

**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: Вебдодаток демо-компілятора для підтримки навчального процесу з
дисципліни «Основи проєктування трансляторів»

Виконала:

студентка IV курсу, групи КВ-61

Курдус Анастасія Олександрівна

_____ (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 _____

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

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

Студент _____
(підпис)

Київ – 2020 року

- Алгоритм роботи сервера вебдодатку;
- Презентація за темою роботи.

6. Консультанти розділів проєкту

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

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	15.12.2019	
2.	Розроблення та узгодження технічного завдання	18.12.2019	
3.	Аналіз існуючих рішень	15.04.2020	
4.	Підготовка матеріалів першого розділу дипломного проєкту	8.05.2020	
5.	Підготовка матеріалів другого розділу дипломного проєкту	10.05.2020	
6.	Підготовка матеріалів третього розділу дипломного проєкту	11.05.2020	
7.	Підготовка матеріалів четвертого розділу дипломного проєкту	13.05.2020	
8.	Підготовка графічної частини дипломного проєкту	14.05.2020	
9.	Оформлення документації дипломного проєкту	17.05.2020	
10.	Попередній огляд матеріалів диплому	20.05.2020	

Студент

(підпис)

Анастасія КУРДУС

Керівник проєкту

(підпис)

Олександр МАРЧЕНКО

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (69 с., 70 рис., 1 табл., 4 додатки).

Об'єкт розробки – створення вебдодатку демо-компілятора, який допоможе краще зрозуміти та пізнати предмет «Основи проєктування транслятора».

Розроблений вебдодаток дозволяє:

- наочно побачити результат роботи транслятора на етапі scanner;
- наочно побачити результат роботи транслятора на етапі parser;
- наочно побачити результат роботи транслятора на етапі generator;
- вказати на лексичні або синтаксичні помилки в коді, якщо такі існують.

В процесі розробки була використана мова програмування JavaScript з використанням React та Redux, а також Common Lisp.

В ході виконання дипломного проєкту:

- розроблено архітектуру сервера;
- розроблено архітектуру клієнтського коду;
- проведено аналіз існуючих рішень;
- обрано інструменти для реалізації вебдодатку.

Використання цього вебдодатку дозволить студентам краще розібратися з основними етапами трансляції.

Ключові слова: ВЕБДОДАТОК, ТРАНСЛЯТОР, ЕТАПИ ТРАНСЛЯЦІЇ, JAVASCRIPT, REACT, REDUX, COMMON LISP, АРХІТЕКТУРА.

ABSTRACT

Qualifying work includes an explanatory note (6 9p., 70 fig., 1 tables, 4 applications).

The object of development is to create a demo-compiler web application that will help to better understand and learn the subject "Basics of translator design".

The developed web application allows:

- clearly see the result of the translator at the stage of scanner;
- clearly see the result of the translator at the parser stage;
- clearly see the result of the translator at the generator stage;
- indicate lexical or syntactic errors in the code, if any;

The development process used the JavaScript programming language using React and Redux, as well as Common Lisp.

During the implementation of the diploma project:

- developed server architecture;
- developed client code architecture;
- the analysis of existing decisions is carried out;
- selected tools for implementing the web application.

Using this web application will allow students to better understand the basic stages of translation.

Keywords: WEB ADDITION, TRANSLATOR, STAGES OF TRANSLATION, JAVASCRIPT, REACT, REDUX, COMMON LISP, ARCHITECTURE.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.045440.002 ТЗ	Вебдодаток демо-компілятора для підтримки навчального процесу з дисципліни «Основи прекування трансляторів» Технічне завдання	4		
	A4	ІАЛЦ.045440.003 ТП	Вебдодаток демо-компілятора для підтримки навчального процесу з дисципліни «Основи прекування трансляторів» Відомість технічного проекту	2		
	A4	ІАЛЦ.045440.004 ПЗ	Вебдодаток демо-компілятора для підтримки навчального процесу з дисципліни «Основи прекування трансляторів» Пояснювальна записка	69		
ІАЛЦ.045440.001 ОА						
Змін.	Арк.	№ докум.	Підпис	Дата		
Розробив	Курдус.А.О.				Літ.	Аркуш
Перевірив	Марченко О.І.					Аркушів
Консульт.						1
Н. контроль	Клятченко Я.М.					2
Зав. каф.	Романкевич В.О.				КПІ ім. Ігоря Сікорського, ФПМ КВ-61	
					Вебдодаток демо-компілятора для підтримки навчального процесу з дисципліни «Основи прекування трансляторів» Опис альбому	

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.	Рекомендовані вимоги до апаратного забезпечення _____	3
5.3.	Вимоги до мінімального програмного забезпечення _____	3
6.	Етапи розробки _____	4

						ІАЛЦ.045440.002 ТЗ				
Зм.	Арк.	№ докум.	Підп.	Дата	Вебдодаток демо-компілятора для підтримки навчального процесу з предмету "Основи проектування трансляторів". Технічне завдання			Літ.	Аркуш	Аркушів
Розроб.	Курдус А.О.								1	4
Перевір.	Марченко О.І									
Н. контр.	Клятченко Я.М.							НТУУ "КПІ ім.І.Сікорського" ФПМ		
Затв.	Романкевич В.О							КВ-61		

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування роботи – «Вебдодаток demo-компілятора для підтримки навчального процесу з предмету “Основи проектування трансляторів”».

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 ім.І.Сікорського».

3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є розробка вебдодатку demo-компілятора, який допоможе більш детально розібратися у кожному з етапів трансляції.

4. ДЖЕРЕЛА РОБОТИ

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

					ІАЛЦ.045440.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підп.	Дата		2

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до системи, що розробляється

Вебдодаток повинен забезпечувати наступні функції:

- можливість додання нових мов для трансляції;
- підтримка етапу трансляції scanner;
- підтримка етапу трансляції parser;
- підтримка етапу трансляції generator;
- візуалізація помилок, якщо вони існують;
- можливість переходу на потрібний етап трансляції.

5.2. Рекомендовані вимоги до апаратного забезпечення

- Процесор: Intel Core i5.
- Оперативна пам'ять: 4 Гб.
- Простір на диску: 2 Гб.

5.3. Вимоги до мінімального програмного забезпечення

- Операційна система Windows, Linux чи Mac OS X.
- Встановлений прт.

					ІАЛЦ.045440.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підп.	Дата		3

6. Етапи розробки

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.12.2019
2.	Розроблення та узгодження технічного завдання	18.12.2019
3.	Аналіз існуючих рішень	15.04.2020
4.	Підготовка матеріалів першого розділу дипломного проекту	8.05.2020
5.	Підготовка матеріалів другого розділу дипломного проекту	10.05.2020
6.	Підготовка матеріалів третього розділу дипломного проекту	11.05.2020
7.	Підготовка матеріалів четвертого розділу дипломного проекту	13.05.2020
8.	Підготовка графічної частини дипломного проекту	14.05.2020
9.	Оформлення документації дипломного проекту	17.05.2020
10.	Попередній огляд матеріалів диплому на кафедрі	20.05.2020

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4	ІАЛЦ.045440.000	Завдання на дипломний проєкт	2	
2	A4	ІАЛЦ.045440.001 ОА	Опис альбому	2	
3	A4	ІАЛЦ.045440.002 ТЗ	Технічне завдання	4	
4	A4	ІАЛЦ.045440.003 ТП	Відомість технічного проєкту	1	
5	A4	ІАЛЦ.045440.004 ПЗ	Пояснювальна записка	69	
6	A4	ІАЛЦ.045440.005 Д1	Структурна схема серверу вебдодатку.	1	
7	A4	ІАЛЦ.045440.006 Д2	Структурна схема клієнтського коду проєкту.	1	
8	A4	ІАЛЦ. 045440.007 ДЗ	Алгоритм взаємодії клієнта з інтерфейсом.	1	
9	A4	ІАЛЦ. 045440.008 Д4	Алгоритм роботи сервера вебдодатку.	1	

				ІАЛЦ.045440.003 ТП		
	ПІБ	Підп.	Дата			
Розробн.	Курдус А.О.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Марченко О.І.				1	1
Консульт.					КП 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: Вебдодаток демо-компілятора для підтримки навчального процесу з дисципліни «Основи проєктування трансляторів»

ЗМІСТ

ВСТУП	3
1. АНАЛІЗ ІСНУЮЧИХ ОНЛАЙН-КОМПІЛЯТОРІВ	4
1.1. Визначення компілятора, види та структура	4
1.2 Аналіз існуючих онлайн-компіляторів	5
1.3 Обґрунтування теми дипломного проєкту	10
Висновок до 1-го розділу.....	10
2. ІНСТРУМЕНТИ ДЛЯ РЕАЛІЗАЦІЇ ПРОЄКТУ	11
2.1 Інструменти для реалізації клієнта.....	11
2.2 Інструменти для реалізації сервера вебдодатка	13
Висновок до 2-го розділу.....	14
3. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ ОНЛАЙН-КОМПІЛЯТОРА.....	15
3.1 Етапи реалізації сервера вебдодатку	15
3.2 Архітектура клієнтського коду	17
3.3 Опис інтерфейсу вебдодатку	44
4. ТЕСТУВАННЯ ВЕБДОДАТКУ	51
Висновок до 4-го розділу.....	66
ВИСНОВОК.....	67

						ІАЛЦ.045440.004 ПЗ				
Зм.	Арк.	№ докум.	Підп.	Дата	Вебдодаток демо-компілятора для підтримки навчального процесу з предмету «Основи проєктування трансляторів». Пояснювальна записка.					
Розроб.	Курдус А.О.							Літ.	Аркуш	Аркушів
Перевір.	Марченко О.І.							1	69	
Н. контр.	Клятенко Я.М.							НТУУ "КПІ" ФПМ		
Затв.	Романкевич В.О.							КВ-61		

ДОДАТКИ

Додаток 1. Копії графічного матеріалу.

ІАЛЦ.045440.005 Д1. Структурна схема серверу вебдодатку.

ІАЛЦ.045440.006 Д2. Структурна схема клієнтського коду вебдодатку.

ІАЛЦ.045440.007 Д3. Алгоритм взаємодії клієнта з інтерфейсом.

ІАЛЦ.045440.008 Д4. Алгоритм роботи сервера вебдодатку.

Додаток 2. Фрагменти програмного коду.

Додаток 3. Публікація за темою роботи.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		2

ВСТУП

Останнім часом онлайн-компілятори набули великої популярності і це не дивно, адже тепер, щоб перевірити і виконати свою програму немає необхідності встановлювати на комп'ютер додаткове програмне забезпечення. Потрібно тільки зайти на сайт, написати свій код та скомпілювати його. Як показує статистика, саме школярі та студенти найчастіше використовують онлайн-компілятори. Такі показники зрозумілі, адже подібний вебдодаток надзвичайно корисний для людей, які тільки починають свій шлях в ІТ. Онлайн-компілятори дають можливість викладачу більш наочно продемонструвати правильність рішення задачі, а учню або студенту краще розібратися в демонстраційному матеріалі. Проте, слід зазначити, що і в комерційній розробці, серед професійних програмістів доволі часто зустрічаються користувачі онлайн-компіляторами. Для розробників подібний вебдодаток може бути корисним в ситуаціях, коли під рукою немає комп'ютера, а є тільки смартфон і потрібно швидко перевірити роботу коду. Звичайно є і певні недоліки, такі як: необхідність наявності підключення до Інтернету та більш довге очікування результатів компіляції, але для навчальних цілей дані проблеми не є критичними.

Завданням дипломного проєкту є розробка покрокового онлайн-компілятора для покращення навчального процесу з предмету «Основи проєктування трансляторів». Компілятор з реалізацією подібного функціоналу допоможе більш глибоко і детально розібратися у всіх етапах проєктування трансляторів.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		3

1. АНАЛІЗ ІСНУЮЧИХ ОНЛАЙН-КОМПІЛЯТОРІВ

1.1. Визначення компілятора, види та структура

Транслятор – це програма, що перетворює програмний текст коду, який написаний на одній мові, в текст, який написаний на іншій.

Компілятор – це транслятор, який виконує перетворення з мови високого рівня в машинну мову.

Структура компілятора поділяється на[1]:

- Лексичний аналіз (рис.1) – виконується заміна послідовності символів вхідної мови на послідовність лексем та виявлення лексичних помилок. До лексем можна віднести константи, ключові слова, ідентифікатори, одно символні та багато символні роздільники.



Рис.1 - Принцип роботи лексичного аналізатора

- Синтаксичний аналіз – процес зіставлення послідовності лексем з граматикою вхідної мови і виявлення синтаксичних помилок.

- Генератор коду – процес заміни вхідної програми в текст вихідної.

Загальний вигляд структури транслятора представлено на рис.2.



Рис.2 - Структура транслятора

1.2 Аналіз існуючих онлайн-компіляторів

Аналізуватися будуть наступні онлайн-компілятори: CodePad.io, C++shell, IdeOne, CodingGround, JDoodle, TryCode[2].

CodePad.io — цей онлайн-компілятор (рис.3) підтримує велику кількість мов програмування. Інтерфейс онлайн-компілятора досить простий і

зрозумілий у використанні. Головна перевага цього сервісу в тому, що він дуже легкий, тобто сайт буде швидко працювати і при повільному інтернеті. Серед наявних функцій, CodePad дає можливість ділитися своїм кодом з колегами електронною поштою або в чаті. Серед мінусів можна назвати відсутність підсвічування коду, що вводиться, і це може створити певний дискомфорт для розробника[2].

Рис.3 - Онлайн-компілятор CodePad

C++shell — онлайн-компілятор, що підтримує лише мови C та C++. Проте така зосередженість на певних мовах програмування дозволяє реалізувати всі можливості оригінального C та C++. Щодо інтерфейсу онлайн-компілятора, то він трохи застарілий[2].

IdeOne — онлайн-компілятор (рис.4), який підтримує 60 мов програмування. Інтерфейс дуже легкий в освоєнні та інтуїтивно зрозумілий. Ще однією перевагою даного онлайн-компілятора є можливість ділитися своїм кодом з колегами. Та найбільшою особливістю IdeOne є те, що всі члени групи, яка працює з кодом, можуть зробити так званий fork файлу[2].

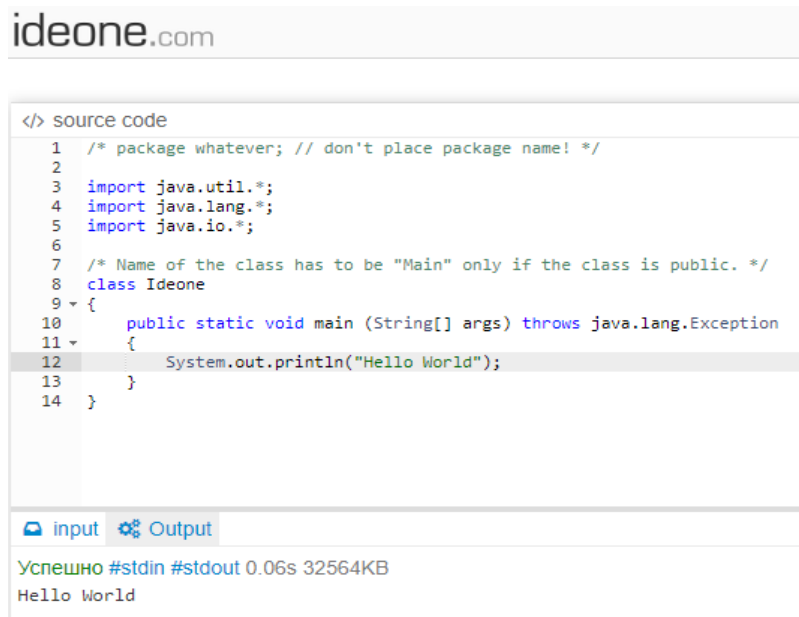


Рис.4 - Онлайн-компілятор IdeOne

JDoodle — це онлайн-компілятор (рис.5), який підтримує 72 мови програмування. Досить корисною функцією онлайн-компілятора є можливість спільної роботи програмістів. Також є підсвічування коду. Проте даний онлайн-компілятор має і недолік, а саме: підтримку всього одного файлу[2].



