

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-DOWNTO-DO (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	16
3.1. Вибір технології програмування.....	16
3.2. Проектування таблиць транслятора та вибір структур даних.....	16
3.4. Розробка генератора коду.....	22
3.5. Розробка генератора коду.....	33
4. НАЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА	40
4.1. Опис інтерфейсу та інструкції користувачу.	44
4.2. Виявлення лексичних і синтаксичних помилок.....	45
4.3. Перевірка роботи транслятора за допомогою тестових задач.....	47
Тестова програма №1	49
Тестова програма №2	51
Тестова програма №3	53
ВИСНОВКИ	57
СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ	58
ДОДАТКИ	59

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 застосовував розширену нотацію Бекуса-Наура (Backus/Naur Form - BNF).

```
program = "STARTPROGRAM", variable_block, ";", code_block;
```

```
variable_block = "INT_4", identifier, { comma_and_identifier }, ";";
```

```
identifier = "_", low_letter, { low_letter | digit }, { 3 };
```

```
comma_and_identifier = ",", identifier;
```

```
code_block = "STARTBLOK", { statement }, "ENDBLOK";
```

```
statement = write | read | assignment | if_statement | goto_statement | label_rule |  
for_to_or_downto_do_rule | while_loop | repeat_until;
```

```
write = "WRITE", "(", equation | string_rule, ")";
```

```
read = "READ", "(", identifier, ")";
```

```
assignment = identifier, "<==", equation;
```

```
if_statement = "IF", "(", equation, ")", code_block, [else_statement];
```

```
else_statement = "ELSE", code_block;
```

```
goto_statement = "GOTO", identifier;
```

```
label_rule = identifier, ":";
```

```
for_to_or_downto_do_rule = "FOR", assignment, ("TO" | "DOWNTO"), equation,  
"DO", code_block;
```

```
while_loop = "WHILE", "(", equation, ")", code_block;
```

```
repeat_until = "REPEAT", code_block, "UNTIL", "(", equation, ")";
```

```
equation = signed_number | identifier | not_rule, {operation_and_identifier_or_number |
equation};
```

```
not_rule = not_operation, (signed_number | identifier | equation);
```

```
operation_and_identifier_or_number = (arithmetic | mult | logic | compare),
(signed_number | identifier | equation);
```

```
arithmetic = "ADD" | "SUB";
```

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

```
logic = "&" | "|";
```

```
not_operation = "!";
```

```
compare = "EQ" | "NE" | "LT" | "GT";
```

```
string_rule = "\"", string, "\"";
```

```
comment = l_comment, string, r_comment;
```

```
l_comment = "#*";
```

```
r_comment = "*#";
```

```
string = {low_letter | up_letter | digit};
```

```
signed_number = [sign], digit, {digit};
```

```
sign = "+" | "-";
```

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

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

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

2.1 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
DOWNTO	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.

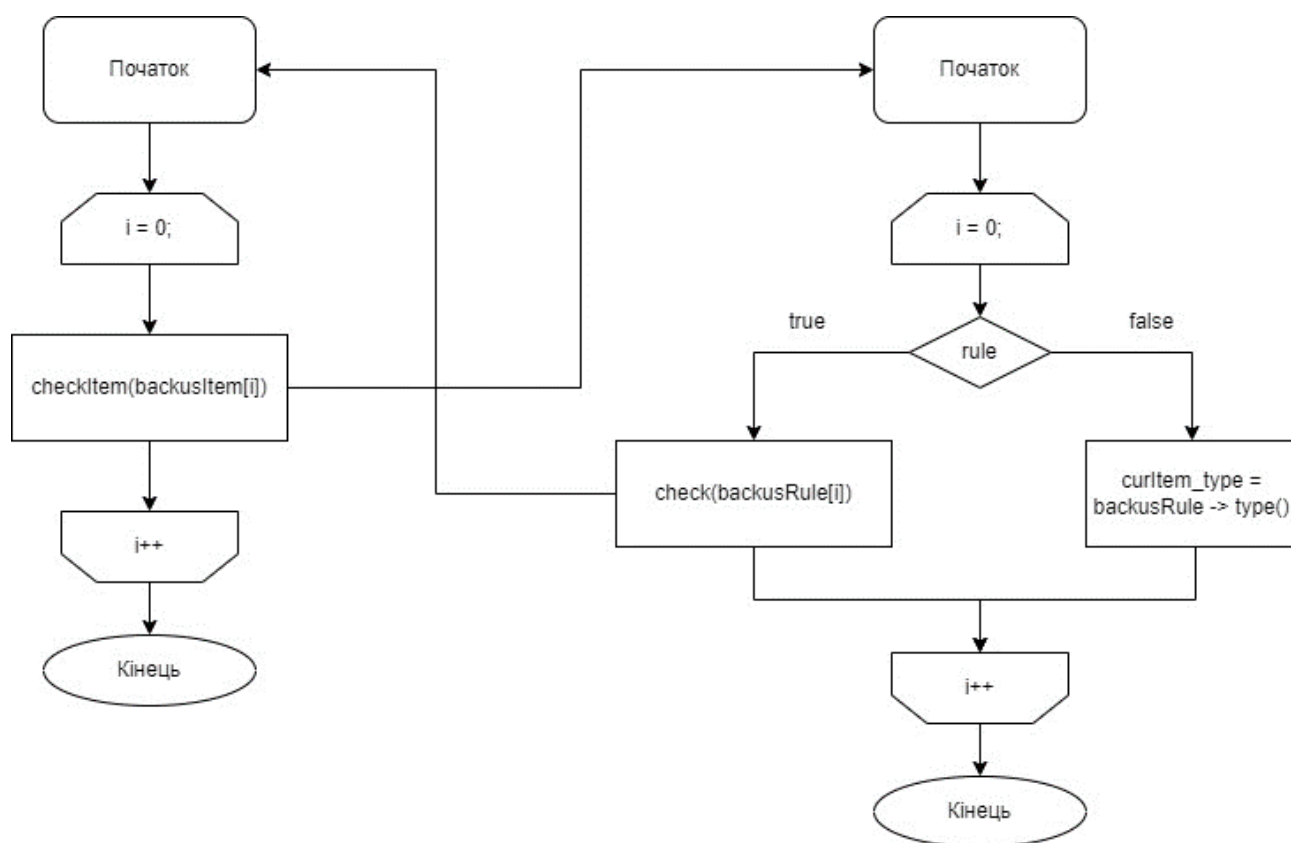


Рис. 3.1. Схема алгоритму роботи лексичного аналізатора.

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, DOWNT0, 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.

