

Міністерство освіти і науки України
Національний університет “Львівська політехніка”



Курсовий проект

З дисципліни «Системне програмування»

на тему: "Розробка системних програмних модулів
та компонент систем програмування.

Розробка транслятора з вхідної мови програмування"

Варіант №24

Виконав: ст. гр. КІ-307

Петренко В.А.

Перевірив:

Козак Н.Б.

Львів-2024

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 скопіювати за допомогою відповідних програм (наприклад, LINK, ML і т. д.).

Зміст

Анотація.....	2
Завдання до курсового проекту.....	4
Вступ.....	6
1. Огляд методів та способів проектування трансляторів.....	7
2. Формальний опис вхідної мови програмування.....	10
2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура...10	
2.2. Опис термінальних символів та ключових слів.....	15
3. Розробка транслятора вхідної мови програмування.....	17
3.1. Вибір технології програмування.	17
3.2. Проектування таблиць транслятора.....	18
3.3. Розробка лексичного аналізатора.	20
3.3.1. Розробка блок-схеми алгоритму.....	21
3.3.2. Опис програми реалізації лексичного аналізатора.....	21
3.4. Розробка синтаксичного та семантичного аналізатора.	23
3.4.1. Опис програми реалізації синтаксичного та семантичного аналізатора	24
3.4.2. Розробка граф-схеми алгоритму.....	25
3.5. Розробка генератора коду.....	26
3.5.1. Розробка граф-схеми алгоритму.....	27
3.5.2. Опис програми реалізації генератора коду.	28
4. Опис програми.....	29
4.1. Опис інтерфейсу та інструкція користувачеві.....	34
5. Відлагодження та тестування програми..	35
5.1. Виявлення лексичних та синтаксичних помилок.....	35
5.2. Виявлення семантичних помилок.....	37
5.3. Загальна перевірка коректності роботи транслятора.....	37
5.4. Тестова програма №1.....	39
5.5. Тестова програма №2.....	41
5.6. Тестова програма №3.....	43
Висновки.....	47
Список використаної літератури.....	48
Додатки.....	49

Завдання до курсового проекту

Варіант 24

Завдання на курсовий проект

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

В моєму випадку це .p24

Опис вхідної мови програмування:

- Тип даних: integer16
- Блок тіла програми: Maimprogram Start Data...; End
- 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-Low перший символ Up
- Регістр ідентифікаторів: Low6 перший символ _
- Операції арифметичні: ++, --, **, Div, Mod
- Операції порівняння: =, <>, Et, 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).

Вступ

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

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

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

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

1. Огляд методів та способів проектування трансляторів

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

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

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

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

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

Оптимізація коду та генерація коду спрямовані на оптимізацію та створення машинно-залежного коду відповідно.

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Формальний опис вхідної мови програмування

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

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

```
labeled_point = label , ":"
goto_label = tokenGOTO, label, ";"
program_name = ident, ";"
value_type = tokenINTEGER16
other_declaration_ident = tokenCOMMA , ident
declaration = value_type , ident , {other_declaration_ident}
unary_operator = tokenNOT | tokenMINUS | tokenPLUS
unary_operation = unary_operator , expression
binary_operator = tokenAND | tokenOR | tokenEQUAL | tokenNOTEQUAL | tokenLESSOREQUAL
| tokenGREATEROREQUAL | tokenPLUS | tokenMINUS | tokenMUL | tokenDIV | tokenMOD
binary_action = binary_operator , expression
left_expression = group_expression | unary_operation | ident | value
expression = left_expression , {binary_action}
group_expression = tokenGROUPEXPRESSIONBEGIN , expression , tokenGROUPEXPRESSIONEND
//
bind_right_to_left = ident , tokenRLBIND , expression
bind_left_to_right = expression , tokenLRBIND , ident
//
if_expression = expression
body_for_true = {statement} , ";"
body_for_false = tokenELSE , {statement} , ";"
cond_block = tokenIF , tokenGROUPEXPRESSIONBEGIN , if_expression ,
tokenGROUPEXPRESSIONEND , body_for_true , [body_for_false];
//
cycle_begin_expression = expression
cycle_counter = ident
cycle_counter_rl_init = cycle_counter , tokenRLBIND , cycle_begin_expression
cycle_counter_lr_init = cycle_begin_expression , tokenLRBIND , cycle_counter
cycle_counter_init = cycle_counter_rl_init | cycle_counter_lr_init
cycle_counter_last_value = value
cycle_body = tokenDO , statement , {statement}
forto_cycle = tokenFOR , cycle_counter_init , tokenTO , cycle_counter_last_value , cycle_body ,
";"
continue_while = tokenCONTINUE , tokenWHILE
```

```

exit_while = tokenEXIT , tokenWHILE
statement_in_while_body = statement | continue_while | exit_while
while_cycle_head_expression = expression
while_cycle = tokenWHILE , while_cycle_head_expression , {statement_in_while_body} ,
tokenEND , tokenWHILE
//
repeat_until_cycle_cond = group_expression
repeat_until_cycle = tokenREPEAT , {statement} , tokenUNTIL , repeat_until_cycle_cond
input = tokenGET , tokenGROUPEXPRESSIONBEGIN , ident , tokenGROUPEXPRESSIONEND
output = tokenPUT , tokenGROUPEXPRESSIONBEGIN , expression , tokenGROUPEXPRESSIONEND
statement = bind_right_to_left | bind_left_to_right | cond_block | forto_cycle | while_cycle |
repeat_until_cycle | labeled_point | goto_label | input | output
program = tokenNAME , program_name , tokenSEMICOLON , tokenBODY , tokenDATA ,
[declaration] , tokenSEMICOLON , {statement} , tokenEND
//
digit = digit_0 | digit_1 | digit_2 | digit_3 | digit_4 | digit_5 | digit_6 | digit_7 | digit_8 | digit_9
non_zero_digit = digit_1 | digit_2 | digit_3 | digit_4 | digit_5 | digit_6 | digit_7 | digit_8 |
digit_9
unsigned_value = ((non_zero_digit , {digit}) | digit_0)
value = [sign] , unsigned_value
// -- hello wolrd
letter_in_lower_case = 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
    letter_in_upper_case = 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
    ident = tokenUNDERSCORE , letter_in_lower_case , letter_in_lower_case ,
letter_in_lower_case , letter_in_lower_case , letter_in_lower_case
    label = letter_in_lower_case , {letter_in_lower_case}
//
sign = sign_plus | sign_minus
sign_plus = '-'
sign_minus = '+'
//
digit_0 = '0'
digit_1 = '1'
digit_2 = '2'
digit_3 = '3'
digit_4 = '4'
digit_5 = '5'
digit_6 = '6'
digit_7 = '7'
digit_8 = '8'
digit_9 = '9'
//
tokenCOLON = ":"

```

```

tokenGOTO = "Goto"
tokenINTEGER16 = "Integer16"
tokenCOMMA = ","
tokenNOT = "!"
tokenAND = "&"
tokenOR = "|"
tokenEQUAL = "="
tokenNOTEQUAL = "<>"
tokenLESSOREQUAL = "Lt"
tokenGREATEROREQUAL = "Et"
tokenPLUS = "++"
tokenMINUS = "--"
tokenMUL = "***"
tokenDIV = "Div"
tokenMOD = "Mod"
tokenGROUPEXPRESSIONBEGIN = "("

tokenGROUPEXPRESSIONEND = ")"
tokenRLBIND = "<-"
tokenLRBIND = ","
tokenELSE = "Else"
tokenIF = "If"
tokenDO = "Do"
tokenFOR = "For"
tokenTO = "To"
tokenWHILE = "While"
tokenCONTINUE = "Continue"
tokenEXIT = "Exit"
tokenREPEAT = "Repeat"
tokenUNTIL = "Until"
tokenGET = "Read"
tokenPUT = "Write"
tokenNAME = "MainProgram"
tokenBODY = "Start"
tokenDATA = "Body"
tokenEND = "End"
tokenSEMICOLON = ""
//
tokenUNDERSCORE = " _"
//
A = "A"
B = "B"
C = "C"
D = "D"
E = "E"

```

F = "F"
G = "G"
H = "H"
I = "I"
J = "J"
K = "K"
L = "L"
M = "M"
N = "N"
O = "O"
P = "P"
Q = "Q"
R = "R"
S = "S"
T = "T"
U = "U"
V = "V"
W = "W"
X = "X"
Y = "Y"
Z = "Z"
//
a = "a"
b = "b"
c = "c"
d = "d"
e = "e"
f = "f"
g = "g"
h = "h"
i = "i"
j = "j"
k = "k"
l = "l"
m = "m"
n = "n"
o = "o"
p = "p"
q = "q"
r = "r"
s = "s"
t = "t"
u = "u"
v = "v"
w = "w"

```
x = "x"  
y = "y"  
z = "z"  
//
```

2.2 Опис термінальних символів та ключових слів

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

Термінальний символ або ключове слово	Значення
Mainprogram	Початок програми
Start	Початок тексту програми
Data	Початок блоку опису змінних
End	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
Continue	Оператор циклу
Exit	Оператор циклу
Repeat	Початок тіла циклу
Until	Оператор циклу
++	Оператор додавання
--	Оператор віднімання
**	Оператор множення
Div	Оператор ділення
Mod	Оператор знаходження залишку від ділення

=	Оператор перевірки на рівність
<>	Оператор перевірки на нерівність
Lt	Оператор перевірки чи менше
Et	Оператор перевірки чи більше
!	Оператор логічного заперечення
&	Оператор кон'юнкції
	Оператор диз'юнкції
integer16	16-ти розрядні знакові цілі
\$\$...	Коментар
,	Розділювач
;	Ознака кінця оператора
(Відкриваюча дужка
)	Закриваюча дужка

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

3. Розробка транслятора вхідної мови програмування

3.1 Вибір технології програмування

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

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

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

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

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

3.2 Проектування таблиць транслятора

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

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

```
std::multimap<int, std::shared_ptr<IToken>> m_priorityTokens;
```

```
std::string m_lexeme; //Лексема
```

```
std::string m_value; //Значення
```

```
int m_line = -1; //Рядок
```

- 2) Таблиця лексичних класів

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

Таблиця 2 Опис термінальних символів та ключових слів

Токен	Значення
Program	Maimprogram
Start	Start
Vars	Data
End	End
VarType	integer16
Read	Read
Write	Write
Assignment	<-
If	If
Else	Else
Goto	Goto
Colon	:
Label	
For	For
To	To

DownTo	Downto
Do	Do
While	While
WhileContinue	Continue
WhileExit	Exit
Repeat	Repeat
Until	Until
Addition	++
Subtraction	--
Multiplication	**
Division	Div
Mod	Mod
Equal	=
NotEqual	<>
Less	Lt
Greate	Et
Not	!
And	&
Or	
Plus	+
Minus	-
Identifier	
Number	
String	
Undefined	
Unknown	
Comma	,
Quotes	“
Semicolon	;
LBraket	(
RBraket)
LComment	\$\$
Comment	

3.3 Розробка лексичного аналізатора

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

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

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

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

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

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

3.3.1 Розробка блок-схеми алгоритму

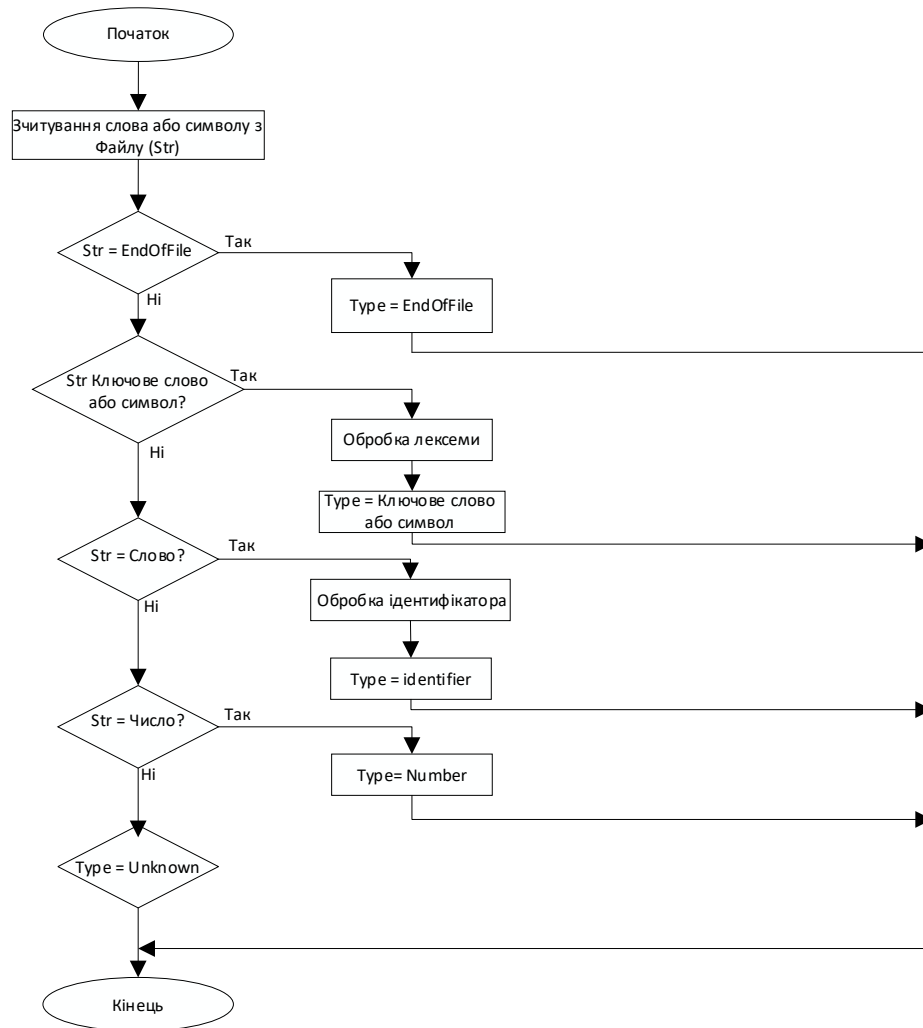


Рис. 3.1 Блок-схема роботи лексичного аналізатора

3.3.2 Опис програми реалізації лексичного аналізатора

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

розпізнаються ідентифікатори, літерали і термінальні символи (операції, ключові слова).

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

При виділенні лексеми вона розпізнається та записується у список `m_tokens` за допомогою відповідного типу лексеми, що є унікальним для кожної лексеми із усього можливого їх набору. Це дає можливість наступним фазам компіляції звертатись до лексеми не як до послідовності символів, а як до унікального типу лексеми, що значно спрощує роботу синтаксичного аналізатора: легко перевіряти належність лексеми до відповідної синтаксичної конструкції та є можливість легкого перегляду програми, як вгору, так і вниз, від поточної позиції аналізу. Також в таблиці лексем ведуться записи, щодо рядка відповідної лексеми – для місця помилки – та додаткова інформація.

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

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

3.4 Розробка синтаксичного та семантичного аналізатора

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

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

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

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

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

3.4.1 Опис програми реалізації синтаксичного та 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.