Рис.5 - Онлайн-компілятор JDoodle

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		7

CodingGround — це онлайн-компілятор (рис.6), який підтримує понад 75 мов програмування. Інтерфейс онлайн-компілятора приємний у користуванні та досить мінімалістичний. Також представлений функціонал для онлайн програмування та можливість ділитися кодом[2].

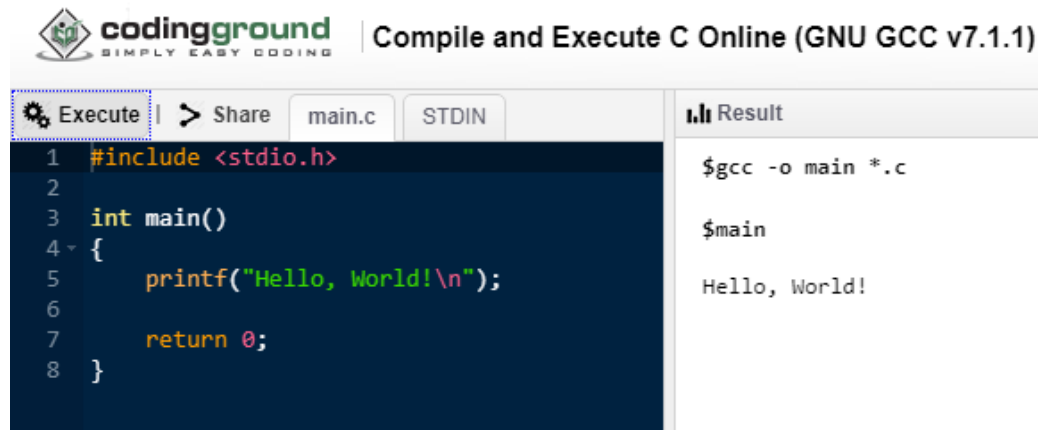


Рис.6 - Онлайн-компілятор CodingGround

TryCode — онлайн-компілятор (рис.7), який підійде для спільної роботи розробників. Вбудована підтримка розробки в режимі реального часу. Проте функціонал і налаштування дещо обмежені. Що стосується інтерфейсу, то він нативний. TryCode є онлайн-компілятором лише для JavaScript[2].

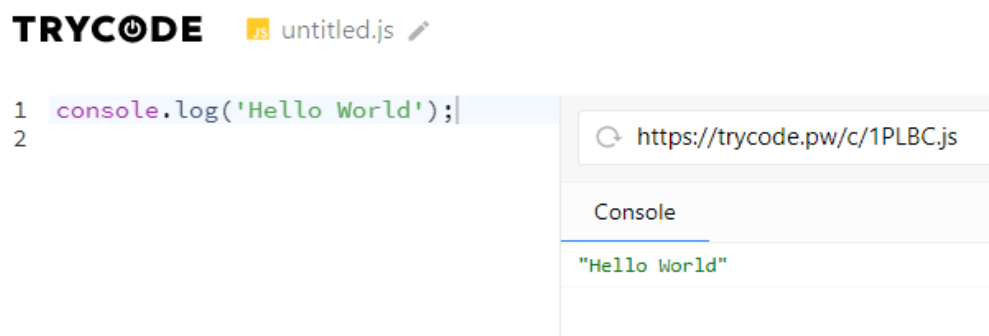


Рис.7 - Онлайн-компілятор TryCode

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		8

Загалом, якщо структурувати все вище сказане, то отримаємо Таблицю 1. В ній зазначені певні критерії за якими порівнюються онлайн-компілятори, а саме: інтерфейс та різноманітність функціоналу, які суб'єктивно оцінюються за п'ятибальною шкалою, спираючись на опис кожного онлайн-компілятора приведеного вище, а також кількість мов програмування, що підтримуються онлайн-компілятором, яка була взята з джерел опису представлених онлайн-компіляторів. Одним з пунктів оцінки онлайн-компіляторів є швидкість їхньої роботи, яка була отримана експериментальним шляхом, методом вимірювання, за допомогою спеціальних функцій, часу компіляції однієї і тієї ж програми. Такий пункт, як популярність серед користувачів – це пропорційно виведена оцінка, яка базується на статистичних даних з різних джерел інформації. Останнім пунктом таблиці є середній бал, він показує середнє значення по всім показникам для кожного онлайн-компілятора, причому всі кількісні показники (швидкість роботи та к-ть мов програмування) були пропорційно оцінені за п'ятибальною шкалою[2].

Таблиця 1. Суб'єктивна оцінка онлайн-компіляторів за певними критеріями[2]

Критерії \ Компілятори	CodePad	C++shell	IdeOne	CodingGround	JDoodle	TryCode
Інтерфейс	3	3	5	4	3	4
К-ть мов, що підтримуються	15	2	60	75+	72	2
Різноманітність функціоналу	3	4	5	5	4	3
Швидкість роботи	37	18	13	31	20	7
Популярність	3	2	4	5	4	3
<i>Середній бал</i>	3	3	4.6	4.2	4	3.2

1.3 Обґрунтування теми дипломного проекту

Як видно, серед представлених онлайн-компіляторів немає такого, який міг би покроково ілюструвати внутрішню роботу транслятора. Така функціональність є корисною для вивчення предмету «Основи проектування трансляторів». Подібний онлайн-компілятор дозволить побачити як проходять всі етапи компілювання, які помилки можуть бути виявленими на кожному з цих етапів, а також допоможе студенту більш ґрунтовно зрозуміти структуру транслятора.

Висновок до 1-го розділу

Якщо розглядати онлайн-компілятори по кількості мов, які підтримуються, то беззаперечним лідером є CodingGround, адже кількість мов більше ніж 75. Проте онлайн-компілятор JDoodle, який підтримує 72 мови програмування також можна назвати лідером, адже кількість мов велика і майже кожен користувач може знайти потрібну йому мову. Також до трійки лідерів можна віднести і IdeOne, який підтримує 60 мов програмування. На одному рівні знаходяться такі онлайн-компілятори, як: CodePad.io, C++shell, TryCode. В кожного є свої переваги і недоліки, але ці онлайн-компілятори складно назвати універсальними. Всі описані компілятори є непоганими, але жоден з них не має можливості демонстрації покрокової трансляції вхідного коду. Саме реалізація такого функціоналу і є метою створення даного вебдодатку.

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		10

2. ІНСТРУМЕНТИ ДЛЯ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1 Інструменти для реалізації клієнта

Основним інструментом для реалізації клієнтського коду є мова програмування JavaScript. JavaScript – це динамічна мова програмування, яка підтримує декларативний, імперативний та об’єктно-орієнтований стилі[3]. JavaScript має велику кількість бібліотек, які розширюють можливості мови. Для написання програмної частини дипломного проєкту використовувалась бібліотека React.

Головним завданням React є вирішення проблеми часткового оновлення вебсторінки. Потреба використання цієї бібліотеки виникає, коли вебдодаток має змінні дані і розробник хоче уникнути перезавантаження сторінки.

Основними перевагами React є[4]:

- простий в написанні та розумінні програмний код
- наявність Virtual DOM
- легко мігрувати між версіями
- відкритий вихідний програмний код

Redux - ще одна бібліотека мови програмування JavaScript, яка була використана для написання програмної частини проєкту. Її завдання це управління станом програми. Стан програми – це об’єкт, який зберігає інформацію про всі елементи компоненти. Найчастіше Redux використовують разом з React, незважаючи на те, що React має власний метод управління станом програми. Це пояснюється поганою масштабованістю останнього. Використання Redux сприяє підвищенню продуктивності роботи вебдодатку.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		11

Redux може бути описаний трьома головними принципами: store, action, reducer[5].

Store – це єдине джерело правди для додатку, тільки в ньому зберігається вся правильна та актуальна інформація.

Action – це об'єкт, який описує суть зміни state. Головною вимогою до action є наявність поля type, яке являє собою рядок.

Reducer – це чиста функція, яка обчислює новий state на основі старого state та беручи до уваги action. Тобто, головним завданням reducer є визначення того, як буде виглядати новий state, після виклику action.

Використання React та Redux бібліотек дозволяє в повній мірі реалізувати архітектуру Model-view-controller. Model-view-controller – це спосіб організації коду, який передбачає ділення коду на блоки, кожен з яких відповідає за рішення певної задачі. Поділ формує блоки, перший з яких відповідає за дані додатку, другий – за зовнішній вигляд, а третій контролює роботу додатку.

До переваг такого підходу слід віднести:

- при змінах в роботі з даними, інтерфейс не зміниться
- існування можливості повторного використання компонентів
- забезпечення гнучкого дизайну додатку

Для створення інтерфейсу вебдодатку використовувався Material-UI. Material-UI – це певний набір компонентів React, котрий реалізує Google Material Design. Ці компоненти є самопідтримуючими, а це говорить про те, що вони можуть працювати ізольовано.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		12

2.2 Інструменти для реалізації сервера вебдодатка

Для написання програмного коду сервера вебдодатка було використано мову програмування Lisp, її бібліотеки hunchentoot та sb-thread.

Мова програмування Lisp – це така мова програмування, яка підтримує фундаментальні парадигми процедурного та функціонального програмування[6]. Вибір мови програмування Lisp для написання транслятора обумовлений тим, що Lisp чудово підходить для реалізації розбору методом рекурсивного спуску, також при написанні транслятора на Lisp, всі базові складові дерева розбору, послідовності складових, а також ієрархічна підпорядкованість між коренем, гілками і листками дерева може бути представлена елементарними структурами Lisp, що спрощує саме дерево і роботу з ним, адже можна працювати із деревом розбору за допомогою стандартних функцій.

Hunchentoot – це інструмент для створення динамічних вебсайтів та вебдодатків. Ця бібліотека надає такі засоби, як: автоматична обробка сеансів, ведення журналів, робота з помилками, яку можна налаштувати та доступ до параметрів GET та POST, які надсилає клієнт. Бібліотека може і не використовувати багатопроцесорну обробку для роботи з декількома запитами одночасно[7].

Sb-thread – це бібліотека для керування потоками даних. Потік – це метод за допомогою якого програма може ділитися на n різних паралельних задач. Потік використовується для реалізації сервера вебдодатка з метою прискорення його роботи. Суть полягає у тому, що сервер використовує множину потоків, які виконуються одночасно, кожен з яких оброблює одну поточну задачу. Для коректної роботи сервера робота потоків повинна бути синхронізована. Синхронізація не допускає моментів, коли різні потоки працюють з однією пам'яттю одночасно. Sb-thread використовує інструмент синхронізації м'ютекс.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		13

Висновок до 2-го розділу

Підсумовуючи все описане в даному розділі, очевидно, що самі такі бібліотеки та мови програмування були необхідні для реалізації дипломного проєкту. Для створення нативної та дружньої до користувача клієнтської сторони, чудовим рішенням було обрати мову програмування JavaScript та її бібліотеки React і Redux для організації коду за принципом MVC. Завдяки використанню Material-UI стало можливим створити сучасний та простий інтерфейс, який є інтуїтивно зрозумілим. Аналогічним чином вдало підібрані інструменти реалізації серверної частини проєкту. Мова програмування Lisp, як зазначалося раніше, чудово підходить для реалізації розбору методом рекурсивного спуску. Бібліотеки Sb-thread та Hunchentoot допомогли зробити сервер швидким і тому відповідь на клієнтський запит не змушує себе чекати.

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		14

3. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ ОНЛАЙН-КОМПІЛЯТОРА

3.1 Етапи реалізації сервера вебдодатку

Так, як сервер вебдодатку, який є результатом дипломного проекту, являє собою транслятор, то його реалізація має такі ж етапи, як і реалізація транслятора.

Перший етап – це написання лексичного аналізатора. Лексичний аналізатор – це програма, яка здійснює перетворення вхідного рядка символів, що являє собою текст програми, в рядок лексем. Також ця програма повинна виявляти лексичні помилки.

До основних функцій лексичного аналізатора можна віднести:

- знаходження лексичних помилок
- створення таблиць з інформацією про символні, числові, рядкові константи та ідентифікатори
- виділення коментарів
- знаходження та згортання лексем

Принцип роботи лексичного аналізатора полягає в наступному: на вхід поступає символ вхідного тексту, він аналізується і, за результатами перевірок, визначається тип поточної лексеми. Як тільки з'являється символ, який поточна лексема не може містити, то це означає кінець цієї лексеми та початок нової. Сформована лексема додається до таблиць. Схематично роботу лексичного аналізатора можна побачити на рис.8.

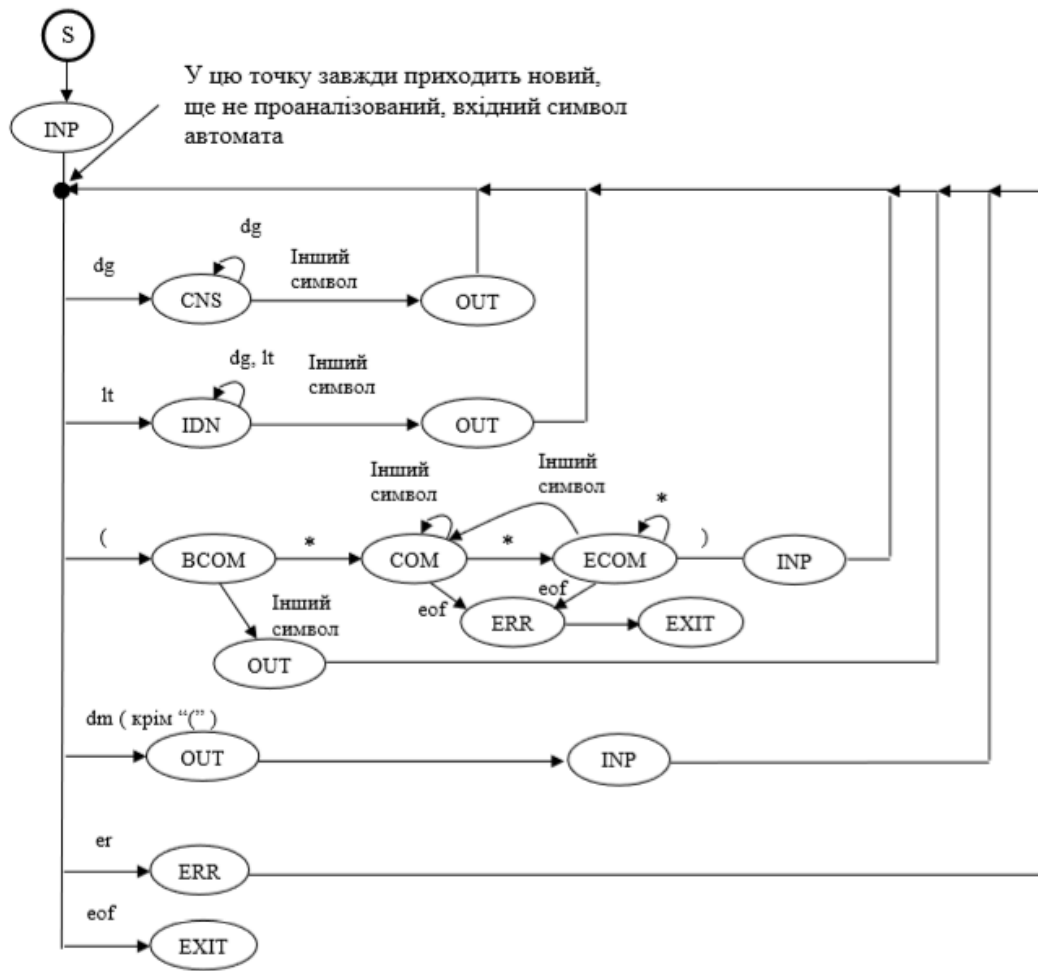


Рис.8 - Схематичне зображення принципу роботи лексичного аналізатора

Як зазначалося вище, однією з функцій лексичного аналізатора є виділення коментарів. Проте, в реалізації серверу вебдодатку, коментарі будуть видалятися, адже для вирішення задачі дипломного проекту вони не несуть ніякої цінності.