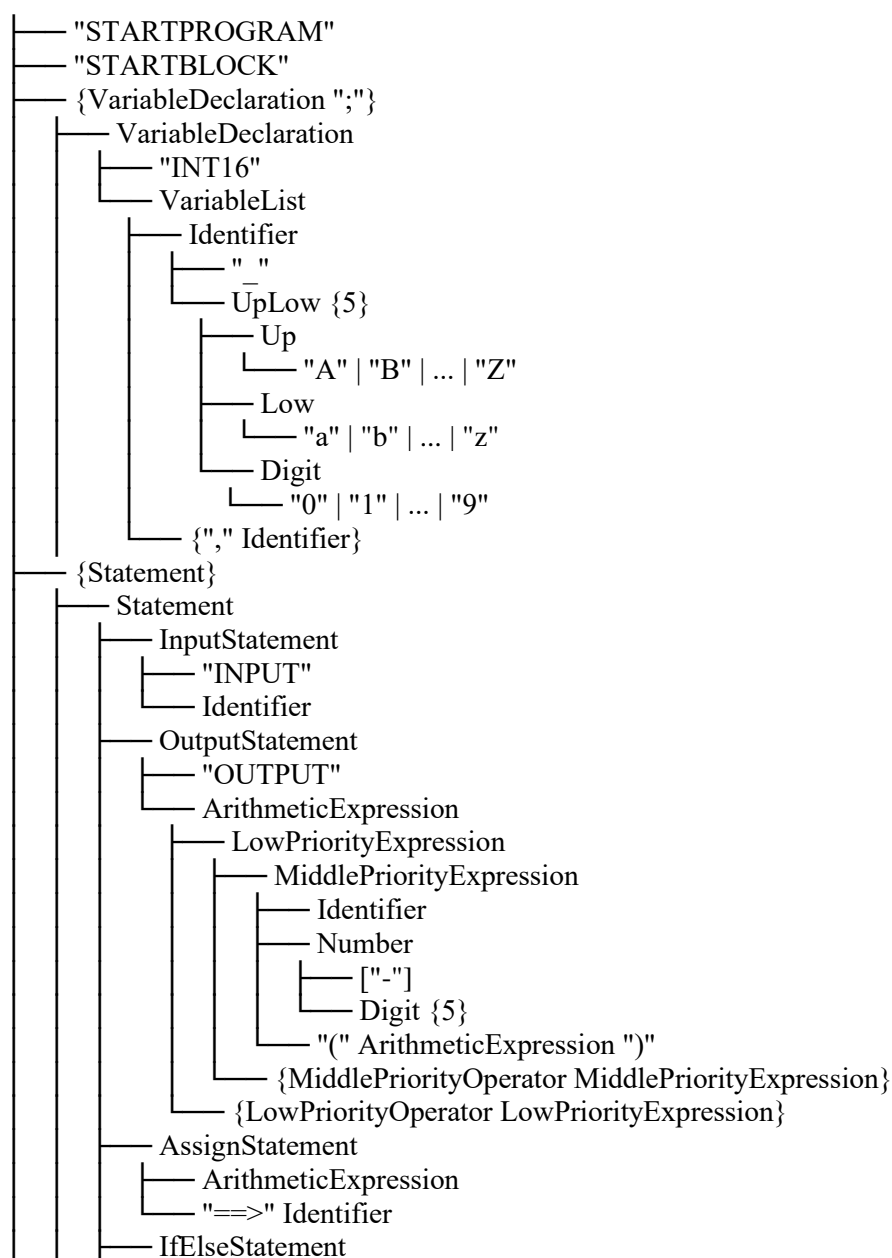
Генератор асемблерного коду приймає масив лексем без помилок. Якщо на двох попередніх етапах виявлено помилки, то ця фаза не виконується.

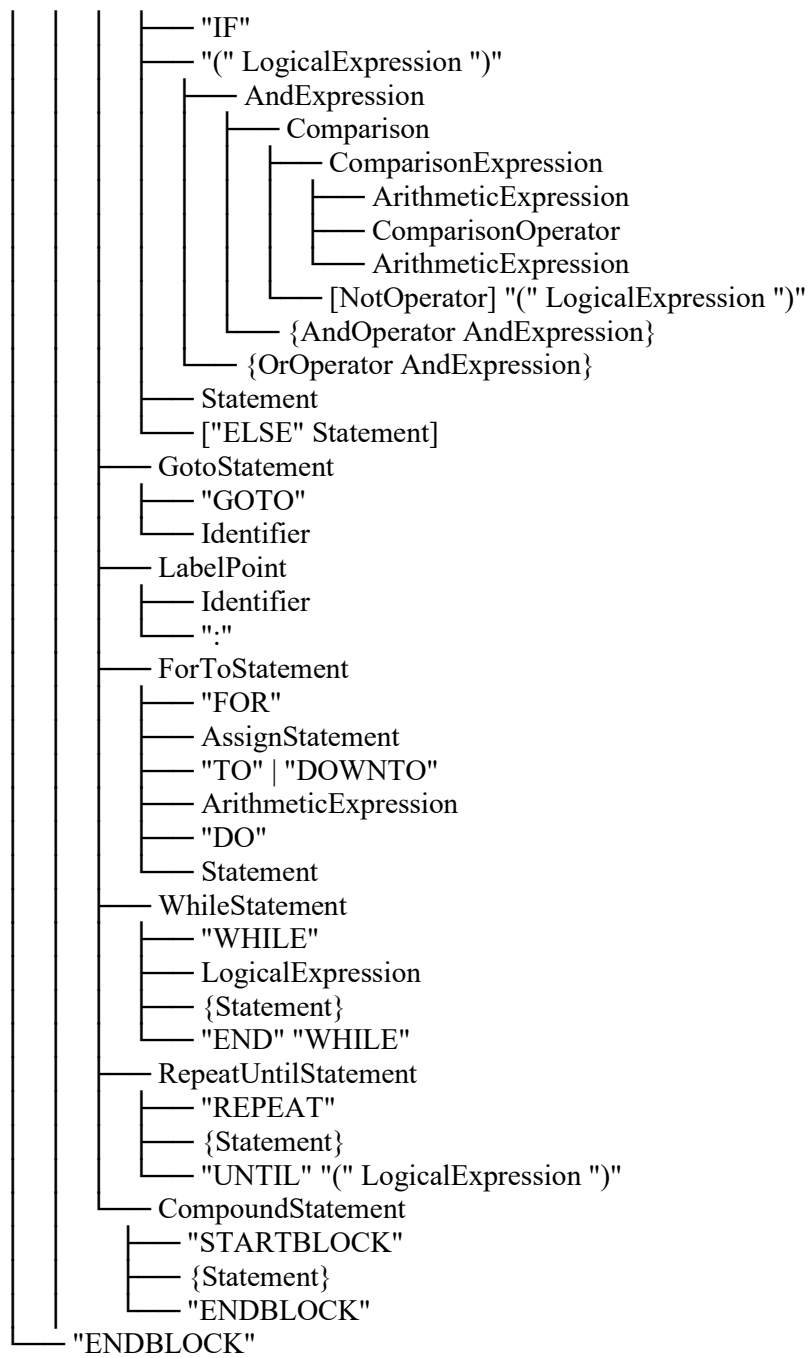
В даному курсовому проєкті генерація коду реалізується як окремий етап. Можливість його виконання є лише за умови, що попередньо успішно виконався етап синтаксичного аналізу. І використовує результат виконання попереднього аналізу, тобто два файли: перший містить згенерований асемблерний код відповідно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

3.4.1. Розробка дерева граматичного розбору.

