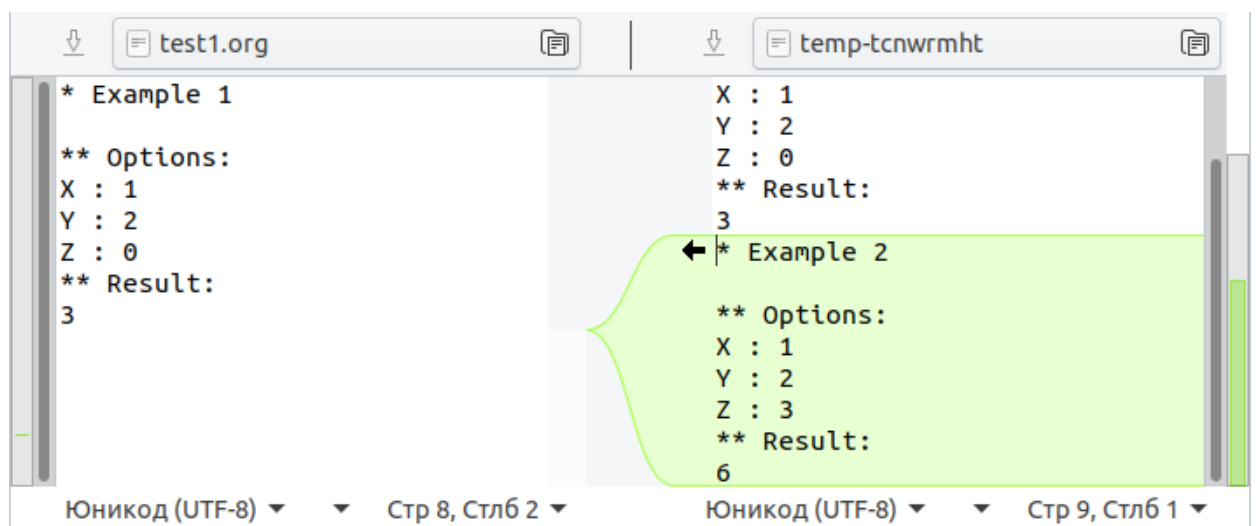


Рис.61 – Результат запуску другого тесту

Як бачимо, пропонується додати даний тест до існуючих. Додамо цей тест у документацію і збережемо внесені зміни.

Розглянемо процес оновлення тестів у випадках, коли представлення тесту зміниться.

- 1) Додамо до першого тесту опис. Після опису тест прийме такий вигляд (рис.62).

```
(define-doctest example-1 "~/tmp/test1.org"
 "This test return sum of 1 and 2."
 (test-fn-1 1 2))
```

Рис.62 – Тест з описом

Запустимо тест і отримаємо такий результат (рис.63). Додамо зміни і збережемо.