Виявлення лексичних помилок – це ще одна важлива функція лексичного аналізатора.

До такого типу помилок відносять:

- недопустимий символ
- неправильна лексема

- неочікуване закінчення файлу

Другим етапом реалізації сервера вебдодатку є синтаксичний аналізатор. Синтаксичний аналізатор – це програма, яка здійснює перетворення вхідного рядка лексем в структурні одиниці мови, спираючись на граматику вхідної мови. Також завдання синтаксичного аналізатора це виявлення синтаксичних помилок.

Існує три типи реалізації синтаксичного аналізатора:

- висхідний аналіз
- низхідний аналіз
- змішана стратегія

Для реалізації серверу вебдодатку використовувалась низхідна стратегія синтаксичного аналізу. Існує три алгоритми реалізації низхідної стратегії, а саме: алгоритм машини Кнута, алгоритм методу рекурсивного спуску, алгоритм таблично-керованого передбачаючого розбору.

При написанні серверу вебдодатку було використано метод рекурсивного спуску, адже, як зазначалося раніше, саме цей алгоритм можна просто та ефективно реалізувати на мові програмування Lisp.

Третій етап – це генерація коду. Генератор коду – це програма, яка, спираючись на семантику вхідної мови, перетворює вхідну програму на текст вихідної мови.

3.2 Архітектура клієнтського коду

Реалізація клієнтського коду міститься в двох основних папках: public та src. Public містить один єдиний файл – це index.html. Саме за допомогою цього файлу і відображається у браузері головна сторінка вебдодатку. Тобто весь текст та зображення, які бачить користувач потрапивши на створену сторінку

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		17

або вебдодаток. В папці src знаходяться ті файли, які відповідають за логічну частину інтерфейсу та роблять можливою взаємодію компонентів один з одним. Ця папка містить такі папки, як:

- components
- modules
- redux
- utils.

Папка utils має в собі файли, розширення яких js, з функціями, які є загальними для декількох компонентів, таким чином досягається мінімізація однакового коду в програмі. Зменшення тотожного коду в проєкті не зробить його швидшим, але це є гарним правилом написання коду. В даному проєкті ця папка містить файл notify.js. Задачею цього файлу є визначати успішним чи помилковим є запит, який виконався. Від результату, що поверне ця функція залежить зовнішній вигляд віконця-повідомлення. Якщо запит виконався успішно, то віконце буде зелене та матиме підпис success, а якщо помилковим, то віконце буде червоне з написом warning. Таким чином користувач зможе дізнатися чи завершився вже запит, чи ні, а також знатиме, як саме він завершився. Це є дуже корисним, адже користувач не має доступу до тих інструментів, якими може користуватися програміст аби зрозуміти чи завершився запит, а знати користувачеві це потрібно. Саме в такі моменти і допомагає вікно-повідомлення.

Папка components – це папка, що містить всі компоненти програми, які відображаються на екрані, а також компоненти, які відповідають за стилі вебдодатку.

Так, як:

- Editor.jsx
- Scanner.jsx

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		18

- Parser.jsx
- Generator.jsx
- Error.jsx
- ActiveButton.jsx
- App.js
- BorderGridStyled.jsx
- BordersGrid.jsx
- ButtonStyled.jsx
- ImportButton.jsx
- InformationAboutCurrentLexemStyled.jsx
- MainTabs.jsx
- Notification.jsx
- NotificationHelper.jsx
- ParserTree.jsx
- RadioButton.jsx
- RadioStyled.jsx
- ScannerTable.jsx
- Steps.jsx
- TableCellStyled.jsx
- TextFieldStyled.jsx

Всі ці файли мають розширення jsx, що говорить про те, що це саме компоненти React. Всі ці компоненти є функціональними, а не класовими. Будь-яку класову компоненту тепер можна робити функціональною, адже в React з'явилися Hooks. Також серед всіх перелічених файлів можна зустріти декілька, які в своїй назві мають слово Styled. Це означає, що ці файли є стильовими. Вони описують стиль відповідних компонент. Так наприклад, BorderGridStyled описує стиль BorderGrid, RadioStyled описує стиль RadioButton, TableCellStyled описує стиль вбудованої в material-ui компоненти

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		19

TableCell, а TextFieldStyled описує стиль TextField. Структура цих файлів має однаковий вигляд. Вона приведена на рис.9.

```
import Radio from '@material-ui/core/Radio'  
import { withStyles } from '@material-ui/core'  
  
export default withStyles({  
  root: {  
    marginLeft: '7px',  
    color: '#3f51b5',  
    '&$checked': {  
      color: '#3f51b5',  
    },  
  },  
  
  },  
  checked: {},  
})(Radio)
```

Рис.9 – Структура файлу зі стилями на прикладі RadioStyled.jsx

Як видно з рисунка, то головною функцією, яка робить можливою таке задання стилів є withStyles. Це вбудована функція material-ui. Результатом її роботи є об'єкт з полем classes. Потім цей об'єкт присвоюється параметру classNames того тегу чий стиль треба змінити. Приклад показано на рис.10.

```
<div className={props.classes.root}>
```

Рис.10 – Приклад використання об'єкту classes

Це дуже зручний метод перевизначення стилів, а також він не потребує ніяких нових вмій та знань. Тільки знання звичайного CSS. Але є одна особливість про яку забувати не можна, а саме те, що при використанні такого підходу всі стилі, які в своїй назві мають дефіс потрібно замінювати на

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		20

camelCase, тобто всі слова з яких складається назва стилю, крім першого, пишуться з великої літери. Наприклад, якщо назва стилю в CSS це padding-top, то з використанням цього підходу повинно бути paddingTop і так далі для всіх назв, що мають дефіс.

Повертаючись до основних компонентів треба зазначити, що не всі вони є окремим елементом, який обов'язково буде зображений у вікні браузера. Деякі з них є підкомпонентами глобальних компонент. Деякі компоненти складаються з n-кількості інших компонент. Це все робиться для того, щоб код цих компонент легко читався та не було ефекту спагеті-коду. Так компонента Notification.jsx містить в собі компоненту NotificationHelper.jsx. Іншими словами компонента Notification.jsx є батьком, а компонента NotificationHelper.jsx є дочірньою компонентою. Аналогічно відбувається з компонентами BorderGrid.jsx та MainTabs.jsx, RadioButton.jsx, Steps.jsx, ActiveButton.jsx, Editor.jsx, Scanner.jsx, Parser.jsx, Generator.jsx, Error.jsx. Тобто зрозуміло, що BorderGrid.jsx є компонентою-батьком, а всі інші є його дітьми. Цікавим є те, що всі компоненти, які є дітьми для якоїсь компоненти, одночасно можуть бути батьками для іншої. Наприклад, компонента Scanner.jsx, яка є дитиною для BorderGrid.jsx, є батьком для ScannerTable.jsx та Notification.jsx, Parser.jsx, яка також є дитиною для BorderGrid.jsx, є батьком для ParserTree.jsx, Generator.jsx також є батьком для Notification.jsx. З цього прикладу ми бачимо, що одна компонента може мати декілька батьків, а також і не одну дитину. Дивлячись на програмний код компоненти легко зрозуміти батьком яких компонент є ця компонента. Це показано в import на початку тексту компоненти. Приклад приведено на рис.11.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		21

```

import React from 'react'
import FilterDramaIcon from '@material-ui/icons/FilterDrama'
import MainTabs from './MainTabs'
import RadioButton from './RadioButton'
import Steps from './Steps'
import ActiveButton from './ActiveButton'
import Editor from './Editor'
import Scanner from './Scanner'
import Parser from './Parser'
import Generator from './Generator'
import {getErrorPresent} from '../modules/selectors/index'
import { connect } from 'react-redux'
import Error from './Error'

```

Рис.11 – Використання import на прикладі BorderGrid.jsx

Є одно неписане правило, яке говорить, що компонента App.jsx є головною компонентою, тобто батьком всіх компонент. В даному проєкті компонента App.jsx також є головною компонентою, яка імпортує компоненту BordersGrid.jsx, що і зображено на рис.12.

```

import React from 'react'
import BordersGrid from './BorderGridStyled'

const App = () => <BordersGrid/>
export default App

```

Рис.12 – Головна компонента App.jsx

Також для зрозумілого написання коду дуже важливо, щоб компоненти мали назви, які відповідають тому, що ця компонента виконує. Розберемо це більш детально на прикладах компонент даного вебдодатку. Будемо рухатись

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		22

від головної компоненти вниз по її дітях. Тож, як вже зазначалося вище головною є компонента App.jsx, а її дитиною є BordersGrid.jsx. Функцією компоненти BordersGrid.jsx є саме створення загальної оболонки та розмітки вікна браузера, так би мовити, виділяє кожному майбутньому елементу своє місце. Робить загальний вигляд вебдодатку. В даному проєкті для розмітки сторінки використовувалася технологія Grid про, що і говорить назва самої компоненти BordersGrid.jsx. Я вирішила обрати саме цю технологію, адже вона досить проста та зрозуміла. Компонента BordersGrid.jsx має серед дітей такі компоненти, як:

- MainTabs.jsx
- RadioButton.jsx
- Steps.jsx
- ActiveButton.jsx
- Editor.jsx
- Scanner.jsx
- Parser.jsx
- Generator.jsx
- Error.jsx

MainTabs.jsx це компонента, яка відповідає за меню, використовуючи, яке користувач може дивитися результати етапів трансляції (scanner, parser, generator, errors). Реалізовано це меню за допомогою вбудованих в material-ui компонент Tab і Tabs. На вхід цій компоненті поступає масив дітей, які являють собою цілі компоненти. MainTabs.jsx виступає в ролі обгортки над іншими компонентами. А компоненти, які в неї загорнуті автоматично є її дітьми, хоча і не зазначені в тексті компоненти, як за імпортовані. Приклад використання цієї компоненти-обгортки приведено на рис.13. Тому, беручи до уваги все вище сказане та рис.15, можна сказати, що компоненти Editor.jsx, Scanner.jsx, Parser.jsx, Generator.jsx, Error.jsx є дітьми компоненти MainTabs.jsx.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		23

```

<div className='tabs'>
  <MainTabs>
    <Editor tablabel='Editor' />
    <Scanner tablabel='Scanner' />
    <Parser tablabel='Parser' />
    <Generator tablabel='Generator' />
    {props.errorPresent ? <Error tablabel='Error' /> : <div tablabel='' />}
  </MainTabs>
</div>

```

Рис.13 – Приклад використання компоненти MainTabs.jsx

RadioButton.jsx – це компонента, яка реалізовує перемикачі, які користувач буде використовувати при виборі поточного етапу трансляції. Реалізована компонента з використанням, вбудованих в material-ui, функцій RadioGroup, FormControlLabel, FormControl та Radio. Особливістю цих кнопок є те, що неможливо обрати одночасно декілька варіантів. Тобто, якщо користувач обрав все один з етапів трансляції, то він вже не може обрати інший. Це застерігає користувача від випадкової помилки. Інакше, якби такої функціональності не було, а користувач, випадковим чином, обрав би одночасно 2 етапи, то програма не зрозуміє, який саме етап виконувати, і може видати неочікувані результати. Тому, саме через наявність такого функціоналу і були обрані ці кнопки.

Steps.jsx – ця компонента, являє собою поле текстового вводу. Воно потрібне для надання користувачеві можливості введення тієї кількості кроків на яку просунеться трансляція коду. Для реалізації цього поля використовувалася компонента TextField з бібліотеки material-ui. Цікавою особливістю цього поля вводу, яке реалізоване саме з використанням компоненти TextField, є те, що виконується запам'ятовування найчастіше введених значень. Це зручно тоді, коли користувач хоче просуватися завжди на однакову кількість кроків. Йому не потрібно постійно вводити одне і те саме

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		24

значення, достатньо просто навести на поле і з'явиться список всіх введених раніше значень. Також це поле має обмеження, а саме користувач може вводити лише цифри або числа і тільки цілі. Це обмеження вводилося спираючись на те, що кількість кроків може бути записана лише числом та лише цілим. Метою цього обмеження є уникнення випадкових помилок при взаємодії з користувача з інтерфейсом.

`ActiveButton.jsx` – це компонента за допомогою якої візуалізується кнопка `Execute`. Ця кнопка запускає процес трансляції коду. Для створення цієї компоненти використовувалася компонента `Button` з бібліотеки `material-ui`. Після кліку на цю кнопку виконується запит на сервер. Це реалізовано з використанням навішування обробника подій на цю кнопку. Робиться це таким чином, що створюється функція в тілі якої відбуваються потрібні дії для досягнення потрібного результату. Далі ця новостворена функція передається, як параметр, самій компоненті. В даному випадку назва цього параметра була `onClick` та передавалася компоненті `Button`. Приклад того, як це реалізовано в програмному коді показано на рис.14.

```
const ActiveButton=(props)=>{
  return (
    <div style={{width:'100%',textAlign:'center'}}>
      <ButtonStyled variant="contained" onClick={()=>props.sendExecuteAsync()}>
        Execute
      </ButtonStyled>
    </div>
  )
}
const mapActionsToProps = (dispatch) =>
  bindActionCreators(
    {
      sendExecuteAsync:sendExecuteAsync
    },
    dispatch
  )
```

Рис.14 – Навішування обробника подій

Editor.jsx – це одна з основних компонент. Результатом відображення цієї компоненти на екрані є поле текстового вводу. Це поле виконує роль текстового редактора. Саме в це поле користувач і вводить текст тієї програми на мові програмування SIGNAL, яку він хоче транлювати в програму на мові С. Реалізовано цю компоненту з використанням TextField з бібліотеки material-ui. Також ця компонента є батьківською компонентою для Button, ImportButton.jsx та Notification.jsx. Як вже зазначалося раніше, компонента Button є компонентою з бібліотеки material-ui. Кнопка, яка реалізована за допомогою цієї компоненти має назву Save. Вона використовується користувачем для збереження написаного ним коду та відправлення цього коду на сервер. Ця кнопка також має алгоритм навішування обробника подій. Аналогічно до вище описаної кнопки, в цій кнопці також використовується параметр onClick. Проте тіло функції, яка буде виконуватися після натискання на кнопку відрізняється. Різниця полягає в тому, що на сервер відсилається зовсім інший запит. Цей запит не повертає ніякого результату трансляції. Він просто надсилає текст коду на сервер для подальшої роботи над цим кодом. Про інші дві дочірні компоненти, ImportButton.jsx та Notification.jsx, більш детально буде описано далі. Але основною задачею компоненти ImportButton.jsx є виведення на екран кнопки Import. Ця кнопка потрібна для того, щоб користувач міг не лише писати код від руки в текстовому редакторі, а і імпортувати потрібні йому файли з кодом. Стосовно компоненти Notification.jsx, то результатом її виконання є віконце повідомлення, яке сповіщає користувача вебдодатка, що дія, яка почалася вже завершилася, наприклад, відправлення запиту все виконалося.

