

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 №20

Виконав:
ст. гр. КІ-307
Маринович Марко
Перевірив:
Козак Н. Б.

Львів-2024

ЗАВДАННЯ НА КУРСОВИЙ ПРОЄКТ

1. Цільова мова транслятора – мова програмування C або асемблер для 32/64 розрядного процесора.
2. Для отримання виконуваного файлу на виході розробленого транслятора скористатися середовищем Microsoft Visual Studio або будь-яким іншим.
3. Мова розробки транслятора: C/C++.
4. Реалізувати графічну 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 C або асемблера;*
 - об'єктний файл;*
 - виконуваний 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:

Опис вхідної мови програмування:

- Тип даних: INT_4
- Блок тіла програми: STARTPROGRAM VARIABLE...; STARTBLOK ENDBLOK
- Оператор вводу: READ ()
- Оператор виводу: WRITE ()

- Оператори: IF ELSE (C)

GOTO (C)

FOR-TO-DO (Паскаль)

FOR-DOWNTODO (Паскаль)

WHILE (Бейсік)

REPEAT-UNTIL (Паскаль)

- Регістр ключових слів: Up
- Регістр ідентифікаторів: Low4 перший символ _
- Операції арифметичні: ADD, SUB, MUL, DIV, MOD
- Операції порівняння: EQ, NE, GT, LT
- Операції логічні: !, &, |
- 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 ПРОЄКТ	2
АНОТАЦІЯ.....	5
ЗМІСТ	6
ВСТУП	7
1. ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЄКТУВАННЯ ТРАНСЛЯТОРІВ	8
2. ФОРМАЛЬНИЙ ОПИС ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ	12
2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура.....	12
3. РОЗРОБКА ТРАНСЛЯТОРА З ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ	15
3.1. Вибір технології програмування.....	15
3.2. Проектування таблиць транслятора та вибір структур даних.....	15
3.4. Розробка генератора коду.....	21
3.5. Розробка генератора коду.....	31
4. НАЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА	39
4.1. Опис інтерфейсу та інструкції користувачу.	44
4.2. Виявлення лексичних і синтаксичних помилок.....	44
4.3. Перевірка роботи транслятора за допомогою тестових задач.....	46
Тестова програма №1	48
Тестова програма №2	50
Тестова програма №3	52
ВИСНОВКИ	56
СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ	57
ДОДАТКИ	58

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 ПРОЄКТУВАННЯ 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 граматичній мові 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 граматик. 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 (Backus/Naur Form - BNF).

```

program = "STARTPROGRAM", "STARTBLOK", {"VARIABLE", variable_declaration, ";"},
{statement, ";"}, "ENDBLOK";
variable_declaration = "INT_4", variable_list;
variable_list = identifier, {"", identifier};
identifier = "#", up, low, low, low, low, "#";
up_low = up | low | digit;
up = "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";
low = "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";
statement = input_statement | output_statement | assign_statement |
if_else_statement | goto_statement | label_point | for_statement | while_statement |
repeat_until_statement | compound_statement;
input_statement = "<-", identifier;
output_statement = "->", arithmetic_expression;
arithmetic_expression = low_priority_expression {low_priority_operator,
low_priority_expression};
low_priority_operator = "ADD" | "SUB";
low_priority_expression = middle_priority_expression {middle_priority_operator,
middle_priority_expression};
middle_priority_operator = "MUL" | "DIV" | "MOD";
middle_priority_expression = identifier | number | "(", arithmetic_expression, ")";
number = ["-"], (nonzero_digit, {digit} | "0");
nonzero_digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
assign_statement = arithmetic_expression, "<-", identifier;
if_else_statement = "IF-ELSE", "(C)", {statement_in_while_body}, ";",
["ELSE", {statement_in_while_body}, ";"];
logical_expression = and_expression {"|", and_expression};
and_expression = comparison {"&", and_expression};
comparison = comparison_expression | [not_operator] "(", logical_expression, ")";
not_operator = "!";
comparison_expression = arithmetic_expression comparison_operator
arithmetic_expression;
comparison_operator = "EQ" | "NE" | "LT" | "GT";
goto_statement = "GOTO (C)", identifier;
label_point = identifier, ":";
for_statement = "FOR-TO" | "FOR-DOWNT", assign_statement,
arithmetic_expression, {statement}, ";";
statement_in_while_body = statement | ("CONTINUE", "WHILE") | ("EXIT", "WHILE");
while_statement = "WHILE", logical_expression, {statement_in_while}, "END WHILE";
repeat_until_statement = "REPEAT-UNTIL", {statement}, "UNTIL", logical_expression;

```

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
STARTPROGRAM	Початок програми
STARTBLOK	Початок тексту програми
VARIABLE	Початок блоку опису змінних
ENDBLOK	Кінець розділу операторів
READ	Оператор вводу змінних
WRITE	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
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

LT	Оператор перевірки чи менше
GT	Оператор перевірки чи більше
!	Оператор логічного заперечення
&	Оператор кон'юнкції
	Оператор диз'юнкції
INT_4	32-ох 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 З 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 та вибір структур даних.

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 символів та ключових слів

Токен	Значення
Program	STARTPROGRAM
Start	STARTBLOK
Vars	VARIABLE
End	ENDBLOK
VarType	INT_4
Read	READ
Write	WRITE
Assignment	<-
If	IF
Else	ELSE
Goto	GOTO
Colon	:
Label	
For	FOR
To	TO
DownTo	DOWNTTO
Do	DO
While	WHILE
Repeat	REPEAT
Until	UNTIL
Addition	ADD
Subtraction	SUB
Multiplication	MUL

Division	DIV
Mod	MOD
Equal	EQ
NotEqual	NE
Less	LT
Greate	GT
Not	!
And	&
Or	
Plus	+
Minus	-
Identifier	
Number	
String	
Undefined	
Unknown	
Comma	,
Quotes	“
Semicolon	;
LBraket	(
RBraket)
LComment	#*
RComment	*#
Comment	

3.3. Розробка лексичного аналізатора

Основна задача лексичного аналізу — розбити вхідний текст, що складається з послідовності символів, на послідовність лексем (слів), тобто виділити ці слова з безперервної послідовності символів. Всі символи вхідної послідовності можна поділити на дві категорії:

1. Символи, що належать яким-небудь лексемам (ключові слова, ідентифікатори, числові константи, оператори та інші).
2. Символи, що розділяють лексеми (пробіли, знаки операцій, нові рядки тощо).

Лексичний аналізатор працює в два основних режими:

- Як підпрограма, що викликається синтаксичним аналізатором для отримання чергової лексеми.
- Як повний прохід, результатом якого є файл лексем, що містить усі лексеми програми.

Ми обрали другий варіант, де спочатку виконується фаза лексичного аналізу, а результат цієї фази передається для подальшої обробки на етап синтаксичного аналізу.

3.3.1. Розробка алгоритму роботи лексичного аналізатора

Лексичний аналізатор працює за принципом скінченного автомату, що містить такі стани:

- Start — початок виділення чергової лексеми.
- Finish — кінець виділення чергової лексеми.
- EndOfFile — кінець файлу, завершення розпізнавання лексем.
- Letter — перший символ є літерою або символом `_`, розпізнаються ключові слова та ідентифікатори.
- Digit — перший символ є цифрою, розпізнаються числові константи.
- Separator — обробка роздільників (пробіли, табуляції, нові рядки).
- SComment — початок коментаря.
- Comment — ігнорування коментаря.
- Another — обробка інших символів, зокрема операторів.

Алгоритм лексичного аналізатора передбачає послідовне читання символів з вхідного файлу. Кожен символ порівнюється з набором правил для лексем. Якщо лексема відповідає певному правилу (наприклад, ключове слово або ідентифікатор), вона записується в таблицю лексем разом з її типом та додатковою інформацією.

Лексеми, які не відповідають жодному з правил, позначаються як невизначені (невідомі) лексеми.

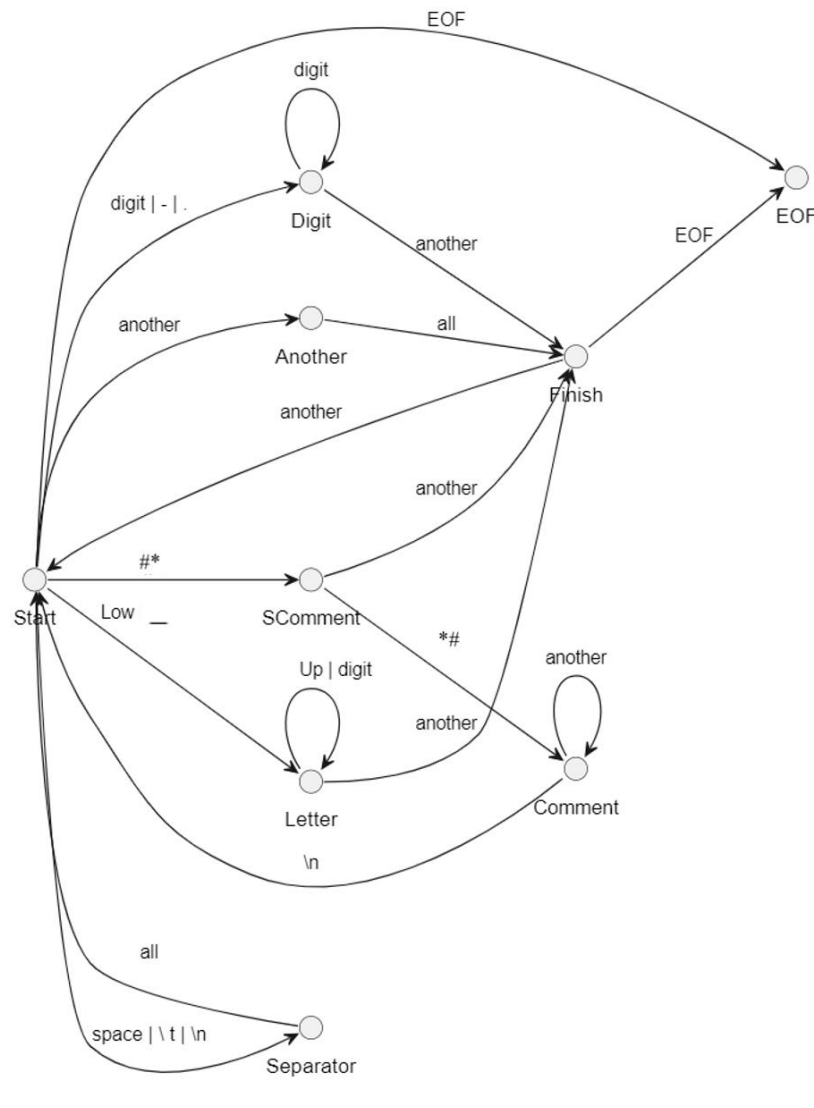


Рис. 3.1. Схема алгоритму роботи лексичного аналізатора

3.3.2. Опис програми реалізації лексичного аналізатора

У програмі для реалізації лексичного аналізу використовуються такі основні компоненти:

1. Типи лексем (Tokens):

- Ключові слова: STARTPROGRAM, STARTBLOCK, VARIABLE, ENDBLOCK, INT16, INPUT, OUTPUT, IF, ELSE, FOR, TO, DOWNT, DO, WHILE, WEND, REPEAT, UNTIL, DIV, MOD.
- Ідентифікатори: рядок, що починається з символу _, за яким йдуть літери або цифри, максимум 6 символів.
- Числові константи: ціле число.
- Оператори: ==>, +, -, *, =, !=, >>, <<, !!, &&, ||.
- Розділювачі: ,, ;.
- Дужки: (,).
- Невідома лексема: символи, що не підпадають під описані правила.

2. Алгоритм роботи лексичного аналізатора:

- Читання файлу та виділення лексем через функцію tokenize().
- Визначення типу лексеми за допомогою порівняння з ключовими словами та шаблонами.
- Формування таблиці лексем m_tokens, де для кожної лексеми вказується її тип, значення та рядок, на якому вона була знайдена.
- Виявлення лексичних помилок, таких як недопустимі символи, неправильні ідентифікатори або числові константи.

3. Структура даних для зберігання стану аналізатора:

```
enum States {
    Start,    // початковий стан
    Finish,   // кінцевий стан
    Letter,   // опрацювання слів (ключові слова та ідентифікатори)
    Digit,    // опрацювання цифр
    Separator, // опрацювання роздільників
    Another,  // опрацювання інших символів
    EndOfFile, // кінець файлу
    SComment, // початок коментаря
    Comment  // ігнорування коментаря
};
```

4. Функції для лексичного аналізатора:

- `unsigned int getTokens(FILE* F)`: основна функція для отримання лексем з файлу.
- `void printTokens(void)`: друк лексем.
- `void fprintfTokens(FILE* F)`: запис лексем у файл.

3.4. 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`.

3.4.1. Розробка дерева граматичного розбору.

Схема дерева розбору виглядає наступним чином:

Процес перевірки EBNF в проєкті реалізований через систему правил Backus та складається з наступних компонентів:

Базова структура перевірки:

- Інтерфейс IBackusRule визначає базовий контракт для всіх правил.

Політики перевірки правил:

Enum RuleCountPolicy визначає можливі варіанти входження правил:

- NoPolicy – без політики
- Optional – необов'язкове правило
- OnlyOne – тільки один раз
- Several – декілька разів
- OneOrMore – один або більше разів
- PairStart/PairEnd – парні конструкції

Реєстрація та зберігання правил:

- Клас Controller відповідає за реєстрацію правил.
- BackusRuleStorage зберігає зареєстровані правила.

Визначення правил граматики:

- Правила визначаються через BackusRuleItem з вказанням політики.
- Підтримується ієрархічна структура правил.

Процес перевірки:

- Базовий клас BackusRuleBase реалізує базову перевірку типів.
- Клас BackusRule реалізує складну перевірку правил з урахуванням політик.

Обробка помилок:

- Помилки збираються в multimap з інформацією про тип помилки та контекст.
- Кожне правило може генерувати власні помилки.

Цей механізм дозволяє:

- Перевіряти відповідність коду заданій EBNF граматиці.
- Гнучко налаштовувати правила перевірки.
- Отримувати детальну інформацію про помилки.

- Розширювати граматику новими правилами.