Схема дерева розбору виглядає наступним чином:

Program





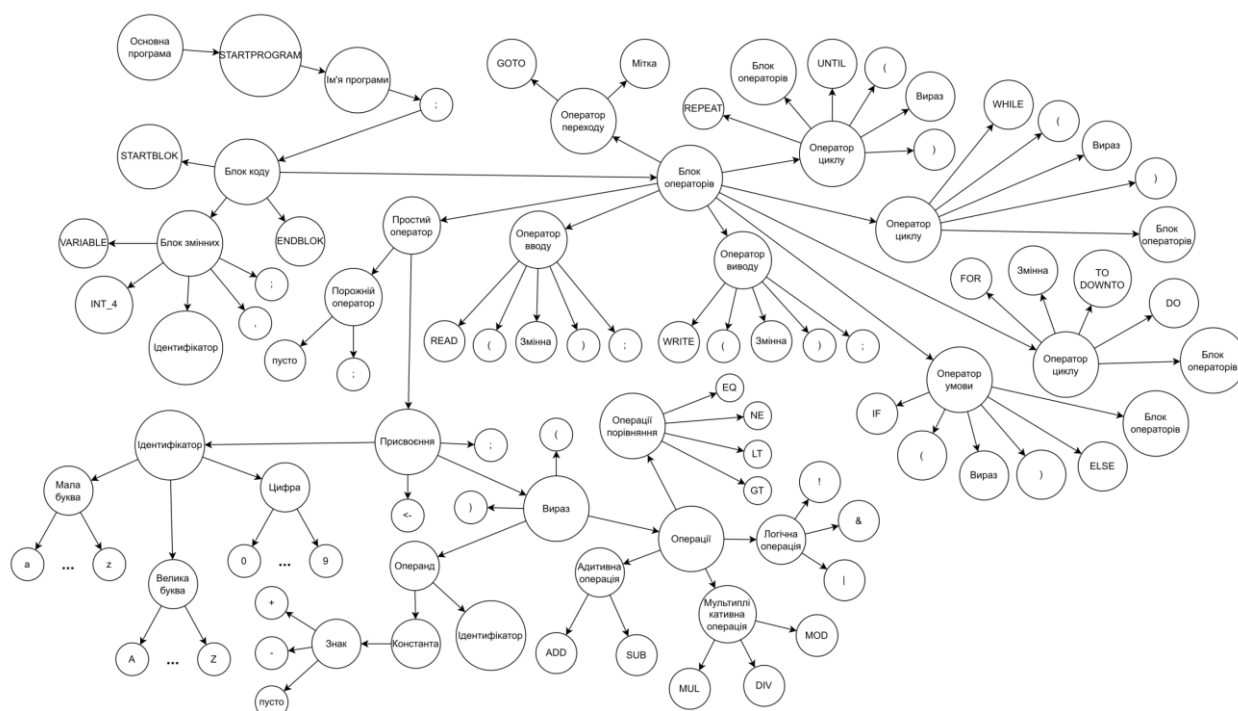


Рис. 3.2. Дерево граматичного розбору.

3.4.1. Розробка алгоритму роботи синтаксичного і семантичного аналізатора.

На вхід синтаксичного аналізатора подіється таблиця лексем створена на етапі лексичного аналізу. Аналізатор проходить по ній і перевіряє чи набір лексем відповідає раніше описаним формам нотації Бекуса-Наура. І разі не відповідності у файл з помилками виводиться інформація про помилку і про рядок на якій вона знаходиться.

При знаходженні оператора присвоєння або математичних виразів здійснюється перевірка балансу дужок(кількість відкриваючих дужок має дорівнювати кількості закриваючих). Також здійснюється перевірка чи не йдуть підряд декілька лексем одного типу

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

В основі синтаксичного аналізатора лежить розпізнавач тексту вхідної програми на основі граматики вхідної мови.

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:

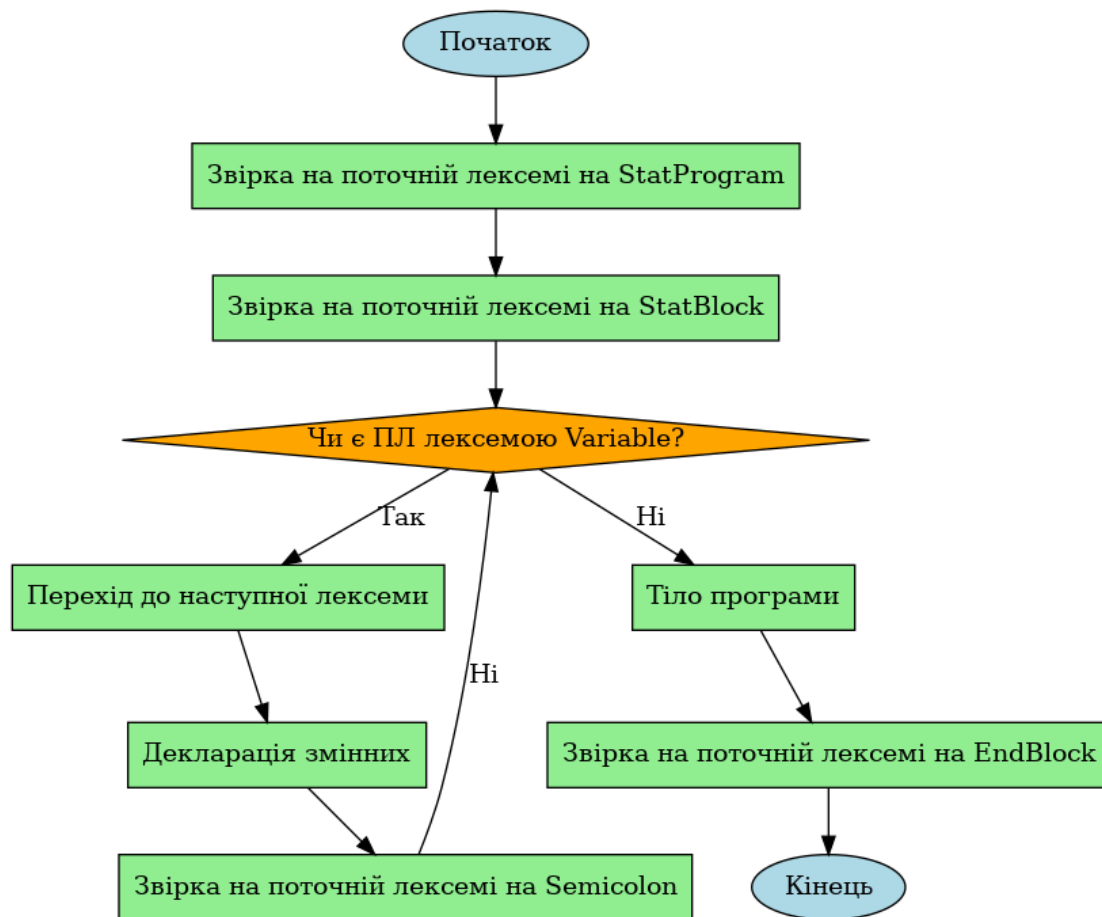


Рис. 3.3. Блок-схема алгоритму роботи синтаксичного аналізатора.

