Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Пояснювальна записка**

До курсового проєкту «СИСТЕМНЕ ПРОГРАМУВАННЯ»

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

Індивідуальне завдання

“РОЗРОБКА ТРАНСЛЯТОРА З ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ”

**Варіант №26**

**Виконав:** ст. гр. КІ-307

Соніч О.В.

**Перевірив:**

Козак Н. Б.

Львів-2024

ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ

**Варіант 26**

Завдання на курсовий проект

1. Цільова мова транслятора – асемблер для 32-розрядного процесора.

2. Для отримання виконавчого файлу на виході розробленого транслятора скористатися програмами ml.exe і link.exe.

3. Мова розробки транслятора: C++.

4. Реалізувати оболонку або інтерфейс з командного рядка.

5. На вхід розробленого транслятора має подаватися текстовий файл, написаний на заданій мові програмування.

6. На виході розробленого транслятора мають створюватись такі файли:

* *файл з лексемами;*
* *файл з повідомленнями про помилки (або про їх відсутність);*
* *файл на мові асемблера;*
* *об’єктний файл;*
* *виконавчий файл.*

7. Назва вхідної мови програмування утворюється від першої букви у прізвищі студента та останніх двох цифр номера його варіанту. Саме таке розширення повинні мати текстові файли, написані на цій мові програмування.

**Деталізація завдання на проектування:**

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

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

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

**Деталізований опис власної мови програмування:**

Розширення файлу - .s26

Опис вхідної мови програмування:

* Тип даних: INTEGER\_2
* Блок тіла програми: NAME <name>; BODY DATA…; END
* Оператор вводу: SCAN ()
* Оператор виводу: PRINT ()
* Оператори: IF ELSE (C)

GOTO (C)

FOR-TO-DO (Паскаль)

FOR-DOWNTO-DO (Паскаль)

WHILE (Бейсік)

REPEAT-UNTIL (Паскаль)

* Регістр ключових слів: Up
* Регістр ідентифікаторів: Low-Up6 перший символ \_
* Операції арифметичні: ADD, SUB, MUL, DIV, MOD
* Операції порівняння: EQ, NE, >=, <=
* Операції логічні: NOT, AND, OR
* Коментар: !!...
* Ідентифікатори змінних, числові константи
* Оператор присвоєння: <-

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

АНОТАЦІЯ

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

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

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

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

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

Зміст

[ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ 2](#_Toc186564488)

[АНОТАЦІЯ 5](#_Toc186564489)

[ВСТУП 7](#_Toc186564490)

[1. ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЕКТУВАННЯ ТРАНСЛЯТОРІВ 8](#_Toc186564491)

[2. ФОРМАЛЬНИЙ ОПИС ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ 11](#_Toc186564492)

[2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура 11](#_Toc186564493)

[2.2 Опис термінальних символів та ключових слів 13](#_Toc186564494)

[3. РОЗРОБКА ТРАНСЛЯТОРА ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ 15](#_Toc186564495)

[3.1 Вибір технології програмування 15](#_Toc186564496)

[3.2 Проектування таблиць транслятора 16](#_Toc186564497)

[3.3 Розробка лексичного аналізатора 17](#_Toc186564498)

[3.3.1 Розробка алгоритму роботи лексичного аналізатора 20](#_Toc186564499)

[3.3.2 Опис програми реалізації лексичного аналізатора 21](#_Toc186564500)

[3.4 Розробка синтаксичного та семантичного аналізатора 23](#_Toc186564501)

[3.4.1 Розробка дерева граматичного розбору. 24](#_Toc186564502)

[3.4.2 Опис програми реалізації синтаксичного та семантичного аналізатора 26](#_Toc186564503)

[3.4.3 Розробка граф-схеми алгоритму 29](#_Toc186564504)

[3.5 Розробка генератора коду 34](#_Toc186564505)

[3.5.1 Розробка алгоритму роботи генератора коду 35](#_Toc186564506)

[3.5.2 Опис програми реалізації генератора коду 36](#_Toc186564507)

[4. ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА 44](#_Toc186564508)

[4.1 Опис інтерфейсу та інструкція користувачеві 44](#_Toc186564509)

[4.2 Виявлення лексичних та синтаксичних помилок 45](#_Toc186564510)

[4.3 Перевірка роботи транслятора за допомогою тестових задач 46](#_Toc186564511)

[4.4 Тестова програма №1 48](#_Toc186564512)

[4.5 Тестова програма №2 49](#_Toc186564513)

[4.6 Тестова програма №3 50](#_Toc186564514)

[ВИСНОВКИ 52](#_Toc186564515)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 53](#_Toc186564516)

[ДОДАТКИ 53](#_Toc186564517)

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 відміну 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 транслюють 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).

Нотація БНФ є набором «продукцій», кожна з яких відповідає зразку:

<символ> = <вираз, що містить символи>

де вираз, що містить символи це послідовність символів або послідовності символів, розділених вертикальною рискою |, що повністю перелічують можливий вибір символ з лівої частини формули.

У розширеній формі нотації Бекуса — Наура вирази, що можна пропускати або які можуть повторятись слід записувати у фігурних дужках { ... }:, а можлива поява може відображатися застосуванням квадратних дужок [ ... ]:.

topRule = "**NAME**", identifier , ";", "**BODY**", varsBlok, ";", operators, "**END**";

varsBlok = "**DATA**", "**INTEGER\_2**", identifier, [{ commaAndIdentifier }];

identifier = “\_”, low\_letter, { up\_letter | number } {6};

commaAndIdentifier = ",", identifier;

codeBlok = "**BODY**", write | read | assignment | ifStatement | goto\_statement | labelRule | forToOrDownToDoRule | while | repeatUntil, "**END**";

operators = {write | read | assignment | ifStatement | goto\_statement | labelRule | forToOrDownToDoRule | while | repeatUntil};

read = "**SCAN**", "(", identifier, ")";

write = "**PRINT**", "(", equation | stringRule, ")";

assignment = identifier, "**<-**", equation;

ifStatement = "**IF**", "(", equation, ")", codeBlok, ["**ELSE**", codeBlok];

goto\_statement = "**GOTO**", ident ;

labelRule = identifier, ":";

forToOrDownToDoRule = "**FOR**", assignment, "**TO**" | "**DOWNTO**", equation, "Do**",** codeBlok;

while = "**WHILE**", "(", equation, ")", "**BODY**", operators | whileContinue | whileExit, "**END**", "**WHILE**";

whileContinue = "**CONTINUE**", "**WHILE**";

whileExit = "**EXIT**", "**WHILE**";

repeatUntil = "**REPEAT**", operators, "**UNTIL**", "(", equation, ")";

equation = signedNumber | identifier | notRule [{ operationAndIdentOrNumber | equation }];

notRule = notOperation, signedNumber | identifier | equation;

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

arithmetic = "**ADD**" | "**SUB**";

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

logic = "**AND**" | "**OR**";

notOperation = "**NOT**";

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

stringRule = "“", string, "“";

comment = "**LComment**" string ;

LComment = "**!!**";

string = { low\_letter | up\_letter | number };

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

sign = "+" | "-";

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

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

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

* 1. Опис термінальних символів та ключових слів

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

|  |  |
| --- | --- |
| Термінальний символ або ключове слово | Значення |
| NAME | Початок програми |
| BODY | Початок тексту програми |
| DATA | Початок блоку опису змінних |
| END | Кінець розділу операторів |
| SCAN | Оператор вводу змінних |
| PRINT | Оператор виводу (змінних або рядкових констант) |
| <- | Оператор присвоєння |
| IF | Оператор умови |
| ELSE | Оператор умови |
| GOTO | Оператор переходу |
| LABEL | Мітка переходу |
| FOR | Оператор циклу |
| TO | Інкремент циклу |
| DOWNTO | Декремент циклу |
| DO | Початок тіла циклу |
| WHILE | Оператор циклу |
| CONTINUE | Оператор циклу |
| EXIT | Оператор циклу |
| REPEAT | Початок тіла циклу |
| UNTIL | Оператор циклу |
| ADD | Оператор додавання |
| SUB | Оператор віднімання |
| MUL | Оператор множення |
| DIV | Оператор ділення |
| MOD | Оператор знаходження залишку від ділення |
| EQ | Оператор перевірки на рівність |
| NE | Оператор перевірки на нерівність |
| <= | Оператор перевірки чи менше |
| >= | Оператор перевірки чи більше |
| NOT | Оператор логічного заперечення |
| AND | Оператор кон’юнкції |
| OR | Оператор диз’юнкції |
| INTEGER\_2 | 16-ти розрядні знакові цілі |
| !!… | Коментар |
| , | Розділювач |
| ; | Ознака кінця оператора |
| ( | Відкриваюча дужка |
| ) | Закриваюча дужка |

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

1. РОЗРОБКА ТРАНСЛЯТОРА ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ
   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: Х – початкова, Y – об`єктна та Z – інструментальна. Транслятор перекладає програми мовою Х 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

* 1. 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

1. Таблиця лексичних класів

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

Таблиця 2 Опис термінальних символі та ключових слів

|  |  |
| --- | --- |
| **Токен** | **Значення** |
| Program | NAME |
| Start | BODY |
| Vars | DATA |
| End | END |
| VarType | INTEGER\_2 |
| Read | SCAN |
| Write | PRINT |
| Assignment | <- |
| If | IF |
| Else | ELSE |
| Goto | GOTO |
| Colon | : |
| Label |  |
| For | FOR |
| To | TO |
| DownTo | DOWNTO |
| Do | DO |
| While | WHILE |
| ContinueWhile | CONTINUE |
| ExitWhile | EXIT |
| Repeat | REPEAT |
| Until | UNTIL |
| Addition | ADD |
| Subtraction | SUB |
| Multiplication | MUL |
| Division | DIV |
| Mod | MOD |
| Equal | EQ |
| NotEqual | NE |
| Less | <= |
| Greate | >= |
| Not | NOT |
| And | AND |
| Or | OR |
| Plus | + |
| Minus | - |
| Identifier |  |
| Number |  |
| String |  |
| Undefined |  |
| Unknown |  |
| Comma | , |
| Quotes | “ |
| Semicolon | ; |
| Lbraket | ( |
| Rbraket | ) |
| LComment | !! |
| Comment |  |

* 1. 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;

Розробимо алгоритм роботи лексичного аналізатора на основі скінченного автомату. Лексичний аналізатор працює за принципом скінченного автомату з такими станами:

* **Start** - початок виділення чергової лексеми;
* **Finish** - кінець виділення чергової лексеми;
* **EndOfFile** - кінець файлу, завершення розпізнавання лексем;
* **Letter** - перший символ буква, розпізнавання слів (ключові слова і ідентифікатори);
* **Digit** - перший символ цифра, розпізнавання числових констант;
* **Separators** - видалення пробілів, символів табуляції і переходу на новий рядок;
* **Scomment** – перші символи “**!!”**, далі може йти коментар;
* **Comment** - видалення тексту коментаря;
* **Another** - опрацювання інших символів.

У стані **Letter** читаємо по одному символи з файлу і виділяємо ланцюжок символів, який починається з символу «\_», а далі слідує один символ нижнього регістру та декілька верхнього регістру. Кінець ланцюжка - якщо прочитаний символ відмінний від букви чи цифри. Виділений ланцюжок порівнюємо з ключовими словами, якщо співпадінь немає, вважаємо його ідентифікатором при умові, що довжина ланцюжка не більше 6-х символів, інакше це невизначена лексема. Переходимо до стану **Finish**.

У стані **Digit** читаємо по одному символи з файлу і виділяємо ланцюжок символів, який починається з крапки, мінуса або ж цифри, далі ж йдуть лише цифри або крапки, вважаємо цей ланцюжок числовою константою. Кінець ланцюжка - якщо прочитаний символ відмінний від цифри. Переходимо до стану **Finish**.

У стані **Scomment** читаємо наступний символ, якщо це пара символів “!!**”**, то далі до кінця рядка йде коментар, який можна проігнорувати, переходимо до стану **Comment**. Якщо ж наступний символи не є парою “**!!”**, то вважаємо що поточна лексема – невідома, читаємо наступний символ і переходимо до стану **Finish**.

У стані **Comment** читаємо символи, поки не зустрінеться символ переходу на новий рядок, після цього переходимо до виділення нової лексеми - до стану **Start**.

У стані **Separators** читаємо наступний символ і переходимо до виділення нової лексеми - до стану **Start**. Тобто пропускаємо усі пробіли, символи табуляції і переходу на новий рядок.

У стані **Another** порівнюємо поточний прочитаний символ з символами, що позначають знаки операцій, розділювачі і круглі дужки і визначаємо одну з лексем. Є кілька лексем, які вимагають ще читання наступного символу з файлу - це оператор присвоєння “<-” і операцій “>=”, “<=”. Якщо співпадіння не виявлено, то поточний символ - невідома лексема, читаємо наступний символ і переходимо до стану **Finish**.

У стані **Finish** записуємо поточну лексему у таблицю лексем і переходимо до виділення нової лексеми, до стану **Start**.

У стані **EndOfFile** завершуємо обробку вхідного файлу, усі символи з файлу прочитані, усі лексеми записані у таблицю лексем.

Алгоритм роботи лексичного аналізатора можна зобразити у вигляді граф- схеми.

* + 1. Розробка алгоритму роботи лексичного аналізатора

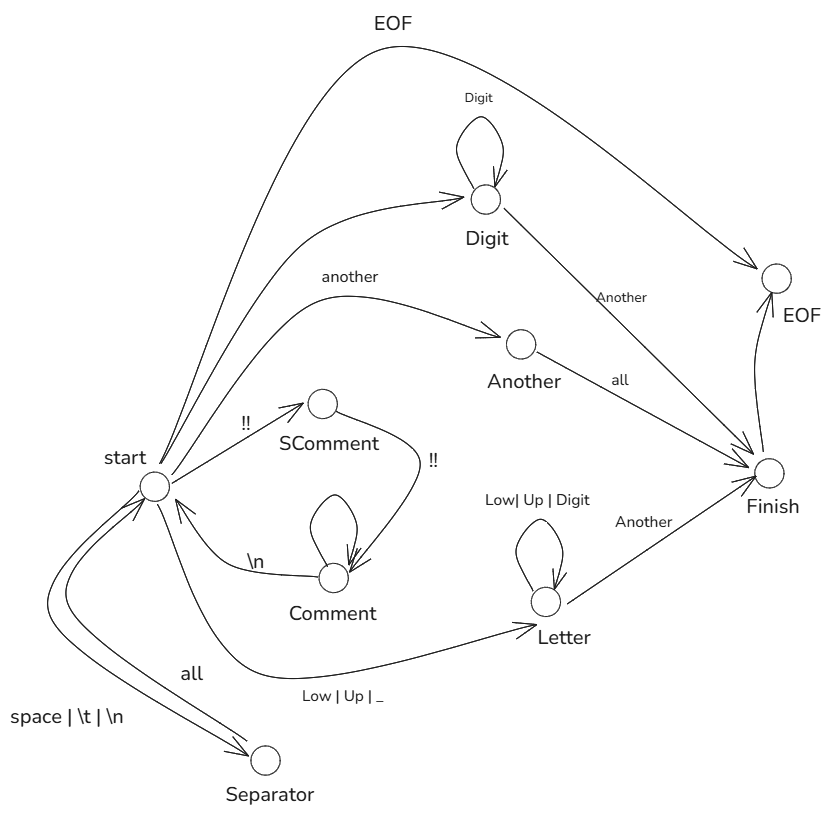


Рис. 3.1 Граф-схема алгоритму роботи лексичного аналізатора

* + 1. 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. Вхiдна програма проглядається послідовно з початку до кінця. Базовi елементи, або лексичнi одиницi, роздiляються пробілами, знаками операцiй i спецiальними символами (новий рядок, знак табуляції), i таким чином видiляються та розпізнаються iдентифiкатори, лiтерали i термiнальнi символи (операцiї, ключові слова).

Програма аналізує файл поки не досягне його кінця. Для вхідного файлу викликається функція tokenize(). Вона зчитує з файлу його вміст та кожну лексему порівнює з зарезервованою 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 компiляції 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 аналiзі виявляються i вiдзначаються лексичнi помилки (наприклад, недопустимi символи i неправильнi iдентифiкатори). Лексична фаза вiдкидає також коментарi та символи лапок у конструкції String, оскiльки вони не мають нiякого 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.

Прототип методу tokenize:

std::list<std::shared\_ptr<IToken>> tokenize(std::istream& input);

Список лексем:

std::list<std::shared\_ptr<IToken>> m\_tokens;

Метод для розпізнавання токену:

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

Метод для виводу таблиці:

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)

* 1. 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-граматик, 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.

* + 1. Розробка дерева граматичного розбору.