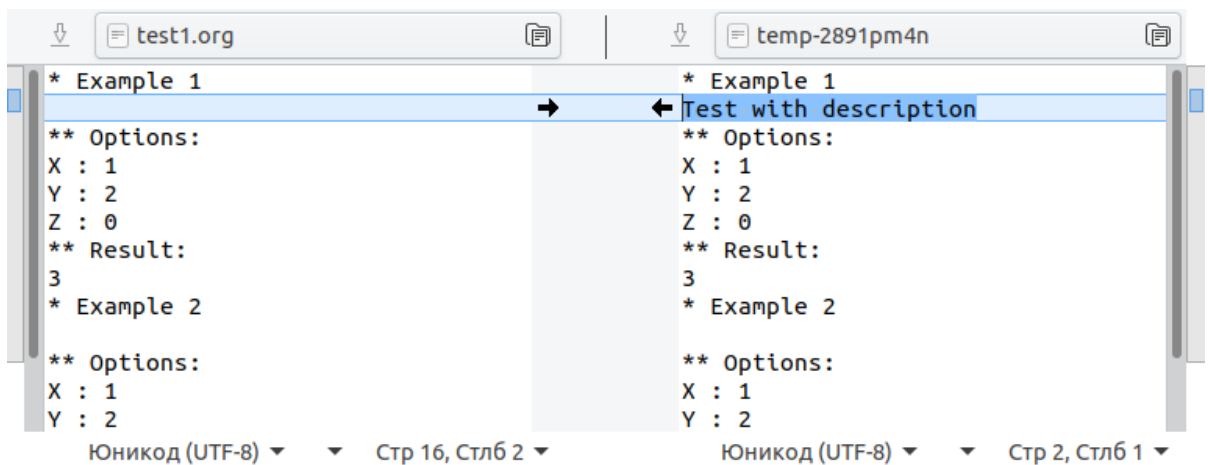


Рис.63 – Результат роботи тесту

- 2) Змінимо поведінку тестової функції, щоб замість суми вхідних аргументів, поверталася їх різниця і запустимо перший тест. Результат роботи показано на рис.64.

```

test1.org | temp-p6b7dnqc
* Example 1 | * Example 1
Test with description | Test with description
** Options: | ** Options:
X : 1 | X : 1
Y : 2 | Y : 2
Z : 0 | Z : 0
** Result: | ** Result:
3 | -1
* Example 2 | * Example 2
** Options: | ** Options:
X : 1 | X : 1
Y : 2 | Y : 2
Юникод (UTF-8) | Стр 16, Стлб 2 | Юникод (UTF-8) | Стр 8, Стлб 1

```

Рис.64 – Результат роботи тесту

3) Додамо ще один опціональний параметр *w* , який також має значення за замовчуванням — 0. Результат запуску тесту показаний на рис.65

```

test1.org | temp-3fjerk54
* Example 1 | * Example 1
Test with description | Test with description
** Options: | ** Options:
X : 1 | X : 1
Y : 2 | Y : 2
Z : 0 | Z : 0
** Result: | ** Result:
-1 | -1
* Example 2 | * Example 2
** Options: | ** Options:
X : 1 | X : 1
Y : 2 | Y : 2
w : 0 |
Юникод (UTF-8) | Стр 16, Стлб 2 | Юникод (UTF-8) | Стр 7, Стлб 1

```

Рис.65 – Результат роботи тесту

Протестуємо функцію `run-doctests` з різними вхідними параметрами.

Для тестування створимо тестову функцію (рис.66) , яка приймає 2 аргументи і повертає їх добуток, а також створимо тести для 2-х файлів документації (рис.67)

```
(defun test-fn (x y)
  (* x y))
```

Рис.66 – Інтерфейс тестової функції

```
(define-doctest example-1-for-file-1 "~/tmp/test1.org"
  "Test with description"
  (test-fn 1 2))

(define-doctest example-2-for-file-1 "~/tmp/test1.org"
  (test-fn 3 4))

(define-doctest example-1-for-file-2 "~/tmp/test2.org"
  "Test with description"
  (test-fn 5 6))

(define-doctest example-2-for-file-2 "~/tmp/test2.org"
  (test-fn 7 8))
```

Рис.67 – Оголошення тестів

Викличемо з інтерпретатора функцію `run-doctests` без параметрів і отримаємо результат зображений на рис.68.

```
Tests:
xxxx
Failed:
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-2
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-2
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-1
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-1
Passed: 0
Failed: 4
Disabled: 0
```

Рис.68 – Результат роботи

Запустимо функцію run-doctests, і передамо в якості аргументу run-meld-for, список, який складається з одного елемента - шлях до першого файлу. В результаті виконання функції, буде показана різниця в meld (рис.69)