ImportButton.jsx – ця компонента, як все зазначалося, потрібна для того, щоб користувач міг не лише писати код від руки в текстовому редакторі, а і імпортувати потрібні йому файли з кодом. Розберемо її більш детально. При натисканні на кнопку Import, яку і відображає дана компонента, спрацьовує

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		26

функція-обробник, яка є навішана на кнопку. Вона викликає вікно імпорту. В цьому вікні користувач може бачити всі файли, що містяться на його локальному комп'ютері. Далі користувач має обрати файл, який його цікавить та підтвердити свій вибір, натиснувши на кнопку Open. Вікно імпорту автоматично зачиниться та програмний код, який міститься в файлі буде відображено на екрані. В подальшому цей програмний код і буде відправлено на сервер для подальшої роботи з ним.

Scanner.jsx – це компонента, яка відображає на екран результати роботи першого етапу трансляції. Назва цього етапу аналогічна до назви компоненти, що чудово ілюструє дотримання вище описаного правила. Ця компонента, як і компонента Editor.jsx є батьком для Notification.jsx, також, крім неї, ще для ScannerTable.jsx та InformationAboutCurrentLexem.jsx. Загалом, ця компонента виводить на екран таблицю з лексемами. Ця таблиця має такі колонки, Name та Type. В колонці Name записані імена вже сформованих лексем. В колонці Type зазначений тип цих лексем. Реалізована ця таблиця за допомогою компоненти ScannerTable.jsx. Більш детально про її реалізацію таблиці всередині компоненти ScannerTable.jsx буде описано далі. Також варто зазначити, що для комфортної роботи з цією таблицею та для гарного зовнішнього вигляду було додано можливість, так званого, скролу цієї таблиці. Нова лексема додається до таблиці тільки тоді, коли ця лексема повністю сформована. Поки лексема прочитана програмою лише частково, ця частина лексеми відображається в блоці над табличкою. В цьому блоці відображається та частина лексеми, яка прочитана програмою на даний момент часу. Також зазначений ймовірний тип цієї лексеми. Припущення на рахунок типу лексеми, робиться на основі цієї частини поточної лексеми, яку програма вже встигла прочитати та обробити. Тому, якщо після завершення зчитування лексеми повністю, тип зміниться і до таблиці запишеться інший, то це вважається нормальною поведінкою системи. Для реалізації цього блоку використовується компонента та

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		27

InformationAboutCurrentLexemStyled.jsx. Вона являє собою звичайний тег div і створюється прямо в стильовій компоненті InformationAboutCurrentLexemStuled.jsx. Така практика використовується досить широко і в тих ситуаціях коли створюється компонента, що в своєму тілі має лише тег div. Подібного роду компоненти використовуються, зазвичай, як стильові обгортки. Створення такої компоненти зображено на рис.15 на прикладі компоненти InformationAboutCurrentLexemStyled.jsx. Всі компоненти, які обгорнуті в цю компоненту, автоматично стають її дітьми. Для того, щоб отримати значення цих дітей та вставити їх в правильне місце використовується об'єкт props та його поле children. Використання цієї компоненти обгортки в компоненті Scanner.jsx показано на рис.16.

```
import React from 'react'
import { withStyles } from '@material-ui/core'

const InformationAboutCurrentLexemStyled=(props)=><div className={props.classes.root}>{props.children}</div>
export default withStyles({
  root: {
    border:'1px solid grey',
    borderRadius:'4px',
    margin:'5px',
    paddingLeft:'10px'
  }
})(InformationAboutCurrentLexemStyled)
```

Рис.15 – Створення стильової обгортки

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		28

```

const Scanner=(props)=>{
  return(
    <React.Fragment>
      {
        (props.type !== '') && <InformationAboutCurrentLexemStyled>
          <p>{`Current token: ${props.type==='WHITESPACE'}' "':props.name}`}</p>
          <p>{`Hypothetical type of current token: ${props.type}`}</p>
        </InformationAboutCurrentLexemStyled>
      }
      <ScannerTable/>
      <Notification/>
    </React.Fragment>
  )
}

```

Рис.16 – Використання стильової обгортки

Parser.jsx – це компонента, яка виводить на екран результат роботи другого етапу транслятора. Цей етап також має однойменну назву з назвою компоненти. Сама по собі, ця компонента має мале тіло, адже вся логіка інкапсульована в дочірній компоненті ParserTree.jsx. Назва цієї компоненти саме така, адже в ній будується дерево розбору. Основною компонентою, яка і реалізує це дерево є компонента Tree з бібліотеки react-d3-tree. На вхід цій компоненті приходять масив об'єктів, які в свою чергу надходять з сервера, як результат роботи відповідного етапу трансляції. Цей об'єкт має такі поля: name та children. Як вже зазначалося раніше саме таку структуру повертає сервер, як відповідь на відповідний запит. Приклад формування відповіді на сервері зображено на рис.17.

```

(defmethod to-js ((term term))
  (new-js
    ("name" (format nil "~A" (term-head term)))
    ("children" (mapcar #'to-js (term-components term)))))

```

Рис.17 – Формування відповіді на сервері

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		29

Generator.jsx – ця компонента відображає на екран результат роботи третього етапу транслятора, а саме етапу generator. Як видно і в цьому випадку назва компоненти і назва етапу збігаються. Являє собою ця компонента текстовий редактор. Він реалізований також за допомогою TextField з бібліотеки material-ui. Проте цей текстовий редактор має особливість. Користувач може лише бачити той текст програми, який приходить, як відповідь з сервера, але не він не може редагувати та змінювати його. Такий функціонал реалізована для забезпечення фіксації правильних результатів роботи третього етапу трансляції. Також ще одним дочірнім компонентом для Generator.jsx є Notification.jsx. Функції цієї компоненти в даному випадку повністю збігаються з її функціями в попередніх варіантах використання. Вона виводить на екран вікно-повідомлення про успішне або ні завершення запитуна сервер. На вхід компоненті Generator.jsx приходить рядок програмного коду на мові програмування C, який в свою чергу є результатом роботи третього етапу трансляції, що виконується програмою на боці сервера.

Error.jsx – ця компонента з’являється лише тоді, коли в вхідному програмному коді на мої програмування SIGNAL є лексичні, синтаксичні або семантичні помилки. Зрозуміло, що ві ці помилки виникають на різних етапах трансляції. Так, наприклад, лексичні можуть виникати на етапі scanner, синтаксичні на етапі parser, а семантичні на етапі generator. Зрозуміло і те, що вкладка Error не з’явиться на етапі scanner, якщо помилка в програмному коді синтаксична і буде відомою лише на етапі parser, також ця вкладка не з’явиться на етапі parser, якщо помилка семантична і з’явиться на етапі generator. Загалом вкладка Error має вигляд таблиці. Ця таблиця має такі колонки, як:

- error message
- stage
- row
- column

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		30

Error message – це і є саме повідомлення про помилку. В ньому зазначено що саме стало причиною помилки. Таке повідомлення допоможе користувачу виправити помилкову частину вхідного програмного коду.

Stage – в цьому полі зазначено на якому саме етапі виявилася помилка в програмному тексті на мові SIGNAL.

Row – в цьому полі записаний рядок в якому виявлена помилка в програмному кодї.

Column – в цьому полі записаний стовбець в якому виявлена помилка в програмному кодї.

Реалізована ця таблиця за допомогою всієї групи табличних компонент з бібліотеки material-ui, а саме:

- TableRow
- TableCell
- Table
- TableHead
- TableBody

Також ця таблиця має скрол. Його мета полегшити роботу з таблицею. Для реалізації скрола використовувалася компонента Scrollbars. Вона бралася з бібліотеки react-custom-scrollbars. Ця компонента використовувалася з метою виконати скрол в більш сучасному та функціональному дизайні. Також використовувалася бібліотека AutoSizer з бібліотеки react-virtualized. Ця компонента вираховує довжину та висоту тих компонент, які знаходяться в середині неї, тобто своїх дочірніх компонент. Ці параметри потрібні для коректної роботи компоненти Scrollbars.

Notification.jsx – цей компонент неодноразово згадувався раніше. Як все відомо результатом відображення цієї компоненти є вікно-повідомлення про

успішність або невдачу завершення якогось процесу. Являє це вікно собою прямокутник з певним написом. Колір цього прямокутника залежить від результату завершення процесу. Якщо процес завершився успішно, то прямокутник буде зелений, а за умови помилкового завершення процесу, прямокутник буде червоний. Аналогічно з кольорами прямокутника, від успішності завершення процесу залежить і текст, який написаний на прямокутнику. При успішному завершенні процесу є повідомлення, що процес завершено успішно та зазначено на якому етапі знаходиться транслятор та скільки кроків вже пройдено. Якщо процес завершився невдало, то з'явиться повідомлення про помилкове завершення процесу. Також дана компонента має дочірню компоненту `NotificationHelper.jsx`. Саме в ній і інкапсульована вся логіка вибору тексту, а також структура та стилі цього тексту. В корені всіх цих компонент лежить `ToastContainer` з бібліотеки `react-toastify`. Саме ця компонента і реалізує весь алгоритм роботи вікон-повідомлень. Значення цих вікон для вебдодатку несуть велике змістовне навантаження. Вони допомагають користувачеві краще та більш продуктивно взаємодіяти з інтерфейсом. Розуміти, що відбувається з вебдодатком в даний момент часу. Допомагають уникнути зайвого непорозуміння, а також роблять інтерфейс більш нативним.

Папка `modules` – це папка, яка містить файли, що відповідають за state програми та роботу з ним. Саме в ній і реалізується інтеграція `redux` в `react`. Ця папка складається з `actions`, `reducers`, `selectors`, `saga`. Папки `actions`, `reducers`, `selectors` відповідають аналогічним за назвою головним принципам `redux`. Розберемо більш детально їх вміст.

Папка `actions` – ця папка містить лише один файл. Назва цього файлу `index.js`. В ньому містяться структури, котрі передають дані з додатку в `store`. Для реалізації цієї передачі даних використовується функція зображена на рис.18.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		32

```
store.dispatch()
```

Рис.18 – Функція для передачі даних в store

Кожна структура в цьому файлі має поля `type` та `payload`. Поле `type` відповідає за тип події, яка виконується. Поле `payload` зберігає самі дані, які треба передати в `store`. Прикладом повного програмного коду однієї такої структури є рис.19.

```
export const setID = (id) => ({  
  type: 'SET_ID',  
  payload: {id}  
})
```

Рис.19 – Приклад структури actions

Папка `reducers` – в цій папці також міститься лише один файл. Назва файлу `compiler.js`. Суть коду, який написаний в цьому файлі полягає в тому, що він допомагає редагувати, змінювати, доповнювати або видаляти значення полів `store`. Також в цьому файлі проходить ініціалізація початкового значення `state`. Приклад саме для даного вебдодатку приведено на рис.20.

```
const initialState={
  code:'',
  id:'',
  steps:0,
  counter: 0,
  generator:'',
  currentLexem:{type:'',name:''},
  rowsScanner:[],
  treeParser:[],
  oldValue:0,
  currentValue:'scanner',
  skipCount:0,
  error:[],
  isDisabled:{
    scanner:false,
    parser:false,
    generator:false
  }
}
```

Рис.20 – Початкові значення state

Проте варто зазначити, що в жодній з функцій, які представлені в цьому файлі не можна знайти наступного:

- виконання слайд-ефектів
- виклик не чистих функцій
- пряма зміна даних, які прийшли в аргументах

Приклад такої функції приведено на рис.21.

```

case 'CLEAN_OLD_RESULT':
return {
  ...state,
  currentLexem:{type:'',name:''},
  rowsScanner:[],
  treeParser:[],
  oldValue:0,
  currentValue:'scanner',
  skipCount:0,
  error:[],
  isDisabled:{
    scanner:false,
    parser:false,
    generator:false
  }
}
}

```

Рис.21 – Приклад reducer

Варто також зауважити, що функція не змінює значення state напряду, вона робить його копію і вже її змінює. Копія робиться на рахунок такого хуку, як ...state. Також слід зазначити, що будь-яка функція reducer являє собою таку умовну конструкцію, як switch. Умовою є значення типу яке передається, як параметр. Прикладом слугує рис.22. Зрозуміло, що присутня гілка default. Ця гілка відповідає за повернення попередньої версії state. Це є дуже важливим повертати попередню версію для невідомого або неопрацьованого action. Приклад зображено на рис.23. Тобто основною функцією reducers є зміна state додатку у відповідь на action. Приймає reducer попереднє значення state, а повертає наступне, змінюючи його копію залежно від тип значень, які прийшли від action.


```

switch (action.type) {
  case 'SET_CODE':
    return {
      ...state,
      code: action.payload.code
    }
  case 'SET_ID':
    return {
      ...state,
      id: action.payload.id
    }
  case 'SET_STEPS':
    return {
      ...state,
      steps: action.payload.steps,
    }
}

```

Рис.22 - Структура switch

```

default:
  return state

```

Рис.23 – Гілка default

Папка selectors – ця папка містить файл index.js. Основною функцією selectors є взяття зі state значень його полів. Приклад функції selector показано на рис.24.

```

export const getCode=(state)=>state.compiler.code
export const getCounter=(state)=>state.compiler.counter
export const getId=(state)=>state.compiler.id

```

Рис.24 – Функції selectors

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		36

Як видно з рисунку, на вхід вони приймають весь об'єкт state, а потім беруть потрібне поле. Інколи для прискорення роботи вебдодатку використовують функцію createSelector з бібліотеки reselect. Вона створює кеш значень і тому не перераховує і не шукає нових значень полів щоразу при виклику, але тільки за умови, що значення цих полів не змінилося. Якщо значення поля змінило своє значення, то selector працює в звичайному режимі. Приклад використання функції createSelector приведено на рис.25.

```
export const totalSelector = createSelector(  
  subtotalSelector,  
  taxSelector,  
  (subtotal, tax) => ({ total: subtotal + tax })  
);
```

Рис.25 – Використання createSelector

Папка saga являє собою папку worker та файл watcher.js. Worker в свою чергу складається з таких файлів, як:

- executeWorker.js
- saveWorker.js
- notifyWorker.js
- importFileWorker.js
- setValueWorker.js.