Визначимо назви процедур, що відповідають нетерміналам граматики таким чином:

```
void program(); // розбір програми
void programBody(); // розбір тіла програми
void variableDeclaration(); // оголошення змінних
void variableList(); // список змінних
void statement(); // оператори
void inputStatement(); // оператор вводу
void outputStatement(); // оператор виводу
void arithmeticExpression(); // арифметичні вирази
void lowPriorityExpression(); // низький пріоритет виразів
void middlePriorityExpression(); // середній пріоритет виразів
void assignStatement(); // оператор присвоєння
void ifStatement(); // оператор if
void logicalExpression(); // логічні вирази
void andExpression(); // вирази з оператором AND
void comparison(); // порівняння
void comparisonExpression(); // порівняння двох арифметичних виразів
void gotoStatement(); // оператор GOTO
void labelPoint(); // мітка
void forStatement(); // оператор FOR
void whileStatement(); // оператор WHILE
void repeatStatement(); // оператор REPEAT
void compoundStatement(); // складний оператор (STARTBLOCK / ENDBLOCK)
```

Блок-схема алгоритму роботи синтаксичного аналізатора виглядатиме наступним чином:

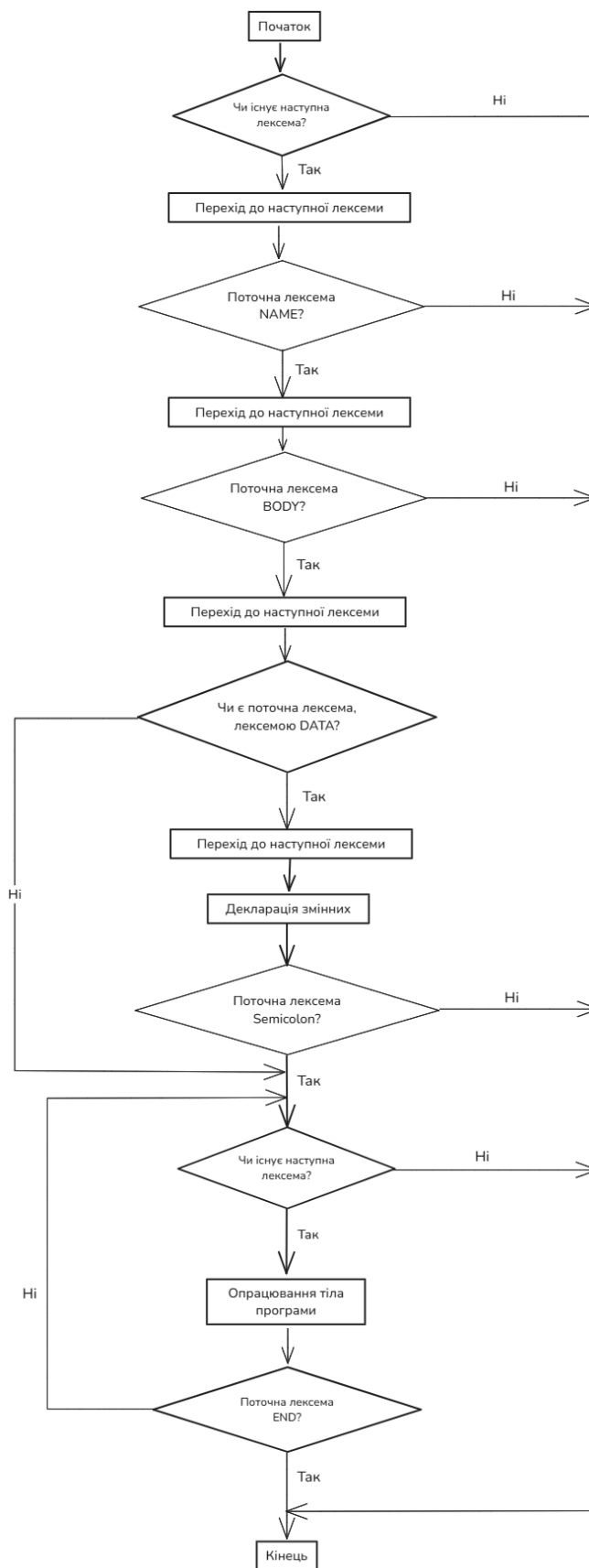


Рис. 3.3. Блок-схема алгоритму роботи синтаксичного аналізатора.

Верхньорівневий код, який описує блок схема 3.3

```

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)
});

```

Код що перевіряє валідність оголошених змінних

```

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;
};

І приватні поля що задають формат:
const std::string m_prefix = "_";
const std::string m_mask = "XXXXXXXXX";

```

3.4.2. 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.

3.4.3. Розробка алгоритму роботи семантичного аналізатора

На етапі семантичного аналізу вирішується завдання ідентифікації ідентифікаторів. Алгоритм складається з двох частин:

- Обробка оголошень ідентифікаторів.
- Обробка використання ідентифікаторів.

Коли лексичний аналізатор виявляє чергову лексему, що є ідентифікатором, він формує структуру з атрибутами, такими як ім'я, тип і лексичний клас. Ця інформація передається семантичному аналізатору. Якщо обробляється оголошення ідентифікатора, основним завданням є запис інформації до таблиці ідентифікаторів.

При обробці використання ідентифікатора семантичний аналізатор використовує раніше створену таблицю ідентифікаторів. Для отримання даних про тип ідентифікатора необхідно прочитати відповідне поле цієї таблиці.

3.4.4. Опис програмної реалізації семантичного аналізатора

Семантичний аналізатор забезпечує перевірку правильності структури та логіки програми, аналізуючи лексеми та граматику. Реалізація включає кілька ключових функцій.

Основні аспекти реалізації:

1. Лексеми та граматика

Семантичний аналізатор працює з таблицею лексем і граматикою, які є результатом лексичного та синтаксичного аналізу. Типи лексем визначаються полем `type`, а функція `GetType` використовується для отримання назви типу.

2. Перевірка конфліктів

Виявляються помилки в ідентифікаторах, щоб уникнути неоднозначностей і забезпечити коректність виконання програми.

3. Обробка помилок

Усі знайдені помилки виводяться до консолі за допомогою механізму semErr.

4. Рекурсивна перевірка правил

Аналізатор підтримує рекурсивну перевірку граматичних правил, обробку необов'язкових конструкцій, парних елементів і використання політик для визначення кількості правил (RuleCountPolicy).

5. Виконання

Функція CheckSemantic відповідає за запуск семантичного аналізу, використовуючи об'єкт Context для зберігання поточного стану.

```
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> errorsMsg;

    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());
            }
            errorsMsg.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()));
    }
}

```

Код який опрацьовує оголошення та використання ідентифікаторів, додає інформацію про ідентифікатор у таблицю ідентифікаторів

```

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;
}

```

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 проєкті 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.

Розробка алгоритму роботи генератора коду

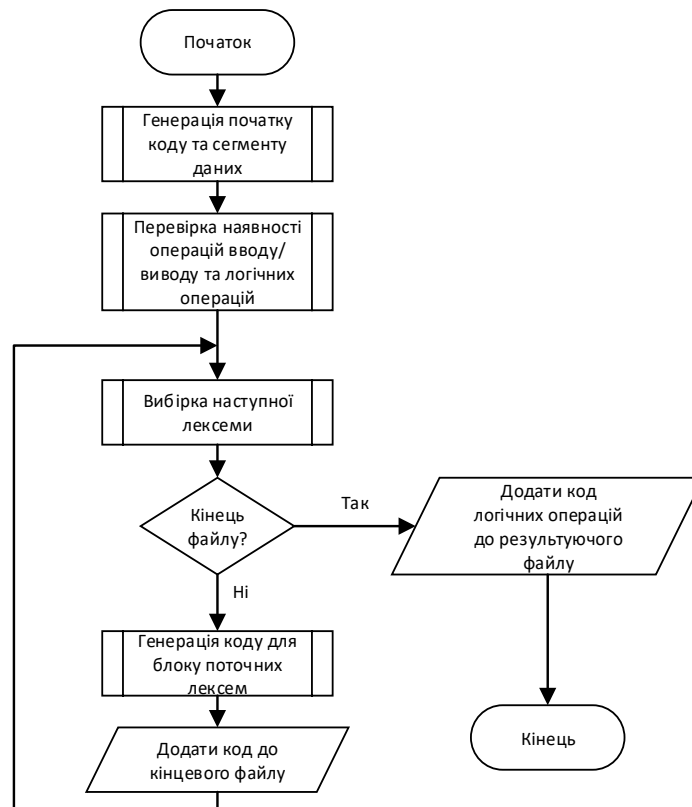


Рис. 3.5 Блок схема генератора коду

3.5.2. Опис програми реалізації генератора коду

Основні особливості реалізації:

1. Архітектура:

- Використовується патерн Singleton через клас `Generator`.
- Базується на шаблоні Visitor: кожен токен або правило має метод `genCode()`.
- Використовує `GeneratorDetails` для зберігання налаштувань і допоміжних даних.

2. Етапи генерації:

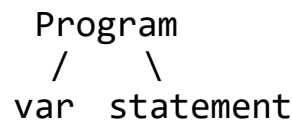
- Генерація сегмента даних.
- Генерація сегмента коду.
- Створення процедур.
- Завершення програми.

3. Технічні особливості:

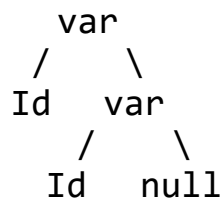
- Обчислення виконуються за стековою архітектурою.
- Підтримується постфіксна форма виразів.
- Для управління потоком виконання використовується система міток:
 - Унікальні мітки для циклів та умов.

- Іменовані мітки для операторів `GOTO`.
4. **Оптимізації:**
 - Мінімізується використання регістрів через стекову модель.
 - Процедури перевикористовуються завдяки механізму реєстрації.
 - Генерація коду оптимізується для простих конструкцій.
 5. **Обробка даних:**
 - Підтримуються числові й рядкові типи.
 - Для введення/виведення використовується Windows API.
 - Передбачена система форматування для різних типів даних.
 6. **Розширюваність:**
 - Легке додавання нових операторів через систему токенів.
 - Реєстрація користувацьких процедур.
 - Гнучкі налаштування через `GeneratorDetails`.

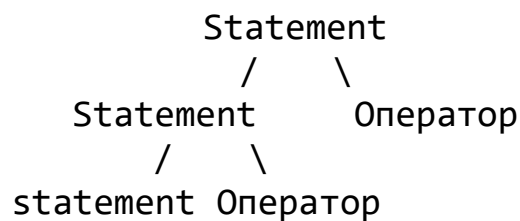
Програма має вигляд:



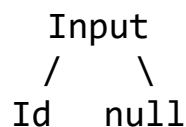
Оголошення змінних:



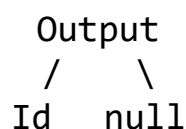
Тіло програми:



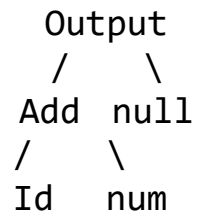
Оператор вводу:



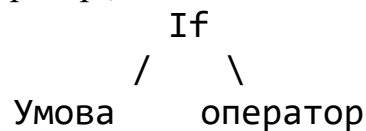
Оператор виводу:



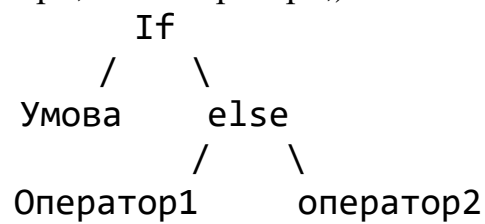
Також оператор виводу може мати за лівого нащадка різні арифметичні вирази, наприклад:



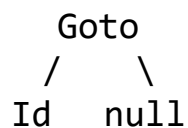
Умовний оператор (If() оператор;):



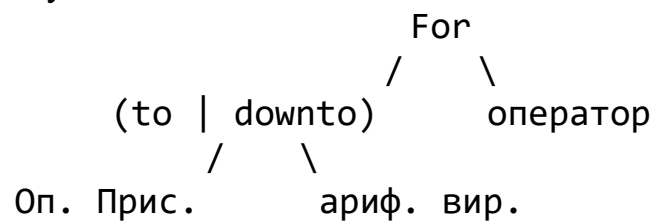
Умовний оператор (If() оператор1; else оператор2;):



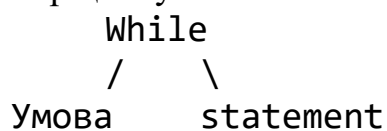
Оператор безумовного переходу:

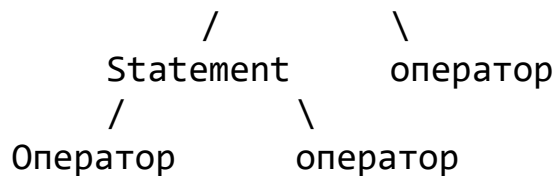


Оператор циклу for:

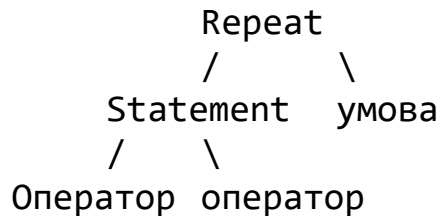


Оператор циклу while:





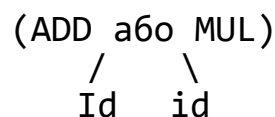
Оператор циклу repeat:



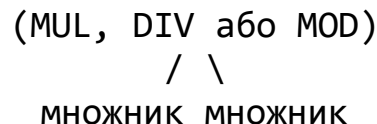
Оператор присвоєння:



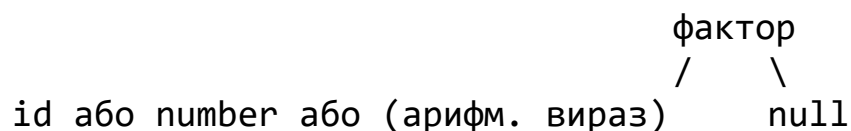
Арифметичний вираз:



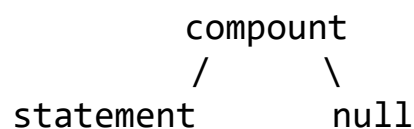
Доданок:



Множник:



Складений оператор:



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({ LBraket::Type()}, OnlyOne),
    BackusRuleItem({ identRule->type()}, OnlyOne),
    BackusRuleItem({ RBraket::Type()}, OnlyOne)
});

auto write = BackusRule::MakeRule("WriteRule", {
    BackusRuleItem({ Write::Type()}, OnlyOne),
    BackusRuleItem({ LBraket::Type()}, OnlyOne | PairStart),
    BackusRuleItem({ stringRule->type(), equation->type() }, OnlyOne),
    BackusRuleItem({ RBraket::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 токенів для певного правила. А в структурі `BackusRuleItem` описані токени, які при перевірці трактуються програмою як «АБО», тобто повинен бути лише один з описаних токенів. Наприклад для `write` послідовно необхідний токен `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 транлюватись 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 токена `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. НАЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА

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 assingmentRule = 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({ LBraket::Type()}, OnlyOne),

    BackusRuleItem({ identRule->type()}, OnlyOne),

    BackusRuleItem({ RBraket::Type()}, OnlyOne)

```

```
});
```

```
auto write = BackusRule::MakeRule("WriteRule", {
    BackusRuleItem({    Write::Type()}, OnlyOne),
    BackusRuleItem({    LBraket::Type()}, OnlyOne | PairStart),
    BackusRuleItem({ stringRule->type(), equation->type() }, OnlyOne),
    BackusRuleItem({    RBraket::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 токенів для певного правила. А в структурі `BackusRuleItem` описані токени, які при перевірці трактуються програмою як «АБО», тобто повинен бути лише один з описаних токенів. Наприклад для `write` послідовно необхідний токен `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:


```

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.

4.1. Опис інтерфейсу та інструкції користувачу.

Вхідним файлом для даної програми є звичайний текстовий файл з розширенням m20. 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_m20.exe <ім'я програми>.m20"

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

4.2. Виявлення лексичних і 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*#

STARTPROGRAM

VARI ABLE INT_4 _a aaa,_bbbb,_xxxx,_yyyy;

STARTBLOK

WRITdE("Input A: ");

READ(_aaaa);

WRITE("Input B: ");

READ(_bbbb);

WRITE("A + B: ");

WRITE(_aaaa ADD _bbbb);

WRITE("\nA - B: ");

WRITE(_aaaa SUB _bbbb);

WRITE("\nA * B: ");

WRITE(_aaaa MUL _bbbb);

WRITE("\nA / B: ");

WRITE(_aaaa DIV _bbbb);

WRITE("\nA % B: ");

WRITE(_aaaa MOD _bbbb);

_xxxx<-( _aaaa SUB _bbbb) MUL 10 ADD ( _aaaa ADD _bbbb) DIV 10;

_yyyy<-_xxxx ADD (_xxxx MOD 10);

WRITE("\nX = (A - B) * 10 + (A + B) / 10\n");

```

```
WRITE(_xxxx);

WRITE("\nY = X + (X MOD 10)\n");

WRITE(_yyyy);

ENDBLOK
```

Текст файлу з повідомленнями про помилки

List of errors

=====

There are 5 lexical errors.

There are 1 syntax errors.

There are 0 semantic errors.

Line 3: Lexical error: Unknown token: VARI

Line 3: Lexical error: Unknown token: ABLE

Line 3: Lexical error: Unknown token: _a

Line 3: Lexical error: Unknown token: aaa

Line 3: Syntax error: Expected: Vars before VARI

Line 5: Lexical error: Unknown token: WRITdE

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*#

STARTPROGRAM

VARIABLE INT_4 _aaaa,_bbbb,_xxxx,_yyyy;

STARTBLOK

WRITE("Input A: ");

READ(_aaaa);

WRITE("Input B: ");

READ(_bbbb);

WRITE("A + B: ");

WRITE(_aaaa ADD _bbbb);

WRITE("\nA - B: ");

WRITE(_aaaa SUB _bbbb);

WRITE("\nA * B: ");

WRITE(_aaaa MUL _bbbb);

WRITE("\nA / B: ");

WRITE(_aaaa DIV _bbbb);

WRITE("\nA % B: ");

WRITE(_aaaa MOD _bbbb);

_xxxx<-( _aaaa SUB _bbbb) MUL 10 ADD ( _aaaa ADD _bbbb) DIV 10;

_yyyy<-_xxxx ADD (_xxxx MOD 10);

WRITE("\nX = (A - B) * 10 + (A + B) / 10\n");

WRITE(_xxxx);

WRITE("\nY = X + (X MOD 10)\n");

WRITE(_yyyy);

ENDBLOK

```

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 отримаємо 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.

Тестова програма №1

Текст програми

```
##Prog1*#

STARTPROGRAM

VARIABLE INT_4 _aaaa,_bbbb,_xxxx,_yyyy;

STARTBLOK

WRITE("Input A: ");

READ(_aaaa);

WRITE("Input B: ");

READ(_bbbb);
```



```

WRITE("A + B: ");

WRITE(_aaaa ADD _bbbb);

WRITE("\nA - B: ");

WRITE(_aaaa SUB _bbbb);

WRITE("\nA * B: ");

WRITE(_aaaa MUL _bbbb);

WRITE("\nA / B: ");

WRITE(_aaaa DIV _bbbb);

WRITE("\nA % B: ");

WRITE(_aaaa MOD _bbbb);

_xxxx<-( _aaaa SUB _bbbb) MUL 10 ADD ( _aaaa ADD _bbbb) DIV 10;

_yyyy<-_xxxx ADD ( _xxxx MOD 10);

WRITE("\nX = (A - B) * 10 + (A + B) / 10\n");

WRITE(_xxxx);

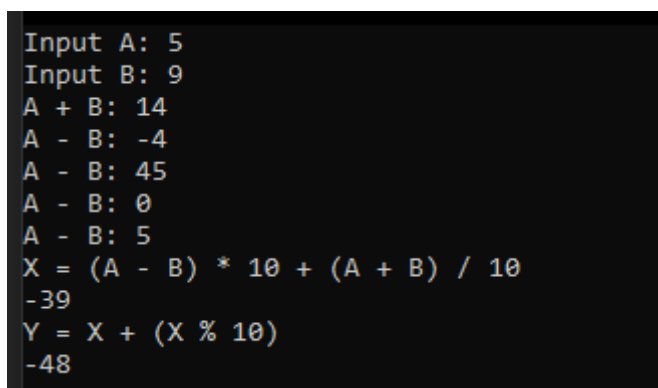
WRITE("\nY = X + (X MOD 10)\n");

WRITE(_yyyy);

ENDBLOK

```

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

Тестова програма №2*Текст програми*

```
##Prog2*#

STARTPROGRAM

VARIABLE INT_4 _aaaa,_bbbb,_cccc;

STARTBLOK

WRITE("Input A: ");

READ(_aaaa);

WRITE("Input B: ");

READ(_bbbb);

WRITE("Input C: ");

READ(_cccc);

IF(_aaaa GT _bbbb)

STARTBLOK

    IF(_aaaa GT _cccc)

STARTBLOK

    GOTO _temp;

ENDBLOK

ELSE

STARTBLOK

    WRITE(_cccc);

    GOTO _outi;

    _temp:

    WRITE(_aaaa);
```

```

        GOTO _outi;

ENDBLOK

ENDBLOK

    IF(_bbbb LT _cccc)

        STARTBLOK

            WRITE(_cccc);

        ENDBLOK

    ELSE

        STARTBLOK

            WRITE(_bbbb);

        ENDBLOK

_outi:

WRITE("\n");

IF((_aaaa EQ _bbbb) & (_aaaa EQ _cccc) & (_bbbb EQ _cccc))

    STARTBLOK

        WRITE(1);

    ENDBLOK

ELSE

    STARTBLOK

        WRITE(0);

    ENDBLOK

WRITE("\n");

IF((_aaaa LT 0) | (_bbbb LT 0) | (_cccc LT 0))

    STARTBLOK

        WRITE(- 1);

```

```

ENDBLOK

ELSE

STARTBLOK

    WRITE(0);

ENDBLOK

WRITE("\n");

IF(!_aaaa LT (_bbbb ADD _cccc))

STARTBLOK

    WRITE(10);

ENDBLOK

ELSE

STARTBLOK

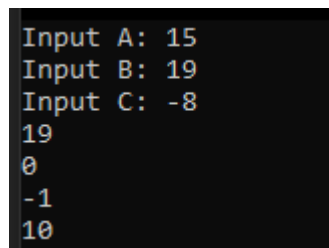
    WRITE(0);

ENDBLOK

ENDBLOK

```

Результат виконання



```

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

```

Рис. 4.3 Результат виконання тестової програми №2

Тестова програма №3

Текст програми

```

#*Prog3*#

STARTPROGRAM

VARIABLE INT_4 _aaaa,_aaa2,_bbbb,_xxxx,_ccc1,_ccc2;

STARTBLOK

WRITE("Input A: ");

READ(_aaaa);

WRITE("Input B: ");

READ(_bbbb);

WRITE("FOR TO DO");

FOR _aaa2<=_aaaa TO _bbbb DO

STARTBLOK

    WRITE("\n");

    WRITE(_aaa2 MUL _aaa2);

ENDBLOK

WRITE("\nFOR DOWNTOW DO");

FOR _aaa2<=_bbbb DOWNTOW _aaaa DO

STARTBLOK

    WRITE("\n");

    WRITE(_aaa2 MUL _aaa2);

ENDBLOK

WRITE("\nWHILE A MUL B: ");

_xxxx<-0;

_ccc1<-0;

WHILE(_ccc1 LT _aaaa)

STARTBLOK

```

```

    _ccc2<-0;

    WHILE (_ccc2 LT _bbbb)

    STARTBLOK

        _xxxx<-_xxxx ADD 1;

        _ccc2<-_ccc2 ADD 1;

    ENDBLOK

    _ccc1<-_ccc1 ADD 1;

    ENDBLOK

    WRITE(_xxxx);


    WRITE("\nREPEAT UNTIL A MUL B: ");

    _xxxx<-0;

    _ccc1<-1;

    REPEAT

        _ccc2<-1;

        REPEAT

            _xxxx<-_xxxx ADD 1;

            _ccc2<-_ccc2 ADD 1;

        UNTIL(!(_ccc2 GT _bbbb))

        _ccc1<-_ccc1 ADD 1;

    UNTIL(!(_ccc1 GT _aaaa))

    WRITE(_xxxx);


    ENDBLOK

```

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 MUL B: 45
REPEAT UNTIL A MUL 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 m20, в термінах розширеної нотації Бекуса-Наура, виділено усі термінальні символи та ключові слова.
 2. Створено компілятор мови програмування m20, 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 m20. 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 m20 були успішно виявлені і відповідно оброблені.

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

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Основи проектування трансляторів: Конспект лекцій : [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 - «Комп'ютерна інженерія» / О. І. Марченко ; КПІ ім. Ігоря Сікорського. - Київ: КПІ ім. Ігоря Сікорського, 2021. - 108 с.
2. Формальні мови, граматики та автомати: Навчальний посібник / Гавриленко С.Ю. - Харків: НТУ «ХПІ», 2021. - 133 с.
3. Сопронюк Т.М. Системне програмування. Частина I. Елементи теорії формальних мов: Навчальний посібник у двох частинах. - Чернівці: ЧНУ, 2008. - 84 с.
4. Сопронюк Т.М. Системне програмування. Частина II. Елементи теорії компіляції: Навчальний посібник у двох частинах. - Чернівці: ЧНУ, 2008. - 84 с.
5. Alfred V. Aho, Monica S. Lam, Ravi Seth, Jeffrey D. Ullma. Compilers, principles, techniques, and tools, Second Edition, New York, 2007. - 1038 с.
6. Системне програмування (курсний проект) [Електронний ресурс] - Режим доступу до ресурсу: <https://vns.lpnu.ua/course/view.php?id=11685>.

MIT OpenCourseWare. Computer Language Engineering [Електронний ресурс] - Режим доступу до ресурсу: <https://ocw.mit.edu/courses/6-035-computer-language-engineering-spring-2010>.

ДОДАТКИ

Додаток А (Таблиці лексем)

Програма 1

=====				
=====				
#	SYMBOL	TYPE	VALUE	LINE
=====				
=====				
1	#*	LComment	#*	1
2		Comment	Prog1	1
3	*#	RComment	*#	1
4	STARTPROGRAM	Program	STARTPROGRAM	2
5	VARIABLE	Vars	VARIABLE	3
6	INT_4	VarType	INT_4	3
7		Identifier	_aaaa	3
8	,	Comma	,	3
9		Identifier	_bbbb	3
10	,	Comma	,	3
11		Identifier	_xxxx	3
12	,	Comma	,	3
13		Identifier	_yyyy	3
14	;	Semicolon	;	3
15	STARTBLOK	Start	STARTBLOK	4
16	WRITE	Write	WRITE	5
17	(LBraket	(5
18	"	Quotes	"	5
19		String	Input A:	5
20	"	Quotes	"	5

21)	RBracket)	5
22	;	Semicolon	;	5
23	READ	Read	READ	6
24	(LBracket	(6
25		Identifier	_aaaa	6
26)	RBracket)	6
27	;	Semicolon	;	6
28	WRITE	Write	WRITE	7
29	(LBracket	(7
30	"	Quotes	"	7
31		String	Input B:	7
32	"	Quotes	"	7
33)	RBracket)	7
34	;	Semicolon	;	7
35	READ	Read	READ	8
36	(LBracket	(8
37		Identifier	_bbbb	8
38)	RBracket)	8
39	;	Semicolon	;	8
40	WRITE	Write	WRITE	9
41	(LBracket	(9
42	"	Quotes	"	9
43		String	A + B:	9
44	"	Quotes	"	9
45)	RBracket)	9
46	;	Semicolon	;	9
47	WRITE	Write	WRITE	10
48	(LBracket	(10

49	Identifier	_aaaa 10
50	ADD Addition	ADD 10
51	Identifier	_bbbb 10
52) RBracket) 10
53	; Semicolon	; 10
54	WRITE Write	WRITE 11
55	(LBracket	(11
56	" Quotes	" 11
57	String	\nA - B: 11
58	" Quotes	" 11
59) RBracket) 11
60	; Semicolon	; 11
61	WRITE Write	WRITE 12
62	(LBracket	(12
63	Identifier	_aaaa 12
64	SUB Subtraction	SUB 12
65	Identifier	_bbbb 12
66) RBracket) 12
67	; Semicolon	; 12
68	WRITE Write	WRITE 13
69	(LBracket	(13
70	" Quotes	" 13
71	String	\nA * B: 13
72	" Quotes	" 13
73) RBracket) 13
74	; Semicolon	; 13
75	WRITE Write	WRITE 14
76	(LBracket	(14

77	Identifier	_aaaa 14
78	MUL Multiplication	MUL 14
79	Identifier	_bbbb 14
80) RBracket) 14
81	; Semicolon	; 14
82	WRITE Write	WRITE 15
83	(LBracket	(15
84	" Quotes	" 15
85	String	\nA / B: 15
86	" Quotes	" 15
87) RBracket) 15
88	; Semicolon	; 15
89	WRITE Write	WRITE 16
90	(LBracket	(16
91	Identifier	_aaaa 16
92	DIV Division	DIV 16
93	Identifier	_bbbb 16
94) RBracket) 16
95	; Semicolon	; 16
96	WRITE Write	WRITE 17
97	(LBracket	(17
98	" Quotes	" 17
99	String	\nA % B: 17
100	" Quotes	" 17
101) RBracket) 17
102	; Semicolon	; 17
103	WRITE Write	WRITE 18
104	(LBracket	(18

105	Identifier	_aaaa 18
106	MOD Mod	MOD 18
107	Identifier	_bbbb 18
108) RBracket) 18
109	;; Semicolon	;; 18
110	Identifier	_xxxx 19
111	<- Assignment	<- 19
112	(LBracket	(19
113	Identifier	_aaaa 19
114	SUB Subtraction	SUB 19
115	Identifier	_bbbb 19
116) RBracket) 19
117	MUL Multiplication	MUL 19
118	Number	10 19
119	ADD Addition	ADD 19
120	(LBracket	(19
121	Identifier	_aaaa 19
122	ADD Addition	ADD 19
123	Identifier	_bbbb 19
124) RBracket) 19
125	DIV Division	DIV 19
126	Number	10 19
127	;; Semicolon	;; 19
128	Identifier	_yyyy 20
129	<- Assignment	<- 20
130	Identifier	_xxxx 20
131	ADD Addition	ADD 20
132	(LBracket	(20

133	Identifier	_xxxx 20
134	MOD Mod	MOD 20
135	Number	10 20
136) RBracket) 20
137	; Semicolon	; 20
138	WRITE Write	WRITE 21
139	(LBracket	(21
140	" Quotes	" 21
141	String	\nX = (A - B) * 10 + (A + B) / 10\n 21
142	" Quotes	" 21
143) RBracket) 21
144	; Semicolon	; 21
145	WRITE Write	WRITE 22
146	(LBracket	(22
147	Identifier	_xxxx 22
148) RBracket) 22
149	; Semicolon	; 22
150	WRITE Write	WRITE 23
151	(LBracket	(23
152	" Quotes	" 23
153	String	\nY = X + (X MOD 10)\n 23
154	" Quotes	" 23
155) RBracket) 23
156	; Semicolon	; 23
157	WRITE Write	WRITE 24
158	(LBracket	(24
159	Identifier	_yyyy 24
160) RBracket) 24

161	;	Semicolon	;	24
162	ENDBLOK	End	ENDBLOK	25
163		EndOfFile		-1

Програма 2