Схема дерева розбору виглядає наступним чином:

Program

├── "NAME"

├── "BODY"

├── {VariableDeclaration ";"}

│ ├── VariableDeclaration

│ │ ├── "INTEGER\_2"

│ │ └── VariableList

│ │ ├── Identifier

│ │ │ ├── "\_"

│ │ │ └── LowUp {5}

│ │ │ ├── Low

│ │ │ │ └── "a" | "b" | ... | "z"

│ │ │ ├── UP

│ │ │ │ └── "A" | "B" | ... | "Z"

│ │ │ └── Digit

│ │ │ └── "0" | "1" | ... | "9"

│ │ └── {"," Identifier}

├── {Statement}

│ ├── Statement

│ │ ├── InputStatement

│ │ │ ├── "SCAN"

│ │ │ └── Identifier

│ │ ├── OutputStatement

│ │ │ ├── "PRINT"

│ │ │ └── ArithmeticExpression

│ │ │ ├── LowPriorityExpression

│ │ │ │ ├── MiddlePriorityExpression

│ │ │ │ │ ├── Identifier

│ │ │ │ │ ├── Number

│ │ │ │ │ │ ├── ["SUB"]

│ │ │ │ │ │ └── Digit {5}

│ │ │ │ │ └── "(" ArithmeticExpression ")"

│ │ │ │ └── {MiddlePriorityOperator MiddlePriorityExpression}

│ │ │ └── {LowPriorityOperator LowPriorityExpression}

│ │ ├── AssignStatement

│ │ │ ├── ArithmeticExpression

│ │ │ └── "<-" Identifier

│ │ ├── IfElseStatement

│ │ │ ├── "IF"

│ │ │ ├── "(" LogicalExpression ")"

│ │ │ │ ├── AndExpression

│ │ │ │ │ ├── Comparison

│ │ │ │ │ │ ├── ComparisonExpression

│ │ │ │ │ │ │ ├── ArithmeticExpression

│ │ │ │ │ │ │ ├── ComparisonOperator

│ │ │ │ │ │ │ └── ArithmeticExpression

│ │ │ │ │ │ └── [NOT] "(" LogicalExpression ")"

│ │ │ │ │ └── {AND AndExpression}

│ │ │ │ └── {OR AndExpression}

│ │ │ ├── Statement

│ │ │ └── ["ELSE" Statement]

│ │ ├── GotoStatement

│ │ │ ├── "GOTO"

│ │ │ └── Identifier

│ │ ├── LabelPoint

│ │ │ ├── Identifier

│ │ │ └── ":"

│ │ ├── ForToStatement

│ │ │ ├── "FOR"

│ │ │ ├── AssignStatement

│ │ │ ├── "TO" | "DOWNTO"

│ │ │ ├── ArithmeticExpression

│ │ │ ├── "DO"

│ │ │ └── Statement

│ │ ├── WhileStatement

│ │ │ ├── "WHILE"

│ │ │ ├── LogicalExpression

│ │ │ ├── {Statement}

│ │ │ └── "END" "WHILE"

│ │ ├── RepeatUntilStatement

│ │ │ ├── "REPEAT"

│ │ │ ├── {Statement}

│ │ │ └── "UNTIL" "(" LogicalExpression ")"

│ │ └── CompoundStatement

│ │ ├── "STARTBLOCK"

│ │ ├── {Statement}

│ │ └── "ENDBLOCK"

└── "END"

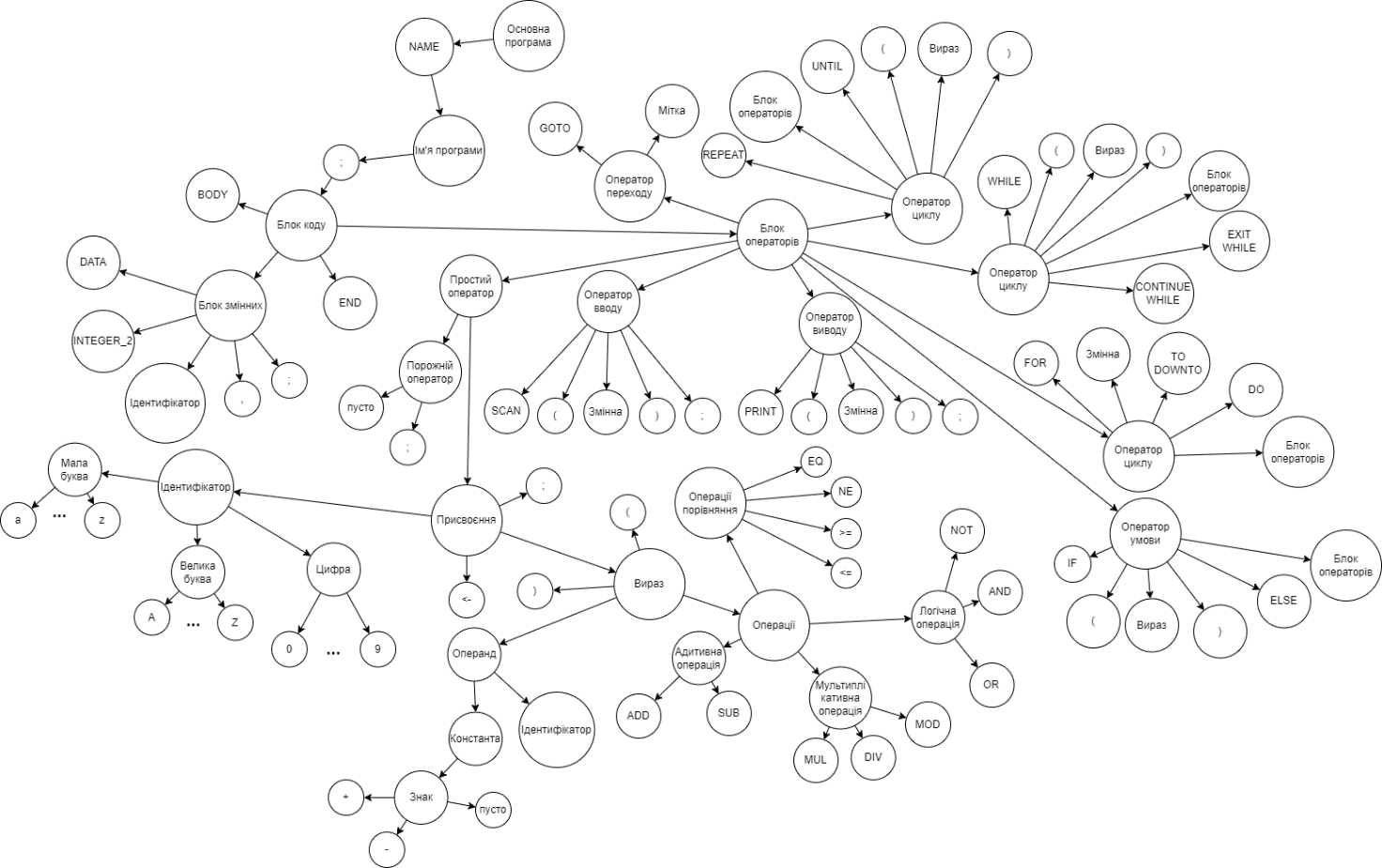


Рис. 3.2 Дерево граматичного розбору

* + 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.

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

### Базова структура перевірки:

* **Інтерфейс IBackusRule** визначає базовий контракт для всіх правил.

### Політики перевірки правил:

**Enum RuleCountPolicy** визначає можливі варіанти входження правил:

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

### Реєстрація та зберігання правил:

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

### Визначення правил граматики:

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

### Процес перевірки:

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

### Обробка помилок:

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

### Додаткові можливості:

* Підтримка **post-handlers** для додаткової обробки після перевірки правила.
* Можливість розширення через додавання нових правил.
* Контекстно-залежна перевірка через **Context**.

### Цей механізм дозволяє:

* Перевіряти відповідність коду заданій EBNF граматиці.
* Гнучко налаштовувати правила перевірки.
* Отримувати детальну інформацію про помилки.
* Розширювати граматику новими правилами.
  + 1. Розробка блок-схеми алгоритму

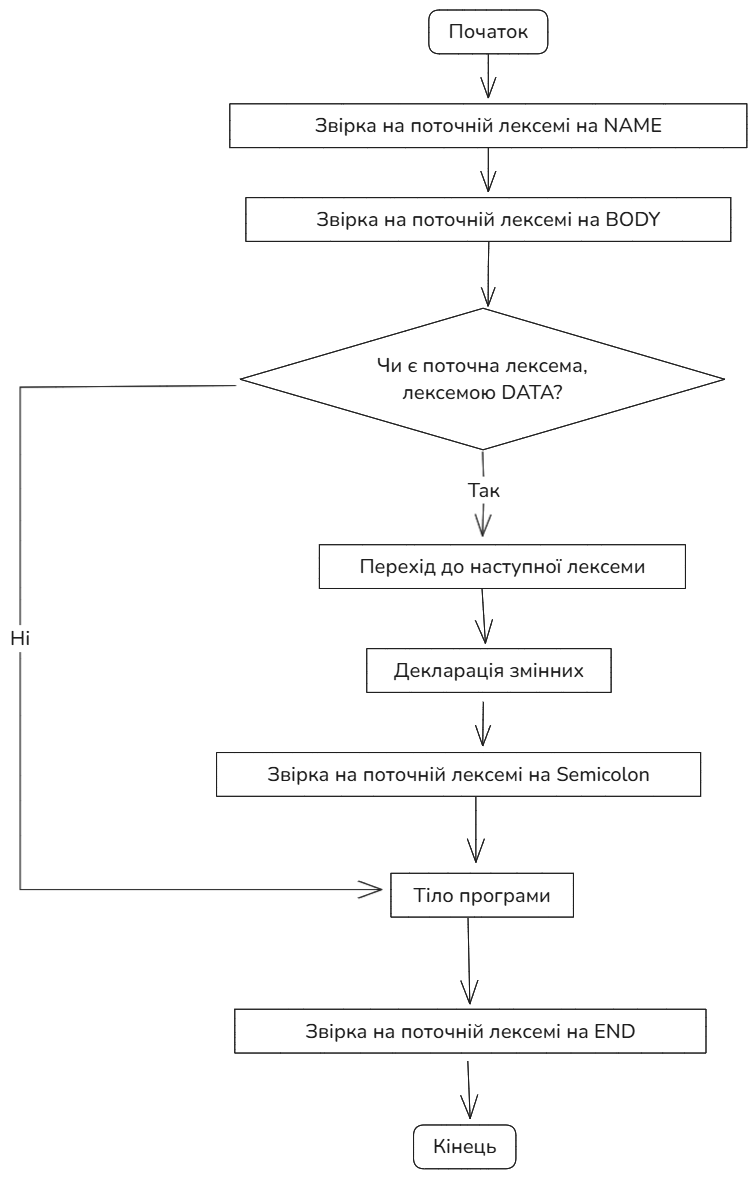


Рис. 3.3 Блок-схема роботи синтаксичного аналізатора

Верхньорівневий код, який описує блок схема 3.3

auto topRule = controller->addRule("TopRule", {

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

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

BackusRuleItem({ Symbols::Semicolon}, OnlyOne),

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

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

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

BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional | OneOrMore),

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

});

Код що перевіряє валідність оголошених змінних

std::shared\_ptr<IToken> tryCreateToken(std::string& lexeme) const override

{

if (lexeme.size() > (m\_mask.size() + m\_prefix.size()))

return nullptr;

bool res = true;

if (!lexeme.starts\_with(m\_prefix))

{

return nullptr;

}

std::string\_view ident{ lexeme.begin() + m\_prefix.size(), lexeme.end() };

for (size\_t i = 0; i < ident.size(); i++)

{

if ((isupper(ident[i]) != isupper(m\_mask[i])) && !isdigit(ident[i]))

{

res &= false;

break;

}

}

std::shared\_ptr<IToken> token = nullptr;

if (res)

{

token = clone();

token->setValue(lexeme);

lexeme.clear();

}

return token;

};

І приватні поля що задають формат:

const std::string m\_prefix = "\_";

const std::string m\_mask = "xXXXXXX";

* + 1. Розробка алгоритму роботи семантичного аналізатора

На етапі семантичного аналізу нам необхідно вирішити задачу ідентифікації ідентифікаторів. Алгоритм ідентифікації складається з двох частин:

* перша частина алгоритму опрацьовує оголошення ідентифікаторів;
* друга частина алгоритму опрацьовує використання ідентифікаторів.

Нехай лексичний аналізатор видав чергову лексему, що є ідентифікатором. Лексичний аналізатор сформував структуру, що містить атрибути виділеної лексеми, такі як ім’я ідентифікатора, його тип і лексичний клас. Далі вся ця інформація передається семантичному аналізатору. Припустимо, що в даний момент опрацьовується оголошення ідентифікатора. Основна семантична дія в цьому випадку полягає в занесенні інформації про ідентифікатор у таблицю ідентифікаторів.

Опрацювання використання ідентифікатора. Припустимо, що уже побудовано (цілком чи частково) таблицю ідентифікаторів. Далі вся ця інформація передається фазі використання ідентифікаторів. Таким чином, відомо, що опрацьовується використання ідентифікатора. Для того, щоб одержати інформацію про тип ідентифікатора нам достатньо прочитати певне поле таблиці ідентифікаторів.

3.4.5. Опис програмної реалізації семантичного аналізатора

Семантичний аналізатор виконує перевірку правильності структур та логіки програми на основі аналізу лексем та граматики. У цьому коді реалізовано кілька функцій, які відповідають за різні аспекти семантичного аналізу.

**Ключові аспекти реалізації**

1. **Лексеми та граматика:**
   * Семантичний аналізатор працює з таблицею лексем та граматикою, які є результатами попередніх етапів аналізу (лексичного та синтаксичного).
   * Типи лексем визначаються полем type, для отримання типу використовується функція GetTypeName.
2. **Перевірка колізій:**  
   Семантичний аналізатор знаходить конфлікти в ідентифікаторах, щоб уникнути неоднозначності або помилок у виконанні програми.
3. **Повідомлення про помилки:**  
   Усі помилки виводяться у консоль. semErr
4. **Опис:**Починається з виклику CheckSemantic, Context зберігає стан семантичного агалізу.
5. **Особливості реалізації**Рекурсивна перевірка правил

Підтримка необов'язкових правил

Обробка парних конструкцій

Використання політик для правил (RuleCountPolicy)

Основна частина методу CheckSemantic (лістинг не включає в себе обробку помилок)

bool CheckSemantic(std::ostream& out, std::list<std::shared\_ptr<T>>& tokens)

{

auto endOfFileType = tokens.back()->type();

std::list<std::shared\_ptr<IBackusRule>> rules;

for (auto token : tokens)

{

if (auto rule = std::dynamic\_pointer\_cast<IBackusRule>(token))

rules.push\_back(rule);

}

auto it = rules.begin();

auto end = rules.end();

std::multimap<int, std::pair<std::string, std::vector<std::string>>> errors;

auto res = Controller::Instance()->topRule()->check(errors, it, end);

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

end = --rules.end();

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

int lexErr = 0;

int synErr = 0;

int semErr = 0;

tokens.clear();

for (auto rule : rules)

{

tokens.push\_back(std::dynamic\_pointer\_cast<T>(rule));

if (rule->type() == Undefined::Type())

{

res = false;

std::string err;

if (auto erMsg = rule->customData("error"); !erMsg.empty())

{

semErr++;

err = "Semantic error: " + erMsg;

}

else

{

semErr++;

err = std::format("Semantic error: Undefined token: {}", rule->value());

}

errorsMsg.emplace(rule->line(), err);

}

else if (rule->type() == token::Unknown::Type())

{

lexErr++;

res = false;

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

}

}

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

identRule->setPostHandler([context](BackusRuleList::iterator&,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

static bool isFirstIdentChecked = !context->IsFirstProgName();

auto isVarBlockChecked = context->IsVarBlockChecked();

auto& identTable = context->IdentTable();

auto identIt = std::prev(it, 1);

if (isVarBlockChecked)

{

if (!identTable.contains((\*identIt)->value()))

{

auto undef = std::make\_shared<Undefined>();

undef->setValue((\*identIt)->value());

undef->setLine((\*identIt)->line());

undef->setCustomData((\*identIt)->customData());

\*identIt = undef;

}

}

else

{

if (isFirstIdentChecked)

{

identTable.insert((\*identIt)->value());

}

else

{

auto progName = std::make\_shared<ProgramName>();

progName->setValue((\*identIt)->value());

progName->setLine((\*identIt)->line());

progName->setCustomData((\*identIt)->customData());

\*identIt = progName;

isFirstIdentChecked = true;

}}

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

});

return identRule;}

* 1. 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. 3азвичай входом для генератора коду служить проміжна форма представлення програми, а на виході може з’являтися об’єктний код або модуль завантаження.

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

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

* + 1. Розробка алгоритму роботи генератора коду



Рис. 3.4 Блок схема генератора коду

* + 1. 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, зміні, 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 своє роботи, генератор формує код завершення ассемблерної програми.

**Основна структура:**

Використовує патерн Singleton через клас Generator

Базується на шаблоні Visitor, де кожен токен/правило реалізує метод genCode()

Використовує GeneratorDetails для зберігання налаштувань та допоміжних даних

**Процес генерації:**

Генерація відбувається в кілька етапів:

Генерація сегменту даних

Генерація сегменту коду

Генерація процедур

Завершення програми

Особливості реалізації:

Використовує стекову архітектуру для обчислень

Підтримує постфіксну форму для виразів

Використовує систему міток для управління потоком виконання

Має механізм реєстрації та генерації допоміжних процедур

**Система міток:**

Використовує customData для передачі інформації між етапами генерації

Автоматично генерує унікальні мітки для циклів та умов

Підтримує іменовані мітки для GOTO

**Оптимізації:**

Мінімізує використання регістрів через стекову модель

Перевикористовує процедури через механізм реєстрації

Оптимізує генерацію коду для простих випадків

**Обробка даних:**

Підтримує числові та рядкові дані

Використовує Windows API для введення/виведення

Має систему форматування для виведення різних типів даних

**Розширюваність:**

Дозволяє легко додавати нові оператори через систему токенів

Підтримує реєстрацію користувацьких процедур

Має гнучку систему налаштувань через GeneratorDetails

Програма має вигляд:

Program

/ \

var statement

Оголошення змінних:

var

/ \

Id var

/ \

Id null

Тіло програми:

Statement

/ \

Statement Оператор

/ \

statement Оператор

Оператор вводу:

Input

/ \

Id null

Оператор виводу:

Output

/ \

Id null

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

Output

/ \

Add null

/ \

Id num

Умовний оператор (IF() оператор;):

If

/ \

Умова оператор

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

If

/ \

Умова else

/ \

Оператор1 оператор2

Оператор безумовного переходу:

Goto

/ \

Id null

Оператор циклу for:

For

/ \

(to | downto) оператор

/ \

Оп. Прис. ариф. вир.

Оператор циклу while:

While

/ \

Умова statement

/ \

Statement оператор

/ \

Оператор оператор

Оператор циклу repeat:

Repeat

/ \

Statement умова

/ \

Оператор оператор

Оператор присвоєння:

<-

/ \

Id арифметичний вираз

Арифметичний вираз:

(ADD або MUL)

/ \

Id id

Доданок:

(MUL, DIV або MOD)  
/ \  
множник множник

Множник:

фактор  
/ \

id або number або (арифм. вираз) null

Складений оператор:

compount

/ \

statement null

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

auto assingmentRule = BackusRule::MakeRule("AssignmentRule", {

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

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

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

});

auto read = BackusRule::MakeRule("ReadRule", {

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

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

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

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

});

auto write = BackusRule::MakeRule("WriteRule", {

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

BackusRuleItem({ LBraket::Type()}, OnlyOne | PairStart),

BackusRuleItem({ stringRule->type(), equation->type() }, OnlyOne),

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

});

auto codeBlok = BackusRule::MakeRule("CodeBlok", {

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

BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional | OneOrMore),

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

});

auto topRule = BackusRule::MakeRule("TopRule", {

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

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

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

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

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

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

});

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

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

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

Кожен токен це окремий клас що наслідує 3 інтерфейси:

* IToken
* IBackusRule
* IGeneratorItem

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

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

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

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

Для класу та токену 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 анадізувати програми великого обсягу.

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

1. ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ТРАНСЛЯТОРА
   1. Опис інтерфейсу та інструкція користувачеві

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

Створений транслятор є консольною програмою, що запускається з командної стрічки з параметром: "CWork\_s26.exe <ім’я програми>.s26"

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

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

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

Для отримання виконавчого файлу необхідно скористатись програмою Masm32.exe

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

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

* 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

NAME \_pROGRA1;

BODY

DATA INT EGER\_2 \_aAAAAAAA,\_bBBBBBB,\_xXXXXXX,\_yYYYYYY;

PRINT("Input A: ");

SCAN(\_aAAeAAAA);

PRI NT("Input B: ");

SCAN(\_bBBBBBB);

PRINT("A + B: ");

PRINT(\_aAAAAAA ADD \_bBBBBBB);

PRINT("\nA - B: ");

PRINT(\_aAAAAAA SUB \_bBBBBBB);

PRINT("\nA \* B: ");

PRINT(\_aAAAAAA MUL \_bBBBBBB);

PRINT("\nA / B: ");

PRINT(\_aAAAAAA DIV \_bBBBBBB);

PRINT("\nA % B: ");

PRINT(\_aAAAAAA MOD \_bBBBBBB);

\_xXXXXXX<-(\_aAAAAAA SUB \_bBBBBBB) MUL 10 ADD (\_aAAAAAA ADD \_bBBBBBB) DIV 10;

\_yYYYYYY<-\_xXXXXXX ADD (\_xXXXXXX MOD 10);

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

PRINT(\_xXXXXXX);

PRINT("\nY = X + (X % 10)\n");

PRINT(\_yYYYYYY);

END

***Текст файлу з повідомленнями про помилки***

List of errors

======================================================================

There are 6 lexical errors.

There are 1 syntax errors.

There are 0 semantic errors.