Ця папка і всі файли, які вона містить реалізують асинхронність всієї програми. Розберемо функції кожного файлу.

executeWorker.js – цей файл забезпечує надсилання запиту на сервер з метою виконання певної кількості кроків на певному етапі трансляції. Разом з

запитом вона передає і параметри для сервера, а саме id, steps, skipCount. Всі ці параметри функція отримує з selector. Програмний код відправки запиту на сервер приведено на рис.26. Також до її функцій входить передати значення отриманого результату в actions. Залежно від етапу трансляції дані, які надійшли з серверу будуть передані різним actions. Так на етапі scanner будуть викликатися actions приведені на рис.27, на етапі parser будуть викликатися actions зображені на рис.28, а на етапі generator actions, що показані на рис.29.

```
const id = yield select(getId)
const steps = yield select(getSteps)
yield put(setCounter(steps))
const skipStagesCount=yield select(getSkipCount)
console.log('skipStagesCount: ', skipStagesCount);
const body = JSON.stringify({id,steps,skipStagesCount})
console.log('body: ', body);
const response = yield call(fetch, 'http://localhost:3001/compiler', {
  method: 'POST',
  body
})
```

Рис.26 – Відправлення запиту на сервер

```
const result = yield response.json()
if(result.stage==='scanner'){
  const {type,name}=result.currentLexem
  const {parsedCode,errors}=result
  const counter = yield select(getCounter)
  yield put(setCurrentLexem(type,name))
  yield put(setRowsForScanner(parsedCode))
  yield put(notify({ message: `Analyzed ${counter}.
                        Stage: ${result.stage}`, type: 'success' })))
  yield put (setError(errors))
}
```

Рис.27 – Виклик actions на етапі scanner

```

else if(result.stage==='parser'){
  const {tree,tokens,errors}=result
  const newDisabled={scanner:true,parser:false,generator:false}
  yield put(setCounter(0))
  yield put(setTreeParser(tree))
  yield put(setRowsForScanner(tokens))
  yield put(changeDisabled(newDisabled))
  yield put (setCurrentValue('parser'))
  yield put(notify({ message: `Ready!
                        Stage: ${result.stage}`, type: 'success' })))
  yield put (setError(errors))
}

```

Рис.28 - Виклик actions на етапі parser

```

else {
  const {tree,tokens,generatedCode,errors}=result
  const newDisabled={scanner:true,parser:true,generator:false}
  yield put(changeDisabled(newDisabled))
  yield put(setTreeParser(tree))
  yield put(setRowsForScanner(tokens))
  yield put (setCurrentValue('generate'))
  yield put (setGenerator(generatedCode))
  yield put (setError(errors))
  yield put(notify({ message: `Ready!
                        Stage: ${result.stage}`, type: 'success' })))
}

```

Рис.29 - Виклик actions на етапі generator

saveWorker.js - цей файл забезпечує надсилання запиту на сервер з метою збереження коду, який ввів користувач. Разом з запитом вона передає і параметри для сервера, а саме текст програмного коду. Всі ці параметри функція отримує з selector. Програмний код відправки запиту на сервер приведено на рис.30.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		39

```
const body = yield select(getCode)
const response = yield call(fetch, 'http://localhost:3001/', {
  method: 'POST',
  body
})
```

Рис.30 – Надсилання запиту на сервер

Також до її функцій входить передати значення отриманого результату в actions. В даному випадку сервер повертає значення id. Це ідентифікатор програмного коду, під яким цей код зберігається на сервері. Він потрібен для подальшої роботи з поточним програмним кодом. Приклад виклику action приведено на рис.31.

```
if (response.ok) {
  const {id} = yield response.json()
  yield put(setId(id))
  yield put(notify({ message: 'Success save file', type: 'success' }))
}
```

Рис.31 – Виклик потрібних actions

notifyWorker.js – цей файл забезпечує роботу віконця-повідомлення. До її функцій входить передати значення type, message, onClose в actions. Покзано це на рис.32.

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		40

```

import { call } from 'redux-saga/effects'
import notify from '../../utils/notify'

export function* notifyWorker({payload}) {
  yield call(
    notify,
    payload.notification.message,
    payload.notification.type,
    payload.notification.onClose
  )
}

```

Рис.32 – Виклик action та передача їй параметрів

importFileWorker.js – цей файл забезпечує можливість імпорту будь-якого файлу з локального комп'ютера користувача. Також до її функцій входить передати значення програмного коду файлу, який за імпортований в actions. В подальшому цей текст буде передано на сервер, де йому буде присвоєно ідентифікатор.

setValueWorker.js - цей файл забезпечує обчислення коректного значення skipCount. Також до її функцій входить передати значення отриманого результату skipCount в actions. Залежно від етапу трансляції, яку обрав користувач значення skipCount буде різним. Програмний код цієї функції приведено на рис.33.

```

import { put, select } from 'redux-saga/effects'
import {getOldValue} from '../selectors/index'
import {setCurrentValue,setSkipCount,setOldValue} from '../actions/index'

export default function* setValueWorker({payload:{value}}) {
  const oldValue = yield select(getOldValue)
  const currentValue= value==='scanner'?0:value==='parser'?1:2
  const skipCount=currentValue-oldValue
  console.log('skipCount: ', skipCount);
  yield put(setSkipCount(skipCount))
  yield put(setCurrentValue(value))
  yield put(setOldValue(currentValue))
}

```

Рис.33 – Програмний код setValueWorker.js

watcher.js – в цій функції оголошуються всі вище приведені функції, а також присвоюється тип цим функціям з папки worker. Приклад оголошення функцій приведено на рис.34.

```

function* sendExecuteWatcher() {
  yield takeEvery('SEND_EXECUTE_ASYNC', executeWorker)
}
function* sendCodeWatcher() {
  yield takeEvery('SEND_CODE_ASYNC', saveWorker)
}
function* setValueWatcher() {
  yield takeEvery('SET_VALUE_ASYNC', setValueWorker)
}

function* importFileWatcher() {
  yield takeEvery('IMPORT_FILE_ASYNC', importFileWorker)
}

function* notifyWatcher() {
  yield takeEvery('NOTIFY', notifyWorker)
}
export default function*() {
  yield all([sendExecuteWatcher(),sendCodeWatcher(),setValueWatcher(),importFileWatcher(),notifyWatcher()])
}

```

Рис.34 – Оголошення функцій

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		42

Папка `reducer` – в цій папці створюється `store`, а також виконується підключення `saga` та `reducer`. Ця папка є завершальним етапом об'єднання `react` та `redux` і функціональної та стильової частин вебдодатку. Вона містить такі папки:

- `store`
- `saga`
- `reducer`

`Store` – створення глобального `store` вебдодатку та підключення `saga` і `reducer` до нього. Приклад зображено на рис.35.

```
import rootReducer from './reducer'
import rootSaga from './saga'

const store = () => {
  const sagaMiddleware = createSagaMiddleware()

  const rootReducerWithBatching = () => enableBatching(rootReducer())

  const storeObj = createStore(
    rootReducerWithBatching(),
    composeWithDevTools(applyMiddleware(sagaMiddleware))
  )
  sagaMiddleware.run(rootSaga)

  return storeObj
}

export default store
```

Рис.35 – Програмний код `saga`

`Saga` – виконується об'єднання локальної `saga` вебдодатка, який буде потім підключатися в `store`. Приклад приведено на рис.36.


```

import { all } from 'redux-saga/effects'
import sendWatcher from '../modules/saga/watcher'

export default function* rootSaga() {
  yield all([sendWatcher()])
}

```

Рис.36 - Об'єднання локальної saga

Reducer – об'єднує локальний reducer вебдодатку, який буде потім підключатися в store. Приклад показано на рис.37.

```

import { combineReducers } from 'redux'
import compiler from '../modules/reducers/compiler'

const rootReducer = () =>
  combineReducers({
    compiler
  })
export default rootReducer

```

Рис.37 - Об'єднання локальної reducer

3.3 Опис інтерфейсу вебдодатку

Інтерфейс вебдодатку складається з заголовку, бокової панелі та основної частини, де знаходяться текстовий редактор, таблиця лексем, дерево розбору, згенерований код та таблиця помилок, якщо вони присутні. Бокова панель містить кнопки вибору етапу трансляції (scanner, parser, generator), поле для вводу кількості кроків трансляції та кнопку Execute, яка відправляє запит на

сервер. Також, наявні кнопки Save та Import, перша зберігає код введений у вікні текстового редактора, а друга імпортує файл з вхідною програмою. Загальний вигляд інтерфейсу приведено на рис.38.



Рис.38 - Загальний вигляд інтерфейсу вебдодатка

Загалом інтерфейс доволі простий, нативний та інтуїтивно зрозумілий. Для досягнення потрібного результату треба у вкладці Editor написати або заімпортувати, за допомогою вище зазначеної кнопки Import, програмний код вхідної програми. Варто зазначити, що вхідною мовою для даного онлайн-компілятора є мова SIGNAL, адже саме вона використовується в курсі «Основи проектування трансляторів». Після натискання на кнопку Import з'явиться вікно з файлами серед яких треба обрати потрібний (рис.39).

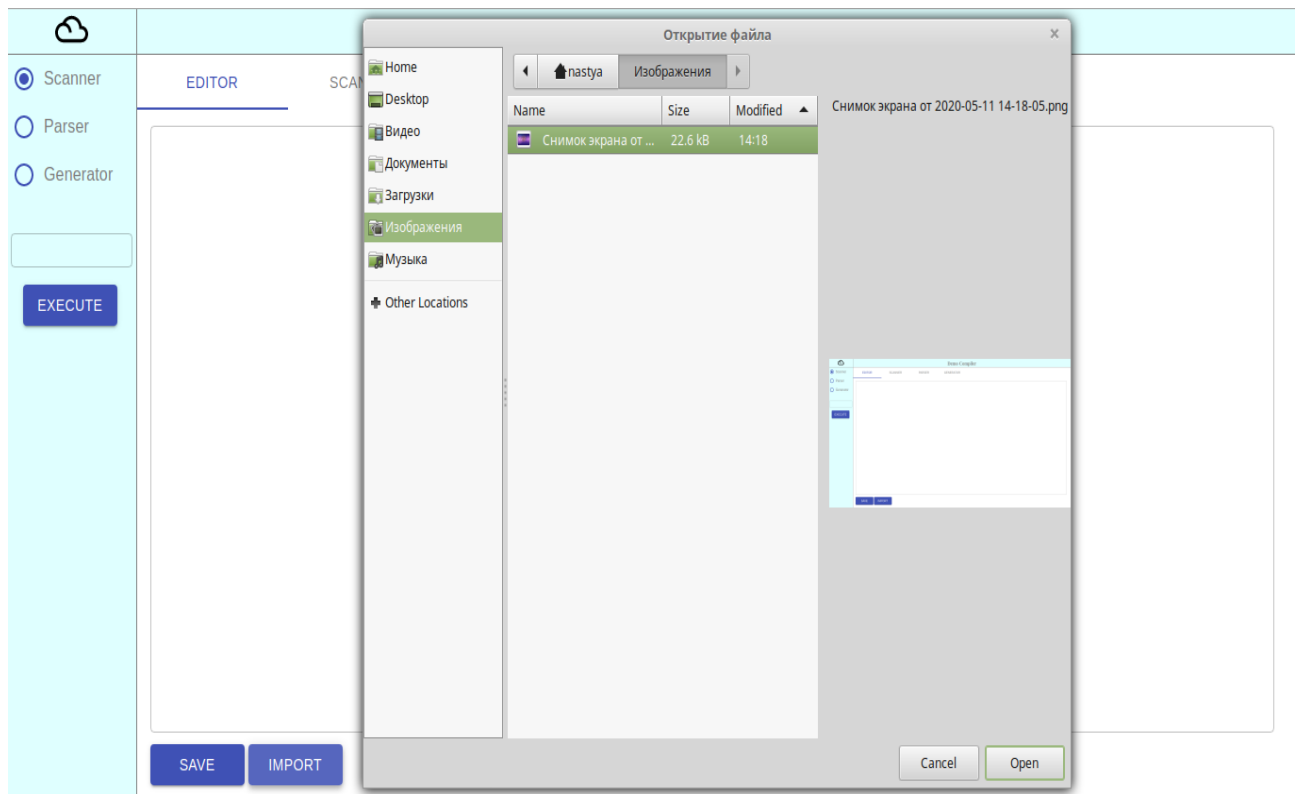


Рис.39 - Вікно імпорту

Після вибору потрібного файлу, текст програмного коду з'явиться на екрані. Далі слід зберегти код за допомогою кнопки Save (рис.40). Потім користувач має змогу обрати етап трансляції, використовуючи радіо-кнопки (scanner, parser, generator), а також ввести кількість кроків трансляції. Для отримання результату, потрібно натиснути кнопку Execute.

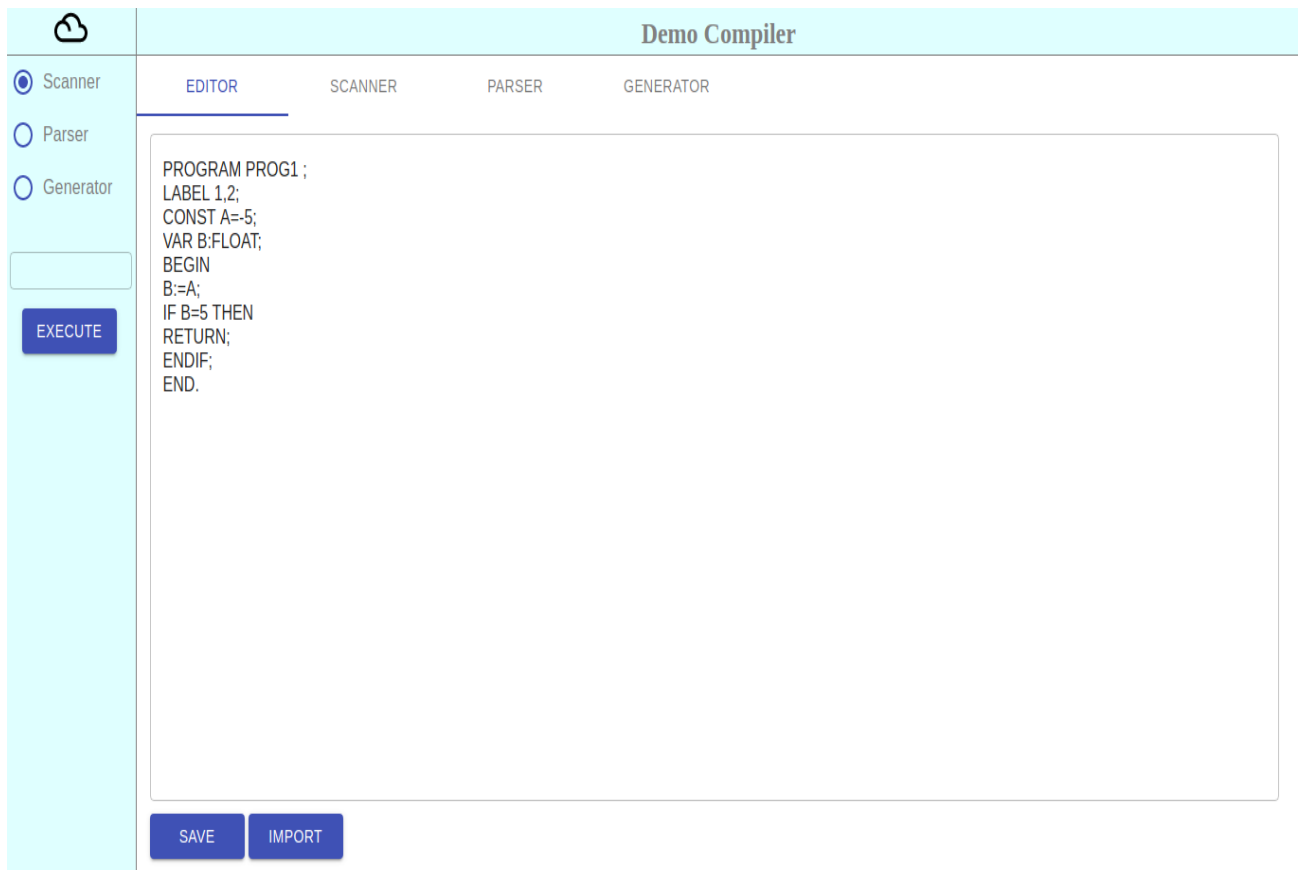
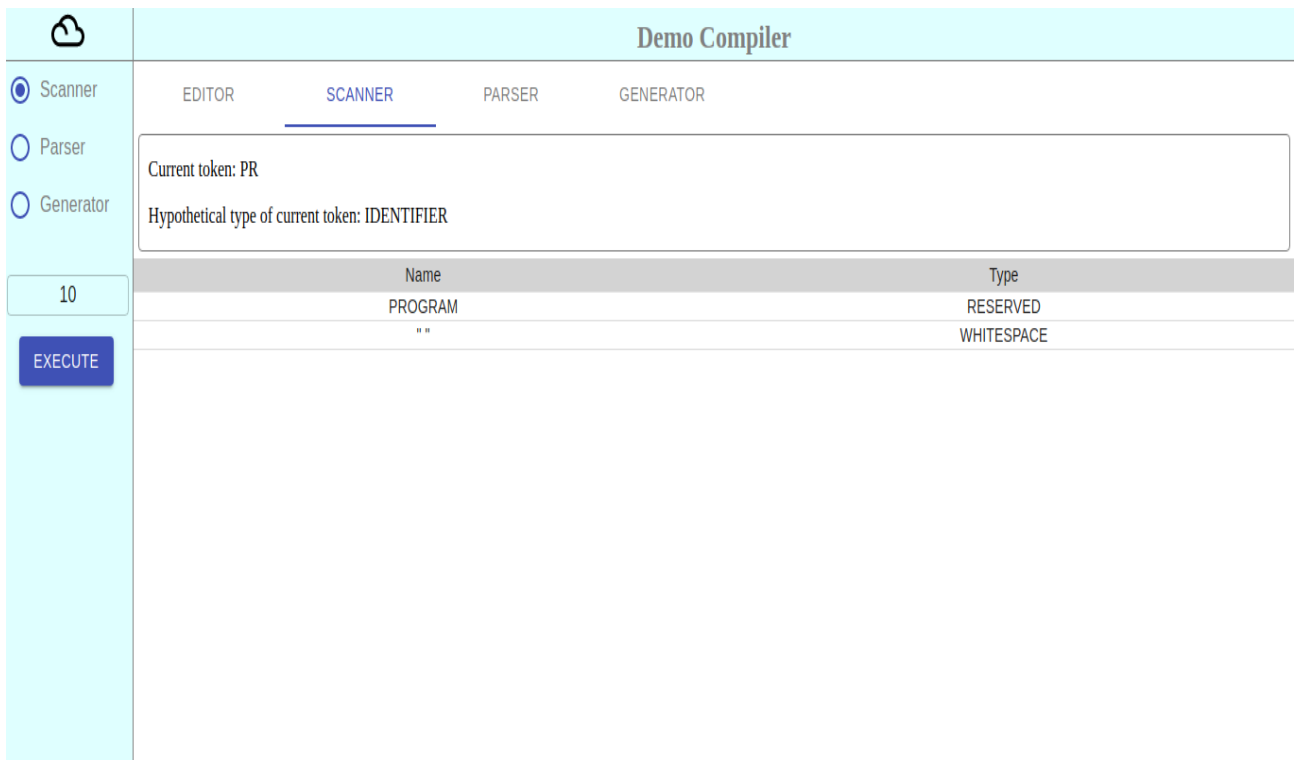


Рис.40 - Результат імпорту

Для кожного етапу трансляції існує вкладка в якій буде відображено результат виконання певної кількості кроків трансляції. Назви цих вкладок аналогічні назвам етапів, а саме Scanner, Parser, Generator. Прикладом, який ілюструє результат трансляції на етапі scanner є рис.41.