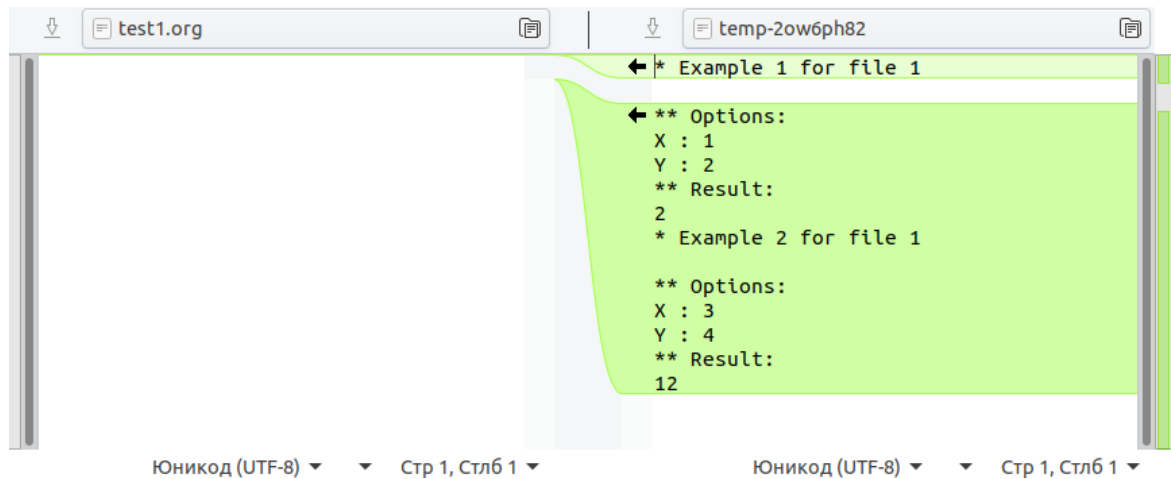


Рис.69 – Результат роботи

Додамо дані тести до файлу, і збережемо результат.

Викличемо функцію run-doctests, але передамо аргумент print-passed, встановлений в t і отримаємо результат, зображений на рис.70.

```
Tests:
xx..
Passed:
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-1
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-1
Failed:
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-2
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-2
Passed: 2
Failed: 2
Disabled: 0
```

Рис.70 – Результат роботи

Викличемо функцію `run-doctests`, і передамо в якості аргументу `pathnames`, список, який складається з одного елемента — шлях до першого файлу і отримаємо результат, зображений на рис.71.

```
Tests:
..
Passed: 2
Failed: 0
Disabled: 0
```

Рис.71 – Результат роботи

Протестуємо механізм ігнорування тестів. Для ігнорування був розроблений макрос `disable-doctest`. Обгорнемо оголошення тестів, які були створені для файлу документації, який знаходиться за шляхом - `~/tmp/test2.org`.

На рис.72 показано приклад оголошення тестів, які будуть ігноруватися.

```
(disable-doctest
 (define-doctest example-1-for-file-2 "~/tmp/test2.org"
  (test-fn 5 6)))

(disable-doctest
 (define-doctest example-2-for-file-2 "~/tmp/test2.org"
  (test-fn 7 8)))
```

Рис.72 – Оголошення тестів, які ігноруються

Запустимо `run-doctests`, але передамо в якості вхідного аргументу `print-disabled` — значення `t` і отримаємо результат, який зображений на рис.73.


```
Tests:
..
Disabled:
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-2
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-2
Passed: 2
Failed: 0
Disabled: 2
```

Рис.73 – Результат роботи

Змінимо порядок тестів для файлу - ~/tmp/test1.org і запусимо run-doctests без параметрів. Як видно з рис.74 результат виконання не змінився, що свідчить про коректну роботу системи.

```
Tests:
..
Passed: 2
Failed: 0
Disabled: 2
```

Рис.74 – Результат роботи

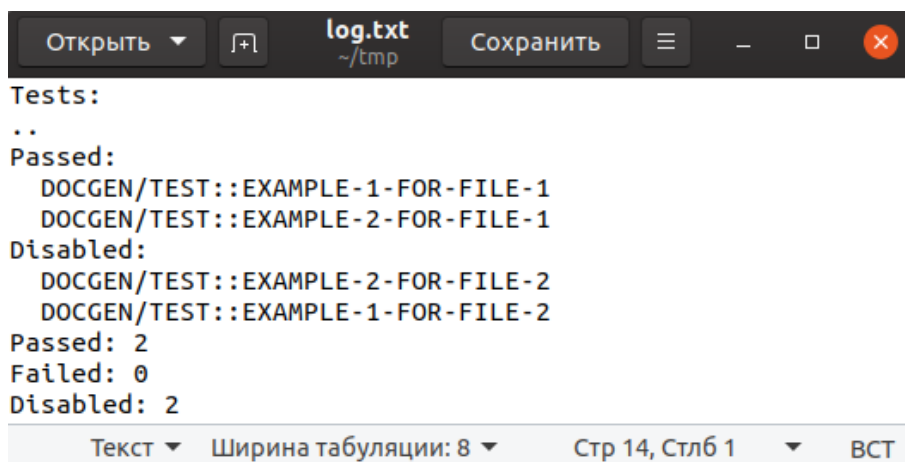
Також run-doctests, має параметр stream — потік у який буде здійснено вивід інформації про тести. Створимо файловий дескриптор, та викличемо функцію run-doctests (рис.75) і передамо в якості аргументу stream — цей дескриптор. Також встановимо аргументи print-passed і print-failed у значення t.

```
CL-USER> (with-open-file (s "~/tmp/log.txt"
                        :if-exists :overwrite
                        :if-does-not-exist :create
                        :direction :output)
          (docgen:run-doctests :stream s :print-disabled t :print-passed t))
```

