

Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Кафедра ЕОМ



Пояснювальна записка

До курсового проєкту «СИСТЕМНЕ ПРОГРАМУВАННЯ»

на тему: “РОЗРОБКА СИСТЕМНИХ ПРОГРАМНИХ МОДУЛІВ ТА
КОМПОНЕНТ СИСТЕМ ПРОГРАМУВАННЯ”

Індивідуальне завдання

“РОЗРОБКА ТРАНСЛЯТОРА З ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ”

Варіант №26

Виконав:
ст. гр. КІ-307
Соніч О.В.
Перевірив:
Козак Н. Б.

Львів-2024

ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ

Варіант 26

Завдання на курсовий проект

1. Цільова мова транслятора – асемблер для 32-розрядного процесора.
2. Для отримання виконавчого файлу на виході розробленого транслятора скористатися програмами ml.exe і link.exe.
3. Мова розробки транслятора: C++.
4. Реалізувати оболонку або інтерфейс з командного рядка.
5. На вхід розробленого транслятора має подаватися текстовий файл, написаний на заданій мові програмування.
6. На виході розробленого транслятора мають створюватись такі файли:
 - *файл з лексемами;*
 - *файл з повідомленнями про помилки (або про їх відсутність);*
 - *файл на мові асемблера;*
 - *об'єктний файл;*
 - *виконавчий файл.*
7. Назва вхідної мови програмування утворюється від першої букви у прізвищі студента та останніх двох цифр номера його варіанту. Саме таке розширення повинні мати текстові файли, написані на цій мові програмування.

Деталізація завдання на проектування:

1. В кожному завданні передбачається блок оголошення змінних; змінні зберігають значення цілих чисел і, в залежності від варіанту, можуть бути 16/32 розрядними. За потребою можна реалізувати логічний тип даних.
2. Необхідно реалізувати арифметичні операції – додавання, віднімання, множення, ділення, залишок від ділення; операції порівняння – перевірка на рівність і нерівність, більше і менше; логічні операції – заперечення, “логічне І” і “логічне АБО”.

Пріоритет операцій наступний – круглі дужки (), логічне заперечення, мультиплікативні (множення, ділення, залишок від ділення), адитивні (додавання, віднімання), відношення (більше, менше), перевірка на рівність і нерівність, логічне І, логічне АБО.

3. За допомогою оператора вводу можна зчитати з клавіатури значення змінної; за допомогою оператора виводу можна вивести на екран значення змінної, виразу чи цілої константи.
4. В кожному завданні обов'язковим є оператор присвоєння за допомогою якого можна реалізувати обчислення виразів з використанням заданих операцій і операції круглі дужки (); у якості операндів можуть бути цілі константи, змінні, а також інші вирази.
5. В кожному завданні обов'язковим є оператор типу “блок” (складений оператор), його вигляд має бути таким, як і блок тіла програми.
6. Необхідно реалізувати задані варіантом оператори, синтаксис операторів наведено у таблиці 1.1. Синтаксис вхідної мови має забезпечити реалізацію обчислень лінійних алгоритмів, алгоритмів з розгалуженням і циклічних алгоритмів. Опис формальної мови студент погоджує з викладачем.
7. Оператори можуть бути довільної вкладеності і в будь-якій послідовності.
8. Для перевірки роботи розробленого транслятора, необхідно написати три тестові програми на вхідній мові програмування.

Деталізований опис власної мови програмування:

Розширення файлу - .s26

Опис вхідної мови програмування:

- Тип даних: INTEGER_2
- Блок тіла програми: NAME <name>; BODY DATA...; END
- Оператор вводу: SCAN ()
- Оператор виводу: PRINT ()
- Оператори: IF ELSE (C)
GOTO (C)
FOR-TO-DO (Паскаль)
FOR-DOWNTO-DO (Паскаль)
WHILE (Бейсік)
REPEAT-UNTIL (Паскаль)
- Регістр ключових слів: Up
- Регістр ідентифікаторів: Low-Up6 перший символ _
- Операції арифметичні: ADD, SUB, MUL, DIV, MOD
- Операції порівняння: EQ, NE, >=, <=
- Операції логічні: NOT, AND, OR

- Коментар: !!...
- Ідентифікатори змінних, числові константи
- Оператор присвоєння: <-

Для отримання виконавчого файлу на виході розробленого транслятора скористатися програмами ml.exe (компілятор мови асемблера) і link.exe (редактор зв'язків).

АНОТАЦІЯ

Цей курсовий проект приводить до розробки транслятора, який здатен конвертувати вхідну мову, визначену відповідно до варіанту, у мову асемблера. Процес трансляції включає в себе лексичний аналіз, синтаксичний аналіз та генерацію коду.

Лексичний аналіз розбиває вхідну послідовність символів на лексеми, які записуються у відповідну таблицю лексем. Кожній лексемі присвоюється числове значення для полегшення порівнянь, а також зберігається додаткова інформація, така як номер рядка, значення (якщо тип лексеми є числом) та інші деталі.

Синтаксичний аналіз: використовується висхідний метод аналізу без повернення. Призначений для побудови дерева розбору, послідовно рухаючись від листків вгору до кореня дерева розбору.

Генерація коду включає повторне прочитання таблиці лексем та створення відповідного асемблерного коду для кожного блоку лексем. Отриманий код записується у результуючий файл, готовий для виконання.

Отриманий після трансляції код можна скомпілювати за допомогою відповідних програм (наприклад, LINK, ML і т. д.).

Зміст

ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ	2
АНОТАЦІЯ	5
ВСТУП	7
1. ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЕКТУВАННЯ ТРАНСЛЯТОРІВ	8
2. ФОРМАЛЬНИЙ ОПИС ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ	11
2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура	11
2.2. Опис термінальних символів та ключових слів.....	13
3. РОЗРОБКА ТРАНСЛЯТОРА ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ.....	15
3.1. Вибір технології програмування.....	15
3.2. Проектування таблиць транслятора.....	16
3.3. Розробка лексичного аналізатора.....	18
3.3.1. Розробка алгоритму роботи лексичного аналізатора	19
3.3.2. Опис програми реалізації лексичного аналізатора	19
3.4. Розробка синтаксичного та семантичного аналізатора	21
3.4.1. Розробка дерева граматичного розбору.	22
3.4.2. Опис програми реалізації синтаксичного та семантичного аналізатора	22
3.4.3. Розробка граф-схеми алгоритму	23
3.5. Розробка генератора коду	23
3.5.1. Розробка алгоритму роботи генератора коду	25
3.5.2. Опис програми реалізації генератора коду.....	26
4. ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА.....	30
4.1. Опис інтерфейсу та інструкція користувачеві.....	30
4.2. Виявлення лексичних та синтаксичних помилок	31
4.3. Перевірка роботи транслятора за допомогою тестових задач.....	32
4.4. Тестова програма №1.....	34
4.5. Тестова програма №2.....	35
4.6. Тестова програма №3.....	36
ВИСНОВКИ	38
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	39
ДОДАТКИ	40

ВСТУП

Термін "транслятор" визначає програму, яка виконує переклад (трансляцію) початкової програми, написаної на вхідній мові, у еквівалентну їй об'єктну програму. У випадку, коли мова високого рівня є вхідною, а мова асемблера або машинна – вихідною, такий транслятор отримує назву компілятора.

Транслятори можуть бути розділені на два основних типи: компілятори та інтерпретатори. Процес компіляції включає дві основні фази: аналіз та синтез. Під час аналізу вхідну програму розбивають на окремі елементи (лексеми), перевіряють її відповідність граматичним правилам і створюють проміжне представлення програми. На етапі синтезу з проміжного представлення формується програма в машинних кодах, яку називають об'єктною програмою. Останню можна виконати на комп'ютері без додаткової трансляції.

У відмінну від компіляторів, інтерпретатор не створює нову програму; він лише виконує – інтерпретує – кожну інструкцію вхідної мови програмування. Подібно компілятору, інтерпретатор аналізує вхідну програму, створює проміжне представлення, але не формує об'єктну програму, а негайно виконує команди, передбачені вхідною програмою.

Компілятор виконує переклад програми з однієї мови програмування в іншу. На вхід компілятора надходить ланцюг символів, який представляє вхідну програму на певній мові програмування. На виході компілятора (об'єктна програма) також представляє собою ланцюг символів, що вже відповідає іншій мові програмування, наприклад, машинній мові конкретного комп'ютера. При цьому сам компілятор може бути написаний на третій мові.

1. ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЕКТУВАННЯ ТРАНСЛЯТОРІВ

Термін "транслятор" визначає обслуговуючу програму, що проводить трансляцію вихідної програми, представленої на вхідній мові програмування, у робочу програму, яка відображена на об'єктній мові. Наведене визначення застосовне до різноманітних трансляторів програм. Однак кожна з таких програм може виявляти свої особливості в організації процесу трансляції. В сучасному контексті транслятори поділяються на три основні групи: асемблери, компілятори та інтерпретатори.

Асемблер - це системна обслуговуюча програма, яка перетворює символічні конструкції в команди машинної мови. Типовою особливістю асемблерів є дослівна трансляція однієї символічної команди в одну машинну.

Компілятор - обслуговуюча програма, яка виконує трансляцію програми, написаної мовою оригіналу програмування, в машинну мову. Схоже до асемблера, компілятор виконує перетворення програми з однієї мови в іншу, найчастіше - у мову конкретного комп'ютера.

Інтерпретатор - це програма чи пристрій, що виконує пооператорну трансляцію та виконання вихідної програми. Відмінно від компілятора, інтерпретатор не створює на виході програму на машинній мові. Розпізнавши команду вихідної мови, він негайно її виконує, забезпечуючи більшу гнучкість у процесі розробки та налагодження програм.

Процес трансляції включає фази лексичного аналізу, синтаксичного та семантичного аналізу, оптимізації коду та генерації коду. Лексичний аналіз розбиває вхідну програму на лексеми, що представляють слова відповідно до визначень мови. Синтаксичний аналіз визначає структуру програми, створюючи синтаксичне дерево. Семантичний аналіз виявляє залежності між частинами програми, недосяжні контекстно-вільним синтаксисом. Оптимізація коду та генерація коду спрямовані на оптимізацію та створення машинно-залежного коду відповідно.

Зазначені фази можуть об'єднуватися або відсутні у трансляторах в залежності від їхньої реалізації. Наприклад, у простих однопрохідних трансляторах може відсутні фаза генерації проміжного представлення та оптимізації, а інші фази можуть об'єднуватися.

Під час процесу виділення лексем лексичний аналізатор може виконувати дві основні функції: автоматично побудову таблиць об'єктів (таких як ідентифікатори, рядки, числа і т. д.) і видачу значень для кожної лексеми при кожному новому зверненні до нього. У цьому контексті таблиці об'єктів формуються в подальших етапах, наприклад, під час синтаксичного аналізу.

На етапі лексичного аналізу виявляються деякі прості помилки, такі як неприпустимі символи або невірний формат чисел та ідентифікаторів.

Основним завданням синтаксичного аналізу є розбір структури програми. Зазвичай під структурою розуміється дерево, яке відповідає розбору в контекстно-вільній граматиці мови програмування. У сучасній практиці найчастіше використовуються методи аналізу, такі як LL (1) або LR (1) та їхні варіанти (рекурсивний спуск для LL (1) або LR (1), LR (0), SLR (1), LALR (1) та інші для LR (1)). Рекурсивний спуск застосовується частіше при ручному програмуванні синтаксичного аналізатора, тоді як LR (1) використовується при автоматичній генерації синтаксичних аналізаторів.

Результатом синтаксичного аналізу є синтаксичне дерево з посиланнями на таблиці об'єктів. Під час синтаксичного аналізу також виявляються помилки, пов'язані зі структурою програми.

На етапі контекстного аналізу виявляються взаємозалежності між різними частинами програми, які не можуть бути адекватно описані за допомогою контекстно-вільної граматики. Ці взаємозалежності, зокрема, включають аналіз типів об'єктів, областей видимості, відповідності параметрів, міток та інших аспектів "опис-використання". У ході контекстного аналізу таблиці об'єктів доповнюються інформацією, пов'язаною з описами (властивостями) об'єктів.

В основі контекстного аналізу лежить апарат атрибутних граматики. Результатом цього аналізу є створення атрибутованого дерева програми, де інформація про об'єкти може бути розсіяна в самому дереві чи сконцентрована в окремих таблицях об'єктів. Під час контекстного аналізу також можуть бути виявлені помилки, пов'язані з неправильним використанням об'єктів.

Після завершення контекстного аналізу програма може бути перетворена во внутрішнє представлення. Це здійснюється з метою оптимізації та/або для полегшення генерації коду. Крім того, перетворення програми у внутрішнє представлення може бути використано для створення переносимого компілятора. У цьому випадку, тільки остання фаза (генерація коду) є залежною від конкретної архітектури. В якості внутрішнього представлення може використовуватися префіксний або постфіксний запис, орієнтований граф, трійки, четвірки та інші формати.

Фаза оптимізації транслятора може включати декілька етапів, які спрямовані на покращення якості та ефективності згенерованого коду. Ці оптимізації часто розподіляються за двома головними критеріями: машинно-залежні та машинно-незалежні, а також локальні та глобальні.

Машинно-залежні оптимізації, як правило, проводяться на етапі генерації коду, і вони орієнтовані на конкретну архітектуру машини. Ці оптимізації можуть включати розподіл регістрів, вибір довгих або коротких переходів та оптимізацію вартості команд для конкретних послідовностей команд.

Глобальна оптимізація спрямована на поліпшення ефективності всієї програми і базується на глобальному потоковому аналізі, який виконується на графі програми. Цей аналіз враховує властивості програми, такі як межпроцедурний аналіз, міжмодульний аналіз та аналіз галузей життя змінних.

Фінальна фаза трансляції - генерація коду, результатом якої є або асемблерний модуль, або об'єктний (або завантажувальний) модуль. На цьому етапі можуть застосовуватися деякі локальні оптимізації для полегшення генерації вартісного та ефективного коду.

Важливо відзначити, що фази транслятора можуть бути відсутніми або об'єднаними в залежності від конкретної реалізації. В простіших випадках, таких як у випадку однопроходових трансляторів, може відсутній окремий етап генерації проміжного представлення та оптимізації, а інші фази можуть бути об'єднані в одну, при цьому не створюється явно побудованого синтаксичного дерева.

2. ФОРМАЛЬНИЙ ОПИС ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ

2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура

Однією з перших задач, що виникають при побудові компілятора, є визначення вхідної мови програмування. Для цього використовують різні способи формального опису, серед яких я застосував розширену нотацію Бекуса-Наура (extended Backus/Naur Form - EBNF).

```
topRule = "NAME", identifier, ";", "BODY", varsBlok, ";", operators, "END";
varsBlok = "DATA", "INTEGER_2", identifier, [{ commaAndIdentifier }];
identifier = "_", low_letter, { up_letter | number } {6};
commaAndIdentifier = ",", identifier;
codeBlok = "BODY", write | read | assignment | ifStatement | goto_statement |
labelRule | forToOrDownToDoRule | while | repeatUntil, "END";
operators = write | read | assignment | ifStatement | goto_statement | labelRule
| forToOrDownToDoRule | while | repeatUntil;
read = "SCAN", "(", identifier, ")";
write = "PRINT", "(", equation | stringRule, ")";
assignment = identifier, "<=", equation;
ifStatement = "IF", "(", equation, ")", codeBlok, ["ELSE", codeBlok];
goto_statement = "GOTO", ident ;
labelRule = identifier, ":";
forToOrDownToDoRule = "FOR", assignment, "TO" | "DOWNT", equation,
"Do", codeBlok;
while = "WHILE", "(", equation, ")", "BODY", operators | whileContinue |
whileExit, "END", "WHILE";
whileContinue = "CONTINUE", "WHILE";
whileExit = "EXIT", "WHILE";
repeatUntil = "REPEAT", operators, "UNTIL", "(", equation, ")";
equation = signedNumber | identifier | notRule [{ operationAndIdentOrNumber
| equation }];
notRule = notOperation, signedNumber | identifier | equation;
```

```

operationAndIdentOrNumber = mult | arithmetic | logic | compare
signedNumber | identifier | equation;

arithmetic = "ADD" | "SUB";

mult = "MUL" | "DIV" | "MOD";

logic = "AND" | "OR";

notOperation = "NOT";

compare = "EQ" | "NE" | "<=" | ">=";

stringRule = "\"", string, "\"";

comment = "LComment" string ;

LComment = "!!";

string = { low_letter | up_letter | number };

signedNumber    = [ sign ] digit [ {digit} ];

sign = "+" | "-";

low_letter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" |
"p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z";

up_letter = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N"
| "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z";

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

```

2.2 Опис термінальних символів та ключових слів

Визначимо окремі термінальні символи та нерозривні набори термінальних символів (ключові слова):

Термінальний символ або ключове слово	Значення
NAME	Початок програми
BODY	Початок тексту програми
DATA	Початок блоку опису змінних
END	Кінець розділу операторів
SCAN	Оператор вводу змінних
PRINT	Оператор виводу (змінних або рядкових констант)
<-	Оператор присвоєння
IF	Оператор умови
ELSE	Оператор умови
GOTO	Оператор переходу
LABEL	Мітка переходу
FOR	Оператор циклу
TO	Інкремент циклу
DOWNT0	Декремент циклу
DO	Початок тіла циклу
WHILE	Оператор циклу
CONTINUE	Оператор циклу
EXIT	Оператор циклу
REPEAT	Початок тіла циклу
UNTIL	Оператор циклу
ADD	Оператор додавання
SUB	Оператор віднімання

MUL	Оператор множення
DIV	Оператор ділення
MOD	Оператор знаходження залишку від ділення
EQ	Оператор перевірки на рівність
NE	Оператор перевірки на нерівність
<=	Оператор перевірки чи менше
>=	Оператор перевірки чи більше
NOT	Оператор логічного заперечення
AND	Оператор кон'юнкції
OR	Оператор диз'юнкції
INTEGER_2	16-ти розрядні знакові цілі
!!...	Коментар
,	Розділювач
;	Ознака кінця оператора
(Відкриваюча дужка
)	Закриваюча дужка

До термінальних символів віднесемо також усі цифри (0-9), латинські букви (a-z, A-Z), символи табуляції, символ переходу на нову стрічку, пробілу.

3. РОЗРОБКА ТРАНСЛЯТОРА ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ

3.1 Вибір технології програмування

Для ефективної роботи створюваної програми важливу роль відіграє попереднє складення алгоритму роботи програми, алгоритму написання програми і вибір технології програмування.

Тому при складанні транслятора треба брати до уваги швидкість компіляції, якість об'єктної програми. Проект повинен давати можливість просто вносити зміни.

В реалізації мов високого рівня часто використовується специфічний тільки для компіляції засіб “розкрутки”. З кожним транслятором завжди зв'язані три мови програмування: X – початкова, Y – об'єктна та Z – інструментальна. Транслятор перекладає програми мовою X в програми, складені мовою Y , при цьому сам транслятор є програмою написаною мовою Z .

При розробці даного курсового проекту був використаний висхідний метод синтаксичного аналізу.

Також був обраний прямий метод лексичного аналізу. Характерною ознакою цього методу є те, що його реалізація відбувається без повернення назад. Його можна сприймати, як один спільний скінченний автомат. Такий автомат на кожному кроці читає один вхідний символ і переходить у наступний стан, що наближає його до розпізнавання поточної лексеми чи формування інформації про помилки. Для лексем, що мають однакові підланцюжки, автомат має спільні фрагменти, що реалізують єдину множину станів. Частини, що відрізняються, реалізуються своїми фрагментами

3.2 Проектування таблиць транслятора

Використання таблиць значно полегшує створення трансляторів, тому у даному випадку використовуються наступне:

- 1) Мульти мапа для лексеми, значення та рядка кожного токена.

```
std::multimap<int, std::shared_ptr<IToken>> m_priorityTokens;  
  
std::string m_lexeme; //Лексема  
std::string m_value;  //Значення  
int m_line = -1;      //Рядок
```

- 2) Таблиця лексичних класів

Якщо у стовпці «Значення» відсутня інформація про токен, то це означає що його значення визначається користувачем під час написання коду на створеній мові програмування.

Таблиця 2 Опис термінальних символі та ключових слів

Токен	Значення
Program	NAME
Start	BODY
Vars	DATA
End	END
VarType	INTEGER_2
Read	SCAN
Write	PRINT
Assignment	<-
If	IF
Else	ELSE
Goto	GOTO
Colon	:
Label	
For	FOR
To	TO
DownTo	DOWNT0
Do	DO

While	WHILE
ContinueWhile	CONTINUE
ExitWhile	EXIT
Repeat	REPEAT
Until	UNTIL
Addition	ADD
Subtraction	SUB
Multiplication	MUL
Division	DIV
Mod	MOD
Equal	EQ
NotEqual	NE
Less	<=
Greate	>=
Not	NOT
And	AND
Or	OR
Plus	+
Minus	-
Identifier	
Number	
String	
Undefined	
Unknown	
Comma	,
Quotes	“
Semicolon	;
Lbraket	(
Rbraket)
LComment	!!
Comment	

3.3 Розробка лексичного аналізатора

На фазі лексичного аналізу вхідна програма, що представляє собою потік літер, розбивається на лексеми - слова у відповідності з визначеннями мови. Лексичний аналізатор може працювати в двох основних режимах: або як підпрограма, що викликається синтаксичним аналізатором для отримання чергової лексеми, або як повний прохід, результатом якого є файл лексем.

Для нашої програми виберемо другий варіант. Тобто, спочатку буде виконуватись фаза лексичного аналізу. Результатом цієї фази буде файл з списком лексем. Але лексеми записуються у файл не як послідовність символів. Кожній лексемі присвоюється певний символ, тип, значення та рядок. Ці дані далі записуються у файл. Такий підхід дозволяє спростити роботу синтаксичного аналізатора.

Також на етапі лексичного аналізу виявляються деякі (найпростіші) помилки (неприпустимі символи, неправильний запис чисел, ідентифікаторів та ін.)

На вхід лексичного аналізатора надходить текст вихідної програми, а вихідна інформація передається для подальшої обробки компілятором на етапі синтаксичного аналізу.