3.4.2 Розробка граф-схеми алгоритму

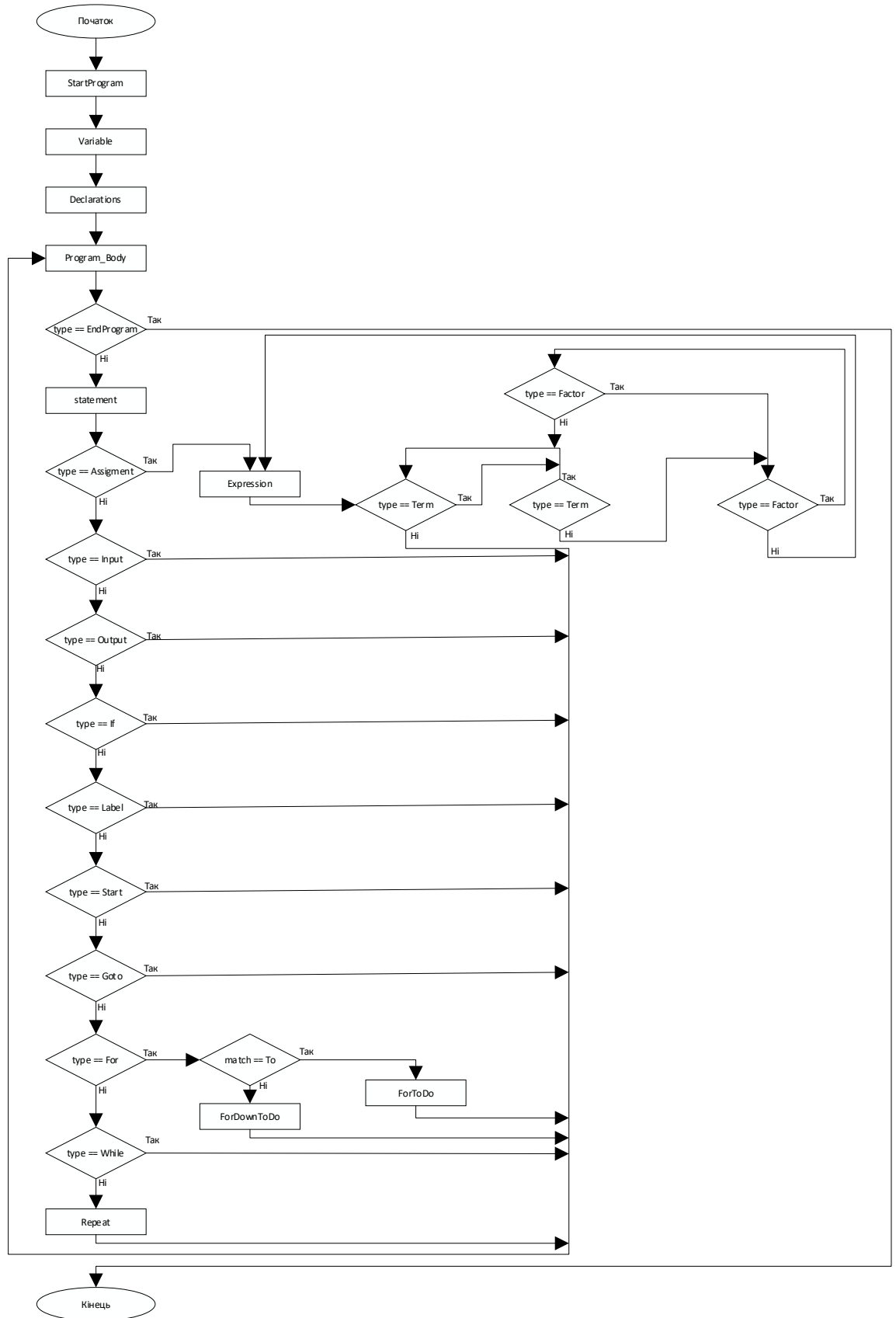


Рис. 3.2 Граф-схема роботи синтаксичного аналізатора

3.5 Розробка генератора коду

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

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

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

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

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

В даному курсовому 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 конструктори записуються в файл asm.

3.5.1 Розробка граф-схеми алгоритму

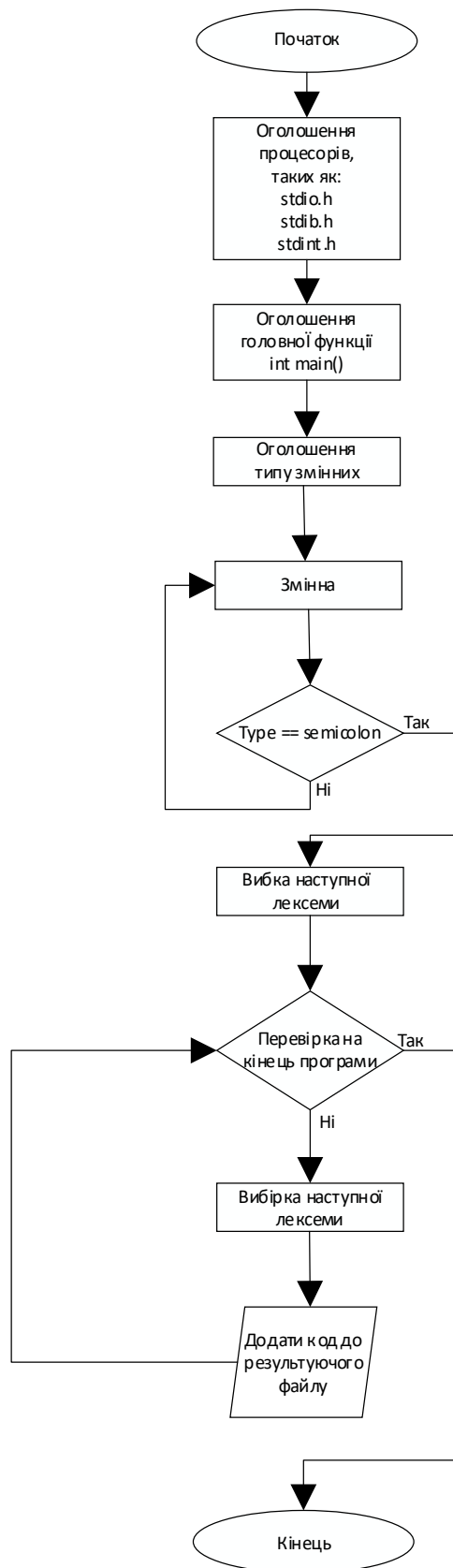


Рис. 3.3 Блок схема генератора коду

3.5.2 Опис програми реалізації генератора коду

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

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

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

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

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

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

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

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

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

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

4. Опис програми

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

```
auto 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 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 << "====Procedure
Greate=====
=====\\n";
    out << "Greate_ PROC\\n";
    out << "\\tpushf\\n";
    out << "\\tpop cx\\n\\n";
    out << "\\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\\n";
    out << "\\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\\n";
    out << "\\tjle greate_false\\n";
    out << "\\tmov " << args.regPrefix << "ax, 1\\n";
    out << "\\tjmp greate_fin\\n";
    out << "greate_false:\\n";
    out << "\\tmov " << args.regPrefix << "ax, 0\\n";
    out << "greate_fin:\\n";
    out << "\\tpush cx\\n";
    out << "\\tpopf\\n\\n";
}

```



```

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Greate_ ENDP\n";

out <<
";=====
=====\\n";

}

```

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

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

4.1 Опис інтерфейсу та інструкція користувачеві

Вхідним файлом для даної програми є звичайний текстовий файл з розширенням p24. 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_p24.exe <ім'я програми>.p24"

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

5. Відлагодження та тестування програми

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

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

5.1 Виявлення лексичних та синтаксичних помилок

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

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

Приклад виявлення:

Текст програми з помилками

```
$$Prog1
Mainprogram
Start
Data Integer16 _aaaaaa,_bbbb,_xxxxxx,_yyyyyy;
Write("Read _aaaaaa: ");
Re ad(_aaaaaa);
Write("Read _bbbbbb: ");
Read(_bbbbbb);
```

```

Write("_aaaaaa + _bbbbbb: ");
Write(_aaaaaa ++ _bbbbbb);
Write("\n_Aaaaaaaa - _bbbbbb: ");
Write(_aaaaaa -- _bbbbbb);
Write("\n_Aaaaaaaa * _bbbbbb: ");
Write(_aaaaaa ** _bbbbbb);
Write("\n_Aaaaaaaa / _bbbbbb: ");
Write(_aaaaaa Div _bbbbbb);
Write("\n_Aaaaaaaa % _bbbbbb: ");
Write(_aaaaaa Mod _bbbbbb);
_xxxxxx<-( _aaaaaa -- _bbbbbb) ** 10 ++ ( _aaaaaa ++ _bbbbbb) Div 10;
_yyyyyy<_xxxxxx ++ (_xxxxxx Mod 10);
Write("\n_xxxxxxx = ( _aaaaaa - _bbbbbb) * 10 + ( _aaaaaa + _bbbbbb) / 10\n");
Write(_xxxxxx);
Write("\n_yyyyyyy = _xxxxxx + (_xxxxxx % 10)\n");
Write(_yyyyyy);
End

```

Текст файлу з повідомленнями про помилки

List of errors

```

=====
=====

```

There are 4 lexical errors.

There are 2 syntax errors.

There are 0 semantic errors.

Line 4: Lexical error: Unknown token: _aaaaaa

Line 4: Lexical error: Unknown token: _bbbb

Line 4: Syntax error: Expected: VarsBlok before _aaaaaa

Line 4: Syntax error: Expected: IdentRule before _aaaaaa

Line 6: Lexical error: Unknown token: Re

Line 6: Lexical error: Unknown token: ad

5.2 Виявлення семантичних помилок

Суттю виявлення семантичних помилок є перевірка числових констант на відповідність типу `integer16`, тобто знаковому цілому числу з відповідним діапазоном значень і перевірку на коректність використання змінних `integer16` у цілочисельних і логічних виразах.

5.3 Загальна перевірка коректності роботи транслятора

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

Текст коректної програми

\$\$Prog1

Mainprogram

Start

Data Integer16 _aaaaaa,_bbbbbb,_xxxxxx,_yyyyyy;

Write("Read _aaaaaa: ");

Read(_aaaaaa);

Write("Read _bbbbbb: ");

Read(_bbbbbb);

Write("_aaaaaa + _bbbbbb: ");

Write(_aaaaaa ++ _bbbbbb);

Write("\n_Aaaaaaaa - _bbbbbb: ");

Write(_aaaaaa -- _bbbbbb);

Write("\n_Aaaaaaaa * _bbbbbb: ");

Write(_aaaaaa ** _bbbbbb);

Write("\n_Aaaaaaaa / _bbbbbb: ");

Write(_aaaaaa Div _bbbbbb);

Write("\n_Aaaaaaaa % _bbbbbb: ");

Write(_aaaaaa Mod _bbbbbb);

```

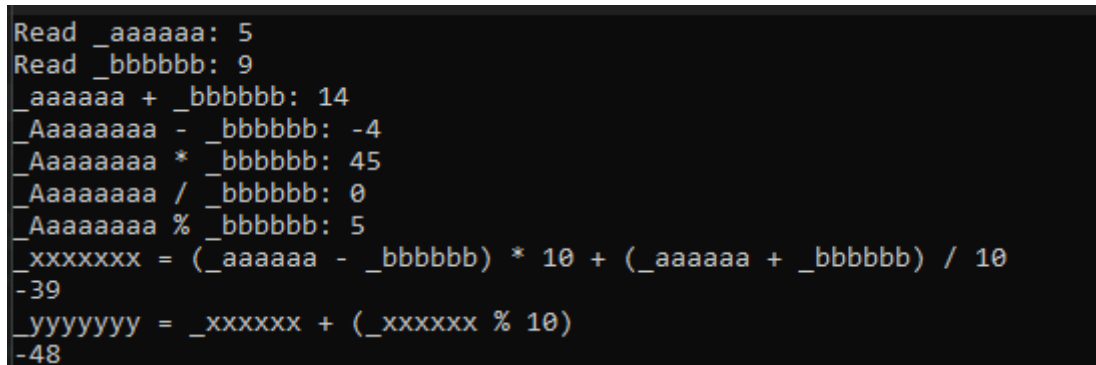
_xxxxxxx<-( _aaaaaa -- _bbbbbb) ** 10 ++ ( _aaaaaa ++ _bbbbbb) Div 10;
_yyyyyyy<-_xxxxxxx ++ ( _xxxxxxx Mod 10);
Write("\n_xxxxxxxx = ( _aaaaaa - _bbbbbb) * 10 + ( _aaaaaa + _bbbbbb) / 10\n");
Write(_xxxxxxx);
Write("\n_yyyyyyyy = _xxxxxxx + ( _xxxxxxx % 10)\n");
Write(_yyyyyyy);
End

```

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

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

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



```

Read _aaaaaa: 5
Read _bbbbbb: 9
_aaaaaa + _bbbbbb: 14
_Aaaaaaaa - _bbbbbb: -4
_Aaaaaaaa * _bbbbbb: 45
_Aaaaaaaa / _bbbbbb: 0
_Aaaaaaaa % _bbbbbb: 5
_xxxxxxxx = ( _aaaaaa - _bbbbbb) * 10 + ( _aaaaaa + _bbbbbb) / 10
-39
_yyyyyyyy = _xxxxxxx + ( _xxxxxxx % 10)
-48

```

Рис. 5.1 Результат виконання коректної програми

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

5.4 Тестова програма №1

Текст програми

\$\$Prog1

Mainprogram

Start

Data Integer16 _aaaaaa,_bbbbbb,_xxxxxx,_yyyyyy;

Write("Read _aaaaaa: ");

Read(_aaaaaa);

Write("Read _bbbbbb: ");

Read(_bbbbbb);

Write("_aaaaaa + _bbbbbb: ");

Write(_aaaaaa ++ _bbbbbb);

Write("\n_Aaaaaaaa - _bbbbbb: ");

Write(_aaaaaa -- _bbbbbb);

Write("\n_Aaaaaaaa * _bbbbbb: ");

Write(_aaaaaa ** _bbbbbb);

Write("\n_Aaaaaaaa / _bbbbbb: ");

Write(_aaaaaa Div _bbbbbb);

Write("\n_Aaaaaaaa % _bbbbbb: ");

Write(_aaaaaa Mod _bbbbbb);

_xxxxxx<-(_aaaaaa -- _bbbbbb) ** 10 ++ (_aaaaaa ++ _bbbbbb) Div 10;

_yyyyyy<-_xxxxxx ++ (_xxxxxx Mod 10);

Write("\n_xxxxxxx = (_aaaaaa - _bbbbbb) * 10 + (_aaaaaa + _bbbbbb) / 10\n");

Write(_xxxxxx);

Write("\n_yyyyyyy = _xxxxxx + (_xxxxxx % 10)\n");

Write(_yyyyyy);

End

Результат виконання

```
Read _aaaaaa: 5
Read _bbbbbb: 9
_aaaaaa + _bbbbbb: 14
_Aaaaaaaa - _bbbbbb: -4
_Aaaaaaaa * _bbbbbb: 45
_Aaaaaaaa / _bbbbbb: 0
_Aaaaaaaa % _bbbbbb: 5
_xxxxxxx = (_aaaaaa - _bbbbbb) * 10 + (_aaaaaa + _bbbbbb) / 10
-39
_yyyyyyy = _xxxxxx + (_xxxxxx % 10)
-48
```

Рис. 5.2 Результат виконання тестової програми №1

5.5 Тестова програма №2

Текст програми

\$\$Prog2

Mainprogram

Start

Data Integer16 _aaaaaa,_bbbbbb,_cccccc;

Write("Read _aaaaaa: ");

Read(_aaaaaa);

Write("Read _bbbbbb: ");

Read(_bbbbbb);

Write("Read _cccccc: ");

Read(_cccccc);

If(_aaaaaa Et _bbbbbb)

Start

 If(_aaaaaa Et _cccccc)

 Start

 Goto _avalue;

 End

 Else

 Start

 Write(_cccccc);

 Goto _outoif;

 _avalue:

 Write(_aaaaaa);

 Goto _outoif;

 End

End

```

    If(_bbbbbb Lt _cccccc)
    Start
        Write(_cccccc);
    End
    Else
    Start
        Write(_bbbbbb);
    End
_outoif:
Write("\n");
If((_aaaaaa = _bbbbbb) & (_aaaaaa = _cccccc) & (_bbbbbb = _cccccc))
Start
    Write(1);
End
Else
Start
    Write(0);
End
Write("\n");
If((_aaaaaa Lt 0) | (_bbbbbb Lt 0) | (_cccccc Lt 0))
Start
    Write(-1);
End
Else
Start
    Write(0);
End
Write("\n");

```

```
If(!_aaaaaa Lt (_bbbbbb ++ _cccccc)))
```

```
Start
```

```
    Write(10);
```

```
End
```

```
Else
```

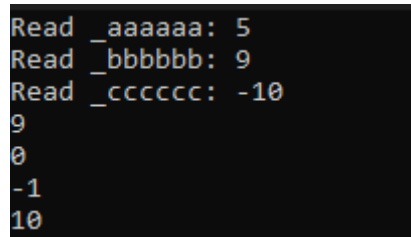
```
Start
```

```
    Write(0);
```

```
End
```

```
End
```

Результат виконання



```
Read _aaaaaa: 5
Read _bbbbbb: 9
Read _cccccc: -10
9
0
-1
10
```

Рис. 5.3 Результат виконання тестової програми №2

5.6 Тестова програма №3

Текст програми

```
$$Prog3
```

```
Mainprogram
```

```
Start
```

```
Data Integer16 _aaaaaa,_aaaaa2,_bbbbbb,_xxxxxx,_cccc1,_cccc2;
```

```
Write("Read _aaaaaa: ");
```

```
Read(_aaaaaa);
```

```
Write("Read _bbbbbb: ");
```

```
Read(_bbbbbb);
```

```
Write("For To do");
```

```

For _aaaaa2<=_aaaaa To _bbbbbb Do
Start
    Write("\n");
    Write(_aaaaa2 ** _aaaaa2);
End
Write("\nFor Downto do");
For _aaaaa2<=_bbbbbb Downto _aaaaa Do
Start
    Write("\n");
    Write(_aaaaa2 ** _aaaaa2);
End

Write("\nWhile _aaaaa * _bbbbbb: ");
_xxxxxx<-0;
_ccccc1<-0;
While(_cccc1 Lt _aaaaa)
Start
    _cccc2<-0;
    While (_cccc2 Lt _bbbbbb)
    Start
        _xxxxxx<=_xxxxxx ++ 1;
        _cccc2<=_cccc2 ++ 1;
    End
    _cccc1<=_cccc1 ++ 1;
End
Write(_xxxxxx);

Write("\nRepeat Until _aaaaa * _bbbbbb: ");

```

```
_xxxxxx<-0;
_ccccc1<-1;
Repeat
  _cccc2<-1;
  Repeat
    _xxxxxx<-_xxxxxx++1;
    _cccc2<-_cccc2++1;
  Until(!(_cccc2 Et _bbbbbb))
  _cccc1<-_cccc1++1;
Until(!(_cccc1 Et _aaaaaa))
Write(_xxxxxx);

End
```

Результат виконання

```
Read _aaaaaa: 5
Read _bbbbbb: 9
For To do
25
36
49
64
81
For Downto do
81
64
49
36
25
While _aaaaaa * _bbbbbb: 45
Repeat Until _aaaaaa * _bbbbbb: 45
```

Рис. 5.4 Результат виконання тестової програми №3

Висновки

В процесі виконання курсового проекту було виконано наступне:

1. Складено формальний опис мови програмування p24, в термінах розширеної нотації Бекуса-Наура, виділено усі термінальні символи та ключові слова.
2. Створено компілятор мови програмування p24, 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 p24. 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 p24 були успішно виявлені і відповідно оброблені.