Рис.75 – Приклад виклику функції run-doctests з файловим дескриптором

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		60

Вміст файлу log.txt після виклику функції зображений на рис.76.



```
Открыть  log.txt  Сохранить  ~/tmp
Tests:
..
Passed:
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-1
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-1
Disabled:
  DOCGEN/TEST::EXAMPLE-2-FOR-FILE-2
  DOCGEN/TEST::EXAMPLE-1-FOR-FILE-2
Passed: 2
Failed: 0
Disabled: 2
Текст  Ширина табуляції: 8  Стр 14, Стлб 1  ВСТ
```

Рис.76 – Вміст файлу

Протестуємо роботу узагальненого методу to-dos, який повертає текстове представлення вхідного аргументу, який він отримує.

В дипломному проєкті було реалізовано методи to-dos для простих вбудованих типів, такі як: хеш-таблички, списки, булеві значення, а також символи. Протестуємо деякі з них

Створимо функцію (рис.77), яка приймає степінь і список чисел, повертає результат у вигляді хеш-таблички, де ключ — це число, а значення — ключ, який піднесений до степеню.

```
(defun pow (power &rest numbers)
  (let ((hmap (make-hash-table :test #'eql)))
    (mapc (lambda (number)
            (assert (numberp number) nil "Number is expected.")
            (setf (gethash number hmap) (expt number power)))
          numbers)
    hmap))
```

Рис.77 – Інтерфейс тестової функції

Створимо декілька тестів і протестуємо роботу розробленої системи(рис.78).

```
(define-doctest example-1 "~/tmp/test1.org"
  (pow 2 1 2 3))

(define-doctest example-2 "~/tmp/test1.org"
  (pow 3 3 4 5))
```

Рис.78 – Оголошення тестів

Виконаємо метод run-doctests, передаючи в якості аргументу run-meld-for — список, який складається з одного елементу — шлях ~/tmp/test1.org. Результат роботи показано на рис.79.

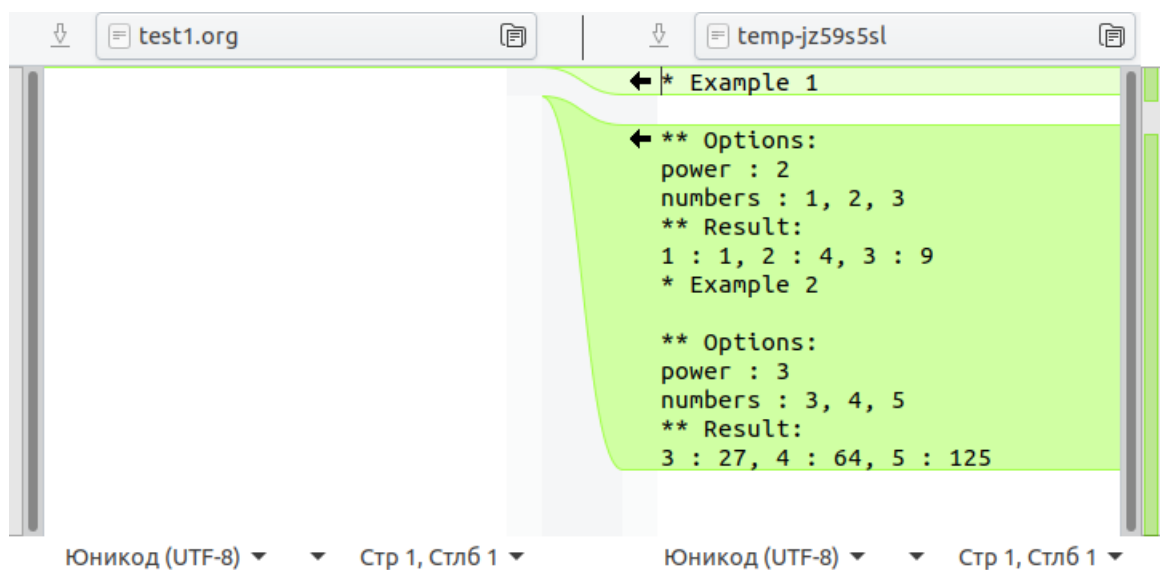


Рис.79 – Результат роботи

Також метод to-doc можна реалізувати для класів, які створив сам користувач. Для прикладу створимо тестовий клас (рис.80).

```
(defclass test-class ()
  ((x :initarg :x :reader x)
   (y :initarg :y :reader y)
   (sum :initarg :sum :reader sum)))
```