Існує кілька причин, з яких до складу практично всіх компіляторів включають лексичний аналіз:

- застосування лексичного аналізатора спрощує роботу з текстом вихідної програми на етапі синтаксичного розбору;
- для виділення в тексті та розбору лексем можливо застосовувати просту, ефективну і теоретично добре пророблену техніку аналізу;

3.3.1 Розробка алгоритму роботи лексичного аналізатора

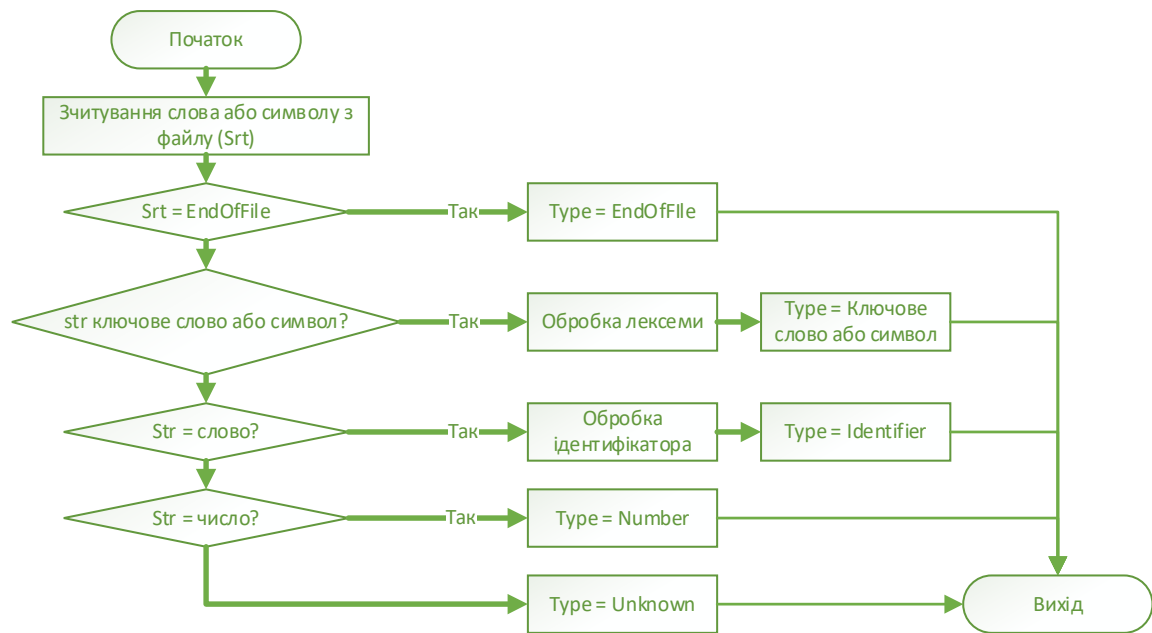


Рис. 3.1 Блок-схема роботи лексичного аналізатора

3.3.2 Опис програми реалізації лексичного аналізатора

Основна задача лексичного аналізу – розбити вихідний текст, що складається з послідовності одиночних символів, на послідовність слів, або лексем, тобто виділити ці слова з безперервної послідовності символів. Всі символи вхідної послідовності з цієї точки зору розділяються на символи, що належать яким-небудь лексемам, і символи, що розділяють лексеми. В цьому випадку використовуються звичайні засоби обробки рядків. Вхідна програма проглядається послідовно з початку до кінця. Базові елементи, або лексичні одиниці, розділяються пробілами, знаками операцій і спеціальними символами (новий рядок, знак табуляції), і таким чином виділяються та розпізнаються ідентифікатори, літерали і термінальні символи (операції, ключові слова).

Програма аналізує файл поки не досягне його кінця. Для вхідного файлу викликається функція `tokenize()`. Вона зчитує з файлу його вміст та кожну лексему порівнює з зарезервованими словами якщо є співпадіння то присвоює лексемі відповідний тип або значення, якщо це числова константа.

При виділенні лексеми вона розпізнається та записується у список `m_tokens` за допомогою відповідного типу лексеми, що є унікальним для кожної лексеми із усього можливого їх набору. Це дає можливість наступним фазам компіляції звертатись до лексеми не як до послідовності символів, а як до унікального типу лексеми, що значно спрощує роботу синтаксичного аналізатора: легко

перевіряти належність лексеми до відповідної синтаксичної конструкції та є можливість легкого перегляду програми, як вгору, так і вниз, від поточної позиції аналізу. Також в таблиці лексем ведуться записи, щодо рядка відповідної лексеми – для місця помилки – та додаткова інформація.

При лексичному аналізі виявляються і відзначаються лексичні помилки (наприклад, недопустимі символи і неправильні ідентифікатори). Лексична фаза відкидає також коментарі та символи лапок у конструкції String, оскільки вони не мають ніякого впливу на виконання програми, отже й на синтаксичний розбір та генерацію коду.

В даному курсовому проекті реалізовано прямий лексичний аналізатор, який виділяє з вхідного тексту програми окремі лексеми і на основі цього формує таблицю.

3.4 Розробка синтаксичного та семантичного аналізатора

Синтаксичний аналізатор - частина компілятора, яка відповідає за виявлення основних синтаксичних конструкцій вхідної мови. У завдання синтаксичного аналізатора входить: знайти і виділити основні синтаксичні конструкції в тексті вхідної програми, встановити тип і перевірити правильність кожної синтаксичної конструкції у вигляді, зручному для подальшої генерації тексту результуючої програми.

В основі синтаксичного аналізатора лежить Розпізнавач тексту вхідної програми на основі граматики вхідного мови. Як правило, синтаксичні конструкції мов програмування можуть бути описані за допомогою КС-грамматик, рідше зустрічаються мови, які можуть бути описані за допомогою регулярних граматик. Найчастіше регулярні граматики застосовні до мов асемблера, а мови високого рівня побудовані на основі КС-мов.

Синтаксичний розбір - це основна частина компіляції на етапі аналізу. Без виконання синтаксичного розбору робота компілятора безглузда, у той час як лексичний аналізатор є зовсім необов'язковим. Усі завдання з перевірки лексики вхідного мови можуть бути вирішені на етапі синтаксичного розбору. Сканер тільки дозволяє позбавити складний за структурою лексичний аналізатор від рішення примітивних завдань з виявлення та запам'ятовування лексем вхідної програми.

В даному курсовому проекті синтаксичний аналіз можна виконувати лише після виконання лексичного аналізу, він являється окремим етапом трансляції.

На вході даного аналізатора є файл лексем, який є результатом виконання лексичного аналізу, на базі цього файлу синтаксичний аналізатор формує таблицю ідентифікаторів та змінних.

3.4.3 Розробка граф-схеми алгоритму

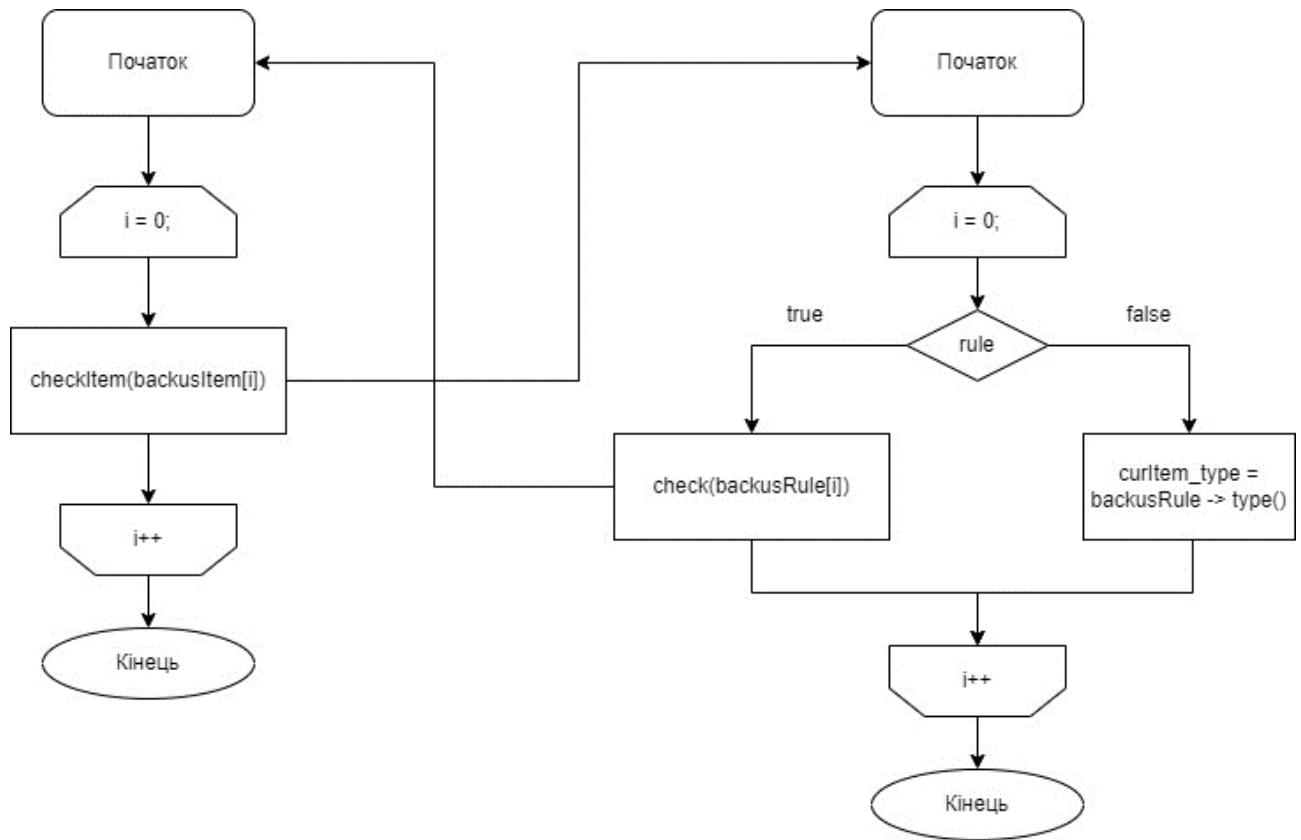


Рис. 3.3 Граф-схема роботи синтаксичного аналізатора

3.5 Розробка генератора коду

Синтаксичне дерево в чистому вигляді несе тільки інформацію про структуру програми. Насправді в процесі генерації коду потрібна також інформація про змінні (наприклад, їх адреси), процедури (також адреси, рівні), мітки і т.д. Для представлення цієї інформації можливі різні рішення. Найбільш поширені два:

- інформація зберігається у таблицях генератора коду;
- інформація зберігається у відповідних вершинах дерева.

Розглянемо, наприклад, структуру таблиць, які можуть бути використані в поєднанні з Лідер-представленням. Оскільки Лідер-представлення не містить інформації про адреси змінних, значить, цю інформацію потрібно формувати в процесі обробки оголошень і зберігати в таблицях. Це стосується і описів масивів, записів і т.д. Крім того, в таблицях також повинна міститися інформація про процедури (адреси, рівні, модулі, в яких процедури описані, і т.д.). При вході в процедуру в таблиці рівнів процедур заводиться новий вхід -

вказівник на таблицю описів. При виході вказівник поновлюється на старе значення. Якщо проміжне представлення - дерево, то інформація може зберігатися в вершинах самого дерева.

Генерація коду – це машинно-залежний етап компіляції, під час якого відбувається побудова машинного еквівалента вхідної програми. Зазвичай входом для генератора коду служить проміжна форма представлення програми, а на виході може з'являтися об'єктний код або модуль завантаження.

Генератор асемблерного коду приймає масив лексем без помилок. Якщо на двох попередніх етапах виявлено помилки, то ця фаза не виконується.

В даному курсовому проекті генерація коду реалізується як окремий етап. Можливість його виконання є лише за умови, що попередньо успішно виконався етап синтаксичного аналізу. І використовує результат виконання попереднього аналізу, тобто два файли: перший містить згенерований асемблерний код відповідно операторам які були в програмі, другий файл містить таблицю змінних. Інформація з них зчитується в відповідному порядку, основні константні конструкції записуються в файл `asm`.

3.5.1 Розробка алгоритму роботи генератора коду

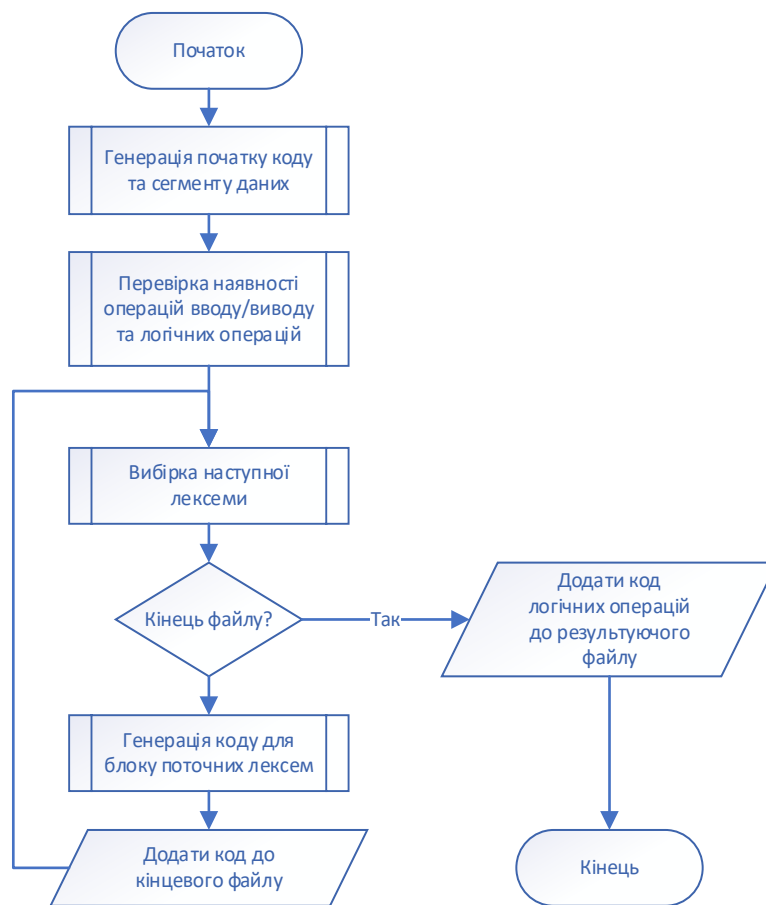


Рис. 3.4 Блок схема генератора коду

3.5.2 Опис програми реалізації генератора коду

У компілятора, реалізованого в даному курсовому проекті, вихідна мова - програма на мові Assembler. Ця програма записується у файл, що має таку ж саму назву, як і файл з вхідним текстом, але розширення “asm”. Генерація коду відбувається одразу ж після синтаксичного аналізу.

В даному трансляторі генератор коду послідовно викликає окремі функції, які записують у вихідний файл частини коду.

Першим кроком генерації коду записується ініціалізація сегменту даних. Далі виконується аналіз коду, та визначаються процедури, зміни, які використовуються.

Проаналізувавши змінні, які є у програмі, генератор формує код даних для асемблерної програми. Для цього з таблиці лексем вибирається ім'я змінної (типи змінних відповідають 4 байтам), та записується 0, в якості початкового значення.

Аналіз наявних процедур необхідний у зв'язку з тим, що процедури введення/виведення, виконання арифметичних та логічних операцій, виконано у вигляді окремих процедур і у випадку їх відсутності немає сенсу записувати у вихідний файл зайву інформацію.

Після цього зчитується лексема з таблиці лексем. Також відбувається перевірка, чи це не остання лексема. Якщо це остання лексема, то функція завершується.

Наступним кроком є аналіз таблиці лексем, та безпосередня генерація коду у відповідності до вхідної програми.

Генератор коду зчитує лексему та генерує відповідний код, який записується у файл. Наприклад, якщо це лексема виведення, то у основну програму записується виклик процедури виведення, попередньо записавши у співпроцесор значення, яке необхідно вивести. Якщо це арифметична операція, так само викликається дана процедура, але як і в попередньому випадку, спочатку у регістри співпроцесора записується інформація, яка вказує над якими значеннями виконувати дії.

Генератор закінчує свою роботу, коли зчитує лексему, що відповідає кінцю файлу.

В кінці своєї роботи, генератор формує код завершення асемблерної програми.

Дана програма написана мовою C++ з при розробці якої було створено структури `BackusRule` та `BackusRuleItem` за допомогою яких можна чітко описати нотатки Бекуса-Наура, які використовуються для семантично-лексичного аналізу написаної програми для заданої мови програмування

```
auto assignmentRule = BackusRule::MakeRule("AssignmentRule", {
    BackusRuleItem({ identRule->type()}, OnlyOne),
    BackusRuleItem({ Assignment::Type()}, OnlyOne),
    BackusRuleItem({ equation->type()}, OnlyOne)
});

auto read = BackusRule::MakeRule("ReadRule", {
    BackusRuleItem({ Read::Type()}, OnlyOne),
    BackusRuleItem({ LBracket::Type()}, OnlyOne),
    BackusRuleItem({ identRule->type()}, OnlyOne),
    BackusRuleItem({ RBracket::Type()}, OnlyOne)
});

auto write = BackusRule::MakeRule("WriteRule", {
    BackusRuleItem({ Write::Type()}, OnlyOne),
    BackusRuleItem({ LBracket::Type()}, OnlyOne | PairStart),
    BackusRuleItem({ stringRule->type(), equation->type() }, OnlyOne),
    BackusRuleItem({ RBracket::Type()}, OnlyOne | PairEnd)
});

auto codeBlok = BackusRule::MakeRule("CodeBlok", {
    BackusRuleItem({ Start::Type()}, OnlyOne),
    BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional |
OneOrMore),
    BackusRuleItem({ End::Type()}, OnlyOne)
});

auto topRule = BackusRule::MakeRule("TopRule", {
    BackusRuleItem({ Program::Type()}, OnlyOne),
    BackusRuleItem({ identRule->type()}, OnlyOne),
    BackusRuleItem({ Semicolon::Type()}, OnlyOne),
    BackusRuleItem({ Vars::Type()}, OnlyOne),
    BackusRuleItem({ varsBlok->type()}, OnlyOne),
    BackusRuleItem({ codeBlok->type()}, OnlyOne)
});
```

Вище наведено приклад опису нотаток Бекуса-Наура за допомогою цих структур. Наприклад `topRule` це правило, що відповідає за правильну структуру написаної програми, тобто якими лексемами вона повинна починатись та які операції можуть бути використанні всередині виконавчого блоку програми.

Всередині структури `BackusRule` описаний порядок tokenів для певного правила. А в структурі `BackusRuleItem` описані токени, які при перевірці трактується програмою як «АБО», тобто повинен бути лише один з описаних tokenів. Наприклад для `write` послідовно необхідний token `Write` після якого йде ліва дужка, далі може бути або певний вираз або рядок тексту який необхідно вивести. І закінчується правило токеном правої дужки.

Основна частина програми складається з 3 компонентів: парсера лексем, правил Бекуса-Наура та генератора асемблерного коду. Кожен з цих компонентів працює зі власним інтерфейсом на певному етапі виконання програми.

Кожен токен це окремий клас що наслідує 3 інтерфейси:

- `IToken`
- `IBackusRule`
- `IGeneratorItem`

Наявність наслідування цих інтерфейсів кожним токеном дозволяє без проблем звертатись до кожного віддільного токена на усіх етапах виконання програми

Для процесу парсингу програми використовується інтерфейс `IToken`. Що дозволяє простіше з точки зору реалізації звертатись до токенів при аналізі вхідної програми.

Правила Бекуса-Наура для своєї роботи використовують інтерфейс `IBackusRule`. Це дозволяє викликати функцію перевірки `check` до кожного прописаного у коді правила запису як програми в цілому так і кожного віддільної операції, що спрощує подальший пошук ймовірних помилок у коді програми, яка буде транслюватись у асемблерний код.

Інтерфейс `IGeneratorItem` використовується генератором асемблерного коду при трансляції вхідної програми. Оскільки кожен токен є віддільним класом, то у ньому була реалізована функція `genCode` яка використовується генератором, що дозволяє записати необхідний асемблерний код який буде згенерований певним токеном. Наприклад:

Для класу та токenu `Greate` що визначає при порівнянні який елемент більший, функція генерації відповідного коду виглядає наступним чином:

```
void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
{
    RegPROC(details);
    out << "\tcall Greate_\n";
};
```

За допомогою функції `RegPROC` токен за потреби реєструє процедуру у генераторі.

```
static void RegPROC(GeneratorDetails& details)
{
    if (!IsRegistered())
    {
        details.registerProc("Greate_", PrintGreate);
        SetRegistered();
    }
}

static void PrintGreate(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
{
    out << "===Procedure
Greate=====\\n";
    out << "Greate_ PROC\\n";
    out << "\tpushf\\n";
```

```

out << "\tpop cx\n\n";
out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";
out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";
out << "\tjle greate_false\n";
out << "\tmov " << args.regPrefix << "ax, 1\n";
out << "\tjmp greate_fin\n";
out << "greate_false:\n";
out << "\tmov " << args.regPrefix << "ax, 0\n";
out << "greate_fin:\n";
out << "\tpush cx\n";
out << "\tpopf\n\n";
GeneratorUtils::PrintResultToStack(out, args);
out << "\tret\n";
out << "Greate_ ENDP\n";
out <<
";=====
=====\\n";

}

```

Така структура програми дозволяє без проблем аналізувати великі програми, написані на вхідній мові програмування. Також використання правил Бекуса-Наура дозволяє ефективно аналізувати програми великого обсягу.

Генератор у свою чергу буде більш оптимізовано генерувати асемблерний код, створюючи код лише тих операцій, що буди використані у вхідній програмі.

4. ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА

4.1 Опис інтерфейсу та інструкція користувачеві

Вхідним файлом для даної програми є звичайний текстовий файл з розширенням s26. У цьому файлі необхідно набрати бажану для трансляції програму та зберегти її. Синтаксис повинен відповідати вхідній мові.

Створений транслятор є консольною програмою, що запускається з командної стрічки з параметром: "CWork_s26.exe <ім'я програми>.s26"

Якщо обидва файли мають місце на диску та правильно сформовані, програма буде запущена на виконання.

Початковою фазою обробки є лексичний аналіз (розбиття на окремі лексеми). Результатом цього етапу є файл lexems.txt, який містить таблицю лексем. Вміст цього файлу складається з 4 полів – 1 – безпосередньо сама лексема; 2 – тип лексеми; 3 – значення лексеми (необхідне для чисел і ідентифікаторів); 4 – рядок, у якому лексема знаходиться. Наступним етапом є перевірка на правильність написання програми (вхідної). Інформацію про наявність чи відсутність помилок можна переглянути у файлі error.txt. Якщо граматичний розбір виконаний успішно, файл буде містити відповідне повідомлення. Інакше, у файлі будуть зазначені помилки з їх описом та вказанням їх місця у тексті програми.

Останнім етапом є генерація коду. Транслятор переходить до цього етапу, лише у випадку, коли відсутні граматичні помилки у вхідній програмі. Згенерований код записується у файлу <ім'я програми>.asm.

Для отримання виконавчого файлу необхідно скористатись програмою Masm32.exe

Тестування програмного забезпечення є важливим етапом розробки продукту. На цьому етапі знаходяться помилки допущені на попередніх етапах. Цей етап дозволяє покращити певні характеристики продукту, наприклад – інтерфейс. Дає можливість знайти та вподальшому виправити слабкі сторони, якщо вони є.