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

Список використаної літератури

1. Language Processors: Assembler, Compiler and Interpreter

URL: [Language Processors: Assembler, Compiler and Interpreter - GeeksforGeeks](#)

2. Error Handling in Compiler Design

URL: [Error Handling in Compiler Design - GeeksforGeeks](#)

3. Symbol Table in Compiler

URL: [Symbol Table in Compiler - GeeksforGeeks](#)

4. Вікіпедія

URL: [Wikipedia](#)

5. Stack Overflow

URL: [Stack Overflow - Where Developers Learn, Share, & Build Careers](#)

Додатки

Додаток А (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=====

aaaaaa dw 0

bbbbbb dw 0

xxxxxx dw 0

yyyyyy dw 0

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

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

String_0 db "Read _aaaaaa: ", 0

```

String_1    db    "Read _bbbbbb: ", 0
String_2    db    "_aaaaaa + _bbbbbb: ", 0
String_3    db    13, 10, "_Aaaaaaaa - _bbbbbb: ", 0
String_4    db    13, 10, "_Aaaaaaaa * _bbbbbb: ", 0
String_5    db    13, 10, "_Aaaaaaaa / _bbbbbb: ", 0
String_6    db    13, 10, "_Aaaaaaaa % _bbbbbb: ", 0
String_7    db    13, 10, "_xxxxxxx = (_aaaaaa - _bbbbbb) * 10 +
(_aaaaaa + _bbbbbb) / 10", 13, 10, 0
String_8    db    13, 10, "_yyyyyyy = _xxxxxx + (_xxxxxx % 10)", 13,
10, 0

```

```

;===Addition

```

```

Data=====
=====

```

```

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

```

```

CharsReadNum    dd    ?
InputBuf        db    15 dup (?)
OutMessage      db    "%hd", 0
ResMessage      db    20 dup (?)

```

```

.CODE

```

```

start:

```

```

invoke AllocConsole

```

```

invoke GetStdHandle, STD_INPUT_HANDLE

```

```

mov hConsoleInput, eax

```

```

invoke GetStdHandle, STD_OUTPUT_HANDLE

```

```

mov hConsoleOutput, eax

    invoke WriteConsoleA, hConsoleOutput, ADDR String_0, SIZEOF String_0
- 1, 0, 0
    call Input_
    mov _aaaaaa_, ax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_1, SIZEOF String_1
- 1, 0, 0
    call Input_
    mov _bbbbbb_, ax
    invoke WriteConsoleA, hConsoleOutput, ADDR String_2, SIZEOF String_2
- 1, 0, 0
    push _aaaaaa_
    push _bbbbbb_
    call Add_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3
- 1, 0, 0
    push _aaaaaa_
    push _bbbbbb_
    call Sub_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4
- 1, 0, 0
    push _aaaaaa_
    push _bbbbbb_
    call Mul_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5
- 1, 0, 0
    push _aaaaaa_

```

```

push _bbbbbb_
call Div_
call Output_
invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6
- 1, 0, 0
push _aaaaaa_
push _bbbbbb_
call Mod_
call Output_
push _aaaaaa_
push _bbbbbb_
call Sub_
push word ptr 10
call Mul_
push _aaaaaa_
push _bbbbbb_
call Add_
push word ptr 10
call Div_
call Add_
pop _xxxxxx_
push _xxxxxx_
push _xxxxxx_
push word ptr 10
call Mod_
call Add_
pop _yyyyyy_
invoke WriteConsoleA, hConsoleOutput, ADDR String_7, SIZEOF String_7
- 1, 0, 0

```

```

    push _xxxxxx_
    call Output_
    invoke WriteConsoleA, hConsoleOutput, ADDR String_8, SIZEOF String_8
- 1, 0, 0
    push _yyyyyy_
    call Output_
exit_label:
    invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1,
0, 0
    invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0
    invoke ExitProcess, 0

```

```

;===Procedure

```

```

Add=====
=====

```

```

Add_ PROC

```

```

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

```

```

Add_ ENDP

```

```

;=====
=====

```

;===Procedure

Div=====

Div_ PROC

pushf

pop cx

mov ax, [esp + 4]

cmp ax, 0

jne end_check

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

jmp exit_label

end_check:

mov ax, [esp + 6]

cmp ax, 0

jge gr

lo:

mov dx, -1

jmp less_fin

gr:

mov dx, 0

less_fin:

mov ax, [esp + 6]

idiv word ptr [esp + 4]

push cx

popf

mov [esp + 6], ax

```

    pop ecx
    pop ax
    push ecx
    ret

```

Div_ ENDP

```

;=====
=====

```

;===Procedure

```

Input=====
=====

```

Input_ PROC

```

    invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR
CharsReadNum, 0

```

```

    invoke crt_atoi, ADDR InputBuf

```

```

    ret

```

Input_ ENDP

```

;=====
=====

```

;===Procedure

```

Mod=====
=====

```

Mod_ PROC

```

    pushf

```

```

    pop cx

```

```

    mov ax, [esp + 4]

```

```

    cmp ax, 0

```

```

        jne end_check

        invoke WriteConsoleA, hConsoleOutput, ADDR ModErrMsg, SIZEOF
ModErrMsg - 1, 0, 0
        jmp exit_label
end_check:
        mov ax, [esp + 6]
        cmp ax, 0
        jge gr
lo:
        mov dx, -1
        jmp less_fin
gr:
        mov dx, 0
less_fin:
        mov ax, [esp + 6]
        idiv word ptr [esp + 4]
        mov ax, dx
        push cx
        popf

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

;=====
=====

```


====Procedure

Mul=====

Mul_ PROC

```
    mov ax, [esp + 6]
    imul word ptr [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret
```

Mul_ ENDP

=====

====Procedure

Output=====

Output_ PROC value: word

```
    invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value
    invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0
    ret 2
```

Output_ ENDP

=====

;===Procedure

Sub=====

Sub_ PROC

mov ax, [esp + 6]

sub ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Sub_ ENDP

=====

end start

Prog2.asm

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User

Data=====

aaaaaa dw 0

bbbbbb dw 0

cccccc dw 0

String_0 db "Read _aaaaaa: ", 0

String_1 db "Read _bbbbbb: ", 0

String_2 db "Read _cccccc: ", 0

String_3 db 13, 10, 0

String_4 db 13, 10, 0

String_5 db 13, 10, 0

;===Addition

Data=====

hConsoleInput dd ?

hConsoleOutput dd ?

endBuff db 5 dup (?)

msg1310 db 13, 10, 0

CharsReadNum dd ?

InputBuf db 15 dup (?)

OutMessage db "%hd", 0

ResMessage db 20 dup (?)

.CODE

start:

invoke AllocConsole

invoke GetStdHandle, STD_INPUT_HANDLE

mov hConsoleInput, eax

invoke GetStdHandle, STD_OUTPUT_HANDLE

mov hConsoleOutput, eax

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

 call Input_

 mov _aaaaaa_, ax

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

 call Input_

 mov _bbbbbb_, ax

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

 call Input_

 mov _cccccc_, ax

 push _aaaaaa_

 push _bbbbbb_

 call Greate_

 pop ax

 cmp ax, 0

 je endIf2

 push _aaaaaa_

 push _cccccc_

 call Greate_

 pop ax

 cmp ax, 0

```

        je elseLabel1
        jmp _avalue_
        jmp endIf1
elseLabel1:
        push _cccccc_
        call Output_
        jmp _outoif_
_avalue_:
        push _aaaaaa_
        call Output_
        jmp _outoif_
endIf1:
endIf2:
        push _bbbbbb_
        push _cccccc_
        call Less_
        pop ax
        cmp ax, 0
        je elseLabel3
        push _cccccc_
        call Output_
        jmp endIf3
elseLabel3:
        push _bbbbbb_
        call Output_
endIf3:
_outoif_:
        invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3
        - 1, 0, 0

```

```

    push _aaaaaa_
    push _bbbbbb_
    call Equal_
    push _aaaaaa_
    push _cccccc_
    call Equal_
    call And_
    push _bbbbbb_
    push _cccccc_
    call Equal_
    call And_
    pop ax
    cmp ax, 0
    je elseLabel4
    push word ptr 1
    call Output_
    jmp endIf4
elseLabel4:
    push word ptr 0
    call Output_
endIf4:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4
- 1, 0, 0
    push _aaaaaa_
    push word ptr 0
    call Less_
    push _bbbbbb_
    push word ptr 0
    call Less_

```

```

    call Or_
    push _cccccc_
    push word ptr 0
    call Less_
    call Or_
    pop ax
    cmp ax, 0
    je elseLabel5
    push word ptr -1
    call Output_
    jmp endIf5
elseLabel5:
    push word ptr 0
    call Output_
endIf5:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5
- 1, 0, 0
    push _aaaaaa_
    push _bbbbbb_
    push _cccccc_
    call Add_
    call Less_
    call Not_
    pop ax
    cmp ax, 0
    je elseLabel6
    push word ptr 10
    call Output_
    jmp endIf6

```

elseLabel6:

push word ptr 0

call Output_

endIf6:

exit_label:

invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1,
0, 0

invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0

invoke ExitProcess, 0

;===Procedure

Add=====

Add_ PROC

mov ax, [esp + 6]

add ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Add_ ENDP

=====

;===Procedure

And=====

And_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, 0

jnz and_t1

jz and_false

and_t1:

mov ax, [esp + 4]

cmp ax, 0

jnz and_true

and_false:

mov ax, 0

jmp and_fin

and_true:

mov ax, 1

and_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

And_ ENDP

=====

;===Procedure

Equal=====

Equal_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jne equal_false

mov ax, 1

jmp equal_fin

equal_false:

mov ax, 0

equal_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Equal_ ENDP

=====

====Procedure

Greate=====

Greate_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jle greate_false

mov ax, 1

jmp greate_fin

greate_false:

mov ax, 0

greate_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Greate_ ENDP

=====

```
;===Procedure
```

```
Input=====
```

```
Input_ PROC
```

```
    invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR  
CharsReadNum, 0
```

```
    invoke crt_atoi, ADDR InputBuf
```

```
    ret
```

```
Input_ ENDP
```

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

```
;===Procedure
```

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

```
Less_ PROC
```

```
    pushf
```

```
    pop cx
```

```
    mov ax, [esp + 6]
```

```
    cmp ax, [esp + 4]
```

```
    jge less_false
```

```
    mov ax, 1
```

```
    jmp less_fin
```

```
less_false:
```

```
    mov ax, 0
```

```
less_fin:
```

```
    push cx
```

```
    popf
```

```
mov [esp + 6], ax
```

```
pop ecx
```

```
pop ax
```

```
push ecx
```

```
ret
```

```
Less_ ENDP
```

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

```
;===Procedure
```

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

```
Not_ PROC
```

```
pushf
```

```
pop cx
```

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

```
cmp ax, 0
```

```
jnz not_false
```

```
not_t1:
```

```
mov ax, 1
```

```
jmp not_fin
```

```
not_false:
```

```
mov ax, 0
```

```
not_fin:
```

```
push cx
```

```
popf
```

```

        mov [esp + 4], ax
        ret
Not_ ENDP

;=====
=====

;===Procedure
Or=====
=====

Or_ PROC
    pushf
    pop cx

    mov ax, [esp + 6]
    cmp ax, 0
    jnz or_true
    jz or_t1
or_t1:
    mov ax, [esp + 4]
    cmp ax, 0
    jnz or_true
or_false:
    mov ax, 0
    jmp or_fin
or_true:
    mov ax, 1
or_fin:

```

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Or_ ENDP

;
=====

;===Procedure

Output=====

Output_ PROC value: word

invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value

invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0

ret 2

Output_ ENDP

;
=====

end start

Prog3.asm

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

```

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

```

```

.DATA

```

```

;===User

```

```

Data=====
=====

```

```

    _aaaaa2_    dw    0
    _aaaaaa_    dw    0
    _bbbbbb_    dw    0
    _cccc1_     dw    0
    _cccc2_     dw    0
    _xxxxxx_    dw    0

```

```

String_0    db    "Read _aaaaaa: ", 0
String_1    db    "Read _bbbbbb: ", 0
String_2    db    "For To do", 0
String_3    db    13, 10, 0
String_4    db    13, 10, "For Downto do", 0
String_5    db    13, 10, 0
String_6    db    13, 10, "While _aaaaaa * _bbbbbb: ", 0
String_7    db    13, 10, "Repeat Until _aaaaaa * _bbbbbb: ", 0

```


;===Addition

Data=====

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

CharsReadNum dd ?
InputBuf db 15 dup (?)
OutMessage db "%hd", 0
ResMessage db 20 dup (?)

.CODE

start:

invoke AllocConsole

invoke GetStdHandle, STD_INPUT_HANDLE

mov hConsoleInput, eax

invoke GetStdHandle, STD_OUTPUT_HANDLE

mov hConsoleOutput, eax

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

call Input_

mov _aaaaaa_, ax

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

call Input_

mov _bbbbbb_, ax

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

```

    push _aaaaaa_
    pop _aaaaa2_
forPasStart1:
    push _bbbbbb_
    push _aaaaa2_
    call Less_
    call Not_
    pop ax
    cmp ax, 0
    je forPasEnd1
    invoke WriteConsoleA, hConsoleOutput, ADDR String_3, SIZEOF String_3
- 1, 0, 0
    push _aaaaa2_
    push _aaaaa2_
    call Mul_
    call Output_
    push _aaaaa2_
    push word ptr 1
    call Add_
    pop _aaaaa2_
    jmp forPasStart1
forPasEnd1:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_4, SIZEOF String_4
- 1, 0, 0
    push _bbbbbb_
    pop _aaaaa2_
forPasStart2:
    push _aaaaaa_
    push _aaaaa2_

```

```

    call Greate_
    call Not_
    pop ax
    cmp ax, 0
    je forPasEnd2
    invoke WriteConsoleA, hConsoleOutput, ADDR String_5, SIZEOF String_5
- 1, 0, 0
    push _aaaaa2_
    push _aaaaa2_
    call Mul_
    call Output_
    push _aaaaa2_
    push word ptr 1
    call Sub_
    pop _aaaaa2_
    jmp forPasStart2
forPasEnd2:
    invoke WriteConsoleA, hConsoleOutput, ADDR String_6, SIZEOF String_6
- 1, 0, 0
    push word ptr 0
    pop _xxxxxx_
    push word ptr 0
    pop _cccccc1_
whileStart2:
    push _cccccc1_
    push _aaaaaa_
    call Less_
    pop ax
    cmp ax, 0

```

```

        je whileEnd2
        push word ptr 0
        pop _cccc2_
whileStart1:
        push _cccc2_
        push _bbbbbb_
        call Less_
        pop ax
        cmp ax, 0
        je whileEnd1
        push _xxxxxx_
        push word ptr 1
        call Add_
        pop _xxxxxx_
        push _cccc2_
        push word ptr 1
        call Add_
        pop _cccc2_
        jmp whileStart1
whileEnd1:
        push _cccc1_
        push word ptr 1
        call Add_
        pop _cccc1_
        jmp whileStart2
whileEnd2:
        push _xxxxxx_
        call Output_

```

invoke WriteConsoleA, hConsoleOutput, ADDR String_7, SIZEOF String_7
- 1, 0, 0

push word ptr 0

pop _xxxxxx_

push word ptr 1

pop _cccc1_

repeatStart2:

push word ptr 1

pop _cccc2_

repeatStart1:

push _xxxxxx_

push word ptr 1

call Add_

pop _xxxxxx_

push _cccc2_

push word ptr 1

call Add_

pop _cccc2_

push _cccc2_

push _bbbbbb_

call Greate_

call Not_

pop ax

cmp ax, 0

je repeatEnd1

jmp repeatStart1

repeatEnd1:

push _cccc1_

push word ptr 1

```

    call Add_
    pop _cccc1_
    push _cccc1_
    push _aaaaa_
    call Greate_
    call Not_
    pop ax
    cmp ax, 0
    je repeatEnd2
    jmp repeatStart2
repeatEnd2:
    push _xxxxxx_
    call Output_
exit_label:
invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1,
0, 0
invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0
invoke ExitProcess, 0

;===Procedure
Add=====
=====

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