Рис.80 – Клас test-class

Даний клас містить 3 поля:

- x — перший доданок;
- y - другий доданок;
- sum — їх сума;

Опишемо метод to-doc, який перетворює цей клас у текстове представлення (рис.81)

```
(defmethod to-doc ((data test-class))
  (format nil "Sum of ~A and ~A is ~A" (x data) (y data) (sum data)))
```

Рис.81 – Реалізація методу to-doc для об'єкту класу test-class

Створимо функцію, яка приймає 2 аргументи і повертає об'єкт класу, який був створений для тестування (рис.82).

```
(defun test-fn (x y)
  (make-instance 'test-class :x x :y y :sum (+ x y)))
```

Рис.82 – Інтерфейс тестової функції

Створимо декілька тестів (рис.83).

```
(define-doctest example-1 "~/tmp/test1.org"
  (test-fn 1 2))

(define-doctest example-2 "~/tmp/test1.org"
  (test-fn 3 4))
```

Рис.83 – Оголошення тестів

Запустимо функцію `run-doctests`, передаючи в якості параметру `run-meld-for` список, який складається з одного елемента — шляху `~/tmp/test1.org` і отримаємо результат, який показано на рис.84.

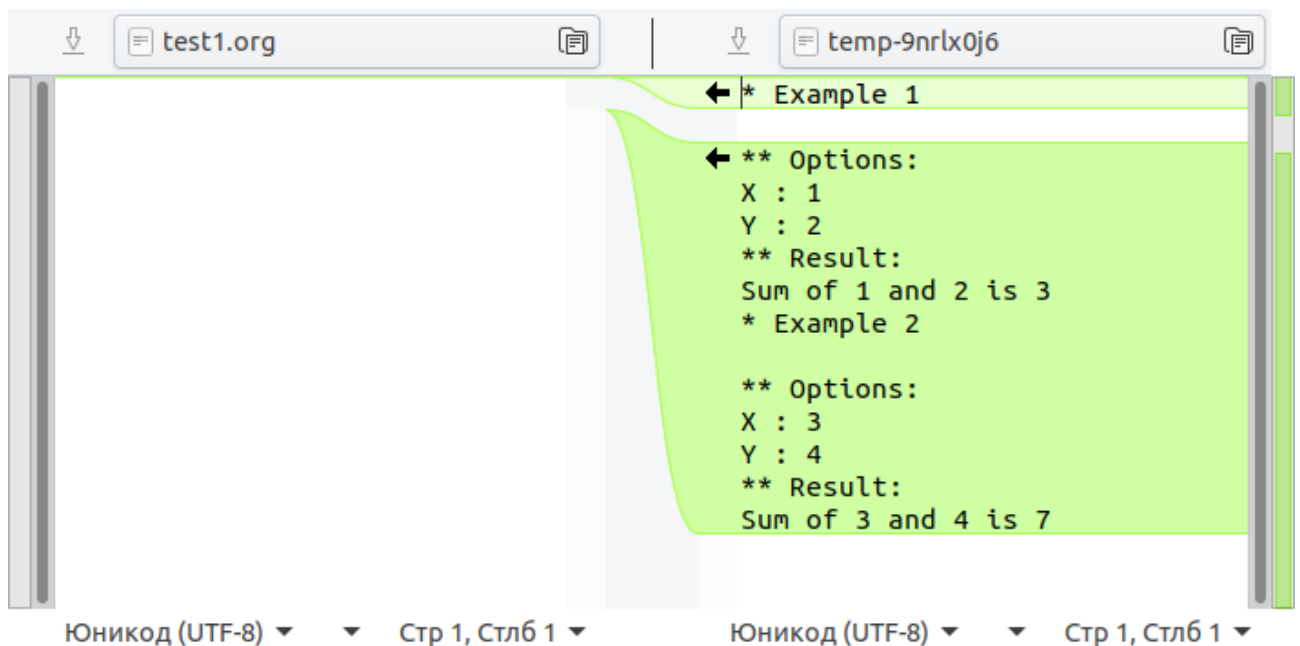


Рис.84 – Результат роботи

Додамо згенеровані тести у документацію і збережемо зміни.