Demo Compiler

EDITOR SCANNER PARSER GENERATOR

Current token: PR
Hypothetical type of current token: IDENTIFIER

Name	Type
PROGRAM	RESERVED
" "	WHITESPACE

Рис.41 - Результат на етапі scanner

Як видно з вище приведенного рисунка, результатом роботи транслятора на етапі scanner є таблиця лексем. В ній назначено ім'я та тип лексеми. Також користувач може бачити, які поточні символи перевіряються і, який тип їм, гіпотетично, може бути присвоєний.

Результат роботи програми на етапі parser зображено на рис.42.

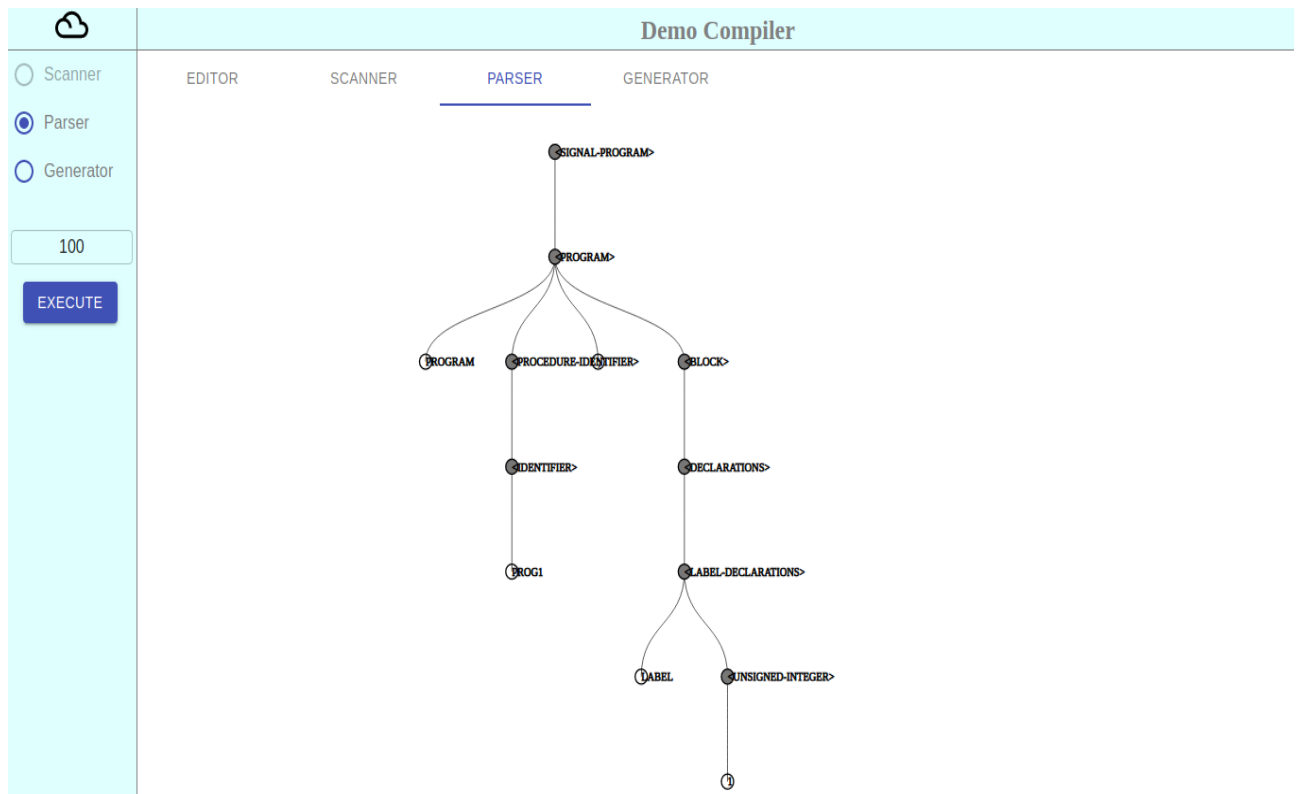


Рис.42 - Результат на етапі parser

На цьому етапі, в якості результату, користувач може спостерігати дерево розбору. Воно відображає синтаксичну структуру тієї послідовності, яка подається на вхід. У вершинах дерева розбору зазначені нетермінали та термінали. Ті вершини, які є терміналами – це листки. Ті вершини, які є нетерміналами мають потомків. Ці потомки відповідають розкриттю свого батька по певному правилу.

Приклад того, який результат буде отримано на етапі generator, приведено на рис.43.

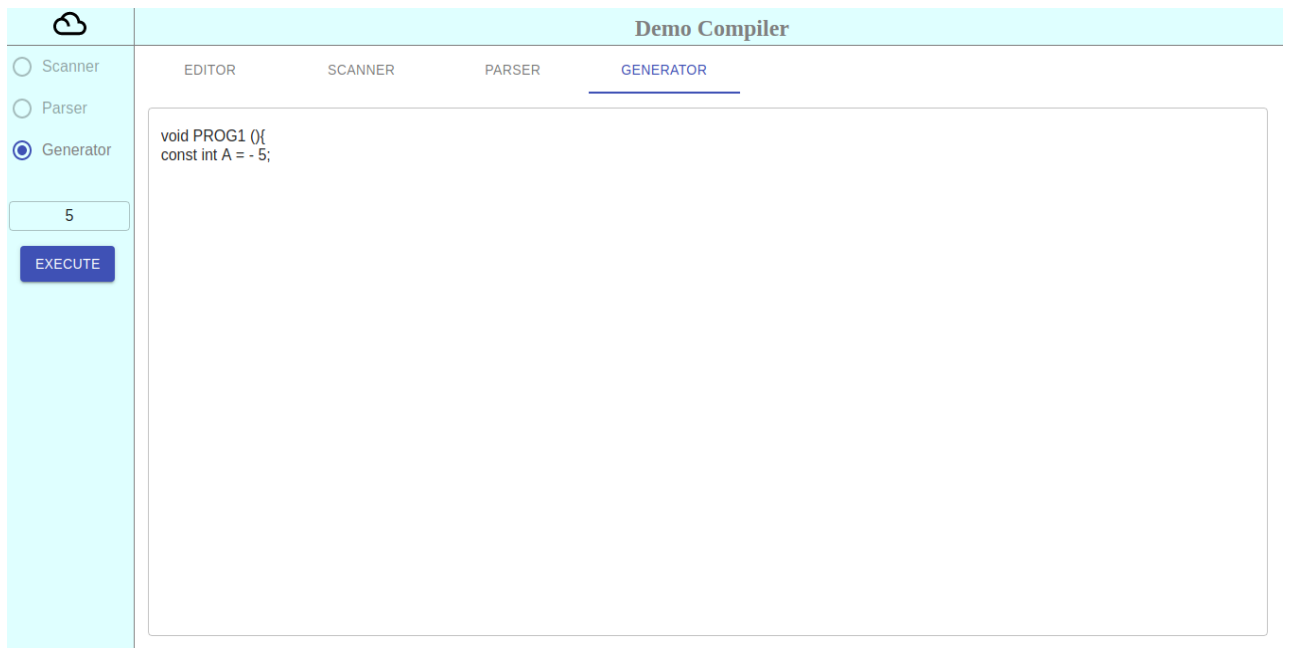


Рис.43 - Результат роботи generator

На цьому етапі користувач може побачити програмний код на вихідній мові. В даному випадку це мова програмування C. Тобто, кінцевий результат роботи програми це перетворення конструкцій, які написані на мові програмування SIGNAL, в аналогічні за смисловою загрузкою, конструкції на мові C.

Висновок до 3-го розділу

Отже, виходячи із вище зазначеного опису, можна зробити висновок, що завдяки використанню бібліотек React та Redux, вдалося досягти правильної організації коду за технологією MVC. Кожен модуль програми має свою конкретну задачу і виконує її. Також добре проглядається і відповідність програмного коду загальним правилам написання коду. Чудовим надбанням стало використання Material-UI для стилізації, адже додавати або змінювати стилі компонент було легко та не займало багато часу, адже всі стилі, що належать компоненті, знаходяться в одному файлі і впливають лише на конкретний елемент, тим самим не ламаючи стилі сусідніх елементів.

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		50

4.ТЕСТУВАННЯ ВЕБДОДАТКУ

Вебдодаток, який є результатом дипломного проекту тестувався на коректну роботу всіх етапів трансляції, а саме: scanner, parser, generator. Також перевірялася поведінка вебдодатку при помилковому програмному вхідному коді. Далі на рисунках буде показано, які результати були отриманні в ході тестування вебдодатку. Варто зазначити, що вхідний програмний код написаний на мові SIGNAL. По закінченню роботи транслятора користувач побачить вихідний програмний код на мові С. Було проведено тестування на трьох різних варіантах коду. Також протестувалися всі можливі ситуації де може виникнути помилка.

Приклад вхідного коду номер 1 на якому тестувалася програма зображена на рис.44.

```
PROGRAM PROG1;  
LABEL 1,2;  
CONST A=-5;  
VAR B:FLOAT;  
BEGIN  
B:=A;  
IF B=5 THEN  
RETURN;  
ENDIF;  
END.
```

Рис.44 - Вхідний програмний код номер 1 на мові SIGNAL

Після завантаження коду і збереження його на сервері, оберемо етап scanner та просунемося на 10 кроків. Отримаємо результат зображений на рис.45.

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		51

Demo Compiler

EDITOR **SCANNER** PARSER GENERATOR

Scanner Parser Generator

10

EXECUTE

Current token: PR
Hypothetical type of current token: IDENTIFIER

Name	Type
PROGRAM	RESERVED
" "	WHITESPACE

Рис.45 - Результат етапу scanner

Коли етап scanner завершиться користувач побачить повну таблицю лексем. Приклад такої таблиці зображено на рис.46.

Demo Compiler

EDITOR **SCANNER** PARSER GENERATOR

Scanner **Parser** Generator

10

EXECUTE

Name	Type
PROGRAM	RESERVED
" "	WHITESPACE
PROG1	IDENTIFIER
" "	WHITESPACE
;	DELIMITER
" "	WHITESPACE
LABEL	RESERVED
" "	WHITESPACE
1	UNSIGNED-INTEGER
;	DELIMITER
2	UNSIGNED-INTEGER
;	DELIMITER
" "	WHITESPACE
CONST	RESERVED
" "	WHITESPACE
A	IDENTIFIER
=	DELIMITER
-	DELIMITER
5	UNSIGNED-INTEGER
;	DELIMITER
" "	WHITESPACE
VAR	RESERVED

Рис.46 - Кінцевий результат етапу scanner

Наступним етапом трансляції є parser. Просунувшись на 10 кроків матимемо результат зображений на рис.47.

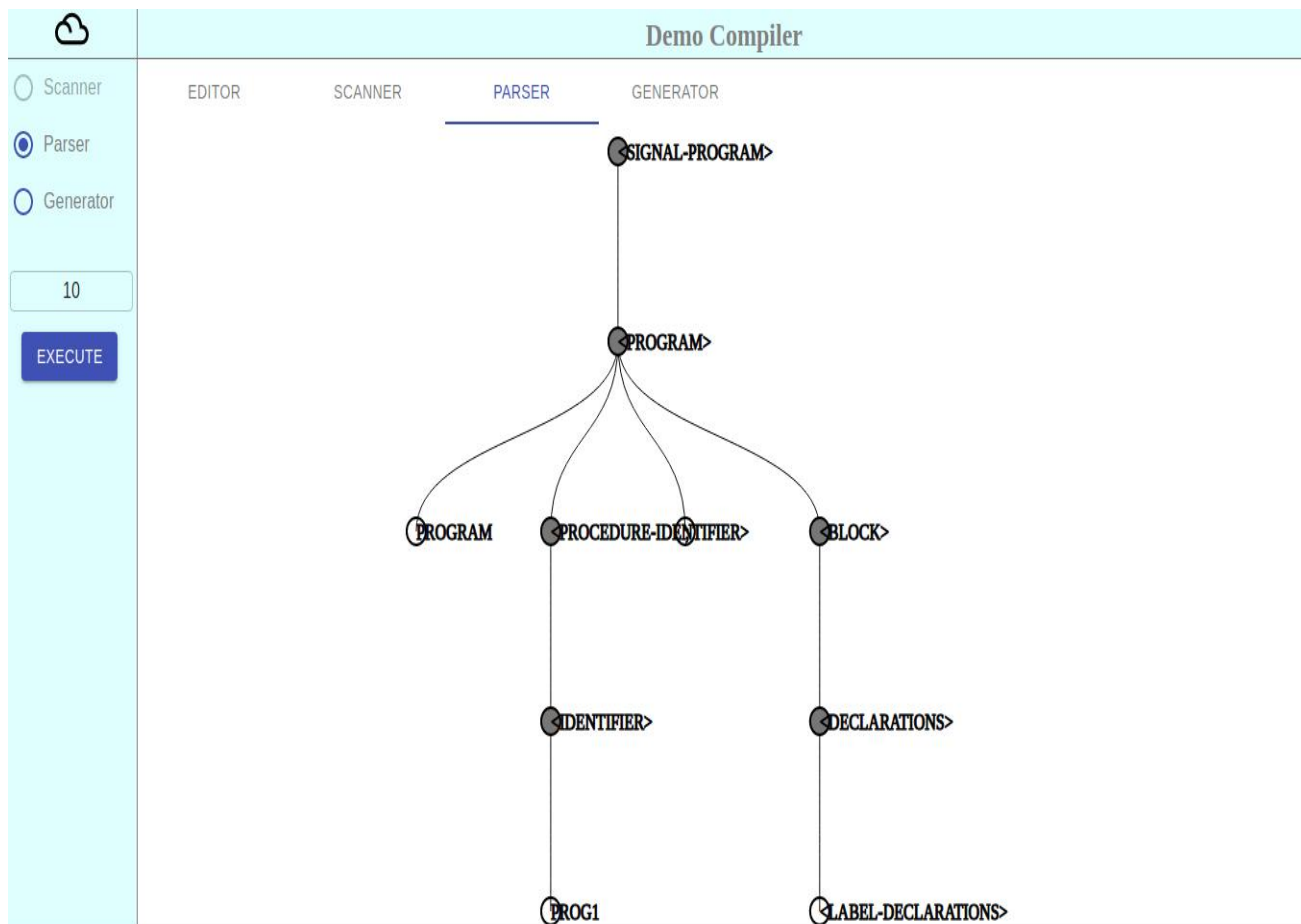


Рис.47 - Результат етапу parser

Кінцевий результат цього етапу зображено на рис.48.