```

```

        push ecx
        ret
Add_ ENDP

;=====
=====

;===Procedure
Greate=====
=====

Greate_ PROC

        pushf
        pop cx

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

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

```

```

        ret

Greate_ ENDP

;=====
=====

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

Input_ PROC

        invoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR
CharsReadNum, 0

        invoke crt_atoi, ADDR InputBuf

        ret

Input_ ENDP

;=====
=====

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

Less_ PROC

        pushf

        pop cx

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

        mov ax, 1

        jmp less_fin

```


less_false:

mov ax, 0

less_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Less_ ENDP

;
=====

;
====Procedure

Mul=====

Mul_ PROC

mov ax, [esp + 6]

imul word ptr [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Mul_ ENDP

;
=====

====Procedure

Not=====

Not_ PROC

pushf

pop cx

mov ax, [esp + 4]

cmp ax, 0

jnz not_false

not_t1:

mov ax, 1

jmp not_fin

not_false:

mov ax, 0

not_fin:

push cx

popf

mov [esp + 4], ax

ret

Not_ ENDP

=====

```

;===Procedure
Output=====
=====

Output_ PROC value: word

    invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value
    invoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0
    ret 2

Output_ ENDP

;=====
=====

;===Procedure
Sub=====
=====

Sub_ PROC

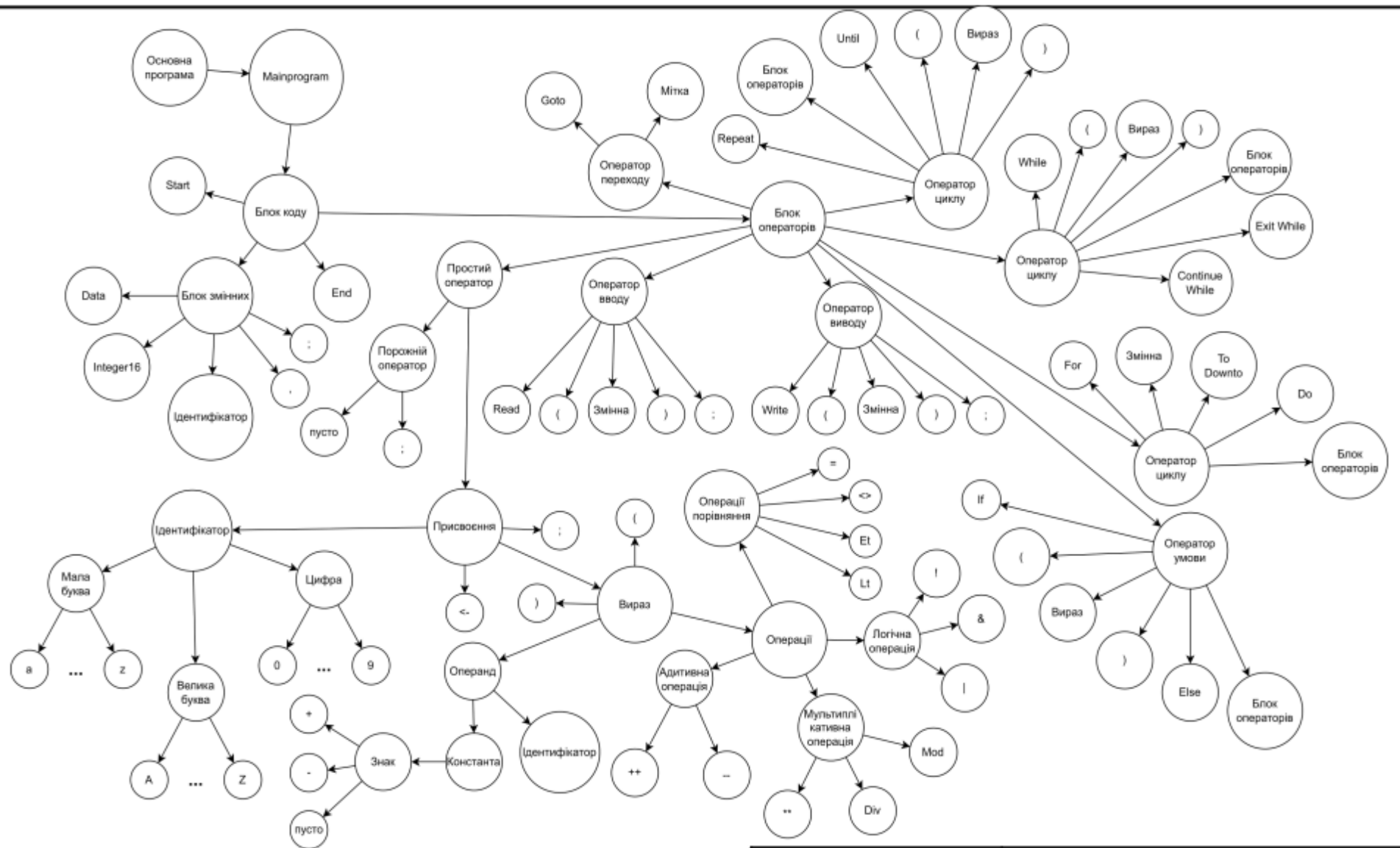
    mov ax, [esp + 6]
    sub ax, [esp + 4]
    mov [esp + 6], ax
    pop ecx
    pop ax
    push ecx
    ret

Sub_ ENDP

;=====
=====

end start

```



Міністерство освіти і науки України					Курсовий проєкт				
					Розробка транслятора з вхідної мови програмування				
					Дерево граматичного розбору				
Зм.	Арк.	№ докум.	Підпис	Дата	Літера		Маса	Масштаб	
Розробив		Петренко В.А.							
Керівник		Козак Н.Б.							
Консулянт					Аркуш		Аркуші 1		
Н.контр.					НУ «ЛП», Каф. ЕОМ, гр. КІ-307				
Зав.каф.									
Рецензент									

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;
}

```

BackusRule.h

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

class Controller;

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

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

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

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

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

private:
    friend class Controller;

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

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

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

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

    static bool HasFlag(RuleCountPolicy policy, RuleCountPolicy flag);

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

BackusRuleBase.h

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

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

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


BackusRuleItem.h

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

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

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

        return m_rules;
    };

    RuleCountPolicy policy() const { return m_policy; };

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

BackusRuleStorage.h

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

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

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

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

            rules.push_back(it->second);
        }

        return rules;
    };

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



```

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);
    }
}

```

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;
};
```


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

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

            ++it;
        }

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

        return postfixForm;
    }

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



```

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

    return res;
}

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

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

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

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

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

    return leftPriority > rightPriority;
}

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

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

    return res;
}

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

    return res;
}

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

```

IGeneratorItem.h

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

__interface IGeneratorItem : public IItem
{
public:
    virtual 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 = 0;
};
```

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("Tokenizer: Token with type " + token->type() + " already registered");
}

void Tokenizer::recognizeToken(std::string& token, int curLine)
{
    if(m_priorityTokens.empty())
        throw std::runtime_error("Tokenizer: 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 Tokenizer::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 Tokenizer::IsNewLine(const char& ch)
{
    return ch == '\n';
}

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

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

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

```

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;
};

};
```

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();
}
```

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> errorMsgs;

    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 errMsg = rule->customData("error"); !errMsg.empty())
            {
                semErr++;
                err = "Semantic error: " + errMsg;
            }
            else
            {
                semErr++;
                err = std::format("Semantic error: Undefined token: {}", rule->value());
            }
            errorMsgs.emplace(rule->line(), err);
        }
        else if (rule->type() == token::Unknown::Type())
        {
            lexErr++;
            res = false;
            errorMsgs.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;
            errorMsgs.emplace(it->first, msg);
        }
    }

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

    return res;
}
```


IToken.hpp

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

__interface IToken : public IItem
{
public:
    virtual std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const = 0;

    virtual void setLine(int line) = 0;

    virtual std::shared_ptr<IToken> clone() const = 0;
};

template <class T>
std::string GetTypeName()
{
    static std::string type;

    if (type.empty())
    {
        std::string name = typeid(T).name();
        type = { name.begin() + 6, name.end() };
    }

    return type;
}
```

TokenBase.hpp

```
#pragma once
#include "stdafx.h"
#include "Core/Tokens/IToken.hpp"

template <class T>
class TokenBase : public IToken
{
public:
    virtual ~TokenBase() = default;

    std::shared_ptr<IToken> tryCreateToken(std::string& lexem) const override
    {
        std::shared_ptr<IToken> token = nullptr;

        if (lexem.find(lexeme()) == 0)
        {
            lexem.erase(0, lexeme().size());
            token = clone();
            token->setValue(lexeme());
        }

        return token;
    };

    static std::string Type() { return GetTypeName<T>(); }

    void setLine(int line) final { m_line = line; };

    std::shared_ptr<IToken> clone() const final { return std::make_shared<T>(); };

protected:
    std::string lexeme_imp() const { return m_lexeme; };
    std::string type_imp() const { return Type(); }
    void setLexeme(const std::string& lexeme) { m_lexeme = lexeme; };
    std::string value_imp() const { return m_value; }
    int line_imp() const { return m_line; };
    void setValue_imp(const std::string& value) { m_value = value; }

private:
    std::string m_lexeme;
    std::string m_value;
    int m_line = -1;
};
```

Item.h

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

__interface IItem
{
    virtual std::string lexeme() const = 0;
    virtual std::string type() const = 0;
    virtual std::string value() const = 0;
    virtual void      setValue(const std::string & value) = 0;
    virtual int       line() const = 0;

    virtual std::string customData(const std::string& id = "default") const = 0;
    virtual void      setCustomData(const std::string& data, const std::string& id = "default") = 0;
};

#define BASE_ITEM public: \
    std::string lexeme() const final { return lexeme_imp(); } \
    std::string type()  const final { return type_imp(); } \
    void      setValue(const std::string& value) final { setValue_imp(value); } \
    std::string value() const final { return value_imp(); } \
    int  line()      const final { return line_imp(); } \
    std::string customData(const std::string& id = "default")  const final { return customData_imp(id); } \
    void      setCustomData(const std::string& data, const std::string& id = "default") final { setCustomData_imp(data, id); }
```

SimpleTokens.h

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

SimpleToken(Comma, ",");
SimpleToken(Colon, ":");
SimpleToken(Semicolon, ";");
SimpleToken(LBraket, "(");
SimpleToken(RBraket, ")");
SimpleToken(Plus, "+");
SimpleToken(Minus, "-");

SimpleToken(Start, "Start");
class End : public TokenBase<End>, public BackusRuleBase<End>, public GeneratorItemBase<End>
{
    BASE_ITEM

public:
    End() { setLexeme("End"); };
    ~End() final = 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().empty())
            out << customData() << std::endl;
    };
};
```

Symbols.h

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

enum class Symbols
{
    Underscore,

    Comma,
    Colon,
    Semicolon,

    LBraket,
    RBraket,

    Plus,
    Minus
};

bool operator==(const Symbols& lhs, const std::string& rhs);
bool operator==(const std::string& lhs, const Symbols& rhs);

#define SimpleToken(name, lexeme) \
class name : public TokenBase<name>, public BackusRuleBase<name>, public GeneratorItemBase<name> \
{ \
    BASE_ITEM \
\
public: \
    name() { setLexeme(lexeme); }; \
    ~name() final = default; \
};
```

Assignment.h

```
#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;
    };
};
```

AssignmentRule.cpp

```
#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;
}
```

AssignmentRule.h

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

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


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("++"); };
    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("--"); };
    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";
    }
};
```

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("="); };
    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("Et"); };
    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("Lt"); };
    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";
    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("<"); };
    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";
    };
};
```

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("&"); };
    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("!"); };
    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";
            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(""); };
    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";
    }
};
```

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("***"); };
    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;
}
```

EquationRule.h

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

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

Number.h

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

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

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

    std::shared_ptr<IToken> tryCreateToken(std::string& lexeme) const override
    {
        for (char c : lexeme) {
            if (!isdigit(c)) {
                return nullptr;
            }
        }

        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
    {
        out << "\tpush " << details.args().numberTypeExtended << " ptr " << value() << std::endl;
    };
};
```

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;

        if (lexeme.size() > (m_mask.size() + m_prefix.size()))
            return nullptr;

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

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

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

        return token;
    };

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

private:
    const std::string m_prefix = " ";
    const std::string m_mask = "xxxxxx";
};
```


IdentRule.cpp

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

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

SimpleToken(ProgramName, "");

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

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

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

    auto context = controller->context();

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

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

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

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

    return identRule;
}
```

IdentRule.h

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

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

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

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("Read"); };
    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", "\t15 dup (?)");
            details.registerRawData("CharsReadNum", "\t1?");
            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;
}
```

ReadRule.h

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

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

String.cpp

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

```
size_t String::index = 0;
```

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;
}
```

StringRule.h

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

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

VarsBlokRule.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;
}
```

VarsBlokRule.h

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

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

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("Integer16"); };
    virtual ~VarType() = default;
};
```

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("Write"); };
    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", "\t\b\t\t" + details.args().numberStrType + "\", 0");
            details.registerRawData("ResMessage", "\t\b\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 wprintf, 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;
}
```

WriteRule.h

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

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


Controller.cpp

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

#include "SimpleTokens.h"

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

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

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

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

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

    auto tokenType = token->type();

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    return rule;
}

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

    return m_topRule;
}

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

```

```

using enum ItemType;

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

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

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

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

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

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

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

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

return topRule;
}

```

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;
};
```

sp_kursova.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 = ".p24";

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

        std::cout << longLine << std::endl;
        std::cout << "TRANSLATOR (" << extention << ">->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() != extention)
        {
            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 + extention, 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;  
}
```

Symbols.cpp

```
#include "stdafx.h"
#include "Symbols.h"
#include "Utils/magic_enum.hpp"

bool operator==(const Symbols& lhs, const std::string& rhs) {
    return magic_enum::enum_name(lhs) == rhs;
}

bool operator==(const std::string& lhs, const Symbols& rhs) {
    return rhs == lhs;
}
```

Do.h

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

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

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

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

DownTo.h

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

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

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

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

    return forToOrDownToDoRule;
}
```

ForRule.h

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

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

To.h

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

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

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

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

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

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

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

Goto.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.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;
}

```

GotoRule.h

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

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


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;
    };
};
```

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.cpp

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

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

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

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

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

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

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

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

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

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

    return ifStatement;
}
```

IfRule.h

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

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

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.cpp

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

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

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

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

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

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

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

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

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

    return repeatUntilRule;
}
```

RepeatUntilRule.h

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

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


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.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.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("{}\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;
}
```

WhileRule.h

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

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

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;
    };
};
```

LComment.h

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

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

public:
    LComment() { setLexeme("$"); };
    virtual ~LComment() = default;
};
```

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;
};
```

Unknown.h

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

namespace token
{
    class Unknown : public TokenBase<Unknown>, public BackusRuleBase<Unknown>, public GeneratorItemBase<Unknown>
    {
        BASE_ITEM

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

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


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("Mainprogram"); };
    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;
};
```