Відлагодження даної програми здійснюється за допомогою набору кількох програм, які відповідають заданій граматиці. Та перевірка коректності коду, що генерується, коректності знаходження помилок та розбивки на лексеми.

4.2 Виявлення лексичних та синтаксичних помилок

Виявлення лексичних помилок відбувається на стадії лексичного аналізу. Під час розбиття вхідної програми на окремі лексеми відбувається перевірка чи відповідає вхідна лексема граматиці. Якщо ця лексема є в граматиці то вона ідентифікується і в таблиці лексем визначається. У випадку неспівпадіння лексемі присвоюється тип "невпізнаної лексеми". Повідомлення про такі помилки можна побачити лише після виконання процедури перевірки таблиці лексем, яка знаходиться в файлі.

Виявлення синтаксичних помилок відбувається на стадії перевірки програми на коректність окремо від синтаксичного аналізу. При цьому перевіряється окремо кожне твердження яке може бути або виразом, або оператором (циклу, вводу/виводу), або оголошенням, та перевіряється структура програми в цілому.

Приклад виявлення:

Текст програми з помилками

```
!!Prog1
NAME _pROGRA1;
BODY
DATA INT EGER_2 _aAAAAAAA, _BBBBBBB, _XXXXXXX, _YYYYYYY;
PRINT("Input A: ");
SCAN(_aAAeAAAA);
PRINT("Input B: ");
SCAN(_BBBBBBB);
PRINT("A + B: ");
PRINT(_aAAAAAAA ADD _BBBBBBB);
PRINT("\nA - B: ");
PRINT(_aAAAAAAA SUB _BBBBBBB);
PRINT("\nA * B: ");
PRINT(_aAAAAAAA MUL _BBBBBBB);
PRINT("\nA / B: ");
PRINT(_aAAAAAAA DIV _BBBBBBB);
PRINT("\nA % B: ");
PRINT(_aAAAAAAA MOD _BBBBBBB);
_xXXXXXX<-( _aAAAAAAA SUB _BBBBBBB) MUL 10 ADD ( _aAAAAAAA ADD _BBBBBBB)
DIV 10;
_yYYYYYYY<- _xxxxxxx ADD ( _xxxxxxx MOD 10);
PRINT("\nX = (A - B) * 10 + (A + B) / 10\n");
PRINT(_xxxxxxx);
PRINT("\nY = X + (X % 10)\n");
PRINT(_yyyyyyyy);
END
```

Текст файлу з повідомленнями про помилки

List of errors

There are 6 lexical errors.
There are 1 syntax errors.
There are 0 semantic errors.

Line 4: Lexical error: Unknown token: INT
Line 4: Lexical error: Unknown token: EGER_2
Line 4: Lexical error: Unknown token: _aAAAAAAA
Line 4: Syntax error: Expected: VarsBlok before INT
Line 6: Lexical error: Unknown token: _aAAeAAAA
Line 7: Lexical error: Unknown token: PRI
Line 7: Lexical error: Unknown token: NT

Суттю виявлення семантичних помилок є перевірка числових констант на відповідність типу INTEGER_2, тобто знаковому цілому числу з відповідним діапазоном значень і перевірку на коректність використання змінних INTEGER_2 у цілочисельних і логічних виразах.

4.3 Перевірка роботи транслятора за допомогою тестових задач

Для того щоб здійснити перевірку коректності роботи транслятора необхідно завантажити коректну до заданої вхідної мови програму.

Текст коректної програми

```
!!Prog1
NAME _pROGRA1;
BODY
DATA INTEGER_2 _aAAAAAA, _bBBBBBB, _xXXXXXX, _yYYYYYY;
PRINT("Input A: ");
SCAN(_aAAAAAA);
PRINT("Input B: ");
SCAN(_bBBBBBB);
PRINT("A + B: ");
PRINT(_aAAAAAA ADD _bBBBBBB);
PRINT("\nA - B: ");
PRINT(_aAAAAAA SUB _bBBBBBB);
PRINT("\nA * B: ");
PRINT(_aAAAAAA MUL _bBBBBBB);
PRINT("\nA / B: ");
PRINT(_aAAAAAA DIV _bBBBBBB);
PRINT("\nA % B: ");
PRINT(_aAAAAAA MOD _bBBBBBB);
_xXXXXXX<-( _aAAAAAA SUB _bBBBBBB) MUL 10 ADD ( _aAAAAAA ADD _bBBBBBB)
DIV 10;
_yYYYYYY<- _xXXXXXX ADD ( _xXXXXXX MOD 10);
PRINT("\nX = (A - B) * 10 + (A + B) / 10\n");
PRINT(_xXXXXXX);
PRINT("\nY = X + (X % 10)\n");
PRINT(_yYYYYYY);
END
```


Оскільки дана програма відповідає граматиці то результати виконання лексичного, синтаксичного аналізів, а також генератора коду будуть позитивними.

В результаті буде отримано асемблерний файл, який є результатом виконання трансляції з заданої вхідної мови на мову Assembler даної програми (його вміст наведений в Додатку А).

Після виконання компіляції даного файлу на виході отримаєм наступний результат роботи програми:

```
Input A: 5
Input B: 9
A + B: 14
A - B: -4
A - B: 45
A - B: 0
A - B: 5
X = (A - B) * 10 + (A + B) / 10
-39
Y = X + (X % 10)
-48
```

Рис. 4.1 Результат виконання коректної програми

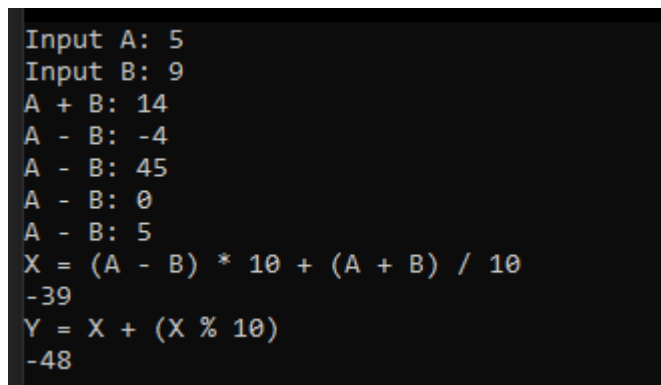
При перевірці отриманого результату, можна зробити висновок про правильність роботи програми, а отже і про правильність роботи транслятора.

4.4 Тестова програма №1

Текст програми

```
!!Prog1
NAME _pROGRA1;
BODY
DATA INTEGER 2 _aAAAAAA, _bBBBBBB, _xXXXXXX, _yYYYYYY;
PRINT("Input A: ");
SCAN(_aAAAAAA);
PRINT("Input B: ");
SCAN(_bBBBBBB);
PRINT("A + B: ");
PRINT(_aAAAAAA ADD _bBBBBBB);
PRINT("\nA - B: ");
PRINT(_aAAAAAA SUB _bBBBBBB);
PRINT("\nA * B: ");
PRINT(_aAAAAAA MUL _bBBBBBB);
PRINT("\nA / B: ");
PRINT(_aAAAAAA DIV _bBBBBBB);
PRINT("\nA % B: ");
PRINT(_aAAAAAA MOD _bBBBBBB);
_xXXXXXX<-( _aAAAAAA SUB _bBBBBBB) MUL 10 ADD ( _aAAAAAA ADD _bBBBBBB)
DIV 10;
_yYYYYYY< _xXXXXXX ADD ( _xXXXXXX MOD 10);
PRINT("\nX = (A - B) * 10 + (A + B) / 10\n");
PRINT(_xXXXXXX);
PRINT("\nY = X + (X % 10)\n");
PRINT(_yYYYYYY);
END
```

Результат виконання



```
Input A: 5
Input B: 9
A + B: 14
A - B: -4
A - B: 45
A - B: 0
A - B: 5
X = (A - B) * 10 + (A + B) / 10
-39
Y = X + (X % 10)
-48
```

Рис. 4.2 Результат виконання тестової програми №1

4.5 Тестова програма №2

Текст програми

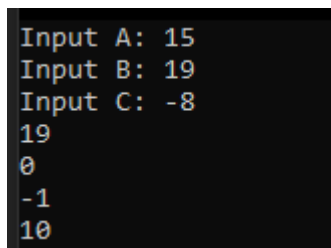
```
!!Prog2
NAME _pROGRA2;
BODY
DATA INTEGER_2 _aAAAAAA,_bBBBBBB,_cCCCCC;
PRINT("Input A: ");
SCAN(_aAAAAAA);
PRINT("Input B: ");
SCAN(_bBBBBBB);
PRINT("Input C: ");
SCAN(_cCCCCC);
IF(_aAAAAAA >= _bBBBBBB)
BODY
    IF(_aAAAAAA >= _cCCCCC)
    BODY
        GOTO _aBIGGER;
    END
    ELSE
    BODY
        PRINT(_cCCCCC);
        GOTO _oUTOFI;
        _aBIGGER:
        PRINT(_aAAAAAA);
        GOTO _oUTOFI;
    END
END
END
    IF(_bBBBBBB <= _cCCCCC)
    BODY
        PRINT(_cCCCCC);
    END
    ELSE
    BODY
        PRINT(_bBBBBBB);
    END
    _oUTOFI:
    PRINT("\n");
    IF((_aAAAAAA EQ _bBBBBBB) AND (_aAAAAAA EQ _cCCCCC) AND (_bBBBBBB EQ
    _cCCCCC))
    BODY
        PRINT(1);
    END
    ELSE
    BODY
        PRINT(0);
    END
    PRINT("\n");
```

```

IF((_aAAAAAA <= 0) OR (_bBBBBBB <= 0) OR (_cCCCCC <= 0))
BODY
    PRINT(- 1);
END
ELSE
BODY
    PRINT(0);
END
PRINT("\n");
IF(NOT(_aAAAAAA <= (_bBBBBBB ADD _cCCCCC)))
BODY
    PRINT(10);
END
ELSE
BODY
    PRINT(0);
END
END

```

Результат виконання



```

Input A: 15
Input B: 19
Input C: -8
19
0
-1
10

```

Рис. 4.3 Результат виконання тестової програми №2

4.6 Тестова програма №3

Текст програми

```

!!Prog3
NAME _pROGRA3;
BODY
DATA INTEGER _2 _aAAAAA,_aAAAAA2,_bBBBBBB,_xXXXXXX,_cCCCCC1,_cCCCCC2;
PRINT("Input A: ");
SCAN(_aAAAAA);
PRINT("Input B: ");
SCAN(_bBBBBBB);
PRINT("FOR TO DO");
FOR _aAAAAA2<=_aAAAAA TO _bBBBBBB DO
BODY
    PRINT("\n");
    PRINT(_aAAAAA2 MUL _aAAAAA2);
END
PRINT("\nFOR DOWNT0 DO");
FOR _aAAAAA2<=_bBBBBBB DOWNT0 _aAAAAA DO
BODY

```

```

PRINT("\n");
PRINT(_aAAAAA2 MUL _aAAAAA2);
END

PRINT("\nWHILE A * B: ");
_xXXXXXX<-0;
_cCCCCC1<-0;
WHILE(_cCCCCC1 <= _aAAAAA)
BODY
    _cCCCCC2<-0;
    WHILE (_cCCCCC2 <= _bBBBBBB)
    BODY
        _xXXXXXX<-_xXXXXXX ADD1;
        _cCCCCC2<-_cCCCCC2 ADD1;
    END WHILE
    _cCCCCC1<-_cCCCCC1 ADD1;
END WHILE
PRINT(_xXXXXXX);

PRINT("\nREPEAT UNTIL A * B: ");
_xXXXXXX<-0;
_cCCCCC1<-1;
REPEAT
    _cCCCCC2<-1;
    REPEAT
        _xXXXXXX<-_xXXXXXX ADD1;
        _cCCCCC2<-_cCCCCC2 ADD1;
    UNTIL(NOT(_cCCCCC2 >= _bBBBBBB))
    _cCCCCC1<-_cCCCCC1 ADD1;
UNTIL(NOT(_cCCCCC1 >= _aAAAAA))
PRINT(_xXXXXXX);

END

```

Результат виконання

```

Input A: 5
Input B: 9
FOR TO DO
25
36
49
64
81
FOR DOWNT0 DO
81
64
49
36
25
WHILE A * B: 45
REPEAT UNTIL A * B: 45

```

Рис. 4.4 Результат виконання тестової програми №3

ВИСНОВКИ

В процесі виконання курсового проекту було виконано наступне:

1. Складено формальний опис мови програмування s26, в термінах розширеної нотації Бекуса-Наура, виділено усі термінальні символи та ключові слова.

2. Створено компілятор мови програмування s26, а саме:

2.1. Розроблено прямий лексичний аналізатор, орієнтований на розпізнавання лексем, що є заявлені в формальному описі мови програмування.

2.2. Розроблено синтаксичний аналізатор на основі низхідного методу. Складено деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура

2.3. Розроблено генератор коду, відповідні процедури якого викликаються після перевірки синтаксичним аналізатором коректності запису чергового оператора, мови програмування s26. Вихідним кодом генератора є програма на мові Assembler(x86).

3. Проведене тестування компілятора на тестових програмах за наступними пунктами:

3.1. На виявлення лексичних помилок.

3.2. На виявлення синтаксичних помилок.

3.3. Загальна перевірка роботи компілятора.

Тестування не виявило помилок в роботі компілятор, і всі помилки в тестових програмах на мові s26 були успішно виявлені і відповідно оброблені.

В результаті виконання даної курсового проекту було засвоєно методи розробки та реалізації компонент систем програмування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Language Processors Explained
URL: [Language Processors Explained - baeldung](#)
2. Error Handling in Compiler Design
URL: [Error Handling in Compiler Design - GeeksforGeeks](#)
3. What are BNF and EBNF in Programming?
URL: [What are BNF and EBNF in Programming? - freeCodeCamp](#)
4. Create Programming Language: Design Principles
URL: [Create Programming Language: Design Principles – daily.dev](#)
5. Symbol Table in Compiler
URL: [Symbol Table in Compiler - GeeksforGeeks](#)
6. Stack Overflow
URL: [Stack Overflow - Where Developers Learn, Share, & Build Careers](#)

ДОДАТКИ

Додаток А (Таблиці лексем)

Див. файли «додаток_A_ТЛ_Проґ1.pdf», «додаток_A_ТЛ_Проґ2.pdf», «додаток_A_ТЛ_Проґ3.pdf»

Додаток Б (Лістинги основного програмного коду)

Main.cpp

```
#include "stdafx.h"
#include "Controller.h"
#include "Core/Parser/TokenRegister.h"
#include "Core/Parser/TokenParser.h"
#include "Core/Generator/Generator.h"

int main(int argc, std::string* argv)
{
    try
    {
        std::filesystem::path file;

        const std::string extension = ".s26";

        const std::string longLine =
            "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~";

        if (argc != 2)
        {
            printf("Input file name\n");
            std::cin >> file;
        }
        else
        {
            file = argv->c_str();
        }

        Init();

        if (file.extension() != extension)
        {
            std::cout << longLine << std::endl;
            std::cout << "Wrong file extension" << std::endl;
            system("pause");
            return 0;
        }

        std::string fileName = file.replace_extension("").string();
        std::string errorFileName = fileName + "_errors.txt";
        std::string lexemsFileName = fileName + "_lexems.txt";
        std::string tokensFileName = fileName + "_tokens.txt";
        std::string asmFileName = fileName + ".asm";

        std::cout << longLine << std::endl;
        std::cout << "DEBUG: Breaking into lexems are starting..." << std::endl;
        std::fstream inputFile{ fileName + extension, std::ios::in };
        auto tokens = TokenParser::Instance()->tokenize(inputFile);
        inputFile.close();
        std::cout << "DEBUG: Breaking into lexems completed. There are " <<
tokens.size() << " lexems" << std::endl;

        std::fstream lexemsFile(lexemsFileName, std::ios::out);
        TokenParser::PrintTokens(lexemsFile, tokens);
        lexemsFile.close();
    }
}
```



```

std::cout << "DEBUG: Report file: " << lexemsFileName << std::endl;
std::cout << longLine << std::endl;

std::cout << "Error checking are starting... " << std::endl;
std::fstream errorFile(errorFileName, std::ios::out);
auto semanticCheckRes = CheckSemantic(errorFile, tokens);
errorFile.close();
if (semanticCheckRes)
{
    std::cout << "There are no errors in the file" << std::endl;
    std::cout << longLine << std::endl;
}
else
{
    std::cout << "There are errors in the file. Check " << errorFileName << "
for more information" << std::endl;
    std::cout << longLine << std::endl;
}

std::fstream tokensFile(tokensFileName, std::ios::out);
TokenParser::PrintTokens(tokensFile, tokens);
tokensFile.close();
std::cout << "There are " << tokens.size() << " tokens." << std::endl;
std::cout << "Report file: " << tokensFileName << std::endl;

if (semanticCheckRes)
{
    std::cout << longLine << std::endl;
    std::cout << "DEBUG: Code generation is starting..." << std::endl;
    std::fstream asmFile(asmFileName, std::ios::out);
    Generator::Instance()->generateCode(asmFile, tokens);
    asmFile.close();

    if (std::filesystem::is_directory("masm32"))
    {
        std::cout << "DEBUG: Code generation is completed" << std::endl;
        std::cout << longLine << std::endl;
        system(std::string("masm32\\bin\\ml /c /coff " + fileName +
".asm").c_str());
        system(std::string("masm32\\bin\\Link /SUBSYSTEM:WINDOWS " + fileName
+ ".obj").c_str());
    }
    else
    {
        std::cout << "WARNING! Can't compile asm file, because masm32 doesn't
exist" << std::endl;
    }
}
}
catch (const std::exception& ex)
{
    std::cout << "Error: " << ex.what() << std::endl;
}
catch (...)
{
    std::cout << "Internal error." << std::endl;
}

system("pause");
return 0;
}

```

BackusRule.h

```
#pragma once
#include "stdafx.h"
#include "Core/Backus/IBackusRule.h"
#include "BackusRuleItem.h"

class Controller;

class BackusRule : public IBackusRule
{
public:
    virtual ~BackusRule() = default;

    bool check(std::multimap<int, std::pair<std::string, std::vector<std::string>>>&
errorsInfo,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end) final;

    std::string type() const final { return m_name; };

    std::string lexeme() const final { return ""; };
    void setValue(const std::string& value) final {};
    std::string value() const final { return ""; };
    int line() const final { return -1; };
    std::string customData(const std::string& id) const final { return ""; };
    void setCustomData(const std::string& data, const std::string& id) final
{};

    void setPostHandler(const
std::function<void(std::list<std::shared_ptr<IBackusRule>>::iterator& ruleBegin,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end)>& handler) final
    {
        m_handler = handler;
    };

private:
    friend class Controller;

    static std::shared_ptr<IBackusRule> MakeRule(std::string name,
std::list<BackusRuleItem> items);

    BackusRule(const std::string& name, const std::list<BackusRuleItem>& items) :
m_name(name), m_backusItem(items) {}

    bool oneOrMoreCheck(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end,
        const BackusRuleItem& item) const;

    bool checkItem(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end,
        const BackusRuleItem& item) const;

    static bool HasFlag(RuleCountPolicy policy, RuleCountPolicy flag);

private:
    std::string m_name;
    std::list<BackusRuleItem> m_backusItem;
    std::function<void(std::list<std::shared_ptr<IBackusRule>>::iterator& ruleBegin,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end)> m_handler;
};
```

BackusRuleBase.h

```
#pragma once
#include "stdafx.h"
#include "Core/Backus/IBackusRule.h"

template <class T>
class BackusRuleBase : public IBackusRule
{
public:
    bool check(std::multimap<int, std::pair<std::string, std::vector<std::string>>>&
errorsInfo,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end) final
    {
        auto res = type() == (*it)->type();
        if (res)
            it++;
        return res;
    }

    void setPostHandler(const
std::function<void(std::list<std::shared_ptr<IBackusRule>>::iterator& ruleBegin,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end)>& handler) final { };
};
```

BackusRuleItem.h

```
#pragma once
#include "stdafx.h"
#include "Core/Backus/IBackusRule.h"
#include "BackusRuleStorage.h"
#include "Symbols.h"
#include "Utils/magic_enum.hpp"

class BackusRuleItem
{
public:
    BackusRuleItem(const std::vector<std::variant<std::string, Symbols>>& rules,
RuleCountPolicy policy) : m_policy(policy)
    {
        for (auto rule : rules)
        {
            if (std::holds_alternative<std::string>(rule))
                m_ruleNames.push_back(std::get<std::string>(rule));
            else
                m_ruleNames.emplace_back(magic_enum::enum_name(std::get<Symbols>(rule)));
        }

        std::vector<std::shared_ptr<IBackusRule>> rules() const
        {
            if (m_rules.empty())
                m_rules = BackusRuleStorage::Instance()->getRules(m_ruleNames);

            return m_rules;
        };

        RuleCountPolicy policy() const { return m_policy; };

private:
    std::vector<std::string> m_ruleNames;
    mutable std::vector<std::shared_ptr<IBackusRule>> m_rules;
    RuleCountPolicy m_policy = NoPolicy;
};
```

BackusRuleStorage.h

```
#pragma once
#include "stdafx.h"
#include "Utils/singleton.hpp"
#include "Core/Backus/IBackusRule.h"

class BackusRuleStorage : public singleton<BackusRuleStorage>
{
public:
    void regRule(std::shared_ptr<IBackusRule> rule)
    {
        auto [it, inserted] = m_rules.try_emplace(rule->type(), rule);
        if (!inserted)
        {
            throw std::runtime_error("BackusRuleStorage::regRule: A rule with the type " + rule->type() + " already exists.");
        }
    }

    std::vector<std::shared_ptr<IBackusRule>> getRules(const std::vector<std::string>& ruleTypes) const
    {
        std::vector<std::shared_ptr<IBackusRule>> rules;

        for (const auto& ruleType : ruleTypes)
        {
            auto it = m_rules.find(ruleType);
            if (it == m_rules.end())
                throw std::runtime_error("BackusRuleStorage::regRule: A rule with the type " + ruleType + " not found.");

            rules.push_back(it->second);
        }

        return rules;
    };

private:
    std::map<std::string, std::shared_ptr<IBackusRule>> m_rules;
};
```

IBackusRule.h

```
#pragma once
#include "stdafx.h"
#include "Core/IItem.h"

enum RuleCountPolicy : std::uint16_t
{
    NoPolicy = 0,
    Optional = 1 << 0,
    OnlyOne = 1 << 1,
    Several = 1 << 2,
    OneOrMore = OnlyOne | Several,
    PairStart = 1 << 3,
    PairEnd = 1 << 4,
};

DEFINE_ENUM_FLAG_OPERATORS(RuleCountPolicy)

__interface IBackusRule : public IItem
{
};
```

```

    virtual bool check(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end) = 0;

    virtual void setPostHandler(const
std::function<void(std::list<std::shared_ptr<IBackusRule>>::iterator& ruleBegin,
        std::list<std::shared_ptr<IBackusRule>>::iterator& it,
        std::list<std::shared_ptr<IBackusRule>>::iterator& end)>& handler) = 0;
};

