Структура згідно методичних рекомендацій	**«Чекліст» компонентів, які обов'язково мають бути продемонстровані** (позначені як [_]), а опціональні — як [X] *(також усі блок-схеми алгоритмів)*	**Допоміжні матеріали (ВНС або назва теки на репозиторії), надані протягом семестру на практичних заняттях**
ТИТУЛЬНА СТОРІНКА		
ЗАВДАННЯ НА КУРСОВЕ ПРОЄКТУВАННЯ	«Скріншот» варіанту завдання (а) та текстовий опис (б): а[_], б[_].	варіанти завдань *(ВНС)*
АНОТАЦІЯ		
ЗМІСТ		
ВСТУП		
1. ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЄКТУВАННЯ ТРАНСЛЯТОРІВ		
2. ФОРМАЛЬНИЙ ОПИС ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ 2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура 2.2. Опис термінальних символів та ключових слів	EBNF: [_]. Код (boost::spirit), що <u>пострічково відповідає EBNF</u> та коректно опрацьовує тестові програми: [_].	EBNFVerify+GrammarVerify EBNFVerify+GrammarVerify verify_syntax_by_EBNF__2025
3. РОЗРОБКА ТРАНСЛЯТОРА З ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ		

3.1. Вибір технології програмування 3.2. Проектування таблиць транслятора та вибір структур даних		
3.3. Розробка лексичного аналізатора 3.3.1. Розробка блок-схеми алгоритму 3.3.2. Опис програми реалізації лексичного аналізатора	**Початковий етап:** Регулярний вираз для токенізатора: [_].	 lexica__2025 lexica_part_dfa__2025
	Регулярний вираз підмови ключових слів: [_].	lexica__2025 lexica_part_dfa__2025
	Регулярний вираз підмови ідентифікаторів: [_].	lexica__2025 lexica_part_dfa__2025
	Регулярний вираз підмови беззнакових літералів із фіксованою комою: [_].	lexica__2025 lexica_part_dfa__2025
	Код на основі регулярних виразів коректно опрацьовує тестові програми: [_].	lexica__2025 lexica_part_dfa__2025
	Безпосередній етап: Детальний опис процесу перетворення регулярних виразів у недетерміновані скінченні автомати (NFA): [X].	 (**KN1**/)sp2019_soft__12345/ addon__09_11_2024/1
	Детальний опис процесу видалення λ-переходів для отриманих NFA: [X].	(**KN1**/)sp2019_soft__12345/ addon__09_11_2024/2
	Детальний опис процесу перетворення NFA (після видалення λ-	

	переходів) у детермінований скінченний автомат (DFA): [X].	(**KN1**/)sp2019_soft__12345/ addon__09_11_2024/3
	Детальний опис процесу мінімізації *(ключовий етап)* DFA: [X].	(**KN1**/)sp2019_soft__12345/ addon__09_11_2024/4
	Результуючий етап:	
	Повністю детермінований скінченний автомат для токенізації (а: двовимірний масив для програмного моделювання; б: орієнтований граф з усіма переходами, окрім тих, що ведуть у «мертвий» стан, для графічного відображення): а[_], б[_].	dfa_generator__2025
	Модель токенізатора на основі детермінованого скінченного автомату, оскільки це нетривіальне завдання теорії автоматів-акцепторів: [_].	*(проаналізувати програмну модель у файлі matcher_by_dfa.hpp)* bult_src
	Повністю детермінований скінченний автомат для розпізнавання підмови ключових слів у вигляді таблиці переходів (а: двовимірний масив для програмного моделювання; б: орієнтований граф з усіма переходами, окрім тих, що ведуть у «мертвий» стан, для графічного відображення): а[_], б[_].	dfa_generator__2025
	Повністю детермінований скінченний автомат для розпізнавання підмови ідентифікаторів у вигляді таблиці переходів (а: двовимірний масив для програмного моделювання; б: орієнтований граф з усіма переходами, окрім тих, що ведуть у «мертвий» стан, для графічного відображення): а[_], б[_].	dfa_generator__2025