```
=====
| # | SYMBOL | TYPE | VALUE | LINE |
|====|=====|=====|=====|=====|
| 1 |      #* | LComment |      #* | 1 |
| 2 |      | Comment |      Prog2 | 1 |
| 3 |      *# | RComment |      *# | 1 |
| 4 | STARTPROGRAM | Program | STARTPROGRAM | 2 |
| 5 | VARIABLE | Vars | VARIABLE | 3 |
| 6 | INT_4 | VarType | INT_4 | 3 |
| 7 |      | Identifier | _aaaa | 3 |
| 8 |      , | Comma |      , | 3 |
| 9 |      | Identifier | _bbbb | 3 |
|10 |      , | Comma |      , | 3 |
|11 |      | Identifier | _cccc | 3 |
|12 |      ; | Semicolon |      ; | 3 |
|13 | STARTBLOK | Start | STARTBLOK | 4 |
|14 | WRITE | Write | WRITE | 5 |
|15 |      ( | LBraket |      ( | 5 |
|16 |      " | Quotes |      " | 5 |
|17 |      | String | Input A: | 5 |
|18 |      " | Quotes |      " | 5 |
|19 |      ) | RBraket |      ) | 5 |
|20 |      ; | Semicolon |      ; | 5 |
```


21	READ	Read	READ	6
22	(LBracket	(6
23		Identifier	_aaaa	6
24)	RBracket)	6
25	;	Semicolon	;	6
26	WRITE	Write	WRITE	7
27	(LBracket	(7
28	"	Quotes	"	7
29		String	Input B:	7
30	"	Quotes	"	7
31)	RBracket)	7
32	;	Semicolon	;	7
33	READ	Read	READ	8
34	(LBracket	(8
35		Identifier	_bbbb	8
36)	RBracket)	8
37	;	Semicolon	;	8
38	WRITE	Write	WRITE	9
39	(LBracket	(9
40	"	Quotes	"	9
41		String	Input C:	9
42	"	Quotes	"	9
43)	RBracket)	9
44	;	Semicolon	;	9
45	READ	Read	READ	10
46	(LBracket	(10
47		Identifier	_cccc	10
48)	RBracket)	10

```

| 49 |      ;| Semicolon |      ;| 10 |
| 50 |    IF |    If |    IF | 11 |
| 51 |    (| LBracket |    (| 11 |
| 52 |      | Identifier |    _aaaa | 11 |
| 53 |    GT |  Greate |    GT | 11 |
| 54 |      | Identifier |    _bbbb | 11 |
| 55 |    )| RBracket |    )| 11 |
| 56 | STARTBLOK |  Start |  STARTBLOK | 12 |
| 57 |    IF |    If |    IF | 13 |
| 58 |    (| LBracket |    (| 13 |
| 59 |      | Identifier |    _aaaa | 13 |
| 60 |    GT |  Greate |    GT | 13 |
| 61 |      | Identifier |    _cccc | 13 |
| 62 |    )| RBracket |    )| 13 |
| 63 | STARTBLOK |  Start |  STARTBLOK | 14 |
| 64 |    GOTO |  Goto |    GOTO | 15 |
| 65 |      | Identifier |    _temp | 15 |
| 66 |      ;| Semicolon |      ;| 15 |
| 67 | ENDBLOK |  End |  ENDBLOK | 16 |
| 68 |    ELSE |  Else |    ELSE | 17 |
| 69 | STARTBLOK |  Start |  STARTBLOK | 18 |
| 70 |    WRITE |  Write |    WRITE | 19 |
| 71 |    (| LBracket |    (| 19 |
| 72 |      | Identifier |    _cccc | 19 |
| 73 |    )| RBracket |    )| 19 |
| 74 |      ;| Semicolon |      ;| 19 |
| 75 |    GOTO |  Goto |    GOTO | 20 |
| 76 |      | Identifier |    _outi | 20 |

```

77	;	Semicolon	;	20
78	Identifier	_temp	21	
79	:	Colon	:	21
80	WRITE	Write	WRITE	22
81	(LBracket	(22
82	Identifier	_aaaa	22	
83)	RBracket)	22
84	;	Semicolon	;	22
85	GOTO	Goto	GOTO	23
86	Identifier	_outi	23	
87	;	Semicolon	;	23
88	ENDBLOK	End	ENDBLOK	24
89	ENDBLOK	End	ENDBLOK	25
90	IF	If	IF	26
91	(LBracket	(26
92	Identifier	_bbbb	26	
93	LT	Less	LT	26
94	Identifier	_cccc	26	
95)	RBracket)	26
96	STARTBLOK	Start	STARTBLOK	27
97	WRITE	Write	WRITE	28
98	(LBracket	(28
99	Identifier	_cccc	28	
100)	RBracket)	28
101	;	Semicolon	;	28
102	ENDBLOK	End	ENDBLOK	29
103	ELSE	Else	ELSE	30
104	STARTBLOK	Start	STARTBLOK	31

105	WRITE	Write	WRITE	32
106	(LBracket	(32
107		Identifier	_bbbb	32
108)	RBracket)	32
109	;	Semicolon	;	32
110	ENDBLOK	End	ENDBLOK	33
111		Identifier	_outi	34
112	:	Colon	:	34
113	WRITE	Write	WRITE	35
114	(LBracket	(35
115	"	Quotes	"	35
116		String	\n	35
117	"	Quotes	"	35
118)	RBracket)	35
119	;	Semicolon	;	35
120	IF	If	IF	36
121	(LBracket	(36
122	(LBracket	(36
123		Identifier	_aaaa	36
124	EQ	Equal	EQ	36
125		Identifier	_bbbb	36
126)	RBracket)	36
127	&	And	&	36
128	(LBracket	(36
129		Identifier	_aaaa	36
130	EQ	Equal	EQ	36
131		Identifier	_cccc	36
132)	RBracket)	36

133	&	And	&	36
134	(LBracket	(36
135		Identifier	_bbbb	36
136	EQ	Equal	EQ	36
137		Identifier	_cccc	36
138)	RBracket)	36
139)	RBracket)	36
140	STARTBLOK	Start	STARTBLOK	37
141	WRITE	Write	WRITE	38
142	(LBracket	(38
143		Number	1	38
144)	RBracket)	38
145	;	Semicolon	;	38
146	ENDBLOK	End	ENDBLOK	39
147	ELSE	Else	ELSE	40
148	STARTBLOK	Start	STARTBLOK	41
149	WRITE	Write	WRITE	42
150	(LBracket	(42
151		Number	0	42
152)	RBracket)	42
153	;	Semicolon	;	42
154	ENDBLOK	End	ENDBLOK	43
155	WRITE	Write	WRITE	44
156	(LBracket	(44
157	"	Quotes	"	44
158		String	\n	44
159	"	Quotes	"	44
160)	RBracket)	44

161	;	Semicolon	;	44
162	IF	If	IF	45
163	(LBracket	(45
164	(LBracket	(45
165	Identifier		_aaaa	45
166	LT	Less	LT	45
167		Number	0	45
168)	RBracket)	45
169		Or		45
170	(LBracket	(45
171	Identifier		_bbbb	45
172	LT	Less	LT	45
173		Number	0	45
174)	RBracket)	45
175		Or		45
176	(LBracket	(45
177	Identifier		_cccc	45
178	LT	Less	LT	45
179		Number	0	45
180)	RBracket)	45
181)	RBracket)	45
182	STARTBLOK	Start	STARTBLOK	46
183	WRITE	Write	WRITE	47
184	(LBracket	(47
185	-	Minus	-	47
186		Number	1	47
187)	RBracket)	47
188	;	Semicolon	;	47

189	ENDBLOK	End	ENDBLOK	48
190	ELSE	Else	ELSE	49
191	STARTBLOK	Start	STARTBLOK	50
192	WRITE	Write	WRITE	51
193	(LBracket	(51
194		Number	0	51
195)	RBracket)	51
196	;	Semicolon	;	51
197	ENDBLOK	End	ENDBLOK	52
198	WRITE	Write	WRITE	53
199	(LBracket	(53
200	"	Quotes	"	53
201		String	\n	53
202	"	Quotes	"	53
203)	RBracket)	53
204	;	Semicolon	;	53
205	IF	If	IF	54
206	(LBracket	(54
207	!	Not	!	54
208	(LBracket	(54
209		Identifier	_aaaa	54
210	LT	Less	LT	54
211	(LBracket	(54
212		Identifier	_bbbb	54
213	ADD	Addition	ADD	54
214		Identifier	_cccc	54
215)	RBracket)	54
216)	RBracket)	54

217) RBraket) 54
218	STARTBLOK Start	STARTBLOK 55
219	WRITE Write	WRITE 56
220	(LBraket	(56
221	Number	10 56
222) RBraket) 56
223	; Semicolon	; 56
224	ENDBLOK End	ENDBLOK 57
225	ELSE Else	ELSE 58
226	STARTBLOK Start	STARTBLOK 59
227	WRITE Write	WRITE 60
228	(LBraket	(60
229	Number	0 60
230) RBraket) 60
231	; Semicolon	; 60
232	ENDBLOK End	ENDBLOK 61
233	ENDBLOK End	ENDBLOK 62
234	EndOfFile	-1

Додаток Б (Лістинги основного програмного коду)

```

Main.cpp
#include "std.h"
#include "Controller.h"
#include "CoreParser/TokenRegister.h"
#include "CoreParser/TokenParser.h"
#include "CoreGenerator/Generator.h"

int main(int argc, std::string* argv)
{
    try
    {
        std::filesystem::path file;
        const std::string extension = ".m20";

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

        std::cout << longLine << std::endl;
        std::cout << "TRANSLATOR" << extension << "ASSEMBLER" << std::endl;
        std::cout << longLine << std::endl;

        if (argc != 2)
        {
            printf("Input file name!\n");
            std::cin >> file;
        }
        else
        {
            file = argv->at(1);
        }

        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 + ".errm.txt";
        std::string lexemFileName = fileName + ".lexem.txt";
        std::string tokenFileName = fileName + ".token.txt";
        std::string asmFileName = fileName + ".asm";