```

BackusRule.cpp

```

#pragma once
#include "stdafx.h"
#include "BackusRule.h"

std::shared_ptr<IBackusRule> BackusRule::MakeRule(std::string name,
std::list<BackusRuleItem> items)
{
    struct EnableMakeShared : public BackusRule { EnableMakeShared(const std::string&
name, const std::list<BackusRuleItem>& items) : BackusRule(name, items) {} };

    return std::make_shared<EnableMakeShared>(name, items);
}

bool BackusRule::check(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
    std::list<std::shared_ptr<IBackusRule>>::iterator& it,
    std::list<std::shared_ptr<IBackusRule>>::iterator& end)
{
    bool res = true;
    bool pairItem = false;
    auto ruleBegin = it;
    for (auto item = m_backusItem.begin(); item != m_backusItem.end(); ++item)
    {
        if (it == end || !pairItem && HasFlag(item->policy(),
RuleCountPolicy::PairEnd))
        {
            if (!HasFlag(item->policy(), RuleCountPolicy::Optional) || item !=
m_backusItem.end())
            {
                std::vector<std::string> types;

                for (const auto& rule : item->rules())
                    types.push_back(rule->type());

                errorsInfo.emplace((*it)->line(), std::make_pair((*it)->value(),
types));
                res = false;
            }
            break;
        }

        if (pairItem && HasFlag(item->policy(), RuleCountPolicy::PairEnd) ||
!HasFlag(item->policy(), RuleCountPolicy::PairEnd))
        {
            bool resItem = true;
            auto startIt = it;
            if (HasFlag(item->policy(), RuleCountPolicy::Several))
                resItem = oneOrMoreCheck(errorsInfo, it, end, *item);
            else
                resItem = checkItem(errorsInfo, it, end, *item);

            if (!resItem && (!HasFlag(item->policy(), RuleCountPolicy::Optional) ||
startIt != it))

```

```

        {
            res &= resItem;
            break;
        }

        if (resItem && HasFlag(item->policy(), RuleCountPolicy::PairStart))
        {
            pairItem = true;
        }

        if (resItem && pairItem && HasFlag(item->policy(),
RuleCountPolicy::PairEnd))
        {
            pairItem = false;
        }
    }
}

if (res && m_handler)
    m_handler(ruleBegin, it, end);

return res;
}

bool BackusRule::oneOrMoreCheck(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
std::list<std::shared_ptr<IBackusRule>>>::iterator& it,
std::list<std::shared_ptr<IBackusRule>>>::iterator& end,
const BackusRuleItem& item) const
{
    bool res = true;
    bool resItem = true;
    while (resItem && it != end && HasFlag(item.policy(), RuleCountPolicy::Several))
    {
        auto startIt = it;
        res &= resItem;
        resItem = checkItem(errorsInfo, it, end, item);

        if (!resItem && startIt != it)
            res = false;
    }

    return res;
}

bool BackusRule::checkItem(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
std::list<std::shared_ptr<IBackusRule>>>::iterator& it,
std::list<std::shared_ptr<IBackusRule>>>::iterator& end,
const BackusRuleItem& item) const
{
    bool res = false;
    std::vector<std::string> types;

    auto startIt = it;
    auto maxIt = it;
    if (it != end)
    {
        std::multimap<int, std::pair<std::string, std::vector<std::string>>> errors;
        for (auto rule : item.rules())
        {
            types.push_back(rule->type());

            if (!res && startIt == it)
            {
                res = rule->check(errors, it, end);
            }
        }
    }
}

```

```

    }

    if (res)
    {
        break;
    }
    else if (!res && startIt != it)
    {
        if(std::distance(maxIt, end) > std::distance(it, end))
            maxIt = it;

        it = startIt;
        errorsInfo.insert(errors.begin(), errors.end());
    }
}

if (std::distance(maxIt, end) < std::distance(it, end))
    it = maxIt;

if (!res)
    errorsInfo.emplace((*startIt)->line(), std::make_pair((*it)->value(), types));
else
    errorsInfo.clear();

return res;
}

bool BackusRule::HasFlag(RuleCountPolicy policy, RuleCountPolicy flag)
{
    return (policy & flag) == flag;
}

```

Generator.h

```

#pragma once
#include "stdafx.h"
#include "Utils/singleton.hpp"
#include "Core/Generator/GeneratorItemBase.h"

class Generator : public singleton<Generator>
{
public:
    template<class T>
    void generateCode(std::ostream& out, std::list<std::shared_ptr<T>>& items) const
    {
        if (!m_details)
            throw std::runtime_error("Generator details is not set");

        std::list<std::shared_ptr<IGeneratorItem>> generatorItems;
        for (auto item : items)
        {
            generatorItems.push_back(std::dynamic_pointer_cast<IGeneratorItem>(item));
        }
        auto it = generatorItems.begin();
        auto end = generatorItems.end();

        std::stringstream code;
        genCode(code, *m_details, it, end);

        PrintBegin(out, *m_details);
        PrintData(out, *m_details);
        PrintBeginCodeSegment(out, *m_details);
        out << code.str();
        PrintEnding(out, *m_details);
    }
}

```

```

    void setDetails(const GeneratorDetails& details) { m_details =
std::make_shared<GeneratorDetails>(details); }

protected:
    Generator() = default;

private:
    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const;

private:
    static void PrintBegin(std::ostream& out, GeneratorDetails& details);
    static void PrintData(std::ostream& out, GeneratorDetails& details);
    static void PrintBeginCodeSegment(std::ostream& out, GeneratorDetails& details);
    static void PrintEnding(std::ostream& out, GeneratorDetails& details);

private:
    std::shared_ptr<GeneratorDetails> m_details;
};

```

GeneratorDetails.h

```

#pragma once
#include "stdafx.h"

class GeneratorDetails
{
    friend class Generator;
public:
    struct GeneratorArgs
    {
        std::string regPrefix;
        std::string numberType;
        std::string numberTypeExtended;
        size_t argSize;
        size_t posArg0;
        size_t posArg1;
        std::string numberStrType;
    };

public:
    explicit GeneratorDetails(const GeneratorArgs& args) : m_args(args)
    {
        m_args.posArg0 = m_kRetAddrSize + m_args.argSize;
        m_args.posArg1 = m_kRetAddrSize;
    }

    const GeneratorArgs& args() const { return m_args; }

    void registerNumberData(const std::string& name)
    {
        throwIfDataExists(name);
        m_userNumberData[name] = '\t' + name + '\t' + m_args.numberType + '\t' + "0";
    }

    void registerStringData(const std::string& name, const std::string& data)
    {
        throwIfDataExists(name);

        std::string item;
        size_t start = 0;
        size_t end;
        std::string delimiter = "\\n";
    }

```



```

        m_userStringData[name] = '\t' + name + "\tdb\t";

        while ((end = data.find(delimiter, start)) != std::string::npos)
        {
            item = data.substr(start, end - start);
            if (!item.empty())
                m_userStringData[name] += "\"" + item + "\", ";
            m_userStringData[name] += "13, 10, ";
            start = end + delimiter.length();
        }

        item = data.substr(start);
        if (!item.empty())
            m_userStringData[name] += "\"" + item + "\", ";

        m_userStringData[name] += "0";
    }

    void registerRawData(const std::string& name, const std::string& rawData)
    {
        throwIfDataExists(name);
        m_userRawData[name] = '\t' + name + '\t' + rawData;
    }

    void registerProc(const std::string& type, const std::function<void(std::ostream&
out, const GeneratorArgs*)>& generator)
    {
        if (!m_procGenerators.contains(type))
            m_procGenerators[type] = generator;
        else
            throw std::runtime_error("Proc for type " + type + " already exists");
    }

private:
    void throwIfDataExists(const std::string& name) const
    {
        if (m_userNumberData.contains(name) || m_userStringData.contains(name) ||
m_userRawData.contains(name))
            throw std::runtime_error("Data with name " + name + " already exists");
    }

private:
    GeneratorArgs m_args;

    std::map<std::string, std::string> m_userNumberData;
    std::map<std::string, std::string> m_userStringData;
    std::map<std::string, std::string> m_userRawData;
    std::map<std::string, std::function<void(std::ostream& out, const
GeneratorArgs*)>> m_procGenerators;

    static constexpr size_t m_kRetAddrSize = 4;
};

```

GeneratorItemBase.h

```

#pragma once
#include "stdafx.h"
#include "Core/Generator/GeneratorUtils.h"

template <class T>
class GeneratorItemBase : public IGeneratorItem
{
public:
    virtual ~GeneratorItemBase() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,

```

```

        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const
override {};

protected:
    std::string customData_imp(const std::string& id) const { return m_customData[id]; }
}
    void setCustomData_imp(const std::string& data, const std::string& id) {
m_customData[id] = data; }

    static bool IsRegistered() { return registered; }
    static void SetRegistered() { registered = true; }

    static bool registered;

private:
    mutable std::map<std::string, std::string> m_customData{ {"default", ""} };
};

template<class T>
bool GeneratorItemBase<T>::registered = false;

```

GeneratorUtils.h

```

#pragma once
#include "stdafx.h"
#include "Utils/singleton.hpp"
#include "Core/Generator/IGeneratorItem.h"

class GeneratorUtils : public singleton<GeneratorUtils>
{
public:
    void RegisterOperation(const std::string& type, size_t priority)
    {
        m_operations[type] = priority;
    }

    void RegisterOperand(const std::string& type)
    {
        m_operands.insert(type);
    }

    void RegisterEquationEnd(const std::string& type)
    {
        m_equationEnd.insert(type);
    }

    void RegisterLBracket(const std::string& type)
    {
        m_lBracketType = type;
    }

    void RegisterRBracket(const std::string& type)
    {
        m_rBracketType = type;
    }

    std::list<std::shared_ptr<IGeneratorItem>> ConvertToPostfixForm(
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const
    {
        std::list<std::shared_ptr<IGeneratorItem>> postfixForm;
        std::list<std::shared_ptr<IGeneratorItem>> stack;

        while (it != end)
        {

```

```

        auto item = *it;
        auto itemType = item->type();

        if (IsOperand(item))
        {
            postfixForm.push_back(item);
        }
        else if (IsOperation(item))
        {
            while (!stack.empty() && !Prioritet(item, stack.back()) &&
stack.back()->type() != m_lBraketType)
            {
                postfixForm.push_back(stack.back());
                stack.pop_back();
            }
            stack.push_back(item);
        }
        else if (itemType == m_lBraketType)
        {
            stack.push_back(item);
            postfixForm.push_back(item);
        }
        else if (itemType == m_rBraketType)
        {
            while (stack.back()->type() != m_lBraketType)
            {
                postfixForm.push_back(stack.back());
                stack.pop_back();
            }
            stack.pop_back();
            postfixForm.push_back(item);
        }

        if (IsNextEndOfEquation(it, end))
        {
            break;
        }

        ++it;
    }

    while (!stack.empty())
    {
        postfixForm.push_back(stack.back());
        stack.pop_back();
    }

    return postfixForm;
}

static void PrintResultToStack(std::ostream& out, const
GeneratorDetails::GeneratorArgs& args)
{
    out << "\tmov [esp + " << args.posArg0 << "], " << args.regPrefix << "ax\n";
    out << "\tpop ecx\n";
    out << "\tpop " << args.regPrefix << "ax\n";
    out << "\tpush ecx\n";
}

static bool IsNextTokenIs(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end,
const std::string& type)
{
    auto res = false;
    if (it != end && std::next(it) != end && (*std::next(it))->type() == type)

```

```

        res = true;

    return res;
}

private:
    inline bool IsOperand(const std::shared_ptr<IGeneratorItem>& item) const
    {
        return m_operands.contains(item->type());
    }

    inline bool IsOperation(const std::shared_ptr<IGeneratorItem>& item) const
    {
        return m_operations.contains(item->type());
    }

    bool Prioritet(const std::shared_ptr<IGeneratorItem>& left, const
std::shared_ptr<IGeneratorItem>& right) const
    {
        size_t leftPriority = 0;
        size_t rightPriority = 0;

        if (IsOperation(left))
            leftPriority = m_operations.at(left->type());

        if (IsOperation(right))
            rightPriority = m_operations.at(right->type());

        return leftPriority > rightPriority;
    }

    bool IsNextEndOfEquation(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const
    {
        auto res = true;

        if (it != end && std::next(it) != end)
        {
            auto next = *std::next(it);
            res = m_equationEnd.contains(next->type()) || IsNextTokenOnNextLine(it,
end);
        }

        return res;
    }

    static bool IsNextTokenOnNextLine(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end)
    {
        auto res = false;
        if (it != end && std::next(it) != end && ((*it)->line() + 1) ==
(*std::next(it))->line())
            res = true;

        return res;
    }

private:
    std::map<std::string, size_t> m_operations;
    std::set<std::string> m_operands;
    std::set<std::string> m_equationEnd;
    std::string m_lBracketType;
    std::string m_rBracketType;
};

```

IGeneratorItem.h

```
#pragma once
#include "stdafx.h"
#include "Utils/singleton.hpp"
#include "Core/Generator/IGeneratorItem.h"

class GeneratorUtils : public singleton<GeneratorUtils>
{
public:
    void RegisterOperation(const std::string& type, size_t priority)
    {
        m_operations[type] = priority;
    }

    void RegisterOperand(const std::string& type)
    {
        m_operands.insert(type);
    }

    void RegisterEquationEnd(const std::string& type)
    {
        m_equationEnd.insert(type);
    }

    void RegisterLBraket(const std::string& type)
    {
        m_lBraketType = type;
    }

    void RegisterRBraket(const std::string& type)
    {
        m_rBraketType = type;
    }

    std::list<std::shared_ptr<IGeneratorItem>> ConvertToPostfixForm(
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const
    {
        std::list<std::shared_ptr<IGeneratorItem>> postfixForm;
        std::list<std::shared_ptr<IGeneratorItem>> stack;

        while (it != end)
        {
            auto item = *it;
            auto itemType = item->type();

            if (IsOperand(item))
            {
                postfixForm.push_back(item);
            }
            else if (IsOperation(item))
            {
                while (!stack.empty() && !Prioritet(item, stack.back()) &&
                    stack.back()->type() != m_lBraketType)
                {
                    postfixForm.push_back(stack.back());
                    stack.pop_back();
                }
                stack.push_back(item);
            }
            else if (itemType == m_lBraketType)
            {
                stack.push_back(item);
                postfixForm.push_back(item);
            }
        }
    }
};
```

```

    }
    else if (itemType == m_rBracketType)
    {
        while (stack.back()->type() != m_lBracketType)
        {
            postfixForm.push_back(stack.back());
            stack.pop_back();
        }
        stack.pop_back();
        postfixForm.push_back(item);
    }

    if (IsNextEndOfEquation(it, end))
    {
        break;
    }

    ++it;
}

while (!stack.empty())
{
    postfixForm.push_back(stack.back());
    stack.pop_back();
}

return postfixForm;
}

static void PrintResultToStack(std::ostream& out, const
GeneratorDetails::GeneratorArgs& args)
{
    out << "\tmov [esp + " << args.posArg0 << "], " << args.regPrefix << "ax\n";
    out << "\tpop ecx\n";
    out << "\tpop " << args.regPrefix << "ax\n";
    out << "\tpush ecx\n";
}

static bool IsNextTokenIs(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end,
const std::string& type)
{
    auto res = false;
    if (it != end && std::next(it) != end && (*std::next(it))->type() == type)
        res = true;

    return res;
}

private:
inline bool IsOperand(const std::shared_ptr<IGeneratorItem>& item) const
{
    return m_operands.contains(item->type());
}

inline bool IsOperation(const std::shared_ptr<IGeneratorItem>& item) const
{
    return m_operations.contains(item->type());
}

bool Prioritet(const std::shared_ptr<IGeneratorItem>& left, const
std::shared_ptr<IGeneratorItem>& right) const
{
    size_t leftPriority = 0;
    size_t rightPriority = 0;

```

```

        if (IsOperation(left))
            leftPriority = m_operations.at(left->type());

        if (IsOperation(right))
            rightPriority = m_operations.at(right->type());

        return leftPriority > rightPriority;
    }

    bool IsNextEndOfEquation(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const
    {
        auto res = true;

        if (it != end && std::next(it) != end)
        {
            auto next = *std::next(it);
            res = m_equationEnd.contains(next->type()) || IsNextTokenOnNextLine(it,
end);
        }

        return res;
    }

    static bool IsNextTokenOnNextLine(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end)
    {
        auto res = false;
        if (it != end && std::next(it) != end && ((*it)->line() + 1) ==
(*std::next(it))->line())
            res = true;

        return res;
    }

private:
    std::map<std::string, size_t> m_operations;
    std::set<std::string> m_operands;
    std::set<std::string> m_equationEnd;
    std::string m_lBracketType;
    std::string m_rBracketType;
};

```

Generator.cpp

```

#include "stdafx.h"
#include "Generator.h"

void Generator::PrintBegin(std::ostream& out, GeneratorDetails& details)
{
    out << ".386\n";
    out << ".model flat, stdcall\n";
    out << "option casemap :none\n";
    out << std::endl;

    out << "include masm32\\include\\windows.inc\n";
    out << "include masm32\\include\\kernel32.inc\n";
    out << "include masm32\\include\\masm32.inc\n";
    out << "include masm32\\include\\user32.inc\n";
    out << "include masm32\\include\\msvcrt.inc\n";

    out << "includelib masm32\\lib\\kernel32.lib\n";
}

```

```

    out << "includelib masm32\\lib\\masm32.lib\n";
    out << "includelib masm32\\lib\\user32.lib\n";
    out << "includelib masm32\\lib\\msvcrt.lib\n";
}

void Generator::PrintData(std::ostream& out, GeneratorDetails& details)
{
    out << std::endl;
    out << ".DATA\n";
    out << "===User
Data=====\\n";

    for (const auto& [_, data] : details.m_userNumberData)
    {
        out << data << std::endl;
    }
    if (!details.m_userNumberData.empty())
        out << std::endl;

    for (const auto& [_, data] : details.m_userStringData)
    {
        out << data << std::endl;
    }
    if (!details.m_userStringData.empty())
        out << std::endl;

    out << "===Addition
Data=====\\n";
    out << "\\thConsoleInput\\tdd\\t?\\n";
    out << "\\thConsoleOutput\\tdd\\t?\\n";
    out << "\\tendBuff\\t\\t\\tdb\\t5 dup (?)\\n";
    out << "\\tmsg1310\\t\\t\\tdb\\t13, 10, 0\\n";

    if (!details.m_userRawData.empty())
        out << std::endl;

    for (const auto& [_, data] : details.m_userRawData)
    {
        out << data << std::endl;
    }
}

void Generator::PrintBeginCodeSegment(std::ostream& out, GeneratorDetails& details)
{
    out << std::endl;
    out << ".CODE\\n";
    out << "start:\\n";
    out << "invoke AllocConsole\\n";
    out << "invoke GetStdHandle, STD_INPUT_HANDLE\\n";
    out << "mov hConsoleInput, eax\\n";
    out << "invoke GetStdHandle, STD_OUTPUT_HANDLE\\n";
    out << "mov hConsoleOutput, eax\\n";
}

void Generator::PrintEnding(std::ostream& out, GeneratorDetails& details)
{
    out << "exit_label:\\n";
    out << "invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0,
0\\n";
    out << "invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0\\n";
    out << "invoke ExitProcess, 0\\n";

    for (const auto& [_, proc] : details.m_procGenerators)
    {
        out << std::endl << std::endl;
    }
}

```



```

        proc(out, details.args());
    }

    out << "end start\n";
}

void Generator::genCode(std::ostream& out, GeneratorDetails& details,
    std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
    const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const
{
    for (; it != end; ++it)
    {
        (*it)->genCode(out, details, it, end);
    }
}

```

TokenParser.h

```

#pragma once
#include "stdafx.h"
#include "Utils/singleton.hpp"
#include "Core/Tokens/IToken.hpp"
#include "Utils/TablePrinter.h"

class TokenParser : public singleton<TokenParser>
{
public:
    static constexpr int NoPriority = std::numeric_limits<int>::min();

public:
    std::list<std::shared_ptr<IToken>> tokenize(std::istream& input);

    void regToken(std::shared_ptr<IToken> token, int priority = NoPriority);
    void regUnchangedTextToken(std::shared_ptr<IToken> target, std::shared_ptr<IToken>
lBorder, std::shared_ptr<IToken> rBorder);

    template<class T>
    static void PrintTokens(std::ostream& out, const std::list<std::shared_ptr<T>>&
tokens)
    {
        auto getNumCount = [](int k) { return std::to_string(k).size(); };

        size_t maxLemexeLen = 0;
        size_t maxTypeLen = 0;
        size_t maxValueLen = 0;

        for (auto token : tokens)
        {
            maxLemexeLen = std::max(maxLemexeLen, token->lexeme().size());
            maxTypeLen = std::max(maxTypeLen, token->type().size());
            maxValueLen = std::max(maxValueLen, token->value().size());
        }

        const std::string kHeaderColumn0 = "#";
        const std::string kHeaderColumn1 = "SYMBOL";
        const std::string kHeaderColumn2 = "TYPE";
        const std::string kHeaderColumn3 = "VALUE";
        const std::string kHeaderColumn4 = "LINE";

        size_t colPadding = 1;

        auto widthColumn0 = std::max(kHeaderColumn0.size(),
getNumCount(tokens.size())) + 2 * colPadding;
        auto widthColumn1 = std::max(kHeaderColumn1.size(), maxLemexeLen) + 2 *
colPadding;