	Повністю детермінований скінченний автомат для розпізнавання підмови беззнакових літералів із фіксованою комою (а: двовимірний масив для програмного моделювання; б: орієнтований граф з усіма переходами, окрім тих, що ведуть у «мертвий» стан, для графічного відображення): а[_], б[_].	dfa_generator__2025
3.4. Розробка синтаксичного та семантичного аналізатора 3.4.1. Розробка дерев граматичного розбору 3.4.2. Розробка блок-схеми алгоритму 3.4.3. Опис програми реалізації синтаксичного та семантичного аналізатора	*Початковий етап посилається на попередній розділ (2.1), у якому формувалися EBNF та еквівалентний (пострічково) код для його перевірки.* **Безпосередній етап:** Синтез граматики відповідно до EBNF: [_].	EBNFVerify+GrammarVerify
	Синтез множин FIRST та FOLLOW для отриманої граматики: [X].	EBNFVerify+GrammarVerify
	Специфікація lookahead у правилах виведення граматики для моделі LL(2)-аналізатора: [_].	EBNFVerify+GrammarVerify
	Формування параметрів моделі LL(2)-аналізатора: [_].	EBNFVerify+GrammarVerify
	Результуючий етап: Формування моделі LL(2)-аналізатора у вигляді недетермінованого автомата з магазинною пам'яттю (NPDA) (а: формальний опис; б: програмна модель): а[_], б[_].	dpda1_for_ll2_generator__2025
	Формування моделі LL(2)-аналізатора у вигляді детермінованого	

	автомата з магазинною пам'яттю (DPDA): [X].	dpda1_for_ll2_generator__2025
3.5. Розробка генератора коду 3.5.1. Розробка блок-схеми алгоритму 3.5.2. Опис програми реалізації генератора коду	У цьому розділі виконується опис: алгоритму формування польського інверсного запису [_], алгоритму виконання польського інверсного запису з використанням стеку [_], *Сам код генерації універсальний. Додатково можна описати процес формування SSA та інші аспекти [X].*	cw_sp2__2025_2026/ src/implementation/preparer cw_sp2__2025_2026/ src/implementation/generator 28_11_2025 *(+інші теки)*
4. НАЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА 4.1. Опис інтерфейсу та інструкції користувачу 4.2. Виявлення лексичних і синтаксичних помилок 4.3. Перевірка роботи транслятора за допомогою тестових задач	Описати консольний інтерфейс компілятора [_] Три тестові програми згідно варіанту [_] Коректна компіляція трьох тестових програм* [_] **у записці наводиться тестування лише трьох заданих тестових програм, але під час захисту потрібно вміти писати вашою мовою програмування будь-яку просту тестову програму*	cw_sp2__2025_2026/ src/implementation/cli base_test_programs_2025 cw_sp2__2025_2026 (запуск)
ВИСНОВКИ СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ ДОДАТКИ А. Таблиці лексем для тестових прикладів Б. С код (або код на асемблері), отриманий на виході транслятора для тестових прикладів;	Оскільки це курсовий проект (а не курсова робота), то окремо у форматі А3 (що має бути оформлено як стандартне креслення) подається один графічний матеріал. Це може бути приміром	

Б. Документований текст програмних модулів (лістинги)	загальна блок-схема роботи транслятора: [_]	

Безпосередньо захист курсового проекту відбуватиметься під час сесії, починаючи з 05.01.2026 (буде додаткове повідомлення з точними датами, з урахуванням розкладу ваших іспитів).

Перед захистом (орієнтовно 02.01.2026–03.01.2026) у ВНС, у курсі «Системне програмування (курсовий проєкт)», в Оцінювання курсового проекту необхідно завантажити два з трьох елементів:

1) «Програмний проєкт» — потрібно очистити проект від зайвих файлів і тек: видалити теку .vs (прихована тека — потрібно увімкнути відображення прихованих елементів файлової системи у налаштуваннях операційної системи), теку Release (у двох місцях), теку Debug (у двох місцях), а також для 64-розрядної платформи — теку x64 (у двох місцях).

2) «Пояснювальна записка» — завантажити pdf-файл пояснювальної записки. Хоча лістинг програмного проекту подається в окремому розділі, і він, і графічна частина («креслення») мають також бути включені до файлу пояснювальної записки разом з усіма іншими додатками відповідно до змісту.
Додатково *(про що згадувалося у таблиці вище)* сюди потрібно завантажити pdf-файл графічної частини («креслення») формату А3 або А2 із загальною блок-схемою алгоритму функціонування розробленого транслятора. Креслення повинно містити штамп згідно з шаблонами, розміщеними у " Шаблони для оформлення пояснювальної записки і креслення " .

Під час захисту потрібно буде завантажити третій пункт:

3) «Захист» — завантажити окремий звіт, що демонструє роботу компілятора для довільних (заданих під час захисту) простих тестових програм вашою мовою програмування, які будуть відрізнятися від трьох базових тестових програм, наведених у пояснювальній записці *(про що також згадувалося у таблиці вище)*. Звіт оформлюється у довільній формі та має містити самі тестові програми, результати їх обробки вашим компілятором і скріншоти виконання скомпільованих програм.