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

```



```

std::filesystem::path fileName, std::ios::out);
TokenParser::PrintTokens(fileName, tokens);
fileName.close();
std::cout << "Report file: " << fileName << std::endl;
std::cout << longLine << std::endl;

std::cout << "Error checking are starting..." << std::endl;
std::filesystem::path errorFileName, std::ios::out;
auto semanticCheckRes = CheckSemantic(fileName, 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::filesystem::path 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 << "Code generation is starting..." << std::endl;
    std::filesystem::path asmFileName, std::ios::out;
    Generator::Instance()->generateCode(asmFile, tokens);
    asmFile.close();

    if (std::filesystem::is_directory("mam2"))
    {
        std::cout << "Code generation is completed" << std::endl;
        std::cout << longLine << std::endl;
        system(std::string("mam2/bin/jc c:\\src\\" + fileName + ".asm").c_str());
        system(std::string("mam2/bin/jc .\\src\\SYSTEM\\WINDOWS\\" + fileName + ".obj").c_str());
    }
    else
    {
        std::cout << "WARNING!" << std::endl;
        std::cout << "Can't compile asm file, because mam2 doesn't exist" << std::endl;
    }
}

catch (const std::exception& ex)
{
    std::cout << "Error: " << ex.what() << std::endl;
}
catch (...)
{
    std::cout << "Unknown internal error. Better call SaaS!" << std::endl;
}

system("pause");
return 0;
}

BackusRule.h

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

std::shared_ptr<BackusRule> 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<BackusRule>>&::iterator& it,
std::list<std::shared_ptr<BackusRule>>&::iterator& end)
{
    bool res = true;
    bool resItem = false;
    auto startIt = it;
    for (auto item = m_backusItem.begin(); item != m_backusItem.end(); ++item)
    {
        if (it == end || !pairItem && HasFlag(item->policy(), RuleCompPolicy::PairFind))
        {
            if (!HasFlag(item->policy(), RuleCompPolicy::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(), RuleCompPolicy::PairFind) || HasFlag(item->policy(), RuleCompPolicy::PairFind))
        {
            bool resItem = true;
            auto startIt = it;
            if (HasFlag(item->policy(), RuleCompPolicy::Several))
                resItem = oneOfMoreCheck(errorsInfo, it, end, "item");
            else
                resItem = checkItem(errorsInfo, it, end, "item");

            if (!resItem && (!HasFlag(item->policy(), RuleCompPolicy::Optional) || startIt != it))
            {
                res &= resItem;
                break;
            }

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

            if (!resItem && pairItem && HasFlag(item->policy(), RuleCompPolicy::PairFind))
            {
                pairItem = false;
            }
        }
    }

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

    return res;
}

bool BackusRule::oneOfMoreCheck(std::multimap<int, std::pair<std::string, std::vector<std::string>>& errorsInfo,
std::list<std::shared_ptr<BackusRule>>&::iterator& it,
std::list<std::shared_ptr<BackusRule>>&::iterator& end,
const BackusRuleItem& item) const
{
    bool res = true;
    bool resItem = true;
    while (resItem && it != end && HasFlag(item->policy(), RuleCompPolicy::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<BackusRule>>&::iterator& it,
std::list<std::shared_ptr<BackusRule>>&::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())
        {
            (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(RuleCompPolicy policy, RuleCompPolicy flag)
{
    return (policy & flag) == flag;
}

BackusRuleBase.h

#pragma once
#include "stdafx.h"

```



```

        if (iresItem && start1 != it)
            res = false;
    }
    return res;
}

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

    auto start1 = it;
    auto match = it;
    if (it != end)
    {
        std::multimap<int, std::pair<std::string, std::vector<std::string>>>& error;
        for (auto rule : item.rules())
        {
            types.push_back(rule->type());

            if (ires && start1 == it)
            {
                res = rule->check(error, it, end);
            }

            if (res)
            {
                break;
            }
            else if (ires && start1 != it)
            {
                if (std::distance(match, end) > std::distance(it, end))
                    match = it;

                it = start1;
                errorInfo.insert(error.begin(), error.end());
            }
        }
    }

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

    if (ires)
        errorInfo.emplace("start1", line(), std::make_pair("it" -> value(), types));
    else
        errorInfo.clear();

    return res;
}

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

Generator.h
#pragma once
#include "stdafx.h"
#include "Utils\Singleton.h"
#include "Core\Generator\GeneratorBase.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<GeneratorItem>> generatorItems;
        for (auto item : items)
        {
            generatorItems.push_back(std::dynamic_pointer_cast<GeneratorItem>(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");
            PrintHeaderCodeSegment(out, "m_details");
            out << code.str();
            PrintEnd(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<GeneratorItem>>&& iteratork, it,
    const std::list<std::shared_ptr<GeneratorItem>>&& iteratork, end) const;

private:
static void PrintBegin(std::ostream& out, GeneratorDetails& details);
static void PrintData(std::ostream& out, GeneratorDetails& details);
static void PrintHeaderCodeSegment(std::ostream& out, GeneratorDetails& details);
static void PrintEnd(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 regSize;
        size_t posArg;
        size_t posArg1;
        std::string numberStrType;
    };

public:
    explicit GeneratorDetails(const GeneratorArgs& args) : m_args(args)
    {
        m_args.posArg0 = m_kResAddrSize + m_args.regSize;
        m_args.posArg1 = m_kResAddrSize;

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

        void registerNumberData(const std::string& name)
        {
            throw IfDataExists(name);
            m_userNumberData[name] = 'f' + name + 'f' + m_args.numberType + 'f' + '0';
        }

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

            std::string item;
            size_t start = 0;
            size_t end;
            std::string delimiter = "ja";

            m_userStringData[name] = 'f' + name + "jdbf";

            while ((end = data.find(delimiter, start)) != std::string::npos)
            {
                item = data.substr(start, end - start);
                if (!item.empty())
                    m_userStringData[name] += "f" + item + "f, ";
                m_userStringData[name] += "f13, 10, ";
                start = end + delimiter.length();
            }

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

            m_userStringData[name] += "0";
        }

        void registerRawData(const std::string& name, const std::string& rawData)
        {
            throw IfDataExists(name);
            m_userRawData[name] = 'f' + name + 'f' + rawData;
        }

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

private:
void throw IfDataExists(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 "CoreGeneratorGeneratorUtils.h"

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

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::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 "UtilsSingletonApp"
#include "CoreGeneratorGeneratorItem.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 RegisterBraket(const std::string& type)
    {
        m_braketType = type;
    }

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

    std::list<std::shared_ptr<GeneratorItem>> ConvertToPostfixForm(
        std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const
    {
        std::list<std::shared_ptr<GeneratorItem>> postfixForm;
        std::list<std::shared_ptr<GeneratorItem>> 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() && !Priority(item, stack.back()) && stack.back()->type()) != m_braketType)
                {
                    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 << "mov [esp + " << args.posArg0 << "], " << args.regPrefix << "ax\n";
        out << "push ecx\n";
        out << "pop " << args.regPrefix << "ax\n";
        out << "push ecx\n";
    }

    static bool IsNextToken(const std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::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<GeneratorItem> & item) const
    {
        return m_operands.contains(item->type());
    }

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

    bool Priority(const std::shared_ptr<GeneratorItem> & left, const std::shared_ptr<GeneratorItem> & 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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::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_bracketType;
std::string m_bracketType;
};

KGeneratorItem.h
#pragma once
#include "stdafx.h"
#include "Utils\Singleton.hpp"
#include "Core\Generator\KGeneratorItem.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 RegisterBracket(const std::string& type)
    {
        m_bracketType = type;
    }

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

    std::list<std::shared_ptr<KGeneratorItem>> ConvertToPostfixForm(
        std::list<std::shared_ptr<KGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<KGeneratorItem>>::iterator& end) const
    {
        std::list<std::shared_ptr<KGeneratorItem>> postfixForm;
        std::list<std::shared_ptr<KGeneratorItem>> 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() && !Priority(item, stack.back()) && stack.back()->type() != m_bracketType)
                {
                    postfixForm.push_back(stack.back());
                    stack.pop_back();
                }
                stack.push_back(item);
            }
            else if (itemType == m_bracketType)
            {
                stack.push_back(item);
                postfixForm.push_back(item);
            }
            else if (itemType == m_rbracketType)
            {
                while (stack.back()->type() != m_bracketType)
                {
                    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 << "move [eq + " << args.postArg << "], " << args.regPrefix << "ax\n";
        out << "pop ecx\n";
        out << "pop " << args.regPrefix << "ax\n";
        out << "push ecx\n";
    }

    static bool IsNextTokenIs(const std::list<std::shared_ptr<KGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<KGeneratorItem>>::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<KGeneratorItem>& item) const
    {
        return m_operands.contains(item->type());
    }

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

    bool Priority(const std::shared_ptr<KGeneratorItem>& left, const std::shared_ptr<KGeneratorItem>& 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<KGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<KGeneratorItem>>::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<KGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<KGeneratorItem>>::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_bracketType;
    std::string m_rbracketType;
};

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

void Generator::PrintBegin(std::ostream& out, GeneratorDetails& details)
{
    out << ".586\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\\user32.lib\n";
    out << "#include masm32\\include\\user32.lib\n";

    out << "#include lib masm32\\lib\\kernel32.lib\n";
    out << "#include lib masm32\\lib\\masm32.lib\n";
    out << "#include lib masm32\\lib\\user32.lib\n";
    out << "#include lib masm32\\lib\\user32.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 << "a:0xConsoleInputAdd77a";
out << "b:0xConsoleOutputAdd77a";
out << "result:0x0x5454545477a";
out << "sum:0x0x1310f3db13, 10, 0a";

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

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

void Generator::PrintLogCodeSegment(std::ostream& out, GeneratorDetails& details)
{
    out << std::endl;
    out << "CODE\n";
    out << "start\n";
    out << "invoke AllocConsole";
    out << "invoke GetStdHandle, STD_INPUT_HANDLE, 0a";
    out << "move hConsoleInput, eax\n";
    out << "invoke GetStdHandle, STD_OUTPUT_HANDLE, 0a";
    out << "move hConsoleOutput, eax\n";
}

void Generator::PrintEnding(std::ostream& out, GeneratorDetails& details)
{
    out << "exit_label\n";
    out << "invoke WriteConsoleA, hConsoleOutput, ADDR, 0x1310, SIZEOF, 0x1310 - 1, 0, 0a";
    out << "invoke ReadConsoleA, hConsoleInput, ADDR, 0x0, 0a";
    out << "invoke ExitProcess, 0a";

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

    out << "end start\n";
}

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

TokenParser.h
#pragma once
#include "stdafx.h"
#include "Utils Singleton.h"
#include "Common Tokens/Tokens.h"
#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<Token>> tokenize(std::istream& input);

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

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

    size_t maxElemLen = 0;
    size_t maxTypeLen = 0;
    size_t maxValLen = 0;

    for (auto token : tokens)
    {
        maxElemLen = std::max(maxElemLen, token->elem().size());
        maxTypeLen = std::max(maxTypeLen, token->type().size());
        maxValLen = std::max(maxValLen, token->value().size());
    }

    const std::string kHeaderColumn0 = "T";
    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(), maxElemLen) + 2 * colPadding;
    auto widthColumn2 = std::max(kHeaderColumn2.size(), maxTypeLen) + 2 * colPadding;
    auto widthColumn3 = std::max(kHeaderColumn3.size(), maxValLen) + 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 { return std::to_string(index++); };
    auto getElem = [&index]() const { return token->elem(); };
    auto getType = [&index]() const { return token->type(); };
    auto getValue = [&index]() const { return token->value(); };
    auto getLine = [&index]() const { return 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, getElem, getType, getValue, getLine },
colPadding);
}

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

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

private:
    static bool kNewLine(const char& ch);
    static bool kTabulation(const char& ch);
    static bool kAllowedSymbol(const char& ch);
    static bool kAllowedSpecialSymbol(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<Token>, PriorityCompare> m_priorityTokens;
    std::map<std::string, std::tuple<std::shared_ptr<Token>, std::shared_ptr<Token>, std::shared_ptr<Token>>> m_unchangedTextTokens;

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

    std::function<std::shared_ptr<Token>(std::string)> m_getTokenByType = [&this]() const { return type; };
    auto start = m_priorityTokens.lower_bound(static_cast<int>(type.size()));
    auto mapItem = std::find_if(start, m_priorityTokens.end(), [&type]() const { return pair.second->type == type; });

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

    return mapItem->second;
};

TokenParser.cpp
#include "stdafx.h"
#include "Common Tokens/Tokens.h"
#include "Utils String/Utils.h"
#include "Tokens Common EndOfFile.h"

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

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

        if (kNewLine(ch))

```

```

    ++curLine;

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

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

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

            unchangedTextTokenValue += ch;
        } while ((StringUtil::Compare(unchangedTextTokenValue, (BorderLex, StringUtil::EndWith) && input_get(ch));
        unchangedTextTokenValue = unchangedTextTokenValue.substr(0, unchangedTextTokenValue.size() - (BorderLex.size()));
        m_tokens.push_back(target->tryCreateToken(unchangedTextTokenValue));
        m_tokens.back()->set_line(unchangedTextTokenLine);

        if (right)
        {
            m_tokens.push_back(right->tryCreateToken(BorderLex));
            m_tokens.back()->set_line(curLine);
        }

        continue;
    }

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

if (tokens.empty())
    recognizeToken(tokens, curLine);
m_tokens.push_back(std::make_shared<EndOfFile>());
return m_tokens;
}

void TokenParser::regToken(std::shared_ptr<Token> token, int priority)
{
    throw IfTokenRegistered(tokens);

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

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

void TokenParser::regUnchangedTextToken(std::shared_ptr<Token> target, std::shared_ptr<Token> lBorder, std::shared_ptr<Token> rBorder)
{
    if (lBorder)
        throw IfTokenRegistered(lBorder);

    regToken(lBorder);
    throw IfTokenRegistered(target);

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

void TokenParser::throwIfTokenRegistered(std::shared_ptr<Token> 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()->set_line(curLine);
            break;
        }
    }

    if (tokens.empty() && !isUnchangedTextTokenLast())
        recognizeToken(tokens, 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.begin()->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> allowedSymbols{ ".,;:" };
    return allowedSymbols.contains(ch);
}

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

#include "RulesIdentRuleUndefined.h"
#include "TokensCommonUnknown.h"

void Init();

template<typename T>
bool CheckSemantic(std::ostream& out, std::list<std::shared_ptr<T>>& tokens)
{
    auto endOffFileType = tokens.back()->type();
    std::list<std::shared_ptr<BackusRule>> rules;
    for (auto token : tokens)
    {
        if (auto rule = std::dynamic_pointer_cast<BackusRule>(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()->stopRule()->checkErrors(it, end);

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

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

    int lexErr = 0;
    int syntErr = 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 errMsg = rule->customData("error"); !errMsg.empty())
            {
                semErr++;
                err = "Semantic error: " + errMsg;
            }
            else
            {

```