Line 4: Lexical error: Unknown token: INT

Line 4: Lexical error: Unknown token: EGER\_2

Line 4: Lexical error: Unknown token: \_aAAAAAAA

Line 4: Syntax error: Expected: VarsBlok before INT

Line 6: Lexical error: Unknown token: \_aAAeAAAA

Line 7: Lexical error: Unknown token: PRI

Line 7: Lexical error: Unknown token: NT

Суттю виявлення семантичних помилок є перевірка числових констант на відповідність типу INTEGER\_2, тобто знаковому цілому числу з відповідним діапазоном значень і перевірку на коректність використання змінних INTEGER\_2 у цілочисельних і логічних виразах.

* 1. Перевірка роботи транслятора за допомогою тестових задач

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

***Текст коректної програми***

!!Prog1

NAME \_pROGRA1;

BODY

DATA INTEGER\_2 \_aAAAAAA,\_bBBBBBB,\_xXXXXXX,\_yYYYYYY;

PRINT("Input A: ");

SCAN(\_aAAAAAA);

PRINT("Input B: ");

SCAN(\_bBBBBBB);

PRINT("A + B: ");

PRINT(\_aAAAAAA ADD \_bBBBBBB);

PRINT("\nA - B: ");

PRINT(\_aAAAAAA SUB \_bBBBBBB);

PRINT("\nA \* B: ");

PRINT(\_aAAAAAA MUL \_bBBBBBB);

PRINT("\nA / B: ");

PRINT(\_aAAAAAA DIV \_bBBBBBB);

PRINT("\nA % B: ");

PRINT(\_aAAAAAA MOD \_bBBBBBB);

\_xXXXXXX<-(\_aAAAAAA SUB \_bBBBBBB) MUL 10 ADD (\_aAAAAAA ADD \_bBBBBBB) DIV 10;

\_yYYYYYY<-\_xXXXXXX ADD (\_xXXXXXX MOD 10);

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

PRINT(\_xXXXXXX);

PRINT("\nY = X + (X % 10)\n");

PRINT(\_yYYYYYY);

END

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

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

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



Рис. 4.1 Результат виконання коректної програми

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

* 1. Тестова програма №1

***Текст програми***

!!Prog1

NAME \_pROGRA1;

BODY

DATA INTEGER\_2 \_aAAAAAA,\_bBBBBBB,\_xXXXXXX,\_yYYYYYY;

PRINT("Input A: ");

SCAN(\_aAAAAAA);

PRINT("Input B: ");

SCAN(\_bBBBBBB);

PRINT("A + B: ");

PRINT(\_aAAAAAA ADD \_bBBBBBB);

PRINT("\nA - B: ");

PRINT(\_aAAAAAA SUB \_bBBBBBB);

PRINT("\nA \* B: ");

PRINT(\_aAAAAAA MUL \_bBBBBBB);

PRINT("\nA / B: ");

PRINT(\_aAAAAAA DIV \_bBBBBBB);

PRINT("\nA % B: ");

PRINT(\_aAAAAAA MOD \_bBBBBBB);

\_xXXXXXX<-(\_aAAAAAA SUB \_bBBBBBB) MUL 10 ADD (\_aAAAAAA ADD \_bBBBBBB) DIV 10;

\_yYYYYYY<-\_xXXXXXX ADD (\_xXXXXXX MOD 10);

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

PRINT(\_xXXXXXX);

PRINT("\nY = X + (X % 10)\n");

PRINT(\_yYYYYYY);

END

***Результат виконання***



Рис. 4.2 Результат виконання тестової програми №1

* 1. Тестова програма №2

***Текст програми***

!!Prog2

NAME \_pROGRA2;

BODY

DATA INTEGER\_2 \_aAAAAAA,\_bBBBBBB,\_cCCCCCC;

PRINT("Input A: ");

SCAN(\_aAAAAAA);

PRINT("Input B: ");

SCAN(\_bBBBBBB);

PRINT("Input C: ");

SCAN(\_cCCCCCC);

IF(\_aAAAAAA >= \_bBBBBBB)

BODY

IF(\_aAAAAAA >= \_cCCCCCC)

BODY

GOTO \_aBIGGER;

END

ELSE

BODY

PRINT(\_cCCCCCC);

GOTO \_oUTOFI;

\_aBIGGER:

PRINT(\_aAAAAAA);

GOTO \_oUTOFI;

END

END

IF(\_bBBBBBB <= \_cCCCCCC)

BODY

PRINT(\_cCCCCCC);

END

ELSE

BODY

PRINT(\_bBBBBBB);

END

\_oUTOFI:

PRINT("\n");

IF((\_aAAAAAA EQ \_bBBBBBB) AND (\_aAAAAAA EQ \_cCCCCCC) AND (\_bBBBBBB EQ \_cCCCCCC))

BODY

PRINT(1);

END

ELSE

BODY

PRINT(0);

END

PRINT("\n");

IF((\_aAAAAAA <= 0) OR (\_bBBBBBB <= 0) OR (\_cCCCCCC <= 0))

BODY

PRINT(- 1);

END

ELSE

BODY

PRINT(0);

END

PRINT("\n");

IF(NOT(\_aAAAAAA <= (\_bBBBBBB ADD \_cCCCCCC)))

BODY

PRINT(10);

END

ELSE

BODY

PRINT(0);

END

END

***Результат виконання***

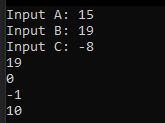


Рис. 4.3 Результат виконання тестової програми №2

* 1. Тестова програма №3

***Текст програми***

!!Prog3

NAME \_pROGRA3;

BODY

DATA INTEGER\_2 \_aAAAAA,\_aAAAAA2,\_bBBBBBB,\_xXXXXXX,\_cCCCCC1,\_cCCCCC2;

PRINT("Input A: ");

SCAN(\_aAAAAA);

PRINT("Input B: ");

SCAN(\_bBBBBBB);

PRINT("FOR TO DO");

FOR \_aAAAAA2<-\_aAAAAA TO \_bBBBBBB DO

BODY

PRINT("\n");

PRINT(\_aAAAAA2 MUL \_aAAAAA2);

END

PRINT("\nFOR DOWNTO DO");

FOR \_aAAAAA2<-\_bBBBBBB DOWNTO \_aAAAAA DO

BODY

PRINT("\n");

PRINT(\_aAAAAA2 MUL \_aAAAAA2);

END

PRINT("\nWHILE A \* B: ");

\_xXXXXXX<-0;

\_cCCCCC1<-0;

WHILE(\_cCCCCC1 <= \_aAAAAA)

BODY

\_cCCCCC2<-0;

WHILE (\_cCCCCC2 <= \_bBBBBBB)

BODY

\_xXXXXXX<-\_xXXXXXX ADD1;

\_cCCCCC2<-\_cCCCCC2 ADD1;

END WHILE

\_cCCCCC1<-\_cCCCCC1 ADD1;

END WHILE

PRINT(\_xXXXXXX);

PRINT("\nREPEAT UNTIL A \* B: ");

\_xXXXXXX<-0;

\_cCCCCC1<-1;

REPEAT

\_cCCCCC2<-1;

REPEAT

\_xXXXXXX<-\_xXXXXXX ADD1;

\_cCCCCC2<-\_cCCCCC2 ADD1;

UNTIL(NOT(\_cCCCCC2 >= \_bBBBBBB))

\_cCCCCC1<-\_cCCCCC1 ADD1;

UNTIL(NOT(\_cCCCCC1 >= \_aAAAAA))

PRINT(\_xXXXXXX);

END

***Результат виконання***

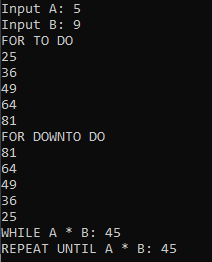


Рис. 4.4 Результат виконання тестової програми №3

ВИСНОВКИ

В процесі виконання курсового проекту було виконано наступне:

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

2. Створено компілятор мови програмування s26, а саме:

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

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

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

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

3.1. На виявлення лексичних помилок.

3.2. На виявлення синтаксичних помилок.

3.3. Загальна перевірка роботи компілятора.

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

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

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

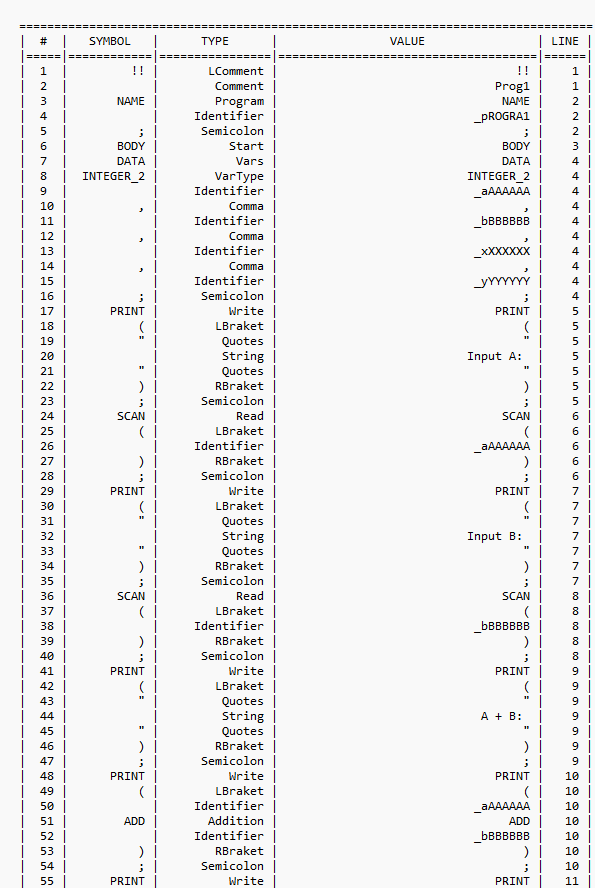
1. Основи проектування трансляторів: Конспект лекцій : [Електронний ресурс] навч. посіб. для студ. спеціальності 123 – «Комп’ютерна інженерія» / О. І. Марченко ; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 108 с.
2. Формальні мови, граматики та автомати: Навчальний посібник / Гавриленко С.Ю. – Харків: НТУ «ХПІ», 2021. – 133 с.
3. Сопронюк Т.М. Системне програмування. Частина І. Елементи теорії формальних мов: Навчальний посібник у двох частинах. – Чернівці: ЧНУ, 2008. – 84 c.
4. Сопронюк Т.М. Системне програмування. Частина ІІ. Елементи теорії компіляції: Навчальний посібник у двох частинах. – Чернівці: ЧНУ, 2008. – 84 c.
5. Alfred V. Aho, Monica S. Lam, Ravi Seth, Jeffrey D. Ullma. Compilers, principles, techniques, and tools, Second Edition, New York, 2007. – 1038 c.
6. Системне програмування (курсовий проект) [Електронний ресурс] –https://vns.lpnu.ua/course/view.php?id=11685.
7. MIT OpenCourseWare. Computer Language Engineering [Електронний ресурс] – Режим доступу до ресурсу: https://ocw.mit.edu/courses/6-035-computer- language-engineering-spring-2010.

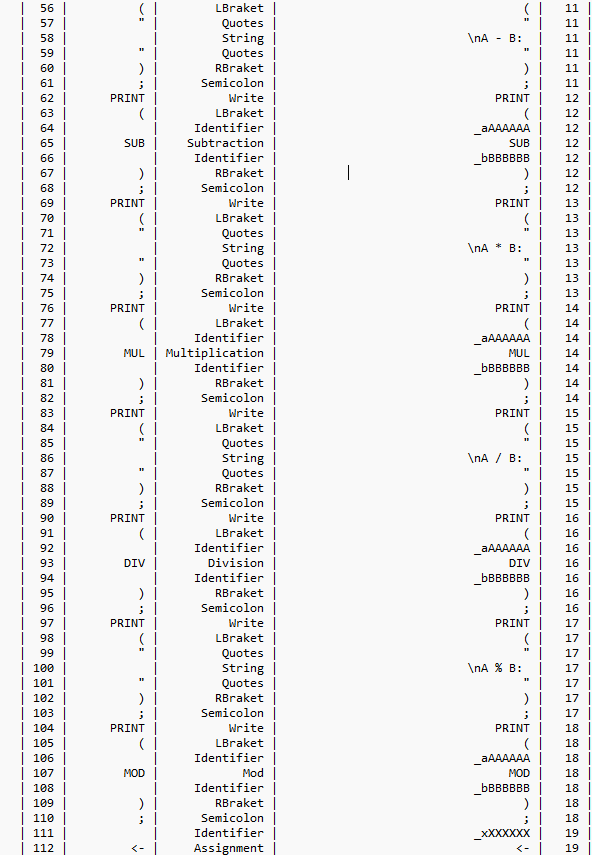
ДОДАТКИ

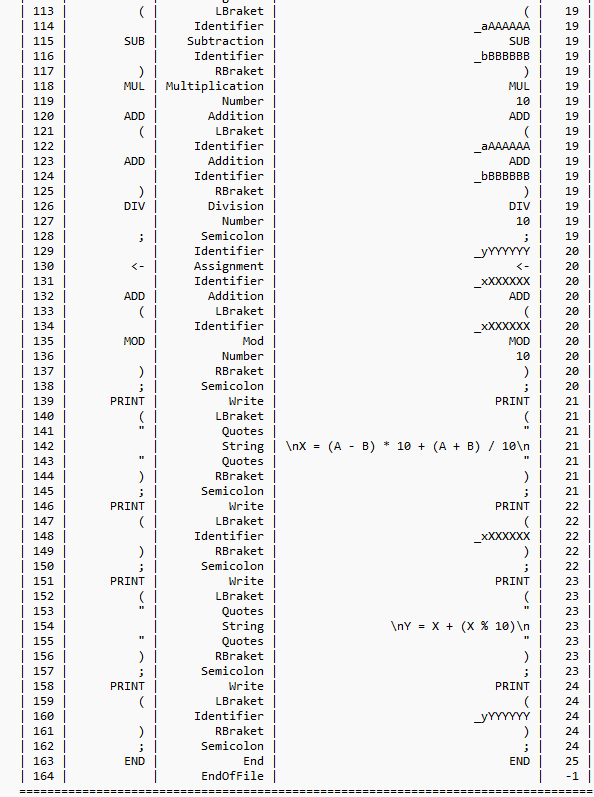
**Додаток А (Таблиці лексем)**

Див. файли «додаток\_А\_ТЛ\_Прог1.pdf», «додаток\_А\_ТЛ\_Прог2.pdf», «додаток\_А\_ТЛ\_Прог3.pdf»