```

```

        auto widthColumn2 = std::max(kHeaderColumn2.size(), maxTypeLen) + 2 *
colPadding;
        auto widthColumn3 = std::max(kHeaderColumn3.size(), maxValueLen) + 2 *
colPadding;
        auto widthColumn4 = std::max(kHeaderColumn4.size(), getNumCount(tokens.back()-
>line())) + 2 * colPadding;

        if ((kHeaderColumn0.size() % 2) != (widthColumn0 % 2)) widthColumn0++;
        if ((kHeaderColumn1.size() % 2) != (widthColumn1 % 2)) widthColumn1++;
        if ((kHeaderColumn2.size() % 2) != (widthColumn2 % 2)) widthColumn2++;
        if ((kHeaderColumn3.size() % 2) != (widthColumn3 % 2)) widthColumn3++;
        if ((kHeaderColumn4.size() % 2) != (widthColumn4 % 2)) widthColumn4++;

        size_t index = 1;
        auto getIndex = [&index](const std::shared_ptr<T>&) { return
std::to_string(index++); };
        auto getLemexe = [](const std::shared_ptr<T>& token) { return token->lexeme();
};
        auto getType = [](const std::shared_ptr<T>& token) { return token->type(); };
        auto getValue = [](const std::shared_ptr<T>& token) { return token->value();
};
        auto getLine = [](const std::shared_ptr<T>& token) { return
std::to_string(token->line()); };

        TablePrinter::PrintTable(out,
        { kHeaderColumn0, kHeaderColumn1, kHeaderColumn2, kHeaderColumn3,
kHeaderColumn4 },
        { widthColumn0, widthColumn1, widthColumn2, widthColumn3, widthColumn4 },
        { TablePrinter::CENTRE, TablePrinter::RIGHT, TablePrinter::RIGHT,
TablePrinter::RIGHT, TablePrinter::RIGHT },
        tokens,
        { getIndex, getLemexe, getType, getValue, getLine },
        colPadding);
    }

private:
    void throwIfTokenRegistered(std::shared_ptr<IToken> token);

    void recognizeToken(std::string& token, int curLine);
    bool isUnchangedTextTokenLast();

private:
    static bool IsNewLine(const char& ch);
    static bool IsTabulation(const char& ch);
    static bool IsAllowedSymbol(const char& ch);
    static bool IsAllowedSpecialSymbol(const char& ch);

private:
    struct PriorityCompare
    {
        bool operator()(const int& a, const int& b) const
        {
            return a > b;
        }
    };

private:
    std::multimap<int, std::shared_ptr<IToken>, PriorityCompare> m_priorityTokens;
    std::map<std::string, std::tuple<std::shared_ptr<IToken>, std::shared_ptr<IToken>,
std::shared_ptr<IToken>>> m_unchangedTextTokens;

    std::list<std::shared_ptr<IToken>> m_tokens;

    std::function<std::shared_ptr<IToken>(std::string)> m_getTokenByType =
[this](const std::string& type) {
        auto start = m_priorityTokens.lower_bound(static_cast<int>(type.size()));

```

```

        auto mapItem = std::find_if(start, m_priorityTokens.end(), [&type](const auto&
pair) { return pair.second->type() == type; });

        if (mapItem == m_priorityTokens.end())
            throw std::runtime_error("TokenParser::getTokenByType: Token with type " +
type + " not found");

        return mapItem->second;
    };
};

```

TokenParser.cpp

```

#include "stdafx.h"
#include "Core/Parser/TokenParser.h"
#include "Utils/StringUtils.h"
#include "Tokens/Common/EndOfFile.h"

std::list<std::shared_ptr<IToken>> TokenParser::tokenize(std::istream& input)
{
    m_tokens.clear();

    int curLine = 1;
    std::string token;
    for (char ch; input.get(ch);)
    {
        if (!token.empty() && ((IsAllowedSymbol(token.front()) != IsAllowedSymbol(ch))
|| IsTabulation(ch)))
            recognizeToken(token, curLine);

        if (IsNewLine(ch))
            ++curLine;

        if (isUnchangedTextTokenLast())
        {
            std::string unchangedTextTokenValue{ token };
            token.clear();
            int unchangedTextTokenLine{ curLine };

            const auto& [target, left, right] = m_unchangedTextTokens[m_tokens.back()-
>lexeme()];
            auto rBorderLex = right ? right->lexeme() : "\n";

            do
            {
                if (IsNewLine(ch))
                    ++curLine;

                unchangedTextTokenValue += ch;
            }
            while (!StringUtils::Compare(unchangedTextTokenValue, rBorderLex,
StringUtils::EndWith) && input.get(ch));

            unchangedTextTokenValue = unchangedTextTokenValue.substr(0,
unchangedTextTokenValue.size() - rBorderLex.size());
            m_tokens.push_back(target->tryCreateToken(unchangedTextTokenValue));
            m_tokens.back()->setLine(unchangedTextTokenLine);

            if (right)
            {
                m_tokens.push_back(right->tryCreateToken(rBorderLex));
                m_tokens.back()->setLine(curLine);
            }

            continue;
        }
    }
}

```

```

        if (!IsTabulation(ch))
            token += ch;
    }

    if (!token.empty())
        recognizeToken(token, curLine);

    m_tokens.push_back(std::make_shared<EndOfFile>());
    return m_tokens;
}

void TokenParser::regToken(std::shared_ptr<IToken> token, int priority)
{
    throwIfTokenRegistered(token);

    if (priority == NoPriority)
        priority = static_cast<int>(token->lexeme().size());

    m_priorityTokens.insert(std::make_pair(priority, token));
}

void TokenParser::regUnchangedTextToken(std::shared_ptr<IToken> target,
std::shared_ptr<IToken> lBorder, std::shared_ptr<IToken> rBorder)
{
    if(rBorder)
        throwIfTokenRegistered(rBorder);

    regToken(lBorder);
    throwIfTokenRegistered(target);

    m_unchangedTextTokens.try_emplace(lBorder->lexeme(), target, lBorder, rBorder);
}

void TokenParser::throwIfTokenRegistered(std::shared_ptr<IToken> token)
{
    auto start = m_priorityTokens.lower_bound(static_cast<int>(token->lexeme().size()));

    auto priorToken = std::find_if(start, m_priorityTokens.end(),
[&token](const auto& pair) {
    return token->type() == pair.second->type();
});

    auto unchTextToken = std::ranges::find_if(m_unchangedTextTokens,
[&token](const auto& pair) {
    auto type = token->type();
    const auto& [main, left, right] = pair.second;
    return type == main->type() ||
        type == left->type() ||
        right && type == right->type();
});

    if(priorToken != m_priorityTokens.end() || unchTextToken !=
m_unchangedTextTokens.end())
        throw std::runtime_error("TokenParser: Token with type " + token->type() + "
already registered");
}

void TokenParser::recognizeToken(std::string& token, int curLine)
{
    if(m_priorityTokens.empty())
        throw std::runtime_error("TokenParser: No tokens registered");

    auto start = m_priorityTokens.lower_bound(static_cast<int>(token.size()));

```

```

    for (auto it = start; it != m_priorityTokens.end(); ++it)
    {
        auto curRegToken = it->second;
        if (auto newToken = curRegToken->tryCreateToken(token); newToken)
        {
            m_tokens.push_back(newToken);
            m_tokens.back()->setLine(curLine);
            break;
        }
    }

    if (!token.empty() && !isUnchangedTextTokenLast())
        recognizeToken(token, curLine);
}

bool TokenParser::isUnchangedTextTokenLast()
{
    if (!m_tokens.empty() && m_unchangedTextTokens.contains(m_tokens.back()-
>lexeme()))
    {
        auto const& [target, left, right] = m_unchangedTextTokens[m_tokens.back()-
>lexeme()];
        if (m_tokens.size() >= 2)
        {
            if (target->type() != (*(++m_tokens.rbegin()))->type())
                return true;
        }
        else
            return true;
    }

    return false;
}

bool TokenParser::IsNewLine(const char& ch)
{
    return ch == '\n';
}

bool TokenParser::IsTabulation(const char& ch)
{
    return ch == ' ' || ch == '\t' || IsNewLine(ch);
}

bool TokenParser::IsAllowedSymbol(const char& ch)
{
    return !isalpha(ch) || !isdigit(ch) || IsAllowedSpecialSymbol(ch);
}

bool TokenParser::IsAllowedSpecialSymbol(const char& ch)
{
    std::set<char> allowedSymblos{ '._' };
    return allowedSymblos.contains(ch);
}

```

TokenRegister.h

```

#pragma once
#include "stdafx.h"
#include "Controller.h"

#include "Rules/IdentRule/Undefined.h"
#include "Tokens/Common/Unknown.h"

void Init();

```

```

template <typename T>
bool CheckSemantic(std::ostream& out, std::list<std::shared_ptr<T>>& tokens)
{
    auto endOfFileType = tokens.back()->type();
    std::list<std::shared_ptr<IBackusRule>> rules;
    for (auto token : tokens)
    {
        if (auto rule = std::dynamic_pointer_cast<IBackusRule>(token))
            rules.push_back(rule);
    }

    auto it = rules.begin();
    auto end = rules.end();
    std::multimap<int, std::pair<std::string, std::vector<std::string>>> errors;
    auto res = Controller::Instance()->topRule()->check(errors, it, end);

    rules.erase(++std::find_if(it, rules.end(), [&endOfFileType](const auto& rule) {
return rule->type() == endOfFileType; }), rules.end());
    end = --rules.end();

    std::multimap<int, std::string> errorMsg;

    int lexErr = 0;
    int synErr = 0;
    int semErr = 0;

    tokens.clear();
    for (auto rule : rules)
    {
        tokens.push_back(std::dynamic_pointer_cast<T>(rule));
        if (rule->type() == Undefined::Type())
        {
            res = false;
            std::string err;
            if (auto erMsg = rule->customData("error"); !erMsg.empty())
            {
                semErr++;
                err = "Semantic error: " + erMsg;
            }
            else
            {
                semErr++;
                err = std::format("Semantic error: Undefined token: {}", rule-
>value());
            }
            errorMsg.emplace(rule->line(), err);
        }
        else if (rule->type() == token::Unknown::Type())
        {
            lexErr++;
            res = false;
            errorMsg.emplace(rule->line(), std::format("Lexical error: Unknown token:
{}", rule->value()));
        }
    }

    for (auto it = errors.rbegin(); it != errors.rend(); ++it)
    {
        auto types = it->second.second;
        std::stringstream ss;
        for (size_t i = 0; i < types.size(); ++i)
        {
            if (!types[i].empty())
            {
                ss << types[i];
            }
        }
    }
}

```

```

        if (i != types.size() - 1)
            ss << " or ";
    }
}

auto ssStr = ss.str();
if (!ssStr.empty())
{
    synErr++;
    std::string msg = "Syntax error: Expected: " + ssStr;

    if (!it->second.first.empty())
        msg += " before " + it->second.first;

    errorsMsg.emplace(it->first, msg);
}
}

out << "List of errors" << std::endl;
out << "===== " <<
std::endl;
out << "There are " << lexErr << " lexical errors." << std::endl;
out << "There are " << synErr << " syntax errors." << std::endl;
out << "There are " << semErr << " semantic errors." << std::endl << std::endl;
for (auto const& [line, msg] : errorsMsg)
{
    out << "Line " << line << ": " << msg << std::endl;
}

return res;
}

```

TokenRegister.cpp

```

#include "stdafx.h"
#include "Core/Parser/TokenRegister.h"
#include "Controller.h"
#include "Tokens/Common.h"

#include "Rules/Operators/If/IfRule.h"
#include "Rules/Operators/Goto/GotoRule.h"
#include "Rules/Operators/For/ForRule.h"
#include "Rules/Operators/WhileC/WhileRule.h"
#include "Rules/Operators/RepeatUntil/RepeatUntilRule.h"

void Init()
{
    Controller::Instance()->regOperatorRule(MakeIf);
    Controller::Instance()->regOperatorRule(MakeGoto, true);
    Controller::Instance()->regOperatorRule(MakeLabel);
    Controller::Instance()->regOperatorRule(MakeFor);
    Controller::Instance()->regOperatorRule(MakeWhile);
    Controller::Instance()->regOperatorRule(MakeRepeatUntil);

    Controller::Instance()->regItem<token::Unknown>(ItemType::TokenAndRule, -2);

    Controller::Instance()->regUnchangedTextToken(std::make_shared<Comment>(),
std::make_shared<LComment>(), nullptr);

    Controller::Instance()->init();
}

```

TokenOperators:

Loops:

Do.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Do : public TokenBase<Do>, public BackusRuleBase<Do>, public
GeneratorItemBase<Do>
{
    BASE_ITEM

public:
    Do() { setLexeme("DO"); };
    virtual ~Do() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        out << "\tpop " << details.args().regPrefix << "ax" << std::endl;
        out << "\tcmp " << details.args().regPrefix << "ax, 0" << std::endl;
        out << "\tje " << customData("endLabel") << std::endl;
    };
};
```

DownTo.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Rules/EquationRule/Greate.h"
#include "Rules/EquationRule/Not.h"
#include "Rules/EquationRule/Subtraction.h"

class DownTo : public TokenBase<DownTo>, public BackusRuleBase<DownTo>, public
GeneratorItemBase<DownTo>
{
    BASE_ITEM

public:
    DownTo() { setLexeme("DOWNT0"); };
    virtual ~DownTo() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        Greate::RegPROC(details);
        Not::RegPROC(details);
        Subtraction::RegPROC(details);
        out << customData("startLabel") << ":" << std::endl;

        it++;
        auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

        auto postIt = postForm.begin();
        auto postEnd = postForm.end();
        for (const auto& item : postForm)
```



```

        item->genCode(out, details, postIt, postEnd);

        out << "\tpush " << customData("ident") << std::endl;
        out << "\tcall Greate_" << std::endl;
        out << "\tcall Not_" << std::endl;
    };
};

```

For.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class For : public TokenBase<For>, public BackusRuleBase<For>, public
GeneratorItemBase<For>
{
    BASE_ITEM

public:
    For() { setLexeme("FOR"); };
    virtual ~For() = default;
};

```

ForRule.h

```

#pragma once
#include "stdafx.h"
#include "Controller.h"

BackusRulePtr MakeFor(std::shared_ptr<Controller> controller);

```

ForRule.cpp

```

#include "stdafx.h"
#include "ForRule.h"

#include "Rules/Operators/For/For.h"
#include "Rules/Operators/For/To.h"
#include "Rules/Operators/For/DownTo.h"
#include "Rules/Operators/For/Do.h"

BackusRulePtr MakeFor(std::shared_ptr<Controller> controller)
{
    using enum ItemType;

    controller->regItem<For>();
    controller->regItem<To>(TokenAndRule | EquationEnd);
    controller->regItem<DownTo>(TokenAndRule | EquationEnd);
    controller->regItem<Do>(TokenAndRule | EquationEnd);

    auto context = controller->context();
    static const auto [lStart, lCodeBlok, lEnd] = context->CodeBlockTypes();

    auto forToOrDownToDoRule = controller->addRule("ForToOrDownToDoRule", {
        BackusRuleItem({ For::Type(), OnlyOne),
        BackusRuleItem({ "AssignmentRule", OnlyOne),
        BackusRuleItem({ To::Type(), DownTo::Type(), OnlyOne),
        BackusRuleItem({ context->EquationRuleName(), OnlyOne),
        BackusRuleItem({ Do::Type(), OnlyOne),
        BackusRuleItem({ lCodeBlok, OnlyOne)
    });
};

```

```

forToOrDownToDoRule->setPostHandler([context](BackusRuleList::iterator& ruleBegin,
BackusRuleList::iterator& it,
BackusRuleList::iterator& end)
{
    static size_t index = 0;
    index++;

    std::string startLabel = std::format("forPasStart{}", index);
    std::string endLabel = std::format("forPasEnd{}", index);

    auto ident = *std::next(ruleBegin, 1);

    bool increment = false;
    for (auto itr = ruleBegin; itr != it; ++itr)
    {
        auto type = (*itr)->type();
        if ((type == To::Type() || type == DownTo::Type()))
        {
            if (type == To::Type())
                increment = true;

            (*itr)->setCustomData(startLabel, "startLabel");
            (*itr)->setCustomData(ident->customData(), "ident");
        }
        else if (type == Do::Type())
        {
            (*itr)->setCustomData(endLabel, "endLabel");
            break;
        }
    }

    std::string code;
    code += std::format("\tpush {}\n", ident->customData());
    code += std::format("\tpush {} ptr 1\n", context-
>Details().args().numberTypeExtended);
    code += std::format("\tcall {}\n", increment ? "Add_" : "Sub_");
    code += std::format("\tpop {}\n", ident->customData());
    code += std::format("\tjmp {}\n", startLabel);
    code += std::format("{}:", endLabel);

    (*std::prev(it, 1))->setCustomData(code);
});

return forToOrDownToDoRule;
}

```

To.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Rules/EquationRule/Less.h"
#include "Rules/EquationRule/Not.h"
#include "Rules/EquationRule/Addition.h"

class To : public TokenBase<To>, public BackusRuleBase<To>, public
GeneratorItemBase<To>
{
    BASE_ITEM

public:
    To() { setLexeme("TO"); };
    virtual ~To() = default;

```

```

void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
{
    Less::RegPROC(details);
    Not::RegPROC(details);
    Addition::RegPROC(details);
    out << customData("startLabel") << ":" << std::endl;

    it++;
    auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

    auto postIt = postForm.begin();
    auto postEnd = postForm.end();
    for (const auto& item : postForm)
        item->genCode(out, details, postIt, postEnd);

    out << "\tpush " << customData("ident") << std::endl;
    out << "\tcall Less_" << std::endl;
    out << "\tcall Not_" << std::endl;
};
};

```

Goto:

Goto.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Goto : public TokenBase<Goto>, public BackusRuleBase<Goto>, public
GeneratorItemBase<Goto>
{
    BASE_ITEM

public:
    Goto() { setLexeme("GOTO"); };
    virtual ~Goto() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        it++;

        out << "\tjmp " << (*it)->customData() << std::endl;
    };
};

```

GotoRule.h

```

#pragma once
#include "stdafx.h"
#include "Controller.h"

BackusRulePtr MakeGoto(std::shared_ptr<Controller> controller);
BackusRulePtr MakeLabel(std::shared_ptr<Controller> controller);

```

GotoRule.cpp

```

#include "stdafx.h"
#include "GotoRule.h"

```

```

#include "Rules/Operators/Goto/Goto.h"
#include "Rules/Operators/Goto/Label.h"

#include "Rules/IdentRule/Identifier.h"
#include "Rules/IdentRule/Undefined.h"

static std::map<std::string, std::optional<BackusRuleList::iterator>> labelTable;

BackusRulePtr MakeLabel(std::shared_ptr<Controller> controller)
{
    using enum ItemType;

    controller->regItem<Label>(Rule);

    auto context = controller->context();
    static const auto [lStart, lCodeBlok, lEnd] = context->CodeBlockTypes();

    auto labelRule = controller->addRule("LabelRule", {
        BackusRuleItem({ context->IdentRuleName()}, OnlyOne),
        BackusRuleItem({ Symbols::Colon}, OnlyOne)
    });

    labelRule->setPostHandler([context](BackusRuleList::iterator&
        BackusRuleList::iterator& it,
        BackusRuleList::iterator& end)
    {
        it = std::prev(it, 2);
        auto identIt = it;
        auto identVal = (*identIt)->value();

        std::shared_ptr<IToken> label;
        if (context->IdentTable().contains((*identIt)->value()))
        {
            label = std::make_shared<Undefined>();
            label->setCustomData("Redefinition", "error");
        }
        else
            label = std::make_shared<Label>();

        label->setValue((*identIt)->value() + ((*++it))->value());
        end = std::remove(it, end, *it);
        label->setLine((*identIt)->line());
        label->setCustomData((*identIt)->customData());
        *identIt = std::dynamic_pointer_cast<IBackusRule>(label);

        if (!labelTable.contains(identVal))
        {
            labelTable.try_emplace(identVal,
std::optional<BackusRuleList::iterator>());
        }
        else
        {
            if (auto optIt = labelTable[identVal]; optIt.has_value())
            {
                auto gotoIdentIt = optIt.value();
                if ((*gotoIdentIt)->type() == Undefined::Type())
                {
                    auto labelName = std::make_shared<Identifier>();
                    labelName->setValue((*gotoIdentIt)->value());
                    labelName->setLine((*gotoIdentIt)->line());
                    labelName->setCustomData((*gotoIdentIt)->customData());
                    *gotoIdentIt = labelName;
                }
            }
        }
    });
}

```

```

    }
    });

    return labelRule;
}

BackusRulePtr MakeGoto(std::shared_ptr<Controller> controller)
{
    controller->regItem<Goto>();

    auto context = controller->context();
    static const auto [lStart, lCodeBlok, lEnd] = context->CodeBlockTypes();

    auto gotoStatement = controller->addRule("GotoStatement", {
        BackusRuleItem({ Goto::Type()}, OnlyOne),
        BackusRuleItem({context->IdentRuleName()}, OnlyOne)
    });

    gotoStatement->setPostHandler([](BackusRuleList::iterator&,
        BackusRuleList::iterator& it,
        BackusRuleList::iterator& end)
    {
        it = std::prev(it, 1);
        auto identIt = it;
        if (!labelTable.contains((*identIt)->value()))
        {
            if ((*identIt)->type() != Undefined::Type())
            {
                auto undef = std::make_shared<Undefined>();
                undef->setValue((*identIt)->value());
                undef->setLine((*identIt)->line());
                undef->setCustomData((*identIt)->customData());
                *identIt = undef;
            }
            labelTable.try_emplace((*identIt)->value(), identIt);
        }
        else
        {
            auto ident = std::make_shared<Identifier>();
            ident->setValue((*identIt)->value());
            ident->setLine((*identIt)->line());
            ident->setCustomData((*identIt)->customData());
            *identIt = ident;
        }
        it = std::next(it);
    });

    return gotoStatement;
}

```

Label.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Label : public TokenBase<Label>, public BackusRuleBase<Label>, public
GeneratorItemBase<Label>
{
    BASE_ITEM

public:
    Label() { setLexeme(""); };
    virtual ~Label() = default;

```

```

void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
{
    out << customData() << ":" << std::endl;
};
};

```

IfElse:

Else.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Else : public TokenBase<Else>, public BackusRuleBase<Else>, public
GeneratorItemBase<Else>
{
    BASE_ITEM

public:
    Else() { setLexeme("ELSE"); };
    virtual ~Else() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        out << "\tjmp " << customData("endLabel") << std::endl;
        out << customData("elseLabel") << ":\n";
    };
};

```

If.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class If : public TokenBase<If>, public BackusRuleBase<If>, public
GeneratorItemBase<If>
{
    BASE_ITEM

public:
    If() { setLexeme("IF"); };
    virtual ~If() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        it++;
        auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

        auto postIt = postForm.begin();
    };
};

```

```

    auto postEnd = postForm.end();
    for (const auto& item : postForm)
        item->genCode(out, details, postIt, postEnd);

    out << "\tpop " << details.args().regPrefix << "ax" << std::endl;
    out << "\tcmp " << details.args().regPrefix << "ax, 0" << std::endl;
    out << "\tje " << customData("label") << std::endl;
};
};

```

IfRule.h

```

#pragma once
#include "stdafx.h"
#include "Controller.h"

BackusRulePtr MakeIf(std::shared_ptr<Controller> controller);