```

        semErr++;
        err = std::format("Semantic error: Undefined token: {}, rule->value{});
    }
    errorsMsg.emplace(rule->line(), err);
}
else if (rule->type) == token::Unknown::Type()
{
    lexErr++;
    res = false;
    errorsMsg.emplace(rule->line(), std::format("Lexical error: Unknown token: {}, rule->value{}));
}
}

for (auto it = errors.begin(); it != errors.end(); ++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 << ", ";
        }
    }

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

        if (!types.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 " << syntaxErr << " 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/Comments.h"

#include "Rules/Operators/IBRule.h"
#include "Rules/Operators/GotoGenRule.h"
#include "Rules/Operators/ForForRule.h"
#include "Rules/Operators/WhileCWhileRule.h"
#include "Rules/Operators/RepeatUntilRepeatUntilRule.h"

void Init()
{
    Controller::Instance()->regOperatorRule(MakeOf);
    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->Item::Type::TokenAndRule, 2);
    Controller::Instance()->regUnchangedTextToken(std::make_shared<Comment>(), std::make_shared<LComment>(), nullptr);
    Controller::Instance()->init();
}

TokenOperator:
Loops:

Do h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        out << "loop " << details.arg0().argPrefix << "as" << std::endl;
        out << "loop " << details.arg0().argPrefix << "as, 0" << std::endl;
        out << "je" << customData("endLabel") << std::endl;
    };
};

DownTo h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.h"
#include "Core/Generator/GeneratorItemBase.h"
#include "Rules/EquationRule/Gen.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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        GenProc::RegPROC(details);
        Not::RegPROC(details);
        Subtraction::RegPROC(details);
        out << customData("endLabel") << " " << 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 << "jump " << customData("ident") << std::endl;
        out << "call GenProc" << std::endl;
        out << "call Not" << std::endl;
    };
};

For h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.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"

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

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

#include "Rules/Operators/ForFor.h"
#include "Rules/Operators/ForTo.h"
#include "Rules/Operators/ForDownTo.h"
#include "Rules/Operators/ForDo.h"

BackusRulePr MakeFor(std::shared_ptr<Controller> controller)
{
    using enum Item::Type;

    controller->regItem::For();
    controller->regItem::To::(TokenAndRule) EquationAnd;
    controller->regItem::DownTo::(TokenAndRule) EquationAnd;
    controller->regItem::Do::(TokenAndRule) EquationAnd;

    auto context = controller->context();
    static const auto [Start, iCodeBlock, End] = context->CodeBlockTypes();

```



```

    }
}

return labelRule;
}

BackusRulePr MakeGen<std::shared_ptr<Controller>, controller>
(
    controller->regItem->Gen<{}>()

    auto context = controller->context();
    static const auto [Start, CodeBlock, End] = context->CodeBlockTypes();

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

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

    return gotoStatement;
}

Label.h
#pragma once
#include "stdafx.h"
#include "Core\Tokens\TokenBase.h"
#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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        out << customData() << ";" << std::endl;
    };
};

//Else:

Else.h
#pragma once
#include "stdafx.h"
#include "Core\Tokens\TokenBase.h"
#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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        out << "jump " << customData("end_label") << std::endl;
        out << customData("else_label") << ";" ;
    };
};

//If:

If.h
#pragma once
#include "stdafx.h"
#include "Core\Tokens\TokenBase.h"
#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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        it++;
        auto posForm = GeneratorUtils::Instance()->ConvertToPosixFormat(it, end);

        auto posIt = posForm.begin();
        auto posEnd = posForm.end();
        for (const auto& item : posForm)
            item->genCode(out, details, posIt, posEnd);

        out << "jump " << details.args()>regPrefix << "ax" << std::endl;
        out << "jump " << details.args()>regPrefix << "ax, 0" << std::endl;
        out << "je" << customData("label") << std::endl;
    };
};

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

BackusRulePr MakeIf(std::shared_ptr<Controller>, controller);

//Rule.cpp
#include "stdafx.h"
#include "IRule.h"
#include "Rules\Operators\If.h"
#include "Rules\Operators\IfElse.h"

BackusRulePr MakeIf(std::shared_ptr<Controller>, controller)
{
    controller->regItem->If<{}>();
    controller->regItem->IfElse<{}>();

    auto context = controller->context();
    static const auto [Start, CodeBlock, End] = context->CodeBlockTypes();

    auto elseStatement = controller->addRule("ElseStatement", {
        BackusRuleItem() { If::Type(), OnlyOne,
        BackusRuleItem() { Symbols::LBraket, OnlyOne,
        BackusRuleItem() { context->IdentifierName(), OnlyOne,
        BackusRuleItem() { Symbols::RBraket, OnlyOne,
        BackusRuleItem() { CodeBlock, OnlyOne,
        BackusRuleItem() { elseStatement->type(), Optional
        });

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

        std::string else_label = std::format("else_label[{}]", index);
        std::string end_label = std::format("end[{}]", index);

        bool hasElse = false;
        size_t count = 0;
        for (auto it = ruleBegin; it != it; ++it)
        {
            auto type = (*it)->type();
            if (type == Start)
            {
                count++;
            }
            else if (type == Else::Type() && count == 0)
            {
                ("it">setCustomData(else_label, "else_label");
                ("it">setCustomData(end_label, "end_label");
                hasElse = true;
            }
            else if (type == End && count == 1 && (*std::next(it))->type() != Else::Type())
            {
                ("it">setCustomData(end_label + ";");
            }
        }
    }
}

```

```

        break;
    }
    else if (type == End && count > 0)
    {
        count--;
    }
}

(*ruleBegin)->setCustomData(hasRule ? elsetLabel : endlLabel, "label");
};

return ifStatement;
}

RepeatUntil:
Repeat.h
#pragma once
#include "stdafx.h"
#include "Core\Tokens\TokenBase.h"
#include "Core\BackusBackusRuleBase.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 getCodeOut::toString& out, GeneratorDetails& details,
    std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
    const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        out << customData("startLabel") << " " << std::endl;
    };

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

BackusRulePr MakeRepeatUtil(std::shared_ptr<Controller> controller);

RepeatUtilRule.cpp
#include "stdafx.h"
#include "RepeatUtilRule.h"
#include "Rules\Operators\RepeatUtil\Repeat.h"
#include "Rules\Operators\RepeatUtil\Util.h"

BackusRulePr MakeRepeatUtil(std::shared_ptr<Controller> controller)
{
    controller->registerRepeat();
    controller->registerUtil();

    auto context = controller->context();
    static const auto [Start, iCodeBlock, End] = context->CodeBlockTypes();
    auto operatorsRuleName = context->OperatorsRuleName();

    auto repeatUtilRule = controller->addRule("RepeatUtilRule", {
        BackusRuleItem() Repeat::Type(), OnlyOne,
        BackusRuleItem() operatorsRuleName, OnlyOne,
        BackusRuleItem() Util::Type(), OnlyOne,
        BackusRuleItem() Symbol::LibRule1, OnlyOne,
        BackusRuleItem() context->EquationRuleName(), OnlyOne,
        BackusRuleItem() Symbol::RRule1, OnlyOne
    });

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

        std::string startLabel = std::format("repeatStart{0}", index);
        std::string endlLabel = std::format("repeatEnd{0}", 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 == Util::Type() && count == 1)
            {
                count--;
                (*itr)->setCustomData(startLabel, "startLabel");
                (*itr)->setCustomData(endlLabel, "endlLabel");
                break;
            }
            else if (type == Util::Type() && count > 0)
            {
                count--;
            }
        }
    });

    return repeatUtilRule;
}

Util.h
#pragma once
#include "stdafx.h"
#include "Core\Tokens\TokenBase.h"
#include "Core\BackusBackusRuleBase.h"
#include "Core\Generator\GeneratorItemBase.h"

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

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

    void getCodeOut::toString& out, GeneratorDetails& details,
    std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
    const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        it++;
        auto postForm = GeneratorUtil::Instance()->ConvertToPostForm(it, end);

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

        out << "prop " << details.args().argPrefix << "ax" << std::endl;
        out << "comp " << details.args().argPrefix << "ax, 0" << std::endl;
        out << "type " << customData("endlLabel") << std::endl;
        out << "jump " << customData("startLabel") << std::endl;
        out << customData("endlLabel") << " " << std::endl;
    };
};

While:
While.h
#pragma once
#include "stdafx.h"
#include "Core\Tokens\TokenBase.h"
#include "Core\BackusBackusRuleBase.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 getCodeOut::toString& out, GeneratorDetails& details,
    std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
    const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        if (customData("noGenerateCode") == "true")
        {
            return;
        }

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

        if (customData("ExitWhile") == "true")
        {
            out << "jump " << customData("endlLabel") << std::endl;
            return;
        }

        it++;
        auto postForm = GeneratorUtil::Instance()->ConvertToPostForm(it, end);

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

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

```

```

    out << "top " << details.args().argPrefix << "ax" << std::endl;
    out << "cmp " << details.args().argPrefix << "ax, 0" << std::endl;
    out << "je " << customData("end.abel") << 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 [Start, ICodeBlock, End] = context->CodeBlockTypes();
    auto operatorRuleName = context->OperatorRuleName();
    auto operatorName = context->OperatorName();
    auto operatorWithSemicolonName = context->OperatorWithSemicolonName();

    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 whileStatement = controller->addRule<"WhileStatement", {
        BackusRuleItem{ While::Type(), OnlyOne,
        BackusRuleItem{ Symbol::LBraket, OnlyOne,
        BackusRuleItem{ context->EquationRuleName(), OnlyOne,
        BackusRuleItem{ Symbol::RBraket, OnlyOne,
        BackusRuleItem{ Start::Type(), OnlyOne,
        BackusRuleItem{ operatorName, operatorWithSemicolonName, whileContinueStatement->type(), whileExitStatement->type(), Optional (OneOrMore),
        BackusRuleItem{ End::Type(), OnlyOne,
        BackusRuleItem{ While::Type(), OnlyOne,
    }>};

    whileStatement->setPostHandler([]){BackusRuleList::iterator& ruleBegin,
        BackusRuleList::iterator& s,
        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 != itr; ++itr)
        {
            auto type = (*itr)->type();

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

            if (type == Start)
            {
                count++;
            }
            else if (type == End && count == 1)
            {
                ("itrj">-setCustomData(std::format("jump {}|{}", startLabel, endLabel);
                break;
            }
            else if (type == ExitWhile::Type) && count == 1 && itr != itr && (*itr->next(itr, 1))->type() == While::Type())
            {
                itr++;
                ("itrj">-setCustomData("true", "ExitWhile");
                ("itrj">-setCustomData(endLabel, "endLabel");
            }
            else if (type == ContinueWhile::Type) && count == 1 && itr != itr && (*itr->next(itr, 1))->type() == While::Type())
            {
                itr++;
                ("itrj">-setCustomData("true", "ContinueWhile");
                ("itrj">-setCustomData(startLabel, "startLabel");
            }
            else if (type == End && count > 0)
            {
                count--;
            }
        }
    }>};

    return whileStatement;
}

Tokens:
Comment.h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/Generator/GeneratorBase.h"

class Comment : public TokenBase<Comment>, public GeneratorBase<Comment>
{
    BASH_ITEM

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

    std::shared_ptr<Token> 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.h"
#include "Core/BackusBackusRuleBase.h"
#include "Core/Generator/GeneratorBase.h"

class Program : public TokenBase<Program>, public BackusRuleBase<Program>, public GeneratorBase<Program>
{
    BASH_ITEM

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

};

Vars.h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.h"
#include "Core/Generator/GeneratorBase.h"

class Var : public TokenBase<Var>, public BackusRuleBase<Var>, public GeneratorBase<Var>
{
    BASH_ITEM

public:
    Var() { self_exeeme("DATA"); };
    virtual ~Var() = default;

};

General:
EndOfFile.h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.h"
#include "Core/Generator/GeneratorBase.h"

class EndOfFile : public TokenBase<EndOfFile>, public BackusRuleBase<EndOfFile>, public GeneratorBase<EndOfFile>
{
    BASH_ITEM

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

};

Rules:
AssignmentRules:
Assignment.h
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.h"

```



```

Generate
#pragma once
#include "stdafx.h"
#include "Com\Tokens\TokenBase.h"
#include "Com\BackusBackusRuleBase.h"
#include "Com\GeneratorGeneratorItemBase.h"

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

public:
    Generate() { self_exeeme("G"); };
    virtual ~Generate() = default;

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

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

private:
    static void PrintGenerate(std::ostream& out, const GeneratorDetails<Generate>& args)
    {
        out << "=====Procedure Generate=====<br>";
        out << "Generate_PROCPa";
        out << "ipushfa";
        out << "ipop cx16a";
        out << "jmn" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]a";
        out << "jncp" << args.regPrefix << "ax, [esp + " << args.posArg1 << "]a";
        out << "jle grate_falsea";
        out << "jmn" << args.regPrefix << "ax, fa";
        out << "jmg grate_falsea";
        out << "grate_falsea";
        out << "jmn" << args.regPrefix << "ax, fa";
        out << "grate_faia";
        out << "ipush cx16a";
        out << "ipopfa";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "retfa";
        out << "Generate_ENDPa";
        out << "=====<br>";
    };
};

Less
#pragma once
#include "stdafx.h"
#include "Com\Tokens\TokenBase.h"
#include "Com\BackusBackusRuleBase.h"
#include "Com\GeneratorGeneratorItemBase.h"

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

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

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

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

private:
    static void PrintLess(std::ostream& out, const GeneratorDetails<Less>& args)
    {
        out << "=====Procedure Less=====<br>";
        out << "Less_PROCPa";
        out << "ipushfa";
        out << "ipop cx16a";
        out << "jmn" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]a";
        out << "jncp" << args.regPrefix << "ax, [esp + " << args.posArg1 << "]a";
        out << "jg less_falsea";
        out << "jmn" << args.regPrefix << "ax, fa";
        out << "jmg less_faia";
        out << "less_faia";
        out << "jmn" << args.regPrefix << "ax, fa";
        out << "less_faia";
        out << "ipush cx16a";
        out << "ipopfa";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "retfa";
        out << "Less_ENDPa";
        out << "=====<br>";
    };
};

NotEqual
#pragma once
#include "stdafx.h"
#include "Com\Tokens\TokenBase.h"
#include "Com\BackusBackusRuleBase.h"
#include "Com\GeneratorGeneratorItemBase.h"
#include "Rules\EquationRuleEqual.h"
#include "Rules\EquationRuleNot.h"