Common.h

```
#include "Tokens/Common/LComment.h"
#include "Tokens/Common/Comment.h"
#include "Tokens/Common/Unknown.h"
```

```
// | V | ( ) | _ | _ | / _ | _ |
// | \ | / _ _ _ _ _ | _ | _ _ _ _ _ | | _ | _ | _ |
// | M | / _ _ _ _ _ | _ | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
// | | | | | | | | | | | | | | | | | | | | | | | | | | | |
// | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ |
//
// _ | https://github.com/Neargye/magic_enum
// _ | version 0.9.5
//
// Licensed under the MIT License <http://opensource.org/licenses/MIT>.
// SPDX-License-Identifier: MIT
// Copyright (c) 2019 - 2023 Daniil Goncharov <neargye@gmail.com>.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef NEARGYE_MAGIC_ENUM_HPP
#define NEARGYE_MAGIC_ENUM_HPP

#define MAGIC_ENUM_VERSION_MAJOR 0
#define MAGIC_ENUM_VERSION_MINOR 9
#define MAGIC_ENUM_VERSION_PATCH 5

#include <array>
#include <cstdint>
#include <stdint>
#include <functional>
#include <limits>
#include <type_traits>
#include <utility>

#if defined(MAGIC_ENUM_CONFIG_FILE)
# include MAGIC_ENUM_CONFIG_FILE
#endif

#if !defined(MAGIC_ENUM_USING_ALIAS_OPTIONAL)
# include <optional>
#endif

#if !defined(MAGIC_ENUM_USING_ALIAS_STRING)
# include <string>
#endif

#if !defined(MAGIC_ENUM_USING_ALIAS_STRING_VIEW)
# include <string_view>
#endif

#if defined(MAGIC_ENUM_NO_ASSERT)
# define MAGIC_ENUM_ASSERT(...) static_cast<void>(0)
#else
# define MAGIC_ENUM_ASSERT(...) assert((__VA_ARGS__))
#endif

#if defined(__clang__)
# pragma clang diagnostic push
# pragma clang diagnostic ignored "-Wunknown-warning-option"
# pragma clang diagnostic ignored "-Wenum-conversion"
# pragma clang diagnostic ignored "-Wuseless-cast" // suppresses 'static_cast<char_type>('\0') for char_type = char (common on Linux).
# elif defined(__GNUC__)
# pragma GCC diagnostic push
# pragma GCC diagnostic ignored "-Wmaybe-uninitialized" // May be used uninitialized 'return {}';
# pragma GCC diagnostic ignored "-Wuseless-cast" // suppresses 'static_cast<char_type>('\0') for char_type = char (common on Linux).
# elif defined(_MSC_VER)
# pragma warning(push)
# pragma warning(disable : 26495) // Variable 'static_str<N>::chars_' is uninitialized.
# pragma warning(disable : 28020) // Arithmetic overflow: Using operator '-' on a 4 byte value and then casting the result to a 8 byte value.
# pragma warning(disable : 26451) // The expression '0<= _Param_(1)&& _Param_(1)<=1-1' is not true at this call.
# pragma warning(disable : 4514) // Unreferenced inline function has been removed.
#endif

// Checks magic_enum compiler compatibility.
#if defined(__clang__) && __clang_major__ >= 5 || defined(__GNUC__) && __GNUC__ >= 9 || defined(_MSC_VER) && _MSC_VER >= 1910 || defined(__RESHARPER__)
# undef MAGIC_ENUM_SUPPORTED
# define MAGIC_ENUM_SUPPORTED 1
#endif

// Checks magic_enum compiler aliases compatibility.
#if defined(__clang__) && __clang_major__ >= 5 || defined(__GNUC__) && __GNUC__ >= 9 || defined(_MSC_VER) && _MSC_VER >= 1920
# undef MAGIC_ENUM_SUPPORTED_ALIASES
# define MAGIC_ENUM_SUPPORTED_ALIASES 1
#endif

// Enum value must be greater or equals than MAGIC_ENUM_RANGE_MIN. By default MAGIC_ENUM_RANGE_MIN = -128.
// If need another min range for all enum types by default, redefine the macro MAGIC_ENUM_RANGE_MIN.
#if !defined(MAGIC_ENUM_RANGE_MIN)
```

```
# define MAGIC_ENUM_RANGE_MIN -128
#endif

// Enum value must be less or equals than MAGIC_ENUM_RANGE_MAX. By default MAGIC_ENUM_RANGE_MAX = 128.
// If need another max range for all enum types by default, redefine the macro MAGIC_ENUM_RANGE_MAX.
#if !defined(MAGIC_ENUM_RANGE_MAX)
# define MAGIC_ENUM_RANGE_MAX 127
#endif

// Improve ReSharper C++ intellisense performance with builtins, avoiding unnecessary template instantiations.
#if defined(__RESHARPER__)
# undef MAGIC_ENUM_GET_ENUM_NAME_BUILTIN
# undef MAGIC_ENUM_GET_TYPE_NAME_BUILTIN
# if __RESHARPER__ >= 20230100
# define MAGIC_ENUM_GET_ENUM_NAME_BUILTIN(V) __rscpp_enumerator_name(V)
# define MAGIC_ENUM_GET_TYPE_NAME_BUILTIN(T) __rscpp_type_name<T>()
# else
# define MAGIC_ENUM_GET_ENUM_NAME_BUILTIN(V) nullptr
# define MAGIC_ENUM_GET_TYPE_NAME_BUILTIN(T) nullptr
# endif
#endif

namespace magic_enum {

// If need another optional type, define the macro MAGIC_ENUM_USING_ALIAS_OPTIONAL.
#if defined(MAGIC_ENUM_USING_ALIAS_OPTIONAL)
MAGIC_ENUM_USING_ALIAS_OPTIONAL
#else
using std::optional;
#endif

// If need another string_view type, define the macro MAGIC_ENUM_USING_ALIAS_STRING_VIEW.
#if defined(MAGIC_ENUM_USING_ALIAS_STRING_VIEW)
MAGIC_ENUM_USING_ALIAS_STRING_VIEW
#else
using std::string_view;
#endif

// If need another string type, define the macro MAGIC_ENUM_USING_ALIAS_STRING.
#if defined(MAGIC_ENUM_USING_ALIAS_STRING)
MAGIC_ENUM_USING_ALIAS_STRING
#else
using std::string;
#endif

using char_type = string_view::value_type;
static_assert((std::is_same_v<string_view::value_type, string::value_type>), "magic_enum::customize requires same string_view::value_type and string::value_type");
static_assert({} {
    if constexpr (std::is_same_v<char_type, wchar_t>) {
        constexpr const char c[] = "abcdefghijklmnopqrstuvwxyz_ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        constexpr const wchar_t wc[] = L"abcdefghijklmnopqrstuvwxyz_ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        static_assert((std::size(c) == std::size(wc)), "magic_enum::customize identifier characters are multichars in wchar_t.");

        for (std::size_t i = 0; i < std::size(c); ++i) {
            if (c[i] != wc[i]) {
                return false;
            }
        }
        return true;
    }
    return false;
} (), "magic_enum::customize wchar_t is not compatible with ASCII.");

namespace customize {

// Enum value must be in range [MAGIC_ENUM_RANGE_MIN, MAGIC_ENUM_RANGE_MAX]. By default MAGIC_ENUM_RANGE_MIN = -128, MAGIC_ENUM_RANGE_MAX = 128.
// If need another range for all enum types by default, redefine the macro MAGIC_ENUM_RANGE_MIN and MAGIC_ENUM_RANGE_MAX.
// If need another range for specific enum type, add specialization enum_range for necessary enum type.
template <typename E>
struct enum_range {
    static_assert((std::is_enum_v<E>), "magic_enum::customize::enum_range requires enum type.");
    static constexpr int min = MAGIC_ENUM_RANGE_MIN;
    static constexpr int max = MAGIC_ENUM_RANGE_MAX;
    static_assert(max > min, "magic_enum::customize::enum_range requires max > min.");
};

static_assert(MAGIC_ENUM_RANGE_MAX > MAGIC_ENUM_RANGE_MIN, "MAGIC_ENUM_RANGE_MAX must be greater than MAGIC_ENUM_RANGE_MIN.");
static_assert((MAGIC_ENUM_RANGE_MAX - MAGIC_ENUM_RANGE_MIN) < (std::numeric_limits<std::uint16_t>::max)(), "MAGIC_ENUM_RANGE must be less than UINT16_MAX.");

namespace detail {

enum class customize_tag {
    default_tag,
    invalid_tag,
    custom_tag
};

} // namespace magic_enum::customize::detail

class customize_t : public std::pair<detail::customize_tag, string_view> {
public:
    constexpr customize_t(string_view srt) : std::pair<detail::customize_tag, string_view>{detail::customize_tag::custom_tag, srt} {}
    constexpr customize_t(const char_type* srt) : customize_t{string_view{srt}} {}
    constexpr customize_t(detail::customize_tag tag) : std::pair<detail::customize_tag, string_view>{tag, string_view{}} {
        MAGIC_ENUM_ASSERT(tag != detail::customize_tag::custom_tag);
    }
};

// Default customize.
inline constexpr auto default_tag = customize_t{detail::customize_tag::default_tag};
// Invalid customize.
inline constexpr auto invalid_tag = customize_t{detail::customize_tag::invalid_tag};
```

```

// If need custom names for enum, add specialization enum_name for necessary enum type.
template <typename E>
constexpr customize_t enum_name(E) noexcept {
    return default_tag;
}

// If need custom type name for enum, add specialization enum_type_name for necessary enum type.
template <typename E>
constexpr customize_t enum_type_name() noexcept {
    return default_tag;
}

} // namespace magic_enum::customize

namespace detail {

template <typename T>
struct supported
#ifdef (MAGIC_ENUM_SUPPORTED) && MAGIC_ENUM_SUPPORTED || defined(MAGIC_ENUM_NO_CHECK_SUPPORT)
    : std::true_type {};
#else
    : std::false_type {};
#endif

template <auto V, typename E = std::decay_t<decltype(V)>, std::enable_if_t<std::is_enum_v<E>, int> = 0>
using enum_constant = std::integral_constant<E, V>;

template <typename... T>
inline constexpr bool always_false_v = false;

template <typename T, typename = void>
struct has_is_flags : std::false_type {};

template <typename T>
struct has_is_flags<T, std::void_t<decltype(customize::enum_range<T>::is_flags)>> : std::bool_constant<std::is_same_v<bool, std::decay_t<decltype(customize::enum_range<T>::is_flags)>>>> {};

template <typename T, typename = void>
struct range_min : std::integral_constant<int, MAGIC_ENUM_RANGE_MIN> {};

template <typename T>
struct range_min<T, std::void_t<decltype(customize::enum_range<T>::min)>> : std::integral_constant<decltype(customize::enum_range<T>::min), customize::enum_range<T>::min> {};

template <typename T, typename = void>
struct range_max : std::integral_constant<int, MAGIC_ENUM_RANGE_MAX> {};

template <typename T>
struct range_max<T, std::void_t<decltype(customize::enum_range<T>::max)>> : std::integral_constant<decltype(customize::enum_range<T>::max), customize::enum_range<T>::max> {};

struct str_view {
    const char* str_ = nullptr;
    std::size_t size_ = 0;
};

template <std::uint16_t N>
class static_str {
public:
    constexpr explicit static_str(str_view str) noexcept : static_str{str.str_, std::make_integer_sequence<std::uint16_t, N>{}} {
        MAGIC_ENUM_ASSERT(str.size_ == N);
    }

    constexpr explicit static_str(string_view str) noexcept : static_str{str.data(), std::make_integer_sequence<std::uint16_t, N>{}} {
        MAGIC_ENUM_ASSERT(str.size() == N);
    }

    constexpr const char_type* data() const noexcept { return chars_; }

    constexpr std::uint16_t size() const noexcept { return N; }

    constexpr operator string_view() const noexcept { return {data(), size()}; }

private:
    template <std::uint16_t... I>
    constexpr static_str(const char* str, std::integer_sequence<std::uint16_t, I...>) noexcept : chars_{static_cast<char_type>(str[I]),..., static_cast<char_type>('\0')} {}

    template <std::uint16_t... I>
    constexpr static_str(string_view str, std::integer_sequence<std::uint16_t, I...>) noexcept : chars_{str[I],..., static_cast<char_type>('\0')} {}

    char_type chars_[static_cast<std::size_t>(N) + 1];
};

template <>
class static_str<0> {
public:
    constexpr explicit static_str() = default;

    constexpr explicit static_str(str_view) noexcept {}

    constexpr explicit static_str(string_view) noexcept {}

    constexpr const char_type* data() const noexcept { return nullptr; }

    constexpr std::uint16_t size() const noexcept { return 0; }

    constexpr operator string_view() const noexcept { return {}; }
};

template <typename Op = std::equal_to<>>
class case_insensitive {
    static constexpr char_type to_lower(char_type c) noexcept {
        return (c >= static_cast<char_type>('A') && c <= static_cast<char_type>('Z')) ? static_cast<char_type>(c + (static_cast<char_type>('a') - static_cast<char_type>('A')) : c;
    }
}

```

```

public:
    template <typename L, typename R>
    constexpr auto operator()(L lhs, R rhs) const noexcept -> std::enable_if_t<std::is_same_v<std::decay_t<L>, char_type> && std::is_same_v<std::decay_t<R>, char_type>, bool> {
        return Op{ }(to_lower(lhs), to_lower(rhs));
    }
};

constexpr std::size_t find(string_view str, char_type c) noexcept {
    #if defined(__clang__) && __clang_major__ < 9 && defined(__GLIBCXX__) || defined(_MSC_VER) && _MSC_VER < 1920 && !defined(__clang__)
    // https://stackoverflow.com/questions/56484834/constexpr-stdstring-viewfind-last-of-doesnt-work-on-clang-8-with-libstdc
    // https://developercommunity.visualstudio.com/content/problem/360432/vs20178-regression-c-failed-in-test.html
    constexpr bool workaround = true;
    #else
    constexpr bool workaround = false;
    #endif

    if constexpr (workaround) {
        for (std::size_t i = 0; i < str.size(); ++i) {
            if (str[i] == c) {
                return i;
            }
        }

        return string_view::npos;
    } else {
        return str.find(c);
    }
}

template <typename BinaryPredicate>
constexpr bool is_default_predicate() noexcept {
    return std::is_same_v<std::decay_t<BinaryPredicate>, std::equal_to<string_view::value_type>> ||
        std::is_same_v<std::decay_t<BinaryPredicate>, std::equal_to<>>;
}

template <typename BinaryPredicate>
constexpr bool is_nothrow_invocable() {
    return is_default_predicate<BinaryPredicate>() ||
        std::is_nothrow_invocable_r_v<bool, BinaryPredicate, char_type, char_type>;
}

template <typename BinaryPredicate>
constexpr bool cmp_equal(string_view lhs, string_view rhs, [[maybe_unused]] BinaryPredicate&& p) noexcept(is_nothrow_invocable<BinaryPredicate>()) {
    #if defined(_MSC_VER) && _MSC_VER < 1920 && !defined(__clang__)
    // https://developercommunity.visualstudio.com/content/problem/360432/vs20178-regression-c-failed-in-test.html
    // https://developercommunity.visualstudio.com/content/problem/232218/c-constexpr-string-view.html
    constexpr bool workaround = true;
    #else
    constexpr bool workaround = false;
    #endif

    if constexpr (!is_default_predicate<BinaryPredicate>() || workaround) {
        if (lhs.size() != rhs.size()) {
            return false;
        }

        const auto size = lhs.size();
        for (std::size_t i = 0; i < size; ++i) {
            if (!p(lhs[i], rhs[i])) {
                return false;
            }
        }

        return true;
    } else {
        return lhs == rhs;
    }
}

template <typename L, typename R>
constexpr bool cmp_less(L lhs, R rhs) noexcept {
    static_assert(std::is_integral_v<L> && std::is_integral_v<R>, "magic_enum::detail::cmp_less requires integral type.");

    if constexpr (std::is_signed_v<L> == std::is_signed_v<R>) {
        // If same signedness (both signed or both unsigned).
        return lhs < rhs;
    } else if constexpr (std::is_same_v<L, bool>) { // bool special case
        return static_cast<R>(lhs) < rhs;
    } else if constexpr (std::is_same_v<R, bool>) { // bool special case
        return lhs < static_cast<L>(rhs);
    } else if constexpr (std::is_signed_v<R>) {
        // If 'right' is negative, then result is 'false', otherwise cast & compare.
        return rhs > 0 && lhs < static_cast<std::make_unsigned_t<R>>(rhs);
    } else {
        // If 'left' is negative, then result is 'true', otherwise cast & compare.
        return lhs < 0 || static_cast<std::make_unsigned_t<L>>(lhs) < rhs;
    }
}

template <typename I>
constexpr I log2(I value) noexcept {
    static_assert(std::is_integral_v<I>, "magic_enum::detail::log2 requires integral type.");

    if constexpr (std::is_same_v<I, bool>) { // bool special case
        return MAGIC_ENUM_ASSERT(false), value;
    } else {
        auto ret = I{0};
        for (; value > I{1}; value >>= I{1}, ++ret) {}

        return ret;
    }
}