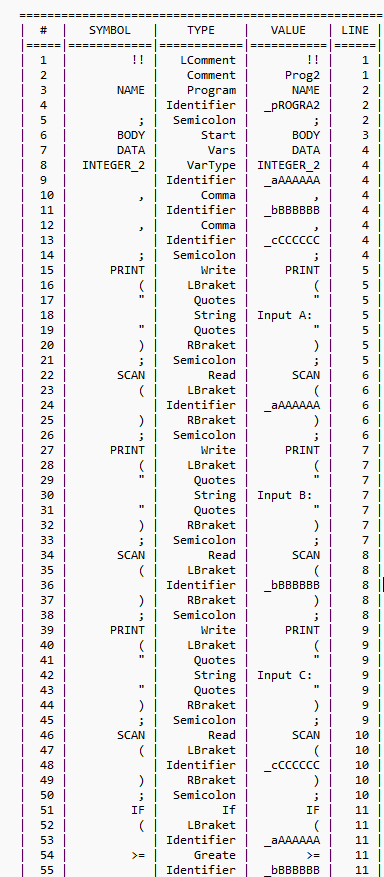
Програма 1

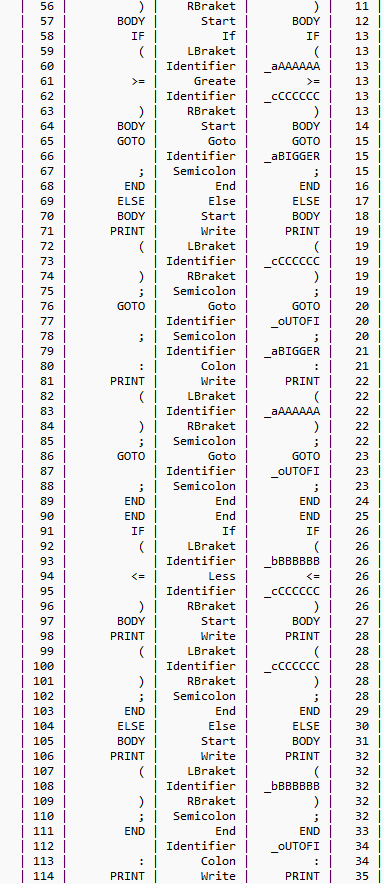


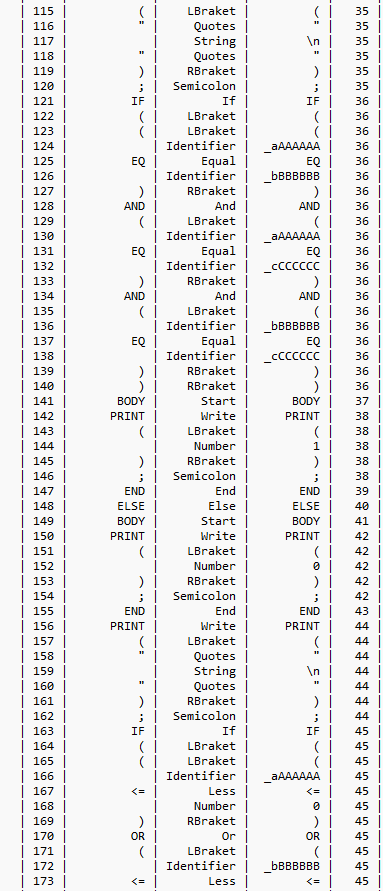


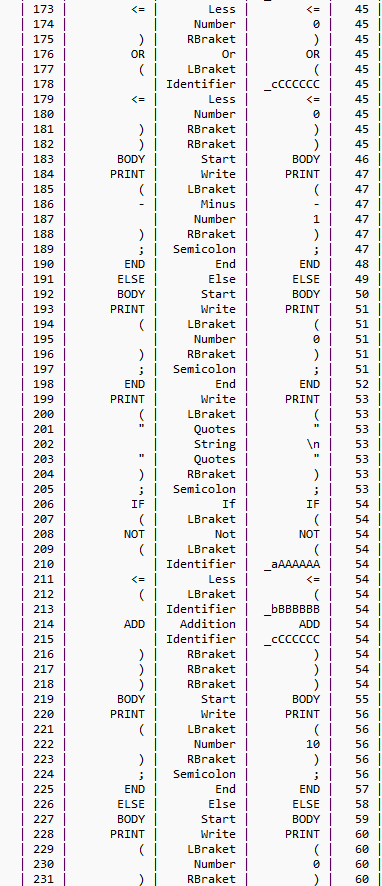


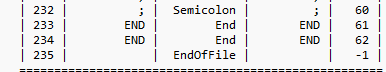
Програма 2



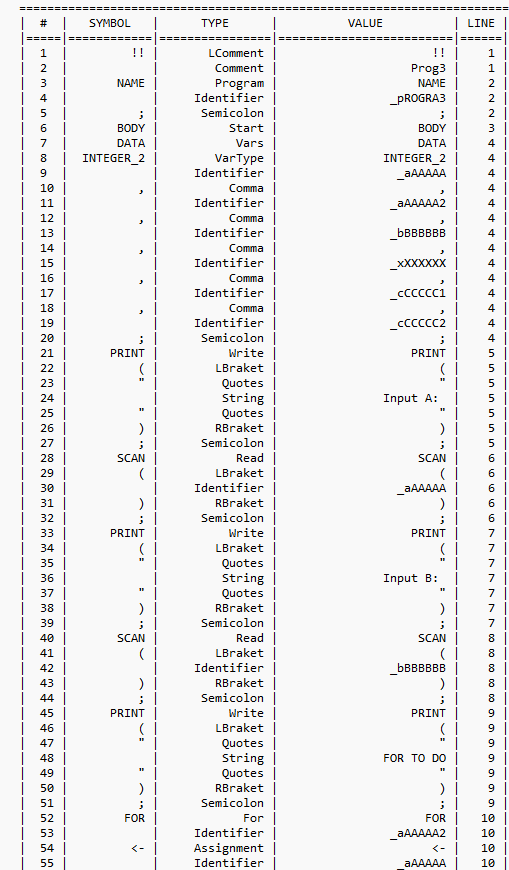


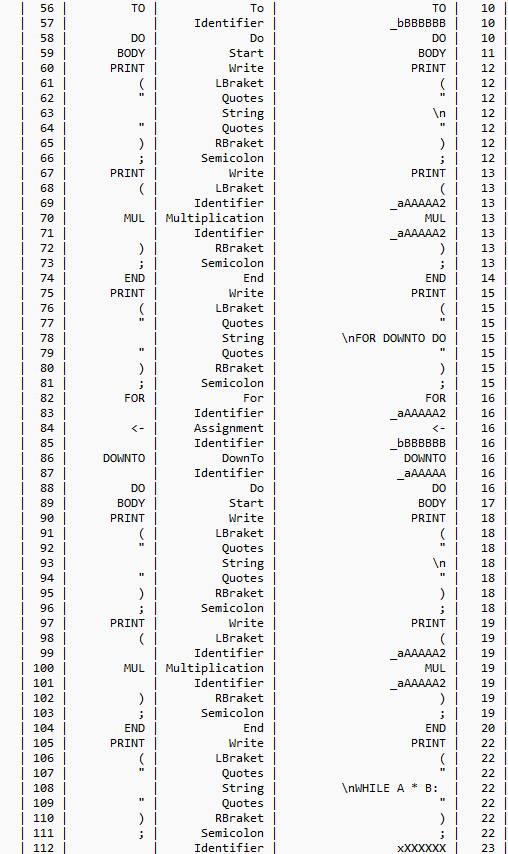


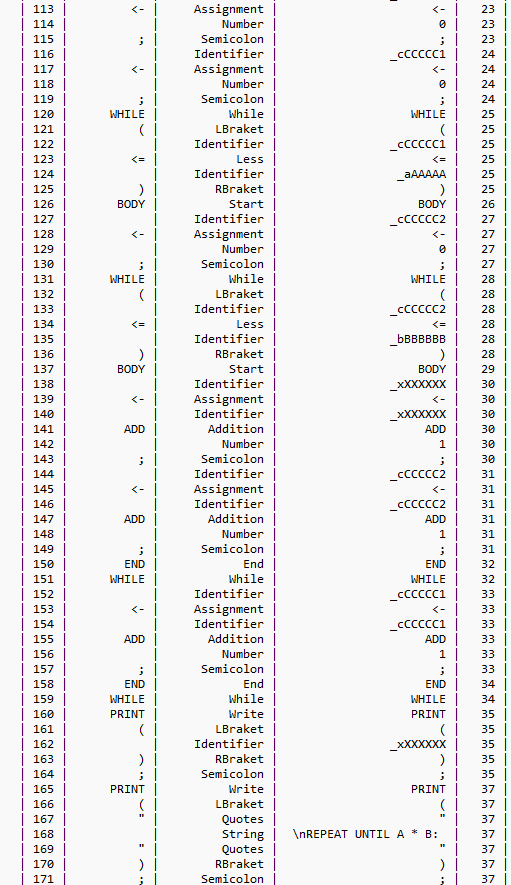


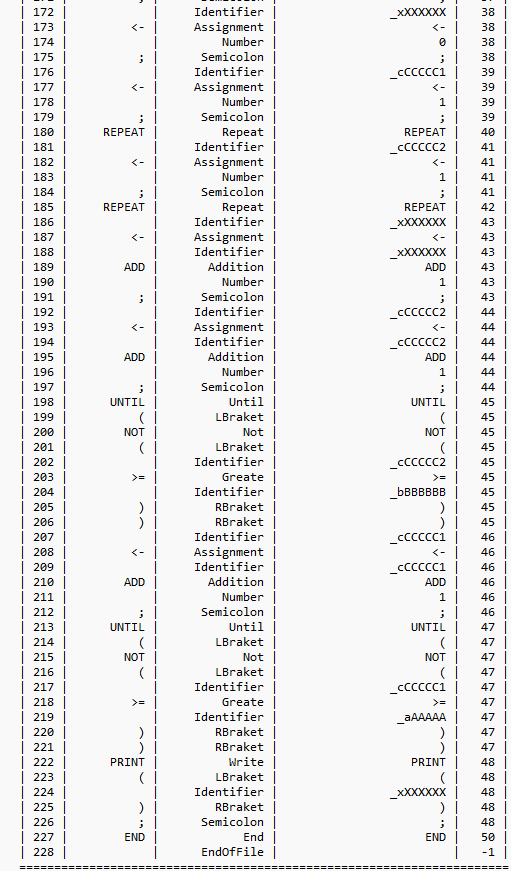


Програма 3







  
  
**Додаток Б (Лістинги основного програмного коду)**

Main.cpp

#include "stdafx.h"

#include "Controller.h"

#include "Core/Parser/TokenRegister.h"

#include "Core/Parser/TokenParser.h"

#include "Core/Generator/Generator.h"

int main(int argc, std::string\* argv)

{

try

{

std::filesystem::path file;

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

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

if (argc != 2)

{

printf("Input file name\n");

std::cin >> file;

}

else

{

file = argv->c\_str();

}

Init();

if (file.extension() != 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 << "DEBUG: 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 << "DEBUG: Breaking into lexems completed. There are " << tokens.size() << " lexems" << std::endl;

std::fstream lexemsFile(lexemsFileName, std::ios::out);

TokenParser::PrintTokens(lexemsFile, tokens);

lexemsFile.close();

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

std::cout << longLine << std::endl;

std::cout << "Error checking are starting... " << std::endl;

std::fstream errorFile(errorFileName, std::ios::out);

auto semanticCheckRes = CheckSemantic(errorFile, tokens);

errorFile.close();

if (semanticCheckRes)

{

std::cout << "There are no errors in the file" << std::endl;

std::cout << longLine << std::endl;

}

else

{

std::cout << "There are errors in the file. Check " << errorFileName << " for more information" << std::endl;

std::cout << longLine << std::endl;

}

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

TokenParser::PrintTokens(tokensFile, tokens);

tokensFile.close();

std::cout << "There are " << tokens.size() << " tokens." << std::endl;

std::cout << "Report file: " << tokensFileName << std::endl;

if (semanticCheckRes)

{

std::cout << longLine << std::endl;

std::cout << "DEBUG: Code generation is starting..." << std::endl;

std::fstream asmFile(asmFileName, std::ios::out);

Generator::Instance()->generateCode(asmFile, tokens);

asmFile.close();

if (std::filesystem::is\_directory("masm32"))

{

std::cout << "DEBUG: Code generation is completed" << std::endl;

std::cout << longLine << std::endl;

system(std::string("masm32\\bin\\ml /c /coff " + fileName + ".asm").c\_str());

system(std::string("masm32\\bin\\Link /SUBSYSTEM:WINDOWS " + fileName + ".obj").c\_str());

}

else

{

std::cout << "WARNING! Can't compile asm file, because masm32 doesn't exist" << std::endl;

}

}

}

catch (const std::exception& ex)

{

std::cout << "Error: " << ex.what() << std::endl;

}

catch (...)

{

std::cout << "Internal error." << std::endl;

}

system("pause");

return 0;

}

BackusRule.h

#pragma once

#include "stdafx.h"

#include "Core/Backus/IBackusRule.h"

#include "BackusRuleItem.h"

class Controller;

class BackusRule : public IBackusRule

{

public:

virtual ~BackusRule() = default;

bool check(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end) final;

std::string type() const final { return m\_name; };

std::string lexeme() const final { return ""; };

void setValue(const std::string& value) final {};

std::string value() const final { return ""; }

int line() const final { return -1; };

std::string customData(const std::string& id) const final { return ""; }

void setCustomData(const std::string& data, const std::string& id) final {};

void setPostHandler(const std::function<void(std::list<std::shared\_ptr<IBackusRule>>::iterator& ruleBegin,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end)>& handler) final

{

m\_handler = handler;

};

private:

friend class Controller;

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

BackusRule(const std::string& name, const std::list<BackusRuleItem>& items) : m\_name(name), m\_backusItem(items) {}

bool oneOrMoreCheck(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end,

const BackusRuleItem& item) const;

bool checkItem(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end,

const BackusRuleItem& item) const;

static bool HasFlag(RuleCountPolicy policy, RuleCountPolicy flag);

private:

std::string m\_name;

std::list<BackusRuleItem> m\_backusItem;

std::function<void(std::list<std::shared\_ptr<IBackusRule>>::iterator& ruleBegin,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end)> m\_handler;

};

BackusRuleBase.h

#pragma once

#include "stdafx.h"

#include "Core/Backus/IBackusRule.h"

template <class T>

class BackusRuleBase : public IBackusRule

{

public:

bool check(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end) final

{

auto res = type() == (\*it)->type();

if (res)

it++;

return res;

}

void setPostHandler(const std::function<void(std::list<std::shared\_ptr<IBackusRule>>::iterator& ruleBegin,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end)>& handler) final { };

};

BackusRuleItem.h

#pragma once

#include "stdafx.h"

#include "Core/Backus/IBackusRule.h"

#include "BackusRuleStorage.h"

#include "Symbols.h"

#include "Utils/magic\_enum.hpp"

class BackusRuleItem

{

public:

BackusRuleItem(const std::vector<std::variant<std::string, Symbols>>& rules, RuleCountPolicy policy) : m\_policy(policy)

{

for (auto rule : rules)

{

if (std::holds\_alternative<std::string>(rule))

m\_ruleNames.push\_back(std::get<std::string>(rule));

else

m\_ruleNames.emplace\_back(magic\_enum::enum\_name(std::get<Symbols>(rule)));

}

}

std::vector<std::shared\_ptr<IBackusRule>> rules() const

{

if (m\_rules.empty())

m\_rules = BackusRuleStorage::Instance()->getRules(m\_ruleNames);

return m\_rules;

};

RuleCountPolicy policy() const { return m\_policy; };

private:

std::vector<std::string> m\_ruleNames;

mutable std::vector<std::shared\_ptr<IBackusRule>> m\_rules;

RuleCountPolicy m\_policy = NoPolicy;

};

BackusRuleStorage.h

#pragma once

#include "stdafx.h"

#include "Utils/singleton.hpp"

#include "Core/Backus/IBackusRule.h"

class BackusRuleStorage : public singleton<BackusRuleStorage>

{

public:

void regRule(std::shared\_ptr<IBackusRule> rule)

{

auto [it, inserted] = m\_rules.try\_emplace(rule->type(), rule);

if (!inserted)

{

throw std::runtime\_error("BackusRuleStorage::regRule: A rule with the type " + rule->type() + " already exists.");

}

}

std::vector<std::shared\_ptr<IBackusRule>> getRules(const std::vector<std::string>& ruleTypes) const

{

std::vector<std::shared\_ptr<IBackusRule>> rules;

for (const auto& ruleType : ruleTypes)

{

auto it = m\_rules.find(ruleType);

if (it == m\_rules.end())

throw std::runtime\_error("BackusRuleStorage::regRule: A rule with the type " + ruleType + " not found.");

rules.push\_back(it->second);

}

return rules;

};

private:

std::map<std::string, std::shared\_ptr<IBackusRule>> m\_rules;

};

IBackusRule.h

#pragma once

#include "stdafx.h"

#include "Core/IItem.h"

enum RuleCountPolicy : std::uint16\_t

{

NoPolicy = 0,

Optional = 1 << 0,

OnlyOne = 1 << 1,

Several = 1 << 2,

OneOrMore = OnlyOne | Several,

PairStart = 1 << 3,

PairEnd = 1 << 4,

};

DEFINE\_ENUM\_FLAG\_OPERATORS(RuleCountPolicy)

\_\_interface IBackusRule : public IItem

{

virtual bool check(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end) = 0;

virtual void setPostHandler(const std::function<void(std::list<std::shared\_ptr<IBackusRule>>::iterator& ruleBegin,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end)>& handler) = 0;

};

BackusRule.cpp

#pragma once

#include "stdafx.h"

#include "BackusRule.h"

std::shared\_ptr<IBackusRule> BackusRule::MakeRule(std::string name, std::list<BackusRuleItem> items)

{

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

return std::make\_shared<EnableMakeShared>(name, items);

}

bool BackusRule::check(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end)

{

bool res = true;

bool pairItem = false;

auto ruleBegin = it;

for (auto item = m\_backusItem.begin(); item != m\_backusItem.end(); ++item)

{

if (it == end || !pairItem && HasFlag(item->policy(), RuleCountPolicy::PairEnd))

{

if (!HasFlag(item->policy(), RuleCountPolicy::Optional) || item != m\_backusItem.end())

{

std::vector<std::string> types;

for (const auto& rule : item->rules())

types.push\_back(rule->type());

errorsInfo.emplace((\*it)->line(), std::make\_pair((\*it)->value(), types));

res = false;

}

break;

}

if (pairItem && HasFlag(item->policy(), RuleCountPolicy::PairEnd) || !HasFlag(item->policy(), RuleCountPolicy::PairEnd))

{

bool resItem = true;

auto startIt = it;

if (HasFlag(item->policy(), RuleCountPolicy::Several))

resItem = oneOrMoreCheck(errorsInfo, it, end, \*item);

else

resItem = checkItem(errorsInfo, it, end, \*item);

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

{

res &= resItem;

break;

}

if (resItem && HasFlag(item->policy(), RuleCountPolicy::PairStart))

{

pairItem = true;

}

if (resItem && pairItem && HasFlag(item->policy(), RuleCountPolicy::PairEnd))

{

pairItem = false;

}

}

}

if (res && m\_handler)

m\_handler(ruleBegin, it, end);

return res;

}

bool BackusRule::oneOrMoreCheck(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end,

const BackusRuleItem& item) const

{

bool res = true;

bool resItem = true;

while (resItem && it != end && HasFlag(item.policy(), RuleCountPolicy::Several))

{

auto startIt = it;

res &= resItem;

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

if (!resItem && startIt != it)

res = false;

}

return res;

}

bool BackusRule::checkItem(std::multimap<int, std::pair<std::string, std::vector<std::string>>>& errorsInfo,

std::list<std::shared\_ptr<IBackusRule>>::iterator& it,

std::list<std::shared\_ptr<IBackusRule>>::iterator& end,

const BackusRuleItem& item) const

{

bool res = false;

std::vector<std::string> types;

auto startIt = it;

auto maxIt = it;

if (it != end)

{

std::multimap<int, std::pair<std::string, std::vector<std::string>>> errors;

for (auto rule : item.rules())

{

types.push\_back(rule->type());

if (!res && startIt == it)

{

res = rule->check(errors, it, end);

}

if (res)

{

break;

}

else if (!res && startIt != it)

{

if(std::distance(maxIt, end) > std::distance(it, end))

maxIt = it;

it = startIt;

errorsInfo.insert(errors.begin(), errors.end());

}

}

}

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

it = maxIt;

if (!res)

errorsInfo.emplace((\*startIt)->line(), std::make\_pair((\*it)->value(), types));

else

errorsInfo.clear();

return res;

}

bool BackusRule::HasFlag(RuleCountPolicy policy, RuleCountPolicy flag)

{

return (policy & flag) == flag;

}

Generator.h

#pragma once

#include "stdafx.h"

#include "Utils/singleton.hpp"

#include "Core/Generator/GeneratorItemBase.h"

class Generator : public singleton<Generator>

{

public:

template<class T>

void generateCode(std::ostream& out, std::list<std::shared\_ptr<T>>& items) const

{

if (!m\_details)

throw std::runtime\_error("Generator details is not set");

std::list<std::shared\_ptr<IGeneratorItem>> generatorItems;

for (auto item : items)

{

generatorItems.push\_back(std::dynamic\_pointer\_cast<IGeneratorItem>(item));

}

auto it = generatorItems.begin();

auto end = generatorItems.end();

std::stringstream code;

genCode(code, \*m\_details, it, end);

PrintBegin(out, \*m\_details);

PrintData(out, \*m\_details);

PrintBeginCodeSegment(out, \*m\_details);

out << code.str();

PrintEnding(out, \*m\_details);

}

void setDetails(const GeneratorDetails& details) { m\_details = std::make\_shared<GeneratorDetails>(details); }

protected:

Generator() = default;

private:

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const;

private:

static void PrintBegin(std::ostream& out, GeneratorDetails& details);

static void PrintData(std::ostream& out, GeneratorDetails& details);

static void PrintBeginCodeSegment(std::ostream& out, GeneratorDetails& details);

static void PrintEnding(std::ostream& out, GeneratorDetails& details);

private:

std::shared\_ptr<GeneratorDetails> m\_details;

};

GeneratorDetails.h

#pragma once

#include "stdafx.h"

class GeneratorDetails

{

friend class Generator;

public:

struct GeneratorArgs

{

std::string regPrefix;

std::string numberType;

std::string numberTypeExtended;

size\_t argSize;

size\_t posArg0;

size\_t posArg1;

std::string numberStrType;

};

public:

explicit GeneratorDetails(const GeneratorArgs& args) : m\_args(args)

{

m\_args.posArg0 = m\_kRetAddrSize + m\_args.argSize;

m\_args.posArg1 = m\_kRetAddrSize;

}

const GeneratorArgs& args() const { return m\_args; }

void registerNumberData(const std::string& name)

{

throwIfDataExists(name);

m\_userNumberData[name] = '\t' + name + '\t' + m\_args.numberType + '\t' + "0";

}

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

{

throwIfDataExists(name);

std::string item;

size\_t start = 0;

size\_t end;

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

m\_userStringData[name] = '\t' + name + "\tdb\t";

while ((end = data.find(delimiter, start)) != std::string::npos)

{

item = data.substr(start, end - start);

if (!item.empty())

m\_userStringData[name] += "\"" + item + "\", ";

m\_userStringData[name] += "13, 10, ";

start = end + delimiter.length();

}

item = data.substr(start);

if (!item.empty())

m\_userStringData[name] += "\"" + item + "\", ";

m\_userStringData[name] += "0";

}

void registerRawData(const std::string& name, const std::string& rawData)

{

throwIfDataExists(name);

m\_userRawData[name] = '\t' + name + '\t' + rawData;

}

void registerProc(const std::string& type, const std::function<void(std::ostream& out, const GeneratorArgs&)>& generator)

{

if (!m\_procGenerators.contains(type))

m\_procGenerators[type] = generator;

else

throw std::runtime\_error("Proc for type " + type + " already exists");

}

private:

void throwIfDataExists(const std::string& name) const

{

if (m\_userNumberData.contains(name) || m\_userStringData.contains(name) || m\_userRawData.contains(name))

throw std::runtime\_error("Data with name " + name + " already exists");

}

private:

GeneratorArgs m\_args;

std::map<std::string, std::string> m\_userNumberData;

std::map<std::string, std::string> m\_userStringData;

std::map<std::string, std::string> m\_userRawData;

std::map<std::string, std::function<void(std::ostream& out, const GeneratorArgs&)>> m\_procGenerators;

static constexpr size\_t m\_kRetAddrSize = 4;

};

GeneratorItemBase.h

#pragma once

#include "stdafx.h"

#include "Core/Generator/GeneratorUtils.h"

template <class T>

class GeneratorItemBase : public IGeneratorItem

{

public:

virtual ~GeneratorItemBase() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const override {};

protected:

std::string customData\_imp(const std::string& id) const { return m\_customData[id]; }

void setCustomData\_imp(const std::string& data, const std::string& id) { m\_customData[id] = data; }

static bool IsRegistered() { return registered; }

static void SetRegistered() { registered = true; }

static bool registered;

private:

mutable std::map<std::string, std::string> m\_customData{ {"default",""} };

};

template<class T>

bool GeneratorItemBase<T>::registered = false;

GeneratorUtils.h

#pragma once

#include "stdafx.h"

#include "Utils/singleton.hpp"

#include "Core/Generator/IGeneratorItem.h"

class GeneratorUtils : public singleton<GeneratorUtils>

{

public:

void RegisterOperation(const std::string& type, size\_t priority)

{

m\_operations[type] = priority;

}

void RegisterOperand(const std::string& type)

{

m\_operands.insert(type);

}

void RegisterEquationEnd(const std::string& type)

{

m\_equationEnd.insert(type);

}

void 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\_lBraketType;

std::string m\_rBraketType;

};

IGeneratorItem.h

#pragma once

#include "stdafx.h"

#include "Utils/singleton.hpp"

#include "Core/Generator/IGeneratorItem.h"

class GeneratorUtils : public singleton<GeneratorUtils>

{

public:

void RegisterOperation(const std::string& type, size\_t priority)

{

m\_operations[type] = priority;

}

void RegisterOperand(const std::string& type)

{

m\_operands.insert(type);

}

void RegisterEquationEnd(const std::string& type)

{

m\_equationEnd.insert(type);

}

void RegisterLBraket(const std::string& type)

{

m\_lBraketType = type;

}

void RegisterRBraket(const std::string& type)

{

m\_rBraketType = type;

}

std::list<std::shared\_ptr<IGeneratorItem>> ConvertToPostfixForm(

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const

{

std::list<std::shared\_ptr<IGeneratorItem>> postfixForm;

std::list<std::shared\_ptr<IGeneratorItem>> stack;

while (it != end)

{

auto item = \*it;

auto itemType = item->type();

if (IsOperand(item))

{

postfixForm.push\_back(item);

}

else if (IsOperation(item))

{

while (!stack.empty() && !Prioritet(item, stack.back()) && stack.back()->type() != m\_lBraketType)

{

postfixForm.push\_back(stack.back());

stack.pop\_back();

}

stack.push\_back(item);

}

else if (itemType == m\_lBraketType)

{

stack.push\_back(item);

postfixForm.push\_back(item);

}

else if (itemType == m\_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\_lBraketType;

std::string m\_rBraketType;

};

Generator.cpp

#include "stdafx.h"

#include "Generator.h"

void Generator::PrintBegin(std::ostream& out, GeneratorDetails& details)

{

out << ".386\n";

out << ".model flat, stdcall\n";

out << "option casemap :none\n";

out << std::endl;

out << "include masm32\\include\\windows.inc\n";

out << "include masm32\\include\\kernel32.inc\n";

out << "include masm32\\include\\masm32.inc\n";

out << "include masm32\\include\\user32.inc\n";

out << "include masm32\\include\\msvcrt.inc\n";

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

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

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

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

}

void Generator::PrintData(std::ostream& out, GeneratorDetails& details)

{

out << std::endl;

out << ".DATA\n";

out << ";===User Data================================================================================\n";

for (const auto& [\_, data] : details.m\_userNumberData)

{

out << data << std::endl;

}

if (!details.m\_userNumberData.empty())

out << std::endl;

for (const auto& [\_, data] : details.m\_userStringData)

{

out << data << std::endl;

}

if (!details.m\_userStringData.empty())

out << std::endl;

out << ";===Addition Data============================================================================\n";

out << "\thConsoleInput\tdd\t?\n";

out << "\thConsoleOutput\tdd\t?\n";

out << "\tendBuff\t\t\tdb\t5 dup (?)\n";

out << "\tmsg1310\t\t\tdb\t13, 10, 0\n";

if (!details.m\_userRawData.empty())

out << std::endl;

for (const auto& [\_, data] : details.m\_userRawData)

{

out << data << std::endl;

}

}

void Generator::PrintBeginCodeSegment(std::ostream& out, GeneratorDetails& details)

{

out << std::endl;

out << ".CODE\n";

out << "start:\n";

out << "invoke AllocConsole\n";

out << "invoke GetStdHandle, STD\_INPUT\_HANDLE\n";

out << "mov hConsoleInput, eax\n";

out << "invoke GetStdHandle, STD\_OUTPUT\_HANDLE\n";

out << "mov hConsoleOutput, eax\n";

}

void Generator::PrintEnding(std::ostream& out, GeneratorDetails& details)

{

out << "exit\_label:\n";

out << "invoke WriteConsoleA, hConsoleOutput, ADDR msg1310, SIZEOF msg1310 - 1, 0, 0\n";

out << "invoke ReadConsoleA, hConsoleInput, ADDR endBuff, 5, 0, 0\n";

out << "invoke ExitProcess, 0\n";

for (const auto& [\_, proc] : details.m\_procGenerators)

{

out << std::endl << std::endl;

proc(out, details.args());

}

out << "end start\n";

}

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const

{

for (; it != end; ++it)

{

(\*it)->genCode(out, details, it, end);

}

}

TokenParser.h

#pragma once

#include "stdafx.h"

#include "Utils/singleton.hpp"

#include "Core/Tokens/IToken.hpp"

#include "Utils/TablePrinter.h"

class TokenParser : public singleton<TokenParser>

{

public:

static constexpr int NoPriority = std::numeric\_limits<int>::min();

public:

std::list<std::shared\_ptr<IToken>> tokenize(std::istream& input);

void regToken(std::shared\_ptr<IToken> token, int priority = NoPriority);

void regUnchangedTextToken(std::shared\_ptr<IToken> target, std::shared\_ptr<IToken> lBorder, std::shared\_ptr<IToken> rBorder);

template<class T>

static void PrintTokens(std::ostream& out, const std::list<std::shared\_ptr<T>>& tokens)

{

auto getNumCount = [](int k) { return std::to\_string(k).size(); };

size\_t maxLemexeLen = 0;

size\_t maxTypeLen = 0;

size\_t maxValueLen = 0;

for (auto token : tokens)

{

maxLemexeLen = std::max(maxLemexeLen, token->lexeme().size());

maxTypeLen = std::max(maxTypeLen, token->type().size());

maxValueLen = std::max(maxValueLen, token->value().size());

}

const std::string kHeaderColumn0 = "#";

const std::string kHeaderColumn1 = "SYMBOL";

const std::string kHeaderColumn2 = "TYPE";

const std::string kHeaderColumn3 = "VALUE";

const std::string kHeaderColumn4 = "LINE";

size\_t colPadding = 1;

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

auto widthColumn1 = std::max(kHeaderColumn1.size(), maxLemexeLen) + 2 \* colPadding;

auto widthColumn2 = std::max(kHeaderColumn2.size(), maxTypeLen) + 2 \* colPadding;

auto widthColumn3 = std::max(kHeaderColumn3.size(), maxValueLen) + 2 \* colPadding;

auto widthColumn4 = std::max(kHeaderColumn4.size(), getNumCount(tokens.back()->line())) + 2 \* colPadding;

if ((kHeaderColumn0.size() % 2) != (widthColumn0 % 2)) widthColumn0++;

if ((kHeaderColumn1.size() % 2) != (widthColumn1 % 2)) widthColumn1++;

if ((kHeaderColumn2.size() % 2) != (widthColumn2 % 2)) widthColumn2++;

if ((kHeaderColumn3.size() % 2) != (widthColumn3 % 2)) widthColumn3++;

if ((kHeaderColumn4.size() % 2) != (widthColumn4 % 2)) widthColumn4++;

size\_t index = 1;

auto getIndex = [&index](const std::shared\_ptr<T>&) { return std::to\_string(index++); };

auto getLemexe = [](const std::shared\_ptr<T>& token) { return token->lexeme(); };

auto getType = [](const std::shared\_ptr<T>& token) { return token->type(); };

auto getValue = [](const std::shared\_ptr<T>& token) { return token->value(); };

auto getLine = [](const std::shared\_ptr<T>& token) { return std::to\_string(token->line()); };

TablePrinter::PrintTable(out,

{ kHeaderColumn0, kHeaderColumn1, kHeaderColumn2, kHeaderColumn3, kHeaderColumn4 },

{ widthColumn0, widthColumn1, widthColumn2, widthColumn3, widthColumn4 },

{ TablePrinter::CENTRE, TablePrinter::RIGHT, TablePrinter::RIGHT , TablePrinter::RIGHT , TablePrinter::RIGHT },

tokens,

{ getIndex, getLemexe, getType, getValue, getLine },

colPadding);

}

private:

void throwIfTokenRegistered(std::shared\_ptr<IToken> token);

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

bool isUnchangedTextTokenLast();

private:

static bool IsNewLine(const char& ch);

static bool IsTabulation(const char& ch);

static bool IsAllowedSymbol(const char& ch);

static bool IsAllowedSpecialSymbol(const char& ch);

private:

struct PriorityCompare

{

bool operator()(const int& a, const int& b) const

{

return a > b;

}

};

private:

std::multimap<int, std::shared\_ptr<IToken>, PriorityCompare> m\_priorityTokens;

std::map<std::string, std::tuple<std::shared\_ptr<IToken>, std::shared\_ptr<IToken>, std::shared\_ptr<IToken>>> m\_unchangedTextTokens;

std::list<std::shared\_ptr<IToken>> m\_tokens;

std::function<std::shared\_ptr<IToken>(std::string)> m\_getTokenByType = [this](const std::string& type) {

auto start = m\_priorityTokens.lower\_bound(static\_cast<int>(type.size()));

auto mapItem = std::find\_if(start, m\_priorityTokens.end(), [&type](const auto& pair) { return pair.second->type() == type; });

if (mapItem == m\_priorityTokens.end())

throw std::runtime\_error("TokenParser::getTokenByType: Token with type " + type + " not found");

return mapItem->second;

};

};

TokenParser.cpp

#include "stdafx.h"

#include "Core/Parser/TokenParser.h"

#include "Utils/StringUtils.h"

#include "Tokens/Common/EndOfFile.h"

std::list<std::shared\_ptr<IToken>> TokenParser::tokenize(std::istream& input)

{

m\_tokens.clear();

int curLine = 1;

std::string token;

for (char ch; input.get(ch);)

{

if (!token.empty() && ((IsAllowedSymbol(token.front()) != IsAllowedSymbol(ch)) || IsTabulation(ch)))

recognizeToken(token, curLine);

if (IsNewLine(ch))

++curLine;

if (isUnchangedTextTokenLast())

{

std::string unchangedTextTokenValue{ token };

token.clear();

int unchangedTextTokenLine{ curLine };

const auto& [target, left, right] = m\_unchangedTextTokens[m\_tokens.back()->lexeme()];

auto rBorderLex = right ? right->lexeme() : "\n";

do

{

if (IsNewLine(ch))

++curLine;

unchangedTextTokenValue += ch;

}

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

unchangedTextTokenValue = unchangedTextTokenValue.substr(0, unchangedTextTokenValue.size() - rBorderLex.size());

m\_tokens.push\_back(target->tryCreateToken(unchangedTextTokenValue));

m\_tokens.back()->setLine(unchangedTextTokenLine);

if (right)

{

m\_tokens.push\_back(right->tryCreateToken(rBorderLex));

m\_tokens.back()->setLine(curLine);

}

continue;

}

if (!IsTabulation(ch))

token += ch;

}

if (!token.empty())

recognizeToken(token, curLine);

m\_tokens.push\_back(std::make\_shared<EndOfFile>());

return m\_tokens;

}

void TokenParser::regToken(std::shared\_ptr<IToken> token, int priority)

{

throwIfTokenRegistered(token);

if (priority == NoPriority)

priority = static\_cast<int>(token->lexeme().size());

m\_priorityTokens.insert(std::make\_pair(priority, token));

}

void TokenParser::regUnchangedTextToken(std::shared\_ptr<IToken> target, std::shared\_ptr<IToken> lBorder, std::shared\_ptr<IToken> rBorder)

{

if(rBorder)

throwIfTokenRegistered(rBorder);

regToken(lBorder);

throwIfTokenRegistered(target);

m\_unchangedTextTokens.try\_emplace(lBorder->lexeme(), target, lBorder, rBorder);

}

void TokenParser::throwIfTokenRegistered(std::shared\_ptr<IToken> token)

{

auto start = m\_priorityTokens.lower\_bound(static\_cast<int>(token->lexeme().size()));

auto priorToken = std::find\_if(start, m\_priorityTokens.end(),

[&token](const auto& pair) {

return token->type() == pair.second->type();

});

auto unchTextToken = std::ranges::find\_if(m\_unchangedTextTokens,

[&token](const auto& pair) {

auto type = token->type();

const auto& [main, left, right] = pair.second;

return type == main->type() ||

type == left->type() ||

right && type == right->type();

});

if(priorToken != m\_priorityTokens.end() || unchTextToken != m\_unchangedTextTokens.end())

throw std::runtime\_error("TokenParser: Token with type " + token->type() + " already registered");

}

void TokenParser::recognizeToken(std::string& token, int curLine)

{

if(m\_priorityTokens.empty())

throw std::runtime\_error("TokenParser: No tokens registered");

auto start = m\_priorityTokens.lower\_bound(static\_cast<int>(token.size()));

for (auto it = start; it != m\_priorityTokens.end(); ++it)

{

auto curRegToken = it->second;

if (auto newToken = curRegToken->tryCreateToken(token); newToken)

{

m\_tokens.push\_back(newToken);

m\_tokens.back()->setLine(curLine);

break;

}

}

if (!token.empty() && !isUnchangedTextTokenLast())

recognizeToken(token, curLine);

}

bool TokenParser::isUnchangedTextTokenLast()

{

if (!m\_tokens.empty() && m\_unchangedTextTokens.contains(m\_tokens.back()->lexeme()))

{

auto const& [target, left, right] = m\_unchangedTextTokens[m\_tokens.back()->lexeme()];

if (m\_tokens.size() >= 2)

{

if (target->type() != (\*(++m\_tokens.rbegin()))->type())

return true;

}

else

return true;

}

return false;

}

bool TokenParser::IsNewLine(const char& ch)

{

return ch == '\n';

}

bool TokenParser::IsTabulation(const char& ch)

{

return ch == ' ' || ch == '\t' || IsNewLine(ch);

}

bool TokenParser::IsAllowedSymbol(const char& ch)

{

return !!isalpha(ch) || !!isdigit(ch) || IsAllowedSpecialSymbol(ch);

}

bool TokenParser::IsAllowedSpecialSymbol(const char& ch)

{

std::set<char> allowedSymblos{ '\_' };

return allowedSymblos.contains(ch);

}

TokenRegister.h

#pragma once

#include "stdafx.h"

#include "Controller.h"

#include "Rules/IdentRule/Undefined.h"

#include "Tokens/Common/Unknown.h"

void Init();

template <typename T>

bool CheckSemantic(std::ostream& out, std::list<std::shared\_ptr<T>>& tokens)

{

auto endOfFileType = tokens.back()->type();

std::list<std::shared\_ptr<IBackusRule>> rules;

for (auto token : tokens)

{

if (auto rule = std::dynamic\_pointer\_cast<IBackusRule>(token))

rules.push\_back(rule);

}

auto it = rules.begin();

auto end = rules.end();

std::multimap<int, std::pair<std::string, std::vector<std::string>>> errors;

auto res = Controller::Instance()->topRule()->check(errors, it, end);

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

end = --rules.end();

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

int lexErr = 0;

int synErr = 0;

int semErr = 0;

tokens.clear();

for (auto rule : rules)

{

tokens.push\_back(std::dynamic\_pointer\_cast<T>(rule));

if (rule->type() == Undefined::Type())

{

res = false;

std::string err;

if (auto erMsg = rule->customData("error"); !erMsg.empty())

{

semErr++;

err = "Semantic error: " + erMsg;

}

else

{

semErr++;

err = std::format("Semantic error: Undefined token: {}", rule->value());

}

errorsMsg.emplace(rule->line(), err);

}

else if (rule->type() == token::Unknown::Type())

{

lexErr++;

res = false;

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

}

}

for (auto it = errors.rbegin(); it != errors.rend(); ++it)

{

auto types = it->second.second;

std::stringstream ss;

for (size\_t i = 0; i < types.size(); ++i)

{

if (!types[i].empty())

{

ss << types[i];

if (i != types.size() - 1)

ss << " or ";

}

}

auto ssStr = ss.str();

if (!ssStr.empty())

{

synErr++;

std::string msg = "Syntax error: Expected: " + ssStr;

if (!it->second.first.empty())

msg += " before " + it->second.first;

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

}

}

out << "List of errors" << std::endl;

out << "======================================================================" << std::endl;

out << "There are " << lexErr << " lexical errors." << std::endl;

out << "There are " << synErr << " syntax errors." << std::endl;

out << "There are " << semErr << " semantic errors." << std::endl << std::endl;

for (auto const& [line, msg] : errorsMsg)

{

out << "Line " << line << ": " << msg << std::endl;

}

return res;

}

TokenRegister.cpp

#include "stdafx.h"

#include "Core/Parser/TokenRegister.h"

#include "Controller.h"

#include "Tokens/Common.h"

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

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

#include "Rules/Operators/For/ForRule.h"

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

#include "Rules/Operators/RepeatUntil/RepeatUntilRule.h"

void Init()

{

Controller::Instance()->regOperatorRule(MakeIf);

Controller::Instance()->regOperatorRule(MakeGoto, true);

Controller::Instance()->regOperatorRule(MakeLabel);

Controller::Instance()->regOperatorRule(MakeFor);

Controller::Instance()->regOperatorRule(MakeWhile);

Controller::Instance()->regOperatorRule(MakeRepeatUntil);

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

Controller::Instance()->regUnchangedTextToken(std::make\_shared<Comment>(), std::make\_shared<LComment>(), nullptr);

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

}

TokenOperators:

Loops:

Do.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Do() { setLexeme("DO"); };

virtual ~Do() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

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

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

out << "\tje " << customData("endLabel") << std::endl;

};

};

DownTo.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

#include "Rules/EquationRule/Greate.h"

#include "Rules/EquationRule/Not.h"

#include "Rules/EquationRule/Subtraction.h"

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

{

BASE\_ITEM

public:

DownTo() { setLexeme("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.h

#pragma once

#include "stdafx.h"

#include "Controller.h"

BackusRulePtr MakeFor(std::shared\_ptr<Controller> controller);

ForRule.cpp

#include "stdafx.h"

#include "ForRule.h"

#include "Rules/Operators/For/For.h"

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

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

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

BackusRulePtr MakeFor(std::shared\_ptr<Controller> controller)

{

using enum ItemType;

controller->regItem<For>();

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

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

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

auto context = controller->context();