```

IfRule.cpp

```

#include "stdafx.h"
#include "IfRule.h"

#include "Rules/Operators/If/If.h"
#include "Rules/Operators/If/Else.h"

BackusRulePtr MakeIf(std::shared_ptr<Controller> controller)
{
    controller->regItem<If>();
    controller->regItem<Else>();

    auto context = controller->context();
    static const auto [lStart, lCodeBlok, lEnd] = context->CodeBlockTypes();

    auto elseStatement = controller->addRule("ElseStatement", {
        BackusRuleItem({ Else::Type(), OnlyOne},
        BackusRuleItem({ lCodeBlok}, OnlyOne),
    });

    auto ifStatement = controller->addRule("IfStatement", {
        BackusRuleItem({ If::Type(), OnlyOne},
        BackusRuleItem({ Symbols::LBraket}, OnlyOne),
        BackusRuleItem({ context->EquationRuleName(), OnlyOne},
        BackusRuleItem({ Symbols::RBraket}, OnlyOne),
        BackusRuleItem({ lCodeBlok}, OnlyOne),
        BackusRuleItem({ elseStatement->type(), Optional)
    });

    ifStatement->setPostHandler([(BackusRuleList::iterator& ruleBegin,
        BackusRuleList::iterator& it,
        BackusRuleList::iterator& end)
    {
        static size_t index = 0;
        index++;

        std::string elseLabel = std::format("elseLabel{}", index);
        std::string endLabel = std::format("endIf{}", index);

        bool hasElse = false;
        size_t count = 0;
        for (auto itr = ruleBegin; itr != it; ++itr)
        {
            auto type = (*itr)->type();
            if (type == lStart)
            {
                count++;
            }
        }
    }
    });
}

```

```

    }
    else if (type == Else::Type() && count == 0)
    {
        (*itr)->setCustomData(elseLabel, "elseLabel");
        (*itr)->setCustomData(endLabel, "endLabel");
        hasElse = true;
    }
    else if (type == lEnd && count == 1 && (*std::next(itr))->type() !=
Else::Type())
    {
        (*itr)->setCustomData(endLabel + ':');
        break;
    }
    else if (type == lEnd && count > 0)
    {
        count--;
    }
}

(*ruleBegin)->setCustomData(hasElse ? elseLabel : endLabel, "label");
});

    return ifStatement;
}

```

RepeatUntil:

Repeat.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Repeat : public TokenBase<Repeat>, public BackusRuleBase<Repeat>, public
GeneratorItemBase<Repeat>
{
    BASE_ITEM

public:
    Repeat() { setLexeme("REPEAT"); };
    virtual ~Repeat() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        out << customData("startLabel") << ":" << std::endl;
    };
};

```

RepeatUntilRule.h

```

#pragma once
#include "stdafx.h"
#include "Controller.h"

BackusRulePtr MakeRepeatUntil(std::shared_ptr<Controller> controller);

```


RepeatUntilRule.cpp

```
#include "stdafx.h"
#include "RepeatUntilRule.h"

#include "Rules/Operators/RepeatUntil/Repeat.h"
#include "Rules/Operators/RepeatUntil/Until.h"

BackusRulePtr MakeRepeatUntil(std::shared_ptr<Controller> controller)
{
    controller->regItem<Repeat>();
    controller->regItem<Until>();

    auto context = controller->context();
    static const auto [lStart, lCodeBlok, lEnd] = context->CodeBlockTypes();
    auto operatorsRuleName = context->OperatorsRuleName();

    auto repeatUntilRule = controller->addRule("RepeatUntilRule", {
        BackusRuleItem({ Repeat::Type(), OnlyOne),
        BackusRuleItem({ operatorsRuleName, OnlyOne),
        BackusRuleItem({ Until::Type(), OnlyOne),
        BackusRuleItem({ Symbols::LBraket, OnlyOne),
        BackusRuleItem({ context->EquationRuleName(), OnlyOne),
        BackusRuleItem({ Symbols::RBraket, OnlyOne)
    });

    repeatUntilRule->setPostHandler([](BackusRuleList::iterator& ruleBegin,
        BackusRuleList::iterator& it,
        BackusRuleList::iterator& end)
    {
        static size_t index = 0;
        index++;

        std::string startLabel = std::format("repeatStart{}", index);
        std::string endLabel = std::format("repeatEnd{}", index);

        (*ruleBegin)->setCustomData(startLabel, "startLabel");

        size_t count = 0;
        for (auto itr = ruleBegin; itr != it; ++itr)
        {
            auto type = (*itr)->type();
            if (type == Repeat::Type())
            {
                count++;
            }
            else if (type == Until::Type() && count == 1)
            {
                count--;
                (*itr)->setCustomData(startLabel, "startLabel");
                (*itr)->setCustomData(endLabel, "endLabel");
                break;
            }
            else if (type == Until::Type() && count > 0)
            {
                count--;
            }
        }
    });

    return repeatUntilRule;
}
```

Until.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Until : public TokenBase<Until>, public BackusRuleBase<Until>, public
GeneratorItemBase<Until>
{
    BASE_ITEM

public:
    Until() { setLexeme("UNTIL"); };
    virtual ~Until() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        it++;
        auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

        auto postIt = postForm.begin();
        auto postEnd = postForm.end();
        for (const auto& item : postForm)
            item->genCode(out, details, postIt, postEnd);

        out << "\ttop " << details.args().regPrefix << "ax" << std::endl;
        out << "\tcmp " << details.args().regPrefix << "ax, 0" << std::endl;
        out << "\tje " << customData("endLabel") << std::endl;
        out << "\tjmp " << customData("startLabel") << std::endl;
        out << customData("endLabel") << ":" << std::endl;
    };
};
```

While:

While.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class While : public TokenBase<While>, public BackusRuleBase<While>, public
GeneratorItemBase<While>
{
    BASE_ITEM

public:
    While() { setLexeme("WHILE"); };
    virtual ~While() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        if (customData("noGenerateCode") == "true")
        {
            return;
        }
    };
};
```

```

    if (customData("ContinueWhile") == "true")
    {
        out << "\tjmp " << customData("startLabel") << std::endl;
        return;
    }

    if (customData("ExitWhile") == "true")
    {
        out << "\tjmp " << customData("endLabel") << std::endl;
        return;
    }

    it++;
    auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

    out << customData("startLabel") << ":" << std::endl;

    auto postIt = postForm.begin();
    auto postEnd = postForm.end();
    for (const auto& item : postForm)
        item->genCode(out, details, postIt, postEnd);

    out << "\tpop " << details.args().regPrefix << "ax" << std::endl;
    out << "\tcmp " << details.args().regPrefix << "ax, 0" << std::endl;
    out << "\tje " << customData("endLabel") << std::endl;
};
};

```

WhileRule.h

```

#pragma once
#include "stdafx.h"
#include "Controller.h"

BackusRulePtr MakeWhile(std::shared_ptr<Controller> controller);

```

WhileRule.cpp

```

#include "stdafx.h"
#include "WhileRule.h"

#include "Rules/Operators/WhileC/While.h"

#include "SimpleTokens.h"

SimpleToken(ExitWhile, "EXIT")
SimpleToken(ContinueWhile, "CONTINUE")

BackusRulePtr MakeWhile(std::shared_ptr<Controller> controller)
{
    controller->regItem<While>();
    controller->regItem<ExitWhile>();
    controller->regItem<ContinueWhile>();

    auto context = controller->context();
    static const auto [lStart, lCodeBlok, lEnd] = context->CodeBlockTypes();
    auto operatorsRuleName = context->OperatorsRuleName();
    auto operatorsName = context->OperatorsName();
    auto operatorsWithSemicolonsName = context->OperatorsWithSemicolonsName();

    auto whileExitStatement = controller->addRule("WhileExitStatement", {
        BackusRuleItem({ ExitWhile::Type() }, OnlyOne),
        BackusRuleItem({ While::Type() }, OnlyOne)
    });
}

```

```

});

auto whileContinueStatement = controller->addRule("WhileContinueStatement", {
    BackusRuleItem({ ContinueWhile::Type()}, OnlyOne),
    BackusRuleItem({ While::Type()}, OnlyOne)
});

auto whileCStatement = controller->addRule("WhileStatement", {
    BackusRuleItem({ While::Type()}, OnlyOne),
    BackusRuleItem({ Symbols::LBraket}, OnlyOne),
    BackusRuleItem({ context->EquationRuleName()}, OnlyOne),
    BackusRuleItem({ Symbols::RBraket}, OnlyOne),
    BackusRuleItem({ Start::Type()}, OnlyOne),
    BackusRuleItem({ operatorsName,
operatorsWithSemicolonsName,whileContinueStatement->type(), whileExitStatement->type()}, Optional | OneOrMore),
    BackusRuleItem({ End::Type()}, OnlyOne),
    BackusRuleItem({ While::Type()}, OnlyOne),
});

whileCStatement->setPostHandler([](BackusRuleList::iterator& ruleBegin,
    BackusRuleList::iterator& it,
    BackusRuleList::iterator& end)
{
    static size_t index = 0;
    index++;

    std::string startLabel = std::format("whileStart{}", index);
    std::string endLabel = std::format("whileEnd{}", index);

    (*ruleBegin)->setCustomData(startLabel, "startLabel");
    (*ruleBegin)->setCustomData(endLabel, "endLabel");

    size_t count = 0;
    for (auto itr = ruleBegin; itr != it; ++itr)
    {
        auto type = (*itr)->type();

        if (type == lEnd && itr != it && (*std::next(itr, 1))->type() ==
While::Type())
        {
            (*std::next(itr, 1))->setCustomData("true", "noGenerateCode");
        }

        if (type == lStart)
        {
            count++;
        }
        else if (type == lEnd && count == 1)
        {
            (*itr)->setCustomData(std::format("\tjmp {} \n{}", startLabel,
endLabel));
            break;
        }
        else if (type == ExitWhile::Type() && count == 1 && itr != it &&
(*std::next(itr, 1))->type() == While::Type())
        {
            itr++;
            (*itr)->setCustomData("true", "ExitWhile");
            (*itr)->setCustomData(endLabel, "endLabel");
        }
        else if (type == ContinueWhile::Type() && count == 1 && itr != it &&
(*std::next(itr, 1))->type() == While::Type())
        {

```

```

        itr++;
        (*itr)->setCustomData("true", "ContinueWhile");
        (*itr)->setCustomData(startLabel, "startLabel");
    }
    else if (type == lEnd && count > 0)
    {
        count--;
    }
}
});

return whileCStatement;
}

```

Tokens:

Comment.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Generator/GeneratorItemBase.h"

class Comment : public TokenBase<Comment>, public GeneratorItemBase<Comment>
{
    BASE_ITEM

public:
    Comment() { setLexeme(""); };
    virtual ~Comment() = default;

    std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const override
    {
        auto token = clone();
        token->setValue(lexeme);
        lexeme.clear();
        return token;
    };
};

```

KWords:

Program.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Program : public TokenBase<Program>, public BackusRuleBase<Program>, public
GeneratorItemBase<Program>
{
    BASE_ITEM

public:
    Program() { setLexeme("NAME"); };
    virtual ~Program() = default;
};

```

Vars.h

```

#pragma once
#include "stdafx.h"

```

```

#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Vars : public TokenBase<Vars>, public BackusRuleBase<Vars>, public
GeneratorItemBase<Vars>
{
    BASE_ITEM

public:
    Vars() { setLexeme("DATA"); };
    virtual ~Vars() = default;
};

```

General:

EndOfFile.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class EndOfFile : public TokenBase<EndOfFile>, public BackusRuleBase<EndOfFile>,
public GeneratorItemBase<EndOfFile>
{
    BASE_ITEM

public:
    EndOfFile() { setLexeme(""); };
    virtual ~EndOfFile() = default;
};

```

Rules:

AssignmentRules:

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Assignment : public TokenBase<Assignment>, public BackusRuleBase<Assignment>,
public GeneratorItemBase<Assignment>
{
    BASE_ITEM

public:
    Assignment() { setLexeme("<-"); };
    virtual ~Assignment() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        auto ident = *std::prev(it);
        it++;
        auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

        auto postIt = postForm.begin();
        auto postEnd = postForm.end();
        for (const auto& item : postForm)

```

```

        item->genCode(out, details, postIt, postEnd);
    out << "\tpop " << ident->customData() << std::endl;
};
};

#include "stdafx.h"
#include "AssignmentRule.h"

#include "Rules/AssignmentRule/Assignment.h"

BackusRulePtr MakeAssignmentRule(std::shared_ptr<Controller> controller)
{
    controller->regItem<Assignment>();

    auto context = controller->context();

    auto assingmentRule = controller->addRule(context->AssignmentRuleName(), {
        BackusRuleItem({ context->IdentRuleName()}, OnlyOne),
        BackusRuleItem({ Assignment::Type()}, OnlyOne),
        BackusRuleItem({ context->EquationRuleName()}, OnlyOne)
    });

    return assingmentRule;
}

```

EquationRules:

Arithmetic:

Addition.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Addition : public TokenBase<Addition>, public BackusRuleBase<Addition>, public
GeneratorItemBase<Addition>
{
    BASE_ITEM

public:
    Addition() { setLexeme("ADD"); };
    virtual ~Addition() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\tcall Add_\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Add_", PrintAdd);
            SetRegistered();
        }
    }

private:

```

```

static void PrintAdd(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
{
    out << "===Procedure
Add=====\\n";
    out << "Add_ PROC\\n";
    out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
    out << "\\tadd " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
    GeneratorUtils::PrintResultToStack(out, args);
    out << "\\tret\\n";
    out << "Add_ ENDP\\n";
    out <<
";=====\\n";
}
};

```

Subtraction.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Subtraction : public TokenBase<Subtraction>, public BackusRuleBase<Subtraction>,
public GeneratorItemBase<Subtraction>
{
    BASE_ITEM

public:
    Subtraction() { setLexeme("SUB"); };
    virtual ~Subtraction() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\\tcall Sub_\\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Sub_", PrintSub);
            SetRegistered();
        }
    }

private:
    static void PrintSub(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << "===Procedure
Sub=====\\n";
        out << "Sub_ PROC\\n";
        out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
        out << "\\tsub " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\\tret\\n";
        out << "Sub_ ENDP\\n";
        out <<
";=====\\n";
    }
};

```



```

}
};

```

Compare:

Equal.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Equal : public TokenBase<Equal>, public BackusRuleBase<Equal>, public
GeneratorItemBase<Equal>
{
    BASE_ITEM

public:
    Equal() { setLexeme("EQ"); };
    virtual ~Equal() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\tcall Equal_\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Equal_", PrintEqual);
            SetRegistered();
        }
    }

private:
    static void PrintEqual(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << "===Procedure
Equal=====\\n";
        out << "Equal_ PROC\\n";
        out << "\tpushf\\n";
        out << "\tpop cx\\n\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
        out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
        out << "\tjne equal_false\\n";
        out << "\tmov " << args.regPrefix << "ax, 1\\n";
        out << "\tjmp equal_fin\\n";
        out << "equal_false:\\n";
        out << "\tmov " << args.regPrefix << "ax, 0\\n";
        out << "equal_fin:\\n";
        out << "\tpush cx\\n";
        out << "\tpopf\\n\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\\n";
        out << "Equal_ ENDP\\n";
        out <<
";=====\\n";
    }
};

```

Greate.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Greate : public TokenBase<Greate>, public BackusRuleBase<Greate>, public
GeneratorItemBase<Greate>
{
    BASE_ITEM

public:
    Greate() { setLexeme(">="); };
    virtual ~Greate() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\tcall Greate_\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Greate_", PrintGreate);
            SetRegistered();
        }
    }

private:
    static void PrintGreate(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << ";;;Procedure
Greate=====\\n";
        out << "Greate_ PROC\\n";
        out << "\tpushf\\n";
        out << "\tpop cx\\n\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
        out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
        out << "\tjle greate_false\\n";
        out << "\tmov " << args.regPrefix << "ax, 1\\n";
        out << "\tjmp greate_fin\\n";
        out << "greate_false:\\n";
        out << "\tmov " << args.regPrefix << "ax, 0\\n";
        out << "greate_fin:\\n";
        out << "\tpush cx\\n";
        out << "\tpopf\\n\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\\n";
        out << "Greate_ ENDP\\n";
        out <<
"=====\\n";
        }
    };
};
```

Less.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Less : public TokenBase<Less>, public BackusRuleBase<Less>, public
GeneratorItemBase<Less>
{
    BASE_ITEM

public:
    Less() { setLexeme("<="); };
    virtual ~Less() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\tcall Less_\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Less_", PrintLess);
            SetRegistered();
        }
    }

private:
    static void PrintLess(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << ";===Procedure
Less=====\\n";
        out << "Less_ PROC\\n";
        out << "\tpushf\\n";
        out << "\tpop cx\\n\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
        out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
        out << "\tjge less_false\\n";
        out << "\tmov " << args.regPrefix << "ax, 1\\n";
        out << "\tjmp less_fin\\n";
        out << "less_false:\\n";
        out << "\tmov " << args.regPrefix << "ax, 0\\n";
        out << "less_fin:\\n";
        out << "\tpush cx\\n";
        out << "\tpopf\\n\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\\n";
        out << "Less_ ENDP\\n";
        out <<
";=====\\n";
        =====\\n";

    }
};
```

NotEqual.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Rules/EquationRule/Equal.h"
#include "Rules/EquationRule/Not.h"

class NotEqual : public TokenBase<NotEqual>, public BackusRuleBase<NotEqual>, public
GeneratorItemBase<NotEqual>
{
    BASE_ITEM

public:
    NotEqual() { setLexeme("NE"); };
    virtual ~NotEqual() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        Equal::RegPROC(details);
        Not::RegPROC(details);
        out << "\tcall Equal_\n";
        out << "\tcall Not_\n";
    };
};
```

Logic:

And.h

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class And : public TokenBase<And>, public BackusRuleBase<And>, public
GeneratorItemBase<And>
{
    BASE_ITEM

public:
    And() { setLexeme("AND"); };
    virtual ~And() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\tcall And_\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("And_", PrintAnd);
            SetRegistered();
        }
    }
};
```

```

private:
    static void PrintAnd(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << ";;==Procedure
And=====\\n";
        out << "And_ PROC\\n";
        out << "\\tpushf\\n";
        out << "\\tpop cx\\n\\n";
        out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
        out << "\\tcmp " << args.regPrefix << "ax, 0\\n";
        out << "\\tjnz and_t1\\n";
        out << "\\tjz and_false\\n";
        out << "and_t1:\\n";
        out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
        out << "\\tcmp " << args.regPrefix << "ax, 0\\n";
        out << "\\tjnz and_true\\n";
        out << "and_false:\\n";
        out << "\\tmov " << args.regPrefix << "ax, 0\\n";
        out << "\\tjmp and_fin\\n";
        out << "and_true:\\n";
        out << "\\tmov " << args.regPrefix << "ax, 1\\n";
        out << "and_fin:\\n";
        out << "\\tpush cx\\n";
        out << "\\tpopf\\n\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\\tret\\n";
        out << "And_ ENDP\\n";
        out <<
";=====\\n";
    }
};

```

Not.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Not : public TokenBase<Not>, public BackusRuleBase<Not>, public
GeneratorItemBase<Not>
{
    BASE_ITEM

public:
    Not() { setLexeme("NOT"); };
    virtual ~Not() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\\tcall Not_\\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Not_", PrintNot);
            SetRegistered();
        }
    }
}

```

```

    }

private:
    static void PrintNot(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << ";===Procedure
Not=====\\n";
        out << "Not_ PROC\\n";
        out << "\\tpushf\\n";
        out << "\\tpop cx\\n\\n";
        out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
        out << "\\tcmp " << args.regPrefix << "ax, 0\\n";
        out << "\\tjnz not_false\\n";
        out << "not_t1:\\n";
        out << "\\tmov " << args.regPrefix << "ax, 1\\n";
        out << "\\tjmp not_fin\\n";
        out << "not_false:\\n";
        out << "\\tmov " << args.regPrefix << "ax, 0\\n";
        out << "not_fin:\\n";
        out << "\\tpush cx\\n";
        out << "\\tpopf\\n\\n";
        out << "\\tmov [esp + " << args.posArg1 << "], " << args.regPrefix << "ax\\n";
        out << "\\tret\\n";
        out << "Not_ ENDP\\n";
        out <<
        ";=====\\n";
    }
};

```

Or.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Or : public TokenBase<Or>, public BackusRuleBase<Or>, public
GeneratorItemBase<Or>
{
    BASE_ITEM

public:
    Or() { setLexeme("OR"); };
    virtual ~Or() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\\tcall Or_\\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Or_", PrintOr);
            SetRegistered();
        }
    }

private:

```

```

static void PrintOr(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
{
    out << ";;==Procedure
Or=====\\n";
    out << "Or_ PROC\\n";
    out << "\\tpushf\\n";
    out << "\\tpop cx\\n\\n";
    out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
    out << "\\tcmp " << args.regPrefix << "ax, 0\\n";
    out << "\\tjnz or_true\\n";
    out << "\\tjz or_t1\\n";
    out << "or_t1:\\n";
    out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
    out << "\\tcmp " << args.regPrefix << "ax, 0\\n";
    out << "\\tjnz or_true\\n";
    out << "or_false:\\n";
    out << "\\tmov " << args.regPrefix << "ax, 0\\n";
    out << "\\tjmp or_fin\\n";
    out << "or_true:\\n";
    out << "\\tmov " << args.regPrefix << "ax, 1\\n";
    out << "or_fin:\\n";
    out << "\\tpush cx\\n";
    out << "\\tpopf\\n\\n";
    GeneratorUtils::PrintResultToStack(out, args);
    out << "\\tret\\n";
    out << "Or_ ENDP\\n";
    out <<
";=====\\n";
}
};

```

Mult:

Division.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Division : public TokenBase<Division>, public BackusRuleBase<Division>, public
GeneratorItemBase<Division>
{
    BASE_ITEM

public:
    Division() { setLexeme("DIV"); };
    virtual ~Division() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\\tcall Div_\\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {

```

```

        details.registerStringData("DivErrMsg", "\\n" + Type() + ": Error:
division by zero");
        details.registerProc("Div_", PrintDiv);
        SetRegistered();
    }
}

private:
    static void PrintDiv(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << "===Procedure
Div=====\\n";
        out << "Div_ PROC\\n";
        out << "\tpushf\\n";
        out << "\tpop cx\\n\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]"\\n";
        out << "\tcmp " << args.regPrefix << "ax, 0\\n";
        out << "\tjne end_check\\n";
        out << "\tinvoke WriteConsoleA, hConsoleOutput, ADDR DivErrMsg, SIZEOF
DivErrMsg - 1, 0, 0\\n";
        out << "\tjmp exit_label\\n";
        out << "end_check:\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]"\\n";
        out << "\tcmp " << args.regPrefix << "ax, 0\\n";
        out << "\tjge gr\\n";
        out << "lo:\\n";
        out << "\tmov " << args.regPrefix << "dx, -1\\n";
        out << "\tjmp less_fin\\n";
        out << "gr:\\n";
        out << "\tmov " << args.regPrefix << "dx, 0\\n";
        out << "less_fin:\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]"\\n";
        out << "\tidiv " << args.numberTypeExtended << " ptr [esp + " << args.posArg1
<< "]"\\n";
        out << "\tpush cx\\n";
        out << "\tpopf\\n\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\\n";
        out << "Div_ ENDP\\n";
        out <<
";=====\\n";
    }
};

```

Mod.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Mod : public TokenBase<Mod>, public BackusRuleBase<Mod>, public
GeneratorItemBase<Mod>
{
    BASE_ITEM

public:
    Mod() { setLexeme("MOD"); };
    virtual ~Mod() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final