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 = "XXXXXXXX";

```

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;
        }
    }
}
```



```

        errorsMsg.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 само по собі відображає лише структуру програми, але для генерації коду необхідна додаткова інформація про змінні (наприклад, їхні адреси), процедури (адреси, рівні) та мітки. Для цього існують два основні підходи:

1. Зберігання інформації у таблицях генератора коду.
2. Зберігання інформації у відповідних вершинах синтаксичного дерева.

Розглянемо приклад використання таблиць у поєднанні з Лідер-представленням. Лідер-представлення не містить даних про адреси змінних, тому цю інформацію потрібно формувати під час обробки оголошень і записувати в таблиці. Це ж стосується описів масивів, записів тощо.

Таблиці також повинні включати інформацію про процедури: їхні адреси, рівні, модулі, в яких вони описані тощо. При вході у процедуру в таблиці рівнів процедур створюється новий запис — вказівник на таблицю описів. При виході цей вказівник повертається до попереднього значення. У разі використання дерева як проміжного представлення додаткова інформація може зберігатися безпосередньо у вершинах цього дерева.

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:

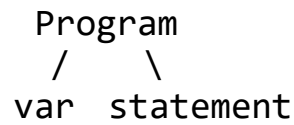
- Підтримуються числові й рядкові типи.
- Для введення/виведення використовується Windows API.
- Передбачена система форматування для різних типів даних.

6. Розширюваність:

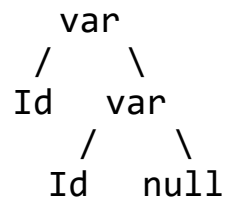
- Легке додавання нових операторів через систему токенів.

- Реєстрація користувацьких процедур.
- Гнучкі налаштування через `GeneratorDetails`.

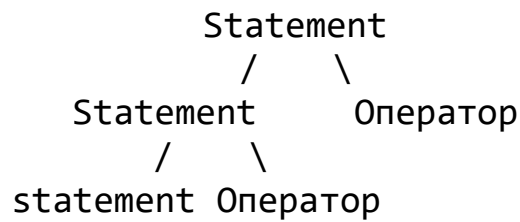
Програма має вигляд:



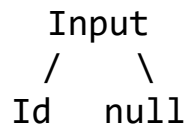
Оголошення змінних:



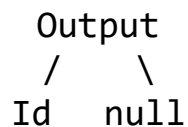
Тіло програми:



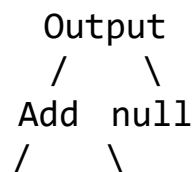
Оператор вводу:



Оператор виводу:

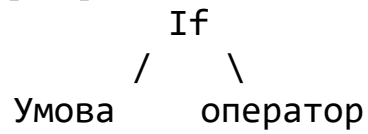


Також оператор виводу може мати за лівого нащадка різні арифметичні вирази, наприклад:

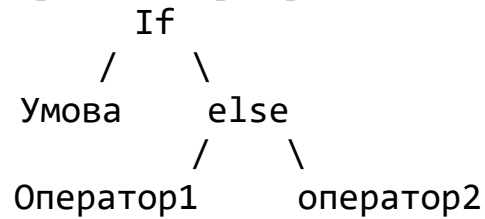


Id num

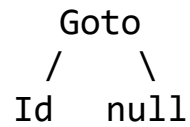
Умовний оператор (IF() оператор;):



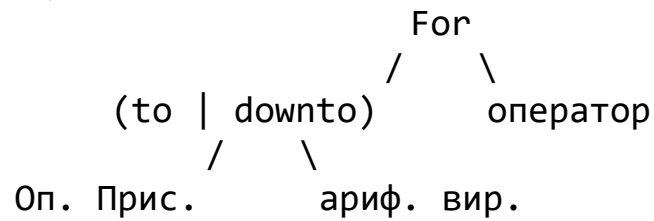
Умовний оператор (IF() оператор1; else оператор2;):



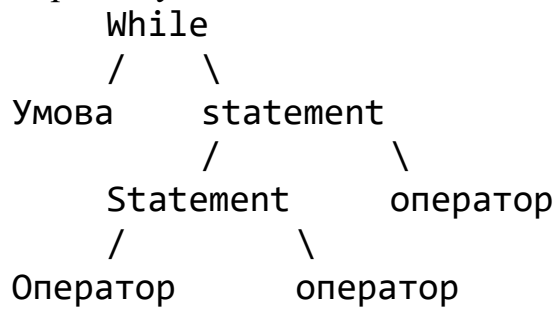
Оператор безумовного переходу:



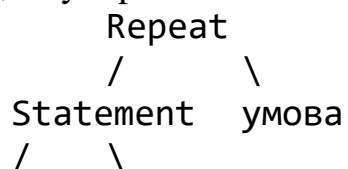
Оператор циклу for:



Оператор циклу while:



Оператор циклу repeat:

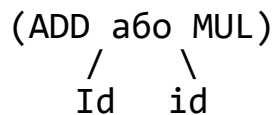


Оператор оператор

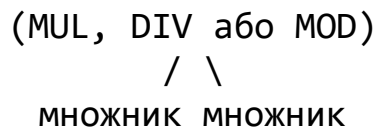
Оператор присвоєння:



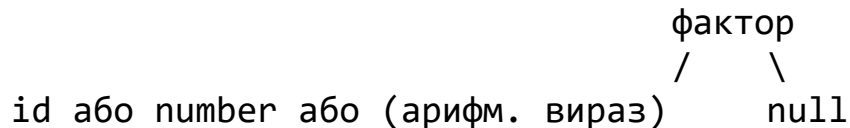
Арифметичний вираз:



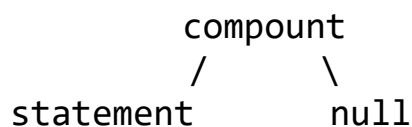
Доданок:



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

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

Кожен token 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 tokenів при аналізі вхідної програми.

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

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

Для класу та 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";
    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:

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 << "Greate_====";
    out << "====\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.

4.1. 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
```

```

VARIABLE 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);

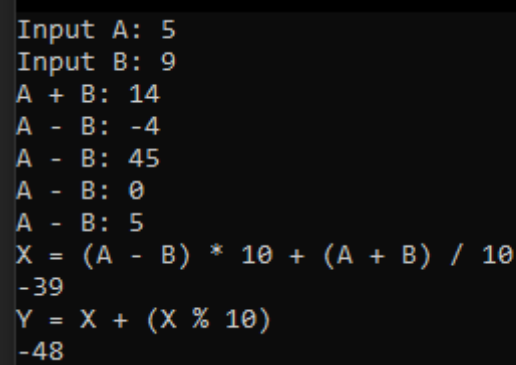
WRITE("\nY = X + (X MOD 10)\n");

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

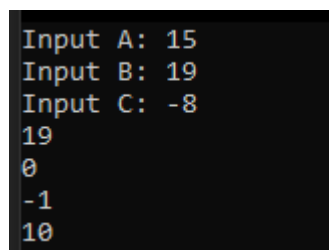
    WRITE(0);

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)	RBracket)	54
218	STARTBLOK	Start	STARTBLOK	55
219	WRITE	Write	WRITE	56
220	(LBracket	(56
221		Number	10	56
222)	RBracket)	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	(LBracket	(60
229		Number	0	60
230)	RBracket)	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 "stdafx.h"
#include "Controller.h"
#include "Core/Parser/TokenRegister.h"
#include "Core/Parser/TokenParser.h"
#include "Core/Generator/Generator.h"

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

        const std::string extention = ".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->c_str();
}

Init();

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

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

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

std::fstream lexemsFile(lexemsFileName, std::ios::out);
TokenParser::PrintTokens(lexemsFile, tokens);
lexemsFile.close();
std::cout << "Report file: " << lexemsFileName << std::endl;
std::cout << longLine << std::endl;

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

std::fstream tokensFile(tokensFileName, std::ios::out);