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

auto forToOrDownToDoRule = controller->addRule("ForToOrDownToDoRule", {

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

BackusRuleItem({ "AssignmentRule"}, OnlyOne),

BackusRuleItem({ To::Type(), DownTo::Type()}, OnlyOne),

BackusRuleItem({ context->EquationRuleName()}, OnlyOne),

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

BackusRuleItem({ lCodeBlok}, OnlyOne)

});

forToOrDownToDoRule->setPostHandler([context](BackusRuleList::iterator& ruleBegin,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

static size\_t index = 0;

index++;

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

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

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

bool increment = false;

for (auto itr = ruleBegin; itr != it; ++itr)

{

auto type = (\*itr)->type();

if ((type == To::Type() || type == DownTo::Type()))

{

if (type == To::Type())

increment = true;

(\*itr)->setCustomData(startLabel, "startLabel");

(\*itr)->setCustomData(ident->customData(), "ident");

}

else if (type == Do::Type())

{

(\*itr)->setCustomData(endLabel, "endLabel");

break;

}

}

std::string code;

code += std::format("\tpush {}\n", ident->customData());

code += std::format("\tpush {} ptr 1\n", context->Details().args().numberTypeExtended);

code += std::format("\tcall {}\n", increment ? "Add\_" : "Sub\_");

code += std::format("\tpop {}\n", ident->customData());

code += std::format("\tjmp {}\n", startLabel);

code += std::format("{}:", endLabel);

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

});

return forToOrDownToDoRule;

}

To.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

#include "Rules/EquationRule/Less.h"

#include "Rules/EquationRule/Not.h"

#include "Rules/EquationRule/Addition.h"

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

{

BASE\_ITEM

public:

To() { setLexeme("TO"); };

virtual ~To() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

Less::RegPROC(details);

Not::RegPROC(details);

Addition::RegPROC(details);

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

it++;

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

auto postIt = postForm.begin();

auto postEnd = postForm.end();

for (const auto& item : postForm)

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

out << "\tpush " << customData("ident") << std::endl;

out << "\tcall Less\_" << std::endl;

out << "\tcall Not\_" << std::endl;

};

};

Goto:

Goto.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Goto() { setLexeme("GOTO"); };

virtual ~Goto() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

it++;

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

};

};

GotoRule.h

#pragma once

#include "stdafx.h"

#include "Controller.h"

BackusRulePtr MakeGoto(std::shared\_ptr<Controller> controller);

BackusRulePtr MakeLabel(std::shared\_ptr<Controller> controller);

GotoRule.cpp

#include "stdafx.h"

#include "GotoRule.h"

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

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

#include "Rules/IdentRule/Identifier.h"

#include "Rules/IdentRule/Undefined.h"

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

BackusRulePtr MakeLabel(std::shared\_ptr<Controller> controller)

{

using enum ItemType;

controller->regItem<Label>(Rule);

auto context = controller->context();

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

auto labelRule = controller->addRule("LabelRule", {

BackusRuleItem({ context->IdentRuleName()}, OnlyOne),

BackusRuleItem({ Symbols::Colon}, OnlyOne)

});

labelRule->setPostHandler([context](BackusRuleList::iterator&,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

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

auto identIt = it;

auto identVal = (\*identIt)->value();

std::shared\_ptr<IToken> label;

if (context->IdentTable().contains((\*identIt)->value()))

{

label = std::make\_shared<Undefined>();

label->setCustomData("Redefinition", "error");

}

else

label = std::make\_shared<Label>();

label->setValue((\*identIt)->value() + (\*(++it))->value());

end = std::remove(it, end, \*it);

label->setLine((\*identIt)->line());

label->setCustomData((\*identIt)->customData());

\*identIt = std::dynamic\_pointer\_cast<IBackusRule>(label);

if (!labelTable.contains(identVal))

{

labelTable.try\_emplace(identVal, std::optional<BackusRuleList::iterator>());

}

else

{

if (auto optIt = labelTable[identVal]; optIt.has\_value())

{

auto gotoIdentIt = optIt.value();

if ((\*gotoIdentIt)->type() == Undefined::Type())

{

auto labelName = std::make\_shared<Identifier>();

labelName->setValue((\*gotoIdentIt)->value());

labelName->setLine((\*gotoIdentIt)->line());

labelName->setCustomData((\*gotoIdentIt)->customData());

\*gotoIdentIt = labelName;

}

}

}

});

return labelRule;

}

BackusRulePtr MakeGoto(std::shared\_ptr<Controller> controller)

{

controller->regItem<Goto>();

auto context = controller->context();

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

auto gotoStatement = controller->addRule("GotoStatement", {

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

BackusRuleItem({context->IdentRuleName()}, OnlyOne)

});

gotoStatement->setPostHandler([](BackusRuleList::iterator&,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

it = std::prev(it, 1);

auto identIt = it;

if (!labelTable.contains((\*identIt)->value()))

{

if ((\*identIt)->type() != Undefined::Type())

{

auto undef = std::make\_shared<Undefined>();

undef->setValue((\*identIt)->value());

undef->setLine((\*identIt)->line());

undef->setCustomData((\*identIt)->customData());

\*identIt = undef;

}

labelTable.try\_emplace((\*identIt)->value(), identIt);

}

else

{

auto ident = std::make\_shared<Identifier>();

ident->setValue((\*identIt)->value());

ident->setLine((\*identIt)->line());

ident->setCustomData((\*identIt)->customData());

\*identIt = ident;

}

it = std::next(it);

});

return gotoStatement;

}

Label.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Label() { setLexeme(""); };

virtual ~Label() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

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

};

};

IfElse:

Else.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Else() { setLexeme("ELSE"); };

virtual ~Else() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

out << "\tjmp " << customData("endLabel") << std::endl;

out << customData("elseLabel") << ":\n";

};

};

If.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

If() { setLexeme("IF"); };

virtual ~If() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

it++;

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

auto postIt = postForm.begin();

auto postEnd = postForm.end();

for (const auto& item : postForm)

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

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

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

out << "\tje " << customData("label") << std::endl;

};

};

IfRule.h

#pragma once

#include "stdafx.h"

#include "Controller.h"

BackusRulePtr MakeIf(std::shared\_ptr<Controller> controller);

IfRule.cpp

#include "stdafx.h"

#include "IfRule.h"

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

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

BackusRulePtr MakeIf(std::shared\_ptr<Controller> controller)

{

controller->regItem<If>();

controller->regItem<Else>();

auto context = controller->context();

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

auto elseStatement = controller->addRule("ElseStatement", {

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

BackusRuleItem({ lCodeBlok}, OnlyOne),

});

auto ifStatement = controller->addRule("IfStatement", {

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

BackusRuleItem({ Symbols::LBraket}, OnlyOne),

BackusRuleItem({ context->EquationRuleName()}, OnlyOne),

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

BackusRuleItem({ lCodeBlok}, OnlyOne),

BackusRuleItem({ elseStatement->type()}, Optional)

});

ifStatement->setPostHandler([](BackusRuleList::iterator& ruleBegin,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

static size\_t index = 0;

index++;

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

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

bool hasElse = false;

size\_t count = 0;

for (auto itr = ruleBegin; itr != it; ++itr)

{

auto type = (\*itr)->type();

if (type == lStart)

{

count++;

}

else if (type == Else::Type() && count == 0)

{

(\*itr)->setCustomData(elseLabel, "elseLabel");

(\*itr)->setCustomData(endLabel, "endLabel");

hasElse = true;

}

else if (type == lEnd && count == 1 && (\*std::next(itr))->type() != Else::Type())

{

(\*itr)->setCustomData(endLabel + ':');

break;

}

else if (type == lEnd && count > 0)

{

count--;

}

}

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

});

return ifStatement;

}

RepeatUntil:

Repeat.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Repeat() { setLexeme("REPEAT"); };

virtual ~Repeat() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

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

};

};

RepeatUntilRule.h

#pragma once

#include "stdafx.h"

#include "Controller.h"

BackusRulePtr MakeRepeatUntil(std::shared\_ptr<Controller> controller);

RepeatUntilRule.cpp

#include "stdafx.h"

#include "RepeatUntilRule.h"

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

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

BackusRulePtr MakeRepeatUntil(std::shared\_ptr<Controller> controller)

{

controller->regItem<Repeat>();

controller->regItem<Until>();

auto context = controller->context();

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

auto operatorsRuleName = context->OperatorsRuleName();

auto repeatUntilRule = controller->addRule("RepeatUntilRule", {

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

BackusRuleItem({operatorsRuleName}, OnlyOne),

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

BackusRuleItem({ Symbols::LBraket}, OnlyOne),

BackusRuleItem({ context->EquationRuleName()}, OnlyOne),

BackusRuleItem({ Symbols::RBraket}, OnlyOne)

});

repeatUntilRule->setPostHandler([](BackusRuleList::iterator& ruleBegin,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

static size\_t index = 0;

index++;

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

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

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

size\_t count = 0;

for (auto itr = ruleBegin; itr != it; ++itr)

{

auto type = (\*itr)->type();

if (type == Repeat::Type())

{

count++;

}

else if (type == Until::Type() && count == 1)

{

count--;

(\*itr)->setCustomData(startLabel, "startLabel");

(\*itr)->setCustomData(endLabel, "endLabel");

break;

}

else if (type == Until::Type() && count > 0)

{

count--;

}

}

});

return repeatUntilRule;

}

Until.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Until() { setLexeme("UNTIL"); };

virtual ~Until() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

it++;

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

auto postIt = postForm.begin();

auto postEnd = postForm.end();

for (const auto& item : postForm)

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

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

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

out << "\tje " << customData("endLabel") << std::endl;

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

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

};

};

While:

While.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

While() { setLexeme("WHILE"); };

virtual ~While() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

if (customData("noGenerateCode") == "true")

{

return;

}

if (customData("ContinueWhile") == "true")

{

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

return;

}

if (customData("ExitWhile") == "true")

{

out << "\tjmp " << customData("endLabel") << std::endl;

return;

}

it++;

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

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

auto postIt = postForm.begin();

auto postEnd = postForm.end();

for (const auto& item : postForm)

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

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

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

out << "\tje " << customData("endLabel") << std::endl;

};

};

WhileRule.h

#pragma once

#include "stdafx.h"

#include "Controller.h"

BackusRulePtr MakeWhile(std::shared\_ptr<Controller> controller);

WhileRule.cpp

#include "stdafx.h"

#include "WhileRule.h"

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

#include "SimpleTokens.h"

SimpleToken(ExitWhile, "EXIT")

SimpleToken(ContinueWhile, "CONTINUE")

BackusRulePtr MakeWhile(std::shared\_ptr<Controller> controller)

{

controller->regItem<While>();

controller->regItem<ExitWhile>();

controller->regItem<ContinueWhile>();

auto context = controller->context();

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

auto operatorsRuleName = context->OperatorsRuleName();

auto operatorsName = context->OperatorsName();

auto operatorsWithSemicolonsName = context->OperatorsWithSemicolonsName();

auto whileExitStatement = controller->addRule("WhileExitStatement", {

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

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

});

auto whileContinueStatement = controller->addRule("WhileContinueStatement", {

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

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

});

auto whileCStatement = controller->addRule("WhileStatement", {

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

BackusRuleItem({ Symbols::LBraket}, OnlyOne),

BackusRuleItem({ context->EquationRuleName()}, OnlyOne),

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

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

BackusRuleItem({ operatorsName, operatorsWithSemicolonsName,whileContinueStatement->type(), whileExitStatement->type()}, Optional | OneOrMore),

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

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

});

whileCStatement->setPostHandler([](BackusRuleList::iterator& ruleBegin,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

static size\_t index = 0;

index++;

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

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

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

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

size\_t count = 0;

for (auto itr = ruleBegin; itr != it; ++itr)

{

auto type = (\*itr)->type();

if (type == lEnd && itr != it && (\*std::next(itr, 1))->type() == While::Type())

{

(\*std::next(itr, 1))->setCustomData("true", "noGenerateCode");

}

if (type == lStart)

{

count++;

}

else if (type == lEnd && count == 1)

{

(\*itr)->setCustomData(std::format("\tjmp {}\n{}:", startLabel, endLabel));

break;

}

else if (type == ExitWhile::Type() && count == 1 && itr != it && (\*std::next(itr, 1))->type() == While::Type())

{

itr++;

(\*itr)->setCustomData("true", "ExitWhile");

(\*itr)->setCustomData(endLabel, "endLabel");

}

else if(type == ContinueWhile::Type() && count == 1 && itr != it && (\*std::next(itr, 1))->type() == While::Type())

{

itr++;

(\*itr)->setCustomData("true", "ContinueWhile");

(\*itr)->setCustomData(startLabel, "startLabel");

}

else if (type == lEnd && count > 0)

{

count--;

}

}

});

return whileCStatement;

}

Tokens:

Comment.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Comment() { setLexeme(""); };

virtual ~Comment() = default;

std::shared\_ptr<IToken> tryCreateToken(std::string& lexeme) const override

{

auto token = clone();

token->setValue(lexeme);

lexeme.clear();

return token;

};

};

KWords:

Program.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Program() { setLexeme("NAME"); };

virtual ~Program() = default;

};

Vars.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Vars() { setLexeme("DATA"); };

virtual ~Vars() = default;

};

General:

EndOfFile.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

EndOfFile() { setLexeme(""); };

virtual ~EndOfFile() = default;

};

**Rules:**

AssigmentRules:

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Assignment() { setLexeme("<-"); };

virtual ~Assignment() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

auto ident = \*std::prev(it);

it++;

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

auto postIt = postForm.begin();

auto postEnd = postForm.end();

for (const auto& item : postForm)

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

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

};

};

#include "stdafx.h"

#include "AssignmentRule.h"

#include "Rules/AssignmentRule/Assignment.h"

BackusRulePtr MakeAssignmentRule(std::shared\_ptr<Controller> controller)

{

controller->regItem<Assignment>();

auto context = controller->context();

auto assingmentRule = controller->addRule(context->AssignmentRuleName(), {

BackusRuleItem({ context->IdentRuleName()}, OnlyOne),

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

BackusRuleItem({ context->EquationRuleName()}, OnlyOne)

});

return assingmentRule;

}

EquationRules:

Arithmetic:

Adittion.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Addition() { setLexeme("ADD"); };

virtual ~Addition() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Add\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Add\_", PrintAdd);

SetRegistered();

}

}

private:

static void PrintAdd(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Add============================================================================\n";

out << "Add\_ PROC\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tadd " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Add\_ ENDP\n";

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

}

};

Subtraction.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Subtraction() { setLexeme("SUB"); };

virtual ~Subtraction() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Sub\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Sub\_", PrintSub);

SetRegistered();

}

}

private:

static void PrintSub(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Sub============================================================================\n";

out << "Sub\_ PROC\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tsub " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Sub\_ ENDP\n";

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

}

};

Compare:

Equal.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Equal() { setLexeme("EQ"); };

virtual ~Equal() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Equal\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Equal\_", PrintEqual);

SetRegistered();

}

}

private:

static void PrintEqual(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Equal==========================================================================\n";

out << "Equal\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tjne equal\_false\n";

out << "\tmov " << args.regPrefix << "ax, 1\n";

out << "\tjmp equal\_fin\n";

out << "equal\_false:\n";

out << "\tmov " << args.regPrefix << "ax, 0\n";

out << "equal\_fin:\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Equal\_ ENDP\n";

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

}

};

Greate.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Greate() { setLexeme(">="); };

virtual ~Greate() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Greate\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Greate\_", PrintGreate);

SetRegistered();

}

}

private:

static void PrintGreate(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Greate=========================================================================\n";

out << "Greate\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tjle greate\_false\n";

out << "\tmov " << args.regPrefix << "ax, 1\n";

out << "\tjmp greate\_fin\n";

out << "greate\_false:\n";

out << "\tmov " << args.regPrefix << "ax, 0\n";

out << "greate\_fin:\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Greate\_ ENDP\n";

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

}

};

Less.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Less() { setLexeme("<="); };

virtual ~Less() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Less\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Less\_", PrintLess);

SetRegistered();

}

}

private:

static void PrintLess(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Less===========================================================================\n";

out << "Less\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tjge less\_false\n";

out << "\tmov " << args.regPrefix << "ax, 1\n";

out << "\tjmp less\_fin\n";

out << "less\_false:\n";

out << "\tmov " << args.regPrefix << "ax, 0\n";

out << "less\_fin:\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Less\_ ENDP\n";

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

}

};

NotEqual.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

#include "Rules/EquationRule/Equal.h"

#include "Rules/EquationRule/Not.h"

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

{

BASE\_ITEM

public:

NotEqual() { setLexeme("NE"); };

virtual ~NotEqual() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

Equal::RegPROC(details);

Not::RegPROC(details);

out << "\tcall Equal\_\n";

out << "\tcall Not\_\n";

};

};

Logic:

And.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

And() { setLexeme("AND"); };

virtual ~And() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall And\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("And\_", PrintAnd);

SetRegistered();

}

}

private:

static void PrintAnd(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure And============================================================================\n";

out << "And\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjnz and\_t1\n";

out << "\tjz and\_false\n";

out << "and\_t1:\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjnz and\_true\n";

out << "and\_false:\n";

out << "\tmov " << args.regPrefix << "ax, 0\n";

out << "\tjmp and\_fin\n";

out << "and\_true:\n";

out << "\tmov " << args.regPrefix << "ax, 1\n";

out << "and\_fin:\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "And\_ ENDP\n";

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

}

};

Not.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Not() { setLexeme("NOT"); };

virtual ~Not() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Not\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Not\_", PrintNot);

SetRegistered();

}

}

private:

static void PrintNot(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Not============================================================================\n";

out << "Not\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjnz not\_false\n";

out << "not\_t1:\n";

out << "\tmov " << args.regPrefix << "ax, 1\n";

out << "\tjmp not\_fin\n";

out << "not\_false:\n";

out << "\tmov " << args.regPrefix << "ax, 0\n";

out << "not\_fin:\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

out << "\tmov [esp + " << args.posArg1 << "], " << args.regPrefix << "ax\n";

out << "\tret\n";

out << "Not\_ ENDP\n";

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

}

};

Or.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Or() { setLexeme("OR"); };

virtual ~Or() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Or\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Or\_", PrintOr);

SetRegistered();

}

}

private:

static void PrintOr(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Or=============================================================================\n";

out << "Or\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjnz or\_true\n";

out << "\tjz or\_t1\n";

out << "or\_t1:\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjnz or\_true\n";

out << "or\_false:\n";

out << "\tmov " << args.regPrefix << "ax, 0\n";

out << "\tjmp or\_fin\n";

out << "or\_true:\n";

out << "\tmov " << args.regPrefix << "ax, 1\n";

out << "or\_fin:\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Or\_ ENDP\n";

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

}

};

Mult:

Division.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Division() { setLexeme("DIV"); };

virtual ~Division() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Div\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerStringData("DivErrMsg", "\\n" + Type() + ": Error: division by zero");

details.registerProc("Div\_", PrintDiv);

SetRegistered();

}

}

private:

static void PrintDiv(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Div============================================================================\n";

out << "Div\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjne end\_check\n";

out << "\tinvoke WriteConsoleA, hConsoleOutput, ADDR DivErrMsg, SIZEOF DivErrMsg - 1, 0, 0\n";

out << "\tjmp exit\_label\n";

out << "end\_check:\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjge gr\n";

out << "lo:\n";

out << "\tmov " << args.regPrefix << "dx, -1\n";

out << "\tjmp less\_fin\n";

out << "gr:\n";

out << "\tmov " << args.regPrefix << "dx, 0\n";

out << "less\_fin:\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tidiv " << args.numberTypeExtended << " ptr [esp + " << args.posArg1 << "]\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Div\_ ENDP\n";