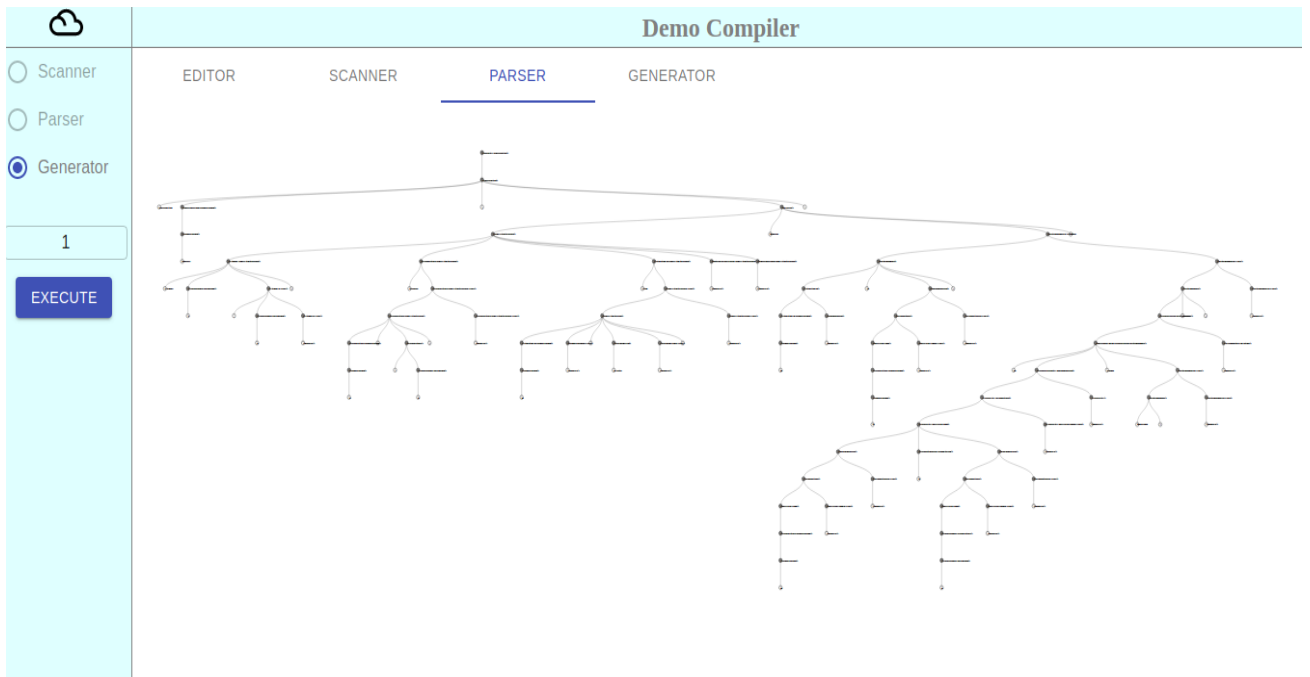


Рис.48 - Кінцевий результат етапу parser

Наступний етап – це generator. Зробивши 5 кроків, отримаємо результат зображений на рис.49.



Рис.49 - Результат роботи етапу generator

Пройшовши весь этап отримаємо результат зображений на рис.50.



Рис.50 - Кінцевий результат роботи етапу generator

Приклад вхідного коду номер 2 на якому тестувалася програма зображена на рис.51.

```
PROGRAM PROG1;  
LABEL 1,2;  
CONST A=-5;  
VAR B:FLOAT;  
BEGIN  
LOOP B:=B+1; ENDLOOP;  
RETURN;  
END.
```

Рис.51 - Вхідний програмний код номер 2 на мові SIGNAL

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		55

Після завантаження коду і збереження його на сервері, оберемо етап scanner та просунемося на 10 кроків. Отримаємо результат зображений на рис.52.

The screenshot shows the 'Demo Compiler' interface. On the left sidebar, 'Scanner' is selected. The main area has 'SCANNER' as the active tab. Below the tabs, the current token is 'PR' and its hypothetical type is 'IDENTIFIER'. A table below shows the following tokens:

Name	Type
PROGRAM	RESERVED
" "	WHITESPACE

Рис.52 - Результат етапу scanner

Коли етап scanner завершиться користувач побачить повну таблиця лексем. Приклад такої таблиці зображено на рис.53.

Demo Compiler			
EDITOR	SCANNER	PARSER	GENERATOR
Name	Type		
PROGRAM	RESERVED		
""	WHITESPACE		
PROG1	IDENTIFIER		
""	WHITESPACE		
:	DELIMITER		
""	WHITESPACE		
LABEL	RESERVED		
""	WHITESPACE		
1	UNSIGNED-INTEGER		
:	DELIMITER		
2	UNSIGNED-INTEGER		
:	DELIMITER		
""	WHITESPACE		
CONST	RESERVED		
""	WHITESPACE		
A	IDENTIFIER		
=	DELIMITER		
-	DELIMITER		
5	UNSIGNED-INTEGER		
:	DELIMITER		
""	WHITESPACE		
VAR	RESERVED		

Рис.53 - Кінцевий результат етапу scanner

Наступним етапом трансляції є parser. Просунувшись на 10 кроків матимемо результат зображений на рис.54.

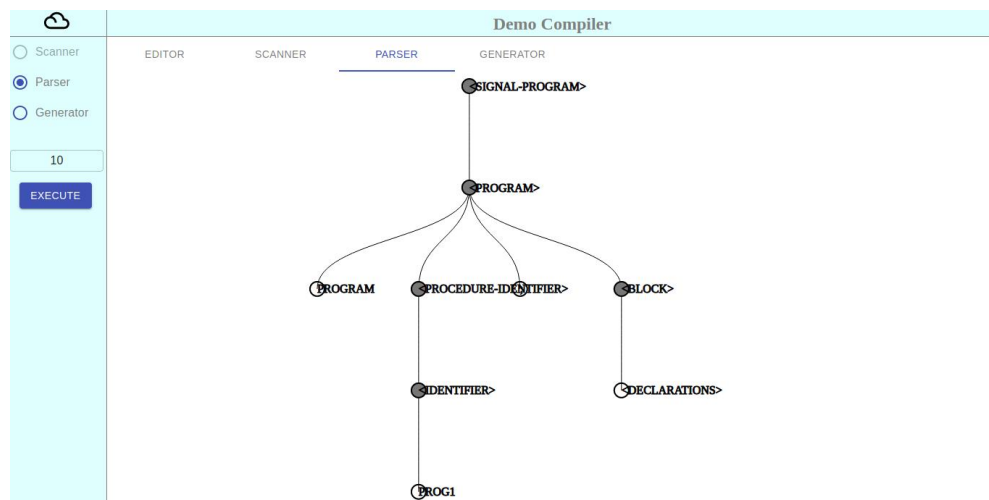


Рис.54 - Результат етапу parser

Кінцевий результат цього етапу зображено на рис.55.

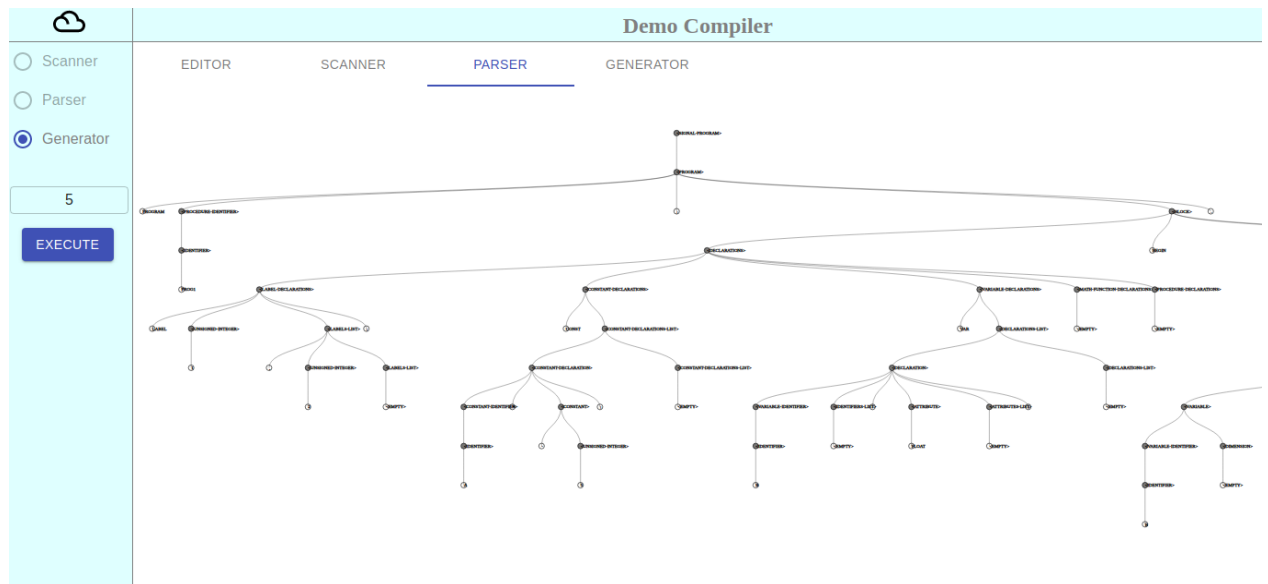


Рис.55 - Кінцевий результат етапу parser

Наступний етап – це generator. Зробивши 5 кроків, отримаємо результат зображений на рис.56.



Рис.56 - Результат роботи етапу generator

Пройшовши весь етап отримаємо результат зображений на рис.57.



Рис.57 - Кінцевий результат роботи етапу generator

Приклад вхідного коду номер 3 на якому тестувалася програма зображена на рис.58.

```
PROGRAM PROG1;
LABEL 1,2;
CONST A=-5;
VAR B:FLOAT;
BEGIN
GOTO 1;
RETURN;
END.
```

Рис.58- Вхідний програмний код номер 3 на мові SIGNAL

Після завантаження коду і збереження його на сервері, оберемо етап scanner та просунемося на 10 кроків. Отримаємо результат зображений на рис.59.

Demo Compiler

EDITOR SCANNER PARSER GENERATOR

Scanner
Parser
Generator

Current token: PR
Hypothetical type of current token: IDENTIFIER

Name	Type
PROGRAM	RESERVED
" "	WHITESPACE

10
EXECUTE

Рис.59 - Результат етапу scanner

Коли етап scanner завершиться користувач побачить повну таблицю лексем. Приклад такої таблиці зображено на рис.60.

Demo Compiler

EDITOR SCANNER PARSER GENERATOR

Scanner
Parser
Generator

Name	Type
.	DELIMITER
5	UNSIGNED-INTEGER
;	DELIMITER
" "	WHITESPACE
VAR	RESERVED
" "	WHITESPACE
B	IDENTIFIER
:	DELIMITER
FLOAT	RESERVED
:	DELIMITER
" "	WHITESPACE
BEGIN	RESERVED
" "	WHITESPACE
GOTO	RESERVED
" "	WHITESPACE
1	UNSIGNED-INTEGER
:	DELIMITER
" "	WHITESPACE
RETURN	RESERVED
:	DELIMITER
" "	WHITESPACE
END	RESERVED
.	DELIMITER

1
EXECUTE

Рис.60 - Кінцевий результат етапу scanner

Наступним етапом трансляції є parser. Просунувшись на 10 кроків матимемо результат зображений на рис.61.

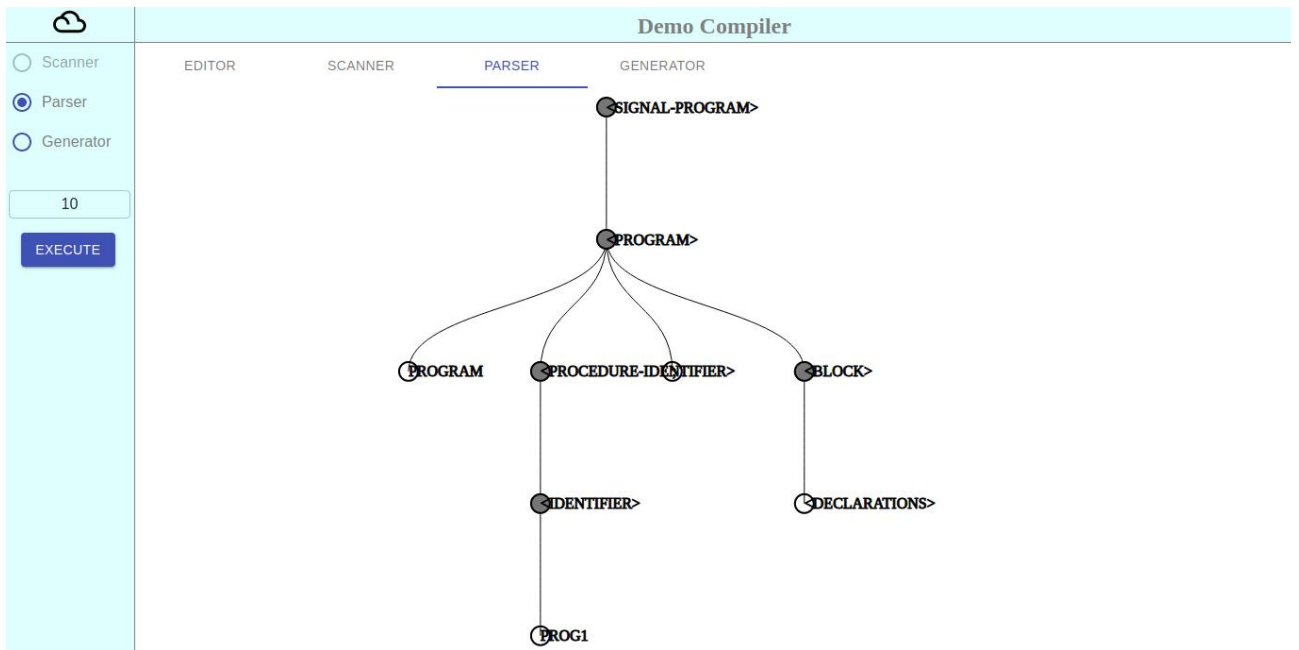


Рис.61 - Результат етапу parser

Кінцевий результат цього етапу зображено на рис.62.