```

```
#if defined(__cpp_lib_array_constexpr) && __cpp_lib_array_constexpr >= 201603L
# define MAGIC_ENUM_ARRAY_CONSTEXPR 1
#else
template <typename T, std::size_t N, std::size_t... I>
constexpr std::array<std::remove_cv_t<T>, N> to_array(T (&a)[N], std::index_sequence<I...>) noexcept {
    return {{a[I]...}};
}
#endif

template <typename T>
inline constexpr bool is_enum_v = std::is_enum_v<T> && std::is_same_v<T, std::decay_t<T>>;

template <typename E>
constexpr auto n() noexcept {
    static_assert(is_enum_v<E>, "magic_enum::detail::n requires enum type.");

    if constexpr (supported<E>::value) {
    #if defined(MAGIC_ENUM_GET_TYPE_NAME_BUILTIN)
        constexpr auto name_ptr = MAGIC_ENUM_GET_TYPE_NAME_BUILTIN(E);
        constexpr auto name = name_ptr ? str_view{name_ptr, std::char_traits<char>::length(name_ptr)} : str_view{};
    #elif defined(__clang__)
        str_view name;
        if constexpr (sizeof(__PRETTY_FUNCTION__) == sizeof(__FUNCTION__)) {
            static_assert(always_false_v<E>, "magic_enum::detail::n requires __PRETTY_FUNCTION__.");
            return str_view{};
        } else {
            name.size_ = sizeof(__PRETTY_FUNCTION__) - 36;
            name.str_ = __PRETTY_FUNCTION__ + 34;
        }
    #elif defined(__GNUC__)
        auto name = str_view{__PRETTY_FUNCTION__, sizeof(__PRETTY_FUNCTION__) - 1};
        if constexpr (sizeof(__PRETTY_FUNCTION__) == sizeof(__FUNCTION__)) {
            static_assert(always_false_v<E>, "magic_enum::detail::n requires __PRETTY_FUNCTION__.");
            return str_view{};
        } else if (name.str_[name.size_ - 1] == ']') {
            name.size_ -= 50;
            name.str_ += 49;
        } else {
            name.size_ -= 40;
            name.str_ += 37;
        }
    #elif defined(_MSC_VER)
        // CL/C++ workaround (see https://github.com/Neargye/magic_enum/issues/284).
        str_view name;
        name.str_ = __FUNCSIG__;
        name.str_ += 40;
        name.size_ += sizeof(__FUNCSIG__) - 57;
    #else
        auto name = str_view{};
    #endif
}

std::size_t p = 0;
for (std::size_t i = name.size_; i > 0; --i) {
    if (name.str_[i] == ':') {
        p = i + 1;
        break;
    }
}
if (p > 0) {
    name.size_ -= p;
    name.str_ += p;
}
return name;
} else {
    return str_view{}; // Unsupported compiler or Invalid customize.
}
}

template <typename E>
constexpr auto type_name() noexcept {
    [[maybe_unused]] constexpr auto custom = customize::enum_type_name<E>();
    static_assert(std::is_same_v<std::decay_t<decltype(custom)>, customize::customize_t>, "magic_enum::customize requires customize_t type.");
    if constexpr (custom.first == customize::detail::customize_tag::custom_tag) {
        constexpr auto name = custom.second;
        static_assert(!name.empty(), "magic_enum::customize requires not empty string.");
        return static_str<name.size()>{name};
    } else if constexpr (custom.first == customize::detail::customize_tag::invalid_tag) {
        return static_str<0>{};
    } else if constexpr (custom.first == customize::detail::customize_tag::default_tag) {
        constexpr auto name = n<E>();
        return static_str<name.size>_{name};
    } else {
        static_assert(always_false_v<E>, "magic_enum::customize invalid.");
    }
}

template <typename E>
inline constexpr auto type_name_v = type_name<E>();

template <auto V>
constexpr auto n() noexcept {
    static_assert(is_enum_v<decltype(V)>, "magic_enum::detail::n requires enum type.");

    if constexpr (supported<decltype(V)>::value) {
    #if defined(MAGIC_ENUM_GET_ENUM_NAME_BUILTIN)
        constexpr auto name_ptr = MAGIC_ENUM_GET_ENUM_NAME_BUILTIN(V);
        auto name = name_ptr ? str_view{name_ptr, std::char_traits<char>::length(name_ptr)} : str_view{};
    #elif defined(__clang__)
        str_view name;
        if constexpr (sizeof(__PRETTY_FUNCTION__) == sizeof(__FUNCTION__)) {
            static_assert(always_false_v<decltype(V)>, "magic_enum::detail::n requires __PRETTY_FUNCTION__.");
            return str_view{};
        }
```



```

    } else {
        name.size_ = sizeof(__PRETTY_FUNCTION__) - 36;
        name.str_ = __PRETTY_FUNCTION__ + 34;
    }
    if (name.size_ > 22 && name.str_[0] == ' ' && name.str_[1] == 'a' && name.str_[10] == ' ' && name.str_[22] == ':') {
        name.size_ -= 23;
        name.str_ += 23;
    }
    if (name.str_[0] == ' ' || name.str_[0] == '-' || (name.str_[0] >= '0' && name.str_[0] <= '9')) {
        name = str_view{};
    }
#endif defined(__GNUC__)
auto name = str_view{__PRETTY_FUNCTION__, sizeof(__PRETTY_FUNCTION__) - 1};
if constexpr (sizeof(__PRETTY_FUNCTION__) == sizeof(__FUNCTION__)) {
    static_assert(always_false_v<decltype(V)>, "magic_enum::detail::n requires __PRETTY_FUNCTION__.");
    return str_view{};
} else if (name.str_[name.size_ - 1] == ']') {
    name.size_ -= 55;
    name.str_ += 54;
} else {
    name.size_ -= 40;
    name.str_ += 37;
}
if (name.str_[0] == '(') {
    name = str_view{};
}
#endif defined(_MSC_VER)
str_view name;
if ((__FUNCSIG__[5] == '.' && __FUNCSIG__[35] != '(') || (__FUNCSIG__[5] == 'c' && __FUNCSIG__[41] != '(')) {
    // C/C++ workaround (see https://github.com/Neargye/magic_enum/issues/284).
    name.str_ = __FUNCSIG__;
    name.str_ += 35;
    name.size_ = sizeof(__FUNCSIG__) - 52;
}
#else
auto name = str_view{};
#endif
std::size_t p = 0;
for (std::size_t i = name.size_; i > 0; --i) {
    if (name.str_[i] == ':') {
        p = i + 1;
        break;
    }
}
if (p > 0) {
    name.size_ -= p;
    name.str_ += p;
}
return name;
} else {
    return str_view{}; // Unsupported compiler or Invalid customize.
}
}

#if defined(_MSC_VER) && !defined(__clang__) && _MSC_VER < 1920
# define MAGIC_ENUM_VS_2017_WORKAROUND 1
#endif

#if defined(MAGIC_ENUM_VS_2017_WORKAROUND)
template <typename E, E V>
constexpr auto n() noexcept {
    static_assert(is_enum_v<E>, "magic_enum::detail::n requires enum type.");

    # if defined(MAGIC_ENUM_GET_ENUM_NAME_BUILTIN)
    constexpr auto name_ptr = MAGIC_ENUM_GET_ENUM_NAME_BUILTIN(V);
    auto name = name_ptr ? str_view{name_ptr, std::char_traits<char>::length(name_ptr)} : str_view{};
    # else
    // C/C++ workaround (see https://github.com/Neargye/magic_enum/issues/284).
    str_view name;
    name.str_ = __FUNCSIG__;
    name.size_ = sizeof(__FUNCSIG__) - 17;
    std::size_t p = 0;
    for (std::size_t i = name.size_; i > 0; --i) {
        if (name.str_[i] == '.' || name.str_[i] == ':') {
            p = i + 1;
            break;
        }
    }
    if (p > 0) {
        name.size_ -= p;
        name.str_ += p;
    }
    if (name.str_[0] == ' ' || name.str_[0] == '-' || (name.str_[0] >= '0' && name.str_[0] <= '9')) {
        name = str_view{};
    }
    return name;
    # endif
}
#endif

template <typename E, E V>
constexpr auto enum_name() noexcept {
    [[maybe_unused]] constexpr auto custom = customize::enum_name<E>(V);
    static_assert(std::is_same_v<std::decay_t<decltype(custom)>, customize::customize_t>, "magic_enum::customize requires customize_t type.");
    if constexpr (custom.first == customize::detail::customize_tag::custom_tag) {
        constexpr auto name = custom.second;
        static_assert(!name.empty(), "magic_enum::customize requires not empty string.");
        return static_str<name.size()>{name};
    } else if constexpr (custom.first == customize::detail::customize_tag::invalid_tag) {
        return static_str<0>{};
    } else if constexpr (custom.first == customize::detail::customize_tag::default_tag) {
        #if defined(MAGIC_ENUM_VS_2017_WORKAROUND)