```

```

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

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

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

system("pause");
return 0;
}

```

BackusRule.h

```

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

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

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

bool BackusRule::check(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
std::list<std::shared_ptr<IBackusRule>>::iterator& it,
std::list<std::shared_ptr<IBackusRule>>::iterator& end)
{
    bool res = true;
    bool pairItem = false;

```

```

auto ruleBegin = it;
for (auto item = m_backusItem.begin(); item != m_backusItem.end(); ++item)
{
    if (it == end || !pairItem && HasFlag(item->policy(), RuleCountPolicy::PairEnd))
    {
        if (!HasFlag(item->policy(), RuleCountPolicy::Optional) || item !=
m_backusItem.end())
        {
            std::vector<std::string> types;

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

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

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

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

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

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

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

return res;
}

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

```

```

        res &= resItem;
        resItem = checkItem(errorsInfo, it, end, item);

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

    return res;
}

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

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

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

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

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

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

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

    return res;
}

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

```

BackusRuleBase.h

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

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

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

BackusRuleItem.h

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

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

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

        return m_rules;
    };

    RuleCountPolicy policy() const { return m_policy; };

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

BackusRuleStorage.h

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

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

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

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

            rules.push_back(it->second);
        }

        return rules;
    };

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

IBackusRule.h

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

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

DEFINE_ENUM_FLAG_OPERATORS(RuleCountPolicy)

__interface IBackusRule : public IItem
{
    virtual bool check(std::multimap<int, std::pair<std::string,
std::vector<std::string>>>& errorsInfo,
```

```
std::list<std::shared_ptr<IBackusRule>>::iterator& it,
std::list<std::shared_ptr<IBackusRule>>::iterator& end) = 0;
```

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

BackusRule.cpp

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

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

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

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

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

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

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

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



```

    }

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

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

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

return res;
}

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

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

    return res;
}

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

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

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

            if (res)

```

```

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

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

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

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

return res;
}

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

```

Generator.h

```

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

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

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

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

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

```

```

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

protected:
    Generator() = default;

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

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

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

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

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

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

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

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

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

        std::string item;
        size_t start = 0;
        size_t end;

```

```

std::string delimiter = "\\n";

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

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

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

m_userStringData[name] += "0";
}

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

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

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

private:
    GeneratorArgs m_args;

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

    static constexpr size_t m_kRetAddrSize = 4;
};

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

template <class T>
class GeneratorItemBase : public IGeneratorItem

```

```

{
public:
    virtual ~GeneratorItemBase() = default;

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

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

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

    static bool registered;

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

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

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

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

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

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

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

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

    std::list<std::shared_ptr<IGeneratorItem>> ConvertToPostfixForm(
        std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,

```

```

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

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

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

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

        ++it;
    }

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

    return postfixForm;
}

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

```

```

        out << "\tpop " << args.regPrefix << "ax\n";
        out << "\tpush ecx\n";
    }

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

        return res;
    }

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

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

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

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

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

        return leftPriority > rightPriority;
    }

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

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

        return res;
    }

    static bool IsNextTokenOnNextLine(const
std::list<std::shared_ptr<IGeneratorItem>>::iterator& it,
        const std::list<std::shared_ptr<IGeneratorItem>>::iterator& end)
    {

```

```

        auto res = false;
        if (it != end && std::next(it) != end && ((*it)->line() + 1) ==
(*std::next(it))->line())
            res = true;

        return res;
    }

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

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

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

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

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

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

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

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

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

```



```

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

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

        ++it;
    }

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

    return postfixForm;
}

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

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

```

```

        if (it != end && std::next(it) != end && (*std::next(it))->type() == type)
            res = true;

        return res;
    }

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

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

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

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

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

        return leftPriority > rightPriority;
    }

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

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

        return res;
    }

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

        return res;
    }

private:
    std::map<std::string, size_t> m_operations;

```

```

    std::set<std::string> m_operands;
    std::set<std::string> m_equationEnd;
    std::string m_lBracketType;
    std::string m_rBracketType;
};

Generator.cpp

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

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

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

    out << "includelib masm32\\lib\\kernel32.lib\n";
    out << "includelib masm32\\lib\\masm32.lib\n";
    out << "includelib masm32\\lib\\user32.lib\n";
    out << "includelib masm32\\lib\\msvcrt.lib\n";
}

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

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

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

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

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

    for (const auto& [_, data] : details.m_userRawData)

```

```

    {
        out << data << std::endl;
    }
}

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

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

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

    out << "end start\n";
}

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

```

TokenParser.h

```

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

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

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

    void regToken(std::shared_ptr<IToken> token, int priority = NoPriority);
}

```

```

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

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

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

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

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

    size_t colPadding = 1;

    auto widthColumn0 = std::max(kHeaderColumn0.size(), getNumCount(tokens.size()))
+ 2 * colPadding;
    auto widthColumn1 = std::max(kHeaderColumn1.size(), maxLemexeLen) + 2 *
colPadding;
    auto widthColumn2 = std::max(kHeaderColumn2.size(), maxTypeLen) + 2 *
colPadding;
    auto widthColumn3 = std::max(kHeaderColumn3.size(), maxValueLen) + 2 *
colPadding;
    auto widthColumn4 = std::max(kHeaderColumn4.size(), getNumCount(tokens.back()-
>line())) + 2 * colPadding;

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

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

    TablePrinter::PrintTable(out,
        { kHeaderColumn0, kHeaderColumn1, kHeaderColumn2, kHeaderColumn3,
kHeaderColumn4 },
        { widthColumn0, widthColumn1, widthColumn2, widthColumn3, widthColumn4 },
        { TablePrinter::CENTRE, TablePrinter::RIGHT, TablePrinter::RIGHT ,
TablePrinter::RIGHT , TablePrinter::RIGHT },
        tokens,

```

```

        { getIndex, getLemexe, getType, getValue, getLine },
        colPadding);
    }

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

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

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

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

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

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

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

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

        return mapItem->second;
    };
};

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

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

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

```

```

        recognizeToken(token, curLine);

    if (IsNewLine(ch))
        ++curLine;

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

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

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

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

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

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

        continue;
    }

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

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

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

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

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

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

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

```

```

{
    if(rBorder)
        throwIfTokenRegistered(rBorder);

    regToken(lBorder);
    throwIfTokenRegistered(target);

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

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

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

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

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

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

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

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

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

bool TokenParser::isUnchangedTextTokenLast()
{
    if (!m_tokens.empty() && m_unchangedTextTokens.contains(m_tokens.back()->lexeme()))
    {
        auto const& [target, left, right] = m_unchangedTextTokens[m_tokens.back()-
>lexeme()];
    }
}

```



```

        if (m_tokens.size() >= 2)
        {
            if (target->type() != (*(++m_tokens.rbegin()))->type())
                return true;
        }
        else
            return true;
    }

    return false;
}

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

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

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

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

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

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

void Init();

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

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

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

```

```

end = --rules.end();

std::multimap<int, std::string> 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;
        errorsMsg.emplace(rule->line(), std::format("Lexical error: Unknown token:
{}", rule->value()));
    }
}