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

}

};

Mod.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Mod() { setLexeme("MOD"); };

virtual ~Mod() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Mod\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerStringData("ModErrMsg", "\\n" + Type() + ": Error: division by zero");

details.registerProc("Mod\_", PrintMod);

SetRegistered();

}

}

private:

static void PrintMod(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Mod============================================================================\n";

out << "Mod\_ PROC\n";

out << "\tpushf\n";

out << "\tpop cx\n\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg1 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjne end\_check\n";

out << "\tinvoke WriteConsoleA, hConsoleOutput, ADDR ModErrMsg, SIZEOF ModErrMsg - 1, 0, 0\n";

out << "\tjmp exit\_label\n";

out << "end\_check:\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tcmp " << args.regPrefix << "ax, 0\n";

out << "\tjge gr\n";

out << "lo:\n";

out << "\tmov " << args.regPrefix << "dx, -1\n";

out << "\tjmp less\_fin\n";

out << "gr:\n";

out << "\tmov " << args.regPrefix << "dx, 0\n";

out << "less\_fin:\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\tidiv " << args.numberTypeExtended << " ptr [esp + " << args.posArg1 << "]\n";

out << "\tmov " << args.regPrefix << "ax, " << args.regPrefix << "dx\n";

out << "\tpush cx\n";

out << "\tpopf\n\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Mod\_ ENDP\n";

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

}

};

Multiplication.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Multiplication() { setLexeme("MUL"); };

virtual ~Multiplication() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

out << "\tcall Mul\_\n";

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerProc("Mul\_", PrintMul);

SetRegistered();

}

}

private:

static void PrintMul(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Mul============================================================================\n";

out << "Mul\_ PROC\n";

out << "\tmov " << args.regPrefix << "ax, [esp + " << args.posArg0 << "]\n";

out << "\timul " << args.numberTypeExtended << " ptr [esp + " << args.posArg1 << "]\n";

GeneratorUtils::PrintResultToStack(out, args);

out << "\tret\n";

out << "Mul\_ ENDP\n";

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

}

};

EquationRule.cpp

#include "stdafx.h"

#include "EquationRule.h"

#include "Rules/EquationRule/Number.h"

#include "Rules/EquationRule/Addition.h"

#include "Rules/EquationRule/Subtraction.h"

#include "Rules/EquationRule/Multiplication.h"

#include "Rules/EquationRule/Division.h"

#include "Rules/EquationRule/Mod.h"

#include "Rules/EquationRule/And.h"

#include "Rules/EquationRule/Or.h"

#include "Rules/EquationRule/Equal.h"

#include "Rules/EquationRule/Greate.h"

#include "Rules/EquationRule/Less.h"

#include "Rules/EquationRule/NotEqual.h"

#include "Rules/EquationRule/Not.h"

BackusRulePtr MakeEquationRule(std::shared\_ptr<Controller> controller)

{

using enum ItemType;

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

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

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

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

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

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

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

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

controller->regItem <Equal>(TokenAndRule | Operation, 2);

controller->regItem <NotEqual>(TokenAndRule | Operation, 2);

controller->regItem <Greate>(TokenAndRule | Operation, 3);

controller->regItem <Less>(TokenAndRule | Operation, 3);

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

auto context = controller->context();

auto equationRuleName = context->EquationRuleName();

auto sign = controller->addRule("Sign", { BackusRuleItem({ Symbols::Plus, Symbols::Minus }, Optional) });

auto signedNumber = controller->addRule("SignedNumber", {

BackusRuleItem({ sign->type()}, Optional),

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

});

signedNumber->setPostHandler([](BackusRuleList::iterator&,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

auto begRuleIt = std::prev(it, 2);

if ((\*begRuleIt)->type() == Symbols::Minus)

{

it = begRuleIt;

end = std::remove(it, end, \*it);

(\*it)->setValue('-' + (\*it)->value());

it++;

}

});

auto arithmetic = controller->addRule("Arithmetic", { BackusRuleItem({ Addition::Type(), Subtraction::Type() }, OnlyOne) });

auto mult = controller->addRule("Mult", { BackusRuleItem({ Multiplication::Type(), Division::Type(), Mod::Type() }, OnlyOne) });

auto logic = controller->addRule("Logic", { BackusRuleItem({ And::Type(), Or::Type() }, OnlyOne) });

auto compare = controller->addRule("Compare", { BackusRuleItem({ Equal::Type(), Greate::Type(), Less::Type(), NotEqual::Type() }, OnlyOne) });

auto operationAndEquation = controller->addRule("OperationAndEquation", {

BackusRuleItem({ mult->type(), arithmetic->type(), logic->type(), compare->type() }, OnlyOne),

BackusRuleItem({ equationRuleName }, OnlyOne)

});

auto notRule = controller->addRule("NotRule", {

BackusRuleItem({ Not::Type()}, OnlyOne),

BackusRuleItem({ equationRuleName}, Optional | OneOrMore)

});

auto equationWithBrakets = controller->addRule("EquationWithBrakets", {

BackusRuleItem({ Symbols::LBraket }, OnlyOne | PairStart),

BackusRuleItem({ equationRuleName }, OnlyOne),

BackusRuleItem({ Symbols::RBraket }, OnlyOne | PairEnd)

});

auto equation = controller->addRule(equationRuleName, {

BackusRuleItem({signedNumber->type(), context->IdentRuleName(), notRule->type(), equationWithBrakets->type()}, OnlyOne),

BackusRuleItem({operationAndEquation->type()}, Optional | OneOrMore)

});

return equation;

}

IdentRule:

Identifier.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

#include "Rules/AssignmentRule/Assignment.h"

#include "Tokens/Common/EndOfFile.h"

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

{

BASE\_ITEM

public:

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

virtual ~Identifier() = default;

std::shared\_ptr<IToken> tryCreateToken(std::string& lexeme) const override

{

if (lexeme.size() > (m\_mask.size() + m\_prefix.size()))

return nullptr;

bool res = true;

if (!lexeme.starts\_with(m\_prefix))

{

return nullptr;

}

std::string\_view ident{ lexeme.begin() + m\_prefix.size(), lexeme.end() };

for (size\_t i = 0; i < ident.size(); i++)

{

if ((isupper(ident[i]) != isupper(m\_mask[i])) && !isdigit(ident[i]))

{

res &= false;

break;

}

}

std::shared\_ptr<IToken> token = nullptr;

if (res)

{

token = clone();

token->setValue(lexeme);

lexeme.clear();

}

return token;

};

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

if (!GeneratorUtils::IsNextTokenIs(it, end, Assignment::Type()))

{

if ((\*std::prev(end))->type() == EndOfFile::Type())

details.registerNumberData(customData());

else

out << "\tpush " << customData() << std::endl;

}

};

private:

const std::string m\_prefix = "\_";

const std::string m\_mask = "xXXXXXX";

};

IdentRule.cpp

#include "stdafx.h"

#include "IdentRule.h"

#include "Rules/IdentRule/Identifier.h"

#include "Rules/IdentRule/Undefined.h"

SimpleToken(ProgramName, "");

BackusRulePtr MakeIdentRule(std::shared\_ptr<Controller> controller)

{

using enum ItemType;

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

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

auto context = controller->context();

auto identRule = controller->addRule(context->IdentRuleName(), {

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

});

identRule->setPostHandler([context](BackusRuleList::iterator&,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

static bool isFirstIdentChecked = !context->IsFirstProgName();

auto isVarBlockChecked = context->IsVarBlockChecked();

auto& identTable = context->IdentTable();

auto identIt = std::prev(it, 1);

if (isVarBlockChecked)

{

if (!identTable.contains((\*identIt)->value()))

{

auto undef = std::make\_shared<Undefined>();

undef->setValue((\*identIt)->value());

undef->setLine((\*identIt)->line());

undef->setCustomData((\*identIt)->customData());

\*identIt = undef;

}

}

else

{

if (isFirstIdentChecked)

{

identTable.insert((\*identIt)->value());

}

else

{

auto progName = std::make\_shared<ProgramName>();

progName->setValue((\*identIt)->value());

progName->setLine((\*identIt)->line());

progName->setCustomData((\*identIt)->customData());

\*identIt = progName;

isFirstIdentChecked = true;

}

}

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

});

return identRule;

}

Undefined.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

Undefined() { setLexeme(""); };

virtual ~Undefined() = default;

std::shared\_ptr<IToken> tryCreateToken(std::string& lexeme) const override

{

auto token = clone();

token->setValue(lexeme);

lexeme.clear();

return token;

};

};

ReadRule:

Read.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

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

virtual ~Read() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

RegPROC(details);

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

out << "\tcall Input\_\n";

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

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

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerRawData("InputBuf", "\tdb\t15 dup (?)");

details.registerRawData("CharsReadNum", "dd\t?");

details.registerProc("Input\_", PrintInput);

SetRegistered();

}

}

private:

static void PrintInput(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Input==========================================================================\n";

out << "Input\_ PROC\n";

out << "\tinvoke ReadConsoleA, hConsoleInput, ADDR InputBuf, 13, ADDR CharsReadNum, 0\n";

out << "\tinvoke crt\_atoi, ADDR InputBuf\n";

out << "\tret\n";

out << "Input\_ ENDP\n";

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

}

};

ReadRule.cpp

#include "stdafx.h"

#include "ReadRule.h"

#include "Rules/ReadRule/Read.h"

BackusRulePtr MakeReadRule(std::shared\_ptr<Controller> controller)

{

controller->regItem<Read>();

auto context = controller->context();

auto read = controller->addRule("ReadRule", {

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

BackusRuleItem({ Symbols::LBraket}, OnlyOne),

BackusRuleItem({ context->IdentRuleName()}, OnlyOne),

BackusRuleItem({ Symbols::RBraket}, OnlyOne)

});

return read;

}

StringRule:

String.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

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

virtual ~String() = default;

std::string stringName() const { return m\_stringName; };

std::shared\_ptr<IToken> tryCreateToken(std::string& lexeme) const override

{

auto token = clone();

token->setValue(lexeme);

lexeme.clear();

return token;

};

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

m\_stringName = std::format("String\_{}", index++);

details.registerStringData(m\_stringName, value());

};

private:

mutable std::string m\_stringName;

static size\_t index;

};

StringRule.cpp

#include "stdafx.h"

#include "StringRule.h"

#include "Rules/StringRule/String.h"

SimpleToken(Quotes, "\"");

BackusRulePtr MakeStringRule(std::shared\_ptr<Controller> controller)

{

using enum ItemType;

controller->regUnchangedTextToken(std::make\_shared<String>(), std::make\_shared<Quotes>(), std::make\_shared<Quotes>());

controller->regItem<Quotes>(Rule);

controller->regItem<String>(Rule);

auto stringRule = controller->addRule("StringRule", {

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

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

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

});

stringRule->setPostHandler([](BackusRuleList::iterator&,

BackusRuleList::iterator& it,

BackusRuleList::iterator& end)

{

it = std::prev(it, 3);

end = std::remove(it, end, \*it);

it++;

end = std::remove(it, end, \*it);

});

return stringRule;

}

VarsBlockRule:

VarsBlockRule.cpp

#include "stdafx.h"

#include "VarsBlokRule.h"

#include "Rules/VarsBlokRule/VarType.h"

BackusRulePtr MakeVarsBlokRule(std::shared\_ptr<Controller> controller)

{

controller->regItem<VarType>();

auto context = controller->context();

auto commaAndIdentifier = controller->addRule("CommaAndIdentifier", {

BackusRuleItem({ Symbols::Comma}, OnlyOne),

BackusRuleItem({ context->IdentRuleName()}, OnlyOne)

});

auto varsBlok = controller->addRule("VarsBlok", {

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

BackusRuleItem({ context->IdentRuleName()}, OnlyOne),

BackusRuleItem({commaAndIdentifier->type()}, Optional | OneOrMore),

BackusRuleItem({ Symbols::Semicolon}, OnlyOne)

});

varsBlok->setPostHandler([context](BackusRuleList::iterator&, BackusRuleList::iterator&, BackusRuleList::iterator&)

{

auto isVarBlockChecked = context->IsVarBlockChecked();

context->SetVarBlockChecked();

});

return varsBlok;

}

VarType.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

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

{

BASE\_ITEM

public:

VarType() { setLexeme("INTEGER\_2"); };

virtual ~VarType() = default;

};

WriteRule:

Write.h

#pragma once

#include "stdafx.h"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRuleBase.h"

#include "Core/Generator/GeneratorItemBase.h"

#include "Rules/StringRule/String.h"

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

{

BASE\_ITEM

public:

Write() { setLexeme("PRINT"); };

virtual ~Write() = default;

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

std::list<std::shared\_ptr<IGeneratorItem>>::iterator& it,

const std::list<std::shared\_ptr<IGeneratorItem>>::iterator& end) const final

{

if (auto string = std::dynamic\_pointer\_cast<String>(\*std::next(it, 2)))

{

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

string->genCode(out, details, it, end);

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

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

}

else

{

RegPROC(details);

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

auto postIt = postForm.begin();

auto postEnd = postForm.end();

for (const auto& item : postForm)

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

out << "\tcall Output\_\n";

}

};

static void RegPROC(GeneratorDetails& details)

{

if (!IsRegistered())

{

details.registerRawData("OutMessage", "\tdb\t\"" + details.args().numberStrType + "\", 0");

details.registerRawData("ResMessage", "\tdb\t20 dup (?)");

details.registerProc("Output\_", PrintOutput);

SetRegistered();

}

}

private:

static void PrintOutput(std::ostream& out, const GeneratorDetails::GeneratorArgs& args)

{

out << ";===Procedure Output=========================================================================\n";

out << "Output\_ PROC value: " << args.numberTypeExtended.c\_str() << std::endl;

out << "\tinvoke wsprintf, ADDR ResMessage, ADDR OutMessage, value\n";

out << "\tinvoke WriteConsoleA, hConsoleOutput, ADDR ResMessage, eax, 0, 0\n";

out << "\tret " << args.argSize << std::endl;

out << "Output\_ ENDP\n";

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

}

};

WriteRule.cpp

#include "stdafx.h"

#include "WriteRule.h"

#include "Rules/StringRule/StringRule.h"

#include "Rules/WriteRule/Write.h"

BackusRulePtr MakeWriteRule(std::shared\_ptr<Controller> controller)

{

controller->regItem<Write>();

auto context = controller->context();

auto stringRule = MakeStringRule(controller);

auto write = controller->addRule("WriteRule", {

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

BackusRuleItem({ Symbols::LBraket}, OnlyOne | PairStart),

BackusRuleItem({ stringRule->type(), context->EquationRuleName() }, OnlyOne),

BackusRuleItem({ Symbols::RBraket}, OnlyOne | PairEnd)

});

return write;

}

Controller.h

#pragma once

#include "stdafx.h"

#include "Utils/singleton.hpp"

#include "Core/Tokens/TokenBase.hpp"

#include "Core/Backus/BackusRule.h"

#include "Core/Generator/GeneratorItemBase.h"

#include "Core/Generator/GeneratorDetails.h"

#include "Symbols.h"

using BackusRulePtr = std::shared\_ptr<IBackusRule>;

using BackusRuleList = std::list<BackusRulePtr>;

using BackusRuleListIt = BackusRuleList::iterator;

using RuleMaker = std::function<BackusRulePtr(std::shared\_ptr<Controller>)>;

class Context

{

public:

std::string IdentRuleName() const { return "IdentRule"; }

std::string EquationRuleName() const { return "Equation"; }

std::string OperatorsRuleName() const { return "OperatorsRule"; }

std::string OperatorsName() const { return "Operators"; }

std::string OperatorsWithSemicolonsName() const { return "OperatorsWithSemicolon"; }

std::string AssignmentRuleName() const { return "AssignmentRule"; }

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

bool IsVarBlockChecked() const { return m\_isVarBlockChecked; }

void SetVarBlockChecked() { m\_isVarBlockChecked = true; }

bool IsFirstProgName() const { return true; }

std::set<std::string>& IdentTable() { return m\_identTable; }

const GeneratorDetails& Details() const { return m\_details; }

private:

std::set<std::string> m\_identTable{};

bool m\_isVarBlockChecked = false;

const GeneratorDetails m\_details{ {

.numberType = "dw", .numberTypeExtended = "word",

.argSize = 2,

.numberStrType = "%hd"

} };

};

enum class ItemType : uint32\_t

{

None = 0,

Token = 1 << 0,

Rule = 1 << 1,

TokenAndRule = Token | Rule,

Operand = 1 << 2,

Operation = 1 << 3,

EquationEnd = 1 << 4,

LBracket = 1 << 5,

RBracket = 1 << 6

};

DEFINE\_ENUM\_FLAG\_OPERATORS(ItemType)

class Controller : public singleton<Controller>

{

public:

static constexpr int NoPriority = std::numeric\_limits<int>::min();

public:

void init();

template<typename T>

void regItem(ItemType type = ItemType::TokenAndRule, int priority = NoPriority) const

{

auto item = std::make\_shared<T>();

regItem(item, item, type, priority);

}

void regUnchangedTextToken(std::shared\_ptr<IToken> target, std::shared\_ptr<IToken> lBorder, std::shared\_ptr<IToken> rBorder) const;

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

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

BackusRulePtr topRule();

std::shared\_ptr<Context> context() { return m\_context; }

protected:

Controller() { m\_context = std::make\_shared<Context>(); }

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

BackusRulePtr MakeTopRule(std::shared\_ptr<Controller> controller) const;

private:

BackusRulePtr m\_topRule;

std::set<std::string> m\_operatorRuleNames;

std::set<std::string> m\_operatorRuleWithSemicolonNames;

std::shared\_ptr<Context> m\_context;

};

Controller.cpp

#include "stdafx.h"

#include "Controller.h"

#include "Core/Parser/TokenParser.h"

#include "Core/Backus/BackusRuleStorage.h"

#include "Core/Generator/Generator.h"

#include "Core/Generator/GeneratorUtils.h"

#include "SimpleTokens.h"

#include "Tokens/Common/Program.h"

#include "Tokens/Common/Vars.h"

#include "Rules/IdentRule/IdentRule.h"

#include "Rules/VarsBlokRule/VarsBlokRule.h"

#include "Rules/EquationRule/EquationRule.h"

#include "Rules/ReadRule/ReadRule.h"

#include "Rules/WriteRule/WriteRule.h"

#include "Rules/AssignmentRule/AssignmentRule.h"

void Controller::regItem(std::shared\_ptr<IToken> token, std::shared\_ptr<IBackusRule> rule, ItemType type, int priority) const

{

using enum ItemType;

if ((type & Token) == Token)

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

if ((type & Rule) == Rule)

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

auto tokenType = token->type();

if ((type & Operand) == Operand)

GeneratorUtils::Instance()->RegisterOperand(tokenType);

if ((type & Operation) == Operation)

{

if (priority == TokenParser::NoPriority)

throw std::runtime\_error("Controller::RegItem: Operation " + token->type() + " priority is not set");

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

}

if ((type & EquationEnd) == EquationEnd)

GeneratorUtils::Instance()->RegisterEquationEnd(tokenType);

if ((type & LBracket) == LBracket)

GeneratorUtils::Instance()->RegisterLBraket(tokenType);

if ((type & RBracket) == RBracket)

GeneratorUtils::Instance()->RegisterRBraket(tokenType);

}

void Controller::init()