```

```
constexpr auto name = n<E, V>();
#else
constexpr auto name = n<V>();
#endif
return static_str<name.size_>{name};
} else {
static_assert(always_false_v<E>, "magic_enum::customize invalid.");
}
}

template <typename E, E V>
inline constexpr auto enum_name_v = enum_name<E, V>();

template <typename E, auto V>
constexpr bool is_valid() noexcept {
#ifdef __clang__ && __clang_major__ >= 16
// https://reviews.lvm.org/D130058, https://reviews.lvm.org/D131307
constexpr E v = __builtin_bit_cast(E, V);
#else
constexpr E v = static_cast<E>(V);
#endif
[[maybe_unused]] constexpr auto custom = customize::enum_name<E>(v);
static_assert(std::is_same_v<std::decay_t<decltype(custom)>, customize::customize_t>, "magic_enum::customize requires customize_t type.");
if constexpr (custom.first == customize::detail::customize_tag::custom_tag) {
constexpr auto name = custom.second;
static_assert(!name.empty(), "magic_enum::customize requires not empty string.");
return name.size() != 0;
} else if constexpr (custom.first == customize::detail::customize_tag::default_tag) {
#ifdef MAGIC_ENUM_VS_2017_WORKAROUND
return n<E, v>().size_ != 0;
#else
return n<v>().size_ != 0;
#endif
} else {
return false;
}
}

enum class enum_subtype {
common,
flags
};

template <typename E, int O, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr U ualue(std::size_t i) noexcept {
if constexpr (std::is_same_v<U, bool>) { // bool special case
static_assert(O == 0, "magic_enum::detail::ualue requires valid offset.");

return static_cast<U>(i);
} else if constexpr (S == enum_subtype::flags) {
return static_cast<U>({1} << static_cast<U>(static_cast<int>(i) + O));
} else {
return static_cast<U>(static_cast<int>(i) + O);
}
}

template <typename E, int O, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr E value(std::size_t i) noexcept {
return static_cast<E>(ualue<E, O, S>(i));
}

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr int reflected_min() noexcept {
if constexpr (S == enum_subtype::flags) {
return 0;
} else {
constexpr auto lhs = range_min<E>::value;
constexpr auto rhs = (std::numeric_limits<U>::min)();

if constexpr (cmp_less(rhs, lhs)) {
return lhs;
} else {
return rhs;
}
}
}

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr int reflected_max() noexcept {
if constexpr (S == enum_subtype::flags) {
return std::numeric_limits<U>::digits - 1;
} else {
constexpr auto lhs = range_max<E>::value;
constexpr auto rhs = (std::numeric_limits<U>::max)();

if constexpr (cmp_less(lhs, rhs)) {
return lhs;
} else {
return rhs;
}
}
}

#define MAGIC_ENUM_FOR_EACH_256(T) \
T( 0)/T( 1)/T( 2)/T( 3)/T( 4)/T( 5)/T( 6)/T( 7)/T( 8)/T( 9)/T(10)/T(11)/T(12)/T(13)/T(14)/T(15)/T(16)/T(17)/T(18)/T(19)/T(20)/T(21)/T(22)/T(23)/T(24)/T(25)/T(26)/T(27)/T(28)/T(29)/T(30)/T(31) \
T(32)/T(33)/T(34)/T(35)/T(36)/T(37)/T(38)/T(39)/T(40)/T(41)/T(42)/T(43)/T(44)/T(45)/T(46)/T(47)/T(48)/T(49)/T(50)/T(51)/T(52)/T(53)/T(54)/T(55)/T(56)/T(57)/T(58)/T(59)/T(60)/T(61)/T(62)/T(63) \
T(64)/T(65)/T(66)/T(67)/T(68)/T(69)/T(70)/T(71)/T(72)/T(73)/T(74)/T(75)/T(76)/T(77)/T(78)/T(79)/T(80)/T(81)/T(82)/T(83)/T(84)/T(85)/T(86)/T(87)/T(88)/T(89)/T(90)/T(91)/T(92)/T(93)/T(94)/T(95) \
T(96)/T(97)/T(98)/T(99)/T(100)/T(101)/T(102)/T(103)/T(104)/T(105)/T(106)/T(107)/T(108)/T(109)/T(110)/T(111)/T(112)/T(113)/T(114)/T(115)/T(116)/T(117)/T(118)/T(119)/T(120)/T(121)/T(122)/T(123)/T(124)/T(125)/T(126)/T(127) \
T(128)/T(129)/T(130)/T(131)/T(132)/T(133)/T(134)/T(135)/T(136)/T(137)/T(138)/T(139)/T(140)/T(141)/T(142)/T(143)/T(144)/T(145)/T(146)/T(147)/T(148)/T(149)/T(150)/T(151)/T(152)/T(153)/T(154)/T(155)/T(156)/T(157)/T(158)/T(159) \
T(160)/T(161)/T(162)/T(163)/T(164)/T(165)/T(166)/T(167)/T(168)/T(169)/T(170)/T(171)/T(172)/T(173)/T(174)/T(175)/T(176)/T(177)/T(178)/T(179)/T(180)/T(181)/T(182)/T(183)/T(184)/T(185)/T(186)/T(187)/T(188)/T(189)/T(190)/T(191) \
T(192)/T(193)/T(194)/T(195)/T(196)/T(197)/T(198)/T(199)/T(200)/T(201)/T(202)/T(203)/T(204)/T(205)/T(206)/T(207)/T(208)/T(209)/T(210)/T(211)/T(212)/T(213)/T(214)/T(215)/T(216)/T(217)/T(218)/T(219)/T(220)/T(221)/T(222)/T(223) \
T(224)/T(225)/T(226)/T(227)/T(228)/T(229)/T(230)/T(231)/T(232)/T(233)/T(234)/T(235)/T(236)/T(237)/T(238)/T(239)/T(240)/T(241)/T(242)/T(243)/T(244)/T(245)/T(246)/T(247)/T(248)/T(249)/T(250)/T(251)/T(252)/T(253)/T(254)/T(255)
```

```
template <typename E, enum_subtype S, std::size_t Size, int Min, std::size_t I>
constexpr void valid_count(bool* valid, std::size_t& count) noexcept {
#define MAGIC_ENUM_V(O)
    if constexpr ((I + O) < Size) {
        if constexpr (is_valid<E, uvalue<E, Min, S>(I + O)>()) { \
            valid[I + O] = true;
            ++count;
        }
    }
}

MAGIC_ENUM_FOR_EACH_256(MAGIC_ENUM_V)

if constexpr ((I + 256) < Size) {
    valid_count<E, S, Size, Min, I + 256>(valid, count);
}
#undef MAGIC_ENUM_V
}

template <std::size_t N>
struct valid_count_t {
    std::size_t count = 0;
    bool valid[N] = { };
};

template <typename E, enum_subtype S, std::size_t Size, int Min>
constexpr auto valid_count() noexcept {
    valid_count_t<Size> vc;
    valid_count<E, S, Size, Min, 0>(vc.valid, vc.count);
    return vc;
}

template <typename E, enum_subtype S, std::size_t Size, int Min>
constexpr auto values() noexcept {
    constexpr auto vc = valid_count<E, S, Size, Min>();

    if constexpr (vc.count > 0) {
#if defined(MAGIC_ENUM_ARRAY_CONSTEXPR)
        std::array<E, vc.count> values = {};
#else
        E values[vc.count] = {};
#endif
        for (std::size_t i = 0, v = 0; v < vc.count; ++i) {
            if (vc.valid[i]) {
                values[v++] = value<E, Min, S>(i);
            }
        }
    }
#if defined(MAGIC_ENUM_ARRAY_CONSTEXPR)
    return values;
#else
    return to_array(values, std::make_index_sequence<vc.count>{});
#endif
} else {
    return std::array<E, 0>{};
}
}

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr auto values() noexcept {
    constexpr auto min = reflected_min<E, S>();
    constexpr auto max = reflected_max<E, S>();
    constexpr auto range_size = max - min + 1;
    static_assert(range_size > 0, "magic_enum::enum_range requires valid size.");
    static_assert(range_size < (std::numeric_limits<std::uint16_t>::max)(), "magic_enum::enum_range requires valid size.");

    return values<E, S, range_size, min>();
}

template <typename E, typename U = std::underlying_type_t<E>>
constexpr enum_subtype subtype(std::true_type) noexcept {
    if constexpr (std::is_same_v<U, bool>) { // bool special case
        return enum_subtype::common;
    } else if constexpr (has_is_flags<E>::value) {
        return customize::enum_range<E>::is_flags ? enum_subtype::flags : enum_subtype::common;
    } else {
#if defined(MAGIC_ENUM_AUTO_IS_FLAGS)
        constexpr auto flags_values = values<E, enum_subtype::flags>();
        constexpr auto default_values = values<E, enum_subtype::common>();
        if (flags_values.size() == 0 || default_values.size() > flags_values.size()) {
            return enum_subtype::common;
        }
        for (std::size_t i = 0; i < default_values.size(); ++i) {
            const auto v = static_cast<U>(default_values[i]);
            if (v != 0 && (v & (v - 1)) != 0) {
                return enum_subtype::common;
            }
        }
    }
    return enum_subtype::flags;
} else
    return enum_subtype::common;
#endif
}
}

template <typename T>
constexpr enum_subtype subtype(std::false_type) noexcept {
    // For non-enum type return default common subtype.
    return enum_subtype::common;
}
}

template <typename E, typename D = std::decay_t<E>>
inline constexpr auto subtype_v = subtype<D>(std::is_enum<D>{});
```

```
template <typename E, enum_subtype S>
inline constexpr auto values_v = values<E, S>();

template <typename E, enum_subtype S, typename D = std::decay_t<E>>
using values_t = decltype((values_v<D, S>));

template <typename E, enum_subtype S>
inline constexpr auto count_v = values_v<E, S>.size();

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
inline constexpr auto min_v = (count_v<E, S> > 0) ? static_cast<U>(values_v<E, S>.front()) : U{0};

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
inline constexpr auto max_v = (count_v<E, S> > 0) ? static_cast<U>(values_v<E, S>.back()) : U{0};

template <typename E, enum_subtype S, std::size_t... I>
constexpr auto names(std::index_sequence<I...>) noexcept {
    constexpr auto names = std::array<string_view, sizeof...(I)>{{enum_name_v<E, values_v<E, S>[I]>...}};
    return names;
}

template <typename E, enum_subtype S>
inline constexpr auto names_v = names<E, S>(std::make_index_sequence<count_v<E, S>>{});

template <typename E, enum_subtype S, typename D = std::decay_t<E>>
using names_t = decltype((names_v<D, S>));

template <typename E, enum_subtype S, std::size_t... I>
constexpr auto entries(std::index_sequence<I...>) noexcept {
    constexpr auto entries = std::array<std::pair<E, string_view>, sizeof...(I)>{{(values_v<E, S>[I], enum_name_v<E, values_v<E, S>[I]>)...}};
    return entries;
}

template <typename E, enum_subtype S>
inline constexpr auto entries_v = entries<E, S>(std::make_index_sequence<count_v<E, S>>{});

template <typename E, enum_subtype S, typename D = std::decay_t<E>>
using entries_t = decltype((entries_v<D, S>));

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr bool is_sparse() noexcept {
    if constexpr (count_v<E, S> == 0) {
        return false;
    } else if constexpr (std::is_same_v<U, bool>) { // bool special case
        return false;
    } else {
        constexpr auto max = (S == enum_subtype::flags) ? log2(max_v<E, S>) : max_v<E, S>;
        constexpr auto min = (S == enum_subtype::flags) ? log2(min_v<E, S>) : min_v<E, S>;
        constexpr auto range_size = max - min + 1;

        return range_size != count_v<E, S>;
    }
}

template <typename E, enum_subtype S = subtype_v<E>>
inline constexpr bool is_sparse_v = is_sparse<E, S>();

template <typename E, enum_subtype S, typename U = std::underlying_type_t<E>>
constexpr U values_ors() noexcept {
    static_assert(S == enum_subtype::flags, "magic_enum::detail::values_ors requires valid subtype.");

    auto ors = U{0};
    for (std::size_t i = 0; i < count_v<E, S>; ++i) {
        ors |= static_cast<U>(values_v<E, S>[i]);
    }

    return ors;
}

template <bool, typename R>
struct enable_if_enum {};

template <typename R>
struct enable_if_enum<true, R> {
    using type = R;
    static_assert(supported<R>::value, "magic_enum unsupported compiler (https://github.com/Neargye/magic\_enum#compiler-compatibility).");
};

template <typename T, typename R, typename BinaryPredicate = std::equal_to<>, typename D = std::decay_t<T>>
using enable_if_t = typename enable_if_enum<std::is_enum_v<D> && std::is_invocable_r_v<bool, BinaryPredicate, char_type, char_type>, R>::type;

template <typename T, std::enable_if_t<std::is_enum_v<std::decay_t<T>>, int> = 0>
using enum_concept = T;

template <typename T, bool = std::is_enum_v<T>>
struct is_scoped_enum : std::false_type {};

template <typename T>
struct is_scoped_enum<T, true> : std::bool_constant<!std::is_convertible_v<T, std::underlying_type_t<T>>> {};

template <typename T, bool = std::is_enum_v<T>>
struct is_unscoped_enum : std::false_type {};

template <typename T>
struct is_unscoped_enum<T, true> : std::bool_constant<std::is_convertible_v<T, std::underlying_type_t<T>>> {};

template <typename T, bool = std::is_enum_v<std::decay_t<T>>>
struct underlying_type {};

template <typename T>
struct underlying_type<T, true> : std::underlying_type<std::decay_t<T>> {};
```

```
#if defined(MAGIC_ENUM_ENABLE_HASH) || defined(MAGIC_ENUM_ENABLE_HASH_SWITCH)

template <typename Value, typename = void>
struct constexpr_hash_t;

template <typename Value>
struct constexpr_hash_t<Value, std::enable_if_t<is_enum_v<Value>>> {
    constexpr auto operator()(Value value) const noexcept {
        using U = typename underlying_type<Value>::type;
        if constexpr (std::is_same_v<U, bool>) { // bool special case
            return static_cast<std::size_t>(value);
        } else {
            return static_cast<U>(value);
        }
    }
};

using secondary_hash = constexpr_hash_t;

template <typename Value>
struct constexpr_hash_t<Value, std::enable_if_t<std::is_same_v<Value, string_view>>> {
    static constexpr std::uint32_t crc_table[256] {
        0x00000000L, 0x7073096L, 0xae0e612cL, 0x990951baL, 0x076dc419L, 0x706af48fL, 0xe963a535L, 0x9e6495a3L,
        0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
        0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL, 0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L,
        0x136c9856L, 0x6464ba8cL, 0xf6d2f97aL, 0x8a65c9ecL, 0x14015c4fL, 0x63066cd9L, 0xfa0f3d63L, 0x8d080df5L,
        0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L, 0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
        0x35b5a8faL, 0x42b2986cL, 0xdbbcb9d6L, 0xacbcf940L, 0x32d86cceL, 0x45df5c75L, 0xdcdd60dcfL, 0xabd13d59L,
        0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfb9a959L, 0xb8bda50fL,
        0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L, 0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb666d3dL,
        0x76dc4190L, 0x01db7106L, 0x98d220bcL, 0xfedf5102aL, 0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L, 0xe8b8d433L,
        0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
        0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L,
        0x65bd9c9L, 0x12b7e950L, 0x8bbeb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L, 0xfbd44c65L,
        0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L, 0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL,
        0x4369e96aL, 0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L,
        0x5005713cL, 0x270241aaL, 0xe0b1010L, 0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
        0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L, 0x2eb40d81L, 0xb7bd5c3bL, 0x0ba6cadL,
        0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL, 0x74b1d29aL, 0xeada5739L, 0x9dd27affL, 0x04db2615L, 0x73dc1683L,
        0xe3630b12L, 0x94643b84L, 0xd6d66a3eL, 0x7a6a5aa8L, 0xe40ecf0bL, 0x9309ff9dL, 0xa0a00ae27L, 0x7d079eb1L,
        0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf76257dL, 0x806567cbL, 0x196c3671L, 0x6e6b06e7L,
        0xfed41b76L, 0x89d32be0L, 0x10da7a5aL, 0x67dd4accL, 0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
        0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL,
        0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eefL, 0x46699e79L,
        0xcb61b38cL, 0xbcc6681aL, 0x256fd2a0L, 0x5268e236L, 0xccc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL,
        0xc55ba3bbeL, 0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d2ffa7L, 0xb5d0cf31L, 0x2cd99e8bL, 0x5bdeae1dL,
        0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL, 0x9c0906a9L, 0xeb05c363fL, 0x72076785L, 0x05005713L,
        0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L, 0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L,
        0x86d3d2d4L, 0xf1dd4e24L, 0x68dddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L, 0x18b74777L,
        0x88085ae6L, 0xff0fa70L, 0x66063bcaL, 0x11010b5cL, 0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L,
        0xa00ae278L, 0xd70dd2eeL, 0x4e048354L, 0x39033bc2L, 0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e3e677dbL,
        0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L, 0x37d83bf0L, 0xa9bcae53L, 0xdeb99ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
        0xabddbf21L, 0xcabac28aL, 0x53b3930L, 0x24b4a3a6L, 0xbad03605L, 0xcdd70693L, 0x54de5729L, 0x23d967bfL,
        0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L, 0x2a6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
    };

    constexpr std::uint32_t operator()(string_view value) const noexcept {
        auto crc = static_cast<std::uint32_t>(0xffffffffL);
        for (const auto c : value) {
            crc = (crc >> 8) ^ crc_table[(crc ^ static_cast<std::uint32_t>(c)) & 0xff];
        }
        return crc ^ 0xffffffffL;
    }
};

struct secondary_hash {
    constexpr std::uint32_t operator()(string_view value) const noexcept {
        auto acc = static_cast<std::uint64_t>(2166136261ULL);
        for (const auto c : value) {
            acc = ((acc ^ static_cast<std::uint64_t>(c)) * static_cast<std::uint64_t>(16777619ULL)) & (std::numeric_limits<std::uint32_t>::max());
        }
        return static_cast<std::uint32_t>(acc);
    }
};

template <typename Hash>
inline constexpr Hash hash_v[];

template <auto* GlobValues, typename Hash>
constexpr auto calculate_cases(const Hash_t Page) noexcept {
    constexpr std::array values = *GlobValues;
    constexpr std::size_t size = values.size();

    using switch_t = std::invoke_result_t<Hash, typename decltype(values)::value_type>;
    static_assert(std::is_integral_v<switch_t> && !std::is_same_v<switch_t, bool>);
    const std::size_t values_to = (std::min)(static_cast<std::size_t>(256), size - Page);

    std::array<switch_t, 256> result{};
    auto fill = result.begin();
    {
        auto first = values.begin() + static_cast<std::ptrdiff_t>(Page);
        auto last = values.begin() + static_cast<std::ptrdiff_t>(Page + values_to);
        while (first != last) {
            *fill++ = hash_v<Hash>(*first++);
        }
    }

    // dead cases, try to avoid case collisions
    for (switch_t last_value = result[values_to - 1]; fill != result.end() && last_value != (std::numeric_limits<switch_t>::max()); *fill++ = ++last_value) {}

    {
        auto it = result.begin();
    }
};
```

```

    auto last_value = (std::numeric_limits<switch_t>::min)();
    for (; fill != result.end(); *fill++ = last_value++) {
        while (last_value == *it) {
            ++last_value, ++it;
        }
    }
}

return result;
}

template <typename R, typename F, typename... Args>
constexpr R invoke_r(F&& f, Args&&... args) noexcept(std::is_nothrow_invocable_r_v<R, F, Args...>) {
    if constexpr (std::is_void_v<R>) {
        std::forward<F>(f)(std::forward<Args>(args)...);
    } else {
        return static_cast<R>(std::forward<F>(f)(std::forward<Args>(args)...));
    }
}

enum class case_call_t {
    index,
    value
};

template <typename T = void>
inline constexpr auto default_result_type_lambda = []() noexcept(std::is_nothrow_default_constructible_v<T>) { return T{}; };

template <>
inline constexpr auto default_result_type_lambda<void> = []() noexcept {};

template <auto* Arr, typename Hash>
constexpr bool has_duplicate() noexcept {
    using value_t = std::decay_t<decltype>(*Arr)[0]>;
    using hash_value_t = std::invoke_result_t<Hash, value_t>;
    std::array<hash_value_t, Arr->size()> hashes{};
    std::size_t size = 0;
    for (auto elem : *Arr) {
        hashes[size] = hash_v<Hash>(elem);
        for (auto i = size++; i > 0; --i) {
            if (hashes[i] < hashes[i - 1]) {
                auto tmp = hashes[i];
                hashes[i] = hashes[i - 1];
                hashes[i - 1] = tmp;
            } else if (hashes[i] == hashes[i - 1]) {
                return false;
            } else {
                break;
            }
        }
    }
    return true;
}

#define MAGIC_ENUM_CASE(val)
case cases[val]:
    if constexpr ((val) + Page < size) {
        if (!pred(values[val + Page], searched)) {
            break;
        }
    }
    if constexpr (CallValue == case_call_t::index) {
        if constexpr (std::is_invocable_r_v<result_t, Lambda, std::integral_constant<std::size_t, val + Page>>) {
            return detail::invoke_r<result_t>(std::forward<Lambda>(lambda), std::integral_constant<std::size_t, val + Page>{});
        } else if constexpr (std::is_invocable_v<Lambda, std::integral_constant<std::size_t, val + Page>>) {
            MAGIC_ENUM_ASSERT(false && "magic_enum::detail::constexpr_switch wrong result type.");
        }
    }
    if constexpr (CallValue == case_call_t::value) {
        if constexpr (std::is_invocable_r_v<result_t, Lambda, enum_constant<values[val + Page]>>) {
            return detail::invoke_r<result_t>(std::forward<Lambda>(lambda), enum_constant<values[val + Page]>{});
        } else if constexpr (std::is_invocable_r_v<result_t, Lambda, enum_constant<values[val + Page]>>) {
            MAGIC_ENUM_ASSERT(false && "magic_enum::detail::constexpr_switch wrong result type.");
        }
    }
    break;
} else [[fallthrough]];

template <auto* GlobValues,
        case_call_t CallValue,
        std::size_t Page = 0,
        typename Hash = constexpr_hash_t<typename std::decay_t<decltype(*GlobValues)>::value_type>,
        typename BinaryPredicate = std::equal_to<>,
        typename Lambda,
        typename ResultGetterType>
constexpr decltype(auto) constexpr_switch(
    Lambda&& lambda,
    typename std::decay_t<decltype(*GlobValues)>::value_type searched,
    ResultGetterType&& def,
    BinaryPredicate&& pred = {} ) {
    using result_t = std::invoke_result_t<ResultGetterType>;
    using hash_t = std::conditional_t<has_duplicate<GlobValues, Hash>(), Hash, typename Hash::secondary_hash>;
    static_assert(has_duplicate<GlobValues, hash_t>(), "magic_enum::detail::constexpr_switch duplicated hash found, please report it: https://github.com/Neargye/magic_enum/issues.");
    constexpr std::array values = *GlobValues;
    constexpr std::size_t size = values.size();
    constexpr std::array cases = calculate_cases<GlobValues, hash_t>(Page);

    switch (hash_v<hash_t>(searched)) {
        MAGIC_ENUM_FOR_EACH_256(MAGIC_ENUM_CASE)
    default:
        if constexpr (size > 256 + Page) {
            return constexpr_switch<GlobValues, CallValue, Page + 256, Hash>(std::forward<Lambda>(lambda), searched, std::forward<ResultGetterType>(def));
        }
        break;
    }
}

```

```

}
return def();
}

#undef MAGIC_ENUM_CASE

#endif

} // namespace magic_enum::detail

// Checks is magic_enum supported compiler.
inline constexpr bool is_magic_enum_supported = detail::supported<void>::value;

template <typename T>
using Enum = detail::enum_concept<T>;

// Checks whether T is an Unscoped enumeration type.
// Provides the member constant value which is equal to true, if T is an [Unscoped enumeration](https://en.cppreference.com/w/cpp/language/enum#Unscoped_enumeration) type. Otherwise, value is equal to false.
template <typename T>
struct is_unscoped_enum : detail::is_unscoped_enum<T> {};

template <typename T>
inline constexpr bool is_unscoped_enum_v = is_unscoped_enum<T>::value;

// Checks whether T is an Scoped enumeration type.
// Provides the member constant value which is equal to true, if T is an [Scoped enumeration](https://en.cppreference.com/w/cpp/language/enum#Scoped_enumerations) type. Otherwise, value is equal to false.
template <typename T>
struct is_scoped_enum : detail::is_scoped_enum<T> {};

template <typename T>
inline constexpr bool is_scoped_enum_v = is_scoped_enum<T>::value;

// If T is a complete enumeration type, provides a member typedef type that names the underlying type of T.
// Otherwise, if T is not an enumeration type, there is no member type. Otherwise (T is an incomplete enumeration type), the program is ill-formed.
template <typename T>
struct underlying_type : detail::underlying_type<T> {};

template <typename T>
using underlying_type_t = typename underlying_type<T>::type;

template <auto V>
using enum_constant = detail::enum_constant<V>;

// Returns type name of enum.
template <typename E>
[[nodiscard]] constexpr auto enum_type_name() noexcept -> detail::enable_if_t<E, string_view> {
    constexpr string_view name = detail::type_name_v<std::decay_t<E>>;
    static_assert(!name.empty(), "magic_enum::enum_type_name enum type does not have a name.");

    return name;
}

// Returns number of enum values.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_count() noexcept -> detail::enable_if_t<E, std::size_t> {
    return detail::count_v<std::decay_t<E>, S>;
}

// Returns enum value at specified index.
// No bounds checking is performed: the behavior is undefined if index >= number of enum values.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_value(std::size_t index) noexcept -> detail::enable_if_t<E, std::decay_t<E>> {
    using D = std::decay_t<E>;

    if constexpr (detail::is_sparse_v<D, S>) {
        return MAGIC_ENUM_ASSERT(index < detail::count_v<D, S>), detail::values_v<D, S>[index];
    } else {
        constexpr auto min = (S == detail::enum_subtype::flags) ? detail::log2(detail::min_v<D, S>) : detail::min_v<D, S>;

        return MAGIC_ENUM_ASSERT(index < detail::count_v<D, S>), detail::value<D, min, S>(index);
    }
}

// Returns enum value at specified index.
template <typename E, std::size_t I, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_value() noexcept -> detail::enable_if_t<E, std::decay_t<E>> {
    using D = std::decay_t<E>;
    static_assert(I < detail::count_v<D, S>, "magic_enum::enum_value out of range.");

    return enum_value<D, S>(I);
}

// Returns std::array with enum values, sorted by enum value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_values() noexcept -> detail::enable_if_t<E, detail::values_t<E, S>> {
    return detail::values_v<std::decay_t<E>, S>;
}

// Returns integer value from enum value.
template <typename E>
[[nodiscard]] constexpr auto enum_integer(E value) noexcept -> detail::enable_if_t<E, underlying_type_t<E>> {
    return static_cast<underlying_type_t<E>>(value);
}

// Returns underlying value from enum value.
template <typename E>
[[nodiscard]] constexpr auto enum_underlying(E value) noexcept -> detail::enable_if_t<E, underlying_type_t<E>> {
    return static_cast<underlying_type_t<E>>(value);
}

// Obtains index in enum values from enum value.
// Returns optional with index.