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

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

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

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

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

```

```

    }
}

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

    return res;
}

```

```

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

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

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

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

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

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

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

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

```

```

public:
    Do() { setLexeme("D0"); };
    virtual ~Do() = default;

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

```

```

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

```

```

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

```

```

    BASE_ITEM

```

```

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

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

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

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

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

```

```

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

```

```

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

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

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

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

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

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

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

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

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

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

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

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

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

        bool increment = false;

```

```

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

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

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

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

return forToOrDownToDoRule;
}

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

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

public:
    To() { setLexeme("T0"); };
    virtual ~To() = default;

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

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

```

```

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

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

Goto:

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

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

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

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

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

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

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

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

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

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

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

BackusRulePtr MakeLabel(std::shared_ptr<Controller> controller)
{

```

```

using enum ItemType;

controller->regItem<Label>(Rule);

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

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

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

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

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

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

return labelRule;
}

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

```



```

{
    controller->regItem<Goto>();

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

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

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

    return gotoStatement;
}

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

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

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

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

```

```

    {
        out << customData() << ":" << std::endl;
    };
};

```

IfElse:

```

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

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

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

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

```

```

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

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

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

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

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

        out << "\tpop " << details.args().regPrefix << "ax" << std::endl;
    };
};

```

```

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

```

```

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

```

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

```

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

```

```

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

```

```

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

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

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

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

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

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

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

```

```

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

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

    return ifStatement;
}

```

RepeatUntil:

Repeat.h

```

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

```

```

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

```

```

    BASE_ITEM

```

```

public:

```

```

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

```

```

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

```

RepeatUntilRule.h

```

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

```

```

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

```

RepeatUntilRule.cpp

```

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

```

```

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

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

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

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

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

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

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

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

    return repeatUntilRule;
}

```

```

Until.h
#pragma once

```

```

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

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

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

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

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

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

```

While:

```

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

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

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

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

```

```

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

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

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

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

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

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

```

```

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

```

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

```

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

```

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

```
#include "SimpleTokens.h"
```

```

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

```

```

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

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

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

```

```

BackusRuleItem({                While::Type()}, OnlyOne)
});

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

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

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

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

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

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

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

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

```



```

        (*itr)->setCustomData(endLabel, "endLabel");
    }
    else if(type == ContinueWhile::Type() && count == 1 && itr != it &&
(*std::next(itr, 1))->type() == While::Type())
    {
        itr++;
        (*itr)->setCustomData("true", "ContinueWhile");
        (*itr)->setCustomData(startLabel, "startLabel");
    }
    else if (type == lEnd && count > 0)
    {
        count--;
    }
}
});

return whileCStatement;
}

```

```

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

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

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

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

```

```

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

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

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

```

```

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

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

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

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

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

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

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

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

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

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

```

```

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

        out << "\tpop " << ident->customData() << std::endl;
    };
};

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

#include "Rules/AssignmentRule/Assignment.h"

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

    auto context = controller->context();

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

    return assingmentRule;
}

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

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

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

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

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

```

```

        SetRegistered();
    }
}

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

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

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

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

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

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

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

```

```

        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\n";
        out << "Sub_ ENDP\n";
        out <<
";=====
=====\\n";
    }
};

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

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

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

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

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

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

```

```

        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\n";
        out << "Equal_ ENDP\n";
        out <<
";=====
=====\\n";
    }
};

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

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

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

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

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

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

```

```

        out << "\tret\n";
        out << "Greate_ ENDP\n";
        out <<
";=====
=====\\n";

    }
};

```

Less.h

```

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

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

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

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

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

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

```

```

        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\n";
        out << "Less_ ENDP\n";
        out <<
";=====
=====\\n";

    }
};

```

```

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

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

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

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

```

```

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

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

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

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

```



```

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

```

```

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

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

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

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

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

```

```

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

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

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

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

```

```

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

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

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

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

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

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

```

```

        out << "\tmov " << args.regPrefix << "ax, " << args.regPrefix << "dx\n";
        out << "\tpush cx\n";
        out << "\tpopf\n\n";
        GeneratorUtils::PrintResultToStack(out, args);
        out << "\tret\n";
        out << "Mod_ ENDP\n";
        out <<
";=====
=====\n";
    }
};

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

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

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

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

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

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

```

```

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

#include "Rules/EquationRule/Number.h"

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

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

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

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

#include "Rules/EquationRule/Not.h"

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

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

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

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

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

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

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

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

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

    signedNumber->setPostHandler([](BackusRuleList::iterator&,
BackusRuleList::iterator& it,
BackusRuleList::iterator& end)
    {
        auto begRuleIt = std::prev(it, 2);

```

```

        if ((*begRuleIt)->type() == Symbols::Minus)
        {
            it = begRuleIt;
            end = std::remove(it, end, *it);
            (*it)->setValue('-' + (*it)->value());
            it++;
        }
    });

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

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

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

    auto equationWithBrakets = controller->addRule("EquationWithBrakets", {
    BackusRuleItem({ Symbols::LBraket }, OnlyOne | PairStart),
    BackusRuleItem({ equationRuleName }, OnlyOne),
    BackusRuleItem({ Symbols::RBraket }, OnlyOne | PairEnd)
    });

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

    return equation;
}

```

```

IdentRule:
Identifier.h

```

```

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

```

```

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

```

```

    BASE_ITEM

```

```

public:
    Identifier() { setLexeme(""); };

```



```

virtual ~Identifier() = default;

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

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

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

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

    return token;
};

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

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

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

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

SimpleToken(ProgramName, "");

BackusRulePtr MakeIdentRule(std::shared_ptr<Controller> controller)

```

```

{
    using enum ItemType;

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

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

    auto context = controller->context();

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

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

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

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

    return identRule;
}

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

```

```

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

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

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

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

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

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

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

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

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

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

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

```

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

ReadRule.cpp

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

```
#include "ReadRule.h"
```

```
#include "Rules/ReadRule/Read.h"
```

```
BackusRulePtr MakeReadRule(std::shared_ptr<Controller> controller)
```

```
{
    controller->regItem<Read>();

    auto context = controller->context();

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

    return read;
}
```

StringRule:

String.h

```
#pragma once
```

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

```
#include "Core/Tokens/TokenBase.hpp"
```

```
#include "Core/Backus/BackusRuleBase.h"
```

```
#include "Core/Generator/GeneratorItemBase.h"
```

```
class String : public TokenBase<String>, public BackusRuleBase<String>, public
GeneratorItemBase<String>
```

```
{
```

```
    BASE_ITEM
```

```
public:
```

```
    String() { setLexeme(""); };
```

```
    virtual ~String() = default;
```

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

```
    std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const override
{
```

```
        auto token = clone();
```

```

        token->setValue(lexeme);
        lexeme.clear();
        return token;
    };

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

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

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

#include "Rules/StringRule/String.h"

SimpleToken(Quotes, "\\");

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

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

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

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

    return stringRule;
}

VarsBlockRule:
VarsBlockRule.cpp
#include "stdafx.h"
#include "VarsBlokRule.h"

#include "Rules/VarsBlokRule/VarType.h"

```

```

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

    auto context = controller->context();