Перепишемо функцію `test-fn` (рис.85), так щоб вона приймала не числа, а спеціальні класи `test-arg`, які зберігають значення аргументів (рис.86).

```
(defun test-fn (x y)
  (let ((val-x (value x))
        (val-y (value y)))
    (make-instance 'test-class
                   :x val-x
                   :y val-y
                   :sum (+ val-x val-y))))
```

Рис.85 – Інтерфейс тестової функції

```
(defclass test-arg ()
  ((value :initarg :value :reader value)))
```

Рис.86 – Клас test-arg

Опишемо метод to-doc для об'єктів даного класу (рис.87).

```
(defmethod to-doc ((data test-arg))
  (format nil "~A" (value data)))
```

Рис.87 – Реалізація методу to-doc для об'єктів класу test-arg

Також перепишемо оголошення тестів для документації (рис.88).

```
(define-doctest example-1 "~/tmp/test1.org"
  (test-fn (make-instance 'test-arg :value 1)
           (make-instance 'test-arg :value 2)))

(define-doctest example-2 "~/tmp/test1.org"
  (test-fn (make-instance 'test-arg :value 3)
           (make-instance 'test-arg :value 4)))
```

Рис.88 – Оголошення тестів

Зм	Лист	№ докум.	Підп.	Дата

Запустимо функцію `run-doctests`, з параметром `print-passed`, який встановлений у значення `t`. Як видно з отриманого результату (рис.89), нові тести аналогічні тим, що записані в документації, що свідчить про коректну роботу системи.

```
Tests:
..
Passed:
  DOCGEN/TEST::EXAMPLE-1
  DOCGEN/TEST::EXAMPLE-2
Passed: 2
Failed: 0
Disabled: 0
```

Рис.89 – Результат роботи

На останок створимо функцію (рис.90), яка повертає хеш-табличку, яка в якості ключів містить клас типу `value` (рис.91), який містить число, а в якості значення — також клас типу `value`, який містить дане число піднесене в квадрат.

```
(defun square (&rest values)
  (let ((hmap (make-hash-table :test #'eq)))
    (mapc (lambda (obj)
            (let ((number (val obj)))
              (assert (numberp number) nil "Number is expected.")
              (setf (gethash obj hmap)
                    (make-instance 'value
                                   :val (* number number)))))
          values)
    hmap))
```

Рис.90 – Інтерфейс тестовою функції

```
(defclass value ()
  ((val :initarg :numbers :reader val)))
```

Рис.91 – Клас value

Також реалізуємо метод to-doc для об'єктів даного класу(рис.92) і тести для тестування функції square(рис.93). Після цього запусимо функцію run-doctests і отримаємо результат, зображений на рис.94.

```
(defmethod to-doc ((data value))  
  (format nil "~A" (val data)))
```

Рис.92 – Реалізація методу to-doc для об'єктів класу value

```
(define-doctest example-1 "~/tmp/test1.org"  
  (square (make-instance 'value :val 1)  
          (make-instance 'value :val 2)))  
  
(define-doctest example-2 "~/tmp/test1.org"  
  (square (make-instance 'value :val 3)  
          (make-instance 'value :val 4)))
```