```

```
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_index(E value) noexcept -> detail::enable_if_t<E, optional<std::size_t>> {
    using D = std::decay_t<E>;
    using U = underlying_type_t<D>;

    if constexpr (detail::count_v<D, S> == 0) {
        static_cast<void>(value);
        return {}; // Empty enum.
    } else if constexpr (detail::is_sparse_v<D, S> || (S == detail::enum_subtype::flags)) {
#ifdef MAGIC_ENUM_ENABLE_HASH
        return detail::constexpr_switch<&detail::values_v<D, S>, detail::case_call_t::index>(
            [(std::size_t i) { return optional<std::size_t>{i}; },
             value,
             detail::default_result_type_lambda<optional<std::size_t>>];
        );
    #else
        for (std::size_t i = 0; i < detail::count_v<D, S>; ++i) {
            if (enum_value<D, S>(i) == value) {
                return i;
            }
        }
        return {}; // Invalid value or out of range.
    #endif
    } else {
        const auto v = static_cast<U>(value);
        if (v >= detail::min_v<D, S> && v <= detail::max_v<D, S>) {
            return static_cast<std::size_t>(v - detail::min_v<D, S>);
        }
        return {}; // Invalid value or out of range.
    }
}

// Obtains index in enum values from enum value.
// Returns optional with index.
template <detail::enum_subtype S, typename E>
[[nodiscard]] constexpr auto enum_index(E value) noexcept -> detail::enable_if_t<E, optional<std::size_t>> {
    using D = std::decay_t<E>;

    return enum_index<D, S>(value);
}

// Obtains index in enum values from static storage enum variable.
template <auto V, detail::enum_subtype S = detail::subtype_v<std::decay_t<decltype(V)>>>
[[nodiscard]] constexpr auto enum_index() noexcept -> detail::enable_if_t<decltype(V), std::size_t> {
    constexpr auto index = enum_index<std::decay_t<decltype(V)>, S>(V);
    static_assert(index, "magic_enum::enum_index enum value does not have a index.");

    return *index;
}

// Returns name from static storage enum variable.
// This version is much lighter on the compile times and is not restricted to the enum_range limitation.
template <auto V>
[[nodiscard]] constexpr auto enum_name() noexcept -> detail::enable_if_t<decltype(V), string_view> {
    constexpr string_view name = detail::enum_name_v<std::decay_t<decltype(V)>, V>;
    static_assert(!name.empty(), "magic_enum::enum_name enum value does not have a name.");

    return name;
}

// Returns name from enum value.
// If enum value does not have name or value out of range, returns empty string.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_name(E value) noexcept -> detail::enable_if_t<E, string_view> {
    using D = std::decay_t<E>;

    if (const auto i = enum_index<D, S>(value)) {
        return detail::names_v<D, S>[*i];
    }
    return {};
}

// Returns name from enum value.
// If enum value does not have name or value out of range, returns empty string.
template <detail::enum_subtype S, typename E>
[[nodiscard]] constexpr auto enum_name(E value) -> detail::enable_if_t<E, string_view> {
    using D = std::decay_t<E>;

    return enum_name<D, S>(value);
}

// Returns std::array with names, sorted by enum value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_names() noexcept -> detail::enable_if_t<E, detail::names_t<E, S>> {
    return detail::names_v<std::decay_t<E>, S>;
}

// Returns std::array with pairs (value, name), sorted by enum value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_entries() noexcept -> detail::enable_if_t<E, detail::entries_t<E, S>> {
    return detail::entries_v<std::decay_t<E>, S>;
}

// Allows you to write magic_enum::enum_cast<foo>("bar", magic_enum::case_insensitive);
inline constexpr auto case_insensitive = detail::case_insensitive<>{};

// Obtains enum value from integer value.
// Returns optional with enum value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_cast(underlying_type_t<E> value) noexcept -> detail::enable_if_t<E, optional<std::decay_t<E>>> {
    using D = std::decay_t<E>;

    if constexpr (detail::count_v<D, S> == 0) {
```



```

static_cast<void>(value);
return {}; // Empty enum.
} else {
    if constexpr (detail::is_sparse_v<D, S> || (S == detail::enum_subtype::flags)) {
#ifdef MAGIC_ENUM_ENABLE_HASH
        return detail::constexpr_switch<&detail::values_v<D, S>, detail::case_call_t::value>(
            [](D v) { return optional<D>{v}; },
            static_cast<D>(value),
            detail::default_result_type_lambda<optional<D>>());
    }
#else
        for (std::size_t i = 0; i < detail::count_v<D, S>; ++i) {
            if (value == static_cast<underlying_type_t<D>>(enum_value<D, S>(i))) {
                return static_cast<D>(value);
            }
        }
        return {}; // Invalid value or out of range.
    }
#endif
} else {
    if (value >= detail::min_v<D, S> && value <= detail::max_v<D, S>) {
        return static_cast<D>(value);
    }
    return {}; // Invalid value or out of range.
}
}

// Obtains enum value from name.
// Returns optional with enum value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>, typename BinaryPredicate = std::equal_to<>>
[[nodiscard]] constexpr auto enum_cast(string_view value, [[maybe_unused]] BinaryPredicate p = {}) noexcept(detail::is_nothrow_invocable<BinaryPredicate>()) -> detail::enable_if_t<E, optional<std::decay_t<E>>, BinaryPredicate>
{
    using D = std::decay_t<E>;

    if constexpr (detail::count_v<D, S> == 0) {
        static_cast<void>(value);
        return {}; // Empty enum.
    }
#ifdef MAGIC_ENUM_ENABLE_HASH
    } else if constexpr (detail::is_default_predicate<BinaryPredicate>()) {
        return detail::constexpr_switch<&detail::names_v<D, S>, detail::case_call_t::index>(
            [](std::size_t i) { return optional<D>(detail::values_v<D, S>[i]); },
            value,
            detail::default_result_type_lambda<optional<D>>(),
            [&p](string_view lhs, string_view rhs) { return detail::cmp_equal(lhs, rhs, p); });
    }
#else
    } else {
        for (std::size_t i = 0; i < detail::count_v<D, S>; ++i) {
            if (detail::cmp_equal(value, detail::names_v<D, S>[i], p)) {
                return enum_value<D, S>(i);
            }
        }
        return {}; // Invalid value or out of range.
    }
}

// Checks whether enum contains value with such value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_contains(E value) noexcept -> detail::enable_if_t<E, bool> {
    using D = std::decay_t<E>;
    using U = underlying_type_t<D>;

    return static_cast<bool>(enum_cast<D, S>(static_cast<U>(value)));
}

// Checks whether enum contains value with such value.
template <detail::enum_subtype S, typename E>
[[nodiscard]] constexpr auto enum_contains(E value) noexcept -> detail::enable_if_t<E, bool> {
    using D = std::decay_t<E>;
    using U = underlying_type_t<D>;

    return static_cast<bool>(enum_cast<D, S>(static_cast<U>(value)));
}

// Checks whether enum contains value with such integer value.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>>
[[nodiscard]] constexpr auto enum_contains(underlying_type_t<E> value) noexcept -> detail::enable_if_t<E, bool> {
    using D = std::decay_t<E>;

    return static_cast<bool>(enum_cast<D, S>(value));
}

// Checks whether enum contains enumerator with such name.
template <typename E, detail::enum_subtype S = detail::subtype_v<E>, typename BinaryPredicate = std::equal_to<>>
[[nodiscard]] constexpr auto enum_contains(string_view value, BinaryPredicate p = {}) noexcept(detail::is_nothrow_invocable<BinaryPredicate>()) -> detail::enable_if_t<E, bool, BinaryPredicate> {
    using D = std::decay_t<E>;

    return static_cast<bool>(enum_cast<D, S>(value, std::move(p)));
}

template <bool AsFlags = true>
inline constexpr auto as_flags = AsFlags ? detail::enum_subtype::flags : detail::enum_subtype::common;

template <bool AsFlags = true>
inline constexpr auto as_common = AsFlags ? detail::enum_subtype::common : detail::enum_subtype::flags;

namespace bitwise_operators {

template <typename E, detail::enable_if_t<E, int> = 0>
constexpr E operator~(E rhs) noexcept {
    return static_cast<E>(~static_cast<underlying_type_t<E>>(rhs));
}

template <typename E, detail::enable_if_t<E, int> = 0>

```

```
constexpr E operator|(E lhs, E rhs) noexcept {
    return static_cast<E>(static_cast<underlying_type_t<E>>(lhs) | static_cast<underlying_type_t<E>>(rhs));
}

template <typename E, detail::enable_if_t<E, int> = 0>
constexpr E operator&(E lhs, E rhs) noexcept {
    return static_cast<E>(static_cast<underlying_type_t<E>>(lhs) & static_cast<underlying_type_t<E>>(rhs));
}

template <typename E, detail::enable_if_t<E, int> = 0>
constexpr E operator^(E lhs, E rhs) noexcept {
    return static_cast<E>(static_cast<underlying_type_t<E>>(lhs) ^ static_cast<underlying_type_t<E>>(rhs));
}

template <typename E, detail::enable_if_t<E, int> = 0>
constexpr E& operator|=(E& lhs, E rhs) noexcept {
    return lhs |= (lhs | rhs);
}

template <typename E, detail::enable_if_t<E, int> = 0>
constexpr E& operator&=(E& lhs, E rhs) noexcept {
    return lhs = (lhs & rhs);
}

template <typename E, detail::enable_if_t<E, int> = 0>
constexpr E& operator^=(E& lhs, E rhs) noexcept {
    return lhs = (lhs ^ rhs);
}

} // namespace magic_enum::bitwise_operators

} // namespace magic_enum

#ifdef __clang__
# pragma clang diagnostic pop
#elif defined(__GNUC__)
# pragma GCC diagnostic pop
#elif defined(_MSC_VER)
# pragma warning(pop)
#endif

#undef MAGIC_ENUM_GET_ENUM_NAME_BUILTIN
#undef MAGIC_ENUM_GET_TYPE_NAME_BUILTIN
#undef MAGIC_ENUM_VS_2017_WORKAROUND
#undef MAGIC_ENUM_ARRAY_CONSTEXPR
#undef MAGIC_ENUM_FOR_EACH_256

#endif // NEARGYE_MAGIC_ENUM_HPP
```

singleton.hpp

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

template<class T>
class singleton
{
public:
    static std::shared_ptr<T> Instance()
    {
        struct EnableMakeShared : public T { EnableMakeShared() : T() {} };

        std::call_once(m_onceFlag, []() { m_instance = std::make_shared<EnableMakeShared>(); });

        return m_instance;
    }

protected:
    singleton() = default;
    virtual ~singleton() = default;

    BLOCK_COPY_MOVE(singleton);

private:
    static std::shared_ptr<T> m_instance;
    static std::once_flag m_onceFlag;
};

template<class T>
std::shared_ptr<T> singleton<T>::m_instance = nullptr;

template<class T>
std::once_flag singleton<T>::m_onceFlag;
```


stdafx.h

```
#pragma once
#define NOMINMAX

#include <Windows.h>

#include <string>
#include <vector>
#include <map>
#include <functional>
#include <fstream>
#include <algorithm>
#include <iostream>
#include <variant>
#include <filesystem>
#include <memory>
#include <type_traits>
#include <limits>
#include <numeric>
#include <set>

#define BLOCK_COPY(classname) \
private: \
    classname(const classname&) = delete; \
    classname& operator=(const classname&) = delete; \

#define BLOCK_MOVE(classname) \
private: \
    classname(const classname&&) = delete; \
    classname& operator=(const classname&&) = delete

#define BLOCK_COPY_MOVE(classname) \
    BLOCK_COPY(classname) \
    BLOCK_MOVE(classname)
```

StringUtils.h

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

namespace StringUtils
{
    enum ComparisonMode : std::uint16_t
    {
        BeginsWith = 0,
        Contains,
        EndWith,
        Iequals
    };

    template <typename C, typename T>
    auto Compare(const std::basic_string<C>& str1, const T& str2, ComparisonMode mode, bool caseSensitive = true)
        -> typename std::enable_if<std::is_convertible_v<T, std::basic_string<C>>, bool>
    {
        std::basic_string<C> internalStr1{ str1 };
        std::basic_string<C> internalStr2{ str2 };

        if (!caseSensitive)
        {
            std::transform(internalStr1.begin(), internalStr1.end(), internalStr1.begin(), ::tolower);
            std::transform(internalStr2.begin(), internalStr2.end(), internalStr2.begin(), ::tolower);
        }

        switch (mode)
        {
            case ComparisonMode::BeginsWith:
                return internalStr1.find(internalStr2) == 0;
            case ComparisonMode::Contains:
                return internalStr1.find(internalStr2) != std::basic_string<C>::npos;
            case ComparisonMode::EndWith:
            {
                auto pos = internalStr1.rfind(internalStr2);
                return (pos == (internalStr1.length() - internalStr2.length())) && (pos != std::basic_string<C>::npos);
            }
            case ComparisonMode::Iequals:
                return internalStr1 == internalStr2;
        }

        return false;
    }
}
```

TablePrinter.cpp

```
#include "stdafx.h"
#include "Utils/TablePrinter.h"

void TablePrinter::PrintItem(std::ostream& out, const std::string& item, size_t width, Alignment alignment, size_t padding, bool lastItemInRow)
{
    out << "|";
    width -= 2 * padding;

    switch (alignment)
    {
    case RIGHT:
        out << std::string(padding, ' ') << std::setw(width) << std::right << item << std::string(padding, ' ');
        break;
    case LEFT:
        out << std::string(padding, ' ') << std::setw(width) << std::left << item << std::string(padding, ' ');
        break;
    case CENTRE:
    {
        if (auto paddingCenter = width - item.size() + 2 * padding)
        {
            int paddingRight = paddingCenter / 2;
            int paddingLeft = paddingCenter - paddingRight;
            out << std::setw(paddingLeft + item.size()) << item << std::setw(paddingRight) << " ";
        }
        else
        {
            out << std::string(padding, ' ') << item << std::string(padding, ' ');
        }
        break;
    }
    }

    if (lastItemInRow)
        out << "|";
}
```

TablePrinter.h

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

class TablePrinter
{
public:
    enum Alignment
    {
        RIGHT,
        LEFT,
        CENTRE
    };

    template <template <typename, typename...> class ContainerType, typename T, typename... Args>
    static void PrintTable(std::ostream& out,
        const std::vector<std::string>& headers,
        const std::vector<size_t>& columnWidths,
        const std::vector<Alignment>& columnAlignments,
        const ContainerType<T, Args...>& items,
        const std::vector<std::function<std::string(const T&)>>& dataGetters,
        size_t padding = 0)
    {
        if(headers.size() != columnWidths.size() || headers.size() != columnAlignments.size() || headers.size() != dataGetters.size())
            throw std::invalid_argument("TablePrinter::PrintTable: Number of headers, column widths, column alignments and data getters must be the same");

        auto totalWidth = std::accumulate(columnWidths.begin(), columnWidths.end(), size_t(0)) + headers.size() * padding + 1;

        out << std::string(totalWidth, '=') << std::endl;
        for (size_t i = 0; i < headers.size(); i++)
        {
            PrintItem(out, headers[i], columnWidths[i], CENTRE, 0, i == headers.size() - 1);
        }
        out << std::endl;

        for (size_t i = 0; i < headers.size(); i++)
        {
            PrintItem(out, std::string(columnWidths[i], '='), columnWidths[i], CENTRE, 0, i == headers.size() - 1);
        }
        out << std::endl;

        for (const auto& item : items)
        {
            for (size_t i = 0; i < dataGetters.size(); i++)
            {
                PrintItem(out, dataGetters[i](item), columnWidths[i], columnAlignments[i], padding, i == headers.size() - 1);
            }
            out << std::endl;
        }
        out << std::string(totalWidth, '=') << std::endl;
    }

private:
    static void PrintItem(std::ostream& out, const std::string& item, size_t width, Alignment alignment, size_t padding = 0, bool lastItemInRow = false);
};
```