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

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

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

        context->SetVarBlockChecked();
    });

    return varsBlok;
}

```

```

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

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

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

```

```

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

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

```

```

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

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

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

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

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

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

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

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

```

WriteRule.cpp

```

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

#include "Rules/StringRule/StringRule.h"

#include "Rules/WriteRule/Write.h"

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

    auto context = controller->context();

    auto stringRule = MakeStringRule(controller);

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

    return write;
}

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

#include "Symbols.h"

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

class Context
{
public:
    std::string IdentRuleName() const { return "IdentRule"; }
    std::string EquationRuleName() const { return "Equation"; }
    std::string OperatorsRuleName() const { return "OperatorsRule"; }
    std::string OperatorsName() const { return "Operators"; }
    std::string OperatorsWithSemicolonsName() const { return "OperatorsWithSemicolon"; }
    std::string AssignmentRuleName() const { return "AssignmentRule"; }
    std::tuple<std::string, std::string, std::string> CodeBlockTypes() const { return {
        "Start", "CodeBlok", "End" }; }

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

    bool IsFirstProgName() const { return true; }

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

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

```



```

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

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

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

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

public:
    void init();

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

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

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

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

    BackusRulePtr topRule();

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

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

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

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

private:

```

```

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

    std::shared_ptr<Context> m_context;
};

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

#include "SimpleTokens.h"

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

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

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

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

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

    auto tokenType = token->type();

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

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

    return rule;
}

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

    return m_topRule;
}

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

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

    controller->regItem<Comma>();
    controller->regItem<Colon>();
    controller->regItem<Semicolon>(TokenAndRule | EquationEnd);
}

```

```

controller->regItem<LBraket>(TokenAndRule | LBracket);
controller->regItem<RBraket>(TokenAndRule | RBracket);
controller->regItem<Plus>();
controller->regItem<Minus>();

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

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

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

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

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

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

return topRule;
}

```

Додаток В(Код на мові Асемблер)
Prog1.asm

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User

Data=====

=====

_aaaa_dd	0	
bbbb	dd	0
xxxx	dd	0
yyyy	dd	0
DivErrMsg	db	13, 10, "Division: Error: division by zero", 0
ModErrMsg	db	13, 10, "Mod: Error: division by zero", 0
String_0	db	"Input A: ", 0
String_1	db	"Input B: ", 0
String_2	db	"A + B: ", 0
String_3	db	13, 10, "A - B: ", 0
String_4	db	13, 10, "A * B: ", 0
String_5	db	13, 10, "A / B: ", 0
String_6	db	13, 10, "A % B: ", 0
String_7	db	13, 10, "X = (A - B) * 10 + (A + B) / 10", 13, 10, 0
String_8	db	13, 10, "Y = X + (X MOD 10)", 13, 10, 0

;===Addition

Data=====

=====

hConsoleInput	dd	?
hConsoleOutput	dd	?
endBuff	db	5 dup (?)
msg1310	db	13, 10, 0
CharsReadNum	dd	?
InputBuf	db	15 dup (?)
OutMessage	db	"%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 _aaaaaaaa_, 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 _aaaaaaaa_
    push _bbbbbbbbb_
    call Add_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
    push _aaaaaaaa_
    push _bbbbbbbbb_
    call Sub_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
    push _aaaaaaaa_
    push _bbbbbbbbb_
    call Mul_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
    push _aaaaaaaa_
    push _bbbbbbbbb_
    call Div_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6 - 1, 0, 0
    push _aaaaaaaa_
    push _bbbbbbbbb_
    call Mod_
    call Output_
    push _aaaaaaaa_
    push _bbbbbbbbb_
    call Sub_
    push dword ptr 10
    call Mul_
    push _aaaaaaaa_
    push _bbbbbbbbb_
    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 endBuff, 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

```

```

    jge gr
lo:
    mov edx, -1
    jmp less_fin
gr:
    mov edx, 0
less_fin:
    mov eax, [esp + 8]
    idiv dword ptr [esp + 4]
    push cx
    popf

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

```

```

;=====
=====

```

```

;===Procedure

```

```

Input=====
=====

```

```

    Input_ PROC
        invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0
        invoke crt_atoi, ADDR InputBuf
        ret
    Input_ ENDP

```

```

;=====
=====

```

```

;===Procedure

```

```

Mod=====
=====

```

```

    Mod_ PROC
        pushf
        pop cx

        mov 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

```



```

    jge gr
lo:
    mov edx, -1
    jmp less_fin
gr:
    mov edx, 0
less_fin:
    mov eax, [esp + 8]
    idiv dword ptr [esp + 4]
    mov eax, edx
    push cx
    popf

    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 wsprintf, ADDR ResMessage, ADDR OutMessage, value
    invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 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\masm32.inc
```

```
include masm32\include\user32.inc
```

```
include masm32\include\msvcrt.inc
```

```
includelib masm32\lib\kernel32.lib
```

```
includelib masm32\lib\masm32.lib
```

```
includelib masm32\lib\user32.lib
```

```
includelib masm32\lib\msvcrt.lib
```

```
.DATA
```

```
;===User
```

```
Data=====
```

```
=====
```

```
    _aaaa_ dd    0
```

```
    _bbbb_ dd    0
```

```
    _cccc_ 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=====
```

```
=====
```

```

hConsoleInputdd    ?
hConsoleOutput    dd    ?
endBuff           db     5 dup (?)
msg1310           db     13, 10, 0

```

```

CharsReadNum      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 _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 endIf2

push _aaaaaaa_

push _cccccccc_

call Create_

pop eax

cmp eax, 0

je elseLabel1

jmp _temporal_

jmp endIf1

elseLabel1:

push _cccccccc_

call Output_

jmp _outugoto_

temporal:

push _aaaaaaa_

call Output_

jmp _outugoto_

endIf1:

```

endif2:
    push _bbbbbbbb_
    push _ccccccc_
    call Less_
    pop eax
    cmp eax, 0
    je elseLabel3
    push _ccccccc_
    call Output_
    jmp endif3
elseLabel3:
    push _bbbbbbbb_
    call Output_
endif3:
_outugoto_:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
    push _aaaaaaaa_
    push _bbbbbbbb_
    call Equal_
    push _aaaaaaaa_
    push _ccccccc_
    call Equal_
    call And_
    push _bbbbbbbb_
    push _ccccccc_
    call Equal_
    call And_
    pop eax
    cmp eax, 0
    je elseLabel4
    push dword ptr 1
    call Output_
    jmp endif4
elseLabel4:
    push dword ptr 0
    call Output_
endif4:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
    push _aaaaaaaa_
    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 elseLabel5
    push dword ptr -1
    call Output_
    jmp endIf5
elseLabel5:
    push dword ptr 0
    call Output_
endIf5:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
    push _aaaaaaaa_
    push _bbbbbbbb_
    push _ccccccc_
    call Add_
    call Less_
    call Not_
    pop eax
    cmp eax, 0
    je elseLabel6
    push dword ptr 10
    call Output_
    jmp endIf6
elseLabel6:
    push dword ptr 0
    call Output_
endIf6:
exit_label:
    invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0, 0
    invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0
    invoke ExitProcess, 0

```