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

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

    void genCode(std::ostream& out, GeneratorDetails& details,
        std::list<std::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        Equal-RegPROC(details);
        Not-RegPROC(details);
        out << "call Equal_in";
        out << "call Not_in";
    };

private:
    static void PrintNotEqual(std::ostream& out, const GeneratorDetails<NotEqual>& args)
    {
        out << "=====Procedure NotEqual=====<br>";
        out << "NotEqual_PROCPa";
        out << "ipushfa";
        out << "ipop cx16a";
        out << "jmn" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]a";
        out << "jncp" << args.regPrefix << "ax, fa";
        out << "jmg and_faia";
        out << "jle and_faia";
        out << "and_faia";
        out << "jmn" << args.regPrefix << "ax, [esp + " << args.posArg1 << "]a";
        out << "jncp" << args.regPrefix << "ax, fa";
        out << "jmg and_faia";
        out << "and_faia";
        out << "jmn" << args.regPrefix << "ax, fa";
        out << "jmg and_faia";
    };
};

And
#pragma once
#include "stdafx.h"
#include "Com\Tokens\TokenBase.h"
#include "Com\BackusBackusRuleBase.h"
#include "Com\GeneratorGeneratorItemBase.h"

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

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

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

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

private:
    static void PrintAnd(std::ostream& out, const GeneratorDetails<And>& args)
    {
        out << "=====Procedure And=====<br>";
        out << "And_PROCPa";
        out << "ipushfa";
        out << "ipop cx16a";
        out << "jmn" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]a";
        out << "jncp" << args.regPrefix << "ax, fa";
        out << "jmg and_faia";
        out << "jle and_faia";
        out << "and_faia";
        out << "jmn" << args.regPrefix << "ax, fa";
        out << "jncp" << args.regPrefix << "ax, fa";
        out << "jmg and_faia";
        out << "and_faia";
    };
};

```

```

        out << "and_not ia";
        out << "jmove" << args.regPrefix << "ax, 1ja";
        out << "and_fin ia";
        out << "jpush cja";
        out << "jpop fja";
        GeneratorUtil::PrintResultToStack(out, args);
        out << "iret ia";
        out << "And_ENDPia";
        out << "=====ja";
    }
};

Not
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.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(out::ostream& out, GeneratorDetails& details,
        std::list<out::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<out::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "call Not_ia";
    };

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

private:
    static void PrintNot(out::ostream& out, const GeneratorDetails::GeneratorArgs& args)
    {
        out << "=====Procedure Not=====ja";
        out << "Not_ PROCia";
        out << "jpush ia";
        out << "jpop cja";
        out << "jmove" << args.regPrefix << "ax, [esp + " << args.posArg1 << "]ja";
        out << "jcmp" << args.regPrefix << "ax, 0ja";
        out << "jnz not_false ia";
        out << "and, if ia";
        out << "jmove" << args.regPrefix << "ax, 1ja";
        out << "jpush ax, fin ia";
        out << "not_false ia";
        out << "jmove" << args.regPrefix << "ax, 0ja";
        out << "not_fin ia";
        out << "jpush cja";
        out << "jpop fja";
        out << "jmove [esp + " << args.posArg1 << "], " << args.regPrefix << "axja";
        out << "iret ia";
        out << "Not_ENDPia";
        out << "=====ja";
    };
};

Or
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.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(out::ostream& out, GeneratorDetails& details,
        std::list<out::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<out::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "call Or_ia";
    };

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

private:
    static void PrintOr(out::ostream& out, const GeneratorDetails::GeneratorArgs& args)
    {
        out << "=====Procedure Or=====ja";
        out << "Or_ PROCia";
        out << "jpush ia";
        out << "jpop cja";
        out << "jmove" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]ja";
        out << "jcmp" << args.regPrefix << "ax, 0ja";
        out << "jnz or_true ia";
        out << "jnz or_false ia";
        out << "or, if ia";
        out << "jmove" << args.regPrefix << "ax, [esp + " << args.posArg1 << "]ja";
        out << "jcmp" << args.regPrefix << "ax, 0ja";
        out << "jnz or_true ia";
        out << "or_false ia";
        out << "jmove" << args.regPrefix << "ax, 0ja";
        out << "jpush ax, fin ia";
        out << "jpop fja";
        GeneratorUtil::PrintResultToStack(out, args);
        out << "iret ia";
        out << "Or_ENDPia";
        out << "=====ja";
    };
};

Mul
Division
#pragma once
#include "stdafx.h"
#include "Core/Tokens/TokenBase.h"
#include "Core/BackusBackusRuleBase.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(out::ostream& out, GeneratorDetails& details,
        std::list<out::shared_ptr<GeneratorItem>>::iterator& it,
        const std::list<out::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);
        out << "call Div_ia";
    };

    static void RegPROC(GeneratorDetails& details)
    {
        if (!IsRegistered())
        {
            details.registerStringData("DivErrMsg", "ja" + Type() + ": Error: division by zero");
            details.registerProc("Div_", PrintDiv);
            SetRegistered();
        }
    }

private:
    static void PrintDiv(out::ostream& out, const GeneratorDetails::GeneratorArgs& args)
    {
        out << "=====Procedure Div=====ja";
        out << "Div_ PROCia";
        out << "jpush ia";
        out << "jpop cja";
        out << "jmove" << args.regPrefix << "ax, [esp + " << args.posArg1 << "]ja";
        out << "jcmp" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]ja";
        out << "jpe end_checkia";
        out << "invoke WinConsole.h::ConsoleOutput, ADDR DivErrMsg, SIZEOF DivErrMsg - 1, 0, 0ja";
        out << "jpush exit, label ia";
        out << "end_checkia";
        out << "jmove" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]ja";
        out << "jcmp" << args.regPrefix << "ax, 0ja";
        out << "jge gra";
        out << "je pa";
        out << "jmove" << args.regPrefix << "dx, -1ja";
        out << "jpush_low, fin ia";
        out << "gra";
        out << "jmove" << args.regPrefix << "dx, 0ja";
        out << "low, fin ia";
        out << "jmove" << args.regPrefix << "ax, [esp + " << args.posArg0 << "]ja";
        out << "jdx" << "jdx" << "jdxnumberTypeExtended << " ja [esp + " << args.posArg1 << "]ja";
        out << "jpush cja";
        out << "jpop fja";
        GeneratorUtil::PrintResultToStack(out, args);
        out << "iret ia";
        out << "Div_ENDPia";
        out << "=====ja";
    };
};

```



```

auto equationWithBrackets = controller->addRule("EquationWithBrackets", {
    BackusRuleItem( Symbol::LBraket ), OnlyOne( PartStart),
    BackusRuleItem( equationRuleName ), OnlyOne(),
    BackusRuleItem( Symbol::RBraket ), OnlyOne( PartEnd)
});

auto equation = controller->addRule(equationRuleName, {
    BackusRuleItem( signedNumber->type(), context->IdentRuleName(), noRule->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/BackusBackusRuleBase.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<Token> tryCreateToken(std::string& lexeme) const override
    {
        if (lexeme.size() > (m_mask.size() + m_prefix.size()))
            return nullptr;

        bool rx = 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]))
            {
                rx = false;
                break;
            }
        }

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

        return token;
    };

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

private:
    const std::string m_prefix = "-";
    const std::string m_mask = "XXXXXXXX";
};

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

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

SimpleTokenProgramName, "");

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

    controller->registerIdentifier(Tokens::AndRule, -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("identity">value))
            {
                auto undef = std::make_shared<Undefined>();
                undef->setValue("identity">value);
                undef->setLine("identity">line);
                undef->setCustomData("identity">customData);
                *identity = undef;
            }
        }
        else
        {
            if (!isFirstIdentChecked)
            {
                identTable.insert("identity">value);
            }
            else
            {
                auto progName = std::make_shared<ProgramName>();
                progName->setLine("identity">value);
                progName->setLine("identity">line);
                progName->setCustomData("identity">customData);
                *identity = progName;
                isFirstIdentChecked = true;
            }
        }

        ("identity">setCustomData("identity">value) + "-");
    };

    return identRule;
}

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

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

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

    std::shared_ptr<Token> 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/BackusBackusRuleBase.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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::iterator& end) const final
    {
        RegPROC(details);

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

        out << "tread input_16";
    };
};

```

```

    out << "new " << "(%)->customData) << ", " << details.args().regPrefix << "axja";

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

static void RegPROC(GeneratorDetails& details)
{
    if (!IsRegistered())
    {
        details.registerRawData("InputBuf", "tdb15.dsp (?)");
        details.registerRawData("CharReadNum", "dd17");
        details.registerProc("Input_", "PrintInput");
        SetRegistered();
    }
}

private:
static void PrintInput(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)
{
    out << "-----Procedure Input-----ja";
    out << "Input_PROCja";
    out << "Invoke ReadConsoleA_hConsoleProc, ADDR InputBuf, 13, ADDR CharReadNum, 0ja";
    out << "Invoke crt_atoi, ADDR InputBufja";
    out << "Ifja";
    out << "Type_ENDPja";
    out << "-----ja";
}

};

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

#include "Rules/ReadRule/Read.h"

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

    auto context = controller->context();

    auto read = controller->addRule("ReadRule", {
        BackusRuleItem{ Read::Type(), OnlyOne},
        BackusRuleItem{ Symbol::LBraket(), OnlyOne},
        BackusRuleItem{ context->IdentRuleName(), OnlyOne},
        BackusRuleItem{ Symbol::RBraket(), OnlyOne}
    });

    return read;
}

StringRule
String.h
#pragma once
#include "stdafx.h"
#include "Core/Token/TokenBase.h"
#include "Core/BackusBackusRuleBase.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<Token> 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<GeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<GeneratorItem>>::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, "">;

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

    controller->reg[unchangedTextToken](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);
        end = std::remove(it, end, "");
        it++;
        end = std::remove(it, end, "");
    });

    return stringRule;
}

VarBlockRule
VarBlockRule.cpp
#include "stdafx.h"
#include "VarBlockRule.h"

#include "Rules/VarBlockRule/VarType.h"

BackusRulePr MakeVarBlockRule(std::shared_ptr<Controller> controller)
{
    controller->regItem->VarType-<();

    auto context = controller->context();

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

    auto varBlock = controller->addRule("VarBlock", {
        BackusRuleItem{ VarType::Type(), OnlyOne},
        BackusRuleItem{ context->IdentRuleName(), OnlyOne},
        BackusRuleItem{ commaAndIdentifier->pre(), Optional(OneOrMore)},
        BackusRuleItem{ Symbol::Semicolon(), OnlyOne}
    });

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

        context->SetVarBlockChecked();
    });

    return varBlock;
}

VarType.h
#pragma once
#include "stdafx.h"
#include "Core/Token/TokenBase.h"
#include "Core/BackusBackusRuleBase.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;

private:
    WriteRule
    Write.h
    #pragma once
    #include "stdafx.h"
    #include "Core/Token/TokenBase.h"
    #include "Core/BackusBackusRuleBase.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<GeneratorItem>>::iterator& it,

```



```

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()->RegisterLBracket(tokenType);

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

void Controller::init()
{
    m_topRule = MakeTopRule()();
    Generator::Instance()->setDetails(context)->Details();
}

void Controller::regUnchangedTextToken(std::shared_ptr<Token> target, std::shared_ptr<Token> lBorder, std::shared_ptr<Token> 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("Controller::RegOperatorRule: Rule with name {} already registered", ruleName());

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

std::shared_ptr<BackusRule> 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-Var-();
    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 identifier = MakeIdentifierRule(controller);
    auto varDecl = MakeVarDeclRule(controller);
    auto equation = MakeEquationRule(controller);
    auto read = MakeReadRule(controller);
    auto write = MakeWriteRule(controller);
    auto assignmentRule = MakeAssignmentRule(controller);

    auto operatorWithSemicolonTypes = std::vector<std::variant<std::string, Symbol>->{ read->type(), write->type(), assignmentRule->type() };
    operatorWithSemicolonTypes.insert(operatorWithSemicolonTypes.end(), m_operatorRuleWithSemicolonNames.begin(), m_operatorRuleWithSemicolonNames.end());
    auto operatorWithSemicolon = controller->addRule("Operators With Semicolon", {
        BackusRuleItem{ operatorWithSemicolonTypes | OnlyOne,
        BackusRuleItem{ Symbol::Semicolon | OnlyOne
    });

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

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

    auto codeBlock = controller->addRule("CodeBlock", {
        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{ identifier->type(), OnlyOne,
        BackusRuleItem{ Symbol::Semicolon | OnlyOne,
        BackusRuleItem{ Start->type(), OnlyOne,
        BackusRuleItem{ Var->type(), OnlyOne,
        BackusRuleItem{ varDecl->type(), OnlyOne,
        BackusRuleItem{ varBlock->type(), OnlyOne,
        BackusRuleItem{ operators->type(), operatorsWithSemicolon->type(), Optional | OneOrMore,
        BackusRuleItem{ End->type(), OnlyOne
    });

    return topRule;
}

;Journé B(K02 na woi Accelfop)
Prog1.asm
386
model flat, stdcall
option casemap none

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

.DATA
;====User Data=====
;====Addition Data=====

bConsoleInput dd ?
bConsoleOutput dd ?
exitBuf db 5 dup (?)
msg1310 db 13,10,0

CharReadNum dd ?
InputBuf db 15 dup (?)
OutMessage db "%d",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 _auxnum, eax
invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1 - 1, 0, 0
call Input,
mov _bbbbbbb, eax

```

```

invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2 - 1, 0, 0
push _aaaaaaa,
push _bbbbbbbb,
call Add_
call Output,
invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
push _aaaaaaa,
push _bbbbbbbb,
call Sub_
call Output,
invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
push _aaaaaaa,
push _bbbbbbbb,
call Mul_
call Output,
invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
push _aaaaaaa,
push _bbbbbbbb,
call Div_
call Output,
invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6 - 1, 0, 0
push _aaaaaaa,
push _bbbbbbbb,
call Mod_
call Output,
push _aaaaaaa,
push _bbbbbbbb,
call Sub_
push dword ptr 10
call Mul_
push _aaaaaaa,
push _bbbbbbbb,
call Add_
push dword ptr 10
call Div_
call Add_
pop _xxxxxxx,
push _xxxxxxx,
push _xxxxxxx,
push dword ptr 10
call Mod_
call Add_
pop _yyyyyyyy,
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 _yyyyyyyy,
call Output,

exit_label:
invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0, 0
invoke ReadConsoleA, hConsoleInput, ADDR endlBuff, 5, 0, 0
invoke ExitProcess, 0