```



```

{
    RegPROC(details);
    out << "\tcall Mod_\n";
};

static void RegPROC(GeneratorDetails& details)
{
    if (!IsRegistered())
    {
        details.registerStringData("ModErrMsg", "\\n" + Type() + ": Error:
division by zero");
        details.registerProc("Mod_", PrintMod);
        SetRegistered();
    }
}

private:
static void PrintMod(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
{
    out << "===Procedure
Mod=====\\n";
    out << "Mod_ PROC\\n";
    out << "\tpushf\\n";
    out << "\tpop cx\\n\\n";
    out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]"\\n";
    out << "\tcmp " << args.regPrefix << "ax, 0\\n";
    out << "\tjne end_check\\n";
    out << "\tinvoke WriteConsoleA, hConsoleOutput, ADDR ModErrMsg, SIZEOF
ModErrMsg - 1, 0, 0\\n";
    out << "\tjmp exit_label\\n";
    out << "end_check:\\n";
    out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]"\\n";
    out << "\tcmp " << args.regPrefix << "ax, 0\\n";
    out << "\tjge gr\\n";
    out << "lo:\\n";
    out << "\tmov " << args.regPrefix << "dx, -1\\n";
    out << "\tjmp less_fin\\n";
    out << "gr:\\n";
    out << "\tmov " << args.regPrefix << "dx, 0\\n";
    out << "less_fin:\\n";
    out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]"\\n";
    out << "\tidiv " << args.numberTypeExtended << " ptr [esp + " << args.posArg1
<< "]"\\n";
    out << "\tmov " << args.regPrefix << "ax, " << args.regPrefix << "dx\\n";
    out << "\tpush cx\\n";
    out << "\tpopf\\n\\n";
    GeneratorUtils::PrintResultToStack(out, args);
    out << "\tret\\n";
    out << "Mod_ ENDP\\n";
    out <<
    "=====\\n";
}
};

```

Multiplication.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Multiplication : public TokenBase<Multiplication>, public
BackusRuleBase<Multiplication>, public GeneratorItemBase<Multiplication>

```

```

{
    BASE_ITEM

public:
    Multiplication() { setLexeme("MUL"); };
    virtual ~Multiplication() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "\tcall Mul_\n";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerProc("Mul_", PrintMul);
            SetRegistered();
        }
    }

private:
    static void PrintMul(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << "===Procedure
Mul=====\\n";
        out << "Mul_ PROC\\n";
        out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
        out << "\timul " << args.numberTypeExtended << " ptr [esp + " << args.posArg1
<< "]\\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\\n";
        out << "Mul_ ENDP\\n";
        out <<
";=====\\n";
    }
};

```

EquationRule.cpp

```

#include "stdafx.h"
#include "EquationRule.h"

#include "Rules/EquationRule/Number.h"

#include "Rules/EquationRule/Addition.h"
#include "Rules/EquationRule/Subtraction.h"

#include "Rules/EquationRule/Multiplication.h"
#include "Rules/EquationRule/Division.h"
#include "Rules/EquationRule/Mod.h"

#include "Rules/EquationRule/And.h"
#include "Rules/EquationRule/Or.h"

#include "Rules/EquationRule/Equal.h"
#include "Rules/EquationRule/Greate.h"
#include "Rules/EquationRule/Less.h"
#include "Rules/EquationRule/NotEqual.h"

#include "Rules/EquationRule/Not.h"

```

```

BackusRulePtr MakeEquationRule(std::shared_ptr<Controller> controller)
{
    using enum ItemType;

    controller->regItem<Number>(TokenAndRule | Operand, 0);

    controller->regItem      <Addition>(TokenAndRule | Operation, 4);
    controller->regItem      <Subtraction>(TokenAndRule | Operation, 4);

    controller->regItem<Multiplication>(TokenAndRule | Operation, 5);
    controller->regItem      <Division>(TokenAndRule | Operation, 5);
    controller->regItem      <Mod>(TokenAndRule | Operation, 5);

    controller->regItem      <And>(TokenAndRule | Operation, 1);
    controller->regItem      <Or>(TokenAndRule | Operation, 1);

    controller->regItem      <Equal>(TokenAndRule | Operation, 2);
    controller->regItem      <NotEqual>(TokenAndRule | Operation, 2);
    controller->regItem      <Greate>(TokenAndRule | Operation, 3);
    controller->regItem      <Less>(TokenAndRule | Operation, 3);

    controller->regItem      <Not>(TokenAndRule | Operation, 6);

    auto context = controller->context();
    auto equationRuleName = context->EquationRuleName();

    auto sign = controller->addRule("Sign", { BackusRuleItem({ Symbols::Plus,
Symbols::Minus }, Optional) });
    auto signedNumber = controller->addRule("SignedNumber", {
        BackusRuleItem({ sign->type()}, Optional),
        BackusRuleItem({Number::Type()}, OnlyOne)
    });

    signedNumber->setPostHandler([](BackusRuleList::iterator&,
BackusRuleList::iterator& it,
BackusRuleList::iterator& end)
    {
        auto begRuleIt = std::prev(it, 2);
        if ((*begRuleIt)->type() == Symbols::Minus)
        {
            it = begRuleIt;
            end = std::remove(it, end, *it);
            (*it)->setValue('-' + (*it)->value());
            it++;
        }
    });

    auto arithmetic = controller->addRule("Arithmetic", { BackusRuleItem({
Addition::Type(), Subtraction::Type() }, OnlyOne) });
    auto mult = controller->addRule("Mult", { BackusRuleItem({
Multiplication::Type(), Division::Type(), Mod::Type() }, OnlyOne) });
    auto logic = controller->addRule("Logic", { BackusRuleItem({ And::Type(),
Or::Type() }, OnlyOne) });
    auto compare = controller->addRule("Compare", { BackusRuleItem({ Equal::Type(),
Greate::Type(), Less::Type(), NotEqual::Type() }, OnlyOne) });

    auto operationAndEquation = controller->addRule("OperationAndEquation", {
        BackusRuleItem({ mult->type(), arithmetic->type(), logic->type(), compare-
>type() }, OnlyOne),
        BackusRuleItem({ equationRuleName }, OnlyOne)
    });

    auto notRule = controller->addRule("NotRule", {
        BackusRuleItem({ Not::Type()}, OnlyOne),
        BackusRuleItem({ equationRuleName}, Optional | OneOrMore)
    });
}

```

```

});

auto equationWithBrackets = controller->addRule("EquationWithBrackets", {
    BackusRuleItem({ Symbols::LBracket }, OnlyOne | PairStart),
    BackusRuleItem({ equationRuleName }, OnlyOne),
    BackusRuleItem({ Symbols::RBracket }, OnlyOne | PairEnd)
});

auto equation = controller->addRule(equationRuleName, {
    BackusRuleItem({signedNumber->type(), context->IdentRuleName(), notRule-
>type(), equationWithBrackets->type()}, OnlyOne),
    BackusRuleItem({operationAndEquation->type()}, Optional | OneOrMore)
});

return equation;
}

```

IdentRule:

Identifier.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Rules/AssignmentRule/Assignment.h"
#include "Tokens/Common/EndOfFile.h"

class Identifier : public TokenBase<Identifier>, public BackusRuleBase<Identifier>,
public GeneratorItemBase<Identifier>
{
    BASE_ITEM

public:
    Identifier() { setLexeme(""); };
    virtual ~Identifier() = default;

    std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const override
    {
        if (lexeme.size() > (m_mask.size() + m_prefix.size()))
            return nullptr;

        bool res = true;
        if (!lexeme.starts_with(m_prefix))
        {
            return nullptr;
        }

        std::string_view ident{ lexeme.begin() + m_prefix.size(), lexeme.end() };
        for (size_t i = 0; i < ident.size(); i++)
        {
            if ((isupper(ident[i]) != isupper(m_mask[i])) && !isdigit(ident[i]))
            {
                res &= false;
                break;
            }
        }

        std::shared_ptr<IToken> token = nullptr;
        if (res)
        {
            token = clone();
            token->setValue(lexeme);
            lexeme.clear();
        }
    }
}

```

```

    }

    return token;
};

void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
{
    if (!GeneratorUtils::IsNextTokenIs(it, end, Assignment::Type()))
    {
        if ((*std::prev(end))->type() == EndOfFile::Type())
            details.registerNumberData(customData());
        else
            out << "\tpush " << customData() << std::endl;
    }
};

private:
    const std::string m_prefix = "_";
    const std::string m_mask = "XXXXXX";
};

```

IdentRule.cpp

```

#include "stdafx.h"
#include "IdentRule.h"

#include "Rules/IdentRule/Identifier.h"
#include "Rules/IdentRule/Undefined.h"

SimpleToken(ProgramName, "");

BackusRulePtr MakeIdentRule(std::shared_ptr<Controller> controller)
{
    using enum ItemType;

    controller->regItem<Identifier>(TokenAndRule, -1);

    GeneratorUtils::Instance()->RegisterOperand(Identifier::Type());

    auto context = controller->context();

    auto identRule = controller->addRule(context->IdentRuleName(), {
        BackusRuleItem({ Identifier::Type()}, OnlyOne)
    });

    identRule->setPostHandler([context](BackusRuleList::iterator&,
        BackusRuleList::iterator& it,
        BackusRuleList::iterator& end)
    {
        static bool isFirstIdentChecked = !context->IsFirstProgName();
        auto isVarBlockChecked = context->IsVarBlockChecked();
        auto& identTable = context->IdentTable();

        auto identIt = std::prev(it, 1);
        if (isVarBlockChecked)
        {
            if (!identTable.contains((*identIt)->value()))
            {
                auto undef = std::make_shared<Undefined>();
                undef->setValue((*identIt)->value());
                undef->setLine((*identIt)->line());
                undef->setCustomData((*identIt)->customData());
                *identIt = undef;
            }
        }
    });
}

```

```

    }
    else
    {
        if (isFirstIdentChecked)
        {
            identTable.insert((*identIt)->value());
        }
        else
        {
            auto progName = std::make_shared<ProgramName>();
            progName->setValue((*identIt)->value());
            progName->setLine((*identIt)->line());
            progName->setCustomData((*identIt)->customData());
            *identIt = progName;
            isFirstIdentChecked = true;
        }
    }

    (*identIt)->setCustomData((*identIt)->value() + "_");
});

return identRule;
}

```

Undefined.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Undefined : public TokenBase<Undefined>, public BackusRuleBase<Undefined>,
public GeneratorItemBase<Undefined>
{
    BASE_ITEM

public:
    Undefined() { setLexeme(""); };
    virtual ~Undefined() = default;

    std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const override
    {
        auto token = clone();
        token->setValue(lexeme);
        lexeme.clear();
        return token;
    };
};

```

ReadRule:

Read.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class Read : public TokenBase<Read>, public BackusRuleBase<Read>, public
GeneratorItemBase<Read>
{
    BASE_ITEM

public:
    Read() { setLexeme("SCAN"); };
};

```

```

virtual ~Read() = default;

void genCode(std::ostream& out, GeneratorDetails& details,
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
{
    RegPROC(details);

    it = std::next(it, 2);

    out << "\tcall Input_\n";
    out << "\tmov " << (*it)->customData() << ", " << details.args().regPrefix <<
"ax\n";

    it = std::next(it, 2);
};

static void RegPROC(GeneratorDetails& details)
{
    if (!IsRegistered())
    {
        details.registerRawData("InputBuf", "\tdb\t15 dup (?)");
        details.registerRawData("CharsReadNum", "dd\t?");
        details.registerProc("Input_", PrintInput);
        SetRegistered();
    }
}

private:
static void PrintInput(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
{
    out << ";;;Procedure
Input=====\\n";
    out << "Input_ PROC\\n";
    out << "\tinvoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR
CharsReadNum, 0\\n";
    out << "\tinvoke crt_atoi, ADDR InputBuf\\n";
    out << "\tret\\n";
    out << "Input_ ENDP\\n";
    out <<
";=====\\n";
}
};

```

ReadRule.cpp

```

#include "stdafx.h"
#include "ReadRule.h"

#include "Rules/ReadRule/Read.h"

BackusRulePtr MakeReadRule(std::shared_ptr<Controller> controller)
{
    controller->regItem<Read>();

    auto context = controller->context();

    auto read = controller->addRule("ReadRule", {
        BackusRuleItem({ Read::Type(), OnlyOne),
        BackusRuleItem({ Symbols::LBracket, OnlyOne),
        BackusRuleItem({ context->IdentRuleName(), OnlyOne),
        BackusRuleItem({ Symbols::RBracket, OnlyOne)
    });
}

```

```

    return read;
}

```

StringRule:

String.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class String : public TokenBase<String>, public BackusRuleBase<String>, public
GeneratorItemBase<String>
{
    BASE_ITEM

public:
    String() { setLexeme(""); };
    virtual ~String() = default;

    std::string stringName() const { return m_stringName; };

    std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const override
    {
        auto token = clone();
        token->setValue(lexeme);
        lexeme.clear();
        return token;
    };

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        m_stringName = std::format("String_{}", index++);
        details.registerStringData(m_stringName, value());
    };

private:
    mutable std::string m_stringName;
    static size_t index;
};

```

StringRule.cpp

```

#include "stdafx.h"
#include "StringRule.h"

#include "Rules/StringRule/String.h"

SimpleToken(Quotes, "\\");

BackusRulePtr MakeStringRule(std::shared_ptr<Controller> controller)
{
    using enum ItemType;

    controller->regUnchangedTextToken(std::make_shared<String>(),
std::make_shared<Quotes>(), std::make_shared<Quotes>());
    controller->regItem<Quotes>(Rule);
    controller->regItem<String>(Rule);

    auto stringRule = controller->addRule("StringRule", {
        BackusRuleItem({ Quotes::Type(), OnlyOne),
        BackusRuleItem({ String::Type(), OnlyOne),
        BackusRuleItem({ Quotes::Type(), OnlyOne)

```



```

});

stringRule->setPostHandler([](BackusRuleList::iterator&,
    BackusRuleList::iterator& it,
    BackusRuleList::iterator& end)
{
    it = std::prev(it, 3);
    end = std::remove(it, end, *it);
    it++;
    end = std::remove(it, end, *it);
});

return stringRule;
}

```

VarsBlockRule:

VarsBlockRule.cpp

```

#include "stdafx.h"
#include "VarsBlokRule.h"

#include "Rules/VarsBlokRule/VarType.h"

BackusRulePtr MakeVarsBlokRule(std::shared_ptr<Controller> controller)
{
    controller->regItem<VarType>();

    auto context = controller->context();

    auto commaAndIdentifier = controller->addRule("CommaAndIdentifier", {
        BackusRuleItem({ Symbols::Comma}, OnlyOne),
        BackusRuleItem({ context->IdentRuleName()}, OnlyOne)
    });

    auto varsBlok = controller->addRule("VarsBlok", {
        BackusRuleItem({ VarType::Type()}, OnlyOne),
        BackusRuleItem({ context->IdentRuleName()}, OnlyOne),
        BackusRuleItem({ commaAndIdentifier->type()}, Optional | OneOrMore),
        BackusRuleItem({ Symbols::Semicolon}, OnlyOne)
    });

    varsBlok->setPostHandler([context](BackusRuleList::iterator&,
        BackusRuleList::iterator&, BackusRuleList::iterator&)
    {
        auto isVarBlockChecked = context->IsVarBlockChecked();

        context->SetVarBlockChecked();
    });

    return varsBlok;
}

```

VarType.h

```

#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"

class VarType : public TokenBase<VarType>, public BackusRuleBase<VarType>, public
GeneratorItemBase<VarType>
{

```

```

    BASE_ITEM

public:
    VarType() { setLexeme("INTEGER_2"); };
    virtual ~VarType() = default;
};

WriteRule:
Write.h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Rules/StringRule/String.h"

class Write : public TokenBase<Write>, public BackusRuleBase<Write>, public
GeneratorItemBase<Write>
{
    BASE_ITEM

public:
    Write() { setLexeme("PRINT"); };
    virtual ~Write() = default;

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end) const final
    {
        if (auto string = std::dynamic_pointer_cast<String>(*std::next(it, 2)))
        {
            it = std::next(it, 2);
            string->genCode(out, details, it, end);
            it = std::next(it, 2);

            out << "\tinvoke WriteConsoleA, hConsoleOutput, ADDR " << string-
>stringName() << ", SIZEOF " << string->stringName() << " - 1, 0, 0\n";
        }
        else
        {
            RegPROC(details);

            auto postForm = GeneratorUtils::Instance()->ConvertToPostfixForm(it, end);

            auto postIt = postForm.begin();
            auto postEnd = postForm.end();
            for (const auto& item : postForm)
                item->genCode(out, details, postIt, postEnd);

            out << "\tcall Output_\n";
        }
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerRawData("OutMessage", "\tdb\t\" +
details.args().numberStrType + "\", 0");
            details.registerRawData("ResMessage", "\tdb\t20 dup (?");
            details.registerProc("Output_", PrintOutput);
            SetRegistered();
        }
    }
};

```

```

    }

private:
    static void PrintOutput(std::ostream& out, const GeneratorDetails::GeneratorArgs&
args)
    {
        out << "===Procedure
Output=====\\n";
        out << "Output_ PROC value: " << args.numberTypeExtended.c_str() << std::endl;
        out << "\\tinvoke wsprintf, ADDR ResMessage, ADDR OutMessage, value\\n";
        out << "\\tinvoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0\\n";
        out << "\\tret " << args.argSize << std::endl;
        out << "Output_ ENDP\\n";
        out <<
        "=====\\n";
    }
};

```

WriteRule.cpp

```

#include "stdafx.h"
#include "WriteRule.h"

#include "Rules/StringRule/StringRule.h"

#include "Rules/WriteRule/Write.h"

BackusRulePtr MakeWriteRule(std::shared_ptr<Controller> controller)
{
    controller->regItem<Write>();

    auto context = controller->context();

    auto stringRule = MakeStringRule(controller);

    auto write = controller->addRule("WriteRule", {
        BackusRuleItem({ Write::Type()}, OnlyOne),
        BackusRuleItem({ Symbols::LBraket}, OnlyOne | PairStart),
        BackusRuleItem({ stringRule->type(), context->EquationRuleName() }, OnlyOne),
        BackusRuleItem({ Symbols::RBraket}, OnlyOne | PairEnd)
    });

    return write;
}

```

Controller.h

```

#pragma once
#include "stdafx.h"
#include "Utils/singleton.hpp"
#include "Core/Tokens/TokenBase.hpp"
#include "Core/Backus/BackusRule.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Core/Generator/GeneratorDetails.h"

#include "Symbols.h"

using BackusRulePtr = std::shared_ptr<IBackusRule>;
using BackusRuleList = std::list<BackusRulePtr>;
using BackusRuleListIt = BackusRuleList::iterator;
using RuleMaker = std::function<BackusRulePtr(std::shared_ptr<Controller>)>;

class Context
{
public:

```

```

std::string IdentRuleName() const { return "IdentRule"; }
std::string EquationRuleName() const { return "Equation"; }
std::string OperatorsRuleName() const { return "OperatorsRule"; }
std::string OperatorsName() const { return "Operators"; }
std::string OperatorsWithSemicolonsName() const { return "OperatorsWithSemicolon"; }
}

std::string AssignmentRuleName() const { return "AssignmentRule"; }
std::tuple<std::string, std::string, std::string> CodeBlockTypes() const { return
{ "Start", "CodeBlok", "End" }; }

bool IsVarBlockChecked() const { return m_isVarBlockChecked; }
void SetVarBlockChecked() { m_isVarBlockChecked = true; }

bool IsFirstProgName() const { return true; }

std::set<std::string>& IdentTable() { return m_identTable; }

const GeneratorDetails& Details() const { return m_details; }

private:
std::set<std::string> m_identTable{};
bool m_isVarBlockChecked = false;

const GeneratorDetails m_details{ {
    .numberType = "dw", .numberTypeExtended = "word",
    .argSize = 2,
    .numberStrType = "%hd"
} } };

};

enum class ItemType : uint32_t
{
    None = 0,
    Token = 1 << 0,
    Rule = 1 << 1,
    TokenAndRule = Token | Rule,
    Operand = 1 << 2,
    Operation = 1 << 3,
    EquationEnd = 1 << 4,
    LBracket = 1 << 5,
    RBracket = 1 << 6
};
DEFINE_ENUM_FLAG_OPERATORS(ItemType)

class Controller : public singleton<Controller>
{
public:
    static constexpr int NoPriority = std::numeric_limits<int>::min();

public:
    void init();

    template<typename T>
    void regItem(ItemType type = ItemType::TokenAndRule, int priority = NoPriority)
const
    {
        auto item = std::make_shared<T>();
        regItem(item, item, type, priority);
    }

    void regUnchangedTextToken(std::shared_ptr<IToken> target, std::shared_ptr<IToken>
lBorder, std::shared_ptr<IToken> rBorder) const;

    void regOperatorRule(const RuleMaker& rule, bool isNeedSemicolon = false);

```

```

    std::shared_ptr<IBackusRule> addRule(const std::string& name, const
std::list<BackusRuleItem>& items) const;

    BackusRulePtr topRule();

    std::shared_ptr<Context> context() { return m_context; }

protected:
    Controller() { m_context = std::make_shared<Context>(); }

    void regItem(std::shared_ptr<IToken> token, std::shared_ptr<IBackusRule> rule,
Itemtype type, int priority) const;

    BackusRulePtr MakeTopRule(std::shared_ptr<Controller> controller) const;

private:
    BackusRulePtr m_topRule;
    std::set<std::string> m_operatorRuleNames;
    std::set<std::string> m_operatorRuleWithSemicolonNames;

    std::shared_ptr<Context> m_context;
};

```

Controller.cpp

```

#include "stdafx.h"
#include "Controller.h"
#include "Core/Parser/TokenParser.h"
#include "Core/Backus/BackusRuleStorage.h"
#include "Core/Generator/Generator.h"
#include "Core/Generator/GeneratorUtils.h"

#include "SimpleTokens.h"

#include "Tokens/Common/Program.h"
#include "Tokens/Common/Vars.h"