```

;===Procedure

```

```

Add=====

```

```

=====

```

```

Add_ PROC

```

```

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

```

```

Add_ ENDP

```

```

;=====

```

```

=====

```

```

;===Procedure
And=====
=====
And_ PROC
    pushf
    pop cx

    mov eax, [esp + 8]
    cmp eax, 0
    jnz and_t1
    jz and_false
and_t1:
    mov eax, [esp + 4]
    cmp eax, 0
    jnz and_true
and_false:
    mov eax, 0
    jmp and_fin
and_true:
    mov eax, 1
and_fin:
    push cx
    popf

    mov [esp + 8], eax
    pop ecx
    pop eax
    push ecx
    ret
And_ ENDP
;=====
=====

```

```

;===Procedure
Equal=====
=====
Equal_ PROC
    pushf
    pop cx

    mov eax, [esp + 8]
    cmp eax, [esp + 4]
    jne equal_false
    mov eax, 1
    jmp equal_fin
equal_false:
    mov eax, 0
equal_fin:

```

```

    push cx
    popf

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

```

```

;=====
=====

```

```

;===Procedure

```

```

Greate=====
===

```

```

Greate_ PROC
    pushf
    pop cx

    mov eax, [esp + 8]
    cmp eax, [esp + 4]
    jle greate_false
    mov eax, 1
    jmp greate_fin
greate_false:
    mov eax, 0
greate_fin:
    push cx
    popf

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

```

```

;=====
=====

```

```

;===Procedure

```

```

Input=====
===

```

```

Input_ PROC
    invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0
    invoke crt_atoi, ADDR InputBuf
    ret
Input_ ENDP

```

```
;=====
```

```
=====
```

```
;===Procedure
```

```
Less=====
```

```
===
```

```
Less_ PROC
```

```
    pushf
```

```
    pop cx
```

```
    mov 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
```

```
Not=====
```

```
=====
```

```
Not_ PROC
```

```
    pushf
```

```
    pop cx
```

```
    mov eax, [esp + 4]
```

```
    cmp eax, 0
```

```
    jnz not_false
```

```
not_t1:
```

```
    mov eax, 1
```

```
    jmp not_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
    jz or_t1
or_t1:
    mov eax, [esp + 4]
    cmp eax, 0
    jnz or_true
or_false:
    mov eax, 0
    jmp or_fin
or_true:
    mov eax, 1
or_fin:
    push cx
    popf

    mov [esp + 8], eax
    pop ecx
    pop eax
    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\masm32.inc  
include masm32\include\user32.inc  
include masm32\include\msvcrt.inc  
includelib masm32\lib\kernel32.lib  
includelib masm32\lib\masm32.lib  
includelib masm32\lib\user32.lib  
includelib masm32\lib\msvcrt.lib
```

```
.DATA  
;===User
```

```
Data=====
```

```
=====
```

_aaa2_dd	0	
_aaaa_dd	0	
bbbb	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	?	
endBuff	db	5 dup (?)	
msg1310	db	13, 10, 0	

```

CharsReadNum    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 _aaaaaaaa_, 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 _aaaaaaaa_
```

```
    pop _aaaaaaa2_
```

```
forPasStart1:
```

```
    push _bbbbbbbbb_
```

```
    push _aaaaaaa2_
```

```
    call Less_
```

```
    call Not_
```

```
    pop eax
```

```
    cmp eax, 0
```

```
    je forPasEnd1
```

```
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3 - 1, 0, 0
```

```
    push _aaaaaaa2_
```

```
    push _aaaaaaa2_
```

```
    call Mul_
```

```
    call Output_
```

```
    push _aaaaaaa2_
```

```
    push dword ptr 1
```

```
    call Add_
```

```
    pop _aaaaaaa2_
```

```
    jmp forPasStart1
```

```
forPasEnd1:
```

```
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4 - 1, 0, 0
```

```
    push _bbbbbbbbb_
```

```
    pop _aaaaaaa2_
```

```
forPasStart2:
```

```
    push _aaaaaaaa_
```

```
    push _aaaaaaa2_
```

```
    call Create_
```

```
    call Not_
```

```

    pop eax
    cmp eax, 0
    je forPasEnd2
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5 - 1, 0, 0
    push _aaaaaaa2_
    push _aaaaaaa2_
    call Mul_
    call Output_
    push _aaaaaaa2_
    push dword ptr 1
    call Sub_
    pop _aaaaaaa2_
    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 _ccccccc1_
whileStart2:
    push _ccccccc1_
    push _aaaaaaa_
    call Less_
    pop eax
    cmp eax, 0
    je whileEnd2
    push dword ptr 0
    pop _ccccccc2_
whileStart1:
    push _ccccccc2_
    push _bbbbbbbb_
    call Less_
    pop eax
    cmp eax, 0
    je whileEnd1
    push _xxxxxxx_
    push dword ptr 1
    call Add_
    pop _xxxxxxx_
    push _ccccccc2_
    push dword ptr 1
    call Add_
    pop _ccccccc2_
    jmp whileStart1
whileEnd1:
    push _ccccccc1_
    push dword ptr 1
    call Add_
    pop _ccccccc1_

```

```

    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 _ccccccc1_
repeatStart2:
    push dword ptr 1
    pop _ccccccc2_
repeatStart1:
    push _xxxxxxx_
    push dword ptr 1
    call Add_
    pop _xxxxxxx_
    push _ccccccc2_
    push dword ptr 1
    call Add_
    pop _ccccccc2_
    push _ccccccc2_
    push _bbbbbbbbb_
    call Greate_
    call Not_
    pop eax
    cmp eax, 0
    je repeatEnd1
    jmp repeatStart1
repeatEnd1:
    push _ccccccc1_
    push dword ptr 1
    call Add_
    pop _ccccccc1_
    push _ccccccc1_
    push _aaaaaaaa_
    call Greate_
    call Not_
    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 endBuff, 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
Greate=====
=====
Greate_ PROC
    pushf
    pop cx

    mov eax, [esp + 8]
    cmp eax, [esp + 4]
    jle greate_false
    mov eax, 1
    jmp greate_fin
greate_false:
    mov eax, 0
greate_fin:
    push cx
    popf

    mov [esp + 8], eax
    pop ecx
    pop eax
    push ecx
    ret
Greate_ ENDP
;=====
=====

```

```

;===Procedure
Input=====
=====

```

```

Input_ PROC
    invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0
    invoke crt_atoi, ADDR InputBuf
    ret

```

```

Input_ ENDP

```

```

;=====

```

```

=====

```

```

;===Procedure

```

```

Less=====

```

```

===

```

```

Less_ PROC

```

```

    pushf
    pop cx

```

```

    mov 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 eax
    push ecx
    ret

```

```

Mul_ ENDP

```

```

;=====

```

```

=====

```

```

;===Procedure

```

```

Not=====

```

```

=====

```

```

    Not_ PROC

```

```
        pushf

```

```
        pop cx

```

```

        mov eax, [esp + 4]

```

```
        cmp eax, 0

```

```
        jnz not_false

```

```
not_t1:
```

```
    mov eax, 1

```

```
    jmp not_fin

```

```
not_false:
```

```
    mov eax, 0

```

```
not_fin:
```

```
    push cx

```

```
    popf

```

```

        mov [esp + 4], eax

```

```
        ret

```

```
Not_ 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

```

```

;=====

```

```

=====

```

```

;===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
```