{

m\_topRule = MakeTopRule(Instance());

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

}

void Controller::regUnchangedTextToken(std::shared\_ptr<IToken> target, std::shared\_ptr<IToken> lBorder, std::shared\_ptr<IToken> rBorder) const

{

TokenParser::Instance()->regUnchangedTextToken(target, lBorder, rBorder);

}

void Controller::regOperatorRule(const RuleMaker& rule, bool isNeedSemicolon)

{

auto ruleName = rule(Instance())->type();

if (ruleName.empty())

throw std::runtime\_error("Controller::RegOperatorRule: Rule name is empty");

if (m\_operatorRuleNames.contains(ruleName) || m\_operatorRuleWithSemicolonNames.contains(ruleName))

throw std::runtime\_error(std::format("Controller::RegOperatorRule: Rule with name {} already registered", ruleName));

if (isNeedSemicolon)

m\_operatorRuleWithSemicolonNames.insert(ruleName);

else

m\_operatorRuleNames.insert(ruleName);

}

std::shared\_ptr<IBackusRule> Controller::addRule(const std::string& name, const std::list<BackusRuleItem>& items) const

{

auto rule = BackusRule::MakeRule(name, items);

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

return rule;

}

BackusRulePtr Controller::topRule()

{

if (!m\_topRule)

throw(std::runtime\_error("Controller is not inited"));

return m\_topRule;

}

BackusRulePtr Controller::MakeTopRule(std::shared\_ptr<Controller> controller) const

{

using enum ItemType;

controller->regItem<Program>();

controller->regItem<Vars>();

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

controller->regItem<End>();

controller->regItem<Comma>();

controller->regItem<Colon>();

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

controller->regItem<LBraket>(TokenAndRule | LBracket);

controller->regItem<RBraket>(TokenAndRule | RBracket);

controller->regItem<Plus>();

controller->regItem<Minus>();

auto identRule = MakeIdentRule(controller);

auto varsBlok = MakeVarsBlokRule(controller);

auto equation = MakeEquationRule(controller);

auto read = MakeReadRule(controller);

auto write = MakeWriteRule(controller);

auto assingmentRule = MakeAssignmentRule(controller);

auto operatorWithSemicolonTypes = std::vector<std::variant<std::string, Symbols>>{ read->type(), write->type(), assingmentRule->type() };

operatorWithSemicolonTypes.insert(operatorWithSemicolonTypes.end(), m\_operatorRuleWithSemicolonNames.begin(), m\_operatorRuleWithSemicolonNames.end());

auto operatorsWithSemicolon = controller->addRule("OperatorsWithSemicolon", {

BackusRuleItem({ operatorWithSemicolonTypes }, OnlyOne),

BackusRuleItem({ Symbols::Semicolon }, OnlyOne)

});

auto operatorTypes = std::vector<std::variant<std::string, Symbols>>{ m\_operatorRuleNames.begin(), m\_operatorRuleNames.end() };

auto operators = controller->addRule("Operators", {

BackusRuleItem({ operatorTypes }, OnlyOne)

});

auto operatorsRule = controller->addRule("OperatorsRule", {

BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional | OneOrMore),

});

auto codeBlok = controller->addRule("CodeBlok", {

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

BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional | OneOrMore),

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

});

auto topRule = controller->addRule("TopRule", {

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

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

BackusRuleItem({ Symbols::Semicolon}, OnlyOne),

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

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

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

BackusRuleItem({ operators->type(), operatorsWithSemicolon->type()}, Optional | OneOrMore),

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

});

return topRule;

}

**Додаток В (Код на мові Асемблер)**

**Prog1.asm**

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User Data================================================================================

\_aAAAAAA\_ dw 0

\_bBBBBBB\_ dw 0

\_xXXXXXX\_ dw 0

\_yYYYYYY\_ dw 0

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

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

String\_0 db "Input A: ", 0

String\_1 db "Input B: ", 0

String\_2 db "A + B: ", 0

String\_3 db 13, 10, "A - B: ", 0

String\_4 db 13, 10, "A \* B: ", 0

String\_5 db 13, 10, "A / B: ", 0

String\_6 db 13, 10, "A % B: ", 0

String\_7 db 13, 10, "X = (A - B) \* 10 + (A + B) / 10", 13, 10, 0

String\_8 db 13, 10, "Y = X + (X % 10)", 13, 10, 0

;===Addition Data============================================================================

hConsoleInput dd ?

hConsoleOutput dd ?

endBuff db 5 dup (?)

msg1310 db 13, 10, 0

CharsReadNum dd ?

InputBuf db 15 dup (?)

OutMessage db "%hd", 0

ResMessage db 20 dup (?)

.CODE

start:

invoke AllocConsole

invoke GetStdHandle, STD\_INPUT\_HANDLE

mov hConsoleInput, eax

invoke GetStdHandle, STD\_OUTPUT\_HANDLE

mov hConsoleOutput, eax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_0, SIZEOF String\_0 - 1, 0, 0

call Input\_

mov \_aAAAAAA\_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_1, SIZEOF String\_1 - 1, 0, 0

call Input\_

mov \_bBBBBBB\_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_2, SIZEOF String\_2 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

call Add\_

call Output\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_3, SIZEOF String\_3 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

call Sub\_

call Output\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_4, SIZEOF String\_4 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

call Mul\_

call Output\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_5, SIZEOF String\_5 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

call Div\_

call Output\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_6, SIZEOF String\_6 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

call Mod\_

call Output\_

push \_aAAAAAA\_

push \_bBBBBBB\_

call Sub\_

push word ptr 10

call Mul\_

push \_aAAAAAA\_

push \_bBBBBBB\_

call Add\_

push word ptr 10

call Div\_

call Add\_

pop \_xXXXXXX\_

push \_xXXXXXX\_

push \_xXXXXXX\_

push word ptr 10

call Mod\_

call Add\_

pop \_yYYYYYY\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_7, SIZEOF String\_7 - 1, 0, 0

push \_xXXXXXX\_

call Output\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_8, SIZEOF String\_8 - 1, 0, 0

push \_yYYYYYY\_

call Output\_

exit\_label:

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

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

invoke ExitProcess, 0

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

Add\_ PROC

mov ax, [esp + 6]

add ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Add\_ ENDP

;============================================================================================

;===Procedure Div============================================================================

Div\_ PROC

pushf

pop cx

mov ax, [esp + 4]

cmp ax, 0

jne end\_check

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

jmp exit\_label

end\_check:

mov ax, [esp + 6]

cmp ax, 0

jge gr

lo:

mov dx, -1

jmp less\_fin

gr:

mov dx, 0

less\_fin:

mov ax, [esp + 6]

idiv word ptr [esp + 4]

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Div\_ ENDP

;============================================================================================

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

Input\_ PROC

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

invoke crt\_atoi, ADDR InputBuf

ret

Input\_ ENDP

;============================================================================================

;===Procedure Mod============================================================================

Mod\_ PROC

pushf

pop cx

mov ax, [esp + 4]

cmp ax, 0

jne end\_check

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

jmp exit\_label

end\_check:

mov ax, [esp + 6]

cmp ax, 0

jge gr

lo:

mov dx, -1

jmp less\_fin

gr:

mov dx, 0

less\_fin:

mov ax, [esp + 6]

idiv word ptr [esp + 4]

mov ax, dx

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Mod\_ ENDP

;============================================================================================

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

Mul\_ PROC

mov ax, [esp + 6]

imul word ptr [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Mul\_ ENDP

;============================================================================================

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

Output\_ PROC value: word

invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value

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

ret 2

Output\_ ENDP

;============================================================================================

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

Sub\_ PROC

mov ax, [esp + 6]

sub ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Sub\_ ENDP

;============================================================================================

end start

**Prog2.asm**

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User Data================================================================================

\_aAAAAAA\_ dw 0

\_bBBBBBB\_ dw 0

\_cCCCCCC\_ dw 0

String\_0 db "Input A: ", 0

String\_1 db "Input B: ", 0

String\_2 db "Input C: ", 0

String\_3 db 13, 10, 0

String\_4 db 13, 10, 0

String\_5 db 13, 10, 0

;===Addition Data============================================================================

hConsoleInput dd ?

hConsoleOutput dd ?

endBuff db 5 dup (?)

msg1310 db 13, 10, 0

CharsReadNum dd ?

InputBuf db 15 dup (?)

OutMessage db "%hd", 0

ResMessage db 20 dup (?)

.CODE

start:

invoke AllocConsole

invoke GetStdHandle, STD\_INPUT\_HANDLE

mov hConsoleInput, eax

invoke GetStdHandle, STD\_OUTPUT\_HANDLE

mov hConsoleOutput, eax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_0, SIZEOF String\_0 - 1, 0, 0

call Input\_

mov \_aAAAAAA\_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_1, SIZEOF String\_1 - 1, 0, 0

call Input\_

mov \_bBBBBBB\_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_2, SIZEOF String\_2 - 1, 0, 0

call Input\_

mov \_cCCCCCC\_, ax

push \_aAAAAAA\_

push \_bBBBBBB\_

call Greate\_

pop ax

cmp ax, 0

je endIf2

push \_aAAAAAA\_

push \_cCCCCCC\_

call Greate\_

pop ax

cmp ax, 0

je elseLabel1

jmp \_aBIGGER\_

jmp endIf1

elseLabel1:

push \_cCCCCCC\_

call Output\_

jmp \_oUTOFI\_

\_aBIGGER\_:

push \_aAAAAAA\_

call Output\_

jmp \_oUTOFI\_

endIf1:

endIf2:

push \_bBBBBBB\_

push \_cCCCCCC\_

call Less\_

pop ax

cmp ax, 0

je elseLabel3

push \_cCCCCCC\_

call Output\_

jmp endIf3

elseLabel3:

push \_bBBBBBB\_

call Output\_

endIf3:

\_oUTOFI\_:

invoke WriteConsoleA, hConsoleOutput, ADDR String\_3, SIZEOF String\_3 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

call Equal\_

push \_aAAAAAA\_

push \_cCCCCCC\_

call Equal\_

call And\_

push \_bBBBBBB\_

push \_cCCCCCC\_

call Equal\_

call And\_

pop ax

cmp ax, 0

je elseLabel4

push word ptr 1

call Output\_

jmp endIf4

elseLabel4:

push word ptr 0

call Output\_

endIf4:

invoke WriteConsoleA, hConsoleOutput, ADDR String\_4, SIZEOF String\_4 - 1, 0, 0

push \_aAAAAAA\_

push word ptr 0

call Less\_

push \_bBBBBBB\_

push word ptr 0

call Less\_

call Or\_

push \_cCCCCCC\_

push word ptr 0

call Less\_

call Or\_

pop ax

cmp ax, 0

je elseLabel5

push word ptr -1

call Output\_

jmp endIf5

elseLabel5:

push word ptr 0

call Output\_

endIf5:

invoke WriteConsoleA, hConsoleOutput, ADDR String\_5, SIZEOF String\_5 - 1, 0, 0

push \_aAAAAAA\_

push \_bBBBBBB\_

push \_cCCCCCC\_

call Add\_

call Less\_

call Not\_

pop ax

cmp ax, 0

je elseLabel6

push word ptr 10

call Output\_

jmp endIf6

elseLabel6:

push word ptr 0

call Output\_

endIf6:

exit\_label:

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

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

invoke ExitProcess, 0

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

Add\_ PROC

mov ax, [esp + 6]

add ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Add\_ ENDP

;============================================================================================

;===Procedure And============================================================================

And\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, 0

jnz and\_t1

jz and\_false

and\_t1:

mov ax, [esp + 4]

cmp ax, 0

jnz and\_true

and\_false:

mov ax, 0

jmp and\_fin

and\_true:

mov ax, 1

and\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

And\_ ENDP

;============================================================================================

;===Procedure Equal==========================================================================

Equal\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jne equal\_false

mov ax, 1

jmp equal\_fin

equal\_false:

mov ax, 0

equal\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Equal\_ ENDP

;============================================================================================

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

Greate\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jle greate\_false

mov ax, 1

jmp greate\_fin

greate\_false:

mov ax, 0

greate\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Greate\_ ENDP

;============================================================================================

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

Input\_ PROC

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

invoke crt\_atoi, ADDR InputBuf

ret

Input\_ ENDP

;============================================================================================

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

Less\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jge less\_false

mov ax, 1

jmp less\_fin

less\_false:

mov ax, 0

less\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Less\_ ENDP

;============================================================================================

;===Procedure Not============================================================================

Not\_ PROC

pushf

pop cx

mov ax, [esp + 4]

cmp ax, 0

jnz not\_false

not\_t1:

mov ax, 1

jmp not\_fin

not\_false:

mov ax, 0

not\_fin:

push cx

popf

mov [esp + 4], ax

ret

Not\_ ENDP

;============================================================================================

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

Or\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, 0

jnz or\_true

jz or\_t1

or\_t1:

mov ax, [esp + 4]

cmp ax, 0

jnz or\_true

or\_false:

mov ax, 0

jmp or\_fin

or\_true:

mov ax, 1

or\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Or\_ ENDP

;============================================================================================

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

Output\_ PROC value: word

invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value

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

ret 2

Output\_ ENDP

;============================================================================================

end start

**Prog3.asm**

.386

.model flat, stdcall

option casemap :none

include masm32\include\windows.inc

include masm32\include\kernel32.inc

include masm32\include\masm32.inc

include masm32\include\user32.inc

include masm32\include\msvcrt.inc

includelib masm32\lib\kernel32.lib

includelib masm32\lib\masm32.lib

includelib masm32\lib\user32.lib

includelib masm32\lib\msvcrt.lib

.DATA

;===User Data================================================================================

\_aAAAAA2\_ dw 0

\_aAAAAA\_ dw 0

\_bBBBBBB\_ dw 0

\_cCCCCC1\_ dw 0

\_cCCCCC2\_ dw 0

\_xXXXXXX\_ dw 0

String\_0 db "Input A: ", 0

String\_1 db "Input B: ", 0

String\_2 db "FOR TO DO", 0

String\_3 db 13, 10, 0

String\_4 db 13, 10, "FOR DOWNTO DO", 0

String\_5 db 13, 10, 0

String\_6 db 13, 10, "WHILE A \* B: ", 0

String\_7 db 13, 10, "REPEAT UNTIL A \* B: ", 0

;===Addition Data============================================================================

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 \_bBBBBBB\_, ax

invoke WriteConsoleA, hConsoleOutput, ADDR String\_2, SIZEOF String\_2 - 1, 0, 0

push \_aAAAAA\_

pop \_aAAAAA2\_

forPasStart1:

push \_bBBBBBB\_

push \_aAAAAA2\_

call Less\_

call Not\_

pop ax

cmp ax, 0

je forPasEnd1

invoke WriteConsoleA, hConsoleOutput, ADDR String\_3, SIZEOF String\_3 - 1, 0, 0

push \_aAAAAA2\_

push \_aAAAAA2\_

call Mul\_

call Output\_

push \_aAAAAA2\_

push word ptr 1

call Add\_

pop \_aAAAAA2\_

jmp forPasStart1

forPasEnd1:

invoke WriteConsoleA, hConsoleOutput, ADDR String\_4, SIZEOF String\_4 - 1, 0, 0

push \_bBBBBBB\_

pop \_aAAAAA2\_

forPasStart2:

push \_aAAAAA\_

push \_aAAAAA2\_

call Greate\_

call Not\_

pop ax

cmp ax, 0

je forPasEnd2

invoke WriteConsoleA, hConsoleOutput, ADDR String\_5, SIZEOF String\_5 - 1, 0, 0

push \_aAAAAA2\_

push \_aAAAAA2\_

call Mul\_

call Output\_

push \_aAAAAA2\_

push word ptr 1

call Sub\_

pop \_aAAAAA2\_

jmp forPasStart2

forPasEnd2:

invoke WriteConsoleA, hConsoleOutput, ADDR String\_6, SIZEOF String\_6 - 1, 0, 0

push word ptr 0

pop \_xXXXXXX\_

push word ptr 0

pop \_cCCCCC1\_

whileStart2:

push \_cCCCCC1\_

push \_aAAAAA\_

call Less\_

pop ax

cmp ax, 0

je whileEnd2

push word ptr 0

pop \_cCCCCC2\_

whileStart1:

push \_cCCCCC2\_

push \_bBBBBBB\_

call Less\_

pop ax

cmp ax, 0

je whileEnd1

push \_xXXXXXX\_

push word ptr 1

call Add\_

pop \_xXXXXXX\_

push \_cCCCCC2\_

push word ptr 1

call Add\_

pop \_cCCCCC2\_

jmp whileStart1

whileEnd1:

push \_cCCCCC1\_

push word ptr 1

call Add\_

pop \_cCCCCC1\_

jmp whileStart2

whileEnd2:

push \_xXXXXXX\_

call Output\_

invoke WriteConsoleA, hConsoleOutput, ADDR String\_7, SIZEOF String\_7 - 1, 0, 0

push word ptr 0

pop \_xXXXXXX\_

push word ptr 1

pop \_cCCCCC1\_

repeatStart2:

push word ptr 1

pop \_cCCCCC2\_

repeatStart1:

push \_xXXXXXX\_

push word ptr 1

call Add\_

pop \_xXXXXXX\_

push \_cCCCCC2\_

push word ptr 1

call Add\_

pop \_cCCCCC2\_

push \_cCCCCC2\_

push \_bBBBBBB\_

call Greate\_

call Not\_

pop ax

cmp ax, 0

je repeatEnd1

jmp repeatStart1

repeatEnd1:

push \_cCCCCC1\_

push word ptr 1

call Add\_

pop \_cCCCCC1\_

push \_cCCCCC1\_

push \_aAAAAA\_

call Greate\_

call Not\_

pop ax

cmp ax, 0

je repeatEnd2

jmp repeatStart2

repeatEnd2:

push \_xXXXXXX\_

call Output\_

exit\_label:

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

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

invoke ExitProcess, 0

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

Add\_ PROC

mov ax, [esp + 6]

add ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Add\_ ENDP

;============================================================================================

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

Greate\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jle greate\_false

mov ax, 1

jmp greate\_fin

greate\_false:

mov ax, 0

greate\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Greate\_ ENDP

;============================================================================================

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

Input\_ PROC

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

invoke crt\_atoi, ADDR InputBuf

ret

Input\_ ENDP

;============================================================================================

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

Less\_ PROC

pushf

pop cx

mov ax, [esp + 6]

cmp ax, [esp + 4]

jge less\_false

mov ax, 1

jmp less\_fin

less\_false:

mov ax, 0

less\_fin:

push cx

popf

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Less\_ ENDP

;============================================================================================

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

Mul\_ PROC

mov ax, [esp + 6]

imul word ptr [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Mul\_ ENDP

;============================================================================================

;===Procedure Not============================================================================

Not\_ PROC

pushf

pop cx

mov ax, [esp + 4]

cmp ax, 0

jnz not\_false

not\_t1:

mov ax, 1

jmp not\_fin

not\_false:

mov ax, 0

not\_fin:

push cx

popf

mov [esp + 4], ax

ret

Not\_ ENDP

;============================================================================================

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

Output\_ PROC value: word

invoke wsprintf, ADDR ResMessage, ADDR OutMessage, value

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

ret 2

Output\_ ENDP

;============================================================================================

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

Sub\_ PROC

mov ax, [esp + 6]

sub ax, [esp + 4]

mov [esp + 6], ax

pop ecx

pop ax

push ecx

ret

Sub\_ ENDP

;============================================================================================

end start