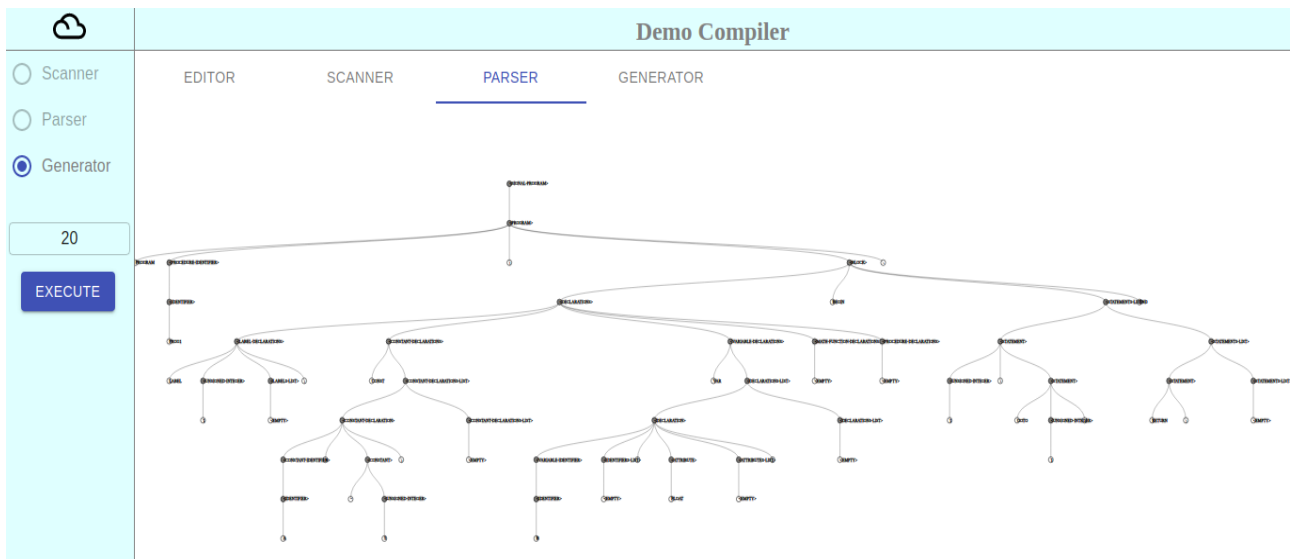


Рис.62 - Кінцевий результат етапу parser

Зм	Лист	№ докум.	Підп.	Дата
----	------	----------	-------	------

Наступний етап – це generator. Зробивши 5 кроків, отримаємо результат зображений на рис.63.

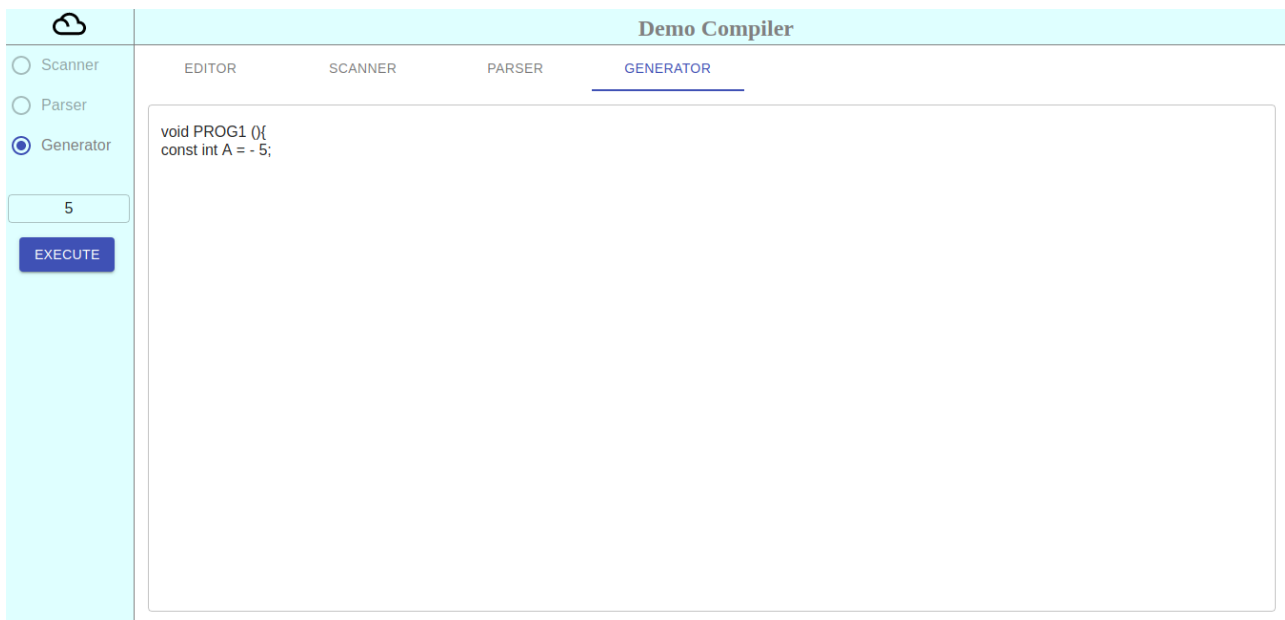


Рис.63 - Результат роботи етапу generator

Пройшовши весь етап отримаємо результат зображений на рис.64.

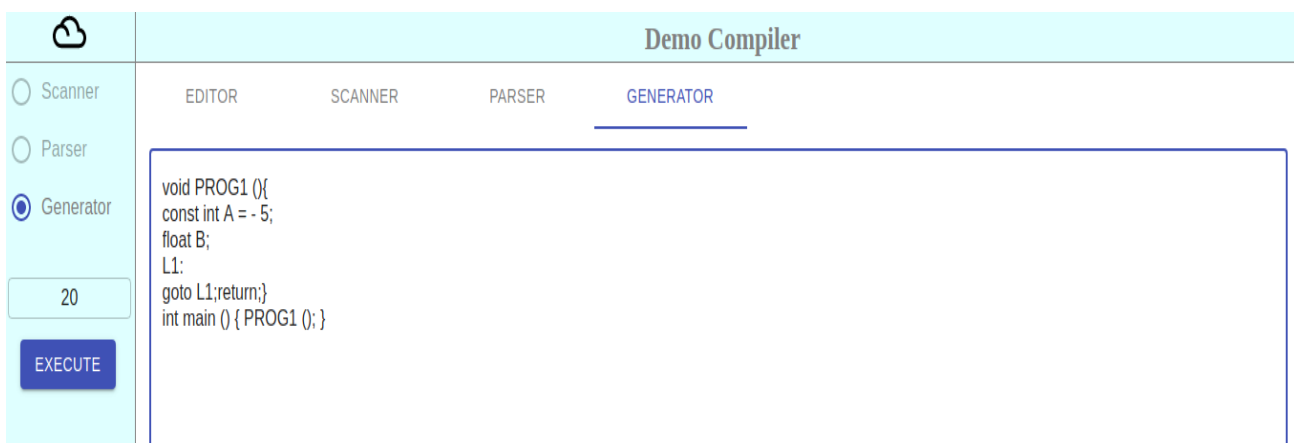


Рис.64 - Кінцевий результат роботи етапу generator

Також проводилася перевірка на коректність роботи вебдодатку при умові помилкового програмного коду. Приклад програмного коду з лексичною помилкою приведено на рис.65.

```
PROGRAM PROG1;
LABEL 1,2;
CONST A=-5;
VAR B:FLOAT;
BEGIN
B:=A;
IF B=5 THEN
RETURN;
ENDIF;
END.
(*
```

Рис.65 - Програмний код з лексичною помилкою

Представлення таблиці з вказаною помилкою приведено на рис.66.

The screenshot shows the 'Demo Compiler' interface. On the left, there is a sidebar with three radio buttons: 'Scanner' (selected), 'Parser', and 'Generator'. Below the sidebar is a text input field containing '25' and a blue 'EXECUTE' button. The main area is divided into five tabs: 'EDITOR', 'SCANNER', 'PARSER', 'GENERATOR', and 'ERROR' (which is active). The 'ERROR' tab displays a table with the following data:

Error	Stage	Row	Column
Unexpected end of file.	scanner	12	3

Рис.66 - Таблиця лексичних помилок

Як видно вище, лексичні помилки виявляються на етапі scanner. Помилки синтаксичні, в свою чергу, виявляються на етапі parser. Зразок програмного коду з синтаксичною помилкою зображено на рис.67. Приклад таблиці з синтаксичними помилками зображено на рис.68.

```
PROGRAM PROG1;
LABEL 1,2;
CONST A=-5;
VAR B:FLOAT
BEGIN
B:=A;
IF B=5 THEN
RETURN;
ENDIF;
END.
```

Рис.67 - Програмний код з синтаксичною помилкою

The screenshot shows a web-based 'Demo Compiler' interface. On the left, there are three radio buttons for 'Scanner', 'Parser', and 'Generator', with 'Parser' selected. Below them is a text input field containing '100' and a blue 'EXECUTE' button. The main area displays a table with the following content:

Demo Compiler				
EDITOR	SCANNER	PARSER	GENERATOR	ERROR
Error	Stage	Row	Column	
Expected ',', but got 'BEGIN'.	parser	5	1	

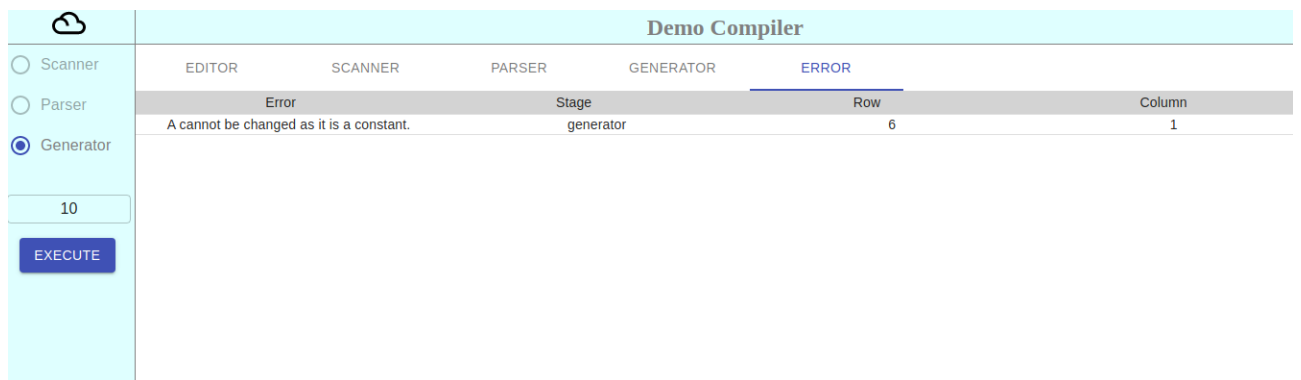
Рис.68 - Таблиця синтаксичних помилок

На етапі generator виявляються семантичні помилки. Програмний код для тестування зображений на рис.69. А приклад таблиці з семантичними

помилками на рис.70.

```
PROGRAM PROG1;  
LABEL 1,2;  
CONST A=-5;  
VAR B:FLOAT;  
BEGIN  
A:=B;  
B:=A;  
IF B=5 THEN  
RETURN;  
ENDIF;  
END.
```

Рис.69 - Програмний код з семантичною помилкою



The screenshot shows a web-based compiler interface titled "Demo Compiler". On the left, there is a sidebar with three radio buttons: "Scanner", "Parser", and "Generator", with "Generator" selected. Below the sidebar is a text input field containing "10" and a blue "EXECUTE" button. The main area is a table with columns for "EDITOR", "SCANNER", "PARSER", "GENERATOR", and "ERROR". The "ERROR" column is active, displaying a table with the following data:

Error	Stage	Row	Column
A cannot be changed as it is a constant.	generator	6	1

Рис.70 - Таблиця семантичних помилок

Висновок до 4-го розділу

Тестування вебдодатку включало в себе перевірку програми на трьох різних варіантах коду, кожен з яких мав індивідуальні конструкції мови SIGNAL. План тестування для всіх варіантів був однаковим і складався з: 10-ти кроків на етапі scanner, 10-ти кроків на етапі parser, 5-ти кроків на етапі generator, а також кінцеві результати кожного етапу трансляції. Всі етапи трансляції відображаються на екрані користувача, як показано на рисунках в даному розділі. Також програма тестувалася з використанням помилкового коду. Ці помилки були 3-х типів, а саме: лексичні, синтаксичні та семантичні. Про помилки в програмному кодї, якщо такі існують, користувач також повідомляється. Загалом, з врахуванням всіх проведених тестів, можна зробити висновок, що робота, як серверної так і клієнтської сторін вебдодатку є коректною.

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		66

ВИСНОВОК

Вебдодаток демо-компілятора, який є результатом дипломного проєкту, був створений з метою підтримки навчального процесу з предмету «Основи проєктування трансляторів». Цей вебдодаток допомагає студентам більш ґрунтовно розібратися з етапами трансляції вхідного коду. Крім того, користувачі можуть самостійно обирати потрібний етап трансляції, а також кількість кроків на цьому етапі. Такий підхід та надання подібних можливостей допомагають користувачу підлаштувати цей вебдодаток під свої потреби, тим самим зосередити свою увагу на тих етапах, які викликають труднощі в розумінні.

Проведений аналіз існуючих онлайн-компіляторів показав, що жоден з них не має такого функціоналу, як покрокове перетворення програмного коду на вхідній мові в програмний код на вихідній мові, а також, в жодному з онлайн-компіляторів, які аналізувалися, не представлена можливість вибору етапу трансляції. Всі ці онлайн-компілятори не демонструють внутрішню роботу транслятора при перетворенні програмного коду з вхідної мови у вихідну. Результатом їхньої роботи є текст програми на вихідній мові, але жоден з них не відображає проміжних результатів кожного етапу трансляції, тим самим не показуючи, як цей вихідний код було отримано.

При реалізації вебдодатку демо-компілятора основною задачею було зробити його нативним та інтуїтивно зрозумілим для користувача. Це важливо для вебдодатку такого типу, адже користувач не повинен витратити час на розуміння того, як використовувати цей демо-компілятор. Студенти, а саме на таку аудиторію розрахований вебдодаток, повинні зосередитися на розумінні самого предмету «Основи проєктування трансляторів» и не витратити час на

					ІАЛЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		67

розгляд того, як користуватися вебдодатком. До того ж, як показує статистика, недружні до користувача та ненативні вебдодатки не користуються популярністю. Тому для забезпечення цієї нативності було прийнято рішення використовувати бібліотеку стилів material-ui. Саме з її допомогою реалізовано велику кількість популярних вебдодатків та вебсайтів і тому інтерфейс виконаний зі стилями material-ui буде знайомий користувачу та інтуїтивно зрозумілий, що і було поставлено за мету при створенні даного демо-компілятора.

Загалом вебдодаток демо-компілятора дозволяє:

- наочно побачити результат роботи транслятора на етапі scanner
- наочно побачити результат роботи транслятора на етапі parser
- наочно побачити результат роботи транслятора на етапі generator
- вказати на лексичні, синтаксичні або семантичні помилки в коді, якщо такі існують.

В майбутньому розроблений вебдодаток може бути розширений, наприклад, додаванням різних вхідних та вихідних мов програмування.

					ІАЛІЦ.045440.004 ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		68

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Компілятор: URL: <https://uk.wikipedia.org/wiki/Компілятор>
(дата звернення: 20.03.2020).
2. Курдус А. О. Порівняльний аналіз онлайн-компіляторів. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2020. № 39. С. 141–145.
URL: <http://cit-journal.com.ua/index.php/cit/article/view/137>
(дата звернення: 19.05.2020).
3. JavaScript: URL: <https://uk.wikipedia.org/wiki/JavaScript>
(дата звернення: 26.03.2019).
4. React: URL: <https://uk.wikipedia.org/wiki/React>
(дата звернення: 01.04.2020).
5. Основи redux: URL: <https://rajdee.gitbooks.io/redux-in-russian/content/docs/basics/>
(дата звернення: 04.04.2020).
6. Lisp: URL: <https://uk.wikipedia.org/wiki/Lisp>
(дата звернення: 07.04.2020).
7. Hunchentoot: URL
(дата звернення: 10.04.2020).