Рис.93 – Оголошення тестів

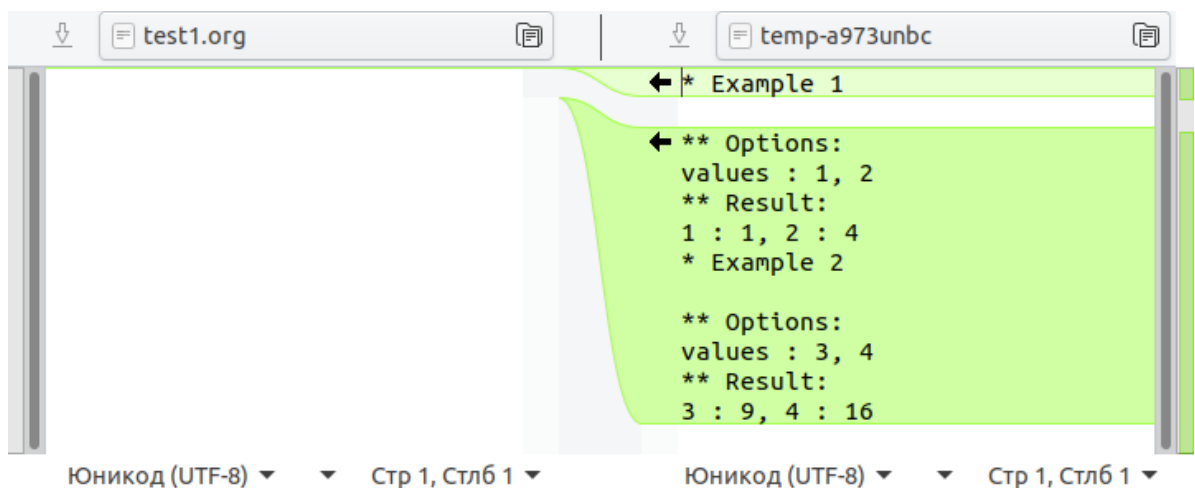


Рис.94 – Результат роботи

4.2 Висновки

В даному розділі дипломного проєкту було протестовано роботу розробленої системи генерації і оновлення документації опису тестів мовою Common Lisp. На прикладах було показано принцип створення тестів – макрос `define-doctest`, протестований механізм ігнорування тестів – макрос `disable-doctest`, а також їх запуску з інтерпретатора – функція `run-doctests` або прямий виклик функції з іменем тесту. Також було показано роботу методу `to-doc`, який повертає рядкове представлення вхідного аргументу як для стандартних типів мови Common Lisp, так і для класів і структур, що створені користувачем. Всі приклади супроводжувались рисунками, на яких зображений результат роботи, а також поведінка генератора при різних вхідних параметрах.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		68

ВИСНОВКИ

Система генерації і оновлення документації опису тестів мовою Common Lisp, яка є результатом дипломного проєкту, була створена для прискорення процесу розробки та зведення написання документації, яка описує поведінку програми при різних вхідних параметрах, до створення тестів у форматі юніт-тестів, за допомогою ,розроблених в даному дипломному проєкті, інструментів.

Після аналізу існуючих генераторів документації, який був проведений в першому розділі дипломного проєкту, не знайдено інструментів, які б мали можливість інтегрувати та/або оновлювати документацію опису тестів, і саме тому, метою створення даного дипломного проєкту – є розробка системи, яка мала б ширші функціональні можливості, які включають в себе засоби для швидкого створення чи оновлення тестів.

Головними перевагами створеного програмного продукту є:

- можливість запуску тестів через інтерпретатор;
- можливість порівняння різних версій файлів документації через графічний інтерфейс;
- механізм ігнорування тестів;
- зберігання документації у зручному форматі – org-mode, який інтегрований в середовище розробки Emacs;
- механізм перетворення у текст, який буде генеруватися у документацію, як для стандартних типів, так і для користувацьких.

В майбутньому в розроблену систему може бути додана підтримка інших мов програмування, а також можливість вибору формату файлів для зберігання документації.

					ІАЛЦ.045490.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		69

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Technical Documentation in Software Development: Types, Best Practices, and Tools. URL: <https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/> (дата звернення 01.05.2020).
2. Касянчук Д.П. Порівняльний аналіз генераторів документації. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2020. № 39. С. 30 – 34. URL: <http://cit-journal.com.ua/index.php/cit/article/view/118> (дата звернення: 03.05.2020).
3. Lightweight markup language. URL: https://en.wikipedia.org/wiki/Lightweight_markup_language (дата звернення 11.05.2020).
4. Markdown. URL: <http://www.aaronsw.com/weblog/001189> (дата звернення 11.05.2020).
5. The Org Manual. URL: <https://orgmode.org/org.html> (дата звернення 11.05.2020).
6. Meld. URL: <https://meldmerge.org> (дата звернення 12.05.2020).
7. DiffUse. URL: <https://www.linuxlinks.com/diffuse/> (дата звернення 12.05.2020).
8. Преимущества Common Lisp. URL: <https://habr.com/ru/post/143490/> (дата звернення 13.05.2020).
9. CL-Yacc – a LALR(1) parser generator for Common Lisp. URL: <https://www.irif.fr/~jch/software/cl-yacc/> (дата звернення 14.05.2020).
10. Cl-Org-Mode. URL: <https://github.com/deepfire/cl-org-mode> (дата звернення 14.05.2020).
11. Bordeaux-Threads. URL: <https://trac.common-lisp.net/bordeaux-threads/wiki/ApiDocumentation> (дата звернення 14.05.2020).
12. Lisp in parallel: lparallel. URL: <https://lparallel.org> (дата звернення 15.05.2020).