#include "Rules/IdentRule/IdentRule.h"
#include "Rules/VarsBlokRule/VarsBlokRule.h"
#include "Rules/EquationRule/EquationRule.h"
#include "Rules/ReadRule/ReadRule.h"
#include "Rules/WriteRule/WriteRule.h"
#include "Rules/AssignmentRule/AssignmentRule.h"

void Controller::regItem(std::shared_ptr<IToken> token, std::shared_ptr<IBackusRule>
rule, Itemtype type, int priority) const
{
    using enum Itemtype;

    if ((type & Token) == Token)
        TokenParser::Instance()->regToken(token, ((type & Operation) == Operation) ?
TokenParser::NoPriority : priority);

    if ((type & Rule) == Rule)
        BackusRuleStorage::Instance()->regRule(rule);

    auto tokenType = token->type();

    if ((type & Operand) == Operand)
        GeneratorUtils::Instance()->RegisterOperand(tokenType);

    if ((type & Operation) == Operation)
    {
        if (priority == TokenParser::NoPriority)
            throw std::runtime_error("Controller::RegItem: Operation " + token->type()
+ " priority is not set");
    }
}

```

```

        GeneratorUtils::Instance()->RegisterOperation(tokenType, priority);
    }

    if ((type & EquationEnd) == EquationEnd)
        GeneratorUtils::Instance()->RegisterEquationEnd(tokenType);

    if ((type & LBracket) == LBracket)
        GeneratorUtils::Instance()->RegisterLBraket(tokenType);

    if ((type & RBracket) == RBracket)
        GeneratorUtils::Instance()->RegisterRBraket(tokenType);
}

void Controller::init()
{
    m_topRule = MakeTopRule(Instance());

    Generator::Instance()->setDetails(context()->Details());
}

void Controller::regUnchangedTextToken(std::shared_ptr<IToken> target,
std::shared_ptr<IToken> lBorder, std::shared_ptr<IToken> rBorder) const
{
    TokenParser::Instance()->regUnchangedTextToken(target, lBorder, rBorder);
}

void Controller::regOperatorRule(const RuleMaker& rule, bool isNeedSemicolon)
{
    auto ruleName = rule(Instance())->type();

    if (ruleName.empty())
        throw std::runtime_error("Controller::RegOperatorRule: Rule name is empty");

    if (m_operatorRuleNames.contains(ruleName) ||
m_operatorRuleWithSemicolonNames.contains(ruleName))
        throw std::runtime_error(std::format("Controller::RegOperatorRule: Rule with
name {} already registered", ruleName));

    if (isNeedSemicolon)
        m_operatorRuleWithSemicolonNames.insert(ruleName);
    else
        m_operatorRuleNames.insert(ruleName);
}

std::shared_ptr<IBackusRule> Controller::addRule(const std::string& name, const
std::list<BackusRuleItem>& items) const
{
    auto rule = BackusRule::MakeRule(name, items);

    BackusRuleStorage::Instance()->regRule(rule);

    return rule;
}

BackusRulePtr Controller::topRule()
{
    if (!m_topRule)
        throw(std::runtime_error("Controller is not inited"));

    return m_topRule;
}

BackusRulePtr Controller::MakeTopRule(std::shared_ptr<Controller> controller) const
{
    using enum ItemType;

```

```

controller->regItem<Program>();
controller->regItem<Vars>();
controller->regItem<Start>(TokenAndRule | EquationEnd);
controller->regItem<End>();

controller->regItem<Comma>();
controller->regItem<Colon>();
controller->regItem<Semicolon>(TokenAndRule | EquationEnd);
controller->regItem<LBracket>(TokenAndRule | LBracket);
controller->regItem<RBracket>(TokenAndRule | RBracket);
controller->regItem<Plus>();
controller->regItem<Minus>();

auto identRule = MakeIdentRule(controller);
auto varsBlok = MakeVarsBlokRule(controller);
auto equation = MakeEquationRule(controller);
auto read = MakeReadRule(controller);
auto write = MakeWriteRule(controller);
auto assingmentRule = MakeAssignmentRule(controller);

auto operatorWithSemicolonTypes = std::vector<std::variant<std::string, Symbols>>{
read->type(), write->type(), assingmentRule->type() };
operatorWithSemicolonTypes.insert(operatorWithSemicolonTypes.end(),
m_operatorRuleWithSemicolonNames.begin(), m_operatorRuleWithSemicolonNames.end());
auto operatorsWithSemicolon = controller->addRule("OperatorsWithSemicolon", {
BackusRuleItem({ operatorWithSemicolonTypes }, OnlyOne),
BackusRuleItem({ Symbols::Semicolon }, OnlyOne)
});

auto operatorTypes = std::vector<std::variant<std::string, Symbols>>{
m_operatorRuleNames.begin(), m_operatorRuleNames.end() };
auto operators = controller->addRule("Operators", {
BackusRuleItem({ operatorTypes }, OnlyOne)
});

auto operatorsRule = controller->addRule("OperatorsRule", {
BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional
| OneOrMore),
});

auto codeBlok = controller->addRule("CodeBlok", {
BackusRuleItem({ Start::Type()}, OnlyOne),
BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional
| OneOrMore),
BackusRuleItem({ End::Type()}, OnlyOne)
});

auto topRule = controller->addRule("TopRule", {
BackusRuleItem({ Program::Type()}, OnlyOne),
BackusRuleItem({ identRule->type()}, OnlyOne),
BackusRuleItem({ Symbols::Semicolon}, OnlyOne),
BackusRuleItem({ Start::Type()}, OnlyOne),
BackusRuleItem({ Vars::Type()}, OnlyOne),
BackusRuleItem({ varsBlok->type()}, OnlyOne),
BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional
| OneOrMore),
BackusRuleItem({ End::Type()}, OnlyOne)
});

return topRule;
}

```

Додаток В (Код на мові Асемблер)

Prog1.asm

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User

Data=====

=====

aAAAAAA dw 0

bBBBBBB dw 0

cXXXXXX dw 0

dYYYYYY dw 0

DivErrMsg db 13, 10, "Division: Error: division by zero", 0

ModErrMsg db 13, 10, "Mod: Error: division by zero", 0

String_0 db "Input A: ", 0

String_1 db "Input B: ", 0

String_2 db "A + B: ", 0

String_3 db 13, 10, "A - B: ", 0

String_4 db 13, 10, "A * B: ", 0

String_5 db 13, 10, "A / B: ", 0

String_6 db 13, 10, "A % B: ", 0

String_7 db 13, 10, "X = (A - B) * 10 + (A + B) / 10", 13, 10, 0

String_8 db 13, 10, "Y = X + (X % 10)", 13, 10, 0

;===Addition

Data=====

=====

hConsoleInput dd ?

hConsoleOutput dd ?

endBuff db 5 dup (?)

msg1310 db 13, 10, 0

CharsReadNum dd ?

InputBuf db 15 dup (?)

OutMessage db "%hd", 0

ResMessage db 20 dup (?)


```

.CODE
start:
invoke AllocConsole
invoke GetStdHandle, STD_INPUT_HANDLE
mov hConsoleInput, eax
invoke GetStdHandle, STD_OUTPUT_HANDLE
mov hConsoleOutput, eax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_0, SIZEOF String_0 - 1, 0, 0
    call Input_
    mov _aAAAAAA_, ax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1 - 1, 0, 0
    call Input_
    mov _bBBBBBB_, ax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    call Add_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    call Sub_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    call Mul_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    call Div_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    call Mod_
    call Output_
    push _aAAAAAA_
    push _bBBBBBB_
    call Sub_
    push word ptr 10
    call Mul_
    push _aAAAAAA_
    push _bBBBBBB_
    call Add_
    push word ptr 10
    call Div_
    call Add_
    pop _xxxxxxx_

```

```

    push _xXXXXXX_
    push _xXXXXXX_
    push word ptr 10
    call Mod_
    call Add_
    pop _yYYYYYY_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_7, SIZEOF String_7 - 1, 0, 0
    push _xXXXXXX_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_8, SIZEOF String_8 - 1, 0, 0
    push _yYYYYYY_
    call Output_
exit_label:
    invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0, 0
    invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0
    invoke ExitProcess, 0

```

```

;===Procedure
Add=====
=====
Add_ PROC
    mov ax, [esp + 6]
    add ax, [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Add_ ENDP
;=====
=====

```

```

;===Procedure
Div=====
=====
Div_ PROC
    pushf
    pop cx

    mov ax, [esp + 4]
    cmp ax, 0
    jne end_check
    invoke WriteConsoleA, hConsoleOutput, ADDR DivErrMsg, SIZEOF DivErrMsg - 1, 0, 0
    jmp exit_label
end_check:
    mov ax, [esp + 6]
    cmp ax, 0
    jge gr

```

```

lo:
    mov dx, -1
    jmp less_fin
gr:
    mov dx, 0
less_fin:
    mov ax, [esp + 6]
    idiv word ptr [esp + 4]
    push cx
    popf

    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Div_ ENDP
;=====
=====

;===Procedure
Input=====
=====
    Input_ PROC
        invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0
        invoke crt_atoi, ADDR InputBuf
        ret
    Input_ ENDP
;=====
=====

;===Procedure
Mod=====
=====
    Mod_ PROC
        pushf
        pop cx

        mov ax, [esp + 4]
        cmp ax, 0
        jne end_check
        invoke WriteConsoleA, hConsoleOutput, ADDR ModErrMsg, SIZEOF ModErrMsg - 1, 0,
0
        jmp exit_label
    end_check:
        mov ax, [esp + 6]
        cmp ax, 0
        jge gr

```

```

lo:
    mov dx, -1
    jmp less_fin
gr:
    mov dx, 0
less_fin:
    mov ax, [esp + 6]
    idiv word ptr [esp + 4]
    mov ax, dx
    push cx
    popf

    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Mod_ ENDP
;=====
=====

;===Procedure
Mul=====
=====
Mul_ PROC
    mov ax, [esp + 6]
    imul word ptr [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Mul_ ENDP
;=====
=====

;===Procedure
Output=====
=====
Output_ PROC value: word
    invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value
    invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0
    ret 2
Output_ ENDP
;=====
=====

```

```

;===Procedure
Sub=====
=====
Sub_ PROC
    mov ax, [esp + 6]
    sub ax, [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Sub_ ENDP
;=====
=====

end start
Prog2.asm
.386
.model flat, stdcall
option casemap :none

include masm32\include\windows.inc
include masm32\include\kernel32.inc
include masm32\include\masm32.inc
include masm32\include\user32.inc
include masm32\include\msvcrt.inc
includelib masm32\lib\kernel32.lib
includelib masm32\lib\masm32.lib
includelib masm32\lib\user32.lib
includelib masm32\lib\msvcrt.lib

.DATA
;===User
Data=====
=====
    _aAAAAAA_ dw    0
    _bBBBBBB_ dw    0
    _cCCCCC_  dw    0

    String_0   db    "Input A: ", 0
    String_1   db    "Input B: ", 0
    String_2   db    "Input C: ", 0
    String_3   db    13, 10, 0
    String_4   db    13, 10, 0
    String_5   db    13, 10, 0

;===Addition
Data=====
=====
    hConsoleInputdd    ?
    hConsoleOutput     dd    ?

```

endBuff	db	5 dup (?)
msg1310	db	13, 10, 0
CharsReadNum	dd	?
InputBuf	db	15 dup (?)
OutMessage	db	"%hd", 0
ResMessage	db	20 dup (?)

.CODE

start:

invoke AllocConsole

invoke GetStdHandle, STD_INPUT_HANDLE

mov hConsoleInput, eax

invoke GetStdHandle, STD_OUTPUT_HANDLE

mov hConsoleOutput, eax

invoke WriteConsoleA, hConsoleOutput, ADDR String_0, SIZEOF String_0 - 1, 0, 0

call Input_

mov _aAAAAAA_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1 - 1, 0, 0

call Input_

mov _bBBBBBB_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2 - 1, 0, 0

call Input_

mov _cCCCCC_, ax

push _aAAAAAA_

push _bBBBBBB_

call Greate_

pop ax

cmp ax, 0

je endIf2

push _aAAAAAA_

push _cCCCCC_

call Greate_

pop ax

cmp ax, 0

je elseLabel1

jmp _aBIGGER_

jmp endIf1

elseLabel1:

push _cCCCCC_

call Output_

jmp _oUTOFI_

aBIGGER:

push _aAAAAAA_

call Output_

jmp _oUTOFI_

endIf1:

endIf2:

push _bBBBBBB_

push _cCCCCC_

```

    call Less_
    pop ax
    cmp ax, 0
    je elseLabel3
    push _cCCCCC_
    call Output_
    jmp endIf3
elseLabel3:
    push _bBBBBBB_
    call Output_
endIf3:
_oUTOFI_:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    call Equal_
    push _aAAAAAA_
    push _cCCCCC_
    call Equal_
    call And_
    push _bBBBBBB_
    push _cCCCCC_
    call Equal_
    call And_
    pop ax
    cmp ax, 0
    je elseLabel4
    push word ptr 1
    call Output_
    jmp endIf4
elseLabel4:
    push word ptr 0
    call Output_
endIf4:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
    push _aAAAAAA_
    push word ptr 0
    call Less_
    push _bBBBBBB_
    push word ptr 0
    call Less_
    call Or_
    push _cCCCCC_
    push word ptr 0
    call Less_
    call Or_
    pop ax
    cmp ax, 0
    je elseLabel5
    push word ptr -1

```

```

    call Output_
    jmp endIf5
elseLabel5:
    push word ptr 0
    call Output_
endIf5:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
    push _aAAAAAA_
    push _bBBBBBB_
    push _cCCCCC_
    call Add_
    call Less_
    call Not_
    pop ax
    cmp ax, 0
    je elseLabel6
    push word ptr 10
    call Output_
    jmp endIf6
elseLabel6:
    push word ptr 0
    call Output_
endIf6:
exit_label:
    invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0, 0
    invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0
    invoke ExitProcess, 0

```

```

;===Procedure

```

```

Add=====
=====

```

```

Add_ PROC
    mov ax, [esp + 6]
    add ax, [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret

```

```

Add_ ENDP

```

```

;=====
=====

```

```

;===Procedure

```

```

And=====
=====

```

```

And_ PROC
    pushf

```



```

    pop cx

    mov ax, [esp + 6]
    cmp ax, 0
    jnz and_t1
    jz and_false
and_t1:
    mov ax, [esp + 4]
    cmp ax, 0
    jnz and_true
and_false:
    mov ax, 0
    jmp and_fin
and_true:
    mov ax, 1
and_fin:
    push cx
    popf

    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret

```

And_ ENDP

```

;=====
=====

```

;===Procedure

```

Equal=====
=====

```

Equal_ PROC

```

    pushf
    pop cx

```

```

    mov ax, [esp + 6]
    cmp ax, [esp + 4]
    jne equal_false
    mov ax, 1
    jmp equal_fin

```

equal_false:

```

    mov ax, 0

```

equal_fin:

```

    push cx
    popf

```

```

    mov [esp + 6], ax
    pop ecx
    pop ax

```

```

        push ecx
        ret
Equal_ ENDP
;=====
=====

;====Procedure
Greate=====
=====
        Greate_ PROC
            pushf
            pop cx

            mov ax, [esp + 6]
            cmp ax, [esp + 4]
            jle greate_false
            mov ax, 1
            jmp greate_fin
greate_false:
            mov ax, 0
greate_fin:
            push cx
            popf

            mov [esp + 6], ax
            pop ecx
            pop ax
            push ecx
            ret
        Greate_ ENDP
;=====
=====

;====Procedure
Input=====
=====
        Input_ PROC
            invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0
            invoke crt_atoi, ADDR InputBuf
            ret
        Input_ ENDP
;=====
=====

;====Procedure
Less=====
=====

```

Less_ PROC

pushf
pop cx

mov ax, [esp + 6]
cmp ax, [esp + 4]
jge less_false
mov ax, 1
jmp less_fin

less_false:

mov ax, 0

less_fin:

push cx
popf

mov [esp + 6], ax
pop ecx
pop ax
push ecx
ret

Less_ ENDP

;
=====

=====
;====Procedure

Not=====

Not_ PROC

pushf
pop cx

mov ax, [esp + 4]
cmp ax, 0
jnz not_false

not_t1:

mov ax, 1
jmp not_fin

not_false:

mov ax, 0

not_fin:

push cx
popf

mov [esp + 4], ax
ret

Not_ ENDP

;
=====

```

;===Procedure
Or=====
=====
Or_ PROC
    pushf
    pop cx

    mov ax, [esp + 6]
    cmp ax, 0
    jnz or_true
    jz or_t1
or_t1:
    mov ax, [esp + 4]
    cmp ax, 0
    jnz or_true
or_false:
    mov ax, 0
    jmp or_fin
or_true:
    mov ax, 1
or_fin:
    push cx
    popf

    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Or_ ENDP
;=====
=====

```

```

;===Procedure
Output=====
=====
Output_ PROC value: word
    invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value
    invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0
    ret 2
Output_ ENDP
;=====
=====
end start

```

Prog3.asm

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User

Data=====

=====

aAAAAA2 dw 0

aAAAAA dw 0

bBBBBBB dw 0

cCCCCC1 dw 0

cCCCCC2 dw 0

XXXXXXXX dw 0

String_0 db "Input A: ", 0

String_1 db "Input B: ", 0

String_2 db "FOR TO DO", 0

String_3 db 13, 10, 0

String_4 db 13, 10, "FOR DOWNT TO DO", 0

String_5 db 13, 10, 0

String_6 db 13, 10, "WHILE A * B: ", 0

String_7 db 13, 10, "REPEAT UNTIL A * B: ", 0

;===Addition

Data=====

=====

hConsoleInputdd ?

hConsoleOutput dd ?

endBuff db 5 dup (?)

msg1310 db 13, 10, 0

CharsReadNum dd ?

InputBuf db 15 dup (?)

OutMessage db "%hd", 0

ResMessage db 20 dup (?)

.CODE

```

start:
invoke AllocConsole
invoke GetStdHandle, STD_INPUT_HANDLE
mov hConsoleInput, eax
invoke GetStdHandle, STD_OUTPUT_HANDLE
mov hConsoleOutput, eax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_0, SIZEOF String_0 - 1, 0, 0
    call Input_
    mov _aAAAAA_, ax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1 - 1, 0, 0
    call Input_
    mov _bBBBBBBB_, ax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2 - 1, 0, 0
    push _aAAAAA_
    pop _aAAAAA2_
forPasStart1:
    push _bBBBBBBB_
    push _aAAAAA2_
    call Less_
    call Not_
    pop ax
    cmp ax, 0
    je forPasEnd1
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
    push _aAAAAA2_
    push _aAAAAA2_
    call Mul_
    call Output_
    push _aAAAAA2_
    push word ptr 1
    call Add_
    pop _aAAAAA2_
    jmp forPasStart1
forPasEnd1:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
    push _bBBBBBBB_
    pop _aAAAAA2_
forPasStart2:
    push _aAAAAA_
    push _aAAAAA2_
    call Greate_
    call Not_
    pop ax
    cmp ax, 0
    je forPasEnd2
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
    push _aAAAAA2_
    push _aAAAAA2_
    call Mul_
    call Output_

```

```

    push _aAAAAA2_
    push word ptr 1
    call Sub_
    pop _aAAAAA2_
    jmp forPasStart2
forPasEnd2:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6 - 1, 0, 0
    push word ptr 0
    pop _xXXXXXXX_
    push word ptr 0
    pop _cCCCCC1_
whileStart2:
    push _cCCCCC1_
    push _aAAAAA_
    call Less_
    pop ax
    cmp ax, 0
    je whileEnd2
    push word ptr 0
    pop _cCCCCC2_
whileStart1:
    push _cCCCCC2_
    push _bBBBBBBB_
    call Less_
    pop ax
    cmp ax, 0
    je whileEnd1
    push _xXXXXXXX_
    push word ptr 1
    call Add_
    pop _xXXXXXXX_
    push _cCCCCC2_
    push word ptr 1
    call Add_
    pop _cCCCCC2_
    jmp whileStart1
whileEnd1:
    push _cCCCCC1_
    push word ptr 1
    call Add_
    pop _cCCCCC1_
    jmp whileStart2
whileEnd2:
    push _xXXXXXXX_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_7, SIZEOF String_7 - 1, 0, 0
    push word ptr 0
    pop _xXXXXXXX_
    push word ptr 1
    pop _cCCCCC1_

```

```

repeatStart2:
    push word ptr 1
    pop _cCCCCC2_
repeatStart1:
    push _xXXXXXX_
    push word ptr 1
    call Add_
    pop _xXXXXXX_
    push _cCCCCC2_
    push word ptr 1
    call Add_
    pop _cCCCCC2_
    push _cCCCCC2_
    push _bBBBBBB_
    call Greate_
    call Not_
    pop ax
    cmp ax, 0
    je repeatEnd1
    jmp repeatStart1
repeatEnd1:
    push _cCCCCC1_
    push word ptr 1
    call Add_
    pop _cCCCCC1_
    push _cCCCCC1_
    push _aAAAAA_
    call Greate_
    call Not_
    pop ax
    cmp ax, 0
    je repeatEnd2
    jmp repeatStart2
repeatEnd2:
    push _xXXXXXX_
    call Output_
exit_label:
invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0, 0
invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0
invoke ExitProcess, 0

```

====Procedure

Add=====

```

Add_ PROC
    mov ax, [esp + 6]
    add ax, [esp + 4]
    mov [esp + 6], ax
    pop ecx

```



```

        pop ax
        push ecx
        ret
Add_ ENDP
;=====
=====

;===Procedure
Greate=====
=====
Greate_ PROC
    pushf
    pop cx

    mov ax, [esp + 6]
    cmp ax, [esp + 4]
    jle greate_false
    mov ax, 1
    jmp greate_fin
greate_false:
    mov ax, 0
greate_fin:
    push cx
    popf

    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Greate_ ENDP
;=====
=====

;===Procedure
Input=====
=====
Input_ PROC
    invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0
    invoke crt_atoi, ADDR InputBuf
    ret
Input_ ENDP
;=====
=====

```

```

;===Procedure
Less=====
=====
Less_PROC
    pushf
    pop cx

    mov ax, [esp + 6]
    cmp ax, [esp + 4]
    jge less_false
    mov ax, 1
    jmp less_fin
less_false:
    mov ax, 0
less_fin:
    push cx
    popf

    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Less_ENDP
;=====
=====

```

```

;===Procedure
Mul=====
=====
Mul_PROC
    mov ax, [esp + 6]
    imul word ptr [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
Mul_ENDP
;=====
=====

```

```

;===Procedure
Not=====
=====
Not_PROC
    pushf
    pop cx

```

```

        mov ax, [esp + 4]
        cmp ax, 0
        jnz not_false
not_t1:
        mov ax, 1
        jmp not_fin
not_false:
        mov ax, 0
not_fin:
        push cx
        popf

        mov [esp + 4], ax
        ret
Not_ENDP
;=====
=====

;===Procedure
Output=====
=====
Output_PROC value: word
        invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value
        invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0
        ret 2
Output_ENDP
;=====
=====

;===Procedure
Sub=====
=====
Sub_PROC
        mov ax, [esp + 6]
        sub ax, [esp + 4]
        mov [esp + 6], ax
        pop ecx
        pop ax
        push ecx
        ret
Sub_ENDP
;=====
=====

end start

```