;====Procedure Add=====
Add_PROC
    mov eax, [esp + 8]
    add eax, [esp + 4]
    mov [esp + 8], eax
    pop ecx
    pop eax
    push ecx
    ret

Add_ENDP
;=====

;====Procedure Div=====
Div_PROC
    pushf
    pop cx
    mov eax, [esp + 4]
    cmp eax, 0
    jne end_check
    invoke WriteConsoleA, hConsoleOutput, ADDR DivErrMsg, SIZEOF DivErrMsg - 1, 0, 0
    jmp exit_label

end_check:
    mov eax, [esp + 8]
    cmp eax, 0
    cmp eax, 0
    jg? gf
    lo:
        mov edx, -1
        jmp less_fin

gf:
    mov edx, 0

less_fin:
    mov eax, [esp + 8]
    idiv dword ptr [esp + 4]
    push cx
    popfd
    mov [esp + 8], eax
    pop ecx
    pop eax
    push ecx
    ret

Div_ENDP
;=====

;====Procedure Input=====
Input_PROC
    invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharReadNum, 0
    invoke crt_atoi, ADDR InputBuf
    ret

Input_ENDP
;=====

;====Procedure Mod=====
Mod_PROC
    pushf
    pop cx
    mov eax, [esp + 4]
    cmp eax, 0
    jne end_check
    invoke WriteConsoleA, hConsoleOutput, ADDR ModErrMsg, SIZEOF ModErrMsg - 1, 0, 0
    jmp exit_label

end_check:
    mov eax, [esp + 8]
    cmp eax, 0
    jg? gf
    lo:
        mov edx, -1
        jmp less_fin

gf:
    mov edx, 0

less_fin:
    mov eax, [esp + 8]
    idiv dword ptr [esp + 4]
    mov eax, edx
    push cx
    popfd
    mov [esp + 8], eax
    pop ecx
    pop eax
    push ecx
    ret

Mod_ENDP
;=====

;====Procedure Mul=====
Mul_PROC
    mov eax, [esp + 8]
    imul dword ptr [esp + 4]
    mov [esp + 8], eax
    pop ecx
    pop eax
    push ecx
    ret

Mul_ENDP
;=====

;====Procedure Output=====
Output_PROC value: dword
    invoke wputnf, ADDR ResMessage, ADDR OutMessage, value
    invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, ecx, 0, 0
    ret 4

Output_ENDP
;=====

;====Procedure Sub=====
Sub_PROC
    mov eax, [esp + 8]
    sub eax, [esp + 4]
    mov [esp + 8], eax
    pop ecx
    pop eax
    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\user32.inc
include masm32\include\user32.inc
include masm32\include\user32.inc
include lib masm32\lib\kernel32.lib
include lib masm32\lib\user32.lib
include lib masm32\lib\user32.lib
include lib masm32\lib\user32.lib

.DATA
;====User Data=====
    _aaa_          dd          0

```

```

        .bbbb,          dd          0
        .ccc,          dd          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=====
        hConsoleInput   dd          ?
        hConsoleOutput  dd          ?
        endBuff         db          5 dup (?)
        msg1310         db          13, 10, 0

        ChanReadNum     dd          ?
        hInputBuf       db          15 dup (?)
        OutMessage      db          "%d": 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, eax
        invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1 - 1, 0, 0
        call Input_
        mov .bbbbbbbb, eax
        invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2 - 1, 0, 0
        call Input_
        mov .cccccccc, eax
        push .aaaaaaa
        push .bbbbbbbb
        call Create_
        pop eax
        cmp eax, 0
        je endI2
        push .aaaaaaa
        push .ccccccc
        call Create_
        pop eax
        cmp eax, 0
        je cselLabel1
        jmp .temporal_

cselLabel1:
        push .ccccccc
        call Output_
        jmp .temporal_

.temporal_:
        push .aaaaaaa
        call Output_
        jmp .outgoto_

endI1:
endI2:
        push .bbbbbbbb
        push .ccccccc
        call Less_
        pop eax
        cmp eax, 0
        je cselLabel3
        push .ccccccc
        call Output_
        jmp endI3

cselLabel3:
        push .bbbbbbbb
        call Output_

endI3:
.outgoto_:
        invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
        push .aaaaaaa
        push .bbbbbbbb
        call Equal_
        push .aaaaaaa
        push .ccccccc
        call Equal_
        call And_
        push .bbbbbbbb
        push .ccccccc
        call Equal_
        pop eax
        cmp eax, 0
        je cselLabel4
        push dword ptr 1
        call Output_
        jmp endI4

cselLabel4:
        push dword ptr 0
        call Output_

endI4:
        invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
        push .aaaaaaa
        push dword ptr 0
        call Less_
        push .bbbbbbbb
        push dword ptr 0
        call Less_
        call Or_
        push .ccccccc
        push dword ptr 0
        call Less_
        call Or_
        pop eax
        cmp eax, 0
        je cselLabel5
        push dword ptr -1
        call Output_
        jmp endI5

cselLabel5:
        push dword ptr 0
        call Output_

endI5:
        invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
        push .aaaaaaa
        push .bbbbbbbb
        push .ccccccc
        call Add_
        call Less_
        call Not_
        pop eax
        cmp eax, 0
        je cselLabel6
        push dword ptr 10
        call Output_
        jmp endI6

cselLabel6:
        push dword ptr 0
        call Output_

endI6:
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 ecx, [esp + 8]
        add ecx, [esp + 4]
        mov [esp + 8], ecx
        pop ecx
        push ecx
        ret

Add_ENDP

;====Procedure And=====
And_PROC
        pushf
        popf

        mov ecx, [esp + 8]
        cmp ecx, 0
        jnz and_1f
        je and_false

and_1f:
        mov ecx, [esp + 4]
        cmp ecx, 0
        jnz and_true

and_false:
        mov ecx, 0
        jmp and_fin

and_true:
        mov ecx, 1

and_fin:
        push cx
        popf

        mov [esp + 8], ecx
        pop ecx
        pop ecx
        push ecx
        ret

And_ENDP

;====Procedure Equal=====
Equal_PROC
        pushf
        popf

        mov ecx, [esp + 8]
        cmp ecx, [esp + 4]
        jnz equal_false
        mov ecx, 1
        jmp equal_fin

equal_false:
        mov ecx, 0

equal_fin:
        mov ecx, 0

```

```

        push cx
        popf

        mov [esp + 8], eax
        pop ecx
        push ecx
        ret
Equal_ENDP
=====

;====Procedure Greater
Greater_PROC
        pushf
        pop cx

        mov eax, [esp + 8]
        cmp eax, [esp + 4]
        jle greater_false
        mov eax, 1
        jmp greater_true

greater_false:
        mov eax, 0

greater_true:
        push cx
        popf

        mov [esp + 8], eax
        pop ecx
        push ecx
        ret
Greater_ENDP
=====

;====Procedure Input
Input_PROC
        invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharReadNum, 0
        invoke crt_atoi, ADDR InputBuf
        ret
Input_ENDP
=====

;====Procedure Less
Less_PROC
        pushf
        pop cx

        mov eax, [esp + 8]
        cmp eax, [esp + 4]
        jge less_false
        mov eax, 1
        jmp less_true

less_false:
        mov eax, 0

less_true:
        push cx
        popf

        mov [esp + 8], eax
        pop ecx
        push ecx
        ret
Less_ENDP
=====

;====Procedure Not
Not_PROC
        pushf
        pop cx

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

not_true:
        mov eax, 1
        jmp not_true_fin

not_false:
        mov eax, 0

not_fin:
        push cx
        popf

        mov [esp + 4], eax
        ret
Not_ENDP
=====

;====Procedure Or
Or_PROC
        pushf
        pop cx

        mov eax, [esp + 8]
        cmp eax, 0
        jnz or_true

or_true:
        mov eax, 1

or_false:
        mov eax, 0
        jmp or_fin

or_fin:
        push cx
        popf

        mov [esp + 8], eax
        pop ecx
        push ecx
        ret
Or_ENDP
=====

;====Procedure Output
Output_PROC value dword
        invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value
        invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0
        ret 4
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\user32.inc
include masm32\include\user32.inc
include masm32\include\user32.inc
include lib masm32\lib\kernel32.lib
include lib masm32\lib\user32.lib
include lib masm32\lib\user32.lib

.DATA
;====User Data
mas2_          dd      0
mas_           dd      0
masb_         dd      0
ccc1_         dd      0
ccc2_         dd      0
xxxx_         dd      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 DOWNT0 DO", 0
String_5       db      13, 10, 0
String_6       db      13, 10, "WHILE A MUL B: ", 0
String_7       db      13, 10, "REPEAT UNTIL A MUL B: ", 0

;====Addition Data
hConsoleInput dd      ?
hConsoleOutput dd     ?
cuiBuf        db      5 dup (?)
mui1310       db      13, 10, 0

CharReadNum   dd      ?
InputBuf       db      15 dup (?)
OutMessage     db      "64", 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 _masmasmas_ eax
        invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1 - 1, 0, 0
        call Input_
        mov _bbbbbbbbb_ eax
        invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2 - 1, 0, 0
        push _masmasmas_
        pop _masmasmas_

forPasStart:
        push _bbbbbbbbb_
        push _masmasmas_

```

```

        call Less_
        call Nul_
        pop eax
        cmp eax, 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 dword ptr 1
        call Add_
        pop _aaaaaa2_
        jmp forPasStart1

forPasEnd1:
        invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
        push _bbbbbbbbb_
        pop _aaaaaa2_

forPasStart2:
        push _aaaaaaa_
        push _aaaaaa2_
        call Create_
        call Nul_
        pop eax
        cmp eax, 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 dword ptr 1
        call Sub_
        pop _aaaaaa2_
        jmp forPasStart2

forPasEnd2:
        invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6 - 1, 0, 0
        push dword ptr 0
        pop _xxxxxxx_
        push dword ptr 0
        pop _cccccc1_

whileStart2:
        push _cccccc1_
        push _aaaaaaa_
        call Less_
        pop eax
        cmp eax, 0
        je whileEnd2
        push dword ptr 0
        pop _cccccc2_

whileStart1:
        push _cccccc2_
        push _bbbbbbbbb_
        call Less_
        pop eax
        cmp eax, 0
        je whileEnd1
        push _xxxxxxx_
        push dword ptr 1
        call Add_
        pop _xxxxxxx_
        push _cccccc2_
        push dword ptr 1
        call Add_
        pop _cccccc2_
        jmp whileStart1

whileEnd1:
        push _cccccc1_
        push dword ptr 1
        call Add_
        pop _cccccc1_
        jmp whileStart2

whileEnd2:
        push _xxxxxxx_
        call Output_
        invoke WriteConsoleA, hConsoleOutput, ADDR String_7, SIZEOF String_7 - 1, 0, 0
        push dword ptr 0
        pop _xxxxxxx_
        push dword ptr 1
        pop _cccccc1_

repeatStart2:
        push dword ptr 1
        pop _cccccc2_

repeatStart1:
        push _xxxxxxx_
        push dword ptr 1
        call Add_
        pop _xxxxxxx_
        push _cccccc2_
        push dword ptr 1
        call Add_
        pop _cccccc2_
        push _cccccc2_
        push _bbbbbbbbb_
        call Create_
        call Nul_
        pop eax
        cmp eax, 0
        je repeatEnd1
        jmp repeatStart1

repeatEnd1:
        push _cccccc1_
        push dword ptr 1
        call Add_
        pop _cccccc1_
        push _cccccc1_
        push _aaaaaaa_
        call Create_
        call Nul_
        pop eax
        cmp eax, 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 endlbuff, 5, 0, 0
        invoke ExitProcess, 0

```

```

;====Procedure Add=====
Add_PROC
        mov eax, [esp + 8]
        add eax, [esp + 4]
        mov [esp + 8], eax
        pop ecx
        pop eax
        push ecx
        ret

Add_ENDP
;=====

```

```

;====Procedure Create=====
Create_PROC
        pushf
        pop cx

        mov eax, [esp + 8]
        cmp eax, [esp + 4]
        jle create_false
        mov eax, 1
        jmp create_fin

create_false:
        mov eax, 0

create_fin:
        push cx
        popf

        mov [esp + 8], eax
        pop ecx
        pop eax
        push ecx
        ret

Create_ENDP
;=====

```

```

;====Procedure Input=====
Input_PROC
        invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharReadNum, 0
        invoke crt_atoi, ADDR InputBuf
        ret

Input_ENDP
;=====

```

```

;====Procedure Less=====
Less_PROC
        pushf
        pop cx

        mov eax, [esp + 8]
        cmp eax, [esp + 4]
        jge less_false
        mov eax, 1
        jmp less_fin

less_false:
        mov eax, 0

less_fin:
        push cx
        popf

        mov [esp + 8], eax
        pop ecx
        pop eax
        push ecx
        ret

Less_ENDP
;=====

```

```

;====Procedure Mul=====
Mul_PROC

```



```

        mov eax, [esp + 8]
        imul dword ptr [esp + 4]
        mov [esp + 8], eax
        pop ecx
        pop ecx
        push ecx
        ret

Mul_ENDP
;-----
;---Procedure Not_
Not_PROC
        pushf
        pop ecx

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

not_1!:
        mov ecx, 1
        jmp not_fin

not_false:
        mov ecx, 0

not_fin:
        push ecx
        popf

        mov [esp + 4], ecx
        ret

Not_ENDP
;-----
;---Procedure Output_
Output_PROC value: dword
        invoke WriteMsg, ADDR ResMessage, ADDR OutMessage, value
        invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, ecx, 0, 0
        ret 4

Output_ENDP
;-----
;---Procedure Sub_
Sub_PROC
        mov eax, [esp + 8]
        sub eax, [esp + 4]
        mov [esp + 8], eax
        pop ecx
        pop ecx
        push ecx
        ret

Sub_ENDP
;-----
end start

```